ALIBABA CLOUD

阿里云

交互式分析Hologres
交互式分析公共云合集

文档版本：20220713

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.动态与公告
1.1. 功能发布记录

本文为您介绍Hologres产品功能的发布信息。

Hologres 2021年10月新增功能

2021年10月正式发布Hologres V1.1版本，新增功能具体如下：

- **运维能力改善**
  - 新增资源组隔离能力（Beta），通过设计多个资源组，实现实例内部不同用户的计算资源线程级负载隔离，可以更好地支撑多用户、多场景的使用方式。详情请参见单实例计算资源隔离（Beta）。
  - 支持Hologres实例在线热升级，升级期间，读（查询）数据不受影响，升级版本时可以提工单使用热升级。
- **引擎能力增强**
  - 支持将表设计为行列共存结构，一份数据同时支持点查、OLAP多种查询场景，详情请参见CREATE TABLE。
  - 支持JDBC实时消费Hologres Binlog（Beta），详情请参见通过JDBC消费Hologres Binlog（Beta）。
  - 支持Hologres Binlog按需启用，配置动态修改，详情请参见订阅Hologres Binlog。
  - 支持重命名列名称，详情请参见ALTER TABLE。
  - 新增JSONB索引（Beta），加速JSON类型数据的查询检索。详情请参见JSON和JSONB类型。
  - 优化内存中元数据管理机制，增加元数据缓存与压缩，更有效率的管理内存。
- **外表能力优化**
  - 支持Hologres数据批量写入OSS外表，进一步降低存储成本。详情请参见Hologres数据导出至OSS。
  - 支持Hologres跨库查询，支持Hologres多实例联邦查询。详情请参见跨库查询。
- **安全增强**
  - 支持Hologres内部表数据存储加密（Beta），提升数据访问安全能力，详情请参见数据存储加密。
  - 支持读MaxCompute加密数据（Beta），丰富Hologres兼容MaxCompute生态。详情请参见查询MaxCompute加密数据（BYOK模式）。
- **行为变更说明**
  - Auto Analyze能力，在Hologres V1.1版本中转为默认开启。
  - Hologres新引擎直读MaxCompute，在V1.1版本中转为默认开启。
  - Resharding函数完成Beta，相关函数名更新。

  更多行为变更，请参见默认行为变更说明。

Hologres 2021年5月新增功能

2021年5月正式发布Hologres V0.10版本，新增功能具体如下：

- **引擎增强**
  - 支持自动采集表的统计信息：数据写入更新时自动采样表的统计信息，以便生成更优的Query Plan，不再需要手工执行Analyze Table。详情请参见ANALYZE和AUTO ANALYZE。

- 支持点查（Key/Value）场景的毫秒级高可靠（Beta）：支持Shard级别多副本配置，支持毫秒级主副本切换和查询重试，显著提高服务场景下的高可靠能力。详情请参见单实例Shard多副本高吞吐（Beta）。
- 新增RoaringBitmap扩展，原生支持Bitmap数据类型及相关函数。详情请参见Roaring Bitmap函数。
- 新增单表维度聚合视图：针对可累加数据定义单表聚合视图，实现对汇聚数据更快的查询性能。详情请参见聚合视图（Beta）。
- 新增bit_construct和bit_match函数：针对圈人、归因等场景进行优化，支持更高效率的基于userid的聚合条件过滤。详情请参见明细圈人函数。
- 新增range_retention_count和range_retention_sum函数：针对留存场景优化多天范围查询。详情请参见漏斗分析函数。
- 新增Resharding工具：内置Resharding函数，修改Shard数无需重新建表，简化调优过程。详情请参见Table Group与Shard Count操作指南。
- 优化列存默认采用AliORC压缩格式，存储压缩比提高30%~50%。详情请参见更改列存表的数据存储格式。

- **外表查询功能**
  - MaxCompute外部表查询性能提升（Beta）：全新外部表加速引擎，相比之前版本，查询性能约有30% ~ 100%的提升。详情请参见优化MaxCompute外部表的查询性能。
  - 新增exec_external_sql函数：通过exec_external_sql函数维护MaxCompute表，实现对DDL的管理。详情请参见执行MaxCompute SQL语句。
  - 新增OSS外表读取：支持读取OSS中parquet、orc、text等格式的文件。详情请参见通过外部表直接访问OSS。
  - 新增集成DLF（Beta）：通过DLF读取OSS数据。详情请参见通过DLF读取OSS数据。
- **性能优化**
  - 点查性能提升：行存总吞吐提升100%，列存总吞吐提升30%。
  - 更新操作优化：Update/Delete优化，性能提升30%。
  - Query Plan缓存：优化Query Plan Cache，降低优化器耗时。
- **企业级运维与安全优化**
  - 慢查询透出，内置查询状态历史，可以查询一个月内所有查询的状态，快速定位慢查询，失败查询。详情请参见慢Query日志查看与分析。

Hologres 2021年1月新增功能

2021年1月正式发布Hologres V0.9版本，新增功能具体如下：

- **引擎增强**
  - 数据类型丰富。
    - JSON和JSONB类型。
    - 时间类型：interval、timetz、time
    - 网络类型：inet
    - 货币类型：money
    - PG系统类型：name、uuid、oid
    - 其他：bytea、bit、varbit

    详情请参见数据类型汇总。

- 函数类型丰富，包括兼容PG的函数和Hologres扩展函数。
  - 数组函数：新增array_length和array_positions。详情请参见数组函数。
  - 查看表和DB存储大小的函数：pg_relation_size和pg_database_size，详情请参见查看表和DB的存储大小。
- 支持通过Hologres SQL命令语句将Hologres数据导出至MaxCompute，实现数据归档。详情请参见通过SQL导出数据至MaxCompute。
- 支持Hologres Binlog 订阅（Beta），详情请参见订阅Hologres Binlog。
- 支持动态修改表bitmap索引和字典编码，支持根据数据特征自动创建字典编码。详情请参见ALTER TABLE。
- 发布Hologres Client Library，适用于大批量离线、实时数据同步至Hologres以及高QPS的点查场景，实现自动攒批，提高吞吐，详情请参见通过Holo Client读写数据。
- 优化JDBC写入链路和查询优化器，显著提升引擎写入效率。
- BI生态连通友好性提升，支持Tableau Server，Superset等更多BI工具，满足多种业务分析需求。

- **安全增强**
  - 支持STS账号通过角色的方式登录Hologres，实现除云账号外等更安全更多元的账号登录体系，详情请参见RAM角色授权模式。

Hologres 2020年10月新增功能

2020年10月正式发布Hologres V0.8版本，新增功能具体如下：

- **引擎增强**
  - 支持通过 `CREATE VIEW` 语句创建视图。您可以基于一张表、多张表（包含内部表和外部表）或者其他视图创建视图，详情请参见VIEW。
  - 新增SERIAL、DATE、TIMESTAMP、VARCHAR(n)及CHAR(n)数据类型。同时，MaxCompute外部表数据支持Array类型映射，详情请参见数据类型汇总。
  - 支持 `INSERT ON CONFLICT` 功能，您可以根据主键配置，在插入数据时更新或跳过重复数据，详情请参见INSERT ON CONFLICT(UPSERT)。
  - 支持 `TRUNCATE` 功能。
  - 内置Proxima向量检索引擎，支持海量数据向量检索功能，该功能目前处于Beta版本，详情请参见Proxima向量计算。

- **安全增强**
  - 新增数据脱敏功能。您可以配置多种脱敏策略，对电话、地址或身份证等私密信息进行脱敏，详情请参见数据脱敏（Beta）。
  - 对接云监控，支持自定义指标监控和一键报警，详情请参见云监控。

- **MaxCompute外部表查询约束与限制**
  - 查询MaxCompute分区表时，扫描分区数的最大值为512个（0.8之前版本为50个）。
  - 每个查询中，最大的底层数据扫描量为200GB（与外部表的数量以及字段数无关，0.8之前版本为100GB）。

  更多内容请参见约束和限制。

1.2. 默认行为变更说明

本文将会记录Hologres每个版本中的默认行为变更。

> **说明** 参数更名是向下兼容的，即低版本的参数名仍可使用，但是更推荐使用新参数名。

V1.1版本默认行为变更说明（2022年04月）

内存优化

Hologres从 V1.1.53版本开始，针对内存进行优化，优化后端运维指标的汇报，相比之前版本常驻内存更少，从而提高计算使用内存。若有需要可以升级至Hologres V1.1.53版本。

V1.1版本默认行为变更说明（2022年03月）

Fixed Plan点查场景优化

Hologres从 V1.1.49版本开始，针对Fixed Plan点查场景进行了优化，在大规模点查的情况下提升了30%以上的吞吐。若有需要请升级Hologres实例至 V1.1.49及以上版本，Fixed Plan使用详情请参见Fixed Plan加速SQL执行。

V1.1版本默认行为变更说明（2022年03月）

分区子表绑定父表进行属性检测

Hologres从 V1.1.42版本开始，对分区子表绑定（ATTACH）父表时，将会进行更严格的属性检测，包括主键（PK）、索引、非空约束等，如果子表的属性与父表不一致，则会报错无法绑定。请在建表前确保分区子表的属性与父表一致，关于分区子表和父表的约束请参见CREATE PARTITION TABLE。

典型场景示例如下，子表的Clustering Key和父表的Clustering Key不一致，会报错无法绑定。

```
--假设已有父表和其子表，其DDL如下：
BEGIN;
CREATE TABLE public.hologres_parent(
  a int,
  b text not null,
  c timestamptz not null,
  ds text,
  PRIMARY KEY(a,ds)
)
 PARTITION BY LIST(ds);
CALL set_table_property('public.hologres_parent', 'orientation', 'column');
CALL set_table_property('public.hologres_parent', 'distribution_key', 'a');
CALL set_table_property('public.hologres_parent', 'clustering_key', 'b');
CALL set_table_property('public.hologres_parent', 'event_time_column', 'c');
CREATE TABLE public.hologres_child PARTITION OF public.hologres_parent FOR VALUES IN('20201
103');
COMMIT;
--创建临时分区子表
BEGIN;
CREATE TABLE IF NOT EXISTS public.tmp_hologres_child(
  a int,
  b text not null,
  c timestamptz not null,
  ds text,
  PRIMARY KEY (a,ds)
) ;
CALL set_table_property('public.tmp_hologres_child', 'orientation', 'column');
CALL set_table_property('public.tmp_hologres_child', 'distribution_key', 'a');
CALL set_table_property('public.tmp_hologres_child', 'clustering_key', 'a,b');
CALL set_table_property('public.tmp_hologres_child', 'event_time_column', 'c');
COMMIT;
--导入外表数据至临时分区子表
insert into public.tmp_hologres_child select * from foreign_table where ds='20201103';
--删除原分区子表，并将临时分区子表绑定至父表上
BEGIN;
DROP TABLE IF EXISTS  public.hologres_child;
ALTER TABLE public.tmp_hologres_child RENAME TO hologres_child;
ALTER TABLE public.hologres_parent ATTACH PARTITION public.hologres_child
FOR VALUES IN ('20201103');
COMMIT ;
--错误原因
ERROR: partition index hologres_child's immutable properties(e.g. clustering_key, event_tim
e_column) is consistent with parent.
  Hint: create partition with [create table ... partition of ...] to be consistent with par
ent.
```

V1.1版本默认行为变更说明（2021年12月）

新建实例公网Endpoint默认关闭

出于数据访问控制的安全要求，自2021年12月7日00:00:00起，新建实例的公网Endpoint默认关闭，默认不提供公网访问能力。如果您需要使用公网Endpoint连接Hologres，请在Hologres管理控制台手动打开。

V1.1版本默认行为变更说明（2021年11月）

单节点计算内存上限取消20GB限制

Hologres从 V1.1.24版本开始，计算节点（Worker Node）运行时内存取消单节点20GB限制，采用动态调整节点内存。计算节点定期检查当前内存水位，从而动态分配计算节点的内存上限，尽量保证运行时内存最大化分配，保障Query获得足够内存分配。Hologres升级到V1.1.24及以上版本后，若是执行Query时还报错超内存限制，在执行计划合理的前提下，说明内存已用到了上限，需要优化SQL或者扩容实例。

> **说明** 一个实例后端由多个节点组成，不同实例规格对应不同节点数，单个节点的总内存为64GB。内存会分为三部分，三分之一分配给计算运行时分配，三分之一分配给缓存，三分之一分配给元数据及常驻执行进程。

V1.1版本默认行为变更说明（2021年10月）

- Auto Analyze默认打开

  Auto Analyze在Hologres V0.10版本透出，经过多用户线上验证，具备生产可用状态，因此在Hologres V1.1版本中默认行为由关闭改为打开。升级实例不受影响，新创建实例默认为打开，同时相关参数名调整如下。

原参数名	新参数名	默认值
hg_experimental_enable_start_auto_analyze_worker	hg_enable_start_auto_analyze_worker	on

  关于Auto Analyze的使用请参见ANALYZE和AUTO ANALYZE。

- Table Group相关函数更名

  Resharding函数在Hologres V0.10版本透出，经过多用户线上验证，具备生产可用状态，在Hologres V1.1版本中相关函数名称调整如下。

原函数名	新函数名
hg_update_table_shard_count('table_name','table_group_name')	hg_move_table_to_table_group('table_name','table_group_name')

  - 关于Resharding的使用请参见Table Group与Shard Count操作指南。
  - 关于Table Group的使用，请参见Table Group设置最佳实践。

- MaxCompute外表访问行为变更

  在Hologres V0.10版本中，Hologres具备了全新的MaxCompute外表加速查询引擎，带来30%以上的性能提升，经过多用户线上验证，具备生产可用状态，因此新版本使用新的查询引擎查询MaxCompute外表。升级用户不受影响，新创建实例默认使用全新外表引擎。同时相关参数名调整如下。

原参数名	新参数名	备注
hg_experimental_enable_access_odps_orc_via_holo	hg_enable_access_odps_orc_via_holo	默认值为on。
hg_experimental_foreign_table_executor_max_dop	hg_foreign_table_executor_max_dop	默认值调整为与实例Core数相同，最大为128。

原参数名	新参数名	备注
无	hg_foreign_table_executor_dml_max_dop	V1.1版本新增，默认为32，针对涉及有外表的DML SQL生效。
hg_experimental_foreign_table_split_size	hg_foreign_table_split_size	默认值为64MB。
hg_experimental_foreign_table_max_partition_limit	hg_foreign_table_max_partition_limit	默认为512，即一次Query查询扫描最大分区为512。
hg_experimental_enable_write_maxcompute	无	V1.1版本默认为on，即默认可回写至MaxCompute，详情请参见通过SQL导出数据至MaxCompute。

有关参数的含义，请参见优化MaxCompute外部表的查询性能。

- pg_stat_activity表支持全局状态

  在Hologres V1.1之前版本中，pg_stat_activity表只记录单个接入节点（FE）的活跃连接状态，对于活跃查询的检查和处理并不方便，在1.1版本中，pg_stat_activity包含了全部接入节点的状态，有关pg_stat_activity活跃查询的管理，请参见Query管理。

- 连接数默认行为更改

  从Hologres V1.1版本开始，优化连接数默认行为，增加Superuser预留连接数，同时也优化HoloWeb连接池逻辑，强保证当连接数超过实例默认规格后，可以通过HoloWeb连接实例进行连接数管控和连接释放。关于连接数的使用请参见连接数管理。

- 修改idle_in_transaction_session_timeout默认参数值

  参数 `idle_in_transaction_session_timeout` 描述了事务进入idle状态后的超时行为，如果不设置参数值，默认不会做事务超时的释放，容易发生查询被锁死的情况。在Hologres V1.1版本中，`idle_in_transaction_session_timeout` 值默认设置为10分钟。关于参数的使用，请参见Query管理。

- 同一个事务（Transaction）内不支持数据DML和DDL同时出现

  Hologres只支持对DDL语句提供Transaction支持，如果在Transaction里包含了DML语句，会消耗事务资源，引起更多潜在不稳定状态。因此在Hologres V1.1版本中，如果将DML与DDL同时写在一个Transaction里，会有错误提示：`insert in ddl transaction is not supported now`，如以下SQL语句。

```
begin;
create table abc(id int);
insert into abc values(1); --报错insert in ddl transaction is not supported now
commit;
```

> **说明** 如果您在事务内DML和DDL已经混用，可以通过如下命令语句修改参数保持数据库级别兼容。
>
> ```
> alter database <db_name> set hg_experimental_enable_dml_in_ddl_transaction_block = on; --db_name为数据库名称
> ```

1.3. 关键缺陷通知

本文将为您介绍Hologres各版本相关缺陷的修复记录，包括问题描述、影响程度等。您可以通过报错或问题描述，检查您当前的业务中是否产生了相关问题，提前进行问题规避。建议您提交工单由对应技术支持协助您将产品升级到最新版本。

背景信息

- **缺陷及修复说明**
  - 缺陷内容向下穿透：当前版本存在的缺陷，在之前的版本中均存在。

    例如，0.9版本中存在某缺陷，在0.8或0.7等版本中均存在。
  - 缺陷修复向上包含：当前版本修复后的缺陷，在之后的版本中均已修复。

    例如，0.9版本中已修复的某缺陷，在0.10或1.1等版本中均已修复。
- **缺陷等级说明**
  - P0：建议立即升级，一旦触发会影响线上的使用（如查询、写入等操作）。
  - P1：推荐升级，提前规避相关问题。
  - P2：选择性升级，偶尔发生的问题，重启即可修复。

2022年06月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	Analyze OSS外部表时，出现OOM（Out Of Memory）。	在对OSS外部表执行Analyze时，OSS行数获取接口会默认采样的行数较大（超过3万行），从而出现OOM。	出现版本： 1.1.76及以下版本。 修复版本： 1.1.77及以上版本。	建议升级到新版本。
P2	含有 `in` 查询的SQL，当 `in` 中常量类型和列的实际类型不一致时，报错：`internal error: Invalid filter value type int32 for column type int16` 。 示例SQL如下。 `create table test(pid smallint); insert into test values (1); select pid from test where pid not in (4, 5);`	`in` 算子指定的常量数据类型和原列类型不同时，优化器（QO）没有对常量进行 `cast` 类型转换，导致在执行器（QE）端报错。	出现版本： 1.1.73及以下版本。 修复版本： 1.1.74及以上版本。	• `in` 中的常量类型和列的类型保持一致。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	创建OSS外部表时，只选择部分字段创建外部表，创建时报错：`Open ORC file failed for schema mismatch`。	选择部分OSS字段创建外部表时，引擎对部分外部表的支持有限制，只能选择全部字段。	出现版本： 1.1.73及以下版本。 修复版本： 1.1.74及以上版本。	• 选择全部字段创建外部表。 • 建议升级到新版本。
P2	删除某一段区间的数据之后（如删除某个分区），立即对同一张表执行`insert`命令，写入的速度变慢。	当删除了一个区间或者一段连续的值之后，此时Compation还未全部完成，执行`insert`命令时，会先去查询该时间段是否有相同的记录，直到遇到第一条没被删除的记录才能停下来，如果查询的Key附近有大量连续删除的记录，会消耗很多时间在遍历这些记录上，导致`insert`命令写入速度慢。	出现版本： 1.1.70及以下版本。 修复版本： 1.1.71及以上版本。	建议升级到新版本。

2022年05月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	查MaxCompute表报错：`Timestamp overflow detected while converting timestampfrom orc VectorBatch to arrow`。	在MaxCompute表中有TIMESTAMP类型，使用Tunnel写入后TIMESTAMP精度会变成纳秒，目前Hologres暂不支持精度为纳秒的TIMESTAMP，导致报错。	出现版本： 1.1.69及以下版本。 修复版本： 1.1.70及以上版本。	• 修改MaxCompute表的TIMESTAMP类型为DATATIME类型。 • 建议升级到新版本。
P2	查询OSS Parquet数据时，通过`count`语句每次查出的结果不一致，而OSS数据没发生过变化。	Hologres读OSS Parquet文件时，接口版本较老，导致读取非Null数据中会随机出现Null值，从而查询结果错误。	出现版本： 1.1.67及以下版本。 修复版本： 1.1.68及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	在Hologres中使用SQL方式将数据回写MaxCompute时报错：`Blocks not match, server:xx client yy`。	在回写MaxCompute时，超时时间默认为300s，导致产生了空的Block，从而出现报错。	出现版本： 1.1.64及以下版本。 修复版本： 1.1.65及以上版本。	• 通过执行如下命令修改超时时间。 `alter server odps_server options (add socket_timeout '600');` 修改超时时间为`600s`，不容易产生空Block。 • 建议升级到新版本。
P2	Hologres V1.1版本在对MaxCompute外部表增加多列时报错：`not support alter table with multi commands`。示例SQL如下。 `ALTER FOREIGN TABLE bank ADD COLUMN cons_conf_idx float8, ADD COLUMN euribor3m float8;`	Hologres V1.1版本增加了对外部表`add column`状态的检查，当增加多列时，状态检查错误，导致增加列失败。	出现版本： 1.1.1至1.1.58版本。 修复版本： 1.1.59及以上版本。	• 一次只新增一个列或者使用IMPORT FOREIGN SCHEMA语法刷新外部表。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	使用 `to_date` 函数带where过滤条件查询时报错： `invalid value \"\" for \"YYYY\" HGERR_detl Field requires 4 characters, but only 0 could be parsed` 。示例查询SQL如下。 `select * from public.test where to_date(content, 'YYYYMMDD' ) BETWEEN '2021-10-22' AND '2021-10-23' limit 10;`	使用 `to_date` 函数带where过滤条件时，where过滤的数据被识别为了非法数据进行转换，导致报错。	出现版本： 1.1.58及以下版本。 修复版本： 1.1.59及以上版本。	建议升级到新版本。
P2	并发读取MaxCompute加密的表时，出现报错： `failed to load row group data from file pangu` 。	在并发读取MaxCompute加密表时，Reader并发解析加密对象，导致解密错误。	出现版本： 1.1.57及以下版本。 修复版本： 1.1.58及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	对NUMERIC或者DECIMAL类型的字段执行求余（%）计算，且下推至HQE中执行，导致计算结果不正确。	HQE不支持NUMERIC和DECIMAL类型的求余，但未做类型校验，导致结果出错。	出现版本： 1.1.55及以下版本。 修复版本： 1.1.56及以上版本。	• 不对NUMERIC和DECIMAL类型的字段执行求余计算。 • 建议升级到新版本。 说明 升级后仍然不支持对NUMERIC和DECIMAL类型的字段执行求余计算，会直接报错。

2022年04月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	通过JDBC订阅Hologres Binlog，启动JDBC消费Binlog作业pgreplicationstream.start()在数据库端同时执行 `drop table xx;` 删除对应表，导致实例出现短暂重启。	订阅Binlog时删除表，会导致订阅时表不存在，但是订阅Binlog需要获取表的table_properties，导致空指针，出现实例重启现象。	出现版本： 1.1.54及以下版本。 修复版本： 1.1.55及以上版本。	• 订阅Hologres Binlog时不删除表。 • 建议升级到新版本。 说明 升级Hologres实例后若是订阅Binlog时删除表，会报表不存在。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	分区子表执行 `detach` 父表之后，再 `attach` 至同一个父表上，无法 `attach` 。	通过 `CREATE TABLE <table_name> PARTITION OF <parent_table>` 命令创建的分区子表，不会继承父表的Table Group属性，当分区子表 `detach` 后再进行 `attach` 时，校验出子表与父表的Table Group属性不一致，导致无法进行 `attach` 。	出现版本： 1.1.52及以下版本。 修复版本： 1.1.53及以上版本。	建议升级到新版本。
P2	当使用 `Grouping sets` 和多个 `Count Distinct` 对分区表进行查询时，查询结果不正确。	`Grouping sets` 和多个 `Count Distinct` 对分区表查询时，优化器未对分区进行裁剪，导致分区过滤条件未命中，从而出现结果不正确。	出现版本： 1.1.52及以下版本。 修复版本： 1.1.53及以上版本。	建议升级到新版本。

2022年03月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	通过DLF查询OSS外部表时报错： `failed to import foreign schema` ，创建user-mapping后能正常查询。	没有设置user-mapping时，鉴权接口传递权限错误，导致查询报错。	出现版本： 1.1.50及以下版本。 修复版本： 1.1.51及以上版本。	• 显式设置user-mapping，详情请参见通过DLF读取OSS数据。 • 建议升级到新版本。 **说明** 升级到新版本后不需要显示创建user-mapping即可正常访问相关数据。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	PrepareStatement模式下查询SQL报错：`unrecognized node type: 0` 或者Hologres实例出现短暂重启。	PrepareStatement模式下可以对反复执行的SQL生成Plan cache，减少接入端的开销。而在当前版本PrepareStatement模式对SQL的Plan cache获取不及时，导致查询出错。	出现版本： 1.1.47至1.1.50版本。 修复版本： 1.1.51及以上版本。	• 使用 `jdbc:postgresql://<host>:<port>/<dbname>preferQueryMode=simple` 命令将JDBC连接串修改为Simple模式。 • 建议升级到新版本。
P0	Blink或者Flink RPC模式写入Hologres时报错：`failed to create channel into server xxx，connection refused to rpc proxy endpoint` 。	使用Blink或者Flink RPC模式写入Hologres时，接口未返回Rpcproxy端口，导致写入报错。	出现版本： 1.1.50及以下版本。 修复版本： 1.1.51及以上版本。	• 将Flink作业切换为JDBC写入模式，详情请参见Hologres结果表。 • 建议升级到新版本。
P2	执行含有 `union all` 的Join SQL命令时报错：`internal error: 0 shard end shard value: xxx doesn\'t` 。	含 `union all` 的JOIN SQL在推导执行计划时错误，导致报错。	出现版本： 1.1.49及以下版本。 修复版本： 1.1.50及以上版本。	建议升级到新版本。
P2	使用 `json_array_elements` 函数且SQL中含有Join命令时，报错：`Duplicate keys detected when building hash table` 。	执行引擎（QE）在执行Join算子时会构建哈希表，但是实际读数据时，没有正常过滤 `json_array_elements` 处理后的数据，导致读取的数据有重复从而报错。	出现版本： 1.1.49及以下版本。 修复版本： 1.1.50及以上版本。	建议升级到新版本。
P2	执行Join SQL时报错：`Explicit remote seek from a source is not supported` 。	Join SQL生成的执行计划（通过 `explain sql` 查看）是Nested Loop Join时，执行引擎获取Nested Loop Join相关的执行计划错误，导致执行报错。	出现版本： 1.1.49及以下版本。 修复版本： 1.1.50及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	SQL过滤条件中含有 `not in` 时，查询结果中仍然含有 `not in` 过滤的数据。示例如下。 `create table if not exists test(id bigint, value int); select id from test where id in (238024008,276941010) and id not in (238024008) and value in (1, 2, 3);`	优化器在生成执行计划时，对 `not in` 处理错误，生成了错误的执行计划，导致 `not in` 过滤条件丢失，结果出错。	出现版本： 1.1.48及以下版本。 修复版本： 1.1.49及以上版本。	建议升级到新版本。
P1	查询慢Query日志时缺少日志，但是监控信息上却显示延迟和QPS。	同一事务（Transaction）中不同Query有相同Query ID，元仓收集Query去重后只保留了一条Query，导致其他Query丢失。	出现版本： 1.1.46至1.1.47版本。 修复版本： 1.1.49及以上版本。	建议升级到新版本。
P2	SLPM权限模型下，修改Schema名称时执行 `CALL slpm_rename_schema ( old_name, new_name )` 命令，报错：`UPDATE is not supported` 。	SLPM权限模型下修改Schema时，权限接口判断错误，导致执行报错。	出现版本： 1.1.47及以下版本。 修复版本： 1.1.48及以上版本。	建议升级到新版本。

2022年02月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	SPM或者SLPM模式下，开启数据脱敏后，进行Auto Analyze或者Analyze失败。	后端会使用表Owner去执行Auto Analyze，但SPM或者SLPM模式下，表的Owner是Developer，没有登录权限，而对脱敏列采样时会走PQE，导致Auto Analyze或者Analyze失败。	出现版本： 1.1.1至1.1.46版本。 修复版本： 1.1.47及以上版本。	• 关闭数据脱敏。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	Analyze外部表时，外部表分区太多（通常是多级分区场景）报错超过分区限制（大于512分区），导致Analyze失败。	Analyze时未对外部表分区进行相应裁剪，导致Analyze失败。	出现版本： 1.1.1至1.1.46版本。 修复版本： 1.1.47及以上版本。	• 若是外部表分区数在1024以内，可以先将外表分区限制调大再执行Analyze。 • 建议升级到新版本。 说明 升级后默认Analyze最多512个外部表分区，若是需要更多分区，请将Analyze分区限制数调大，详情请参见优化MaxCompute外部表的查询性能。
P1	执行 `explain analyze` SQL命令时，结果中 `partitions selected` 为0，与实际命中分区不符。	生成执行计划时对 `partitions selected` 判断错误，导致结果为0。	出现版本： 1.1.1至1.1.46版本。 修复版本： 1.1.47及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	查看慢Query日志时，无法显示查询读取的行数（read_rows）、返回行数（result_rows）等信息。	元仓采集信息不全导致。	出现版本： 1.1.1至1.1.46版本。 修复版本： 1.1.47及以上版本。	建议升级到新版本。 **说明** Hologres V1.1.36版本开始可以通过GUC参数查看，V1.1.47版本后可以不使用GUC参数。
P2	使用JDBC PrepareStatement模式时，`insert`或者`select`多个值时，多执行几次（大于3次）结果出现错行错列，而一次`insert`或者`select`一个值分多次执行时结果正确。示例如下：一条SQL包含32个values一次写入，总共写4次，每次这32行数据在values中的顺序随机变化。而一条SQL只包含一个values，分32条sql写入，结果正确。	PrepareStatement模式下对多个values进行多次`insert`或者`select`时，优化器（QO）生成了错误的执行计划，导致结果出错。	出现版本： 1.1.46及以下版本。 修复版本： 1.1.47及以上版本。	• Query符合Fixed Plan特征时，可以开启Fixed Plan加速SQL执行。 • 将PrepareStatement模式更改为Simple模式。 • 建议升级到新版本。
P2	执行非Join的SQL（例如含有count distinct）时，报错：`error: Hash32 shard function does not support decimal or fixed binary type`。	非Join的SQL也可能会使用Shard Function生成执行计划，而目前Shard Function目前不支持NUMERIC等类型，导致部分非精确类型在执行时报错。	出现版本： 1.1.46及以下版本。 修复版本： 1.1.47及以上版本。	建议升级到新版本。
P1	在使用`key = max(key)`时，出现的结果不符合预期，一直只出现一行数据，使用`key in max(key)`时符合预期。	优化器在生成执行计划时，会将`key = max(key)`，转化成`order by id asc limit 1`，这种查询永远只有一行数据，导致结果不符合预期。	出现版本： 1.1.46及以下版本。 修复版本： 1.1.47及以上版本。	• 使用`key in max(key)`。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	在Hologres中通过 `exec_external_sql` 使用MaxCompute语法时，`exec_external_sql` 中包含未在实例中创建的server名称，如 `select exec_external_sql('odps_test_server', 'seahawks', 'select 1;');` 其中 `odps_test_server` 未在实例中创建过，导致实例出现重启。	接入节点在查找foreign_server 时，遇见没有创建的server名称，无法判断结果，导致实例重启。	出现版本： 1.1.46及以下版本。 修复版本： 1.1.47及以上版本。	• 使用Hologres内置的server名称：`odps_server`。 • 建议升级到新版本。 **说明** 升级完成后使用未创建的server仍然不能执行成功，但实例不会重启。
P2	非PostgreSQL来源（如JDBC）的DDL有SQL代码注释，示例：`create table ttxwsx1(i int); --comments -- xxxxx` 执行成功后，写入或者查询时出现卡死的现象。	DDL命令末尾有注释，会使得同一行最后的分号失去命令间的分隔作用，导致新生成的命令追加到注释后面失效，从而使得SQL不合法，导致节点间Replay失败，造成写入或者查询卡死。	出现版本： 1.1.45及以下版本。 修复版本： 1.1.46及以上版本。	• 删除DDL语句最后的注释。 • 建议升级到新版本。
P1	按照主键点查方式查询行存表时，存在一定概率场景，出现部分行存数据查询不到的情况。	行存表在做后台文件compaction时，在处理并发场景有缺陷，致使部分索引文件定位有误，导致部分行存数据查询不到。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	建议升级到新版本
P2	Hologres实例升级至 V1.1版本后，查询MaxCompute外部表，当外部表有多级分区时（一般3级分区），SQL过滤条件中带有or，查询相比V0.10版本变慢（之前查询只需要几秒钟）或者出现OOM。	Hologres V1.1版本，在多级分区过滤中，优化器对or条件生成的算子无法识别，导致生成的filter为空，即不做任何过滤，从而扫描了所有分区，导致查询变慢或者出现OOM。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	建议升级到新版本

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	CPU占用不高时内存也长期处于高水位，通过监控发现QPS比较高（几百及以上），但是连接数只用了几十个，即一个Connection保持几百个QPS的速度执行SQL。	当执行SQL时，优化器会去获取表的statistics，当一个Connection保持几百个QPS的速度执行SQL，且Connection长期不关闭，导致获取statistics时泄漏，造成内存高水位。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	• 通过添加 `set hg_experimental_alway_show_execution_statistics=off;` 参数解决。 • 建议升级到新版本。
P2	SQL中含有 `not like xxx%` 条件，但是查询结果中仍然出现 `not like` 过滤后的数据。	优化器在生成执行计划时，对like相关的函数预处理规则出错，进行了错误的改写，导致结果不正确。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	• 带like的SQL出错时可以通过添加 `hg_experimental_remove_redundant_cmp=off;` 参数解决。 • 建议升级到新版本。
P1	STS账号登录时，报错：`Cloud authentication failed`，并且检查账号密码等都没有填写错误。	账号认证接口对STS账号的状态判断错误，导致报错。	出现版本： 1.1.43至1.1.44版本。 修复版本： 1.1.45及以上版本。	建议升级到新版本
P2	在HoloWeb中，通过SQL方式回写MaxCompute时报错：`ERROR: SqlTask Run failed: create instance failed!`，通过jdbc/psql等执行不报错。	HoloWeb接口在鉴权 `exec_external_sql` 时失败，导致回写报错。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	• 通过jdbc/psql方式进行回写。 • 建议升级到新版本。
P2	查询MaxCompute外部表时报错：`not an ORC file`。	MaxCompute表是Stream Table，通过Tunnel写入，Hologres查询MaxCompute外部表时接口meta未及时更新，导致查询失败。	出现版本： 1.1.44及以下版本。 修复版本： 1.1.45及以上版本。	建议升级到新版本

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P0	在应用侧数据写入完成，但引擎侧数据写入进程崩溃，有概率存在数据丢失，用户查询时发现数据缺少。	正常流程是用户写数据，WAL（Write Ahead Log）落盘后才返回给上层调用，表示写入完成，保证数据持久化和一致性。但当落盘进程写入超时触发系统重试后，数据会首先写入内存缓存部分，并返回给上层调用，如果此时内存缓存进程崩溃后，会造成应用层返回成功，但实际数据存储层丢失的问题。	出现版本： 0.8及以下版本。 修复版本： 0.9及以上版本。	建议升级到新版本
P1	实例写入和查询数据时失败并报错：`ERROR: Invoke StoreMasterClient failed:ERPC_ERROR_CONNECTION_CLOSED`。	出现报错后，业务侧进行Query重试叠加后端接入节点（FE）重试，导致请求量太高，Store Master（元数据管理）处理不及时而报错。	出现版本： 1.1.43及以下版本。 修复版本： 1.1.44及以上版本。	• 使用`set optimizer_join_order=query`命令，暂时绕过。 • 建议升级到新版本。
P2	新增一列类型为DECIMAL且不指定精度的列，如`alter table add column c0 decimal;` Query执行成功，但是查询新加的列时出现报错：`Schema fields[] has type decimal(x,y) but decimal(x1, y1) is expected.`。	当前新增列不支持DECIMAL不指定精度，但是新增列（Add Column）时没有做精度校验，导致查询报错。修复后在新增列时会对精度校验，未指定精度会报错。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	• 新增列时含有DECIMAL字段时需要指定精度。 • 建议升级到新版本。
P0	当AccessKey被禁用后，仍然能使用被禁用的AccessKey访问Hologres实例。	AccessKey接口对于禁用的AccessKey状态调用错误，导致禁用的AccessKey被当成了正常的AccessKey使用。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	• 取消账号的访问权限。 • 建议升级到新版本。
P2	建表时有Default字段，使用`copy`命令语句时，实例出现重启。	`copy`功能不支持建表时带有Default值，导致实例OOM发生重启。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	执行有外表关联的INNER JOIN查询语句时，报错找不到某一列，如：`ERROR: column "id" does not exist`，而SQL中并没有这一列。	优化器在生成执行计划时，对于等价表达式的推导不对，没有输出的列也作为了等价表达式的推导，导致报错。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	建议升级到新版本。
P1	使用行列共存的表，带有复杂的Nested Loop Join，出现实例重启后又快速恢复。	优化器在检测行列共存的表时，没有生成正确的执行计划，导致报错从而触发实例重启。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	• 不使用行列共存的表 • 建议升级到新版本。
P1	多表（如6个表）Join的复杂导入作业在手动取消后，CPU使用率仍然为100%，持续几个小时不结束，执行`drop table`时也卡住。	比较复杂的Query，执行计划包括Hash Join算子，涉及到的数据量很大，后端出现锁死，导致取消后仍然在后端运行中。	出现版本： 1.1.42及以下版本。 修复版本： 1.1.43及以上版本。	• 重启实例。 • 建议升级到新版本。

2022年01月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	数据通过DataHub写入Hologres分区表时，分区表未提前创建分区子表，Hologres实例重启。	DataHub写入Hologres分区表时，写入接口未做分区校验，引发实例Coredump。	出现版本： 1.1.41及以下版本。 修复版本： 1.1.42及以上版本。	• 在Hologres中提前创建好分区子表再写入数据。 • 建议升级到新版本。 说明 若是分区表，升级后仍然需要提前创建分区子表才能写入数据。
P2	分区子表通过 `attach` 到分区父表后，查分区表时报错： `partition_table_missing` 。	分区子表的建表属性与分区父表不一致（比如not null约束、PK设置，Clustering Key设置等），在 `attach` 时没有对属性进行校验，导致查询报错。	出现版本： 1.1.41及以下版本。 修复版本： 1.1.42及以上版本。	• 建分区子表时，子表的Schema和属性需要同 `attach` 的分区父表保持一致。 • 建议升级到新版本。
P1	使用JDBC PreparedStatment模式时，SQL中的 `where` 命令中含有"<"或者">"过滤条件，执行后出现实例重启。	使用JDBC PreparedStatment模式时， `where` 中的"<"或者">"过滤条件会在生成执行计划时，转换成INTERVAL，在转换时遇见空指针，导致SQL出错引发实例重启。	出现版本： 1.1.0至1.1.40及以下版本。 修复版本： 1.1.41及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	使用JDBC PreparedStatment模式时，`IN`条件超过100个时，有一定概率结果出错或者不符合预期。	当使用JDBC PreparedStatment模式时，`IN`条件超过100个时，生成执行计划时会错误的把`IN`条件给删除，导致数据结果出错。	出现版本： 1.1.0至1.1.40及以下版本。 修复版本： 1.1.41及以上版本。	建议升级到新版本。
P2	行列共存的表使用`IN`条件查询时报错：`An I/O error occurred while sending to the backend.`，使用行存表则不会报错。	当`IN`条件中的字段类型是TEXT时，并且该字段设置了Bitmap，导致行列共存的表生成了错误的执行计划，从而报错。	出现版本： 1.1.0至1.1.40及以下版本。 修复版本： 1.1.41及以上版本。	• 使用行存表。 • 建议升级到新版本。
P2	执行SQL，`where`条件中有`and`连接`in`和`=`时报错：`ERROR: serialized_error_msg is null`。示例SQL： `SELECT * FROM public.conflict_1 where a in (1,31) and a=1;`	后端在判断`and`两侧`in`和`=`的数据类型时判断错误（示例：假如是 a = 1 就是一个int类型，a in (1,2,3) 就是一个array类型），导致执行失败。	出现版本： 1.1.0至1.1.40及以下版本。 修复版本： 1.1.41及以上版本。	建议升级到新版本。
P2	修改分区子表的生命周期（TTL）后出现报错：`Invoke StoreMasterClient failed:ERPC_ERROR_CONNECTION_CLOSED`。	修改子表TTL时，元数据管理器Store Manager（SM）检验Schema变动时出错，导致SQL出现报错。	出现版本： 1.1.0至1.1.40及以下版本。 修复版本： 1.1.41及以上版本。	• 暂不修改子表TTL。 • 建议升级到新版本。
P2	使用`DROP`语句删除表时报错：`invalid table id`，重试时报错：`SE object lock failed`。	一个实例会有多个接入节点，执行SQL时，是先在一个节点执行，再去其他节点重放（reply），当某个节点因为版本等原因无法跟其他节点保持元数据信息一致时，会进行重试（retry）。当并发执行`drop table`时，会触发节点的主动，retry时没有释放表锁导致报错。	出现版本： 1.1.39及以下版本。 修复版本： 1.1.40及以上版本。	• 串行执行`drop table`等DDL。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	开启Auto Analyze功能之后，实例没有明显上涨的QPS，出现报错：`database is not accepting commands to avoid wraparound data loss in database`。	开启Auto Analyze功能之后，接入节点的系统表没有及时执行`auto vacuum`，导致后台不断提交带事务的SQL，导致实例报错。	出现版本： 1.1.38及以下版本。 修复版本： 1.1.39及以上版本。	建议升级到新版本。

P2	基于分区表创建视图，并对分区列做`cast`，导致不能使用静态分区裁剪，导致扫描所有分区性能变差。示例SQL如下：
	`--建view`
	`create view test_partition_table_view asselecttest_partition_table.ds::text as dsfromtest_partition_table;`
	`--查询sql`
	`select * from test_partition_table_view where ds='20211116';`
		封装成View之后，在优化器中的过滤条件是基于`cast`之后的列，而非分区列。分区裁剪只能对分区列生效，导致性能变差。	出现版本：
			1.1.38及以下版本。
			修复版本：
			1.1.39及以上版本。
				• View中不对分区列进行`cast`。
				• 建议升级到新版本。

P2	• 当日期是周日时，执行`to_char(xxx, 'Day')`函数，实例发生重启。 • 当日期是周日时，执行`to_char(xxx, 'D')`函数，结果有误。	当日期是周日时，`to_char()`函数的底层执行逻辑是`toDayOfWeek()`，其返回值为`7`，发生数组越界，导致Hologres实例重启或者结果有误。	出现版本： 1.1.36及以下版本。 修复版本： 1.1.37及以上版本。	建议升级到新版本。
P1	实例开启数据脱敏后，子查询（Sub Query）中含有CTE函数，实例短暂出现连接报错或者I/O口报错。	递归调用处理CTE函数时，数据脱敏处理不正确，导致Hologres实例重启。	出现版本： 1.1.36及以下版本。 修复版本： 1.1.37及以上版本。	• 关闭数据脱敏。 • 建议升级到新版本。

2021年12月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P0	为TEXT类型字段设置Dictionary索引时，实例出现短暂重启，示例SQL如下。 `call set_table_property('tbl', 'dictionary_encoding_columns', 'a');` ，其中a是TEXT类型。	Hologres会给TEXT类型的字段默认设置Dictionary，即为auto属性，手动再给TEXT类型指定Dictionary时，会变为on属性，导致文件状态不一致，无法进行压缩合并（Compaction），从而引发Coredump。	出现版本： 1.1至1.1.35及版本。 修复版本： 1.1.36及以上版本。	• 手动设置dictionary时，设置为 `auto` 属性，即：`call set_table_property('tbl', 'dictionary_encoding_columns', 'a:auto');`。 • 建议升级到新版本。

P2	查看慢Query日志时，无法显示查询读取的行数（read_rows）、返回行数（result_rows）等信息。	元仓采集信息不全导致无法显示。	出现版本：
			1.1至1.1.35及版本。
			修复版本：
			1.1.36及以上版本。
				建议升级到新版本，且需要在查看慢Query的SQL前添加如下命令。
				`set hg_experimental_force_sync_collect_execution_statistics = on;`
				`alter database <dbname> set hg_experimental_force_sync_collect_execution_statistics = on;`

P1	当SQL的where条件中含有 `case when xx in ('')` 时，结果不正确。	Hologres会默认对TEXT类型构建Bitmap，且该列是Nullable属性的情况下，后端对 `case when xx in ('')` 生成了错误的执行计划，导致结果不正确。	出现版本： 1.1.35及以下版本。 修复版本： 1.1.36及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	报错： `Cannot reserve capacity larger than 2^31 - 1 for binary\n` 。	Hologres会默认对TEXT列构建Dictionary字典编码，当插入的字段太大（单字段超过2GB）时，导致构建的Dictionary过大，查询时报错。	出现版本： 1.1.35及以下版本。 修复版本： 1.1.36及以上版本。	建议升级到新版本。

P1	查Binlog时，带有Binlog字段的SQL查主键（PK）字段时查不出数据，不带Binlog的SQL查PK字段时能查询出数据。示例查询（其中a是test表的PK字段）如下。
	`--带Binlog的SQL`
	`SELECT, hg_binlog_lsn ,hg_binlog_event_type,hg_binlog_timestamp_us FROM testwhere a = '723295948321120659' ;`
	`--不带Binlog的SQL`
	`"SELECT * FROM testwhere a = '723295948321120659' ;`
		后端优化器根据PK字段查询时生成了错误的执行计划，导致查询错误。	出现版本：
			1.1.35及以下版本。
			修复版本：
			1.1.36及以上版本。
				建议升级到新版本。

P2	实例在CPU负载满的情况下，在HoloWeb中无法查询活跃Query、活跃连接等信息。	在CPU负载满时，pg_stat_activity等系统表会受资源限制，导致查询失败。	出现版本： 1.1.35及以下版本。 修复版本： 1.1.36及以上版本。	建议升级到新版本。
P1	使用ANY数组为空时，Hologres实例出现重启。	对于ANY数组为空时后端处理不正确，导致实例Coredump。	出现版本： 1.1.35及以下版本。 修复版本： 1.1.36及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	Query包含Lead或Lag函数，同时函数的第三个参数缺省时报错：`Column column5 should be non-nullable but the values contain 1 nulls`。	执行器对于Lead和Lag函数的输出结果的Nullable推导不正确，导致报错。	出现版本： 1.1.34及以下版本。 修复版本： 1.1.35及以上版本。	建议升级到新版本。
P2	Flink写入Hologres时，有RoaringBitmap字段，写入很慢。	带有RoaringBitmap的写入链路没有在后端优化导致写入性能差。	出现版本： 1.1.35及以下版本。 修复版本： 1.1.36及以上版本。	• 不使用Roaring Bitmap。 • 建议升级到新版本。
P1	使用Roaring Bitmap时报错：`An I/O error occurred while sending to the backend`并且在CPU使用率很低时，内存使用率很高。	Roaring Bitmap存在内存泄漏。	出现版本： 1.1.34及以下版本。 修复版本： 1.1.35及以上版本。	• 不使用Roaring Bitmap。 • 建议升级到新版本。
P1	SQL中有`order by`时报错：`PlStmt Translation: Attribute number 4 not found in project list`。	`order by`会生成sort算子，优化器在生成执行计划时下推错误，导致无法生成执行计划，从而报错。	出现版本： 1.1.33及以下版本。 修复版本： 1.1.34及以上版本。	建议升级到新版本。
P1	使用Proxima查询时报错：`HGERR_code XX000 HGERR_msge internal error: record batches is empty`。	后端读取Proxima的文件状态有误，从而报错。	出现版本： 1.1.33及以下版本。 修复版本： 1.1.34及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	实例升级至1.1版本或者在1.1版本对实例执行升降配等重启操作后，第一次查询时，Query的速度变慢，查看执行计划，统计信息不准确。再次执行Query，统计信息正确且性能恢复。	实例升级重启后，第一次执行Query时未能拿到正确的统计信息版本，导致统计信息不准确，性能变差。	出现版本： 1.1至1.1.32版本。 修复版本： 1.1.33及以上版本。	• 可以多次执行Query，使得统计信息变正确，恢复性能。 • 建议升级到新版本。
P0	使用 `drop/truncate` 命令时同时查询表，造成实例重启。	查询结束到数据析构之间，发生表的 `drop/truncate` ，造成实例coredump，从而重启。	出现版本： 1.1.32及以下版本。 修复版本： 1.1.33及以上版本。	建议升级到新版本。
P1	升级至1.1版本后，多表（十几个表）Join出现OOM异常，且升级前运行正常。	优化器预估表的行数过多，导致执行器在初始化阶段OOM，无法进行下一步计算。	出现版本： 1.1至1.1.31版本。 修复版本： 1.1.32及以上版本。	建议升级到新版本。
P2	Serving点查场景，因为客户端凑批导致延迟变高。	每个Worker节点上只有一个点查写入节点，当请求都发到写入节点时容易产生凑批行为，而当前凑批上限过大，导致等待攒批耗时较长，造成点查延迟变高。	出现版本： 1.1至1.1.31版本。 修复版本： 1.1.32及以上版本。	建议升级到新版本。
P1	存储加密的表 `limit offset` 命令不加 `order by limit` 能查到结果，加了就查不出结果。	对于存储加密的表，没有按照文档正确的配置进行操作，生成了错误版本，导致内存表（MemTable）数据丢失，从而无法出结果。	出现版本： 1.1至1.1.31版本。 修复版本： 1.1.32及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	执行 `Truncate` 命令时，当表名称有大写时会报错找不到表。例如执行 `#truncate "Abc";` ，报错： `ERROR: relation "abc" does not exist` 。	当前 `Truncate` 对大小写处理逻辑错误。	出现版本： 1.1.30及以下版本。 修复版本： 1.1.31及以上版本。	建议升级到新版本。
P1	使用函数to_char、to_date和to_timestamp时报错： `time after 2282 not supported` 。	函数to_char、to_date和to_timestamp支持的时间范围是1925 ~ 2282年，超出时间范围就会报错。	出现版本： 1.1.30及以下版本。 修复版本： 1.1.31及以上版本。	建议升级到新版本，升级后可以通过GUC控制时间范围，支持所有时间的数据，如下所示。 • `set hg_experimental_functions_use_pg_implementation = 'to_char';` 。 • `set hg_experimental_functions_use_pg_implementation = 'to_date';` 。 • `set hg_experimental_functions_use_pg_implementation = 'to_timestamp';` 。
P1	SQL中有内连接（inner join），执行后运算结果偏少。	Join算子要求相同的join key数据分布推导在相同并发节点，实际执行时，数据分布推导错误，会错误的将相同数据Shuffle到不同的节点，导致 `Join` 结果错误，表现为结果比实际少。	出现版本： 1.1.30及以下版本。 修复版本： 1.1.31及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	执行SQL时报错：`Query could not generate plan by Hologres : PlStmt Translation: Attribute number 4 not found in project list`。	表连接时没有Join Key，导致执行计划生成失败报错。	出现版本： 1.1至1.1.27版本。 修复版本： 1.1.28及以上版本。	• 重新建表，然后执行`analyze table`命令。 • 建议升级到新版本。
P1	使用get_json_object函数时报错：`Column column0 should be non-nullable but the values contain 1 nulls`。	get_json_object函数的两个参数为非Nullable类型，但是UDF的结果可能为Nullable类型，在生成执行计划时，检查非Nullable失败，导致报错。	出现版本： 1.1.27及以下版本。 修复版本： 1.1.28及以上版本。	建议升级到新版本。
P1	报错：`ERROR: Build query failed: Table group [] from table must equals table group [] from QO.`。	执行计划生成中，DML节点对下游TG有信息要求，但下游某节点推断出的TG属性为NULL，没有满足DML的TG要求，导致报错。	出现版本： 1.1.27及以下版本。 修复版本： 1.1.28及以上版本。	建议升级到新版本。
P1	执行`DROP TABLE`命令时卡死，且重试之后CPU突然飙高。	实例开启了Auto Analyze，Auto Analyze会加share_update_exclusive锁， 同时Auto Analyze会使用连接，新的连接load_stats，会加access_shared_lock；这两个步骤期间，如果用户进行`DROP TABLE`就会卡死。	出现版本： 1.1.27及以下版本。 修复版本： 1.1.28及以上版本。	• 建议在业务低峰期开启Auto Analyze功能。 • 建议升级到新版本。

2021年11月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2（优化）	当实例重启后，查询部分数据结果不一致。	后端某一个节点重启后，需要与其他节点版本追齐，在追齐过程中，重启的节点版本较低，查询的还是原数据，导致结果查询不一致。优化后的行为是，当节点重启后，若是与其他节点版本不一致，则不提供服务，直到追齐版本后再提供服务，保证数据一致性。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	建议升级到新版本。
P2	MaxCompute数据导入时，执行 `set hg_experimental_foreign_table_split_size = 64; INSERT INTO public.lineitem SELECT * FROM public.odps_lineitem_1t;` 时内存很高或者报错OOM，当设置参数值为128时，则没有问题。	底层在Meta中，加载了所有StripesMeta导致内存飙高。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	• 暂时不使用 `set hg_experimental_foreign_table_split_size = 64;` 命令。 • 建议升级到新版本。
P1	当对Distribution Key或者Primary Key使用 `IN` 操作， `IN` 数组超过100个的时候，导致最终结果不正确。	当 `IN` 超过100个后，Shard pruning之后的Shard数量随机变化，导致生成的计划错误，结果不正确。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	• 减少 `IN` 的数量至100以内。 • 建议升级到新版本。
P1	在外部表数据导入内部表的时候，先 `insert` 然后 `delete` 历史数据，但是 `insert` 语句取出来的分区不是最新分区，导致插入数据为0。	在导入过程中，存在存储器异常问题，导致未获取到最新数据。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	建议升级到新版本。
P1	当行很宽，数据量超过数百MB时，单行记录就超出了RECORDBATCH记录批规格的上限，就会输出0行的RECORDBATCH，从而引发缺陷，实例进行重启。	当行很宽时，后端对行数的处理不够，导致实例进行重启。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	报错：`internal error: string decimal literal can not be tentative`。	SQL中有in表达式，例如：`select * from tbl where col in (1.11, 1.2, 1.333);` 当in表达式里面的DECIMAL精度不一致的时候，后端计算引擎处理结果不一致导致报错。	出现版本： 1.1.24及以下版本。 修复版本： 1.1.26及以上版本。	• in表达式里只用一个数据。 • 建议升级到新版本。
P2（优化）	报错：`org.postgresql.util.PSQLException: ERROR: Total memory used by all existing queries exceeded memory limitation 20132659200: xxxxx bytes used.`。	单个节点计算内存超过20GB的上限（单个节点总上限时64G，1/3用于计算，1/3用于缓存，1/3用于元数据）。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	在1.1.24版本支持单个节点内存弹性调整，后台会检测当前节点内存的使用状态，弹性调整计算内存大小，缓解20G计算内存上限的问题。但是Query还是报错，建议优化SQL或者扩容。
P1	报错：`ERROR: Query could not generate plan by Hologres : Query Translation: No attribute entry found due to incorrect normalization of query`。	执行的sql中，选中的列不在`GROUP BY`子句里面，但主键是`GROUP BY`子句的子集，Query无法生成计划，从而导致报错。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	建议升级到新版本。
P1	使用Flink或者Holo Client，往Binlog表里一次写入多条重复的数据，中间数据的Binlog丢失。	写Binlog表，其中有重复的数据时，后端执行器会只生成最后一条数据的Binlog，其他重复的数据会被忽略。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	建议升级到新版本。
P0	查询MaxCompute外部表时，最后两行数据会随机变化，数据类型是DECIMAL类型。	直读MaxCompute的ORC格式数据，当文件中存在DECIMAL类型，存储优化时，Hologres读出来的DECIMAL统计信息存在随机问题。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	报错：`Remote seek with parameters are not supported`。	sort算子默认具有rewindable属性，但底层并不支持，Query生成计划时报错。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	建议升级到新版本。
P1	在HologresV1.1版本设置了资源组，但是在跑Query时OOM（Out Of Memory），出现报错：`used by all existing queries exceeded memory limitation`，即使不跑任何Query，查询慢SQL和活跃Query这些也报错OOM。	QE内存使用超过阈值，跑新Query超过资源组配额，异常时会导致内存泄漏。	出现版本： 1.1至1.1.23版本。 修复版本： 1.1.24及以上版本。	• 重新新建资源组。 • 建议升级到新版本。
P2	偶发报错：`fail to setremoteost invalid remon ip`。	后台进程在检查IP白名单的变量时，变量没有初始化导致偶发报错。	出现版本： 1.1.23及以下版本。 修复版本： 1.1.24及以上版本。	• 请重试几次。 • 建议升级到新版本。
P1	对表执行`Analyze`或者`auto analyze`时，当表中存在同名但是大小写不同的列名时，报错：`CheckSchema failed`。	Frontend节点在从优化以后的树结构里面转化成PowerBuilderTree的时候对应列的序号找错，导致报错。	出现版本： 1.1.22及以下版本。 修复版本： 1.1.23及以上版本。	建议升级到新版本。
P1	执行多表`RightOuterJoin`的SQL时，不带有`limit`子句查询结果仅一条，加上`limit`子句之后会出现多条重复的数据。	实现`RightOuterJoin`的时候，在优化器中生成了错误的计划，导致最终结果数据重复。	出现版本： 1.1.22及以下版本。 修复版本： 1.1.23及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	在case when语句中，TEXT字段同时作为group by和agg的参数时，无法生成计划，出现报错： `ERROR: Query could not generate plan by Hologres : PlStmt Translation: Attribute number 46046320 not found in project list` 。	在case when中取法找到agg参数字段的colref导致计划无法生成。	出现版本： 1.1.22及以下版本。 修复版本： 1.1.23及以上版本。	建议升级到新版本。
P1	报错： `ERROR: internal error: Writting column: item_emb with array size: 682790219 violates fixed size list (32) constraint declared in schema` 。	const数组优化机制在SE没有判断导致执行出错。	出现版本： 1.1至1.1.21版本。 修复版本： 1.1.22及以上版本。	建议升级到新版本。
P0	使用 `insert on conflict do update set` 语句时，语句的subquery中将多行值赋给一行，例如 `SET(mes1, mes2) = (SELECT mes1, mes2 FROM insert_on_conflict_do_update_negative_source)` 导致实例重启。	subquery中将多行值赋给一行的语法产生了多表达式参数，此参数没有进行转换支持column id信息不存在，导致实例重启。	出现版本： 1.1.21及以下版本。 修复版本： 1.1.22及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	对于DECIMAL数据相乘报错：`code: kActorInvokeError msg: "HGERR_code 22003 HGERR_msge numeric field overflow HGERR_detl A field with precision 38, scale 36 must round to an absolute value less than 10^2. HGERR_ctxt HGERR_erno 2 HGERR_end" err_data { filename: "FunctionsCast.cc" lineno: 323 funcname: "DecimalOverflowCheck" sqlerrcode: 50331778 message: "numeric field overflow" detail: "A field with precision 38, scale 36 must round to an absolute value less than 10^2." context: "`。	对于DECIMAL类型的字段进行相乘，例如：`numeric(38, 18)`乘以`numeric(38, 18)`会得到n `umeric(38, 36)`，小数点保存太多位数导致溢出，从而报错。	出现版本： 1.1.21及以下版本。 修复版本： 1.1.22及以上版本。	• 使用`round`函数进行绕过。 • 建议升级到新版本。

2021年9-10月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P0	报错：`"database is not accepting commands to avoid wraparound data loss in database ""template0""`。	后台会给Query设置自增transation id，对于QPS高的实例，ID会超过INT上界，从而导致报错。	出现版本： 0.10.19至0.10.42版本。 修复版本： 1.1及以上版本。	建议升级到新版本。
P1	数据局部列更新入表偶发报错：`internal error: Record batch has 519 rows but length of columns is 7407`。	字段中包含TEXT[]，当前TEXT[]没有进行二层数组的`slice()`，导致其长度不正确，在执行`revserve()`命令时取到了-1导致超过容量。	出现版本： 0.10.41。 修复版本： 0.10.42及以上版本。	• 通过`set hg_experimental_skip_mem_table=on`命令暂时绕过。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	使用 `hg_create_table_like` 创建行存表，对行存表插入数据时报错：`ERROR: internal error: Cannot find index full ID: 51539607554 (table id: 12, index id: 2) in storages or it is deleting!` 。	行存表中有多个主键，获取表主键的时候是需要执行多次 `hg_create_table_like` ，将主键的列取出来放到Set集合里面，导致顺序丢失。	出现版本： 0.10.42。 修复版本： 0.10.45及以上版本。	• 手动执行 `create` 语句创建行存表。 • 建议升级到新版本。
P2	删除分区时报错：`FAILED: ERROR: query id[27xxxxxxxxxxxxx37] SE object lock failed` 。	删除分区时，Query被后端异常退出，导致报错。	出现版本： 0.10.41及以下版本。 修复版本： 0.10.42及以上版本。	• 不对该表进行操作。 • 联系值班重启实例。 • 建议升级到新版本。
P2	查询或者写入数据时报错：`ERROR: internal error: Invalid table id : 641 MDTableGroup`	一般是因为刚做完DDL，后端节点还在重启，这个时候执行DML，就会导致节点间的版本不一致而报错。	出现版本： 1.1.18及以下版本。 修复版本： 1.1.19及以上版本。	• 等待一段时间重试。 • 建议升级到新版本。

2021年08月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	表开启Hologres Binlog，且建表时Binlog的TTL设置比较小的时间，但表的存储数据一直增长（业务数据量并没有增加）。	建表（create table）时，显式指定的Binlog TTL未真正生效，默认为100年。	出现版本： 0.10版本。 修复版本： 1.1版本。	• 需要重新手动更改表Binlog TTL为较小的值，执行 `call set_table_property('schema.table', 'binlog.ttl', '86400');` • 建议升级到V1.1版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	列存表频繁进行Update、Delete、Insert on Conflict操作，引起存储空间持续增长。	Hologres为提高更高的效率，采用标记删除算法，当文件中被标记记录达到一定比率，会触发后台Compaction进程，进行空间的释放。Hologres存在缺陷，在某些情况下Compaction未启动。	出现版本： 0.10.25以下版本。 修复版本： 0.10.25及以上版本。	建议升级到最新版本。
P1	当表正在实时写入（通过Flink、数据集成等方式）时，同时查询数据报错：`ERROR: internal error: Record batch has 742 rows but length of columns is 749. columns=[ColumnHandle(type=string)(table_column_id=3), ColumnHandle(type=string)(table_column_id=4), ColumnHandle(type=string)(table_column_id=5)]`。	实时写入时，数据是先写入MemTable再落到磁盘，在写入期间去查询，查询列标记长度和真实数据长度未对齐，导致查询失败报错。	出现版本： 0.10.41版本。 修复版本： 0.10.42及以上版本。	建议升级到最新版本。
P1	业务没有增加，内存突然增长。	SQL中有如下函数，会出现内存泄漏，导致内存突然增长。 • extract(xxx from time) • extract(xxx from interval) • date_part(xx, interval)	出现版本： 0.10.31以下版本。 修复版本： 0.10.32及以上版本。	• 不使用列表中的函数。 • 建议升级到最新版本。
P2	提示报错：`time before epoch time not supported`。	SQL中使用了`to_char`、`to_date`和`to_timestamp`这些函数的其中一个或多个，且数据有1970年之前的数据，Hologres不支持1970年之前的数据。	出现版本： 0.10及以下版本。 修复版本： 1.1版本。	• 过滤1970年之前的数据。 • 建议升级到最新版本，可支持1925-2282年的数据。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	非Superuser执行 `select hg_dump_script('xxxx')` 函数时报错： `ERROR: permission denied for table pg_subscription` 。	`hg_dump_script` 间接调用了 `pg_subscription` 这个relation，但是 `pg_subscription` 里面可能会存在敏感信息，默认只有Superuser才可以访问这个表。	出现版本： 0.10版本。 修复版本： 1.1版本。	• `pg_subscription` 并未实际存储对于 `hg_dump_script` 有用的信息，已修改该默认行为，在V1.1版本解决。 • 遇到没有权限问题，可以给当前用户授予 `pg_subscription` 的访问权限
P2	SQL中含有 `left join` ，不带有 `limit` 子句查询结果仅一条，加上limit之后会出现多条重复的数据。	`left join` 在底层会转换成 `right outer join` ，引擎在实现 `right outer join` 的时候，生成的right side走broadcast的错误执行计划，导致最终结果数据重复，可以通过执行 `explain sql` 查看执行计划是否走broadcast。	出现版本： 0.10.40及以下版本。 修复版本： 1.1版本。	建议升级到最新版本。
P2	往Binlog表里一次写入多条重复的数据时，中间数据的Binlog会丢失，未保留所有Binlog中间状态变化。	重复数据会被引擎去重，默认保留最后一条，导致中间状态变化丢失。	出现版本： 0.10.30及以下版本。 修复版本： 0.10.39及以上版本。	建议升级到最新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	提示报错：`ERROR: status { code: SERVER_INTERNAL_ERROR message: " HGERR_code 00000 HGERR_msge OptimizedRegularExpression: cannot compile re2: \\c, error: invalid escape sequence: \\c4 HGERR_end[query_id:xx" err_data { message: "OptimizedRegularExpression: cannot compile re2: \\c, error: invalid escape sequence: \\c4" context: "[query_id:xxx]" } }CONTEXT: [query_id:xx]` 。	SQL中的like有\ + 字符或数字的情况，示例SQL如下。 `select * from test_tb where a like '%\c%';select * from test_tb where a like '%F\G%';` 目前引擎对于SQL中的like有\ + 字符或数字的情况处理不够完善，导致报错。	出现版本： 0.10.38及以下版本。 修复版本： 0.10.39及以上版本。	建议升级到最新版本。
P2	行存表根据主键查询时，结果不一致或者报错：`Duplicate keys detected when building hash table` 。	建行存表时，主键和Clustering Key的顺序指定不一致，如 `create table k ( a int, b int, primary key(a, b)); call set_table_property ('k', 'orientation', 'row'); call set_table_property ('k', 'clustering_key', 'b,a');`	出现版本： 0.10.37及以下版本。 修复版本： 0.10.38及以上版本。	• 重新建表，将主键和Clustering Key的顺序保持一致。 • 建议升级到新版本。
P2	在新建的schema下使用数据脱敏，查询脱敏数据时报错：`hg_anon_mask_name(text) doesnt exist` 。	数据脱敏函数被创建在public schema下，导致在新schema下无法查询脱敏数据。	出现版本： 0.10.35及以下版本。 修复版本： 0.10.36及以上版本。	• 只在public schema下使用数据脱敏函数。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P2	报错：`internal error:string decimal literal can not be tentative`。	SQL语句里`in`中包含不同精度的decimal数据，示例SQL如下。 `select * from table where sval in(170344.964,133907.84);`	出现版本： 0.10.34及以下版本。 修复版本： 0.10.35及以上版本。	• 修改SQL语句里`in`中包含的decimal数据精度一致。 • 建议升级到最新版本。

2021年07月

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P0	RoaringBitmap字段被配置为字典编码（Dictionary Encoding）时，造成写入失败，实例不可查询。	RoaringBitmap类型并不支持字典编码，强行设置造成编码逻辑故障，导致写入一直失败。	出现版本： 0.10.24及以下版本。 修复版本： 0.10.25及以上版本。	• RoaringBitmap字段取消Dictionary Encoding。 • 建议升级到新版本。
P0	在非`public schema`执行`add commect on tablename is "is comment"`导致写入或查询卡住。	在非`public schema`下执行`add comment`操作：`add comment on tablename is "comment"`，SQL语句中未指定Schema名，导致单个节点异常，从而导致出现写入/查询卡住现象。	出现版本： 0.10.20及以下版本。 修复版本： 0.10.21及以上版本。	• SQL中`add comment`时加上Schema名：`add comment on schema.tablename is "comment"`。 • 建议升级到新版本。
P0	报错：`cannot acquire lock in time`。	在原有的版本中，会对DDL加锁，高并发查询和删除（Drop）同一张表时，后端节点出现死锁，导致有关这张表的操作都卡住，从而报错	出现版本： 0.9.22及以下版本。 修复版本： 0.9.23及以上版本。	建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	在数据没有写入时，存储空间持续线性增长。	使用 `insert on conflict do update set pk =pk` 语句导入数据，实际上导入前和导入后数据并没有实际变化，触发底层优化BUG，导致存储线性增长。	出现版本： 0.10.23及以下版本。 修复版本： 0.10.24及以上版本。	• 可以通过执行 `insert into values` 语句，触发数据更新，多余的数据将会被删除。 • 建议升级到新版本。
P1	在执行 `extract XXX from timestamptz/timestamp` 时，提示报错： `time before epoch time not supported` 。	EXTRACT函数在处理数据中的NULL值会有误。	出现版本： 0.10.20及以下版本。 修复版本： 0.10.21及以上版本。	• SQL通过过滤条件过滤掉NULL。 • 建议升级到新版本。
P1	报错： `cant determine shard id of shard value` 。	SQL语句中有 `union all` 语句，并且Group by字段中有 `distrubition key` ，导致执行计划错误，从而导致找不到对应的*Shard*。	出现版本： 0.10.20及以下版本。 修复版本： 0.10.21及以上版本。	建议升级到新版本。
P1	`ERROR: Query could not generate plan by gporca : Group by key is type of unsupported type. not supported`	Group by的字段类型是非精确类型，导致出现报错。	出现版本： 0.9及以下版本。 修复版本： 0.10已开发限制。	• Group by中避免非精确数据类型。 • 建议升级到新版本。
P1	读外表时报错： `unsupported column type:list` 。	在MaxCompute已有表中新增一列 `array column` 且该列不导入数据，外表查询该MaxCompute时报错	出现版本： 0.9.22及以下版本。 修复版本： 0.9.23及以上版本。	• MaxCompute新增一列 `array column` 后，及时为该列写入数据。 • 建议升级到新版本。

等级	报错/问题描述	缺陷原因	出现/修复版本	规避建议
P1	报错 `ERROR: internal error: The left child should be column ref, num_children: 1` 。	查询的SQL中，Clustering key为*varchar*类型就会触发。	出现版本： 0.9.24及以下版本。 修复版本： 0.9.25及以上版本。	• 将*varchar*字段改成TEXT字段。 • 建议升级到新版本。
P2	查询外表报错： `code: SERVER_INTERNAL_ERROR message: "query next from foreign table executor failed，Unknown file type: xxx`	MaxCompute集群发生配置更新，同时Hologres依赖的外表元数据未及时更新导致。	出现版本： 0.10.20及以下版本。 修复版本： 0.10.21及以上版本。	无法规避，需要实例重启或者升级到新版本。

1.4. 文档修订记录

本文为您介绍Hologres文档更新的最新动态，基于此您可以及时了解Hologres的新增特性及功能变更。

2022年06月

时间	特性	类别	描述	产品文档
2022.06.21	新增支持时区设置。	更新说明	HoloWeb SQL窗口文档新增时区设置说明。	SQL窗口
2022.06.17	新增常见问题。	更新说明	新增创建外部表时报错： `You don't exist in project xxx` 的常见问题。	MaxCompute权限相关
2022.06.14	全文重构。	更新说明	Proxima向量计算内容结构进行重构，信息聚焦呈现。	Proxima向量计算

时间	特性	类别	描述	产品文档
2022.06.14	新增 `rb_or_null2empty`、`rb_or_null2empty_cardinality`、`rb_and_null2empty`、`rb_and_null2empty_cardinality`、`rb_andnot_null2empty`、`rb_andnot_null2empty_cardinality` 函数。	更新说明	新增Bitmap函数用于计算。	Roaring Bitmap函数

2022年05月

时间	特性	类别	描述	产品文档
2022.05.26	新增产品形态。	新说明	由于世界各国家与地区所处经度不同，地方时也不同，因此会划分为不同的时区。本文为您介绍在Hologres中时区的相关信息。	时区
2022.05.11	新增产品形态。	新说明	本文以一个示例为您介绍Apache Nifi如何连接Hologres。Apache Nifi是一个易用、可靠的数据处理与分发系统，Apache NiFi的设计目标是自动化管理系统间的数据流。	Apache Nifi
2022.05.11	新增产品形态。	新实践	本文为您介绍如何通过DataWorker数据集成和Flink两种方式将MySQL分库分表的数据写入至Hologres同一个表中，通过本实践您可以根据业务场景选择合适的方式将MySQL分库分表数据写入Hologres。	MySQL分库分表实践

时间	特性	类别	描述	产品文档
2022.05.11	新增产品形态。	新说明	本文为您介绍Hologres的产品生命周期策略及产品终止策略、产品版本号及含义、版本对应关系和产品生命周期重要事件点。	产品生命周期策略与版本

2022年04月

时间	特性	类别	描述	产品文档
2022.04.28	新增产品形态。	新说明	本文为您介绍Hologres中的锁及如何排查锁。锁是一种数据库的信号量管理机制，作用是保证SQL执行上不同事务的隔离性。	锁以及排查锁
2022.04.26	新增函数分类。	新说明	本文为您介绍Hologres已支持的时间和日期函数列表及使用用例。	时间和日期函数
2022.04.22	新增产品形态。	新说明	本文为您介绍Hologres中出现OOM情况的原因及对应处理方法。OOM（Out of Memory）描述的是Query的内存消耗超出了系统当前的供给，系统做出的一种异常提示。	OOM常见问题排查指南

2022年03月

时间	特性	类别	描述	产品文档
2022.03.25	新增产品形态。	新说明	Hologres中支持加载扩展（extension）以实现更丰富的功能，本文为您介绍Hologres中支持的extension扩展以及如何加载、查看、卸载extension。	Extension扩展

时间	特性	类别	描述	产品文档
2022.03.16	新增产品形态。	新说明	APPLY_PRIVILEGES函数用于在专家权限模型下，将源表给用户授予的权限复制到目标表，目标表保持与源表一致的权限。本文为您介绍在Hologres中APPLY_PRIVILEGES函数的用法。	APPLY_PRIVILEGES
2022.03.04	新增产品形态。	新说明	本文为您介绍不同的权限模型之间如何快速简单切换。Hologres提供三种权限模型：专家权限模型、简单权限模型（Simple Permission Model，SPM）和基于Schema的简单权限模型（Schema-Level Permission Model，SLPM），可以根据实际业务情况切换合适的权限模型。	权限模型转换
2022.03.04	新增产品形态。	新实践	本文将为您介绍Hologres的实时标签计算最佳实践。Hologres与Flink深度集成，支持高性能的数据写入与更新，数据写入即可查询。实时效果回流和实时标签生产都可以通过Hologres和Flink的组合方案来完成，应用于众多业务场景中。	画像分析 - 实时标签

时间	特性	类别	描述	产品文档
2022.03.04	新增产品形态。	新实践	本文将会介绍如何通过Hologres进行超大规模标签计算、画像分析。通过对标签表构建索引，将用户ID编码后以Bitmap格式保存，将关系运算转化Bitmap的交并差运算，进而加速实时计算性能。在超大规模用户属性洞察分析的场景中，使用RoaringBitmap组件能够实现亚秒级的查询响应。	画像分析 - RoaringBitmap优化方案
2022.03.04	新增产品形态。	新实践	本文为您介绍在Hologres中通过宽表实现标签计算的方案与使用示例。	画像分析 - 标签宽表
2022.03.04	新增产品形态。	新实践	本文为您介绍在Hologres中标签、画像分析场景的最佳实践的背景及解决方案。	用户画像分析概述
2022.03.03	新增产品形态。	新说明	本文为您介绍如何在数据地图中配置Hologres元数据采集器及相关操作。数据地图可以帮助您更好地查找、理解和使用Hologres数据。	数据地图（Beta）

2022年02月

时间	特性	类别	描述	产品文档
2022.02.24	新增产品形态。	新实践	实时UV计算主要依赖Hologres与Flink结合完成，本文将为您介绍Hologres如何进行实时UV精确去重。	实时UV精确去重（Flink+Hologres）
2022.02.24	新增产品形态。	新实践	本文为您介绍在Hologres中如何进行离线UV计算。	离线UV计算

时间	特性	类别	描述	产品文档
2022.02.24	新增产品形态。	新实践	在用户行为分析和圈人场景中，经常需要从亿级甚至几十亿级用户中快速筛选出符合特定标签的指标结果，本文为您介绍Hologres中如何进行用户行为分析。	用户行为分析（UV）概述
2022.02.24	新增产品形态。	新说明	Hologres作为和MaxCompute深度结合的产品，无需数据搬迁，即可通过外部表查询MaxCompute的数据。本文为您介绍如何使用MaxCompute外部表自动加载功能。	MaxCompute外部表自动加载
2022.02.21	新增产品形态。	新说明	本文为您介绍Hologres使用过程中自助健康检查常用命令。	自助健康检查常用命令
2022.02.18	新增产品形态。	新说明	本文为您介绍在Hologres中如何使用SQL实现INSERT OVERWRITE的功能。	INSERT OVERWRITE
2022.02.18	新增产品形态。	新说明	Fixed Plan是Hologres独有的执行引擎优化方式，本文将为您介绍可以被Fixed Plan选中的SQL需要符合的条件和参数配置。	Fixed Plan加速SQL执行
2022.02.14	新增产品形态。	新说明	本文为您介绍Holo Shipper工具，此工具是支持将Hologres实例的部分表导入导出的Hologres生态离线备份工具。实现在Hologres之间搬迁表和将表转储到中间存储然后再恢复的功能。	迁移工具Holo Shipper

时间	特性	类别	描述	产品文档
2022.02.11	新增产品形态。	新实践	本文为您介绍在Hologres中数仓分层的最佳实践，方便快速构建业务，建设集高性能、敏捷化于一体的实时数仓。	Hologres推荐的数仓分层

2022年01月

时间	特性	类别	描述	产品文档
2022.01.27	新增产品形态。	新说明	本文为您介绍Hologres的安全管理功能。	安全白皮书
2022.01.24	新增产品形态。	新说明	本文为您介绍DataGrip相关产品如何连接Hologres。	DataGrip
2022.01.20	新增产品形态。	新说明	本文为您介绍如何通过续费管理对包年包月实例进行续费操作。	续费管理
2022.01.19	新增产品形态。	新说明	本文为您介绍Hologres的执行引擎以及内部的各组件。	执行引擎
2022.01.19	新增产品形态。	新说明	本文为您介绍Hologres的产品架构以及每个组件的作用。	产品架构
2022.01.18	新增产品形态。	新说明	本文为您介绍Hologres中Table Group和Shard Count的概念。	基本概念
2022.01.11	新增最佳实践。	新说明	本文为您介绍如何迁移HBase的语法和数据至Hologres上进行数据开发。	迁移HBase至Hologres
2022.01.11	新增常见问题。	新说明	本文为您介绍使用Hologres过程中关于Blink和Flink的常见问题。	Blink和Flink常见问题及诊断

时间	特性	类别	描述	产品文档
2022.01.07	新增常见问题。	新说明	本文为您介绍使用Hologres监控指标相关的常见问题。	监控指标常见问题
2022.01.07	新增常见问题。	新说明	本文为您介绍使用Hologres对接MaxCompute过程中的常见问题，助您排查解决出现的问题。	对接MaxCompute常见问题与诊断

2021年12月

时间	特性	类别	描述	产品文档
2021.12.30	新增产品形态。	新说明	本文为您介绍Hologres中的系统表、表中的参数和每个表如何使用。	系统表
2021.12.29	新增产品形态。	新说明	本文为您介绍Hologres中GUC参数的含义以及使用方法，更好地支持您丰富的使用场景。	GUC参数
2021.12.20	新增使用示例。	新说明	本文为您介绍如何使用Hologres数据脱敏功能，新增使用示例帮助您使用。	数据脱敏（Beta）
2021.12.16	新增产品形态。	新说明	本文为您介绍Hologres已支持的类型转换函数列表及使用用例。	类型转换函数
2021.12.14	新增常见问题。	新说明	本文新增用户管理过程中的常见问题，助您以可视化方式更细致的管理实例内的用户。	用户管理
2021.12.03	新增产品形态。	新说明	本文为您介绍高可用方案的一些基本原理以及在Hologres中如何配置共享存储多实例。	多实例读写分离高可用部署（共享存储）

时间	特性	类别	描述	产品文档
2021.12.02	新增产品形态。	新说明	本文为您介绍Hologres实例的升级方式和升级所需要提供的信息。	实例升级

2021年11月

时间	特性	类别	描述	产品文档
2021.11.26	新增产品形态。	更新说明	本文为您介绍什么是实时数仓Hologres以及Hologres的功能。	什么是实时数仓Hologres
2021.11.19	新增Region。	更新说明	计费新增法兰克福Region的计费说明。	计费方式
2021.11.17	新增常见问题。	新说明	本文新增使用资源组管理Hologres实例内的计算资源，实现资源隔离过程中的常见问题。	单实例计算资源隔离（Beta）
2021.11.10	新增缺陷修复通知。	新说明	本文为您介绍Hologres10月各版本相关缺陷的修复记录。	关键缺陷通知
2021.11.10	新增最佳实践。	新实践	本文为您介绍如何迁移自建ClickHouse的数据库表和数据至交互式分析Hologres上进行数据开发。	迁移ClickHouse至Hologres

2021年10月

时间	特性	类别	描述	产品文档
2021.10.26	新增产品形态。	新实践	本文为您介绍如何将MaxCompute分区表数据通过DataWorks调度周期性导入Hologres分区表。	通过DataWorks周期性导入MaxCompute数据最佳实践
2021.10.26	新增产品形态。	新说明	本文为您介绍如何使用DataStudio导入MaxCompute数据至Hologres。	一键表数据同步

时间	特性	类别	描述	产品文档
2021.10.26	新增产品形态。	新说明	本文为您介绍如何使用DataStudio一键表结构同步功能批量创建Hologres外部表。	一键表结构导入
2021.10.20	新增产品形态。	新说明	本文为您介绍如何使用HoloWeb的执行计划和运行分析功能查看SQL语句的执行计划。	查看执行计划
2021.10.19	新增产品形态。	新说明	本文为您介绍如何使用HoloWeb新建资源组、删除资源组、修改资源组配额，以及在资源组中解除或绑定用户。	资源组管理
2021.10.09	新增产品形态。	新说明	本文为您介绍Hologres如何使用外部表查询MaxCompute BYOK加密数据。	查询MaxCompute加密数据（BYOK模式）
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres新增支持通过密钥管理服务KMS（Key Management Service）对数据进行加密存储，提供数据静态保护能力，满足企业监管和安全合规需求。	数据存储加密
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres新增支持在不同地域、不同实例和不同数据库之间，通过创建外部表的方式查询数据，操作方便简单。	跨库查询
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres新增支持将表中数据导出至OSS，快速实现Hologres数据统一存储，进一步降低存储成本。	Hologres数据导出至OSS

时间	特性	类别	描述	产品文档
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres新增JSONB索引，加速JSON类型数据的查询检索。	JSON和JSONB类型
2021.10.09	新增产品形态。	新功能	本文为您介绍，Hologres新增支持重命名列名称。	ALTER TABLE
2021.10.09	新增产品形态。	新功能	本文将为您介绍Hologre新增支持Hologres Binlog按需启用，配置动态修改。	订阅Hologres Binlog
2021.10.09	新增产品形态。	新功能	本文为您介绍如何通过JDBC和Holo-Client两种方式消费Hologres Binlog。	通过JDBC消费Hologres Binlog（Beta）
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres支持将表设计为行列共存结构，一份数据同时支持点查、OLAP多种查询场景。	CREATE TABLE
2021.10.09	新增产品形态。	新功能	本文为您介绍Hologres新增资源组隔离能力，通过设计多个资源组，实现实例内部不同用户的计算资源线程级负载隔离，可以更好地支撑多用户、多场景的使用方式。	单实例计算资源隔离（Beta）

2021年09月

时间	特性	类别	描述	产品文档
2021.09.07	新增产品形态。	新说明	本文为您介绍DataWorks数据集成通过Kafka服务的Java SDK从Kafka读取数据，再实时同步至Hologres。	Kafka通过DataWorks实时同步

2021年08月

时间	特性	类别	描述	产品文档
2021.08.31	新增最佳实践。	新实践	本文为您介绍Hologres中Table Group和Shard Count选定的基本原则、适用场景、设置策略、操作指南以及常见问题，以助您获取更优的查询性能。	Table Group设置最佳实践
2021.08.30	新增产品形态。	新说明	本文为您介绍通过DataWorks数据集成实时同步数据库中的数据至Hologres的操作流程，及操作示例。	实时同步数据库的数据至Hologres
2021.08.26	新增产品形态。	新说明	本文为您介绍如何将MaxCompute分区表数据导入到Hologres分区表，及提供DataWorks作业案例实现周期性调度。	MaxCompute分区表数据导入
2021.08.26	新增开发规范。	新说明	本文为您介绍，Hologres在开发过程中的相关规范，帮助您快速了解Hologres的开发要求，避免进行错误的操作。	Hologres开发规范
2021.08.20	新增产品形态。	新说明	本文将为您介绍Hologres和Clickhouse在SSB单表数据集上进行性能对比测试的结果。	Hologres vs Clickhouse性能对比参考测试
2021.08.18	新增用户案例页面。	新说明	本文为您介绍Hologres的用户案例合集内容。	用户案例合集
2021.08.13	新增快速入门。	新说明	本文以一个简单的示例，为您介绍使用Hologres查询外部表和内部表中数据的基本步骤，帮助您直观体验Hologres查询数据性能。	Hologres查询体验快速入门

时间	特性	类别	描述	产品文档
2021.08.06	新增最佳实践。	新实践	本文为您介绍MySQL数据平滑迁移至Hologres的操作方法，以及迁移完成后MySQL与Hologres查询语句与函数的使用区别，方便您更加快速的完成数据迁移。	迁移MySQL至Hologres

2021年07月

时间	特性	类别	描述	产品文档
2021.07.21	新增产品形态	体验优化	本文为您介绍如何通过Spark读取或写入数据至Hologres的操作方法。	Spark的数据写入至Hologres
2021.07.21	新增产品形态	新增功能	Hologres从V0.10开始支持慢Query的查询与分析，帮助您对系统中发生的慢Query或失败Query进行诊断、分析和采取优化措施。	历史慢Query
2021.07.15	新增产品形态	体验优化	为您介绍主账号如何授权RAM用户，从而实现RAM用户连接并使用Hologres。	RAM用户权限授权快速入门
2021.07.14	新增产品形态	新增功能	Hologres各版本相关缺陷的修复记录，包括问题描述、影响程度等。	关键缺陷通知
2021.07.04	新增产品形态	体验优化	为您介绍测试的参考结果。	测试结果参考

2021年06月

时间	特性	类别	描述	产品文档
2021.06.29	新增产品形态	新增功能	通过阿里云操作审计ActionTrail，查询90天内的实例操作事件日志。	查询事件日志

时间	特性	类别	描述	产品文档
2021.06.25	新增最佳实践	体验优化	为您介绍在Hologres中使用空间函数的操作方法。	使用空间函数查询数据方法
2021.06.24	新增产品形态	新增功能	为您介绍PostGIS的使用方法以及PostGIS空间函数说明。	PostGIS (Beta)
2021.06.04	新增产品形态	新增功能	为您介绍HoloWeb支持的视图功能。	视图
2021.06.02	新增产品形态	新增功能	为您介绍如何参考TPC-H测试进行OLAP查询场景和Key/Value点查场景进行性能测试。	测试方案介绍

2021年05月

时间	特性	类别	描述	产品文档
2021.05.21	新增产品形态	新增功能	HoloWeb管控台界面变更，文档内容刷新。	登录实例
2021.05.17	新增产品形态	新增功能	Hologres V0.10版本新增内容发布。	Hologres 2021年5月新增功能
2021.05.08	新增产品形态	新增功能	为您介绍Grafana和Yonghong BI工具如何连接Hologres并可视化分析数据。	• Grafana • Yonghong BI

2021年04月

时间	特性	类别	描述	产品文档
2021.04.28	新增产品形态	新增功能	为您介绍Davinci如何连接Hologres并可视化分析数据。	Davinci
2021.04.21	新增产品形态	新增功能	为您介绍SAP BusinessObjects如何连接Hologres并可视化分析数据。	SAP BusinessObjects

时间	特性	类别	描述	产品文档
2021.04.19	新增产品形态	新增功能	为您介绍IBM Cognos Analytics如何连接Hologres并可视化分析数据。	IBM Cognos Analytics

2021年03月

时间	特性	类别	描述	产品文档
2021.03.25	新增产品形态	新增功能	为您介绍Metabase和Dataiku如何连接Hologres并可视化分析数据。	• Metabase • Dataiku
2021.03.18	新增产品形态	新增功能	为您介绍如何对实例中的连接和Query进行诊断和管理。	连接数管理
2021.03.17	新增产品形态	新增功能	为您介绍如何使用Python访问Hologres。	使用Python访问Hologres
2021.03.17	新增产品形态	新增功能	为您介绍Redash如何连接Hologres并可视化分析数据。	Redash
2021.03.12	新增产品形态	新增功能	为您介绍Apache Zeppelin如何连接Hologres并可视化分析数据。	Apache Zeppelin

2021年02月

时间	特性	类别	描述	产品文档
2021.02.09	新增产品形态	新增功能	为您介绍Hologres中实例的资源规格，您可以根据需要动态调整实例的规格。	实例规格概述
2021.02.09	新增产品形态	新增功能	为您介绍在Hologres中有序聚集函数的用法。	有序聚集函数

时间	特性	类别	描述	产品文档
2021.02.09	新增最佳实践	性能调优	为您介绍在Hologres中，Clustering key的概念以及实现快速查询的设置方法。	Hologres Clustering Key最佳实践

2021年01月

时间	特性	类别	描述	产品文档
2021.01.22	新增产品形态	新增功能	Hologres V0.9版本新增内容发布。	Hologres 2021年1月新增功能
2021.01.20	新增产品形态	新增功能	为您介绍Apache Superset如何连接Hologres并可视化分析数据。	Apache Superset
2021.01.12	新增产品形态	新增功能	为您介绍在Hologres中基于Schema级别的简单权限模型使用。	基于Schema级别的简单权限模型概述
2021.01.07	新增最佳实践	体验优化	为您介绍将原MaxCompute Lightning服务迁移至Hologres共享集群（MaxCompute BI加速版）的相关操作流程。	迁移Lightning至共享集群概述

2020年11月

时间	特性	类别	描述	产品文档
2020.11.27	新增产品形态	新增功能	为您介绍共享集群的相关信息，以及如何购买并使用交互式分析Hologres共享集群（MaxCompute BI加速版）。	概述

2020年10月

时间	特性	类别	描述	产品文档

时间	特性	类别	描述	产品文档
2020.10.26	新增SQL语法	新增功能	为您介绍Hologres中TRUNCATE的语法。	TRUNCATE
2020.10.26	新增数据类型	新增功能	为您介绍Hologres在建表时如何将字段的类型定义为Serial和Bigserial来定义自增的字段。	自增序列 Serial（Beta）
2020.10.26	新增扩展函数	新增功能	为您介绍在Hologres中如何使用向量计算功能。	Proxima向量计算
2020.10.15	新增数据脱敏	新增功能	为您介绍Hologres如何开启、查询及删除数据脱敏功能。	数据脱敏（Beta）
2020.10.14	新增云监控监测	新增功能	为您介绍如何通过云监控监测Hologres实例的相关指标并上报告警。	云监控

2020年09月

时间	特性	类别	描述	产品文档
2020.09.28	新增SQL语法	新增功能	为您介绍Hologres中视图的用法。	VIEW
2020.09.17	新增最佳实践	体验优化	为您介绍在Hologres中，如何基于PostgreSQL标准模型（专家权限模型）进行授权。帮助您简化授权操作并使用更细粒度的权限管理。	基于PostgreSQL标准权限模型授权
2020.09.16	新增数据接入方式的Demo示例	新增功能	为您演示开源Flink如何实时写入数据至Hologres。	开源Flink 1.10实时导入数据至Hologres
2020.09.10	新增最佳实践	体验优化	为您介绍在Hologres中对内部表性能进行调优的最佳实践。	优化内部表的性能

2020年08月

时间	特性	类别	描述	产品文档
2020.08.14	新增HoloWeb功能模块	新增功能	为您介绍HoloWeb支持的管理云账号、一键上传本地文件及查看活跃Query等功能。	• 一键上传本地文件 • 查看活跃Query

2020年07月

时间	特性	类别	描述	产品文档
2020.07.31	新增Hologres产品解读视频	新增视频	为您介绍Hologres的功能及优势。	Hologres产品解读
2020.07.29	新增RAM鉴权功能上线公告	新增公告	为您介绍RAM鉴权功能上线的相关事宜。	RAM鉴权上线公告
2020.07.29	新增HoloStudio开发功能	新增功能	为您介绍如何使用HoloStudio新建文件夹，管理数据库中的开发节点。	文件夹
2020.07.23	新增子账号使用Hologres的功能模块	新增功能	为您介绍主账号如何创建子账号，并授权子账号使用Hologres。	• 授予RAM用户权限 • 授予RAM用户实例的开发权限 • RAM用户权限授权快速入门
2020.07.21	新增SQL语法	新增功能	为您介绍DROP SCHEMA的用法及使用限制。	DROP SCHEMA
2020.07.17	新增HoloWeb功能模块	新增功能	为您介绍HoloWeb支持的数据连接、数据库、模式、表、外部表、文件夹及SQL窗口模块的功能。	• 新增实例 • 数据库 • 模式 • 表 • 外部表 • 文件夹 • SQL窗口

2020年06月

时间	特性	类别	描述	产品文档

时间	特性	类别	描述	产品文档
2020.06.24	新增最佳实践	体验优化	为您介绍在Hologres中如何优化查询MaxCompute外部表数据的性能。	优化MaxCompute外部表的查询性能
2020.06.16	新增最佳实践	体验优化	为您介绍如何使用交互式分析Hologres对接实时计算，快速搭建实时数仓分析大屏。	快速搭建实时数仓分析大屏
2020.06.16	新增最佳实践	体验优化	为您介绍交互式分析Hologres如何实时查询海量MaxCompute数据，并以可视化方式分析和展现查询结果。	实时分析海量MaxCompute数据
2020.06.08	新增HoloWeb功能模块	新增功能	为您介绍使用HoloWeb的基本流程。	连接HoloWeb

2020年04月

时间	特性	类别	描述	产品文档
2020.04.22	新增Holostudio开发功能	新增功能	为您介绍HoloStudio如何使用可视化的方式批量创建外部表。	一键同步MaxCompute表结构
2020.04.22	新增Holostudio开发功能	新增功能	为您介绍HoloStudio如何使用可视化的方式批量导入MaxCompute表数据至Hologres。	一键同步MaxCompute数据
2020.04.22	新增Holostudio开发功能	新增功能	为您介绍HoloStudio如何使用可视化的方式上传本地文件至Hologres。	一键上传本地文件
2020.04.07	新增Holostudio开发功能	新增功能	为您介绍如何绑定Hologres实例至DataWorks工作空间。	绑定Hologres实例

时间	特性	类别	描述	产品文档
2020.04.07	新增数据接入方式	新增功能	Hologres与DataWorks数据集成模块深度融合。您可以使用DataWorks数据集成导入或导出多种异构数据源至Hologres。	• Hologres Writer • Hologres Reader • 数据库中的数据离线同步至Hologres
2020.04.02	新增数据源	新增功能	DataWorks新增Hologres数据源，您可以使用该数据源配置同步任务。	配置Hologres数据源

2020年03月

时间	特性	类别	描述	产品文档
2020.03.30	新增数据接入方式	新增功能	为您介绍如何通过调用实时数据API实时写入数据至Hologres分区表的父表。	Hologres结果表
2020.03.26	新增数据接入方式	新增功能	为您介绍如何使用COPY命令导入客户端数据至Hologres。	使用COPY命令导入或导出本地数据
2020.03.26	新增数据接入方式	新增功能	为您介绍如何使用SQL命令将Hologres中的数据导出至指定的OSS。	使用COPY命令导出Hologres的数据至OSS
2020.03.24	新增函数	新增说明	新增Public API扩展函数。	概览
2020.03.20	新增管理控制台	新增功能	Hologres管理控制台支持实例管控、资源监控和用户管理等多元化功能。	• Hologres管理控制台 • 概览
2020.03.20	新增授权模型	新增说明	为您介绍简单权限模型的相关内容。	简单权限模型概述
2020.03.20	新增API接口	新增说明	Hologres的实时数据API支持实时写入多种数据源。	实时数据API
2020.03.20	新增SQL语法	新增功能	新增IMPORT FOREIGN SCHEMA语句的用法。	IMPORT FOREIGN SCHEMA

时间	特性	类别	描述	产品文档
2020.03.20	新增SQL语法	新增功能	新增UPDATE语句的用法。	UPDATE
2020.03.20	新增SQL语法	新增功能	新增DELETE语句的用法。	DELETE
2020.03.23	新增最佳实践	体验优化	为您介绍如何迁移Lightning服务至交互式分析Hologres进行数据开发。	迁移Lightning至Hologres

1.5. 公告

1.5.1. Hologres实例管理能力升级：支持接入阿里云资源组

本文为您介绍Hologres上线阿里云资源组功能的相关事宜。

尊敬的Hologres用户：

为了方便您更细粒度的在Hologres管理控制台管控实例，Hologres目前已支持阿里云资源组功能。该功能将于2021年12月06日22:00正式上线，该功能可以帮助您更加细致的管理Hologres实例。

使用阿里云资源组管理Hologres实例的最佳实践，请参见使用阿里云资源组管理实例。

1.5.2. RAM鉴权上线公告

本文为您介绍RAM鉴权功能上线的相关事宜。

尊敬的Hologres用户：

为了方便您更细粒度的在Hologres管理控制台管控实例，Hologres目前已支持RAM鉴权功能。该功能将于**2020年8月18日14：00**正式上线，在此之前，建议您使用主账号授予子账号相应的RAM权限，子账号被授权后，才能在权限范围内管理和操作Hologres管理控制台的实例。

RAM授权操作步骤如下：

1. **主账号进入RAM界面**

   主账号登录阿里云官网，并进入访问控制页面。

2. **选择子账号授权**

   单击左侧导航栏**用户**，在界面上选择需要授权的子账号，并单击**添加权限**，前往授权页面给子账号授权。

3. **新增授权**

   在策略中为子账号授予以下权限，子账号就能像主账号一样拥有所有操作权限。

权限策略	描述
AliyunHologresFullAccess	管理Hologres服务的权限。 设置该权限后，子账号可以查看管理控制台所有实例的信息，以及购买实例。 **说明** 您需要设置AliyunRAMReadOnlyAccess权限后，才可以在管理控制台的**用户管理**查看用户信息。
AliyunBSSOrderAccess	在费用中心（BSS）查看、支付以及取消订单的权限。 设置该权限后，子账号可以在管理控制台升级或降级实例的配置，以及为实例续费。
AliyunRAMReadOnlyAccess	只读访问控制（RAM）的权限。 设置该权限后，子账号可以在管理控制台的**用户管理**查看当前实例的用户、组以及授权信息。
AliyunHologresReadOnlyAccess	只读管理Hologres的权限。 设置该权限后，子账号可以查看Hologres管理控制台所有实例的信息，但是无法操作实例，例如修改网络类型。

   说明
   - 子账号购买的实例，主账号和子账号默认均为实例的Superuser。
   - 主账号购买的实例，子账号需要被主账号授权后，才能使用实例。

4. 更细粒度的RAM授权操作请参见授予RAM用户权限。

本次上线的RAM鉴权功能仅影响子账号在Hologres管理控制台的权限，对现有作业的正常运行以及子账号正常使用Hologres无影响。如果您希望方便使用子账号查看实例信息，建议您为子账号授予相关的RAM权限。

2.实例管理

2.1. 访问域名

交互式分析Hologres为您提供单独的域名访问地址Endpoint，您可以通过该地址，访问阿里云不同区域的Hologres服务。

开服域名

Hologres在公共云中，不同地域及网络环境的服务连接对照表如下。

公网网络已开服地域和服务连接对照表

地域	服务所在城市	公共网络Endpoint
华东1	杭州	`<instanceID>-cn-hangzhou.hologres.aliyuncs.com:80`
华北2	北京	`<instanceID>-cn-beijing.hologres.aliyuncs.com:80`
华东2	上海	`<instanceID>-cn-shanghai.hologres.aliyuncs.com:80`
华南1	深圳	`<instanceID>-cn-shenzhen.hologres.aliyuncs.com:80`
华北3	张家口	`<instanceID>-cn-zhangjiakou.hologres.aliyuncs.com:80`

经典网络已开服地域和服务连接对照表

地域	服务所在城市	经典网络Endpoint
华东1	杭州	`<instanceID>-cn-hangzhou-internal.hologres.aliyuncs.com:80`
华北2	北京	`<instanceID>-cn-beijing-internal.hologres.aliyuncs.com:80`
华东2	上海	`<instanceID>-cn-shanghai-internal.hologres.aliyuncs.com:80`

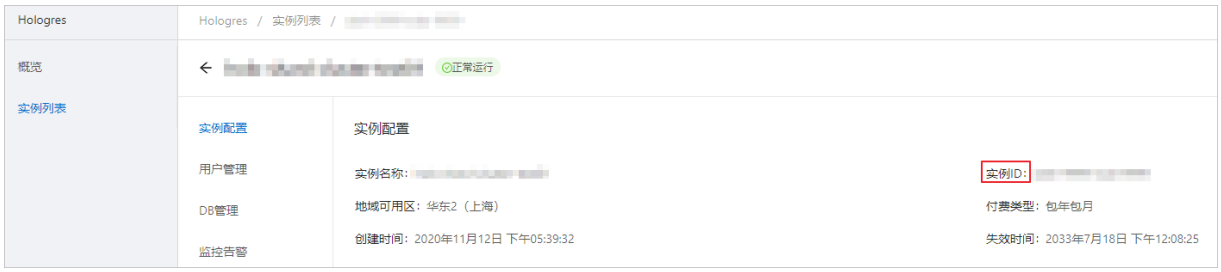
地域	服务所在城市	经典网络Endpoint
华南1	深圳	<instanceID>-cn-shenzhen-internal.hologres.aliyuncs.com:80
华北3	张家口	<instanceID>-cn-zhangjiakou-internal.hologres.aliyuncs.com:80

VPC网络已开服地域和服务连接对照表

地域	服务所在城市	VPC网络Endpoint
华东1	杭州	<instanceID>-cn-hangzhou-vpc.hologres.aliyuncs.com:80
华北2	北京	<instanceID>-cn-beijing-vpc.hologres.aliyuncs.com:80
华东2	上海	<instanceID>-cn-shanghai-vpc.hologres.aliyuncs.com:80
华南1	深圳	<instanceID>-cn-shenzhen-vpc.hologres.aliyuncs.com:80
华北3	张家口	<instanceID>-cn-zhangjiakou-vpc.hologres.aliyuncs.com:80

域名示例

<instanceID>为实例ID，您可以登录Hologres管理控制台，在实例详情页查看，如下图所示。



例如，您在北京购买一个名为*test demo*的Hologres实例。该实例在管理控制台解析出的实例ID为 `hgprecn-cn-xxxk3ovx003` ，则在不同网络下实例的域名如下：

- 公共网络为 `hgprecn-cn-xxxk3ovx003-cn-beijing.hologres.aliyuncs.com:80` 。
- 经典网络为 `hgprecn-cn-xxxk3ovx003-cn-beijing-internal.hologres.aliyuncs.com:80` 。

- VPC网络为 `hgprecn-cn-xxxk3ovx003-cn-beijing-vpc.hologres.aliyuncs.com:80` 。

2.2. 实例规格概述

Hologres不同的实例规格定义了不同的Core和内存资源，由于计算和存储分离架构，存储资源与实例规格不相关。本文将为您介绍实例的资源规格，您可以根据需要动态调整实例的规格，包括升配、降配，独立修改计算和存储资源。

基本概念

Hologres运行时的资源包括用于元数据管理的进程资源、用于查询服务的计算资源、用于优化数据写入的导入链路资源以及缓存服务。所有服务基于云原生容器技术，通过多个并行的容器计算节点实现高性能并行计算能力。

Hologres基于实例的资源规格提供默认的最大连接数和预分配的Shard数，这些参数是针对大多数场景，经过调校和优化的默认配置。其中，**最大连接数不可修改，Shard数可通过创建新的Table Group调整。系统扩容或者缩容时，最大连接数同时调整，但扩缩容之前的DB默认Shard数不调整，需要手动修改，新建的DB其Shard数为对应规格的默认值。**

在扩容后，更多的Core资源可以提供更好的查询并发能力，大多数使用场景不需要调整Shard数。当您需要更大的写入能力时，可以扩大Shard数，提高并发写入的吞吐量。同时，行存表由于天然的分布特性，Shard的个数越多，其读取性能会更高。

实例规格推荐

在使用Hologres实例实践过程中存在数据量可预估，最适宜实例规格以及对应Shard数区间应该设置为多少的问题，由于最适宜实例规格和Shard数不仅和数据存储量有关，还和实际访问频率、实际数据访问量、计算负载的类型（点查、分析等）、写入吞吐、Table Group上表的个数等因素有关，该问题无法给出准确答案。您可参见下表中根据数据量估算的所需Shard数和实例规格的推荐数，选择适合您的参数配置。

> **说明** 下表根据数据量估算的所需Shard数和实例规格的推荐数不是唯一标准，小数据量的表也可以放在多的Shard Count之上，大数据量的表也可以放在单个Shard上。请您根据实际业务场景选择一个合适的Shard Count，既满足有较高的并发度，带来更高计算效率，又满足数据较集中，从而避免不必要的Shuffle开销。

数据总规模	推荐规格	推荐Shard数	使用说明
4000万行以下	32Core以上	10 ~ 20	不适合压力测试，建议用于开发环境。
4000万行 ~ 4亿行	64Core以上	20 ~ 40	适合业务场景较为单一，没有混合负载场景。
4亿行 ~ 40亿行	128Core以上	40 ~ 80	写入查询能力较为均衡，建议生产系统默认起步配置。
40亿行 ~ 400亿行	256Core以上	80 ~ 240	建议考虑两个Table Group，大表和小表分别属于不同的Table Group，设计不同的Shard。
400亿行 ~ 4000亿行	512Core以上	160 ~ 400	建议考虑多个Table Group，仅对部分超大表划分较多Shard，普通表不建议Shard过多。

实例默认资源表

Hologres基于实例的资源规格提供默认的最大连接数和预分配的Shard数，默认规格配置如下表所示。

> 说明
>
> - 每个实例规格包括了计算节点和Frontend接入节点，在512Core及以下规格中，默认计算节点数与Frontend节点数相同，在更大规格中，Frontend节点数会略少于计算节点数。
> - 在规格扩容小于5倍时，不建议调整Shard。该默认规格适合绝大部分场景，考虑了写入和查询的平衡配置。
> - 单个节点的内存上限是64GB，内存会分为三部分，分别为计算、缓存和元数据。在Hologres V1.1.24版本之前，单个节点计算运行时内存上限是20GB，V1.1.24版本之后，计算内存采用运行时弹性分配，当前节点剩余未使用内存都会尽量用于计算时分配，提高整体内存利用率。
> - 最大总连接数=单Frontend节点最大连接数*Frontend节点数，括号中为具体每个节点的规格，其中括号前部分为单接入节点最大连接数，后部分为总Frontend接入节点个数。

实例规格	默认计算节点数	原默认Shard数（适用于V0.10.30及以下版本）	新默认Shard数（适用于V0.10.31及以上版本）	原最大总连接数（适用于V0.10.24及以下版本）	新最大总连接数（适用于V0.10.25及以上版本）	Superuser预留总连接数（适用于V0.10及以下版本）	Superuser预留总连接数（适用于V1.1及以上版本）
32Core	2	20	**20**	128（64*2）	**256（128*2）**	6	10（5*2）
64Core	4	40	**40**	256（128*2）	**512（128*4）**	6	20（5*4）
96Core	6	60	**60**	384（192*2）	**768（128*6）**	6	30（5*6）
128Core	8	80	**80**	513（171*3）	**1024（128*8）**	9	40（5*8）
160Core	10	100	**80**	640（160*4）	**1280（128*10）**	12	50（5*10）
192Core	12	120	**80**	770（154*5）	**1536（128*12）**	15	60（5*12）
256Core	16	160	**120**	1026（171*6）	**2048（128*16）**	18	80（5*16）
384Core	24	-	**160**	-	**3072（128*24）**	-	120（5*24）
512Core	32	320	**160**	2052（171*12）	**4096（128*32）**	36	160（5*32）

新增实例规格

自2022年4月25日起，Hologres提供了512CU至1024CU之间的计算资源规格，如需更高规格，请您提交工单。在升级更大资源规格前，请先将实例升级至V1.1.58及以上版本。默认规格配置如下表所示。

实例规格	默认计算节点数	默认Shard数（适用于V1.1.58及以上版本）	最大总连接数（适用于V1.1.58及以上版本）	Superuser预留总连接数（适用于V1.1.58及以上版本）
640Core	40	160	5120（128*40）	200（5*40）
768Core	48	160	6144（128*48）	240（5*48）
896Core	56	160	7168（128*56）	280（5*56）
1024Core	64	200	8192（128*64）	320（5*64）
1280Core	80	200	10240（128*80）	400（5*80）
1536Core	96	200	12288（128*96）	480（5*96）
1792Core	112	200	14336（128*112）	560（5*112）
2048Core	128	200	16384（128*128）	640（5*128）
2304Core	144	240	18432（128*144）	720（5*144）
2560Core	160	240	20480（128*160）	800（5*160）
3072Core	192	240	24576（128*192）	960（5*192）
3584Core	224	240	28672（128*224）	1120（5*224）
4096Core	256	320	32768（128*256）	1280（5*256）
4608Core	288	320	36864（128*288）	1440（5*288）
5120Core	320	320	40960（128*320）	1600（5*320）
5632Core	352	320	45056（128*352）	1760（5*352）
6144Core	384	320	49152（128*384）	1920（5*384）
6656Core	416	320	53248（128*416）	2080（5*416）
7168Core	448	320	57344（128*448）	2240（5*448）
7680Core	480	320	61440（128*480）	2400（5*480）
8192Core	512	400	65536（128*512）	2560（5*512）

查看并管理实例默认连接数

Hologres支持您查看并管理实例默认连接数。

- 查看连接数。

  当您创建实例并连接开发工具之后，可以执行如下语句进行查看，其中返回值是单个Frontend接入节点的最大连接数。

> **说明** Hologres实例总的最大连接数=单Frontend节点最大连接数*Frontend节点数。

```
--查看单接入节点的最大连接数（实际连接在多个接入节点间均衡分配）。
show max_connections;
```

- 管理连接。

  实例会为Superuser提供预留连接数，当连接数达到默认规格上限时，Superuser可以连接Hologres使用SQL命令查看空闲连接并进行释放，或者根据业务情况升配。查看空闲连接并进行释放连接的具体操作，请参见连接数。

查看并修改实例Shard数

在实例扩容后，大多数情况下不需要调整Shard数，更多的Core资源可以提供更好的查询并发能力。如果您需要更大的写入能力，可以通过扩大Shard数来提高并发写入的吞吐量。

同时，行存表由于天然的分布特性，更多的Shard读取性能会更高。如果因业务需求，需要查看以及修改实例的Shard数，请参见Table Group与Shard Count操作指南。

2.3. Hologres管理控制台

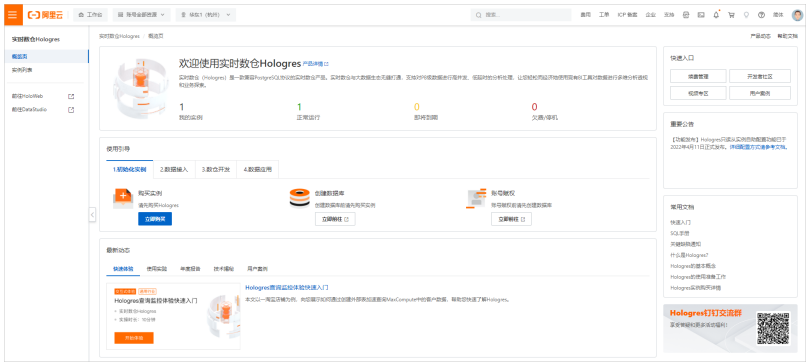
2.3.1. 概览

本文为您介绍Hologres管理控制台概览页面的功能。

交互式分析Hologres是实时交互式分析产品，全面兼容PostgreSQL协议，与大数据生态无缝打通，支持对万亿级数据进行高并发和低延时地多维分析，帮助您轻松使用BI工具分析数据，探索业务。

为满足您对Hologres实例深层次的管理需求，Hologres特推出独立于DataWorks管理控制台的Hologres管理控制台。

您可以使用阿里云账号登录Hologres管理控制台，在**概览页**新增Hologres实例并查看产品资讯。



Hologres管理控制台**概览**页面功能如下：

- **使用引导**

  您可以单击**概览页**的**立即购买**，跳转至Hologres实例的购买页面购买实例。当前实例的收费模式分为**按量付费**和**包年包月**，详情请参见购买Hologres。

- **最新动态**

  您可以在最新动态模块获取关于Hologres产品的各类最新资讯，包括快速体验、使用实践、年度报告、技术揭秘和用户案例。

- **快速入口**

  您可以单击**概览页**右侧的**快速入口**，快速进入对应的功能页面。

- **重要公告**

  您可以单击**概览页**右侧的**重要公告**，获取Hologres产品最新的公告。

- **常用文档**

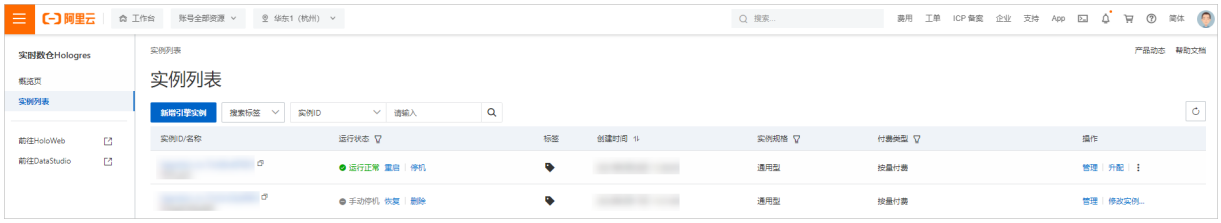
  您可以单击**概览页**右侧的**常用文档**，快速进入对应的产品文档页面，获取文档指导。

2.3.2. 实例列表

Hologres管理控制台的实例列表，罗列了您购买的所有实例。本文为您介绍了实例列表的功能及相关操作。

功能介绍

您可以在**实例列表**页面，查看当前账号购买的所有Hologres实例及实例状态、创建实例或更改实例配置。您也可以单击实例名称，进入实例详情页，更细致的管理控制数据库及用户等实例对象。



实例列表的功能具体如下：

- **新增引擎实例**

  您可以单击**实例列表**页面顶部菜单栏的**新增引擎实例**，跳转至实例的购买页面购买实例。详情请参见购买Hologres。

- **查看实例详情**

  您可以单击**实例列表**中的目标实例名称，跳转至实例的详情页查看实例配置、监控告警信息、进行用户和DB管理操作。详情请参见实例配置。

- **前往HoloWeb**

  HoloWeb是基于Hologres引擎开发的一站式大数据开发平台。

  HoloWeb支持使用可视化和SQL方式进行数据开发。您可以单击**前往HoloWeb**，即可进入HoloWeb的开发界面。关于HoloWeb的操作指导，详情请参见登录实例。

- **前往DataStudio**

  DataStudio是基于Hologres引擎的一站式开发平台，深度集成于DataWorks。

  HoloStudio支持使用可视化和SQL方式进行数据开发。您可以单击**前往DataWorks数仓开发**，即可进入HoloStudio的开发界面。进入HoloStudio开发数据之前，您需要绑定实例至DataWorks工作空间，详情请参见绑定Hologres实例。

- **搜索实例名称**

  如果您在当前地域开通的实例较多，则可以在**实例列表**页面顶部菜单栏的搜索框中，输入实例名称的关键词，即可检索出对应实例。

- **运行状态**

  运行状态列可以显示对应Hologres实例的当前运行状态，类别如下：

- **正常运行**：表示该实例当前运行正常 。
- **创建中**：表示实例刚购买成功，正在创建，请静等3~5分钟。
- **处理中**：表示实例正在进行升配、降配、重启和恢复实例等操作，请静等2~5分钟。
- **已停机**：表示当前实例已经暂停，无法连接访问。

- **付费类型**

  付费类型列表示对应实例所采用的付费模式。当前Hologres实例分为**包年包月**和**按量付费**两种付费模式，详情请参见计费方式。

- **操作**

  在实例列表的**操作**列，您可以对实例进行如下操作：

  - **管理**

    您可以单击**管理**，进入实例的管理页面，更细致化的查看和管理当前实例的数据库及用户等对象。

  - **修改实例名称**

    您可以单击**修改实例名称**，进入**修改实例名称**对话框，输入新的实例名称，单击**确定**完成实例的更名操作。

  - **转包年包月**

    您可以单击**转包年包月**，转换**按量付费**为**包年包月**计费方式。

    > **说明** Hologres目前不支持转换**包年包月**为**按量付费**模式。详情请参见转换计费方式。

  - **升配**

    如果您当前的实例规格无法满足现有业务，则可以单击**升配**扩容实例规格。Hologres仅支持主账号以及被主账号授权的子账号执行升配操作。

    > **说明** 实例升配期间，将会停止服务（一般为几分钟），请选择对业务影响最小的时间进行升配。

  - **降配**

    如果您现有的实例规格存在大量闲置资源，则可以单击**降配**缩容实例规格。Hologres仅支持主账号以及被主账号授权的子账号执行降配操作。

    > **说明** 实例降配期间，将会停止服务（一般为几分钟），请选择对业务影响最小的时间进行降配。

  - **续费**

    如果您购买的实例为**包年包月**模式，则可以单击**续费**，延长当前实例的使用时间。

  - **停机**

    如果您购买的实例为**按量付费**模式，则可以单击**停机**，暂停使用当前实例。单击**停机**后，实例的计算将会停止收费，但存储仍然会继续收费。

  - **删除**

    如果您不需要再使用当前实例，则可以单击**删除**，删除当前实例。

- **重启**

  如果您需要重启当前实例，则可以单击**重启**，重启当前实例。

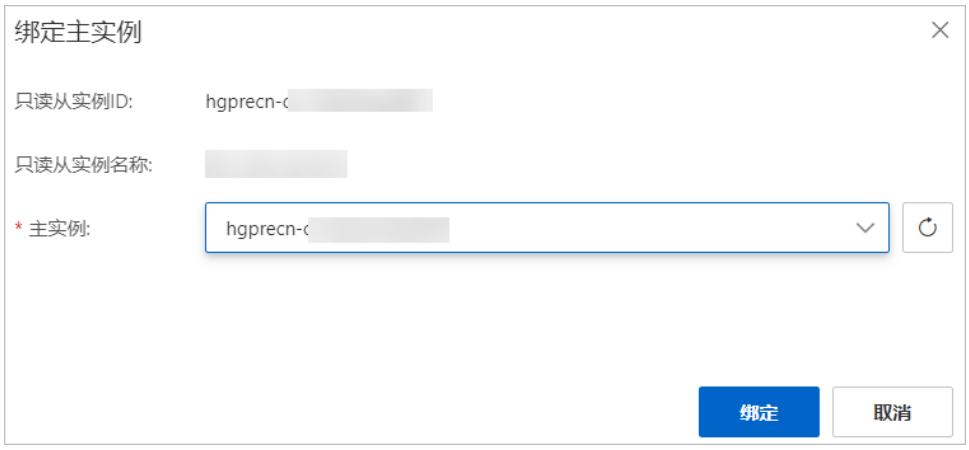
只读从实例绑定和解绑

Hologres 在V1.1版本，支持采用共享存储的多实例部署方案。在该方案中，主实例具备完整能力，数据可读可写，权限、系统参数可配置，而只读从实例处于只读状态，所有的变更都通过主实例完成。详细的产品和架构介绍请参见多实例读写分离高可用部署（共享存储）。

- 使用限制
  - 仅Hologres V1.1及以上版本支持作为主实例，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
  - 只读从实例未绑定主实例时，您无法连接只读从实例。
  - 主实例的版本和只读从实例版本必须一致。
  - 主实例和只读从实例必须处于同一个Region。
- 绑定和解绑操作的权限说明

  您需要为RAM用户授予AliyunHologresFullAccess权限策略，方可进行绑定和解绑只读从实例操作。更多关于RAM角色权限的介绍请参见授予RAM用户权限。
- 绑定只读从实例
  i. 在Hologres管理控制台的**实例列表**页面，单击需要绑定只读从实例**实例类型**列的**绑定**。

  ii. 在**绑定主实例**弹窗选择要绑定至的主实例，单击**绑定**完成绑定只读从实例。



- 解绑只读从实例

  对于已经绑定主实例的只读从实例可以与主实例解绑。

  i. 在Hologres管理控制台的**实例列表**页面，单击需要绑定只读从实例**实例类型**列的**解绑**。

ii. 在**解绑提示**弹窗，单击**解绑**完成解绑只读从实例。

解绑后，您无法连接只读从实例。

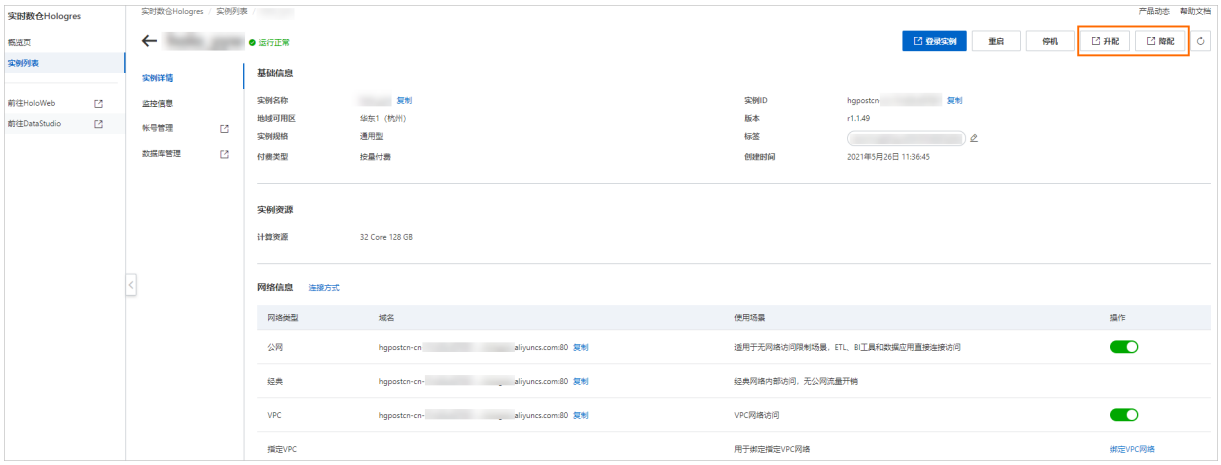
2.3.3. 实例配置

本文为您介绍在交互式分析Hologres管理控制台中，所购买实例的实例配置、网络配置及连接方式等信息。

实例配置

实例配置为您提供当前实例的基本信息，包括实例的名称、ID、地域、版本、付费类型、规格以及开通时间等。

您可以单击**升配**或**降配**，跳转至变更配置页面更改实例规格。



网络配置

网络配置为您提供当前Hologres实例的公共网络、经典网络、VPC网络及指定VPC四种网络类型，您可以根据业务情况选择网络类型并连接相关工具。

网络类型	域名	说明	使用场景	操作

网络类型	域名	说明	使用场景	操作
公网	<instancename>-cn-<region>.hologres.aliyuncs.com:80	公共网络。公网有流量开销，但是目前Hologres暂不针对流量收费，支持带宽200Mbps。	适用于无网络访问限制的场景，包括如下场景。 • 阿里云以外的设备需要通过公网地址访问Hologres实例。 • 需要访问的Hologres实例位于不同的地域或者网络类型。	默认关闭，您可以选择开启或关闭该网络类型。 说明 聚石塔和物流云受安全监管要求，不透出公网Endpoint。
经典	<instancename>-cn-<region>-internal.hologres.aliyuncs.com:80	阿里云的公共基础网络，IP地址、网络的规划和管理由阿里云负责。	用于经典网络内的应用访问Hologres，例如应用部署在ECS的经典网络内，可以连接Hologres的经典网络地址。	系统默认只能开启该网络类型。
VPC	<instancename>-cn-<region>-vpc.hologres.aliyuncs.com:80	逻辑隔离的私有网络，一个VPC就是一个隔离的网络环境，支持通过专线连接。	相对经典网络而言，VPC具有更高的安全性和灵活性。适用于如下场景。 • 应用部署在VPC网络内的ECS中，可以直接连接Hologres的VPC Endpoint（不支持跨Region连接）。 • 若是本地IDC机房，可以通过专线的方式打通到VPC网络，从而可以使用Hologres的VPC Endpoint连接应用。	您可以选择开启或关闭该网络类型。 说明 建议不要关闭该网络类型，关闭后无法使用HoloWeb。
指定VPC	<instancename>-cn-<region>-vpc-st.hologres.aliyuncs.com:80	逻辑隔离的私有网络，且仅与指定的VPC联通。 注意 目前仅在华东2（上海）、华北2（北京）和华南1（深圳）区域开放。	相对VPC网络而言，指定VPC网络仅和您绑定的VPC网络联通，非绑定的VPC网络不通。	默认不绑定任何VPC网络。 绑定VPC网络请单击操作列的**绑定VPC网络**，在对话框中选择**专有网络（VPC）**和**专有网络交互机**。

连接方式

连接方式模块为您展示了常用开发工具的连接方式，您可以根据业务需求选择合适的工具，如下所示：

- **JDBC连接**

使用其他ETL或BI工具通过标准的JDBC连接方式访问Hologres，语句如下：

```
jdbc:postgresql://<AccessID>:<AccessKey>@<Endpoint>:<Port>/<database>?tcpKeepAlive=true
```

关于JDBC的详细配置，请参见JDBC。

- **PSQL客户端连接**

  Hologres兼容PostgreSQL，您可以通过psql客户端连接Hologres进行数据开发，语句如下：

  ```
  PGUSER=<AccessID> PGPASSWORD=<AccessKey> psql -p <Port> -h <Endpoint> -d <Database>
  ```

  关于通过psql客户端连接Hologres的快速入门，请参见PSQL客户端。

参数说明如下表所示。

参数	描述
<AccessID>	当前访问账号的AccessKey ID。 您可以单击用户中心，获取AccessKey ID。
<AccessKey>	当前访问账号的AccessKey Secret。 您可以单击用户中心，获取AccessKey Secret。
<Port>	Hologres实例的网络端口。 您可以进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
<Endpoint>	Hologres实例的网络域名。 您可以进入Hologres管理控制台的实例详情页，从**实例配置**获取网络域名。
<Database>	连接的数据库名称。 您可以进入Hologres管理控制台的实例详情页，从**DB管理**获取数据库名称。 如果您没有新建数据库，请使用系统默认分配的**postgres**数据库连接。

2.3.4. 用户和DB管理

您可以使用Hologres管理控制台的用户管理和DB管理模块，对用户和DB进行管理并为用户授权。

用户管理

当您在实例详情页面单击**账号管理**时，该页面会自动登录并跳转到HoloWeb的对应页面。

您可以在HoloWeb的**用户管理**页面可视化新增或删除用户，以及为用户授权。更多关于用户管理的操作指导，请参见用户管理。

DB管理

当您在实例详情页面单击**Database管理**时，该页面会自动登录并跳转到HoloWeb的对应页面。

您可以在HoloWeb的**DB授权**页面，可视化的创建及删除数据库，并为数据库选择对应的权限策略。更多关于DB管理的操作指导，请参见DB管理。

> **说明** 成功购买实例后，系统默认生成一个名为postgres的数据库，该数据库分配的资源较少，仅用于管理，不会显示在DB管理页面。处理实际业务请您新建数据库。

2.3.5. 查看监控指标

交互式分析Hologres管理控制台的监控告警模块，为您提供了实例的CPU使用率、内存使用率和存储用量，SQL连接数，SQL语句执行的QPS和延迟情况，以及导入数据的RPS等指标监控，方便您查看并及时处理实例的异常。

前提条件

开通交互式分析Hologres，详情请参见购买Hologres。

查看监控指标

1. 登录Hologres管理控制台。
2. 在Hologres管理控制台顶部菜单栏，选择目标地域。
3. 单击Hologres管理控制台左侧导航栏的**实例列表**，进入**Hologres引擎管理**页面。
4. 在实例列表单击目标实例名称，进入实例详情页。

   您也可以单击目标实例**操作**列的**管理**，进入实例详情页。
5. 在实例详情页左侧导航栏，单击**监控信息**，进入**实例总览**页面查看监控指标。

监控指标类别

Hologres提供的实例监控指标请参见Hologres管控台的监控指标。

2.4. 使用阿里云资源组管理实例

Hologres支持阿里云资源组功能管理Hologres实例，该功能可以帮助您更加细致的管理Hologres实例。本文为您介绍使用阿里云资源组管理Hologres实例的最佳实践。

背景信息

某公司A有三个部门，每个部门都会用到多种云资源。公司A只有一个阿里云账号，该阿里云账号下有多个Hologres实例。

公司A有如下要求：

- 部门独立管理：每个管理员各自能够独立管理部门人员及其访问权限。
- 按部门分账：财务部门希望能够根据部门进行出账，以解决财务成本分摊的问题。

公司A有如下解决方案：

- 多账号方案
  - 可以满足部门独立管理：公司A注册三个阿里云账号（对应三个部门），每个阿里云账号有对应部门管理员可以独立管理成员及其访问权限。

- 可以满足按部门分账：每个阿里云账号有默认账单，可以利用阿里云提供的多账号合并记账能力来解决统一账单和发票问题。

- 单账号给资源打标签方案
  - 无法满足部门独立管理：给资源打标签可以模拟部门分组，但无法解决部门管理员独立管理部门成员及其访问权限的问题。
  - 可以满足按部门分账：按照部门组给资源打上对应标签，根据标签实现分账。
- 资源组管理方案
  - 可以满足部门独立管理：每个资源组有对应的管理员，资源组管理员可以独立管理成员及其访问权限。
  - 可以满足按部门分账：账单管理功能支持按资源组进行分账，解决财务成本分摊的问题。

使用限制

由于阿里云的限制，暂时不支持RAM账号进行购买、升降配、续费和按量付费转包年包月行为进行阿里云资源组级别的控制。如果RAM账号需要进行购买、升降配、续费、按量付费转包年包月操作，请赋予该RAM账号云账号全部资源的AliyunBSSOrderAccess权限。

资源组管理方案配置步骤

1. 创建RAM用户

   创建三个RAM用户，账号示例如下，详细步骤请参见创建RAM用户。
   - Alice@secloud.onaliyun.com。
   - Bob@secloud.onaliyun.com。
   - Charlie@secloud.onaliyun.com。

   > **说明** 下面步骤的操作均以RAM用户Alice为例，介绍如何将其设为部门的管理员。

2. 登录资源管理控制台。
3. 在**资源组**页面，单击**创建资源组**。

   更多资源组操作，请参见管理资源组。
4. 在**创建资源组**面板，填写**资源组标识**和**资源组名称**后，单击**确定**。

   > **说明** 创建三个资源组，分别命名为：BU1、BU2、BU3。

5. 权限配置

   以下操作以为RAM用户Alice配置查看和管理资源组BU1中Hologres实例的权限为例。

   i. 单击资源组BU1操作列的**权限管理**。

   ii. 在**权限管理**页签，单击**新增授权**。

iii. 在**新增授权**面板，配置如下参数。

参数	说明
授权范围	选择**指定资源组**并选择BU1。
授权主体	输入Alice@secloud.onaliyun.com。
选择权限	选择**系统策略**下的**AliyunHologresFullAccess**和**AliyunBSSOrderAccess**。更多RAM权限策略的说明，请参见授予RAM用户权限。

iv. 单击**确定**后，单击**完成**，完成权限配置。

此时，Alice具备查看和管理资源组BU1中Hologres实例的权限，可以在资源组BU1中进行购买实例、删除实例、停止实例、为实例续费、为实例升降配等操作。

> **说明** 将Bob和Charlie分别设置为BU2和BU3资源组的管理员，请重复执行以上步骤。

执行结果

Alice、Bob和Charlie分别是BU1、BU2、BU3的Hologres管理员，使用各自RAM账户登录Hologres管理控制台后，可以看到对应资源组，并可以在资源组中购买和管理（包括升降配、续费、停机、删除等操作）Hologres实例。

资源组分账

关于资源组分账的功能，请参见资源组分账。

3.连接开发工具

3.1. 概述

本文为您介绍常见的连接交互式分析Hologres的开发工具。

Hologres兼容PostgreSQL协议，提供JDBC/ODBC Driver，您可以选择熟悉的开发工具连接Hologres进行数据开发。

> 说明
> - 您可以从JDBC官网下载JDBC，使用JDBC连接时，请确保您的JDBC版本在42.2.18及以上。
> - 您可以从ODBC官网下载ODBC，使用ODBC连接时，请确保您的ODBC版本在psqlodbc_11_01_0000及以上。

常见的开发工具如下表所示。

名称	工具描述及使用场景	文档链接
HoloWeb	推荐使用 基于Hologres引擎开发的一站式大数据开发平台。 支持可视化和SQL两种操作，能满足不同开发经验的需求，完全适用于Hologres的各种开发操作。	HoloWeb
DataWorks	深度集成于DataWorks的一站式大数据开发平台。 适用于大数据开发场景，例如需要DataWorks调度作业等，更能满足强依赖DataWorks开发的场景。	概述
PSQL	基于Postgres的一个交互式命令行客户端。 全SQL执行界面，更适用于有SQL开发基础的开发者。	PSQL客户端
JDBC	JDBC（Java Data Base Connectivity，java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。 全SQL执行，更适用于有JDBC开发经验的开发者。	JDBC
Navicat	一款连接多数据库的开发工具。 支持可视化和SQL两种操作方式，更适用于业务上强依赖Navicat的场景。	Navicat

名称	工具描述及使用场景	文档链接
SQL Workbench/J	SQL Workbench/J是一款免费、跨平台的SQL查询分析工具。 全SQL执行界面，更适用于有SQL开发基础的开发者。	SQL Workbench/J

3.2. DataWorks数仓开发

3.2.1. 概述

本文为您介绍什么是DataWorks，以及如何使用DataWorks进行Hologres数据开发。

DataWorks（数据工场，原大数据开发套件）是阿里云重要的PaaS（Platform-as-a-Service）平台产品，为您提供数据集成、数据开发、数据地图、数据质量和数据服务等全方位的产品服务，一站式开发管理的界面，帮助企业专注于数据价值的挖掘和探索。

Hologres与DataWorks深度集成，您可以通过DataWorks绑定Hologres实例，进行一站式实时数仓开发，包括数据集成、数据开发、数据质量、数据服务等，满足不同的业务场景开发和管理需求。在使用DataWorks进行Hologres数据开发时，您可以选择使用以下方式。

- （推荐）DataStudio

  DataWorks的DataStudio（数据开发）模块为您提供了界面化、智能高效的大数据开发与测试服务，您可以在绑定Hologres实例后，通过使用Hologres节点进行Hologres数据开发，包括SQL管理、MaxCompute数据同步等功能，操作简单高效，支持标准化、简单化的开发管理服务和一站式实时数仓构建服务以及高效、便捷的研发服务，助力开发更敏捷和更高效。

  支持的Region：Hologres已开服的所有Region。

- HoloStudio

  HoloStudio是DataWorks旗下的一站式OLAP开发平台。HoloStudio基于Hologres构建，通过可视化、引导式方式为您提供标准化、无门槛的开发服务和一站式实时数仓构建服务以及高效、便捷的管理服务，从而提升研发效率。

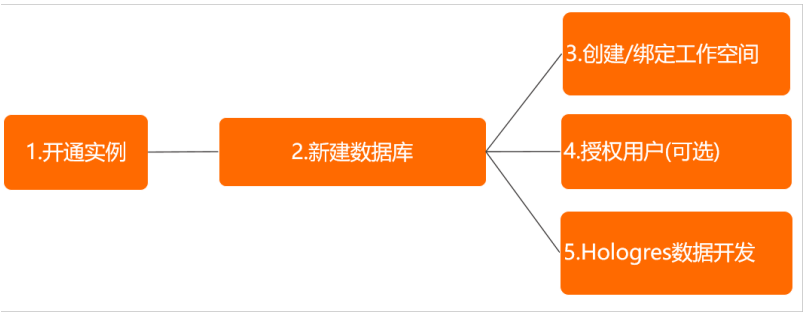
  > **说明** HoloStudio会逐步升级迭代为DataStudio，更建议您使用DataStudio或者HoloWeb进行开发，功能更加丰富且操作更简便。

  支持的Region：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。

3.2.2. DataWorks快速入门

DataWorks作为基于交互式分析（Hologres）引擎的一站式开发平台，在为您提供SQL界面开发的同时，也支持使用可视化UI界面完成Hologres开发，并进行周期性调度。

本小节主要为您介绍使用DataWorks的操作步骤，通过以下简单步骤的综述，帮助您快速掌握在DataWorks中进行Hologres数据开发的基本使用流程。



> **说明** 如果您更倾向于使用psql客户端进行Hologres开发，可参见PSQL客户端。

前提条件

在开始使用DataWorks之前，请确保您已经成功开通Hologres实例，详情请参见购买Hologres。

操作步骤

使用DataWorks进行Hologres数据开发的完整操作步骤如下：

1. **新建数据库**

   实例购买成功后，系统会自动生成一个名叫postgres的数据库，该数据库分配的资源少且仅供管理用途，实际业务需要您新建一个数据库。为节省SQL操作，Hologres管控台支持直接可视化新建数据库。

进入管理控制台的实例详情页，单击左侧导航栏的**数据库管理**，跳转至**DB授权**页面，单击右上角**新增数据库**，在**新增数据库**对话框配置如下参数，即可新建数据库成功。

参数	说明
实例名	选择在哪个Hologres实例上创建数据库。默认展示当前已登录实例的名称，您也可以在下拉框中选择其他Hologres实例。
数据库名称	自定义数据库名称。 **说明** 配置的数据库名称必须唯一。
简单权限策略	您可以为创建的数据库选择一种权限策略。更多关于权限策略的说明，请参见： • SPM：简单权限模型，该权限模型授权是以DB为粒度，划分admin（管理员）、developer（开发者）、writer（读写者）以及viewer（分析师）四种角色，您可以通过少量的权限管理函数，即可对DB中的对象进行方便且安全的权限管理。 • SLPM：基于Schema级别的简单权限模型，该权限模型以Schema为粒度，划分<db>.admin（DB管理员）、<db>.<schema>.developer（开发者）、<db>.<schema>.writer（读写者）以及<db>.<schema>.viewer（分析师），相比于简单权限模型更为细粒度。 • 专家权限模型：Hologres兼容PostgreSQL，使用与Postgres完全一致的权限系统。

说明 Hologres兼容PostgreSQL，使用与PostgreSQL完全一致的权限模型（简称专家模式）。基于对业务理解和实践经验，Hologres抽象了一套简单权限模型（SPM）来简化授权操作，详情可以参见简单权限模型概述。新建DB时，您可以选择为新DB开启简单权限模型（推荐操作）并通过管控台来为新用户可视化授权，以简化授权操作，具体操作请参见DB管理。

2. **创建或绑定工作空间**

   绑定Hologres实例，分为以下两种情况：

   - **无工作空间**

     如果您没有使用过DataWorks工作空间，需要在DataWorks管控台创建一个同地域的工作空间并绑定。

     在DataWorks管理控制台单击左侧导航栏的**工作空间列表**，并单击**创建工作空间**，填写工作空间信息后，进入**DataStudio**页面，进行Hologres开发，详情请参见新建工作空间绑定Hologres实例，您同时也可以根据自身需求，选择是否同时开通其它计算引擎服务。

     > **说明** Hologres在标准DataWorks工作空间分开发环境和生产环境，但由于当前DataStudio暂不支持环境切换，导致生产环境的数据在DataStudio不可查询，建议开发和生产绑定同一个数据库。

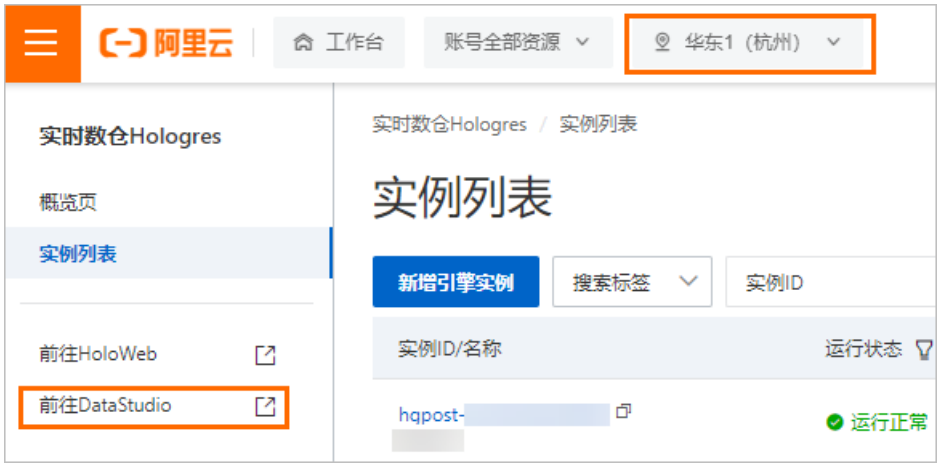
   - **已有工作空间**

     如果您已创建DataWorks工作空间，可以直接使用该工作空间绑定Hologres实例。具体操作步骤如下：

     a. 进入DataStudio

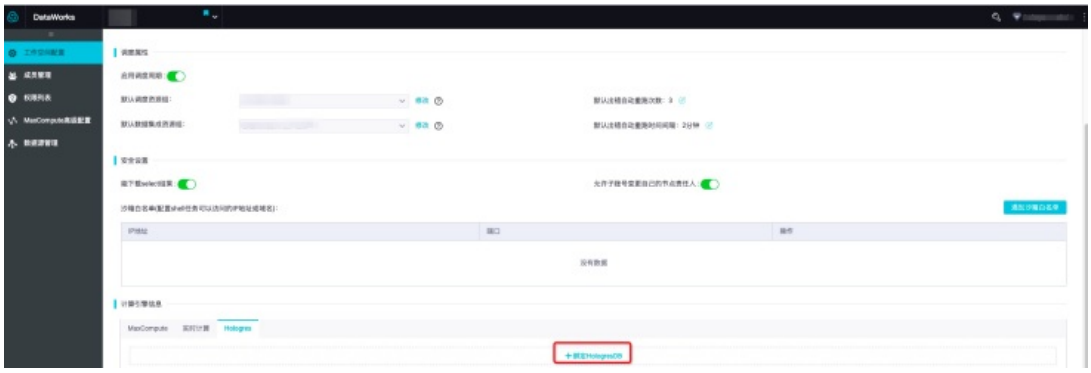
     > **说明** 请确保工作空间和实例在同一Region，否则会绑定不成功。

     在Hologres管控台选择实例开通的区域，单击页面左侧导航栏的**前往DataStudio**，即可进入对应的工作空间。



b. 绑定实例

进入DataStudio后，单击右上角图标，进入**工作空间配置**页面，单击**绑定HologresDB**进行实例绑定。更多关于绑定实例的详细操作步骤参见选择已有工作空间绑定Hologres实例。



绑定完成之后，返回**DataStudio**，在**PG管理**页面单击**刷新**，即可看到已成功绑定的数据库。

3. **授权子账号（可选）**

若您需要给子账号授权，可以按照以下步骤操作：

i. 创建子账号

在Hologres管控台实例详情页单击左侧导航栏**账号管理 > 新增用户**，勾选需要授权的子账号，并选择**成员角色**。可以直接授权为**实例超级管理员Superuser**（拥有实例所有权限，无需额外授权），也可以授权为**普通用户**（访问Hologres还需要额外授权）。

ii. 授权子账号

若您的子账号是普通用户normal，还需要给子账号授权才能访问Hologres实例。您可以选择使用专家权限模式或者简单权限模型（推荐）为子账号授权。

- **简单权限模型（推荐）**

  在管控台新建DB并开启简单权限模型之后，可以直接将子账号加入对应的用户组来授权。

  关于简单权限模型的使用可以参见简单权限模型的使用。

- **专家模式授权**

  专家权限模式授权采用的是标准的PostgreSQL授权语句，您可以参照以下语句进行授权：

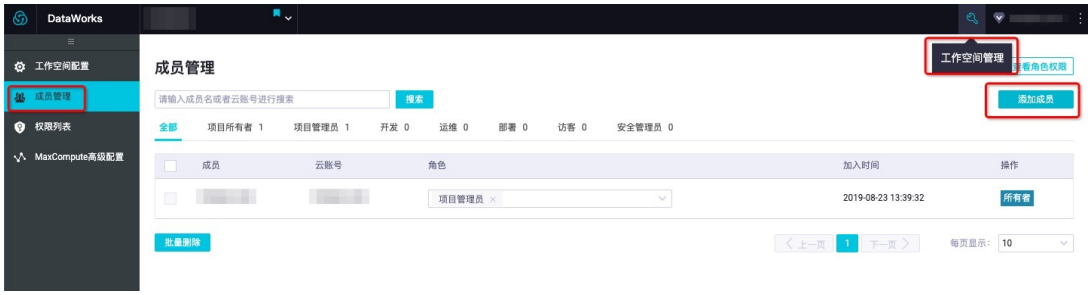
```
GRANT SELECT, INSERT ON TABLE TABLENAME TO "云账号ID/云邮箱";//若是子账号还需要在云账号ID前加"p4_"，即"p4_id"
GRANT SELECT, INSERT ON ALL TABLES IN SCHEMA public TO "云账号ID/云邮箱";//若是子账号还需要在云账号ID前加"p4_"，即"p4_id"
```

  更多关于专家权限模型的使用可以参见专家权限模型。

- 使用DataStudio开发

  若您的子账号需要使用DataStudio完成Hologres开发，还需要将子账号添加进DataWorks的项目工作空间。

  在DataWorks工作空间配置页面单击**工作空间管理 > 成员管理 > 添加成员**，并给成员授予一定的项目权限。



4. **DataStudio开发**

   成功绑定数据库之后，即可在DataStudio中进行数据开发，详情请参见DataStudio概述。

Hologres作为兼容PostgreSQL生态的实时数仓引擎，支持连接所有能对接PostgreSQL的开发工具，关于Hologres的更多开发工具请参见连接开发工具。

3.2.3. 绑定Hologres实例

本文将为您介绍如何将实时数仓（Hologres）实例绑定至DataWorks工作空间。

前提条件

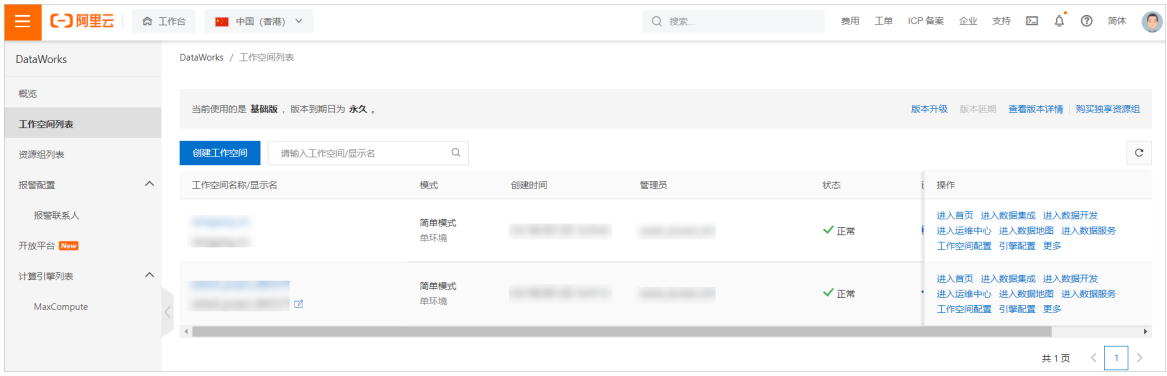
1. 使用DataStudio前，请确保已经成功开通Hologres实例，详情请参见购买Hologres。
2. 实例创建成功后会默认生成一个名叫postgres的数据库，仅供管理用途，实际业务需要您新建一个数据库。

新建工作空间绑定Hologres实例

若您在此之前并未开通DataWorks工作空间，可以按照以下步骤来绑定实例。

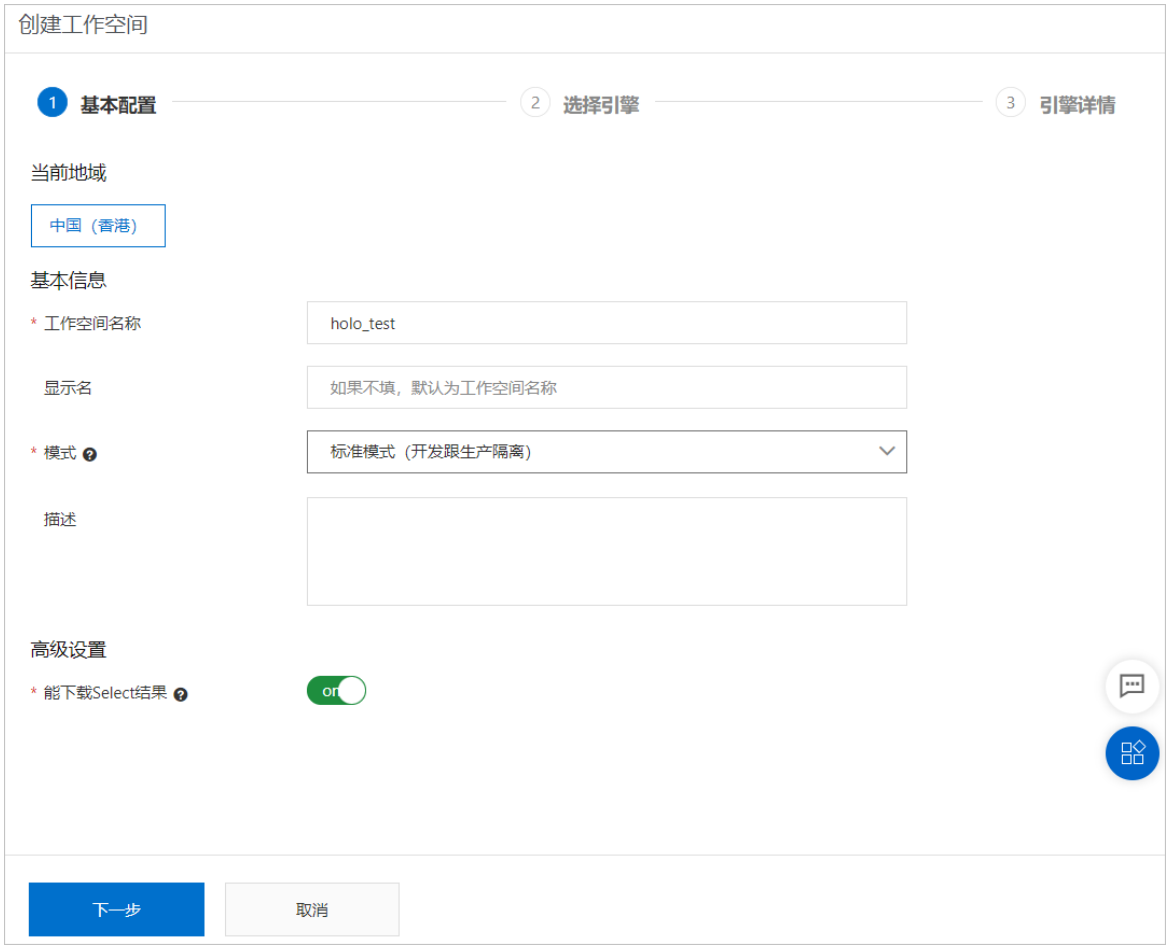
1. **进入DataWorks管控台**

   使用阿里云账号进入DataWorks管控台，选择与实例同一个region，单击左侧菜单栏**工作空间列表**进行新建工作空间。



2. **新建工作空间**

   在工作空间列表页面单击**创建工作空间**，输入工作空间名称，模式选择可根据项目情况选择（详情请参见简单模式和标准模式的区别）。

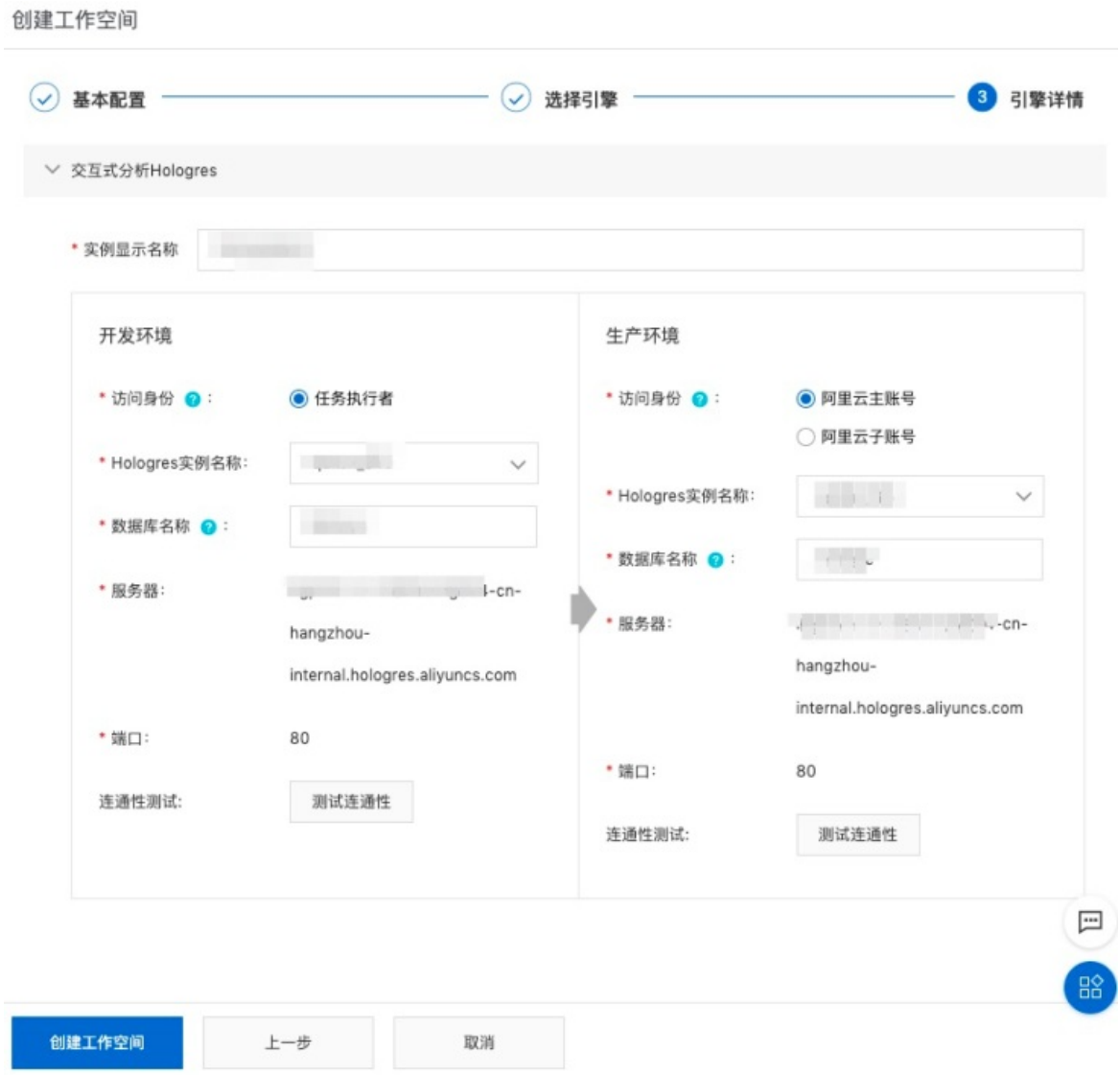


3. **选择计算引擎**

   选择计算引擎为**Hologres**，并选择对应的实例付费方式，可根据项目情况选择是否同时开通其他引擎服务。

4. **填写实例信息**

   填写实例信息，当前标准版的工作空间分开发环境和生产环境，以保证数据的安全性。最后单击**创建工作空间**即可完成创建。

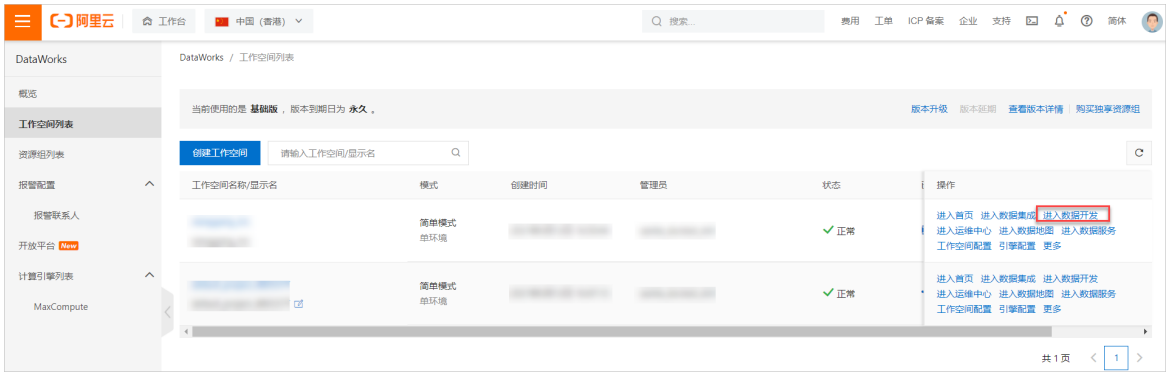


环境	配置项	说明
无	实例显示名称	自定义名称。
开发环境	访问身份	任务执行者。
	Hologres实例名称	下拉框可选。
	数据库名称	需要绑定的数据库名称，实例创建后有一个名为postgres的默认DB（仅供管理用途），实际业务使用请前往Hologres管理控制台新建Database。
	服务器	Hologres实例的网络地址，实例选择后自动生成。
	端口	Hologres实例的网络端口，实例选择后自动生成。
	连通性测试	测试是否连通。

环境	配置项	说明
生产环境	访问身份	DataWorks任务提交调度后，作为在Hologres引擎内执行代码的身份（账号），有如下两种身份。 ○ **阿里云主账号**：阿里云账号。 ○ **阿里云子账号**：阿里云RAM用户。 账号详情请参见账号概述。
	Hologres实例名称	下拉框可选。
	数据库名	需要绑定的数据库，建议绑定开发环境的数据库。
	服务器	Hologres实例的网络地址，实例选择后自动生成。
	端口	Hologres实例的网络端口，实例选择后自动生成。
	连通性测试	测试是否连通。

5. **进入DataStudio**

   工作空间创建成功后会跳转至管控台，可看到已配置好的工作空间信息，单击**进入数据开发**，即可进入DataStudio。



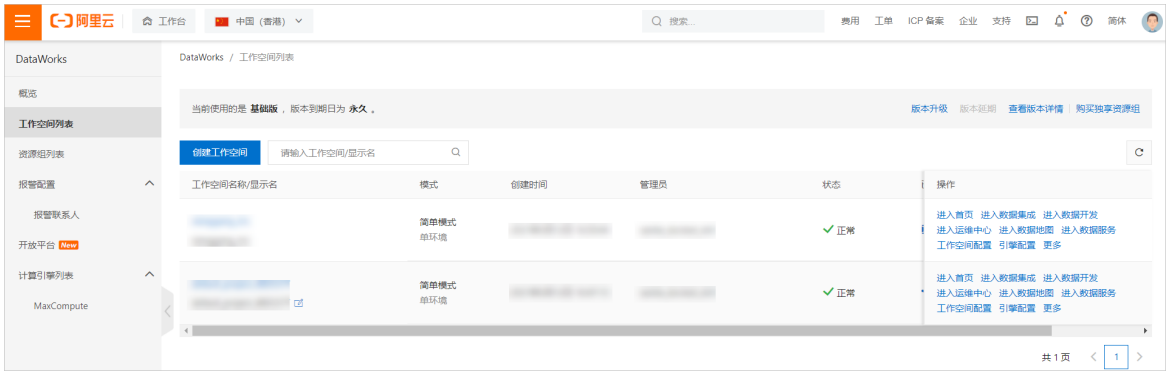
选择已有工作空间绑定Hologres实例

若您已有DataWorks项目空间，可将实例绑定至该工作空间。具体操作步骤如下。

> **说明** 工作空间和开通的Hologres实例必须在同一region才能成功绑定。

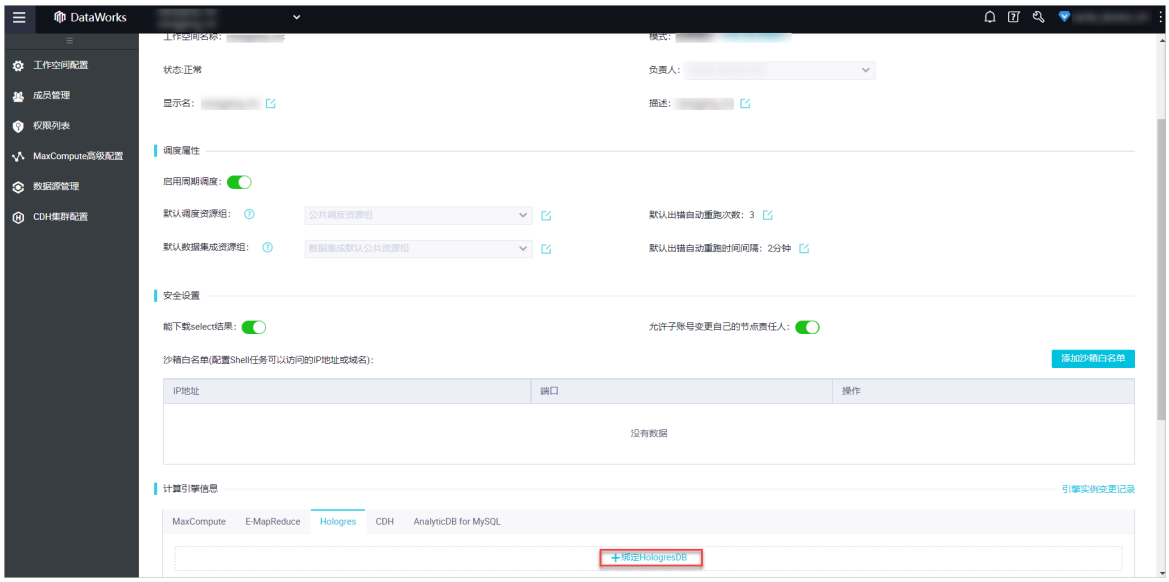
1. **进入工作空间**

   在DataWorks管控台，单击**工作空间列表**。



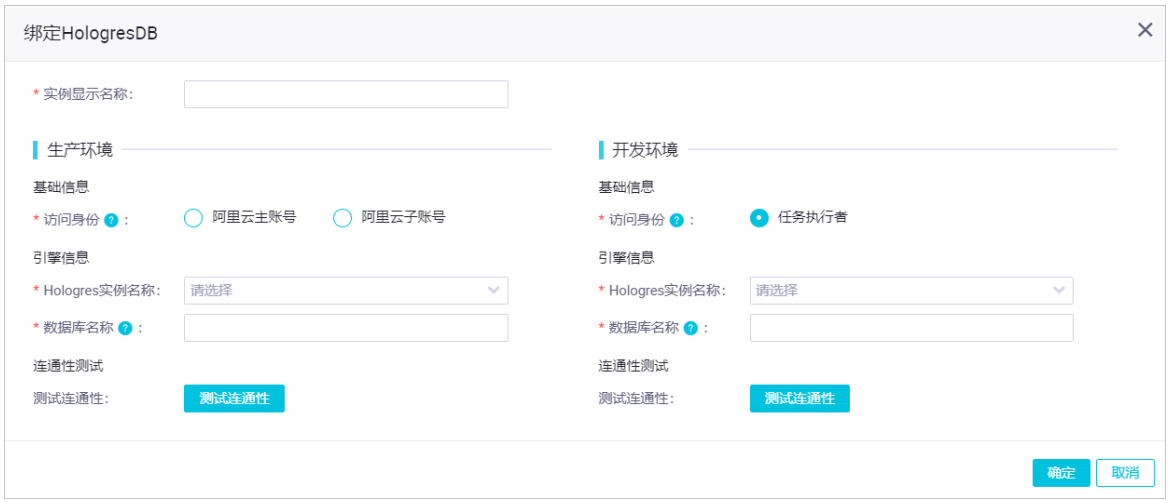
2. **工作空间配置**

   在工作空间列表，单击操作列的**工作空间配置>更多设置**，进入工作空间配置页面，在**计算引擎信息**选择**Hologres**，并单击**绑定HologresDB**进行实例的绑定。



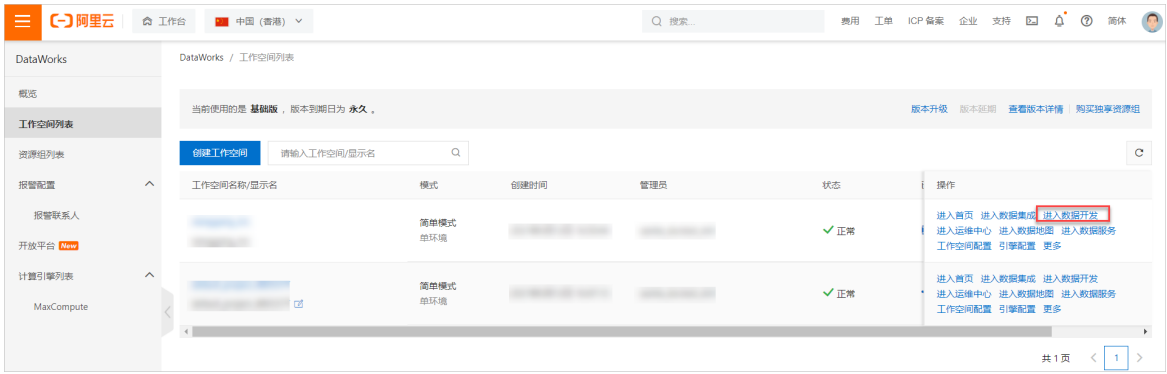
3. **绑定Hologres实例**

   在**绑定HologresDB**页面填写相关信息，并单击**测试连通性**，显示**测试联通性通过**表示能正常绑定，单击**确定**即可。



4. **进入DataStudio**

绑定Hologres实例后，返回**工作空间列表**页面，单击**进入数据开发**，即可进入DataStudio。



常见问题

- 绑定Hologres实例按钮为灰色
  - 问题现象：绑定Hologres实例时，**绑定HologresDB**按钮为灰色。
  - 可能原因：
    - 当前账号不是所登录DataWorks项目空间的管理员。
    - 当前DataWorks绑定的计算引擎实例数量已达当前版本的上限。
  - 解决方法：
    - 联系DataWorks项目空间的管理员，将当前账号设置为项目管理员，即可添加Hologres实例。
    - 给DataWorks项目空间的管理员授予Hologres实例权限，再使用DataWorks项目的管理员添加Hologres实例。
    - 解绑已绑定到DataWorks的计算引擎实例或升级DataWorks版本，不同DataWorks版本支持绑定的计算引擎实例数量请参见各版本功能支持详情。
- 测试连通性不通过
  - 问题现象：绑定Hologres实例时，测试连通性提示不通过。
  - 可能原因：填写的绑定信息有误或者当前账号无Hologres实例权限。
  - 解决方法：
    - 实例显示名称可能在DataWorks里面有重复，可以更换一个显示名称。
    - 检查数据库名是否填写正确。
    - 检查填写的信息是否有空格，如果有空格，请删除空格。
    - 检查当前账号授予Hologres实例权限，再使用DataWorks项目的管理员添加Hologres实例。

3.2.4. 子账号使用DataWorks

本文为您介绍主账号如何创建子账号，并授权子账号使用DataWorks进行数据开发。

背景信息

系统默认设置购买实例的主账号为超级管理员Superuser。Superuser拥有该实例的所有权限。

其他用户必须经过主账号授权才可以访问实例。

子账号包括以下两种权限控制。

- RAM权限

RAM权限为可选权限。子账号被授予相应RAM权限后，可以在Hologres管理控制台管控实例，例如购买或删除实例、升降配实例资源、修改网络类型以及查看实例信息等。

- 实例开发权限

  实例开发权限为必选权限。子账号必须经过主账号授予实例的开发权限后，才能连接实例并进行数据开发。

Hologres支持使用简单权限模型和专家权限模型为子账号授权。两种权限模型的介绍如下。

- 简单权限模型（推荐）

  简单权限模型是Hologres基于PostgreSQL的授权体系，为提升用户体验，制定的一种粗粒度的权限模型，详情请参见简单权限模型概述。

- 专家权限模型

  专家权限模型是与标准PostgreSQL语句相同的授权体系，简称专家模式。您可以按照标准的PostgreSQL授权语句为子账号授权，详情请参见专家权限模型。

创建子账号

如果您已有子账号，请跳过该步骤。

1. 使用主账号登录阿里云官网。
2. 使用主账号登录RAM管理控制台。
3. 在左侧导航栏的**人员管理**菜单下，单击**用户**。
4. 单击**创建用户**。

   您也可以单击**添加用户**，一次性创建多个子账号。
5. 在**用户账号信息**区域下，输入**登录名称**和**显示名称**。
6. 在**访问方式**区域下，选择**控制台访问**。
7. 设置子账号的登录密码。
8. 单击**确定**。

授权子账号

- 授予子账号RAM权限。

  主账号授予子账号相应的RAM权限后，子账号可以在Hologres管理控制台执行查看、购买或删除实例等操作。详情请参见概述。

- 授予子账号实例开发权限。

  子账号必须经过主账号授予实例的开发权限后，才能在Hologres实例中进行数据开发。详情请参见授予RAM用户实例的开发权限。

添加子账号至DataWorks工作空间

您需要添加子账号至相应的DataWorks工作空间，才可以使用子账号在DataWorks中进行数据开发。步骤如下：

1. 进入**工作空间配置**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击目标工作空间操作列的**工作空间配置**。

iv. 在**工作空间配置**对话框，单击**更多设置**，进入**工作空间配置**页面。

您也可以进入**数据开发**页面，单击右上角的图标，进入**工作空间配置**页面。

2. 在左侧导航栏，单击**成员管理**。
3. 在**成员管理**页面，单击右上角的**添加成员**。
4. 在**添加成员**对话框，单击**刷新**，同步当前阿里云账号下的所有子账号至**待添加账号**列表中。

5. 在**待添加账号**勾选需要添加的成员账号，单击**>**，移动需要添加的子账号至**已添加账号**中。
6. 勾选需要授予的角色，单击**确定**。

工作空间的创建者默认为管理员角色。具体角色的权限说明请参见附录：预设角色权限列表（空间级）。角色描述如下表所示。

角色	描述
项目所有者	项目所有者拥有工作空间的所有权限。
空间管理员	除了拥有开发角色和运维角色的全部权限外，还可以进行添加或移出工作空间成员并授予角色，创建自定义资源组等操作。
开发	负责数据开发页面的设计和维护工作。
运维	负责在运维中心页面管理全部任务的运行情况并进行相应处理。
部署	仅在多工作空间模式时审核任务代码并决定是否提交运维。
访客	仅有只读权限，可以查看数据开发页面的业务流程设计和代码内容。
安全管理员	仅有数据保护伞模块的操作权限，详情请参见数据保护伞。

7. 使用子账号登录DataWorks控制台，单击**数据开发**页面的DataStudio 。
8. 您可以根据业务需求进行数据开发，详情请参见概述。

移除DataWorks工作空间的子账号

1. 进入**工作空间配置**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击目标工作空间操作列的**工作空间配置**。

iv. 在**工作空间配置**对话框，单击**更多设置**，进入**工作空间配置**页面。

您也可以进入**数据开发**页面，单击右上角的图标，进入**工作空间配置**页面。

2. 在左侧导航栏，单击**成员管理**。
3. 在**成员管理**页面，单击目标成员操作列的**删除**。
4. 单击**确认**。

> **说明** 子账号被工作空间移除后，仍然拥有Hologres实例的开发权限。

3.2.5. DataStudio（推荐）

3.2.5.1. DataStudio概述

本文为您介绍什么是DataStudio，以及DataStudio支持的Hologres节点功能。

简介

DataWorks基于Hologres、MaxCompute等大数据计算引擎，为您提供专业高效、安全可靠的一站式大数据开发与治理平台，自带阿里巴巴数据中台与数据治理最佳实践，赋能各行业数字化转型。

Hologres与DataWorks深度兼容，通过可视化方式支持您在DataWorks中使用DataStudio模块进行数据的SQL开发，为您提供标准化、无门槛的开发服务和一站式构建实时数据仓库服务，以及高效、便捷的管理服务。同时结合DataWorks的调度等能力，简便、快捷的完成数据的处理和应用。

功能

DataStudio支持的Hologres节点功能如下。

- Hologres SQL

  您可以使用标准的PostgreSQL进行Hologres开发，并将SQL提交调度，进行周期性作业。

- 一键表结构导入

  您可以使用DataStudio一键表结构同步功能批量创建Hologres外部表。

- 一键表数据同步

  您可以使用DataStudio导入MaxCompute数据至Hologres，并可以提交调度周期性作业。

3.2.5.2. Hologres SQL

在DataStudio中，Hologres SQL是基于SQL命令语句的编辑器，支持您在DataStudio中通过SQL命令语句进行Hologres开发，为您提供秒级交互式查询体验。本文为您介绍如何在DataStudio中使用Hologres SQL进行Hologres开发。

前提条件

- 阿里云账号注册，详情请参见阿里云账号注册流程。
- 开通Hologres并绑定至DataWorks工作空间，详情请参见绑定Hologres实例。

1. 新建业务流程

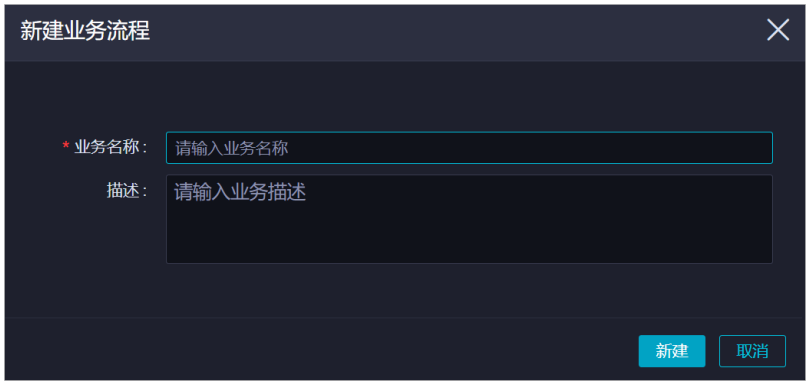
i. 使用阿里云主账号进入DataWorks管控台，选择与实例同一个region，单击左侧菜单栏**工作空间列表**。

ii. 在**工作空间列表**页面，单击工作空间操作列的**进入数据开发**，进入DataStudio页面。

iii. 单击左侧导航栏的图标，进入**数据开发**页面。

iv. 鼠标悬停至顶部菜单栏的**新建**，单击**新建业务流程**。

v. 在**新建业务流程**对话框配置如下参数。



参数	说明
业务名称	自定义业务名称。
描述	自定义业务描述。

vi. 单击**新建**，完成新建业务流程。

2. 新建Hologres SQL节点

i. 在**数据开发**页面，鼠标悬停至顶部菜单栏的**新建**，选择**新建节点** > **Hologres** > **Hologres SQL**。

ii. 在**新建节点**对话框中，选择**引擎实例**、输入**名称**和选择**路径**。

iii. 单击**提交**，完成新建Hologres SQL节点。

3. Hologres开发

打开新建的Hologres SQL节点，输入如下标准的Postgresql语句示例进行Hologres开发，单击图标。

如下命令语句新建一个名称为supplier_holo的表，并给表中插入数据，最后查询表中数据，为您简单的展示一个Hologres开发的完整流程。

```
BEGIN;
CREATE TABLE supplier_holo (
 s_suppkey bigint NOT NULL,
 s_name text NOT NULL,
 s_address text NOT NULL,
 s_nationkey bigint NOT NULL,
 s_phone text NOT NULL,
 s_acctbal bigint NOT NULL,
 s_comment text NOT NULL,
PRIMARY KEY (s_suppkey)
);
CALL SET_TABLE_PROPERTY('supplier_holo', 'bitmap_columns', 's_suppkey,s_nationkey,s_acc
tbal,s_name');
CALL SET_TABLE_PROPERTY('supplier_holo', 'dictionary_encoding_columns', 's_name,s_addre
ss');
CALL SET_TABLE_PROPERTY('supplier_holo', 'time_to_live_in_seconds', '31536000');
COMMIT;
INSERT INTO supplier_holo VALUES
(1, 'Supplier01', 'New York', 17, '27-918-335-1736',  575594, 'careful'),
(6, 'Supplier06', 'London', 14, '24-696-997-4969',  136579, 'final accounts '),
(10, 'Supplier03',  'Beijing', 24, '34-852-489-8585', 389191, 'ing waters'),
(18, 'Supplier04', 'Paris', 16, '26-729-551-1115', 704082, 'accounts snooze'),
(39, 'Supplier05', 'Shanghai',  8, '18-851-856-5633 611565', 88990, 'special packages')
,
(48, 'Supplier06',  'Canada', 14, '24-722-551-9498',563062, 'xpress instructions affix'
);
SELECT * FROM supplier_holo;
```

常见问题

- SQL编辑窗口不显示引擎信息
  - 问题现象

    在进行Hologres开发选择引擎实例时，SQL编辑窗口不显示引擎信息。
  - 可能原因

    引擎实例信息被隐藏。
  - 解决方法

    在**数据开发**页面左上角，单击图标，去勾选**隐藏引擎实例**。

- SQL编辑窗口显示的**Hologres引擎实例**对应是哪个具体Hologres实例？

Hologres引擎实例显示的是Hologres实例的显示名称，单击右上角图标，进入**工作空间管理**页面，在Hologres绑定列表下，查看对应的Hologres实例信息。

3.2.5.3. 一键表结构导入

本文为您介绍如何使用DataStudio一键表结构同步功能批量创建Hologres外部表。

前提条件

- 已开通DataWorks服务并创建DataWorks工作空间，详情请参见创建工作空间。
- 开通Hologres并绑定至DataWorks工作空间，详情请参见绑定Hologres实例。

背景信息

Hologres与MaxCompute在底层无缝连接，支持使用新建外部表的方式加速查询MaxCompute的数据，详情请参见通过创建外部表加速查询MaxCompute数据。

DataStudio支持**一键MaxCompute表结构同步**功能，您可以使用可视化方式批量创建外部表。

操作步骤

1. 新建**一键MaxCompute表结构同步**节点
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**数据开发**。
   iv. 在DataStudio界面的左侧导航栏，单击图标。
   v. 进入**数据开发**页面，单击顶部菜单栏的**新建**，选择**新建节点>Hologres>一键MaxCompute表结构同步**。
   vi. 在**新建节点**对话框中，选择**引擎实例**和**路径**，并输入**名称**，单击**提交**。
2. 在编辑节点页面，配置各项参数。

参数	说明
目标连接	Hologres的实例名称。
目标库	Hologres实例下的数据库名称。
模式	Hologres数据库下的Schema名称。默认为public，若您有新建Schema，可以选择您新建的Schema。
类型	目前仅支持对接MaxCompute。
服务器列表	服务器名称。Hologres实例创建成功后会自动创建名称为odps_server的服务器，可以直接调用，相关原理请参见postgres_fdw。
来源项目	MaxCompute的项目名称。
选择要直接加速的表	您可以根据实际业务情况选择整库加速或者部分加速。
表名冲突	创建表时表名称冲突策略，包含如下策略： ○ 忽略，继续创建其他表。 ○ 更新，修改同名表。 ○ 报错，不再重复创建。
数据类型不支持	创建表时数据类型不支持策略，包含如下策略： ○ 报错，导入失败。 ○ 忽略，跳过不支持字段。

3. 在**Hologres一键表结构导入**节点的编辑页面，单击图标，保存配置信息。
4. 在**Hologres一键表结构导入**节点的编辑页面，单击图标，批量创建外部表结构。
5. 查看批量创建的外部表
   i. 在左侧导航栏，单击图标，进入表管理页面。
   ii. 双击需要查看的外部表，显示表编辑页面。

3.2.5.4. 一键表数据同步

本文为您介绍如何使用DataStudio导入MaxCompute数据至Hologres。

前提条件

- 已开通DataWorks服务并创建DataWorks工作空间，详情请参见创建工作空间。
- 开通Hologres并绑定至DataWorks工作空间，详情请参见绑定Hologres实例。

背景信息

DataStudio支持**一键MaxCompute数据同步**功能，您可以使用可视化方式导入MaxCompute表数据并进行查询。该方式比创建外部表直接查询数据的性能更好。

您也可以使用SQL语句导入MaxCompute数据，详情请参见使用SQL导入MaxCompute的数据至Hologres。

操作步骤

1. 新建**一键MaxCompute数据同步**节点
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**数据开发**。
   iv. 在DataStudio界面的左侧导航栏，单击图标。
   v. 进入**数据开发**页面，单击顶部菜单栏的**新建**，选择**新建节点>Hologres>一键MaxCompute数据同步**。
   vi. 在**新建节点**对话框中，选择**引擎实例**和**路径**，并输入**名称**，单击**提交**。
2. 在编辑节点页面，配置各项参数。

参数	说明
目标连接	Hologres的实例名称。
目标库	Hologres实例下的数据库名称。
外部表来源	○ 已有外部表 已经提前在Hologres中创建映射MaxCompute数据的外部表。 ○ 新建外部表 表示无相应的外部表，需要同步时新建。
外部服务器	Hologres实例创建成功后会自动创建名称为odps_server的服务器，可以直接调用，相关原理请参见postgres_fdw。
MaxCompute项目	MaxCompute的项目名称。
MaxCompute表名	同步数据的MaxCompute表名称。
目标Schema	当前Hologres数据库下的Schema名称。默认为public，若您有新建Schema，可以选择您新建的Schema。
目标表名	需要导入数据的Hologres内部表名称。如已有表，执行后原表和数据将被删除重建。
目标表描述	自定义添加需要导入数据的Hologres内部表的描述。
同步字段	选择需要同步的MaxCompute表字段，可以选择全部字段，也可以选择部分字段。
分区配置	选择需要同步的分区字段。当前Hologres仅支持一级分区。

参数	说明
索引配置	为目标表构建索引。索引的创建请参见设置表属性。
SQL Script	自动解析出当前运行的SQL，方便参照。

3. 在新建节点的编辑页面，单击图标，保存配置信息。
4. 在新建节点的编辑页面，单击图标，导入MaxCompute表数据。
5. 查看同步的MaxCompute表数据
   i. 在左侧导航栏，单击图标，进入表管理页面。
   ii. 双击需要查看的Hologres内部表，显示表编辑页面。
6. （可选）周期性调度
   i. 在节点的编辑页面，单击节点编辑区域右侧的**调度配置**，配置节点的调度属性，详情请参见配置基础属性。
   ii. 在节点的编辑页面，单击工具栏中的图标，保存节点。
   iii. 单击工具栏中的图标，提交节点。
   iv. 在**提交新版本**对话框中，输入**变更描述**，单击**确认**。

3.2.5.5. 通过DataWorks周期性导入MaxCompute数据最佳实践

本文为您介绍如何将MaxCompute分区表数据通过DataWorks调度周期性导入Hologres分区表。

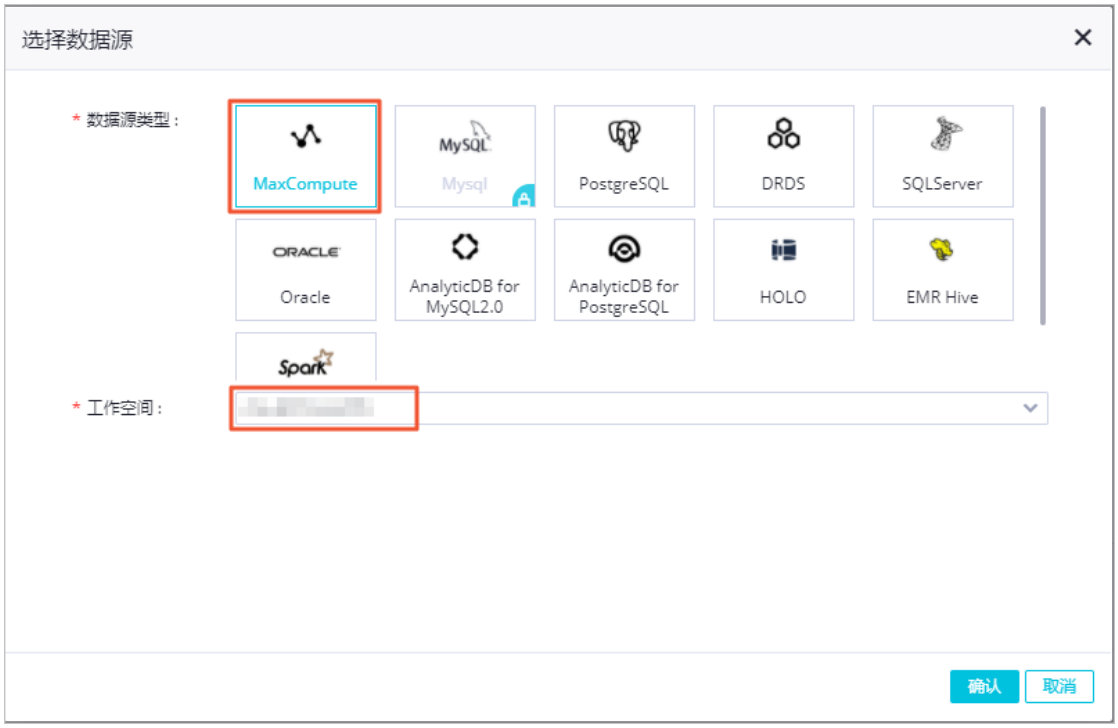
前提条件

- 已购买并开通Hologres实例，开通方法请参见购买Hologres。
- 已开通MaxCompute并创建项目，详情请参见开通MaxCompute和DataWorks。
- 已开通DataWorks服务并创建DataWorks工作空间，详情请参见创建工作空间。

操作步骤

1. MaxCompute数据准备。
   i. 登录MaxCompute控制台，在左上角选择区域，单击**查询编辑**，即可进入查询编辑器界面。

ii. 在**选择数据源**对话框，选择**数据源类型**为**MaxCompute**，**工作空间**为已创建好的项目空间。



> **说明** 如果您选择的**工作空间**模式为标准模式，在查询编辑器中提交作业实际是在开发项目（带dev标识）中提交。

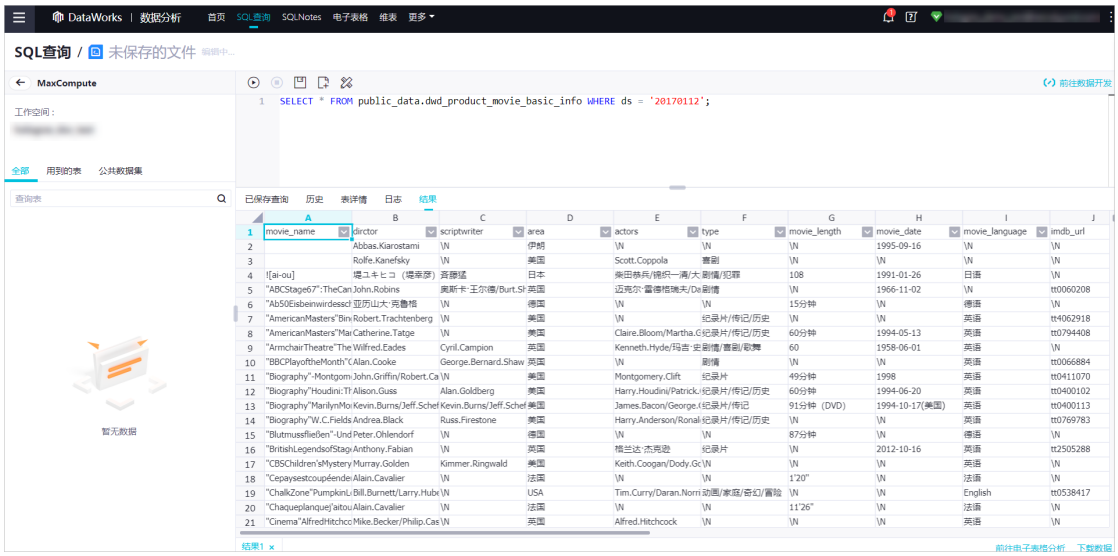
iii. 单击**确认**，即可进入查询编辑器界面。

iv. 在**SQL查询页面**，输入如下SQL语句用于创建分区表，单击**运行**。

MaxCompute分区表选择MaxCompute公共数据集public_data中的分区表dwd_product_movie_basic_info。

```
--MaxCompute分区表DDL
CREATE TABLE IF NOT EXISTS public_data.dwd_product_movie_basic_info(
  movie_name STRING COMMENT '电影名称',
  dirctor STRING COMMENT '导演',
  scriptwriter STRING COMMENT '编剧',
  area STRING COMMENT '制片地区/国家',
  actors STRING COMMENT '主演',
  `type` STRING COMMENT '类型',
  movie_length STRING COMMENT '电影长度',
  movie_date STRING COMMENT '上映日期',
  movie_language STRING COMMENT '语言',
  imdb_url STRING COMMENT 'imdb号'
)
PARTITIONED BY (ds STRING) STORED AS ALIORC;
```

v. 在**SQL查询页面**，输入如下SQL语句用于查看分区表中导入的数据，单击**运行**。



查看分区表20170112分区的数据。

```
SELECT * FROM public_data.dwd_product_movie_basic_info WHERE ds = '20170112';
```

2. Hologres中新建外部表。

新建一张Hologres外部表，用于映射MaxCompute源头表数据。外表的字段顺序和字段类型需要和MaxCompute表的一一对应。

i. 登录Hologres管理控制台，进入**HoloWeb**开发页面新建SQL查询，详情请参见新建SQL查询。

ii. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

如下语句使用 `import foreign schema` 命令，创建名称为dwd_product_movie_basic_info的Hologres外部表。

```
import foreign schema public_data limit to (dwd_product_movie_basic_info) from serv
er odps_server into public options(if_table_exist 'update');
```

3. Hologres中新建真实存储表（内部表）。

在Hologres中新建一张内部表，用于接收并存储数据。

i. 在**HoloWeb**开发页面，单击**新增SQL窗口**。

ii. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

本次示例是将MaxCompute分区表导入Hologres，因此需要在Hologres中创建的内部表为分区表。

> **说明** 如下建表语句仅是简单示例，实际建表DDL请根据实时业务需要创建，并给表设置合理的索引，以达到更优的查询性能。

```
BEGIN;
CREATE TABLE "public"."holo_dwd_product_movie_basic_info" (
 "movie_name" text,
 "dirctor" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
)
PARTITION BY LIST (ds);
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'orientatio
n', 'column');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'bitmap_col
umns', '"movie_name","dirctor","scriptwriter","area","actors","type","movie_length"
,"movie_date","movie_language","imdb_url","ds"');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'dictionary
_encoding_columns', '"movie_name:auto","dirctor:auto","scriptwriter:auto","area:aut
o","actors:auto","type:auto","movie_length:auto","movie_date:auto","movie_language:
auto","imdb_url:auto","ds:auto"');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'time_to_li
ve_in_seconds', '3153600000');
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_name" is '电影
名称';
comment on column "public"."holo_dwd_product_movie_basic_info"."dirctor" is '导演';
comment on column "public"."holo_dwd_product_movie_basic_info"."scriptwriter" is '
编剧';
comment on column "public"."holo_dwd_product_movie_basic_info"."area" is '制片地区/
国家';
comment on column "public"."holo_dwd_product_movie_basic_info"."actors" is '主演';
comment on column "public"."holo_dwd_product_movie_basic_info"."type" is '类型';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_length" is '
电影长度';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_date" is '上映
日期';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_language" is
'语言';
comment on column "public"."holo_dwd_product_movie_basic_info"."imdb_url" is 'imdb
号';
COMMIT;
```

4. 新建分区子表数据开发。

   此步骤是一个Hologres SQL模块，用于分区表跑调度。

   i. 登录DataWorks控制台，进入**数据开发**页面，创建Hologres SQL节点，详情请参见Hologres SQL节点。

   ii. 在节点的编辑页面，输入如下语句。

   在Hologres中不支持直接将分区数据直接写入分区父表，因此需要在Hologres中创建对应MaxCompute分区表中分区键值的分区子表，然后将分区数据导入对应的分区子表。分区键值由参

数${bizdate}控制，在调度系统中自动赋值完成周期性调度，调度参数的更多内容，请参见调度参数概述。

> **说明** 导入的分区数据必须和分区键值（本文示例使用的是ds）保持一致，否则会出现报错。

导入分区数据的逻辑场景比较多，下面有两个场景供参考，请您根据实际业务逻辑两者选其中一个。

- 场景一：导入新的分区数据。

```
--创建临时分区子表
BEGIN;
CREATE TABLE IF NOT EXISTS "public".tmp_holo_dwd_product_movie_basic_info_${bizda
te}  (
 "movie_name" text,
 "dirctor" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
);
COMMIT;
--更新外表数据
import foreign schema public_data limit to (dwd_product_movie_basic_info) from se
rver odps_server into public options(if_table_exist 'update');
--等待30s再导入Hologres，以防Hologres meta信息更新缓存慢导致的数据不一致而同步不成功
select pg_sleep(30);
--将MaxCompute数据导入临时分区子表
INSERT INTO "public".tmp_holo_dwd_product_movie_basic_info_${bizdate}
SELECT
    "movie_name",
    "dirctor",
    "scriptwriter",
    "area",
    "actors",
    "type",
    "movie_length",
    "movie_date",
    "movie_language",
    "imdb_url",
    "ds"
FROM "public".dwd_product_movie_basic_info
WHERE ds='${bizdate}';
--导入新的分区数据
BEGIN;
ALTER TABLE tmp_holo_dwd_product_movie_basic_info_${bizdate} RENAME TO holo_dwd_p
roduct_movie_basic_info_${bizdate};
--将临时分区子表绑定在分区父表上
ALTER TABLE holo_dwd_product_movie_basic_info ATTACH PARTITION holo_dwd_product_m
ovie_basic_info_${bizdate} FOR VALUES in ('${bizdate}');
COMMIT;
```

- 场景二：重新对历史分区数据刷新。

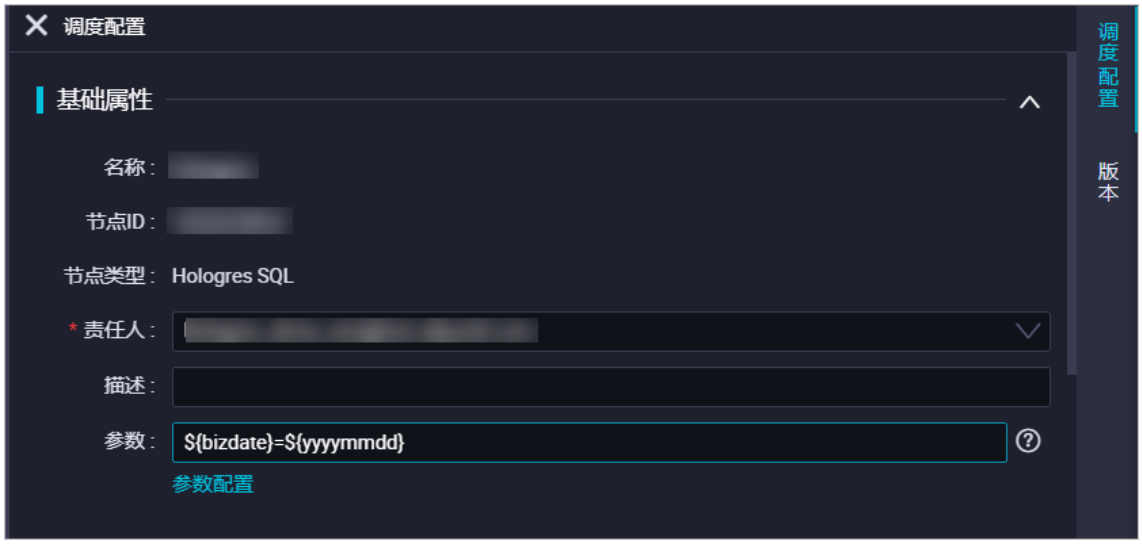
```
--创建临时分区子表
BEGIN;
CREATE TABLE IF NOT EXISTS "public".tmp_holo_dwd_product_movie_basic_info_${bizda
te}  (
 "movie_name" text,
 "dirctor" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
);
COMMIT;
--更新外表数据
import foreign schema public_data limit to (dwd_product_movie_basic_info) from se
rver odps_server into public options(if_table_exist 'update');
--等待30s再导入Hologres，以防Hologres meta信息更新缓存慢导致的数据不一致而同步不成功
select pg_sleep(30);
--将MaxCompute数据导入临时分区子表
INSERT INTO "public".tmp_holo_dwd_product_movie_basic_info_${bizdate}
SELECT
    "movie_name",
    "dirctor",
    "scriptwriter",
    "area",
    "actors",
    "type",
    "movie_length",
    "movie_date",
    "movie_language",
    "imdb_url",
    "ds"
FROM "public".dwd_product_movie_basic_info
WHERE ds='${bizdate}';
重新对历史分区数据刷新
BEGIN;
ALTER TABLE IF EXISTS holo_dwd_product_movie_basic_info DETACH PARTITION holo_dwd
_product_movie_basic_info_${bizdate};
DROP TABLE IF EXISTS holo_dwd_product_movie_basic_info_${bizdate};
ALTER TABLE tmp_holo_dwd_product_movie_basic_info_${bizdate} RENAME TO holo_dwd_p
roduct_movie_basic_info_${bizdate};
--将分区子表绑定在分区父表上
ALTER TABLE holo_dwd_product_movie_basic_info ATTACH PARTITION holo_dwd_product_m
ovie_basic_info_${bizdate} FOR VALUES in ('${bizdate}');
COMMIT;
```

5. 调度配置。

   在**Hologres SQL**编辑页面，单击节点编辑区域右侧的**调度配置**，配置节点的调度属性。

> **说明** 需要更改的参数如下，未提及参数请保持默认值。

- 基础属性



参数	值
参数	${bizdate}=${yyyymmdd}

- 时间属性

参数	值
生成实例方式	*发布后即时生成*
重跑属性	*运行成功后不可重跑，运行失败后可以重跑*
定时调度时间	*00:05*

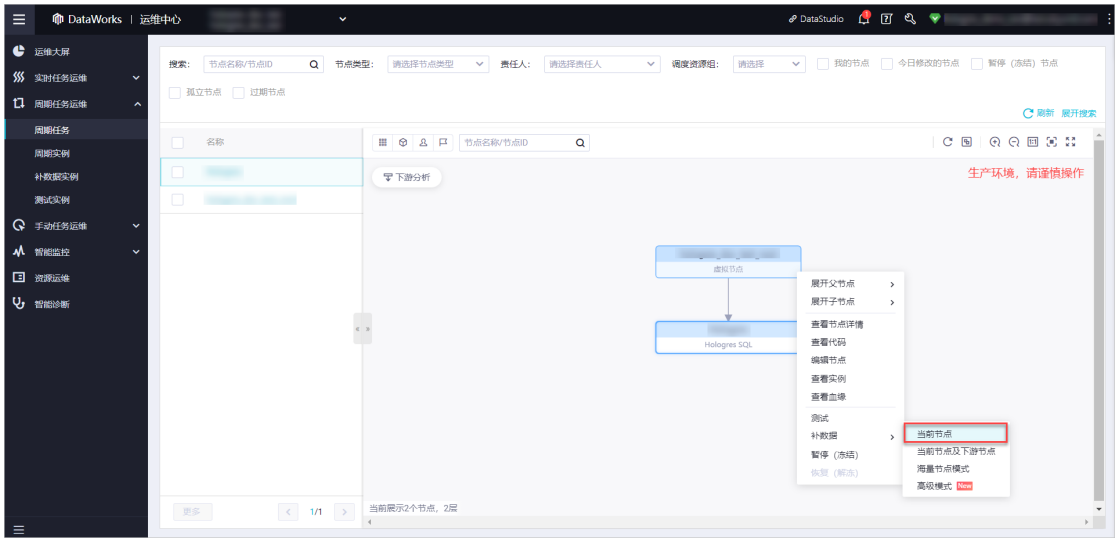
- 调度依赖

  调度依赖为root节点即可（也可以根据业务逻辑选择已有的父节点）。请先将**代码解析**选择为*是*，然后单击**代码解析**，会自动解析出root节点，最后再将**代码解析**选择为*否*。

6. 发布调度。
   i. 在Hologres SQL编辑页面，单击工具栏中的图标，保存节点。
   ii. 单击工具栏中的图标，提交节点。
   iii. 在**提交新版本**对话框中，输入**变更描述**。
   iv. 单击**确认**。
7. 运维中心发布。
   i. 在Hologres SQL编辑页面，单击工具栏中最右侧的**运维**。

ii. 进入**运维中心**页面，单击左侧菜单栏**周期任务运维>周期任务**。

iii. 在**周期任务**页面，右键单击节点，选择**补数据>当前节点**。



iv. 选择左侧菜单栏**周期任务运维 > 补数据实例**，查看正在运行的任务以及任务状态。

8. 查看数据。

任务执行成功之后，将会在Hologres中自动创建对应分区数据的分区子表。

i. 进入**DataStudio数据开发**页面，创建Hologres SQL节点。

ii. 在节点的编辑页面，输入如下语句，进行数据查询。

- 查看分区子表数据。

```
select * from holo_dwd_product_movie_basic_info_20170112;
```

- 查看分区父表总数据。

```
select count (*) from holo_dwd_product_movie_basic_info;
```

3.2.6. HoloStudio

3.2.6.1. 概述

本文为您介绍什么是HoloStudio，以及HoloStudio的核心功能。

HoloStudio是基于交互式分析Hologres构建的一站式OLAP开发平台，深度集成于阿里云智能开发平台DataWorks。

HoloStudio通过可视化方式，为您提供标准化、无门槛的开发服务和一站式构建实时数据仓库服务，以及高效、便捷的管理服务。

HoloStudio在PostgreSQL的标准管理基础上，为您提供更多Hologres功能。

功能

HoloStudio的核心功能如下：

- 外部表

  支持一键创建MaxCompute外部表，同步外部表数据，数据预览和分析达到亚秒级。

- 内部表

  为您提供可视化方式和PostgreSQL语句两种建表模式，满足不同开发层次的需求。

- Hologres开发

  基于DataWorks调度框架，构建MaxCompute离线数据到Hologres索引数据的周期。

- SQL Console

  基于SQL编辑器，为您提供纯SQL语句的开发以及秒级查询体验。

- 终端

  在Web端为您提供psql客户端服务，满足您在终端开发的需求。

3.2.6.2. PG管理

3.2.6.2.1. 创建及管理内部表

HoloStudio的PG管理模块，支持使用可视化方式创建、查看以及删除内部表。本文为您介绍在HoloStudio中基于PG管理的内部表操作。

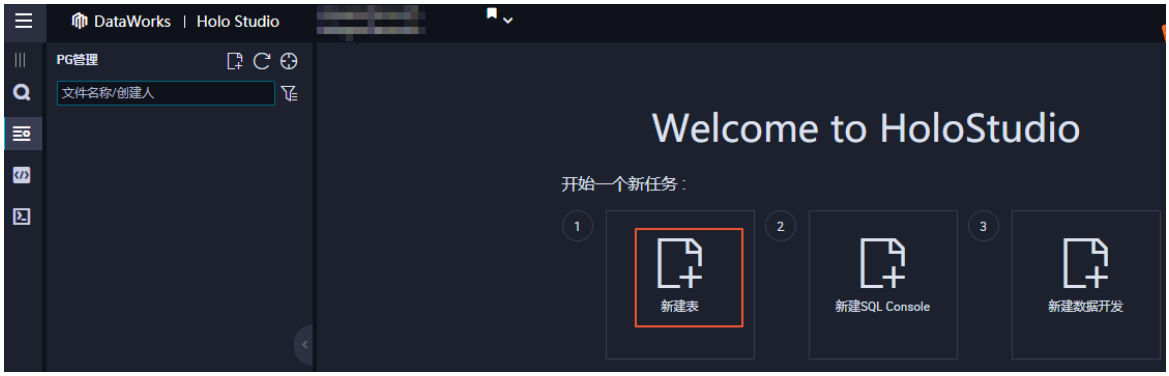
前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

创建内部表

1. 登录Hologres管理控制台，鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
2. 在**实例列表 > Hologres引擎管理**页面，单击**前往DataWorks数仓开发**。
3. 在顶部菜单栏，单击图标，选择目标DataWorks工作空间。
4. 在左侧导航栏，单击图标，进入PG管理界面。
5. 鼠标悬停至图标，单击**表**，创建表。

   您也可以单击HoloStudio首页的**新建表**创建表。



6. 配置编辑表页面的各项参数。

类别	参数	描述
基本属性	**Hologres数据库**	存放新建表的数据库。
	表名	新建表的表名称。
	描述	表的相关描述。
字段	**添加字段**	单击即可添加字段。
	字段名	表中每一列的标识。
	数据类型	字段的属性。
	主键	确定列的唯一标识。
	可空	字段可以为空值。
	数组	有序的元素序列。
	描述	字段的相关描述。
	操作	包括**删除**、**上移**和**下移**。
	存储模式	包括**列存**和**行存**两种存储模式。 默认为**列存**。
	生命周期	如果数据在指定时间内未被修改，则系统将自动删除数据。 默认生命周期为**永久**。
	聚簇索引	排序索引。 索引的类型和列的顺序密切相关。聚簇索引帮助您加速执行索引列的Range和Filter查询。

属性类别	参数	描述
	字典编码列	Hologres支持为指定列的值构建字典映射。 字典编码可以将字符串的比较转换为数字的比较，加速Group By和Filter查询。 默认设置所有**text**列至字典编码列中。
	位图列	Hologres支持在位图列构建比特编码。 位图列可以根据设置的条件快速过滤字段内部的数据。 默认设置所有**text**列至位图列中。
分区表	PARTITION BY LIST	指定分区字段。

7. 单击**提交表**。

   您可以在创建成功的表中进行数据开发。

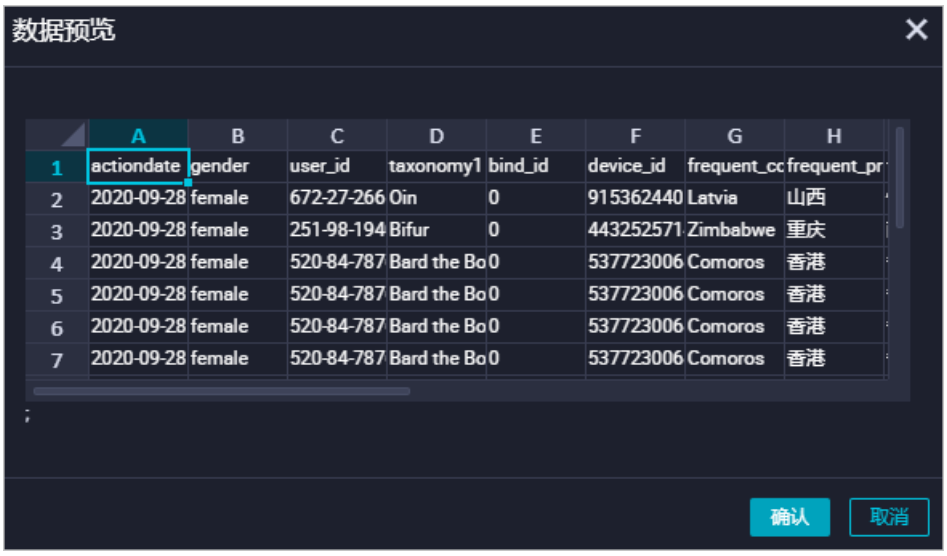
查看表数据

在左侧导航栏单击**PG管理**，双击需要查看的表，显示表编辑页面。

您可以在该页面查看建表语句或预览表数据，如下所示：

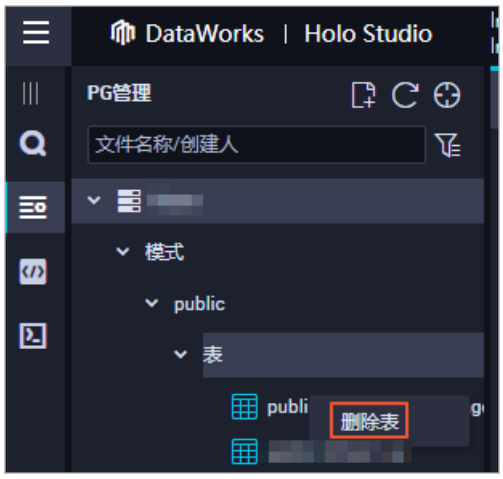
- 单击表编辑页面的**生成DDL语句**，获取建表SQL语句。

- 单击表编辑页面的**数据预览**，预览表数据。



删除内部表

如果您需要删除已创建的内部表，则在PG管理界面，鼠标右键需要删除的表，单击**删除表**即可。



3.2.6.2.2. 创建及管理外部表

外部表是指不存储数据于交互式分析Hologres中的表，仅用于查看外部表的数据。HoloStudio的PG管理模块，支持使用可视化方式创建、查看以及删除外部表。本文为您介绍在HoloStudio中基于PG管理的外部表操作。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

背景信息

Hologres与大数据生态无缝打通，可以直接加速查询外部表数据，也可以将外部表的数据导入至Hologres中处理。Hologres当前仅支持对MaxCompute表进行操作。

使用限制

- Hologres支持跨工作空间读取外部表数据，您当前使用的账号需要拥有其他工作空间的MaxCompute表访问权限，才可以使用HoloStudio一键创建外部表直接读取MaxCompute表数据。
- 当MaxCompute的源头表存在时，您才可以新建外部表并读取外部表数据。
- Hologres中的外部表与MaxCompute的源头表字段一一对应，您可以直接搜索MaxCompute中的表名称，HoloStudio会自动生成对应字段的外部表。

创建外部表

1. 登录Hologres管理控制台，鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
2. 在**实例列表 > Hologres引擎管理**页面，单击**前往DataWorks数仓开发**。
3. 在顶部菜单栏，单击图标，选择目标DataWorks工作空间。
4. 在左侧导航栏，单击图标，进入PG管理界面。
5. 鼠标悬停至图标，单击**外部表**，创建外部表。

6. 配置编辑表页面的各项参数。

类别	参数	描述
基本属性	Hologres数据库	存放新建外部表的数据库。
	表名	新建外部表的表名称。
外部服务	类型	外部表的服务类型。目前仅支持MaxCompute表。
	表	需要映射的MaxCompute源头表。
	字段同步	需要同步的字段。
	分区同步	需要同步的分区。

7. 单击**提交表**。

查看外部表数据

在左侧导航栏单击**PG管理**，双击需要查看的外部表，显示表编辑页面。

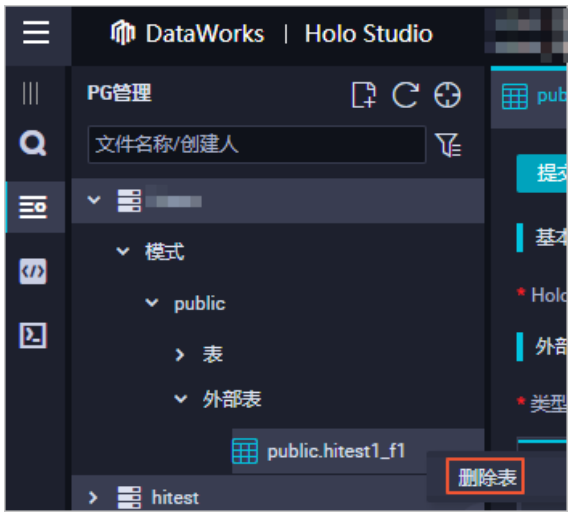
您可以在该页面查看建表语句或预览表数据，如下所示：

- 单击表编辑页面的**生成DDL语句**，获取建表SQL语句。

- 单击表编辑页面的**数据预览**，预览表数据。

删除外部表

如果您需要删除已创建的外部表，则在PG管理界面，鼠标右键需要删除的表，单击**删除表**即可。



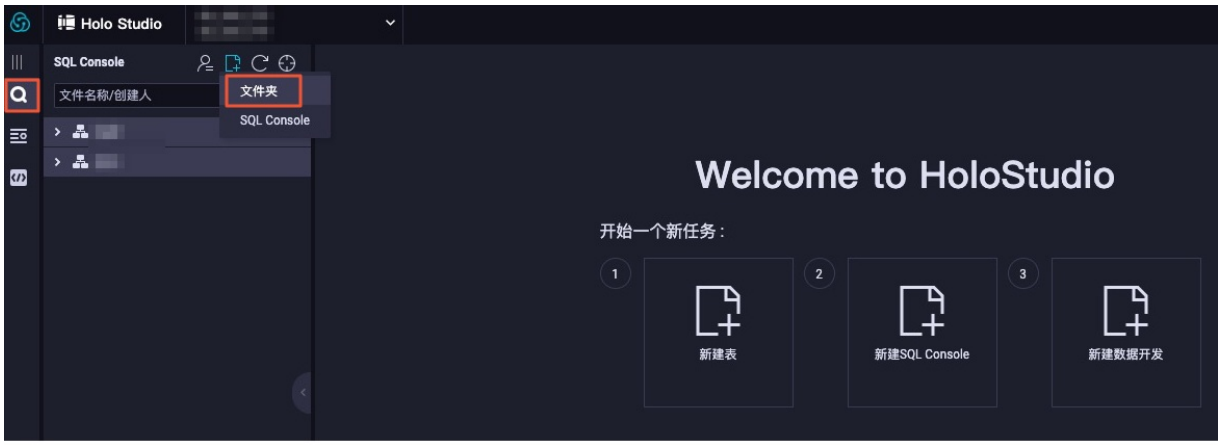
3.2.6.3. SQL Console

SQL Console使用SQL语句开发Hologres，为您提供秒级交互式查询体验。本文为您介绍SQL Console的功能和用法。

文件夹

文件夹模块可以存放新建的临时查询，方便您管理临时查询。

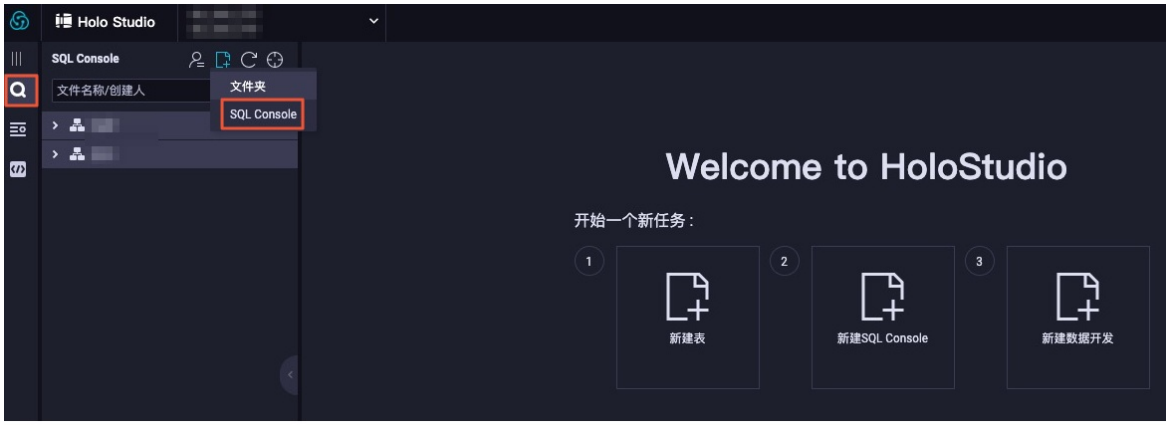
单击左侧菜单栏**SQL Console > 新建 > 文件夹**，创建一个文件夹。您可以在该文件夹里新建临时查询，使用标准的SQL语句完成对表的命令操作。同时您也可以选中文件夹中的某张表，右键单击，即可移动、重命名、删除该表。



SQL Console

SQL Console模块可以生成SQL编辑器，方便您使用标准SQL语句操作。使用之前您需要先创建文件夹，如果已经创建文件夹，则直接进行如下步骤。

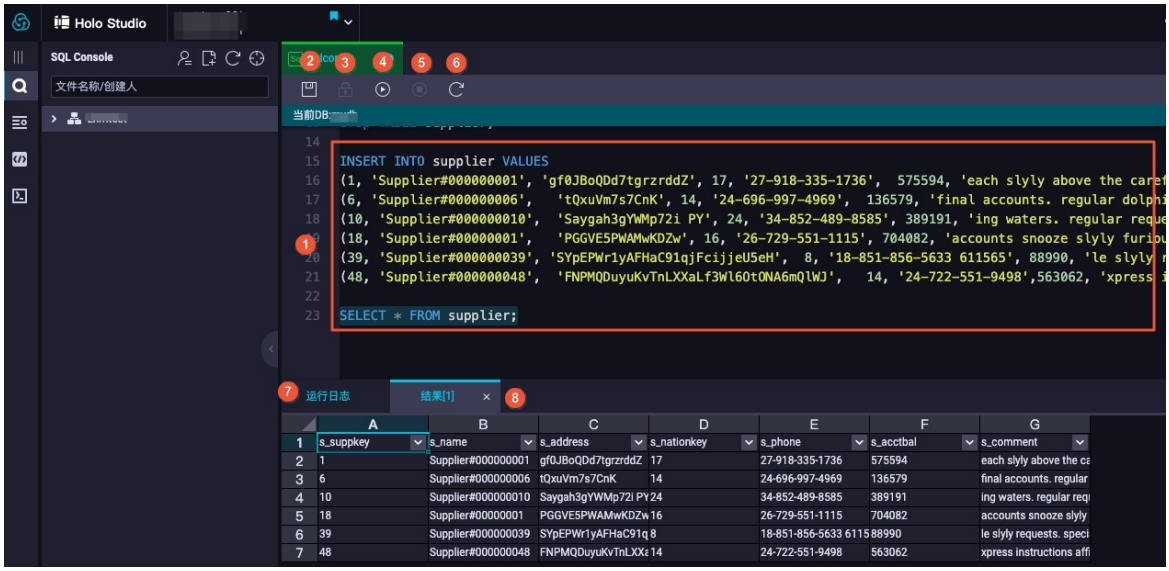
1. 单击左侧菜单栏**SQL Console**，并填写查询的基本信息。



属性	说明
节点名称	临时查询的名称。
目标文件夹	临时查询存放的文件夹位置。
数据库	临时查询的目标数据库。

2. 编写SQL语句，单击上方**运行**，执行查询，完成之后，即可在下方显示框查看到执行结果。示例新建一张表并导入数据，查询表的全过程。

```
CREATE TABLE supplier (
 s_suppkey bigint NOT NULL,
 s_name text NOT NULL,
 s_address text NOT NULL,
 s_nationkey bigint NOT NULL,
 s_phone text NOT NULL,
 s_acctbal bigint NOT NULL,
 s_comment text NOT NULL,
PRIMARY KEY (s_suppkey)
);
INSERT INTO supplier VALUES
(1, 'Supplier#000000001', 'gf0JBoQDd7tgrzrddZ', 17, '27-918-335-1736',  575594, 'each s
lyly above the careful'),
(6, 'Supplier#000000006', 'tQxuVm7s7CnK', 14, '24-696-997-4969',  136579, 'final accoun
ts. regular dolphins use against the furiously ironic decoys. '),
(10, 'Supplier#000000010',  'Saygah3gYWMp72i PY', 24, '34-852-489-8585', 389191, 'ing w
aters. regular requests ar'),
(18, 'Supplier#00000001', 'PGGVE5PWAMwKDZw', 16, '26-729-551-1115', 704082, 'accounts s
nooze slyly furiously bold'),
(39, 'Supplier#000000039', 'SYpEPWr1yAFHaC91qjFcijjeU5eH',  8, '18-851-856-5633 611565'
, 88990, 'le slyly requests. special packages shall are blithely. slyly unusual package
s sleep'),
(48, 'Supplier#000000048',  'FNPMQDuyuKvTnLXXaLf3Wl6OtONA6mQlWJ', 14, '24-722-551-9498'
,563062, 'xpress instructions affix. fluffily even requests boos');
SELECT * FROM supplier;
```



序号	属性	说明
1	SQL编辑框	编写SQL语句。
2	保存	保存SQL编辑框中的全部语句。
3	偷锁编辑	用于非节点责任人编辑节点。
4	运行	运行SQL编辑框内选中的语句或全部语句。单击运行后，开始执行SQL语句，运行结果展现在编辑框底部。 **说明** 不选中则默认运行编辑框中所有语句。
5	停止运行	停止正在运行的SQL语句。
6	重新加载	刷新SQL编辑框的内容。SQL编辑框只保留已保存的语句。
7	运行日志	查看SQL语句的运行情况。例如，SQL语句的运行时长以及报错提示。
8	结果	查看SQL语句运行成功后目标表的内容。

说明 ：对无结果返回的SQL语句，只有运行日志，没有查询结果，例如 `create table` 。

HoloStudio对查询出的表数据可以复制、搜索和隐藏列。

视频操作演示

SQL Console的操作演示视频如下。

3.2.6.4. 数据开发

3.2.6.4.1. 概述

HoloStudio的数据开发模块与DataWorks无缝连接，提供一站式稳定高效的ETL（Extract-Transform-Load）服务，帮助您使用可视化方式周期性调度作业、创建外部表、同步外部表数据以及上传本地文件。

数据开发的核心功能如下：

- **文件夹**

  用于存放并管理数据库中的数据开发节点。详情请参见文件夹。

- **Hologres开发**

  用于周期性调度作业。详情请参见Hologres开发：周期性调度。

- **一键MaxCompute表结构同步**

  支持使用可视化方式批量创建外部表，加速查询MaxCompute的数据。详情请参见一键同步MaxCompute表结构。

- **一键MaxCompute数据同步**

  支持使用可视化方式导入MaxCompute的数据至Hologres。详情请参见一键同步MaxCompute数据。

- **一键本地文件上传**

  用于上传客户端数据至Hologres。详情请参见一键上传本地文件。

3.2.6.4.2. 文件夹

文件夹用于存放数据开发，方便您管理数据库中的开发节点。本文为您介绍如何使用HoloStudio新建文件夹。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

文件夹

1. 登录Hologres管理控制台，鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
2. 在**实例列表 > Hologres引擎管理**页面，单击**前往DataWorks数仓开发**。
3. 在**Holo Studio**页面的左侧导航栏，单击图标。
4. 进入**数据开发**页面，鼠标悬停至顶部菜单栏的图标。
5. 单击**文件夹**。
6. 在**新建文件夹**对话框中，输入**文件夹名称**。
7. 单击**提交**。

   您可以在**Holo Studio**页面，右键单击已创建的文件夹，执行如下操作：

- 选择**Hologres开发**、**一键MaxCompute数据同步**或**一键MaxCompute表结构同步**，新建作业节点。
- 选择**重命名**，修改当前文件夹的名称。
- 选择**删除**，删除当前文件夹。

> **说明** 删除文件夹之前，您需要先删除文件夹中的所有文件。

3.2.6.4.3. Hologres开发：周期性调度

HoloStudio与DataWorks无缝连通，您可以通过HoloStudio将MaxCompute数据导入Hologres，并基于DataWorks的底层能力，前往DataWorks进行定时调度，实现周期性导入数据至Hologres。本文为您介绍如何将MaxCompute源表数据导入Hologres进行周期性调度。

背景信息

- MaxCompute与Hologres的分区无强映射关系，MaxCompute的分区字段映射为Hologres的普通字段。因此，您可以将MaxCompute的分区表或非分区表导入Hologres的分区或非分区表，可以根据实际业务情况选择是否需要分区。
- 前往DataWorks调度会产生一定的调度费用，详细收费情况，请参见DataWorks资源组概述。
- 如果需要实现写入更新，您需要使用INSERT ON CONFLICT(UPSERT)语法。
- 如果MaxCompute数据会定期更新，建议您在Hologres导入数据时，使用IMPORT FOREIGN SCHEMA语句来更新外部表，以便于及时获取MaxCompute的元数据。
- 外表名和内表名必须不一样，否则会报错。如使用import foreign语法创建外表，则外表名和MaxCompute表名必须保持一致。

操作步骤

1. 准备MaxCompute表数据。

   准备一张MaxCompute数据源表，您可以参考创建表进行建表并导入数据。示例选用MaxCompute公共数据集**public_data**的分区表dwd_ product_movie_basic_info表，您可以参照使用公开数据集描述，登录并查询数据集。其建表DDL语句如下：

   ```
   --MaxCompute分区表DDL
   CREATE TABLE IF NOT EXISTS public_data.dwd_product_movie_basic_info(
     movie_name STRING COMMENT '电影名称',
     dirctor STRING COMMENT '导演',
     scriptwriter STRING COMMENT '编剧',
     area STRING COMMENT '制片地区/国家',
     actors STRING COMMENT '主演',
     `type` STRING COMMENT '类型',
     movie_length STRING COMMENT '电影长度',
     movie_date STRING COMMENT '上映日期',
     movie_language STRING COMMENT '语言',
     imdb_url STRING COMMENT 'imdb号'
   )
   PARTITIONED BY (ds STRING) STORED AS ALIORC;
   --查看分区表的某个分区数据
   SELECT * FROM public_data.dwd_product_movie_basic_info WHERE ds = '20170112';
   ```

   查询数据显示如下图所示。

movie_name	dirctor	scriptwriter	area	actors	type	movie_length	movie_date	movie_language	imdb_url	ds
	Abbas.Kiarostami	\N	伊朗	\N	\N	\N	1995-09-16	\N	\N	20170112
	Rolfe.Kanefsky	\N	美国	Scott.Coppola	喜剧	\N	\N	\N	\N	20170112
!(ai-ou]	堤ユキヒコ（堤幸彦）	斉藤猛	日本	柴田恭兵/锦织一清/大	剧情/犯罪	108	1991-01-26	日语	\N	20170112
"ABCStage67":TheCan	John.Robins	奥斯卡·王尔德/Burt.Sh	英国	迈克尔·雷德格瑞夫/Da	剧情	\N	1966-11-02	\N	tt0060208	20170112
"Ab50Eisbeinwirdesscl	亚历山大·克鲁格	\N	德国	\N	\N	15分钟	\N	德语	\N	20170112
"AmericanMasters"Bin	Robert.Trachtenberg	\N	美国	\N	纪录片/传记/历史	\N	\N	英语	tt4062918	20170112

2. 新建外部表。

   进入HoloStudio，在SQL Console中新建一张外部表，用于映射MaxCompute源表数据。外部表的字段顺序和字段类型需要和MaxCompute一一对应。示例新建外部表的SQL语句如下：

```
BEGIN;
CREATE FOREIGN TABLE public.dwd_product_movie_basic_info (
    "movie_name" text,
    "dirctor" text,
    "scriptwriter" text,
    "area" text,
    "actors" text,
    "type" text,
    "movie_length" text,
    "movie_date" text,
    "movie_language" text,
    "imdb_url" text,
    "ds" text
)
SERVER odps_server
OPTIONS (project_name 'public_data', table_name 'dwd_product_movie_basic_info');
comment on column public.dwd_product_movie_basic_info."movie_name" is '电影名称';
comment on column public.dwd_product_movie_basic_info."dirctor" is '导演';
comment on column public.dwd_product_movie_basic_info."scriptwriter" is '编剧';
comment on column public.dwd_product_movie_basic_info."area" is '制片地区/国家';
comment on column public.dwd_product_movie_basic_info."actors" is '主演';
comment on column public.dwd_product_movie_basic_info."type" is '类型';
comment on column public.dwd_product_movie_basic_info."movie_length" is '电影长度';
comment on column public.dwd_product_movie_basic_info."movie_date" is '上映日期';
comment on column public.dwd_product_movie_basic_info."movie_language" is '语言';
comment on column public.dwd_product_movie_basic_info."imdb_url" is 'imdb号';
COMMIT;
```

   OPTIONS的连接参数说明如下表所示。

参数	描述
project_name	MaxCompute的项目名称。
table_name	MaxCompute的表名称。

3. 新建存储表。

   在HoloStudio中新建一张用于接收并存储数据的真实存储表。由于本次示例是将MaxCompute分区表导入Hologres分区表，因此需要在Hologres中创建一张分区表。

   该示例建表DDL仅是简单示例，实际建表DDL请根据实际业务需要创建，并给表设置合理的索引，以达到更优的查询性能，DDL示例如下：

```
BEGIN;
CREATE TABLE "public"."holo_dwd_product_movie_basic_info" (
 "movie_name" text,
 "dirctor" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
)
PARTITION BY LIST (ds);
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'orientation',
'column');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'bitmap_columns
', '"movie_name","dirctor","scriptwriter","area","actors","type","movie_length","movie_
date","movie_language","imdb_url","ds"');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'dictionary_enc
oding_columns', '"movie_name:auto","dirctor:auto","scriptwriter:auto","area:auto","acto
rs:auto","type:auto","movie_length:auto","movie_date:auto","movie_language:auto","imdb_
url:auto","ds:auto"');
CALL SET_TABLE_PROPERTY('"public"."holo_dwd_product_movie_basic_info"', 'time_to_live_i
n_seconds', '3153600000');
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_name" is '电影名称
';
comment on column "public"."holo_dwd_product_movie_basic_info"."dirctor" is '导演';
comment on column "public"."holo_dwd_product_movie_basic_info"."scriptwriter" is '编剧';
comment on column "public"."holo_dwd_product_movie_basic_info"."area" is '制片地区/国家';
comment on column "public"."holo_dwd_product_movie_basic_info"."actors" is '主演';
comment on column "public"."holo_dwd_product_movie_basic_info"."type" is '类型';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_length" is '电影长
度';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_date" is '上映日期
';
comment on column "public"."holo_dwd_product_movie_basic_info"."movie_language" is '语言
';
comment on column "public"."holo_dwd_product_movie_basic_info"."imdb_url" is 'imdb号';
COMMIT;
```

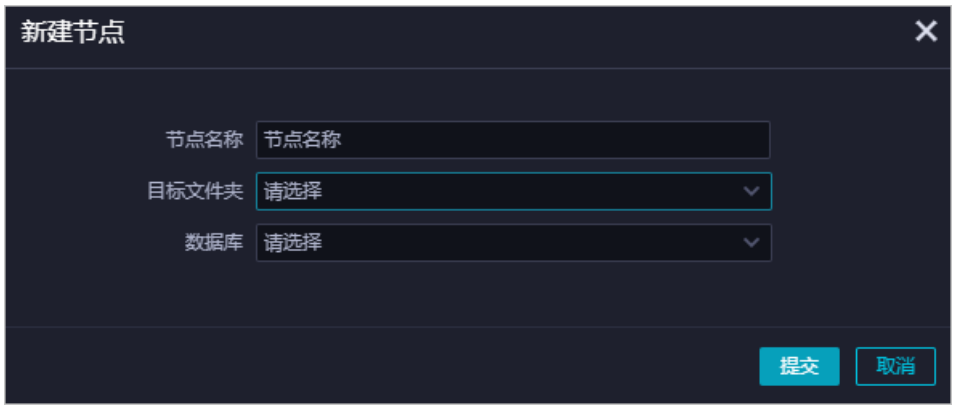
4. 新建分区子表的数据开发。

   由于本示例是将MaxCompute分区表数据导入Hologres分区表，因此需要在Hologres中创建对应的分区键值的分区子表。然后将分区数据导入对应的分区子表。这里通过{bizdate}参数控制分区键值，在调度系统中自动赋值完成周期性调度。关于参数的配置，请参见DataWorks文档DataWorks调度配置。其中，insert的分区数据必须和分区键（本文示例为ds）的值保持一致，否则会报错。

   i. 在HoloStudio中，单击左侧导航栏的图标，进入数据开发页面。

   ii. 鼠标悬停至图标，单击**Hologres开发**。

iii. 配置**新建节点**对话框的**节点名称**、**目标文件夹**及**数据库**参数。



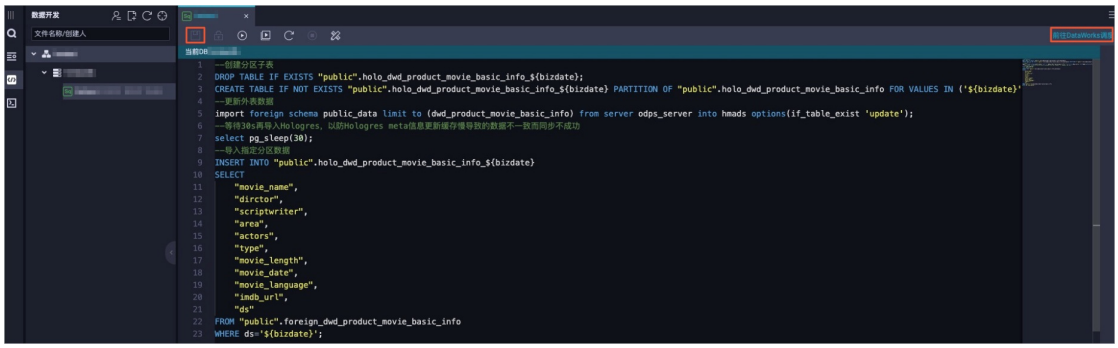
iv. 单击**提交**，成功创建Hologres开发节点。

v. 在新建的Hologres开发节点的编辑界面，输入如下创建分区表的数据开发语句。

```
--创建分区子表
DROP TABLE IF EXISTS "public".holo_dwd_product_movie_basic_info_${bizdate};
CREATE TABLE IF NOT EXISTS "public".holo_dwd_product_movie_basic_info_${bizdate} PA
RTITION OF "public".holo_dwd_product_movie_basic_info FOR VALUES IN ('${bizdate}');
--更新外表数据
import foreign schema public_data limit to (dwd_product_movie_basic_info) from serv
er odps_server into hmads options(if_table_exist 'update');
--等待30s再导入Hologres，以防Hologres meta信息更新缓存慢导致的数据不一致而同步不成功
select pg_sleep(30);
--导入指定分区数据
INSERT INTO "public".holo_dwd_product_movie_basic_info_${bizdate}
SELECT
    "movie_name",
    "dirctor",
    "scriptwriter",
    "area",
    "actors",
    "type",
    "movie_length",
    "movie_date",
    "movie_language",
    "imdb_url",
    "ds"
FROM "public".foreign_dwd_product_movie_basic_info
WHERE ds='${bizdate}';
```

vi. 单击顶部菜单栏左侧的  ，保存Hologres开发。

vii. 单击顶部菜单栏右侧的**前往DataWorks调度**，进入调度页面。



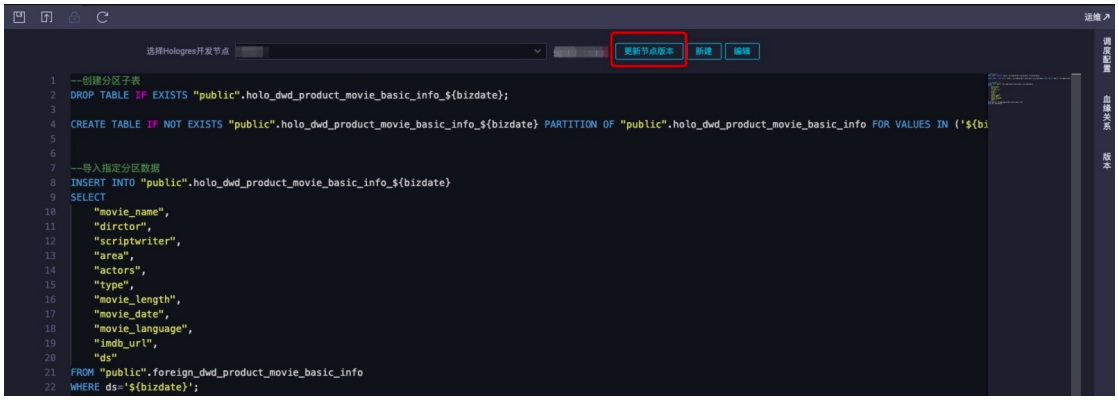
5. 新建分区表的调度作业。

i. 在DataWorks中新建Hologres节点。

单击**前往DataWorks调度**后，页面自动跳转至DataWorks的**新建节点**。您需要配置**新建节点**对话框的**节点类型**、**节点名称**及**目标文件夹**，创建Hologres节点。**节点类型**选择**Hologres开发**。

如果是重新执行已有作业，则可以不需要重新创建节点。

ii. 在新建的节点页面，单击**更新节点版本**，将分区表的信息同步至该节点。



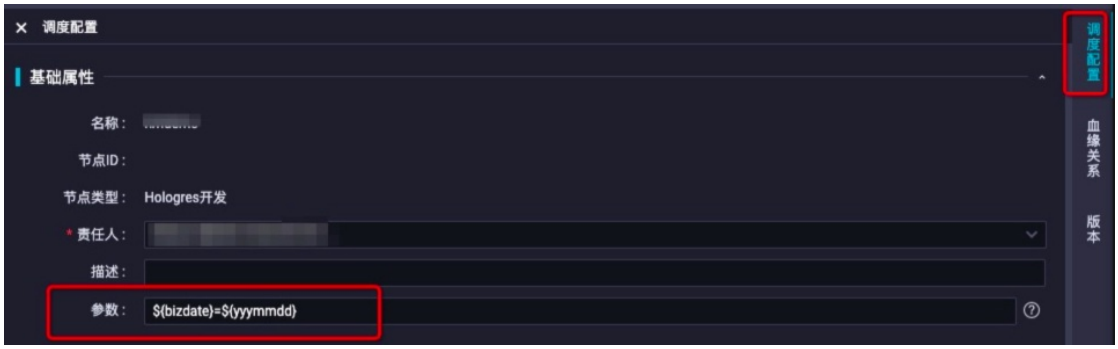
6. 配置调度信息。

在新建的节点页面，单击右侧导航栏的**调度配置**，配置调度参数。详情请参见调度参数概述。本次示例选择的是日调度，因此将会每天自动生成一张分区子表，并导入对应的数据，以此来实现增量数据周期性调度。更多的调度逻辑（包括调度周期、调度参数、调度上下游等）请参见DataWorks调度配置。

具体配置如下所示：

i. 配置基础属性。

在**基础属性**区域，将**参数**赋值为时间节点。



ii. 配置时间属性。

在**时间属性**区域，配置**时间属性**为**正常调度**。其余参数请根据业务实际情况配置，详情请参见时间属性配置说明。

iii. 配置调度依赖。

在**调度依赖**区域，您需要配置调度依赖为root节点，操作如下：

a. 将**自动解析**配置为**是**。

b. 单击**使用工作空间根节点**，自动解析出root节点。

c. 再将**自动解析**配置为**否**，详情请参见配置同周期调度依赖。

您也可以根据业务逻辑选择已有的父节点。

7. 发布调度。

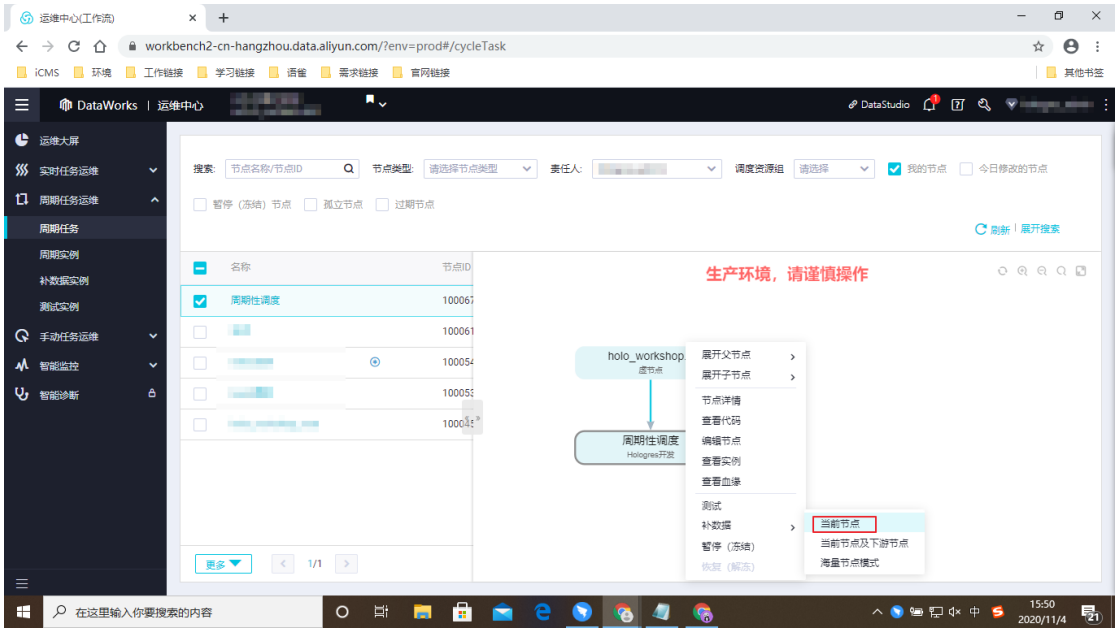
i. 配置完调度参数，在新建节点页面的顶部菜单栏左侧，单击图标，保存调度配置。

ii. 在新建节点页面的顶部菜单栏左侧，单击图标，提交调度信息。

iii. 发布包提交成功后，单击**任务发布**页面顶部菜单栏右侧的**运维中心**，进入运维中心。

iv. 在左侧导航栏，单击**周期任务运维 > 周期任务**，勾选并单击目标任务名称。

v. 在右侧编辑界面，鼠标右键单击目标任务节点，选择**补数据 > 当前节点**。



vi. 根据业务实际情况，配置**补数据**对话框的各项参数。完成调度任务的发布。

8. 任务执行成功后，您可以通过如下方式查询数据。

- 您可以返回Hologres中，通过标准SQL查看父表或分区表的数据，示例如下。

```
--查看分区子表数据
select * from holo_dwd_product_movie_basic_info_20170112;
--查看分区父表总数据
select count (*) from holo_dwd_product_movie_basic_info;
```

- 返回HoloStudio。单击新建节点的图标，刷新当前页面。您可以在**PG管理**模块找到目标分区表，查看调度成功的分区表数据。双击目标表名称，在表编辑页面单击**数据预览**，即可看到已成功导入的数据。

3.2.6.4.4. 一键同步MaxCompute表结构

本文为您介绍如何使用HoloStudio批量创建外部表。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

背景信息

Hologres与MaxCompute在底层无缝连接，支持使用新建外部表的方式加速查询MaxCompute的数据，详情请参见通过创建外部表加速查询MaxCompute数据。

HoloStudio支持**一键MaxCompute表结构同步**功能，您可以使用可视化方式批量创建外部表。

您也可以使用 `IMPORT FOREIGN SCHEMA` 语句批量创建外部表。也可以将该命令语句在数据开发中执行，并前往调度，实现MaxCompute表新增后Hologres外表也自动新增，详情见Hologres开发：周期性调度。

> **说明** Hologres只能加速查询MaxCompute的内表，不能加速查询MaxCompute的外表和View。

操作步骤

1. 登录阿里云官网。
2. 新建**一键MaxCompute表结构同步**节点。
   i. 进入Hologres管理控制台。
   ii. 鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
   iii. 在**实例列表**的**Hologres引擎管理**页面，单击**前往DataWorks-HoloStudio开发**。
   iv. 在**Holo Studio**界面的左侧导航栏，单击图标。
   v. 进入**数据开发**页面，鼠标悬停至顶部菜单栏的图标。
   vi. 单击**一键MaxCompute表结构同步**。
   vii. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**和**数据库**。
   viii. 单击**提交**。
3. 在编辑节点页面，配置各项参数。

参数	描述
目标库	Hologres的数据库名称。
目标Schema	当前数据库的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public**。如果有新建的Schema，您也可以选择新建的Schema。
远程服务类型	默认为**odps**。 目前仅支持MaxCompute。
远程服务器	您可以直接调用Hologres底层已创建的名为**odps_server**的远程服务器。详细原理请参见Postgres FDW。
远程库	MaxCompute的项目名称。
表名规则	您可以使用正则表达式选择需要同步的表名称，默认同步远程数据库中的所有表。更多筛选外部表的规则请参见IMPORT FOREIGN SCHEMA。
正则预览	查看正则表达式的运行结果。

同步外部表的规则如下：

- 如果同步外部表时存在名称相同的表，则忽略当前表。
- 如果同步外部表时，存在外部表不支持的数据类型，则系统会报错请在正则表达式中排除当前表再重新导入。

4. 在新建节点的编辑页面，单击图标，保存配置信息。
5. 在新建节点的编辑页面，单击图标，批量创建外部表结构。
6. 在**PG管理**页面，查看批量创建的外部表并查询表数据，详情请参见创建及管理外部表。

3.2.6.4.5. 一键同步MaxCompute数据

本文为您介绍如何使用HoloStudio导入MaxCompute数据至Hologres。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

背景信息

HoloStudio支持**一键MaxCompute数据同步**功能，您可以使用可视化方式导入MaxCompute表数据并进行查询。该方式比创建外部表直接查询数据的性能更好。

您也可以使用SQL语句导入MaxCompute数据，详情请参见使用SQL导入MaxCompute的数据至Hologres。

操作步骤

1. 登录阿里云官网。
2. 新建**一键MaxCompute数据同步**节点。
   i. 进入Hologres管理控制台。
   ii. 鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
   iii. 在**实例列表 > Hologres引擎管理**页面，单击**前往DataWorks-HoloStudio开发**。
   iv. 在**Holo Studio**页面的左侧导航栏，单击图标。
   v. 进入**数据开发**页面，鼠标悬停至顶部菜单栏的图标。
   vi. 单击**一键MaxCompute数据同步**。
   vii. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**和**数据库**。
   viii. 单击**提交**。
3. 在编辑节点页面，配置各项参数。

参数	描述

参数	描述
外部表来源	○ **已有外部表**：表示在Hologres中已经建立MaxCompute数据映射的外部表。 ○ **新建外部表**：表示外部表在Hologres中未建立MaxCompute数据映射。如果您需要导入MaxCompute表数据，请选中**新建外部表**。
外部表表名字	已创建的外部表表名称。 Hologres中创建的外部表用于映射MaxCompute数据，需要与导入数据的MaxCompute表结构一一对应。
目标库	Hologres的数据库名称。
目标Schema	当前数据库的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public**。如果有新建的Schema，您也可以选择新建的Schema。
目标表名	需要导入数据的表名称。
目标表描述	目标表的信息描述。
同步字段	需要导入的MaxCompute表字段。 您可以选择导入部分或全部字段。
分区配置	需要导入的分区字段。 **一键MaxCompute数据同步**功能仅支持导入一级分区。如果您需要导入多级分区，请在**SQL Console**中使用SQL语句实现，详情请参见SQL Console。
索引配置	您可以为目标表创建索引，详情请参见设置表属性。
SQL Script	自动解析当前可视化操作对应的SQL语句。

4. 在新建节点的编辑页面，单击图标，保存配置信息。
5. 在新建节点的编辑页面，单击图标，导入MaxCompute表数据。
6. 在**PG管理**页面，查看导入的外部表数据，详情请参见创建及管理外部表或SQL Console。

3.2.6.4.6. 一键上传本地文件

本文为您介绍如何使用HoloStudio导入客户端数据至Hologres。

前提条件

- 阿里云账号注册，详情请参见。

- 实名认证，详情请参见或。
- 开通交互式分析Hologres并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。

背景信息

HoloStudio支持**一键本地上传文件**功能，您可以使用可视化方式导入客户端数据。

您也可以使用 `COPY` 命令导入客户端数据，详情请参见使用COPY命令导入或导出本地数据。

操作步骤

1. 登录Hologres管理控制台。
2. 鼠标悬停至顶部菜单栏左侧的图标，选择目标地域。
3. 在**实例列表 > Hologres引擎管理**页面，单击**前往DataWorks-HoloStudio开发**。
4. 新建内部表。

   使用**SQL Console**新建一张用于接收客户端数据的内部表，详情请参见SQL Console。示例SQL语句如下。

   ```
   BEGIN;
   CREATE TABLE if not EXISTS holo_bank (
    age int8,
    job text,
    marital text,
    education text,
    card text,
    housing text,
    loan text,
    contact text,
    month text,
    day_of_week text,
    duration text,
    campaign int8,
    pdays float8,
    previous float8,
    poutcome text,
    emp_var_rate float8,
    cons_price_idx float8,
    cons_conf_idx float8,
    euribor3m float8,
    nr_employed float8,
    y int8
   );
   COMMIT;
   ```

   您也可以使用**PG管理**新建内部表，详情请参见创建及管理内部表。
5. 新建**一键本地上传文件**。

   i. 在Holo Studio界面的左侧导航栏，单击图标。

   ii. 进入**数据开发**页面，鼠标悬停至顶部菜单栏的图标。

   iii. 单击**一键本地上传文件**。

iv. 在**一键本地文件上传**对话框中，配置各项参数。

参数	描述
目标库	Hologres的数据库名称。
目标schema	当前数据库的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public**。如果有新建的Schema，您也可以选择新建的Schema。
选择要导入的数据表	需要导入数据的表。

v. 单击**下一步**。

vi. 在**一键本地文件上传**对话框中，配置各项参数。

参数	描述
选择文件	需要上传的本地文件。 仅支持上传TXT、CSV和LOG类型的文件。
选择分隔符	■ 逗号 ■ Tab ■ 分号 ■ 空格 ■ | ■ # ■ & 您也可以自定义分隔符。
原始字符集	■ GBK ■ UTF-8 ■ CP936 ■ ISO-8859
首行为标题	勾选则设置首行数据为标题。

vii. 单击**提交**。

6. 在**SQL Console**或**PG管理**页面，查看已上传的数据。详情请参见SQL Console或创建及管理内部表。

3.3. HoloWeb

3.3.1. HoloWeb简介

Holoweb是基于Hologres引擎的可视化数据库管理和开发一站式平台，灵活适用于数据库管理、数据库接入、数据开发、数据分析、性能分析和诊断等用户场景。本文为您介绍什么是HoloWeb以及HoloWeb提供的开发和运维功能。

背景信息

Hologres是为大数据设计的一站式实时数仓，支持数据实时写入、PB级数据高并发低延时的分析处理和超高QPS点查。与MaxCompute无缝打通，支持数据加速查询，兼容PostgreSQL生态，可以使用最熟悉的BI工具对海量数据进行自助的多维分析透视和业务探索，满足数仓分析、服务一体化的需求。

HoloWeb实现对Hologres中数据库的可视化管理和开发功能，相比PostgreSQL生态的开发工具，HoloWeb具有如下优势。

- 操作简便

  Holoweb提供Hologres引擎功能的可视化界面，无需写SQL命令就能进行大数据开发，操作简便，可以极大降低大数据开发学习成本。

- 与引擎高度适配

  Hologres引擎提供强大的数仓功能，包括外部表数据一键导入、本地文件一键上传、执行计划可视化调优、慢Query日志等。HoloWeb与引擎功能高度适配，提供可视化界面，实现Hologres引擎的大部分功能。

- 一站式开发与运维

  可以通过HoloWeb进行一站式开发与运维，包括IP白名单、资源隔离等功能，无需单独构建开发界面，一站式满足企业级需求。

HoloWeb由以下模块组成。

- 元数据管理

  元数据管理模块提供对Hologres引擎对象（包括实例、数据库、schema、表、视图等）的管理功能，您可以在元数据管理页面连接Hologres实例，并在实例内进行对象创建，包括可视化建内表、外表、一键数据导入等，也可以对实例对象进行管理、编辑等。

- SQL编辑器

  HoloWeb的SQL编辑器提供标准SQL开发界面，您可以通过标准的PostgreSQL语句进行开发，同时还能查看SQL的执行计划和运行分析等，可以快速对SQL进行诊断优化，支持交互式分析，Ad Hoc分析，不支持执行时间超过60分钟的长运行SQL。

- 诊断与优化

  HoloWeb诊断与优化模块提供实例级别的运维管控能力，可以在该模块查看慢Query日志、活跃Query、连接管理等，并结合管理控制台监控指标，快速对实例异常进行诊断，提升自助化运维能力。

- 数据方案

  HoloWeb数据方案模块提供数据接入能力，您可以在该模块进行本地文件上传、外表数据导入操作，无需写SQL命令，直接可视化操作，降低学习成本和操作难度。

- 安全中心

  HoloWeb安全中心模块提供一站式安全管控能力，包括用户授权、IP白名单、资源组等功能，可以通过HoloWeb直接进行实例的安全管控，满足一站式运维和管控需求。

3.3.2. 元数据管理

3.3.2.1. 登录实例

HoloWeb是Hologres自研的处理数据的开发工具。本文为您介绍如何使用HoloWeb登录至Hologres的实例连接。

前提条件

- 已完成阿里云账号注册，详情请参见阿里云账号注册流程。
- 已完成实名认证，详情请参见个人实名认证或企业实名认证和个体工商户认证。
- 已开通Hologres，详情请参见购买Hologres。

登录实例

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 您可以使用如下方式登录实例：
   - **通过页面弹框进行登录**

     如果您当前没有已登录的实例，当您进入HoloWeb时，会有**登录实例**的弹框，您可以在页面进行配置，单击**确认**登录实例。

参数	说明

参数	说明
实例名	您可以从下拉列表中选择当前已经存在的未登录实例。 **说明** ■ 未登录实例中会展示用户当前区域所在主账号下具有访问权限的所有实例。 ■ 您开通的Hologres引擎实例会默认生成一个图标的VPC网络类型的未登录实例展示在列表中，该实例不支持编辑详细信息和删除。
选择数据库	您可以选择如下两种方式： ■ 登录已有数据库：从数据库名称下拉列表中选择目标数据库。 ■ 新建数据库：您可以根据需求输入数据库名称，并为该数据库配置权限策略。
数据库名称	■ 当您登录已有数据库时，从下拉列表中选择目标数据库。 ■ 当您新建数据库时，您可以根据需求输入数据库名称。
权限策略	当选择数据库项选择为新建数据库时，会出现该配置项。您可以根据业务需求为数据库配置对应权限。更多关于权限策略的说明，请参见： ■ SPM ■ SLPM ■ 专家权限模型

- **通过未登录实例进行登录**

  您可以在未登录实例列表中，鼠标右键单击目标实例，选择**登录实例**。进行参数配置并单击**确认**登录实例。更多关于参数的配置，请参见登录实例参数说明。

- **通过新增实例进行登录**

  您可以通过单击**元数据管理**页签的**新增实例**连接其他区域的实例。进行参数配置时将配置项新增后登录选择**是**登录实例。更多关于新增实例的配置，请参见新增实例。

5. 完成登录的实例会展示在**已登录实例**列表中。您可以右键单击目标实例对其进行管理操作，更多关于实例管理的内容请参见管理实例。

3.3.2.2. 新增实例

HoloWeb是Hologres自研的处理数据的开发工具。本文为您介绍如何使用HoloWeb新增Hologres的实例连接。

前提条件

- 已完成阿里云账号注册，详情请参见阿里云账号注册流程。
- 已完成实名认证，详情请参见个人实名认证或企业实名认证和个体工商户认证。
- 已开通Hologres，详情请参见购买Hologres。

新增实例

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 单击**元数据管理**页签的**新增实例**。
5. 配置**新增实例**对话框的参数，单击**OK**。

参数	描述	是否必选
网络类型	○ **公网**：支持华东2（上海）、华南1（深圳）、华北2（北京）、华东1（杭州）、亚太东南1（新加坡）、中国（香港）、亚太东南3（吉隆坡）及美国西部1（硅谷）等地域。 图标的为公网类型的实例。 ○ **VPC**：仅支持配置为HoloWeb所登录的地域。 图标的为VPC类型的实例，该实例不支持编辑详细信息和删除。	否
实例名称	选择当前账号已创建的实例。	否

参数	描述	是否必选
名称	选择**实例名称**后，**连接名称**默认显示为所选实例的名称。您也可以重新自定义连接名称。	是
描述	连接的描述信息。	否
主机	Hologres实例的网络域名。 您可以进入Hologres管理控制台的实例详情页，从**实例配置**获取主机。 如果您配置了**实例名称**，则系统自动为您匹配该实例的**主机**。您也可以选择手动输入主机地址。	是
端口	Hologres实例的网络端口。 您可以进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。 如果您配置了**实例名称**，则系统自动为您匹配该实例的**端口**。您也可以选择手动输入端口地址。	是
登录方式	○ **当前账户免密登录**：无需输入账号及密码，直接使用当前账户登录。 ○ **账户密码登录**：您可以输入自己或者其他账户的账号及密码，并登录。	是
账号	登录方式配置为*账户密码登录*时，需要配置该参数。 当前账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。	否
密码	登录方式配置为*账户密码登录*时，需要配置该参数。 当前账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。	否
测试连通性	检测连接是否成功： ○ 显示**测试通过**表示连接成功。 ○ 显示**测试不通过**表示连接失败。	否
新增后登录	您可以选择是否登录该实例。 ○ **是**：该实例会登录并展示在左侧已登录实例列表中。 ○ **否**：该实例会展示在左侧未登录实例列表中。	是

6. 连接成功后，单击**实例管理**右侧的图标，刷新当前实例列表。单击**已登录实例**，查看相关信息。

您可以右键单击目标实例对其进行管理操作，更多关于实例管理的内容请参见管理实例。

3.3.2.3. 管理实例

本文为您介绍如何使用在HoloWeb中对已登录和未登录的实例进行编辑、删除等管理操作。

背景信息

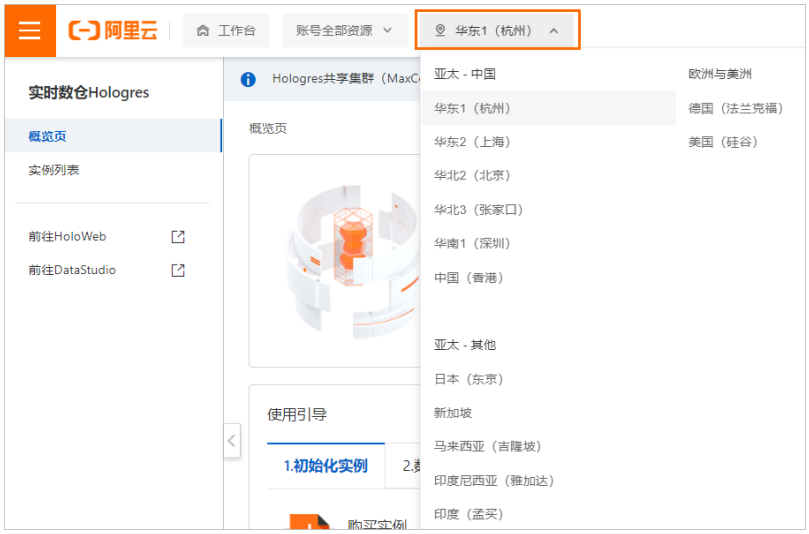
您可以对已登录实例和未登录实例进行相关管理操作，具体涉及的内容如下：

实例类型	管理操作
管理未登录实例	您可以对未登录实例进行如下操作： • 登录实例 • 编辑实例 • 删除实例
管理已登录实例	您可以对未登录实例进行如下操作： • 刷新 • 编辑实例 • 用户管理 • DB授权 • 切换登录数据库 • 新建数据库 • 退出登录 • 删除实例

管理未登录实例

您可以对未登录实例进行登录、编辑和删除实例等操作。

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在**元数据管理**页签下，右键单击未登录目标实例，进行管理操作。

> **说明**
> - 未登录实例中会展示用户当前区域所在主账号下具有访问权限的所有实例。
> - 您开通的Hologres引擎实例会默认生成一个图标的VPC网络类型的未登录实例展示在列表中，该实例不支持编辑详细信息和删除。

您可以进行如下操作：

- **登录实例**

  您可以鼠标右键单击目标实例，选择**登录实例**。更多关于登录实例的操作，请参见登录实例。

- **编辑实例**

  您可以鼠标右键单击目标实例，选择**编辑实例**。更多关于编辑实例参数信息的操作，请参见编辑实例参数说明。

  > **说明**
  > - 图标的为公网类型的实例，支持进行所有参数的编辑修改。
  > - 图标的VPC类型的实例，仅支持对描述进行修改。

- **删除实例**

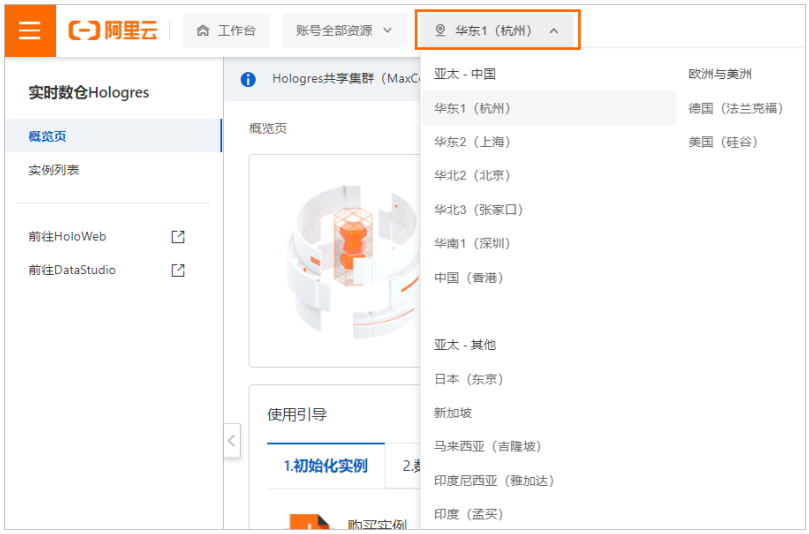
  您可以鼠标右键单击目标实例，选择**删除实例**。确认实例信息后，单击**确认**完成删除实例的操作。

  > **说明** 当前仅支持删除图标的为公网类型的实例，图标的VPC类型的实例不支持删除。

管理已登录实例

您可以对已登录实例进行刷新、编辑、用户和权限管理、切换数据库、退出登录、删除实例等操作。

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在**元数据管理**页签下，右键单击已登录目标实例，进行管理操作。

> **说明**
> - 未登录实例中会展示用户当前区域所在主账号下具有访问权限的所有实例。
> - 您开通的Hologres引擎实例会默认生成一个 图标的VPC网络类型的未登录实例展示在列表中，该实例不支持编辑详细信息和删除。

您可以进行如下操作：

- **刷新**

  当您修改实例内容后，可以刷新当前实例同步信息。

- **编辑实例**

  您可以鼠标右键单击目标实例，选择**编辑实例**，编辑当前实例。更多关于编辑实例参数信息的操作，请参见编辑实例参数说明。

  > **说明**
  > - 图标的为公网类型的实例，支持进行所有参数的编辑修改。
  > - 图标的VPC类型的实例，仅支持对描述进行修改。

- **用户管理**

  您可以鼠标右键单击目标实例，选择**用户管理**。进入**用户管理**页面，您可以对实例的用户进行添加和删除操作。更多关于用户管理的操作，请参见用户管理。

- **DB授权**

  您可以鼠标右键单击目标实例，选择**DB授权**。进入**DB授权**页面，您可以新建数据库并授权，也可以对先有数据库进行权限修改操作。更多关于DB授权的操作，请参见DB管理。

- **切换登录数据库**

  您可鼠标右键单击目标实例，选择**切换登录数据库**。进入**登录实例**页面，对当前实例的数据库进行切换操作。更多关于切换数据库的操作，请参见登录实例。

- **新建数据库**

  您可以鼠标右键单击目标实例，选择**新建数据库**。进入**新建数据库**页面，为当前实例新建一个数据库，创建的数据库会展示在下方列表中。更多关于新建数据库的操作，请参见新建数据库。

- **退出登录**

  您可以鼠标右键单击目标实例，选择**退出登录**。确认实例信息后，单击**确认**完成退出登录的操作。退出登录的实例会再次展示到未登录实例列表中。

- **删除实例**

  您可以删除当前实例，鼠标右键单击目标实例，选择**删除实例**。确认实例信息后，单击**确认**完成删除实例的操作。

  > **说明** 当前仅支持删除 图标的为公网类型的实例， 图标的VPC类型的实例不支持删除。

3.3.2.4. 数据库

本文为您介绍如何使用HoloWeb新建、登录、编辑或删除数据库。

前提条件

存在已登录实例，请参见登录实例。

新建数据库

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面，单击**元数据管理 > 数据库**。

   您也可以在**元数据管理**界面的**已登录实例**列表。鼠标右击目标连接，选择**新建数据库**。

5. 配置**数据库名称**、**权限策略**和**立即登录**参数，单击**确认**。

参数	说明
实例名	默认展示当前数据库所在的已登录实例的名称。
数据库名称	您可以命名当前数据库的名称。 **说明** 配置的数据库名称必须唯一。
权限策略	您可以根据业务需求为数据库配置对应权限。更多关于权限策略的说明，请参见： ○ SPM ○ SLPM ○ 专家权限模型
立即登录	○ 是：登录后您就可以直接使用创建的数据库。 ○ 否：登录数据库之后才可以进行使用。

登录数据库

您可以进入HoloWeb开发页面，在**元数据管理**界面通过如下方式登录数据库：

- 在新建数据库时，配置**立即登录**参数为**是**，即默认登录了数据库，可直接进行使用。
- 在新建数据库时，配置**立即登录**参数为**否**，您需要在**已登录实例**列表中，鼠标右击目标未登录数据库，选择**登录数据库**。

> **说明** 当您的数据库为已登录状态时，图标为亮黑。当数据库未登录时，该图标置灰。

编辑数据库

1. 进入HoloWeb开发页面，在**元数据管理**界面的**已登录实例**列表中，显示所有登录实例名称。
2. 鼠标右击目标数据库，选择**编辑数据库**。
3. 您可以根据业务需求，修改**权限策略**参数的取值。

> **说明** 编辑数据库目前仅支持修改**权限策略**配置项。

删除数据库

1. 进入HoloWeb开发页面，在**元数据管理**界面的**已登录实例**列表中，显示所有登录实例名称。
2. 鼠标右击目标数据库，选择**删除数据库**。
3. 确认数据库内容之后，单击**确认**。

3.3.2.5. 模式

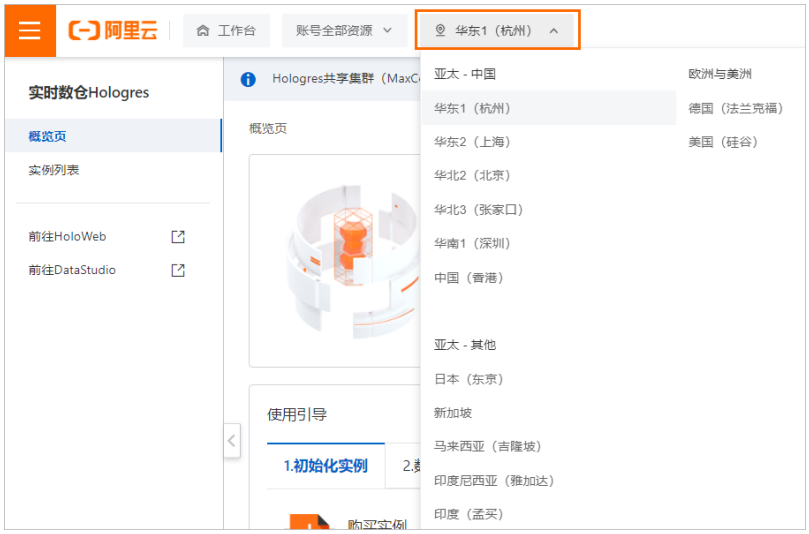
本文为您介绍如何使用HoloWeb新建、编辑或删除模式。

前提条件

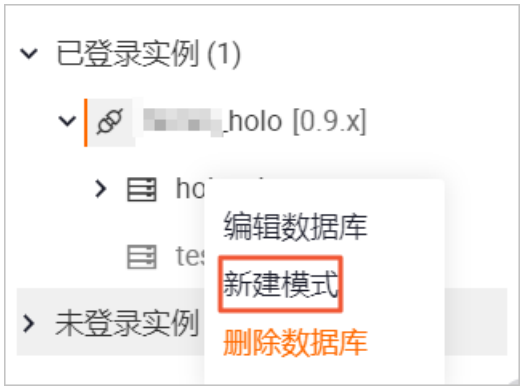
存在已登录实例，请参见登录实例。

新建模式

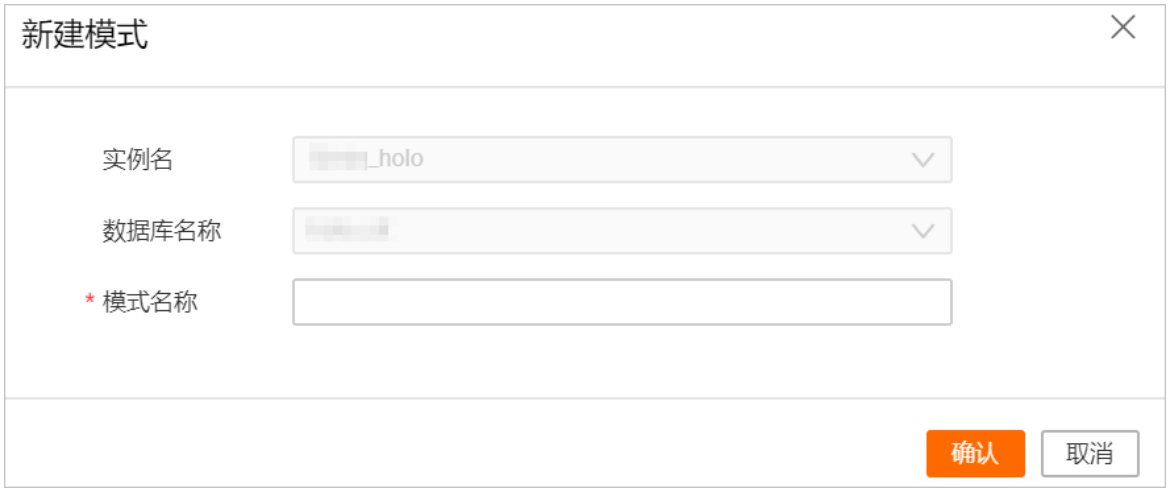
1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 进入HoloWeb开发页面，在**元数据管理**界面的**已登录实例**列表。打开目标实例，鼠标右击已登录目标数据库，选择**新建模式**。



5. 配置**模式名称**，单击**确认**，其中**实例名**和**数据库名称**默认填充当前模式所在的实例和数据库名称，无需修改配置。



新建成功的模式会展示在已登录数据库的列表中。

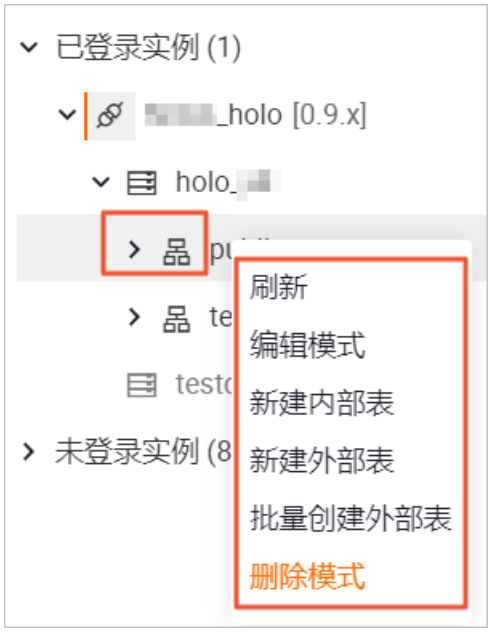
管理模式

您可以对模式进行刷新、编辑、新建内部表、新建外部表、批量创建外部表、删除模式等操作。

1. 进入HoloWeb开发页面，在**元数据管理**界面的**已登录实例**列表中，显示所有登录实例名称。
2. 单击目标实例下的已登录数据库，单击目标数据库，显示所有已创建的模式。

   > **说明** 数据库登录后，系统默认创建命名为public的模式。

3. 鼠标右击目标模式，您可以根据业务需求对当前模式进行管理操作。



您可以进行如下操作：

- **刷新**

  当您修改模式内容后，可以刷新当前模式同步信息。

- **编辑模式**

  您可以鼠标右键单击目标模式，选择**编辑模式**，编辑当前模式仅支持修改**模式名称**。

- **新建内部表、新建外部表、批量内部表**

  您可以鼠标右键单击目标模式，选择目标表类型，您可以在页面右侧配置各项参数。更多关于各种表的配置，请参见MaxCompute加速章节相关内容。

- **删除模式**

  您可以删除当前模式，鼠标右键单击目标模式，选择**删除模式**。确认模式信息后，单击**确认**完成删除模式的操作。

3.3.2.6. 表

本文为您介绍如何使用HoloWeb新建、编辑或删除内部表，以及预览内部表数据和DDL语句。

前提条件

存在已登录实例，请参见登录实例。

新建内部表

1. 登录Hologres管理控制台。

2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面顶部菜单栏，单击**元数据管理 > 表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**新建内部表**。

5. 在**新建内部表**页面，配置各项参数。

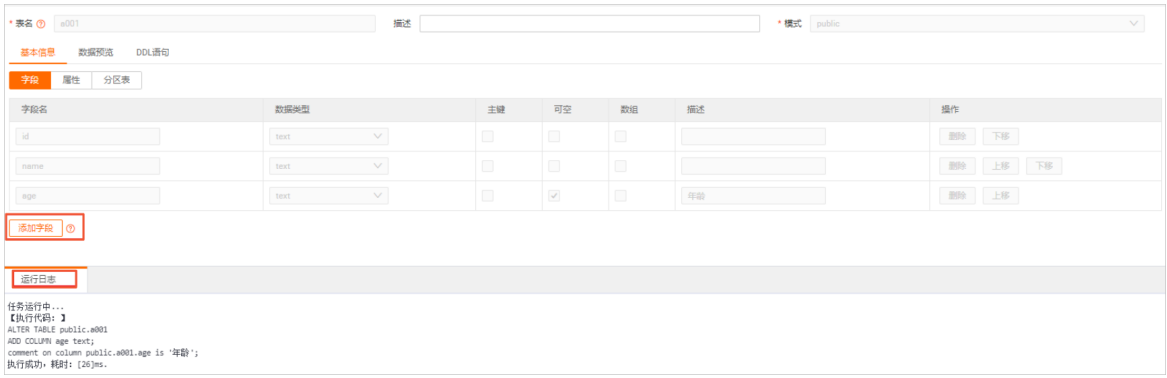
类别	参数	描述
基本属性	实例名	已登录的实例名称。
	数据库	当前已登录实例的数据库名称。
	表名	新建的Hologres内部表名称。
	描述	新建的Hologres内部表描述。
	模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。
字段	字段名	表中每一列的标识。
	数据类型	字段取值的类型。
	主键	表中每条数据的唯一标识。
	可空	字段是否可以设置为空。
	数组	有序的元素序列。
	描述	字段的描述信息。
	操作	包括**删除**、**上移**和**下移**。
属性	存储模式	包括**列存**和**行存**两种存储模式。 默认为**列存**。
	生命周期（秒）	如果数据在指定时间内未被修改，则系统将自动删除数据。 默认生命周期为**永久**。
	聚簇索引	排序索引。 索引的类型和列的顺序密切相关。聚簇索引帮助您加速执行索引列的Range和Filter查询。
	分段键	您可以指定部分列作为分段键。当查询条件包含分段列时，您可以通过分段键快速查找相应数据的存储位置。
	字典编码列	Hologres支持为指定列的值构建字典映射。 字典编码可以将字符串的比较转换为数字的比较，加速Group By和Filter查询。 默认设置所有**text**列至字典编码列中。

类别	参数	描述
	位图列	Hologres支持在位图列构建比特编码。 位图列可以根据设置的条件快速过滤字段内部的数据。 默认设置所有text列至位图列中。
	分布列索引	Hologres会按照分布列指定的列将数据shuffle到各个Shard，同样的数值会在同样的Shard中。以分布列做过滤条件时，可以大大提高执行效率。
分区表	无	选择分区字段。

6. 在页面右上角，单击**提交表**。提交之后，您可以在左侧对应模式下，刷新出新建的内部表。
   您可以单击表编辑页面右上角的**查询表**，跳转至SQL查询窗口，使用SQL语句进行查询。

编辑内部表

1. 在左侧导航栏，单击**已登录实例**，显示所有已登录的实例名称。
2. 单击目标实例下的**数据库**，显示所有已创建的数据库。
3. 单击目标数据库**模式**下的**表**，显示所有已创建的内部表。
4. 鼠标右击目标内部表，选择**编辑表**。
5. 在页面下方单击**添加字段**，使用可视化的方式添加表字段。同时，您可以在**运行日志**编辑框获取相应的SQL语句，查看任务运行信息。



> **说明** 编辑表暂不支持删除已有表字段。

6. 单击**提交表**，完成对当前内部表的编辑。

删除内部表

1. 在左侧导航栏**已登录实例**界面查询目标内部表。
   查询目标内部表的步骤请参见编辑内部表的步骤1~3。
2. 鼠标右击目标内部表，选择**删除表**。
3. 单击**确认**。

数据预览

1. 在左侧导航栏已登录实例界面查询目标内部表。

   查询目标内部表的步骤请参见编辑内部表的步骤1~3。

2. 鼠标双击目标内部表，在表编辑页面单击**数据预览**。

   示例预览的内部表数据如下。

DDL预览

1. 在左侧导航栏已登录实例界面查询目标内部表。

   查询目标内部表的步骤请参见编辑内部表的步骤1~3。

2. 鼠标双击目标内部表，在表编辑页面单击**DDL语句**。

   示例预览的DDL语句如下。

3.3.2.7. 视图

视图是从一个或多个表导出的虚拟表，其具有普通表的结构，但是不实现数据存储。视图内容由查询定义，主要包括单表视图和多表视图。单表视图一般用于查询和修改，会改变表数据。多表视图一般用于查询，不会改变表数据。本文为您介绍如何新建、重命名和删除视图。

前提条件

存在已登录实例，请参见登录实例。

新建视图

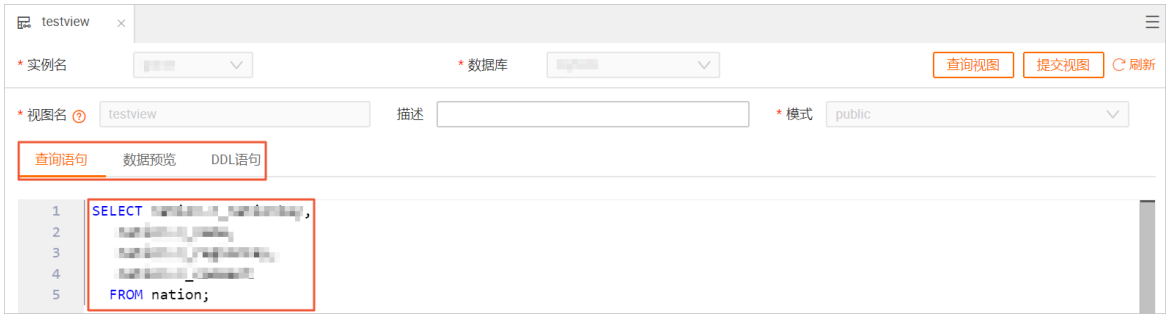
1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面，单击进入**元数据管理**页面。在**已登录实例**列表，依次单击目标数据库和数据库下已创建的目标模式。鼠标右击**视图**，选择**新建视图**。
5. 在**新建视图**页面，配置各项参数。

参数	描述
实例名	已登录的实例名称。
数据库	当前已登录实例的数据库名称。
视图名	新建的Hologres视图表名称。
描述	新建的Hologres视图表描述。
模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。

参数	描述
查询语句	您需要在查询语句下方的编辑框内，输入查询语句，可以是多表数据查询，也可以是单表数据查询。 ○ 单表视图一般用于查询和修改，会改变表数据。即，如果视图中数据来自于一个表时，修改视图中的数据，表数据会随之更新。修改表数据时，对应视图也会更新。 ○ 多表视图一般用于查询，不会改变表数据。即，如果视图数据来源于多个表时，不支持修改视图数据。 **说明** 在使用单表视图时，建议您谨慎修改视图数据，以避免对应的表数据被修改影响业务。

6. 单击页面右上角的**提交视图**，下方运行日志会提示您创建成功，创建完成的视图会展示在左侧对应目录下。
   您可以在视图创建完成后，在视图页面查看其对应的查询语句和DDL语句并进行数据预览。



管理视图

1. 进入HoloWeb开发页面，在**元数据管理**界面的**已登录实例**列表中，显示所有登录实例名称。
2. 在左侧导航栏顶部搜索或者在已登录实例列表中查找视图。

您可以进行如下操作：

- **刷新**

  当您修改视图内容后，可以右键单击**视图**，选择**刷新**同步信息。

- **编辑视图**

  您可以鼠标右键单击目标视图，选择**编辑视图**，在**查询语句**区域，编辑对应的语句后，在页面右上角单击**提交视图**。

- **删除视图**

  您可以删除当前视图，鼠标右键单击目标视图，选择**删除视图**。确认信息后，单击**确认**完成删除操作。

- **查询视图**

  您可以单击页面右上角的**查询视图**。页面将跳转到**临时Query**查询页面，您可以在编辑区域输入对应的SQL语句并单击**运行**，进行视图查询。更多关于Query查询的操作，请参见SQL窗口。

SQL方式创建和删除视图

Hologres也支持您通过SQL方式创建、查看和删除视图。更多关于视图的SQL语句说明，请参见VIEW。

3.3.2.8. MaxCompute加速

3.3.2.8.1. 外部表

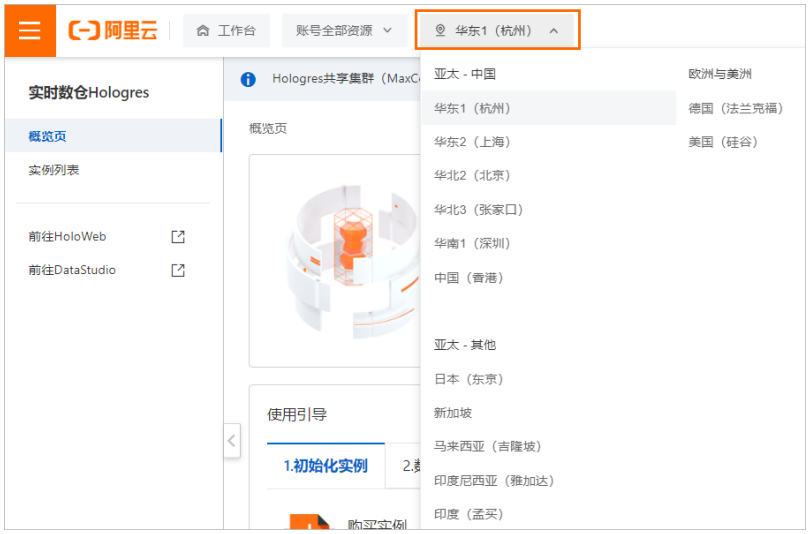
本文为您介绍如何使用HoloWeb新建、编辑或删除外部表，以及预览外部表数据和DDL语句。

前提条件

存在已登录实例，请参见登录实例。

新建外部表

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面的顶部菜单栏，单击**元数据管理 > MaxCompute加速**，单击**创建外部表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**新建外部表**。

5. 在**新建外部表**页面，配置各项参数。单击**提交表**。

参数	描述
实例名	已登录的实例名称。
数据库	Hologres当前已登录实例的数据库名称。
表名	新建的Hologres外部表名称。 输入目标MaxCompute表名后，将会自动创建同名外部表。在创建时不支持更改表名，如果您需要更改表名，可以在外部表创建成功后，在已登录实例列表中右键单击目标表进行修改。
描述	新建的Hologres外部表描述。
模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。

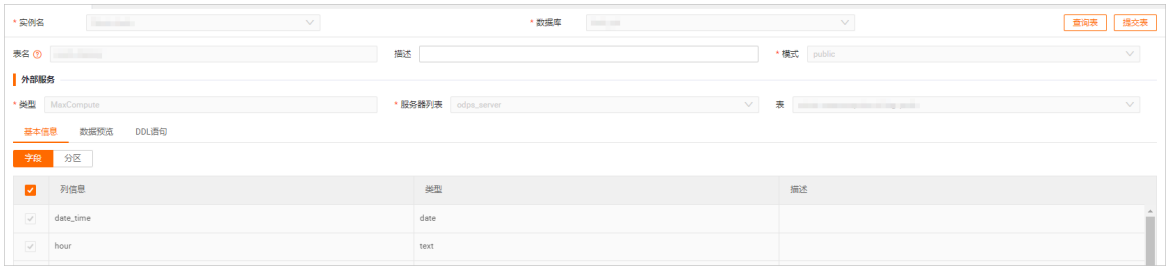
参数	描述
类型	外部表类型。 目前仅支持MaxCompute。
服务器列表	您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
表	MaxCompute的项目名和表名。 格式为**project.table_name**。 **说明** ○ 目前暂不支持跨地域查询外部表数据。 ○ 输入表名称后，会显示外部源表的所有字段，创建外部表时也将会默认创建所有字段。如果您需要创建部分字段，请使用SQL语句创建外部表，请参见CREATE FOREIGN TABLE。

说明 创建外部表同步MaxCompute表的数据时，会将数据库中表字段的Comment和列的Comment一并同步至Hologres。

6. 单击**提交表**，完成外部表的创建。提交之后，您可以在左侧对应模式下，刷新出新建的外部表。
7. （可选）您可以在已创建的外部表页面，单击**查询表**，进入SQL查询窗口，使用标准的PostgreSQL语言进行开发。

编辑外部表

1. 在左侧导航栏，单击**已登录实例**，显示所有已登录的实例名称。
2. 单击目标实例下的**数据库**，显示所有已创建的数据库。
3. 单击目标数据库**模式**下的**外表**，显示所有已创建的外部表。
4. 鼠标右击目标外部表，选择**编辑表**。
5. 您可以根据业务需求，更改需要映射的外部源表的**字段**或**分区**。



6. 单击**提交表**，完成对当前外部表的编辑。

修改外部表名称

1. 在左侧导航栏**已登录实例**界面查询目标外部表。

   查询目标外部表的步骤请参见编辑外部表的步骤1~3。

2. 鼠标右击目标外部表，选择**更名**。
3. 单击**确认**。

删除外部表

1. 在左侧导航栏**已登录实例**界面查询目标外部表。

   查询目标外部表的步骤请参见编辑外部表的步骤1~3。
2. 鼠标右击目标外部表，选择**删除表**。
3. 单击**确认**。

数据预览

1. 在左侧导航栏**已登录实例**界面查询目标外部表。

   查询目标外部表的步骤请参见编辑外部表的步骤1~3。
2. 双击目标外部表，在外部表的编辑界面单击**数据预览**。

3.3.2.8.2. 批量创建外部表

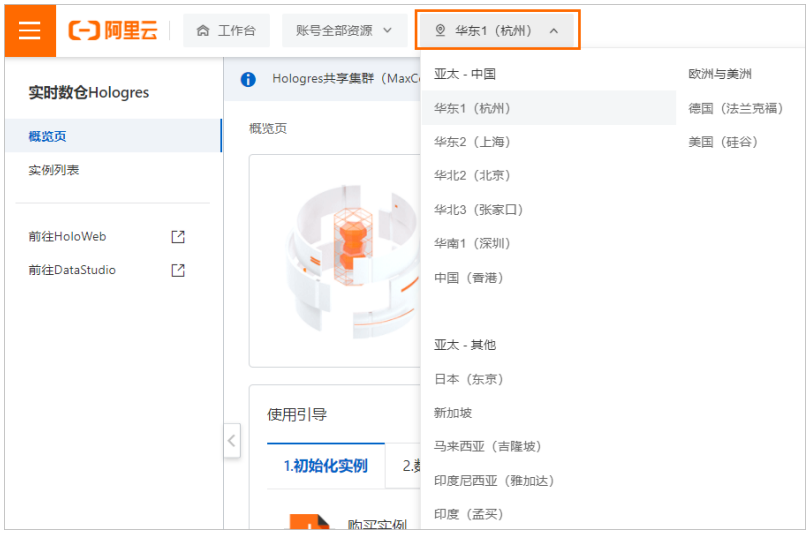
本文为您介绍如何使用HoloWeb，通过可视化的方式批量创建外部表。

前提条件

存在已登录实例，请参见登录实例。

批量创建外部表

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面的顶部菜单栏，单击**元数据管理 > MaxCompute加速**，单击**批量创建外部表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**批量创建外部表**。

5. 在**批量创建外部表**页面，配置各项参数。

类别	参数	描述
基本属性	实例名	已登录的实例名称。
	数据库	存放新创建外部表的Hologres数据库名称。
目标位置	模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。
	类型	目前仅支持MaxCompute外部表。
	服务器列表	您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
	远程库	MaxCompute的项目名称。

类别	参数	描述
来源	选择要直接加速的表	○ **整库加速**：批量创建所选项目下的所有表。 ○ **部分加速**：您可以通过搜索表名称或关键字，选择需要创建的表。 **说明** **部分加速**最多支持显示200张表，超出部分将不显示，但是也会创建成功。 例如，目标项目中共有203个名称中包含*test*的表，搜索*test*时，此处最多只会显示200个相关的表，但实际上203个表都会创建成功。
高级选项	表名冲突	○ **忽略，继续创建其他表**：创建表时，如果数据库中已存在当前创建的表名称，则忽略当前创建的表，继续创建其他表。 ○ **更新，修改同名表**：创建表时，如果数据库中已存在当前创建的表名称，则更新已有表的数据。 ○ **报错，不再重复创建**：创建表时，如果数据库中已存在当前创建的表名称，则发送报错，不再重复创建。
	数据类型不支持	○ **报错，导入失败**：如果创建表时存在不支持的数据类型，则产生报错，导入数据失败。 ○ **忽略，跳过不支持字段**：如果创建表时存在不支持的数据类型，则忽略不支持的字段，继续导入数据。

说明 创建外部表同步MaxCompute表的数据时，会将数据库中表字段的Comment和列的Comment一并同步至Hologres。

6. 单击**运行**，批量创建外部表。

3.3.2.8.3. 一键同步MaxCompute数据

本文为您介绍如何使用HoloWeb，通过可视化的方式快捷导入MaxCompute数据。

前提条件

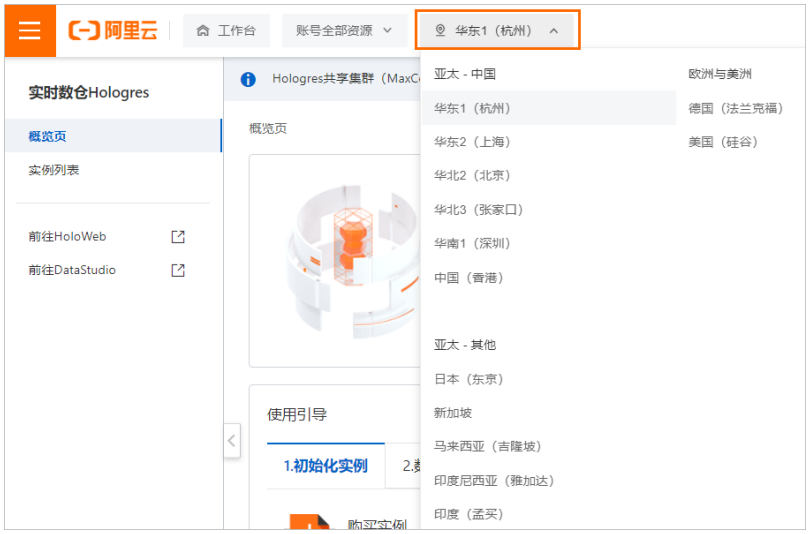
存在已登录实例，请参见登录实例。

背景信息

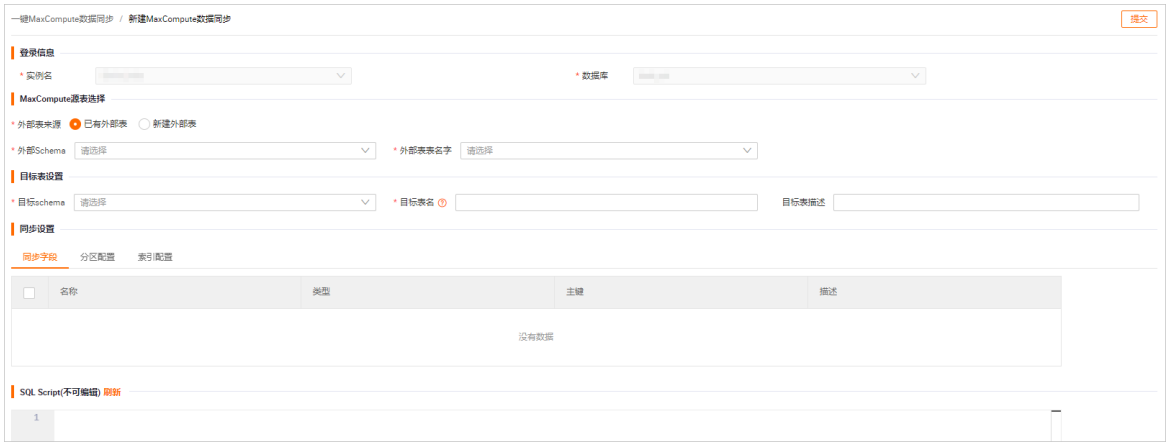
HoloWeb支持一键同步MaxCompute数据功能，您可以使用可视化方式导入MaxCompute表数据并进行查询。该方式比创建外部表直接查询数据的性能更好。

操作步骤

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面的顶部菜单栏，选择**元数据管理 > MaxCompute加速**，单击**一键MaxCompute数据同步**。
5. 配置**新建MaxCompute数据同步**页面的各项参数。



参数描述如下表所示。

类别	参数	描述
选择连接	实例名	已登录的实例名称。
	数据库	Hologres已登录实例的数据库名称。

类别	参数	描述
MaxCompute源表选择	外部表来源	○ **已有外部表**：表示在Hologres中已经建立映射MaxCompute数据的外部表。 ○ **新建外部表**：表示在Hologres中未建立MaxCompute数据映射的外部表。
	外部Schema	Hologres中已创建的MaxCompute外部表所在的Schema。 当**外部表来源**选择**已有外部表**时，需要配置该参数。
	外部表表名字	Hologres中已创建的MaxCompute外部表的名称。 当**外部表来源**选择**已有外部表**时，需要配置该参数。
	服务器列表	Hologres存放新建外部表的服务器。您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。 当**外部表来源**选择**新建外部表**时，需要配置该参数。
	表名	Hologres新建的外部表的项目名和表名。 格式为**project.table_name**。 当**外部表来源**选择**新建外部表**时，需要配置该参数。
目标表设置	目标Schema	当前表所在的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public** Schema。如果有新建的Schema，您也可以选择新建的Schema。
	目标表名	接收MaxCompute表数据的Hologres内部表名称。
	目标表描述	目标表的信息描述。
同步设置	同步字段	需要导入的MaxCompute表字段。 您可以选择导入部分或全部字段。
分区配置	分区字段	选择分区字段，Hologres将会默认将表创建为分区表。 Hologres仅支持一级分区。如果您需要导入MaxCompute的多级分区，则在Hologres中设置一级分区即可，其余分区自动映射为Hologres的普通字段。
	业务日期	如果MaxCompute表使用日期进行分区，则您可以选择具体的分区日期，系统将会导入指定日期的数据至MaxCompute表。

类别	参数	描述
索引配置	存储模式	○ **列存**，适用于各种复杂查询。 ○ **行存**，适用于基于主键的点查询和Scan。 如果不指定存储模式，则默认为**列存**。
	生命周期（秒）	表数据的生命周期。默认为**永久**存储。 指定生命周期后，如果数据在指定时间内未被修改，则引擎将会在到期后的某一个时间段删除数据。
	聚簇索引	排序索引Clustering_key。 索引的类型和列的顺序密切相关。聚簇索引帮助您加速执行索引列的Range和Filter查询。
	分段键	您可以指定部分列作为分段键Segment_key。当查询条件包含分段列时，您可以通过分段键快速查找相应数据的存储位置。
	字典编码列	Hologres支持为指定列的值构建字典映射。 字典编码可以将字符串的比较转换为数字的比较，加速Group By和Filter查询。 默认设置所有**text**列至字典编码列中。
	位图列	Hologres支持在位图列构建比特编码。 位图列可以根据设置的条件快速过滤字段内部的数据。 默认设置所有**text**列至位图列中。
	分布列索引	Hologres会按照分布列指定的列将数据shuffle到各个Shard，同样的数值会在同样的Shard中。以分布列做过滤条件时，可以大大提高执行效率。

SQL Script为您自动解析当前可视化操作对应的SQL语句。

6. 单击右上角的**提交**。

3.3.3. SQL编辑器

3.3.3.1. 文件夹

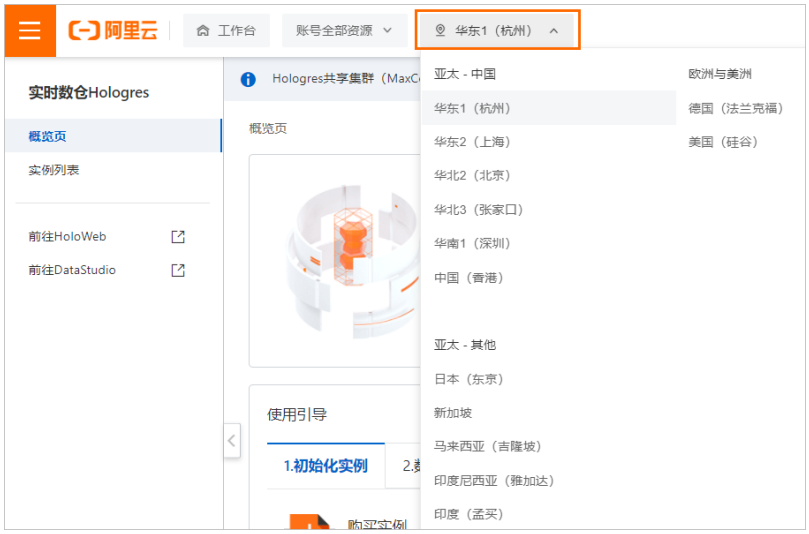
本文为您介绍如何使用HoloWeb新建、重命名、删除以及快速查询文件夹。

前提条件

存在已登录实例，请参见登录实例。

新建文件夹

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。



3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面的顶部菜单栏，单击**SQL编辑器 > 新增文件夹**。

   您也可以在左侧导航栏，鼠标右击**我的SQL查询**，选择**新建文件夹**。同时，HoloWeb支持在已有文件夹中创建子文件夹。
5. 在**新建文件夹**对话框，配置**文件夹名称**和**目录**参数。

   如果您当前已存在对应目录，您可以从下拉列表中选择文件夹存放的目标目录。
6. 单击**确认**，创建完成的文件夹会展示在左侧对应目录下。

重命名文件夹

1. 在左侧导航栏，单击**我的SQL查询**，显示所有已创建的文件夹。
2. 鼠标右击目标文件夹，单击**重命名文件夹**。
3. 您可以根据业务需求，配置**重命名名称**。
4. 单击**确认**。

删除文件夹

1. 在左侧导航栏，单击**我的SQL查询**，显示所有已创建的文件夹。
2. 鼠标右击目标文件夹，单击**删除文件夹**。

> **说明** 删除父文件夹前，您需要先删除父文件夹中所有子文件夹和SQL查询。

3. 单击**确认**。

文件夹快速查询

如果当前**我的SQL查询**中内容较多，需要快速查询并定位到目标文件夹。您可以使用如下方法进行查询：

1. 在左侧导航栏，**Query查询**下方的搜索框输入目标文件夹名称。

2. **我的SQL查询**列表中会列出相关的文件夹名称，单击目标文件夹名称即可查看详情。

3.3.3.2. SQL窗口

本文为您介绍如何使用HoloWeb新建、编辑、删除、复制及重命名SQL查询。SQL窗口适用于做交互式的短查询，不适合用于大数据量的导入导出等离线作业场景，不支持执行超过60分钟的SQL。

前提条件

存在已登录实例，请参见登录实例。

新建SQL查询

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。

4. 在HoloWeb开发页面的顶部菜单栏，单击**SQL编辑器 > 新增SQL窗口**，单击**确认**。

   您也可以在左侧导航栏，鼠标右击**我的SQL查询**，选择**新建SQL查询**。

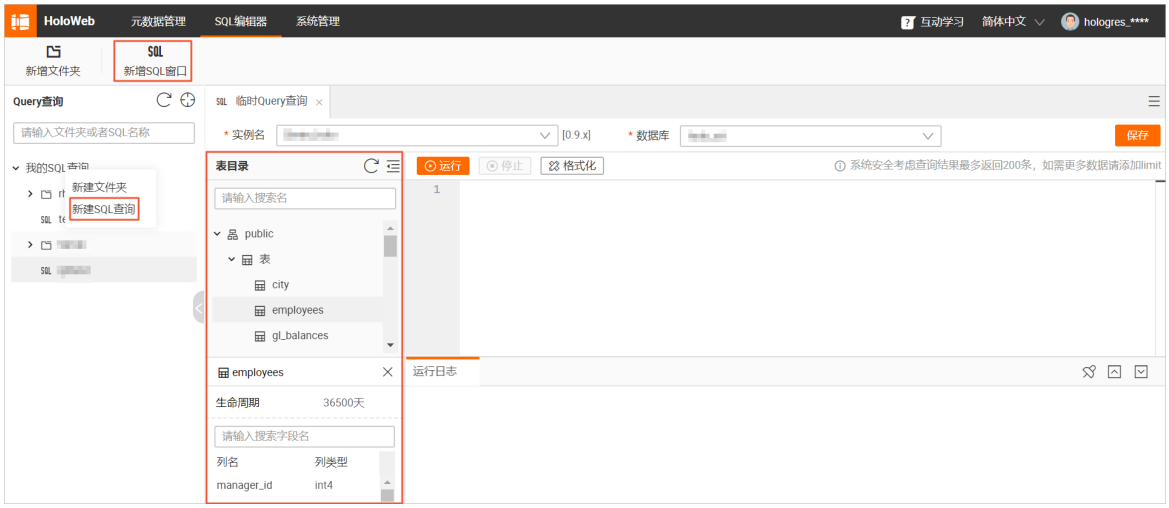
   在**新建SQL查询**对话框，配置各项参数，系统默认会自动填充当前的实例名和数据库名。

   参数描述如下表所示。

参数	描述
作业名称	新建的SQL查询名称。
目录	新建SQL查询存放的位置。默认目录为**我的SQL查询**，您也可以选择存放在已创建的文件夹中。
实例名	您可以从当前已存在实例的下拉列表中选择目标实例名称。 **说明** 当前实例列表会展示已登录和未登录实例名称。当您选择的为未登录实例时，您可以在执行完SQL语句并保存后，登录该实例即可在列表中看到新增的内容。
数据库名称	Hologres当前实例下的数据库名称。

5. 新增生成的SQL会展示在左侧**我的SQL查询**列表中。您可以在页面的**表目录**列表中选择目标表，查看对应表的字段名和类型。您也可以在SQL编辑区域使用标准的PostgreSQL语言进行开发。

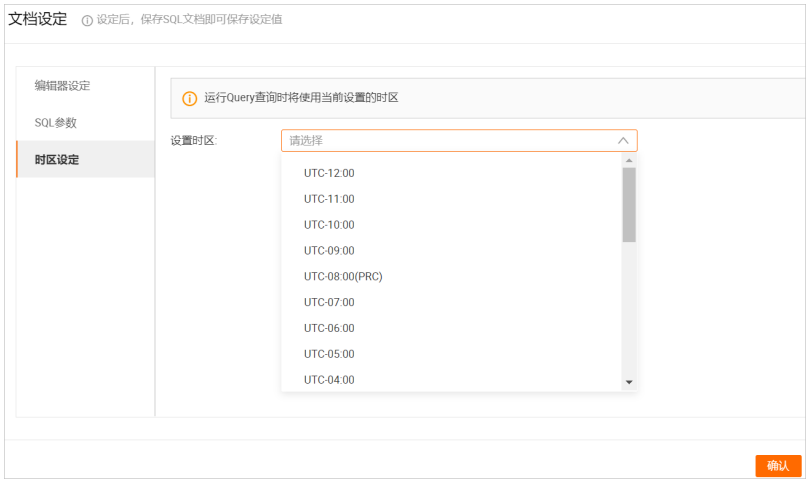
   说明 SQL查询对表名称和字段名称的大小写不敏感。如果您需要精确查询名称大小写完全一致的表，则需要使用双引号将该表名称引起来。



6. （可选）设置时区
   - 针对当前查询页面
     a. 在**SQL查询**页面右上角，单击**文档设定**。

     b. 在**文档设定**对话框左侧导航栏，单击**时区设定**。
     c. 在**设置时区**下拉框选择时区，单击**确认**完成时区设置。

        运行Query查询时将使用此时区。



   - 针对HoloWeb设置全局时区
     a. 在HoloWeb页面右上角，单击用户名称。
     b. 单击**设定**，进入**用户设定**对话框。

     c. 在**用户设定**对话框左侧导航栏，单击**时区设定**。
     d. 在**设置时区**下拉框选择时区，单击**保存**完成时区设置。

> **说明** 如果在当前查询页面设置了时区且针对HoloWeb设置了全局时区，当前查询页面设置的时区优先级高。

7. 单击**运行**，执行输入的SQL语句。

> **说明** HoloWeb支持SQL运行最长60分钟，超过60分钟，会有超时提示：cancel query due to timeout, queryTimeout setting is: 3600s

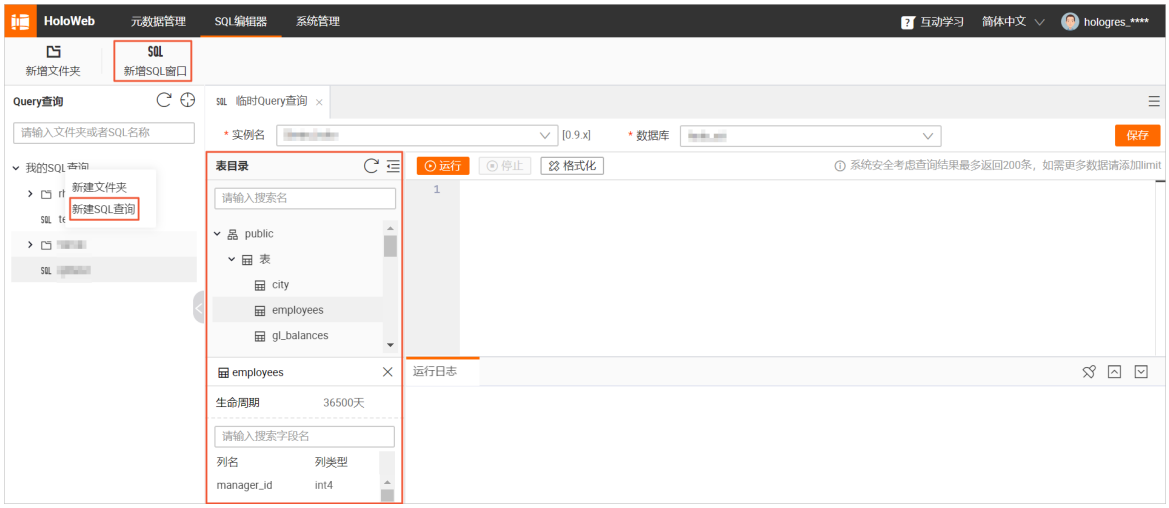
8. 单击**保存**，保存当前SQL查询窗口的相关内容。

表目录

在您完成新建SQL查询和编辑SQL查询之后，您可以查看表目录，方便您查看和了解表内部的结构。

1. 在HoloWeb开发页面的顶部菜单栏，单击**SQL编辑器 > 新增SQL窗口**，单击**确认**。

   您也可以在**我的SQL查询**中打开目标**SQL查询**。



2. 您可以在页面的**表目录**列表查看所有的内部表。选择目标表，查看对应表的字段名和类型。

   您也可以在SQL编辑区域使用标准的PostgreSQL语言进行开发。例如，您可以在SQL编辑区域运行新增表语句，并单击图标，在表目录下方可以展示表结构。

> **说明** SQL查询对表名称和字段名称的大小写不敏感。如果您需要精确查询名称大小写完全一致的表，则需要使用双引号将该表名称引起来。

编辑SQL查询

1. 在左侧导航栏，单击**我的SQL查询**，查找目标SQL查询。

   如果目标SQL查询存放于文件夹中，您可以在**Query查询**下方的搜索框输入目标SQL查询名称。

2. 鼠标双击目标SQL查询，可以**编辑SQL查询**。
3. 您可以在SQL编辑框输入需要执行的SQL语句，单击**运行**。

   示例SQL语句如下。

```
CREATE TABLE test (
 "id" bigint NOT NULL,
 "name" text NOT NULL,
 "age" bigint,
 "class" text NOT NULL,
PRIMARY KEY (id)
);
```

   您可以通过表目录方式查看示例语句的执行结果。
4. 单击**保存**。保存成功后，当前SQL查询会保存所有输入的SQL语句。

删除SQL查询

1. 在左侧导航栏，单击**我的SQL查询**，查找目标SQL查询。

   如果目标SQL查询存放于文件夹中，您可以在**Query查询**下方的搜索框输入目标SQL查询名称。

2. 鼠标右击目标SQL查询，单击**删除SQL查询**。
3. 单击**确认**。

复制SQL查询

1. 在左侧导航栏，单击**我的SQL查询**，查找目标SQL查询。

   如果目标SQL查询存放于文件夹中，您可以在**Query查询**下方的搜索框输入目标SQL查询名称。

2. 鼠标右击目标SQL查询，单击**复制SQL查询**。
   该操作仅复制当前SQL查询中的SQL语句。
3. 您可以将复制的SQL语句粘贴至其他SQL查询窗口使用。

重命名SQL查询

1. 在左侧导航栏，单击**我的SQL查询**，查找目标SQL查询。
   如果目标SQL查询存放于文件夹中，您可以在**Query查询**下方的搜索框输入目标SQL查询名称。

2. 鼠标右击目标SQL查询，单击**重命名**。
3. 您可以根据业务需求，配置**重命名名称**。
4. 单击**确认**。

3.3.4. 诊断与优化

3.3.4.1. 查看活跃Query

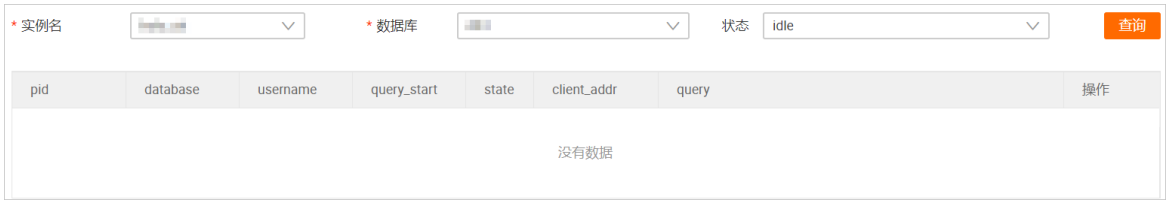
本文为您介绍如何使用HoloWeb查看活跃Query的详细信息。

前提条件

- 注册阿里云账号，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres，详情请参见购买Hologres。

操作步骤

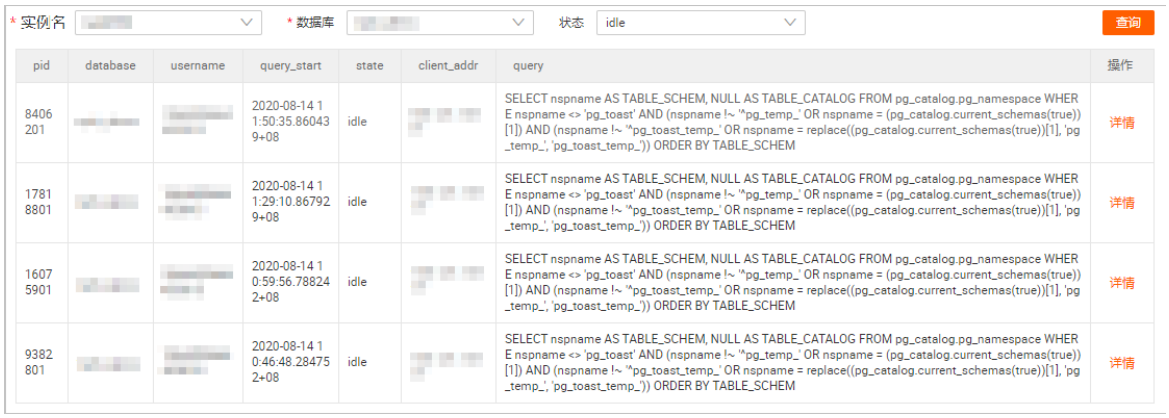
1. 在HoloWeb开发页面的顶部菜单栏，单击**系统管理**。
2. 在左侧导航栏，单击**性能优化 > 活跃Query**。
3. 配置**活跃Query**页面的各项参数。



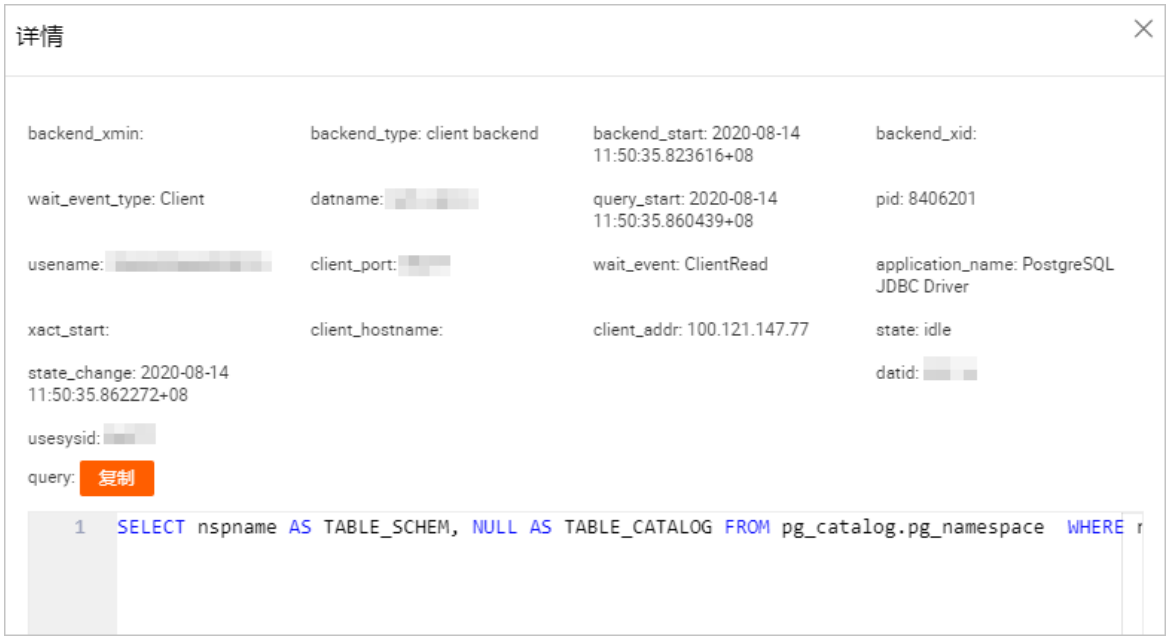
参数	描述
实例名	选择已登录的实例名称。
数据库	所选连接中已创建的数据库名称。
状态	数据库中进程的活动状态。 ○ idle：表示进程在等待新的客户端命令。 ○ active：表示进程正在执行查询操作。 ○ idle in transaction：表示进程处于一个事务中，但是当前没有执行查询操作。 ○ idle in transaction (aborted)：表示进程处于一个事务中，该事务存在语句错误，并且进程当前没有执行查询操作。 ○ fastpath function call：表示进程正在执行一个 `fast-path` 函数。 ○ disabled：表示进程的track activities被禁用。

4. 单击**查询**。

   示例查看数据库中状态为**idle**的查询操作。



   您可以单击目标查询**操作**列的**详情**，查看当前查询的详细信息。



您还可以单击**详情**页面的**复制**，复制当前查询执行的SQL语句。

3.3.4.2. 查看执行计划

HoloWeb支持以树形图的形式展现SQL查询的逻辑执行计划和物理执行计划，以便帮助用户更便捷的分析执行计划。本文为您介绍如何使用HoloWeb的执行计划和运行分析功能查看SQL语句的执行计划。

前提条件

- 注册阿里云账号，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres，详情请参见购买Hologres。

查看执行计划

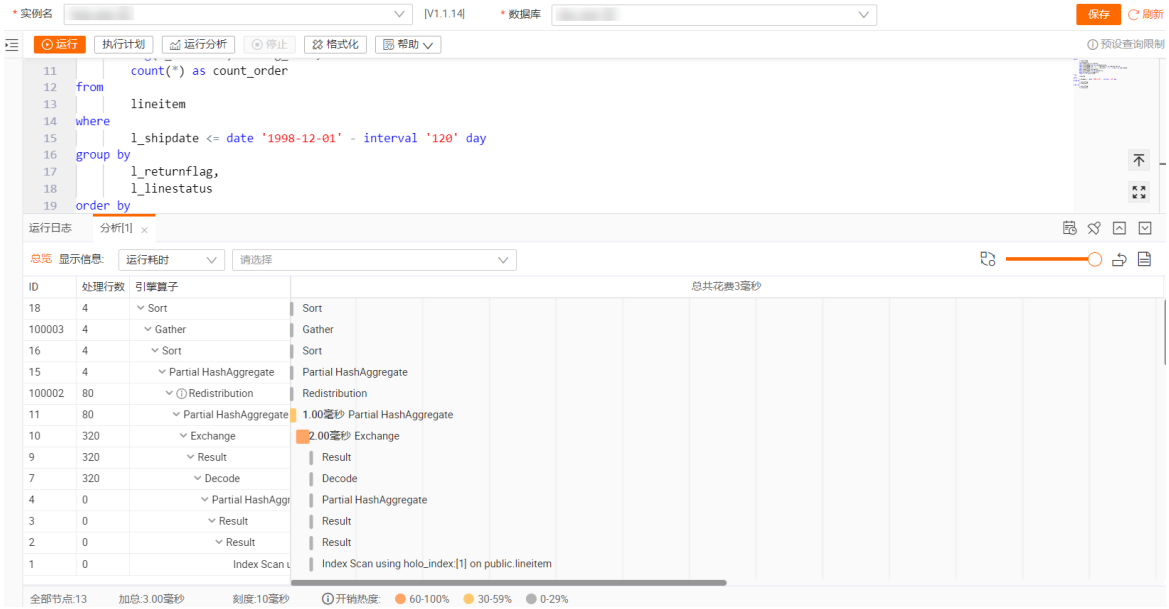
1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**SQL编辑器**。
3. 在**SQL编辑器**页面，单击左上角的**新增SQL窗口**。
4. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，在SQL查询的编辑框输入需要查看执行计划的SQL语句，单击**执行计划**，即可查看图形化的执行计划。

5. （可选）在**计划**页签，单击右上角 图标，将执行计划切换到DAG图显示。

查看运行分析

运行分析对SQL语句进行执行分析，返回SQL语句具体的物理执行计划及其运行开销。

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**SQL编辑器**。
3. 在**SQL编辑器**页面，单击左上角的**新增SQL窗口**。
4. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，在SQL查询的编辑框输入需要查看执行计划的SQL语句，单击**运行分析**，即可查看图形化的运行分析。



5. （可选）在**分析**页签，单击右上角 图标，将执行计划切换到DAG图显示。

> 说明 在DAG图上单击算子，会显示该算子的详细信息。

3.3.4.3. 历史慢Query

Hologres从V0.10开始支持慢Query的查询与分析，帮助您对系统中发生的慢Query或失败Query进行诊断、分析和采取优化措施。本文将为您介绍，如何通过可视化的方式查看并分析历史慢Query。

使用限制

- 该功能仅Hologres V0.10及以上版本支持，请在Hologres管理控制台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单由专业人员为您升级实例。
- 为了保证系统稳定性，避免流量超载，查询最多返回2000条慢Query日志。
- 当前HoloWeb仅支持查看最多7天的历史慢Query日志。

查看慢Query

1. 登录HoloWeb控制台，单击顶部导航栏的**诊断与优化**。
2. 在左侧导航栏单击**历史慢Query**。
3. 在**历史慢Query**详情页，配置如下筛选信息。

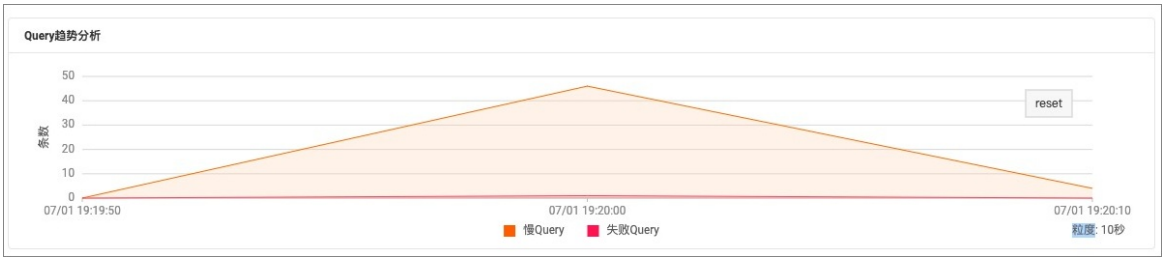
选项名称	选项描述	相关限制
实例名	需要查询Query的实例名称，默认为当前登录的实例。	无
数据库	需要查询慢Query的数据库名称。	需要在权限范围内才能查看慢Query日志，否则只能查看当前账号的相关日志，详情请参见慢Query日志查看与分析。
表名	根据表名查看当前表相关的慢Query。	
用户	根据用户云账号搜索相关慢Query。	
限制行数	查询慢Query日志返回的数据行数。	最多可展示2000条慢Query。
运行时长	SQL的运行时长。	默认采集大于1秒的Query。
图维度	可选择**慢Query**和**失败Query**，用于限制Query趋势分析图的展示维度。	无
Query	搜索SQL，支持使用百分号（%）模糊匹配表名。	无
Type	执行的Query类型，包括DDL以及DML等。	无
PID	连接所对应的PID。	无
时间范围	可查询的时间范围。	最多可选择7天。

4. 配置完成后单击**查询**，即可查看对应的Query趋势分析和Query列表。详情请参见查看Query趋势分析与Query列表。

查看Query趋势分析与Query列表

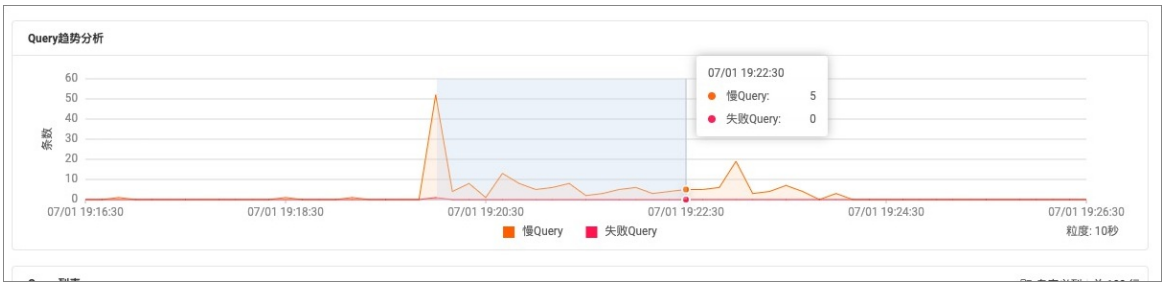
- **Query趋势分析**
  - 查看单个时间点前后的Query。

    鼠标在趋势图上单击时间节点，会展示该时间节点前后十秒钟内的趋势图。

  - 查看区域时间点内的Query。

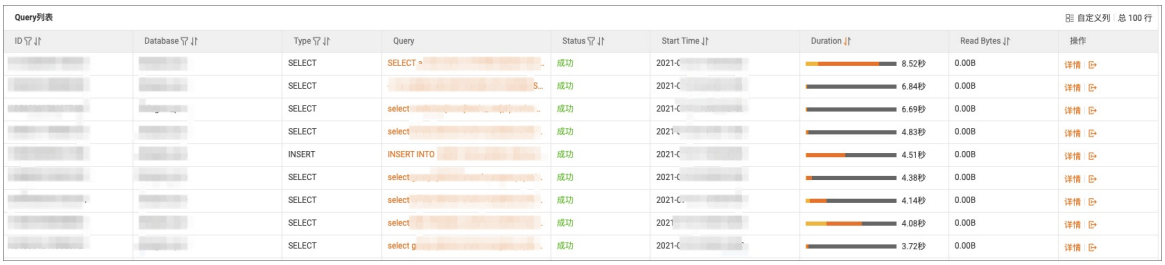
    您可在趋势图上选中某个区间，查看该区间所包含的时间节点内的趋势分析图。

- **Query列表**

- 查看默认列表内容。

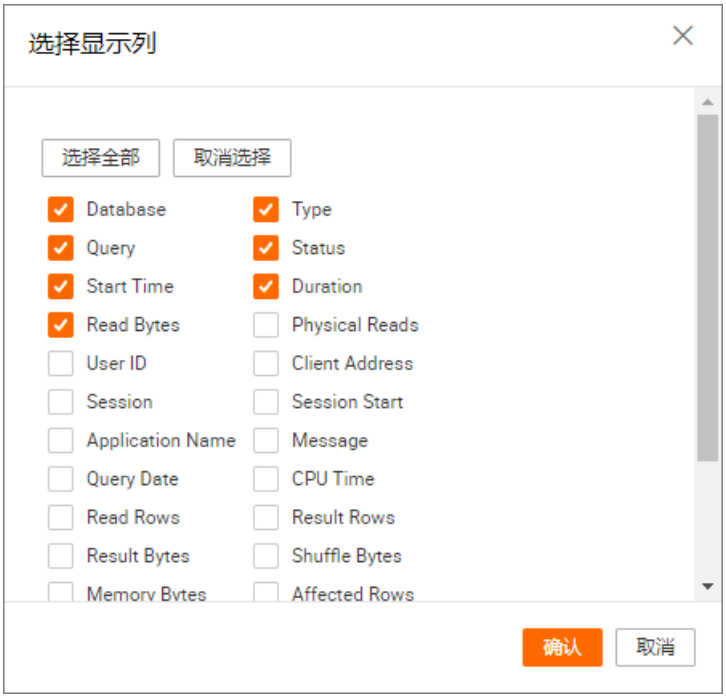
  系统为您提供了部分列表，相关列名含义如下表。



列名称	作用及相关描述
ID	执行SQL命令的用户所属云账号ID。
Databass	Query所属数据库名称。
Type	SQL的操作类型。
Query	单击可查看Query详情。
Status	操作结果，有成功和失败两种状态可选。
Start Time	Query开始执行的时间。
Duration	SQL运行总耗时，包括优化器执行时间、开始执行Query的时间和返回Query结果的时间。
Read Bytes	读取数据的大小。
操作	该列下均为**详情**。 ■ 单击**详情**可进入Query详情页，操作详情可以展示该Query的详细信息，包括基础信息（例如DB、PID）、高级信息、总耗时、启动耗时等。您可以结合优化内部表的性能，为SQL进行自助化调优。 ■ 单击详情右侧的图标可为当前Query直接打开一个新的SQL编辑器。

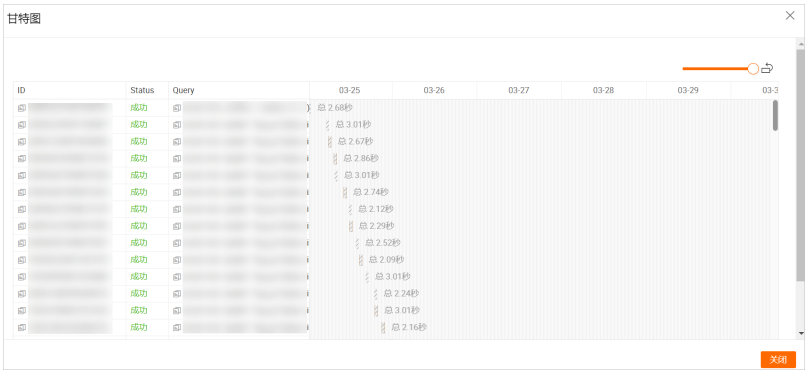
- 查看其他字段内容。

  如系统提供的字段无法满足您的需求，您可单击Query列表右上角的**自定义列**，在弹出的对话框中选中更多内容进行查看。



- 查看甘特图

  HoloWeb支持甘特图分析功能，您可单击Query列表右上角的**甘特图**，在弹出的对话框中查看同一时间Query的并发执行情况，帮助您更好地分析并发场景下的性能瓶颈问题。



3.3.4.4. 查看HoloWeb中SQL语句的执行历史

HoloWeb为您提供当前登录账号在HoloWeb中执行SQL语句的历史记录，方便您查看目标任务执行的语句，以及语句的执行状态及耗时等信息，并根据查询到的信息，快速判断执行的SQL语句是否合理，以便优化不合理语句。本文为您介绍如何查询SQL语句的执行历史信息。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres，详情请参见购买Hologres。
- 存在已登录实例，请参见登录实例。

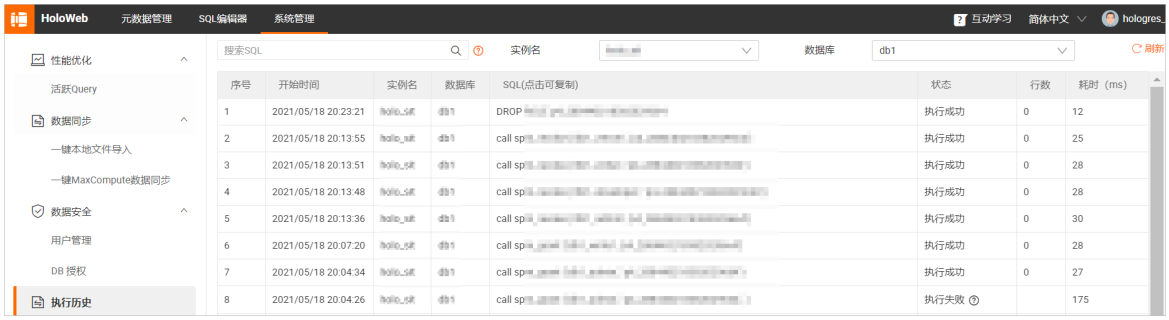
- 存在已登录实例，请参见登录实例。

注意事项

查询的SQL语句执行历史为当前登录账号在HoloWeb的执行历史，不包含其他账号以及其他工具的执行历史。

查看SQL语句的执行历史

1. 在HoloWeb开发页面的顶部菜单栏，单击**系统管理**。
2. 在左侧导航栏，单击**执行历史**。
3. 在**执行历史**页面，配置**实例名**和**数据库**。
4. 单击**刷新**，查看目标数据库中所有已执行的SQL语句及其执行状态、耗时等相关信息。



查询到的参数信息描述如下。

参数	描述
开始时间	SQL语句开始执行的时间。
实例名	当前查询的已登录的实例名称。
数据库	Hologres当前实例对应的数据库名称。
SQL	执行的SQL语句详情。
状态	取值如下： ○ 执行成功。 ○ 执行失败。您可以点击可以查看失败原因。
行数	执行SQL语句进行查询时返回的结果行数。 HoloWeb执行SQL语句时，默认在SQL语句后添加 `Limit 200` 限制，如果您需要展示更多的行数，您可以使用 `Limit N` 语句，设置N的取值大于200即可。
耗时	当前SQL语句的执行时间。 如果SQL语句的执行时间不符合业务预期，您可以结合管理控制台的监控指标进行分析，对相应的任务进行调优，详情请见Hologres管控台的监控指标和优化内部表的性能。

如果不选择**实例名**和**数据库**，则默认展示当前账号在HoloWeb的所有SQL执行历史。

您也可以在**搜索SQL**框，输入需要查询的SQL语句，单击查询指定语句的执行历史。

> **说明** HoloWeb支持使用 % 模糊匹配表名称。

3.3.5. 数据方案

3.3.5.1. 一键上传本地文件

本文为您介绍如何使用HoloWeb导入本地文件至Hologres。

前提条件

- 注册阿里云账号，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres，详情请参见购买Hologres。

背景信息

HoloWeb的可视化方式仅支持导入小于100M的文件至Hologres，大于等于100M的文件请使用 `COPY` 命令导入，详情请参见使用COPY命令导入或导出本地数据。

操作步骤

1. 登录Hologres管理控制台。
2. 在顶部菜单栏左侧，选择相应的地域。

3. 单击**前往HoloWeb**，进入HoloWeb开发页面。
4. 在HoloWeb开发页面的顶部菜单栏，单击**系统管理**。
5. 在左侧导航栏，选择**数据同步 > 一键本地文件导入 > 新建数据导入**。
6. 配置**一键本地文件上传**对话框中**选择目标表**页面的各项参数，单击**下一步**。

参数	描述
作业名称	新建的作业名称。
实例名	选择已登录的实例名称。
目标库	Hologres对应实例中已创建的数据库名称。
目标Schema	Hologres中已创建的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public**。如果有新建的Schema，您也可以选择新建的Schema。
选择要导入的数据表	用于存储本地文件的表名称。 导入本地文件前，您需要在目标数据库中创建一张用于存储本地文件的表。

7. 配置**选择数据源表**页面的各项参数，单击**下一步**。

参数	描述
选择文件	需要上传的本地文件。 仅支持上传TXT、CSV和LOG类型的文件。
选择分隔符	○ 逗号 ○ Tab ○ 分号 ○ 空格 ○ | ○ # ○ & 您也可以自定义分隔符。
原始字符集	○ GBK ○ UTF-8 ○ CP936 ○ ISO-8859
首行为标题	勾选则设置首行数据为标题。

8. 在**导入总览**页面，单击**执行**。

   您可以查看本地文件导入的Schema、数据库、表、状态及启动时间等信息。

3.3.5.2. 管理MaxCompute同步任务

HoloWeb支持使用可视化方式一键同步MaxCompute数据，方便您快速查询MaxCompute数据。本文为您介绍如何使用HoloWeb，新建MaxCompute同步数据任务并查看任务的状态信息。

前提条件

- 阿里云账号注册，详情请参见。
- 实名认证，详情请参见或。
- 开通交互式分析Hologres，详情请参见购买Hologres。

新建MaxCompute数据同步任务

新建MaxCompute数据同步任务，详情请参见一键同步MaxCompute数据。

查看MaxCompute数据同步任务信息

1. 在HoloWeb开发页面的顶部菜单栏，单击**系统管理**。
2. 在左侧导航栏，单击**数据同步 > 一键MaxCompute数据同步**。
3. 配置**一键MaxCompute数据同步**页面的各项参数，单击**查询**。

如果不配置**实例名**、**数据库**和**状态**，默认查询该账号下所有状态的MaxCompute数据同步任务。

参数描述如下表所示。

参数	描述
实例名	已登录的实例名称。
数据库	Hologres当前实例对应的数据库名称。
状态	取值如下： ○ **正在运行**：表示同步任务正在运行中。 ○ **执行成功**：表示同步任务已经执行成功。 ○ **执行失败**：表示同步任务执行失败。 ○ **被中断**：表示同步任务被中断。

您可以对查看到的任务执行如下操作：

- 查看任务详情。

  单击同步任务**操作**列的**详情**，进入**MaxCompute数据同步详情**，查看任务的详细信息。

- 重新运行任务。

  单击同步任务**操作**列的**重新运行**，重新执行该同步任务。

  任务重新运行后，如果您希望中断该任务，则可以单击**操作**列的**停止**。

- 删除同步任务。

  单击同步任务**操作**列的**删除**即可删除该同步任务。

- 查看同步任务的执行历史。

单击同步任务**操作**列的**执行历史**，查看该MaxCompute数据同步任务中每条SQL命令的执行情况。例如，SQL语句执行成功或失败，执行语句的耗时时间。

您可以直接点击相应SQL语句复制使用。

3.3.6. 安全中心

3.3.6.1. 用户管理

您可以使用Hologres管理控制台的用户管理模块，新增或删除用户，以及为用户授权。方便您以可视化方式更细致的管理实例内的用户。

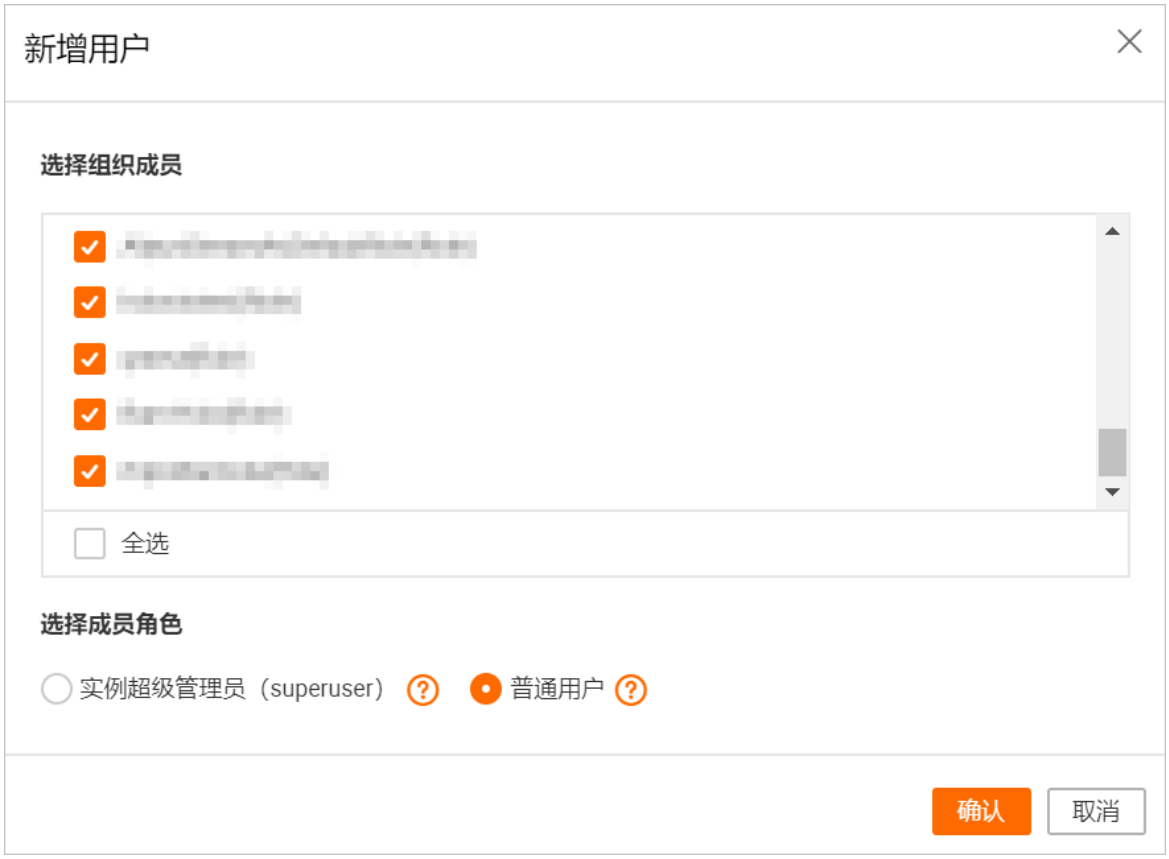
购买实例的阿里云账号默认为当前实例的超级管理员Superuser，拥有实例的所有权限。在未添加其他用户之前，用户管理页面只显示当前阿里云账号的相关信息。用户管理界面显示项如下表所示。

显示项	描述
成员	当前实例中的用户名，包括阿里云账号、RAM用户和自定义账号。
云账号	当前实例中用户的阿里云账号ID。 示例如下： • 阿里云账号：`11822780xxx`。 • RAM用户：`p4_269499383xxxx`。 • 自定义账号：`BASIC$xxx`。
账号类型	当前实例的用户的账号类型。 • 阿里云账号 (ALIYUN)。 • RAM账号 (RAM)。 • BASIC账号 (BASIC)。
角色类型	当前实例的用户拥有的权限类型。 • Superuser • Normal
操作	您可以单击目标成员**操作**列的**删除**，将该用户从实例内删除，删除后用户将没有实例的任何访问权限。

新增用户

您可以在用户管理页面，使用可视化方式为实例新增用户。

1. 在**用户管理**页面选择目标实例名称，在页面右上角单击**新增用户**。
2. 在**新增用户**对话框，选择添加当前阿里云账号下已有的RAM用户，并选择**实例超级管理员（superuser）**或**普通用户**用户类型。



> **说明** 如果当前阿里云账号没有RAM用户，您需要创建一个RAM用户，详情请参见RAM用户权限授权快速入门。

- 实例超级管理员（Superuser）：拥有实例的所有权限，如果子账号被授予为Superuser，则无需再额外进行其他授权。
- 普通用户：仅被创建至实例中，没有实例内任何对象（例如数据库、Schema和表等）的查看及操作权限，需要授予相应的权限，才可以查看或操作实例。

  推荐您前往DB管理页面，使用可视化方式进行授权。您也可以选择SQL语句授权，详情请参见RAM用户权限授权快速入门。

> **说明** Hologres仅支持使用阿里云账号新增用户。

删除用户

您可以在**用户管理**页面选择目标实例名称，在列表中单击目标成员**操作**列的**删除**，将该用户从实例内删除，删除后用户将没有实例的任何访问权限。

创建自定义用户

> **说明** 暂时不支持对自定义用户设置IP白名单。

您可以在用户管理页面，使用可视化方式为实例创建自定义用户。

1. 在**用户管理**页面选择目标实例名称，在页面右上角单击**创建自定义用户**。
2. 在**创建自定义用户**对话框，配置如下参数。

参数	说明
自定义账号	请自定义账号名称。最大长度为57个字符，由小写字母，数字和下划线组成。
选择成员角色	请选择用户类型。 • 实例超级管理员（Superuser）：拥有实例的所有权限，如果子账号被授予为Superuser，则无需再额外进行其他授权。 • 普通用户：仅被创建至实例中，没有实例内任何对象（例如数据库、Schema和表等）的查看及操作权限，需要授予相应的权限，才可以查看或操作实例。
密码	请设置密码，密码需符合如下要求。 • 由大写字母、小写字母、数字、特殊字符其中三种及以上组成。 • 密码长度为8~32个字符。 • 支持的特殊字符包含 `!@#$%^&*()_+-=` 。
确认密码	请再次输入密码。

3. 单击**确认**，完成创建自定义用户。

常见问题

- 问题现象

  自定义账号访问MaxCompute外部表时出现如下报错。

```
ERROR:  Query:[xxxxxx] Build desc failed: failed to check permission: Authorization Faile
d [4002], You don't exist in project hologres_test. Context ID:xxxxxx-xxxx-xxxx-xxxx-xxxx
xxx.    --->Tips: Pricipal:INVALID$BASIC$xxx; You don't exist in project xxx
```

- 问题原因

  创建的自定义账号仅存在Hologres内部，所以默认情况下无法访问MaxCompute外部表。

- 解决方法

  请创建 `User Mapping` ，即将一个自定义账号绑定至一个拥有MaxCompute对应项目和Hologres内部表数据权限的阿里云RAM账号。

  - 语法示例

```
CREATE USER MAPPING
FOR "<user_name>"
SERVER odps_server
OPTIONS
(
    access_id '<Access_id>',
    access_key '<Access_key>'
);
```

  - 参数说明

参数	说明
user_name	自定义账户的用户名。
Access_id	具有当前数据库登录权限账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
Access_key	具有当前数据库登录权限账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。

- 使用示例

```
--为用户BASIC$test创建USER MAPPING
CREATE USER MAPPING
FOR "BASIC$test"
SERVER odps_server
OPTIONS (
  access_id 'LTxxxxxxxxxx',
  access_key 'y8xxxxxxxxxxxxx');
--为当前用户创建USER MAPPING
CREATE USER MAPPING
FOR CURRENT_USER
SERVER odps_server
OPTIONS (
  access_id 'LTxxxxxxxxxx',
  access_key 'y8xxxxxxxxxxxxx');
```

3.3.6.2. DB管理

本文为您介绍在交互式分析Hologres管理控制台的Database管理页面，如何创建并删除数据库，并为数据库选择对应的权限策略。

授权数据库

您可以在DB管理页面，使用可视化方式创建数据库。

1. 在**DB授权**页面选择目标实例名称，在页面右上角单击**新增数据库**。
2. 在**新增数据库**对话框，选择目标实例名称，输入**数据库名称**，并根据业务需求选择**简单权限策略**。

为了简化授权操作，建议您开启简单权限模型（SPM）。

Hologres为您提供的方便且完善的授权体系如下：

- 专家模式：Hologres兼容PostgreSQL，使用与Postgres完全一致的授权体系（简称专家模式）。您可以按照标准的PostgreSQL授权语句为子账号授权，详情请参见专家权限模型。
- 简单权限模型（SPM）：Hologres在PostgreSQL的基础上，为提升您的体验，抽象出一种粗粒度的简单权限模型，通过将用户加入用户组的方式完成授权，详情请参见简单权限模型概述。

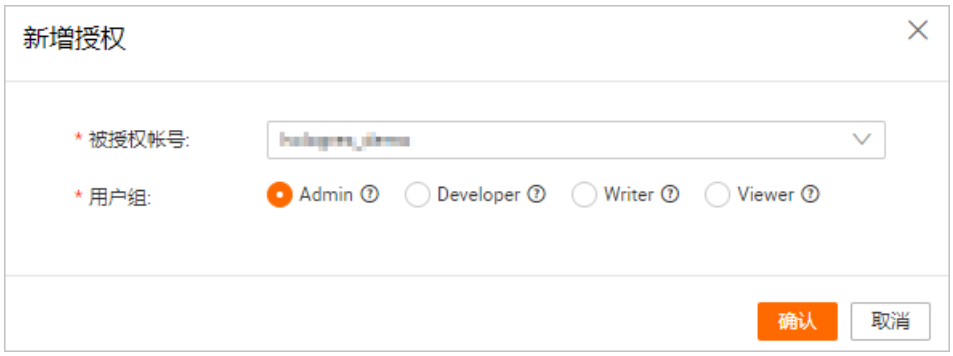
- 基于Schema的简单权限模型（SLPM）：该权限模型基于Schema划分，相比于简单权限模型更为细粒度，若是对权限有严格划分且又希望授权操作简便可以使用该权限模型，详情请参见基于Schema级别的简单权限模型概述。

> **说明** 成功购买实例后，系统默认生成一个名为postgres的数据库，该数据库分配的资源较少，仅用于管理，不会显示在DB管理页面。处理实际业务请您新建数据库。

用户授权

新建的数据库开启简单权限模型后，您可以在管理控制台，使用可视化方式为子账号授权，步骤如下：

1. 在**DB管理**页面，单击目标数据库**操作**列的**用户授权**，进入**权限管理**页面。
2. 在目标数据库的**用户授权**页面，单击**新增授权**。
3. 在**新增授权**对话框，选择**被授权账号**及**用户组**。



用户组取值如下：

- Admin：数据库的Owner，可以访问并操作数据库的所有对象，管理数据库的所有用户组。
- Developer：可以使用DDL语句创建、删除或修改数据库中对象的定义，以及读写数据库对象中的数据。
- Writer：可以读写数据库对象中的数据。
- Viewer：拥有所有数据库对象的只读权限。

4. 单击**确定**。新账号就可以使用开发工具连接当前数据库进行数据开发。

> **说明** ：如果在**被授权账号**的下拉列表找不到对应的账号，则说明该账号并未添加至当前实例，您需要前往**用户管理**页面添加用户。

撤销授权

如果您当前使用的数据库开启了简单权限模型，需要使用可视化方式对子账号撤销授权，步骤如下：

1. 在**DB授权**页面，单击目标数据库**操作**列的**用户授权**，进入**用户授权**页面。
2. 在目标用户的**操作**列，单击**撤销授权**。

3. 单击**确定**。

   关于撤销权限后RAM用户拥有的权限，详情请参见简单权限模型的使用。

删除数据库

在**DB授权**页面，您可以单击目标数据库**操作**列的**删除**，直接删除数据库。删除数据库后，数据库中的数据将会同时删除，并且不能恢复。

3.3.6.3. 资源组管理

通过资源组管理，您可以将实例计算资源分成多个资源组进行管理和应用。本文介绍了如何使用HoloWeb新建资源组、删除资源组、修改资源组配额，以及在资源组中解除或绑定用户。

背景信息

Hologres V1.0及以下版本支持在实例间进行资源隔离，不支持对于实例内部进行更细粒度的用户级别的资源隔离。但是在实际生产环境中，往往需要根据用户在实例内部进行资源隔离，限制每个用户使用的资源上限，以保证用户之间的作业互不影响。为满足上述细粒度的资源隔离诉求，Hologres新增支持使用资源组来助力您管理Hologres实例内的计算资源，实现资源隔离。现在可以使用HoloWeb图形化的对资源组进行管理。

使用限制

- 仅Hologres V1.1及以上版本支持使用资源组管理Hologres实例内的计算资源，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅限具备Superuser权限的用户使用资源组管理Hologres实例内计算资源，否则系统会提示权限不足。
- 计算资源属于实例级别，如果用户有多个数据库，所有数据库共享同一个实例的计算资源，所有数据库共享同一份资源分配方案。

新建资源组

使用HoloWeb可视化方式创建资源组。

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
3. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
4. 在**资源组管理**页面，选择目标实例名称，单击**新增资源组**。

5. 在**新增资源组**对话框，输入**资源组名称**并设置**资源组配额**，单击**确认**，即可完成资源组的新建。

> **说明** 一个Hologres实例内所有资源组配额总和不能超过1，否则系统会报错。

删除资源组

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
3. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
4. 在**资源组管理**页面，单击对应资源组操作列的**删除**，进行资源组的删除。

> **说明** 如果有用户绑定到资源组，则该资源组不能被删除。

调整资源组配额

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
3. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
4. 在**资源组管理**页面目标资源组的资源组配额列，单击**调整配额**。
5. 在**调整配额**对话框，调整资源组配额，单击**确认**。

绑定用户

创建资源组后，您可以使用HoloWeb，可视化的给资源组绑定用户。

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
3. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
4. 在**资源组管理**页面，单击对应资源组操作列的**绑定用户**。
5. 在**绑定资源组**页面，单击**新增绑定用户**。
6. 在**绑定用户**对话框，选择用户，单击**确认**。

> **说明**
>
> - 如果在用户的下拉列表找不到对应的账号，则说明该账号并未添加至当前实例，您需要前往用户管理页面添加用户。
> - 一个用户仅能被绑定一个资源组，若重复绑定用户，以最新绑定的资源组为准。

解绑用户

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。

3. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
4. 在**资源组管理**页面，单击对应资源组操作列的**绑定用户**。
5. 在**绑定资源组**页面，单击对应用户操作列的**解绑用户**。
6. 在**解绑用户**对话框，单击**确认**。

相关SQL

单实例计算资源隔离（Beta）。

3.4. PSQL客户端

Hologres兼容PostgreSQL生态，这意味着大多数PostgreSQL兼容的开发工具或BI工具都能直接连接Hologres，您可以选择熟悉的工具进行开发，帮助您快速构建企业级实时数仓。本文为您介绍PSQL客户端如何连接Hologres，并使用标准的PostgreSQL语句进行数据开发。

安装PSQL客户端

在使用PSQL客户端之前需要官网下载并安装。若您已经安装好PSQL客户端，可忽略本步骤，安装步骤如下。

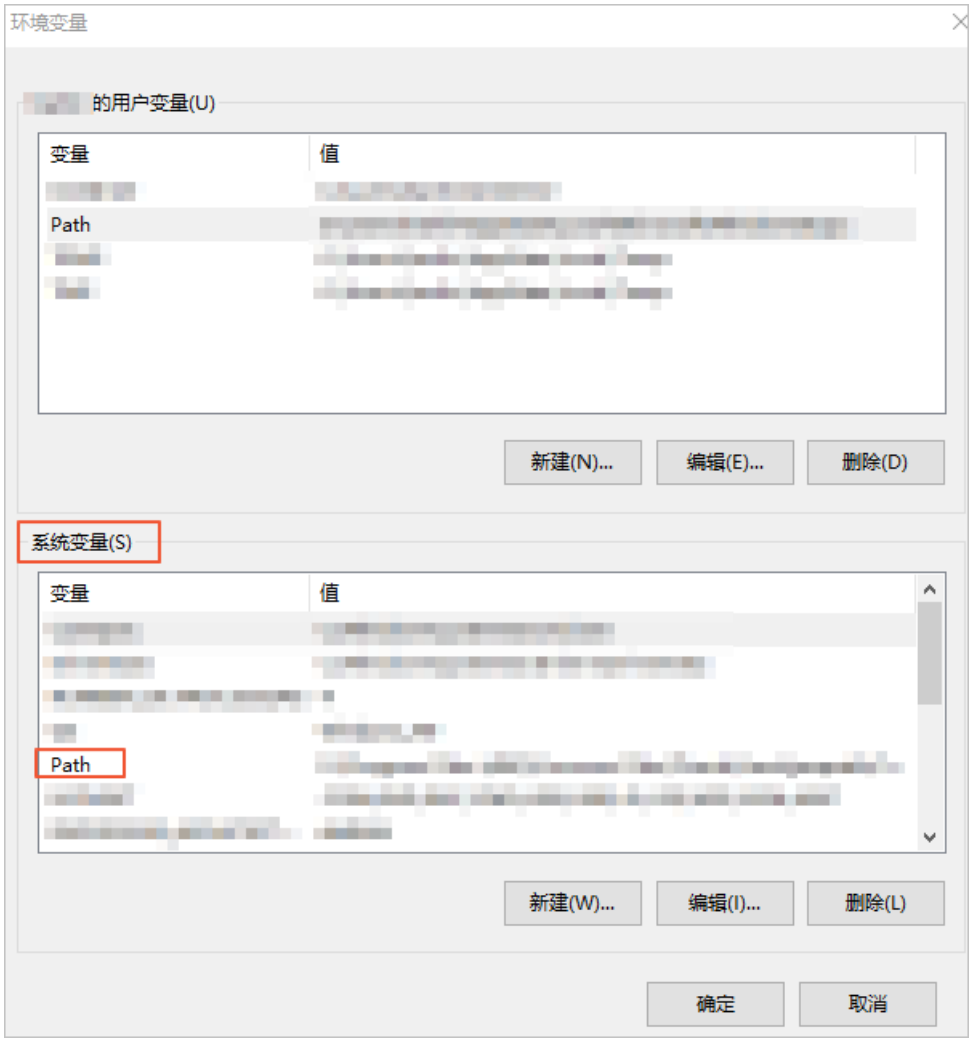
1. 下载PSQL客户端

   您需要进入Postgres官网，下载与电脑系统相匹配的PostgreSQL 11及以上版本的客户端安装包，并根据提示安装。

2. 设置环境变量
   - Windows系统。

a. 在**系统属性 > 高级系统设置**界面，单击**环境变量**。

b. 添加PostgreSQL的bin文件路径至Path中。



c. 单击**确定**。

- 设置macOS系统的环境变量，一般无需设置环境变量，如果需要请参见设置环境变量。

连接Hologres并开发

下载安装完成PSQL客户端之后，可以连接Hologres实例并进行开发。

1. 连接Hologres

   进入PSQL客户端命令行界面，输入连接信息，语法与连接PostgreSQL数据库一致。

   - Linux系统语句如下。

     ```
     psql -h <Endpoint> -p <Port> -U <AccessKey ID> -d <Database>
     ```

     执行完上述语句后，您需要输入AccessKey Secret。

   - macOS系统语句如下。

```
PGUSER=<AccessKey ID> PGPASSWORD=<AccessKey Secret> psql -p <Port> -h <Endpoint> -d <
Database>
```

```
IT-C1MQG4CVG943:~ zhm-may$ PGUSER=                PGPASSWORD=
psql -p 80 -h                                    .hologres.aliyuncs.com -d mydb
psql (11.4, server 11.3)
Type "help" for help.

mydb=#
```

- Windows系统语句如下。

```
Server [localhost]: Endpoint
Database [postgres]: Database
Port [5432]: Port
Username [postgres]: <AccessKey ID>
用户 <AccessKey ID> 的口令：<AccessKey Secret>
```

参数	描述
AccessKey ID	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
AccessKey Secret	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
Port	Hologres实例的公共网络端口。 示例取值 `80` 。
Endpoint	Hologres实例的公共网络地址。 示例取值 `xxx-cn-hangzhou.hologres.aliyuncs.com` 。

参数	描述
Database	Hologres的数据库名称。 开通Hologres实例后，系统自动创建**postgres**数据库。 您可以使用**postgres**数据库链接Hologres，但是该数据库分配到的资源较少，开发实际业务建议您新建数据库。详情请参见创建数据库。 示例取值`mydb`。

> **说明** 您也可以使用熟悉的开发工具连接Hologres，例如使用DataWorks或HoloWeb进行数据开发，详情请参见DataWorks快速入门或连接HoloWeb。

2. （可选）创建数据库

   开通Hologres实例后，系统自动创建postgres数据库。该数据库分配到的资源较少，仅用于运维管理，开发实际业务建议您新建数据库。

   > **说明** 若您在此之前已经创建业务数据库，可忽略此步骤。

   - 命令语法。

     ```
     CREATE Database <DatabaseName>;
     ```

     DatabaseName为要创建的数据库名称。

   - 使用示例。

     ```
     --创建一个名为test的数据库。
     CREATE Database test;
     ```

3. 数据开发

   使用标准的PostgreSQL语句，在PSQL客户端进行数据开发。

   示例在数据库中创建一张表并写入数据，SQL语句如下。

```
BEGIN;
CREATE TABLE nation (
  n_nationkey bigint NOT NULL,
  n_name text NOT NULL,
  n_regionkey bigint NOT NULL,
  n_comment text NOT NULL,
  PRIMARY KEY (n_nationkey)
);
CALL SET_TABLE_PROPERTY('nation', 'bitmap_columns', 'n_nationkey,n_name,n_regionkey');
CALL SET_TABLE_PROPERTY('nation', 'dictionary_encoding_columns', 'n_name,n_comment');
CALL SET_TABLE_PROPERTY('nation', 'time_to_live_in_seconds', '31536000');
COMMIT;
INSERT INTO nation VALUES
(11,'zRAQ', 4,'nic deposits boost atop the quickly final requests? quickly regula'),
(22,'RUSSIA', 3  ,'requests against the platelets use never according to the quickly re
gular pint'),
(2,'BRAZIL',  1 ,'y alongside of the pending deposits. carefully special packages are a
bout the ironic forges. slyly special '),
(5,'ETHIOPIA',  0 ,'ven packages wake quickly. regu'),
(9,'INDONESIA', 2  ,'slyly express asymptotes. regular deposits haggle slyly. carefully
ironic hockey players sleep blithely. carefull'),
(14,'KENYA',  0  ,'pending excuses haggle furiously deposits. pending, express pinto be
ans wake fluffily past t'),
(3,'CANADA',  1 ,'eas hang ironic, silent packages. slyly regular packages are furiousl
y over the tithes. fluffily bold'),
(4,'EGYPT', 4 ,'y above the carefully unusual theodolites. final dugouts are quickly ac
ross the furiously regular d'),
(7,'GERMANY', 3 ,'l platelets. regular accounts x-ray: unusual, regular acco'),
(20 ,'SAUDI ARABIA',  4 ,'ts. silent requests haggle. closely express packages sleep ac
ross the blithely');
SELECT * FROM nation;
```

您可以根据业务场景进行作业开发，示例如下。

- 加速读取MaxCompute数据，详情请参见通过创建外部表加速查询MaxCompute数据。
- 通过Flink实时写入数据至Hologres，详情请参见Hologres结果表。

3.5. JDBC

Hologres为您提供完全兼容PostgreSQL的连接（JDBC/ODBC）接口，您可以通过该接口将SQL客户端工具连接至Hologres。本文为您介绍JDBC如何连接Hologres进行数据开发。

注意事项

- 通过JDBC写入数据至Hologres需要使用42.2.25及以上的Postgres JDBC Driver。
- 使用JDBC连接Hologres后，如果您需要测试写入数据的性能，建议使用VPC网络。公共网络由于自身原因，无法达到性能测试的目标。
- Hologres当前不支持在一个事务中多次写入，因此需要设置 `autoCommit` 为 `true` （JDBC的autoCommit默认为true），不要显式的在代码里面进行Commit操作。如果出现 `ERROR: INSERT in transaction is not supported now` 报错，则需要设置 `autoCommit` 为 `true` ，如下所示。

```
Connection conn = DriverManager.getConnection(url, user, password);
conn.setAutoCommit(true);
```

通过JDBC连接Hologres

通过JDBC连接Hologres的步骤如下。

1. 下载配置。

   客户端工具默认集成了PostgreSQL数据库的驱动，直接使用工具自带的驱动即可。如果未集成驱动，需要下载并安装。

   使用PostgreSQL驱动，请至官网下载PostgreSQL JDBC Driver，需要使用42.2.25以上版本的JDBC驱动，建议您使用最新版本的JDBC。下载成功后需要至Maven仓库配置示例如下。

   ```
   <dependencies>
           <dependency>
               <groupId>org.postgresql</groupId>
               <artifactId>postgresql</artifactId>
               <version>42.2.25</version>
           </dependency>
   </dependencies>
   ```

2. 连接Hologres。

   - 使用以下连接串连接Hologres。

     ```
     jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?user={ACCESS_ID}&password={ACCESS_KEY}
     ```

   - 参数说明如下。

参数	描述
Endpoint	Hologres实例的网络地址。 进入Hologres管理控制台的**实例详情**页获取网络地址。
Port	Hologres实例的端口。 进入Hologres管理控制台的**实例详情**页获取端口。
DBNAME	Hologres创建的数据库名称。
AccessKey ID	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
AccessKey Secret	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。

   - 连接Hologres建议如下。

- 建议JDBC的URL加上ApplicationName参数，此参数为可选参数。使用该方式连接数据库时有助于您在慢Query清单中根据ApplicationName快速定位您的发起请求的应用，连接串如下。

```
jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?user={ACCESS_ID}&password={ACCESS_KEY}
&ApplicationName={APPLICATION_NAME}
```

- 建议在JDBC URL中添加 `reWriteBatchedInserts=true` 配置，系统会以批量写入的方式提交作业，性能更好，连接串如下。

```
jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?ApplicationName={APPLICATION_NAME}&reW
riteBatchedInserts=true
```

- 建议使用Prepared Statement方式来进行数据读取和写入，以实现更高的吞吐。

- 连接示例如下。

```
public class HologresTest {
    private void jdbcExample() throws SQLException {
        String url = "jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?user={ACCESS_ID}&p
assword={ACCESS_KEY}&ApplicationName={APPLICATION_NAME}";
        try (Connection conn = DriverManager.getConnection(url)) {
            try (Statement st = conn.createStatement()) {
                String sql = "SELECT * FROM table where xxx limit 100";
                try (ResultSet rs = st.executeQuery(sql)) {
                    while (rs.next()) {
                        //获取数据表中第一列数据值
                        String c1 = rs.getString(1);
                    }
                }
            }
        }
    }
    private void jdbcPreparedStmtExample() throws SQLException {
        String url = "jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?user={ACCESS_ID}&p
assword={ACCESS_KEY}&ApplicationName={APPLICATION_NAME}";
        try (Connection conn = DriverManager.getConnection(url)) {
            String sql = "insert into test values" +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?), " +
                    "(?, ?)";
            try (PreparedStatement st = conn.prepareStatement(sql)) {
                for (int i = 0; i < 10; ++i) {
                    for (int j = 0; j < 2 * 10; ++j) {
                        st.setString(j + 1, UUID.randomUUID().toString());
                    }
                    System.out.println("affected row => " + st.executeUpdate());
                }
            }
        }
    }
 }
```

使用JDBC开发

JDBC连接Hologres成功后，您可以使用标准的开发语句来开发Hologres，包括写入和读取数据。

- 写入数据

  您可以通过JDBC的Statement或Prepared Statement模式写入数据。通常情况下，推荐您使用Prepared Statment模式，并且设置批量写入数据的条数为256的倍数（建议最低设置为256条）。因为使用Prepared Statment模式时，服务端会缓存SQL编译的结果，从而降低写入数据的延时，并提高吞吐量。

  使用Prepared Statement模式写入数据的示例如下。

- 使用Prepared Statement模式批量写入数据，命令语句如下。

```
/*通过Prepared Statement模式批量写入数据*/
/*该示例中，设置的批量写入大小batchSize为256*/
private static void WriteBatchWithPreparedStatement(Connection conn) throws Exception {
    try (PreparedStatement stmt = conn.prepareStatement("insert into test_tb values (?,
?,?,?)")) {
        int batchSize = 256;
        for (int i = 0; i < batchSize; ++i) {
            stmt.setInt( 1, 1000 + i);
            stmt.setString( 2, "1");
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
            Date parsedDate = dateFormat.parse("1990-11-11 00:00:00");
            stmt.setTimestamp( 3, new java.sql.Timestamp(parsedDate.getTime()));
            stmt.setDouble( 4 , 0.1 );
            stmt.addBatch();
        }
        stmt.executeBatch();
    }
}
```

- 使用Prepared Statement模式写入数据的同时，添加Postgres的 `INSERT ON CONFLICT` 功能，实现写入时更新覆盖原有数据。命令语句如下。

> **说明** 使用INSERT ON CONFLICT语句时目标表必须定义主键。

```
private static void InsertOverwrite(Connection conn) throws Exception {
    try (PreparedStatement stmt = conn.prepareStatement("insert into test_tb values (?,
?,?,?), (?,?,?,?), (?,?,?,?), (?,?,?,?), (?,?,?,?), (?,?,?,?) on conflict(pk) do update
set f1 = excluded.f1, f2 = excluded.f2, f3 = excluded.f3")) {
        int batchSize = 6;
        for (int i = 0; i < batchSize; ++i) {
            stmt.setInt(i * 4 + 1, i);
            stmt.setString(i * 4 + 2, "1");
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
            Date parsedDate = dateFormat.parse("1990-11-11 00:00:00");
            stmt.setTimestamp(i * 4 + 3, new java.sql.Timestamp(parsedDate.getTime()));
            stmt.setDouble(i * 4 + 4, 0.1);
        }
        int affected_rows = stmt.executeUpdate();
        System.out.println("affected rows => " + affected_rows);
    }
}
```

- 数据查询

  数据写入完成之后，可以对数据进行查询。您也可以根据业务需求查询已有表的数据。

Druid连接池配置

- 注意事项
  - 建议配置 `keepAlive=true` ，可以复用连接和避免短链接。
  - 您需使用Druid 1.1.12以上的版本连接Hologres。

- 配置Druid连接池

  > **说明** 其中*initialSize*、*minIdle*和*maxActive*请根据实例大小和实际业务进行设置。

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" d
estroy-method="close">
  <!-- jdbc_url是Hologres实例的连接地址URL，可以在控制台的实例配置页面获取连接URL。 -->
  <property name="url" value="${jdbc_url}" />
  <!-- jdbc_user是Hologres实例中的用户账户的Access ID。 -->
  <property name="username" value="${jdbc_user}" />
  <!-- jdbc_password是Hologres实例中用户账号对应的Access Secret。 -->
  <property name="password" value="${jdbc_password}" />
  <!-- 配置初始化连接池大小、最小连接数、最大连接数。 -->
  <property name="initialSize" value="5" />
  <property name="minIdle" value="10" />
  <property name="maxActive" value="20" />
  <!-- 配置获取连接等待超时的时间。 -->
  <property name="maxWait" value="60000" />
  <!-- 配置间隔多久进行一次检测，检测需要关闭的空闲连接，单位毫秒。 -->
  <property name="timeBetweenEvictionRunsMillis" value="2000" />
  <!-- 配置一个连接在连接池中的最小生存时间，单位毫秒。 -->
  <property name="minEvictableIdleTimeMillis" value="600000" />
  <property name="maxEvictableIdleTimeMillis" value="900000" />
  <property name="validationQuery" value="select 1" />
  <property name="testWhileIdle" value="true" />
  <!-- 配置从连接池获取连接时，是否检查连接有效性，true每次都检查；false不检查。 -->
  <property name="testOnBorrow" value="false" />
  <!-- 配置向连接池归还连接时，是否检查连接有效性，true每次都检查；false不检查。 -->
  <property name="testOnReturn" value="false" />
  <property name="keepAlive" value="true" />
  <property name="phyMaxUseCount" value="100000" />
  <!-- 配置监控统计拦截的filters。 -->
  <property name="filters" value="stat" />
</bean>
```

性能调优实践

使用JDBC时想要达到比较好的性能，有以下需要注意的事项。

- 尽量使用VPC网络，避免使用公网，已避免公网带来的网络开销。
- 通过JDBC驱动写入数据时，JDBC URL中添加reWriteBatchedInserts=true配置，系统会以批量写入的方式提交作业，性能更好。实践证明攒批配置256的倍数（建议最低设置为256条）效果会更好。同时您也可以使用Hologres的Holo Client会自动攒批。

```
jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}?ApplicationName={APPLICATION_NAME}&reWriteBa
tchedInserts=true
```

- 使用Prepared Statment模式，此模式下服务端会缓存SQL编译的结果，从而降低写入数据的延时，并提高吞吐量。

3.6. 通过Holo Client读写数据

本文将为您介绍Holo Client的使用。

Holo Client介绍

随着业务发展，企业数据量也迅猛增加，为了更高效的支持大批量数据的写入（批量、实时同步至Hologres),以及支持高QPS的点查（维表关联）场景，Hologres在JDBC的基础上自研了一款开发接口Holo Client。

Holo Client优势以及使用场景如下。

- 自动攒批，实现大批量数据高性能批量、实时写入以及高QPS基于主键的查询（点查）、删除（DELETE）以及更新（UPDATE），但OLAP查询更建议使用JDBC接口。
- 针对分区表场景，写入时能自动路由对应分区，减少分区提前创建的复杂操作。
- 同JDBC接口一致，支持订阅Hologres Binlog，实现数据的实时消费。

需要注意的是，Holo Client不是替代JDBC接口，而是丰富了JDBC接口不适配的新增功能，在JDBC适合的分析场景，请继续使用JDBC接口查询数据。

使用Holo Client

- Holo Client当前已经开源，使用详情请参见Holo Client。
- Holo Client每个版本都会发布到公网Maven仓库，版本信息请参见Holo Client Release。

3.7. 使用Python访问Hologres

Psycopg是Python用于操作PostgreSQL的库。Hologres兼容PostgreSQL 11，因此您可以通过psycopg访问Hologres。本文将指导您使用pyscopg2访问Hologres，示例使用的操作环境为基于CentOS 7系统的Python 3.8版本。

安装Python3.8

您可以基于Miniconda、Anaconda安装Python 3.8环境。如下内容以CentOS 7系统为例，安装Python 3.8版本。

1. 安装Python 3.8。

   您可以下载对应版本的Python，执行如下命令进行安装。

   ```
   # yum install centos-release-scl
   # yum install rh-python38
   # scl enable rh-python38 bash
   # python --version
   Python 3.8.6
   ```

2. 安装psycopg2模块。

   执行如下命令安装psycopg2模块。

   ```
   # pip install psycopg2-binary
   ```

连接Hologres

Python3.8环境和psycopg2安装完成之后，您可以执行如下操作并连接Hologres。

1. 加载psycopg2。

   如果需要使用psycopg2，您可以执行命令 `import psycopg2` 加载安装的psycopy2。

2. 创建数据库连接。

您可以使用 `psycopg2.connect()` 函数连接Hologres，具体语法和参数说明如下所示。

```
conn = psycopg2.connect(host="<Endpoint>", port=<Port>, dbname="<database>", user="<Access ID>", password="<Access Key>", application_name="<Application Name>")
```

参数	描述
host	Hologres实例的网络地址。 进入Hologres管理控制台的实例详情页，从**实例配置**获取网络地址。
port	Hologres的实例端口。 进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
dbname	Hologres创建的数据库名称。
user	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
password	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
application_name	应用名，可选。 用于记录查询日志时识别SQL代表的应用含义。

使用Hologres

当您成功连接Hologres数据库之后，即可通过psycopg2进行数据开发操作。如下内容将指导您创建表、插入数据、查询和释放资源等操作。如果需要使用Fixed Plan能力实现更高性能的读写操作，需要配置相关GUC参数，请参见Fixed Plan加速SQL执行。

1. 创建游标。

   在进行数据开发之前，您需要执行命令 `cur = conn.cursor()` 来创建连接的游标。

2. 数据开发。

   i. 创建表

   您可以执行如下命令，创建一个表 `holo_test` 并定义表的数据类型为integer。您也可以根据业务需求定义表名称和数据类型。

   ```
   cur.execute("CREATE TABLE holo_test (num integer);")
   ```

   ii. 插入数据

   您可以执行如下命令，为创建的表 `holo_test` 插入数据1~1000。

   ```
   cur.execute("INSERT INTO holo_test SELECT generate_series(%s, %s)", (1, 1000))
   ```

iii. 查询数据

```
cur.execute("SELECT sum(num) FROM holo_test;")
cur.fetchone()
```

3. 提交事务。

   在查询数据的命令之后，您需要执行命令 `conn.commit()` 提交事务，此操作可以确保操作已经提交。也可以把autocommit参数设置为true，实现SQL命令的自动提交。

4. 释放资源。

   为避免影响后续的操作，当操作执行完成后，您需要执行如下命令关闭游标并断开数据库连接。

```
cur.close()
conn.close()
```

Pandas DataFrame快速写入Hologres最佳实践

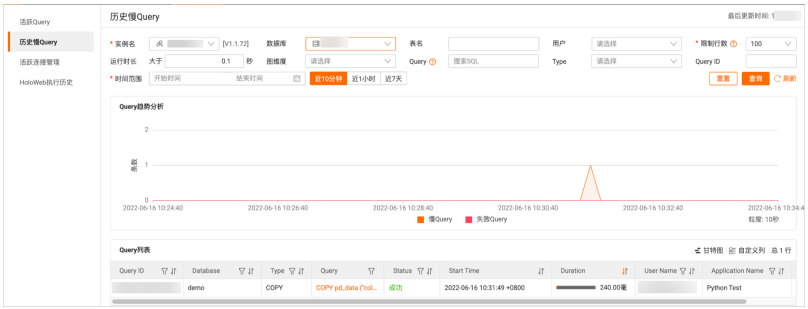
使用Python时，经常会使用Pandas将数据转换为DataFrame，并对DataFrame进行处理，最终将DataFrame导入Hologres，此时希望将DataFrame快速导入Hologres。导入时候常用 `to_sql` 函数，详情请参见Pandas。

推荐使用 `to_sql` 函数的callable方式，使用copy方式导入数据，样例的Python代码如下。

```
# 加载依赖
import pandas as pd
import psycopg2
# 生成连接字符串
host="hgpostcn-cn-xxxxxx-cn-hangzhou.hologres.aliyuncs.com"
port=80
dbname="demo"
user="LTAI5xxxxx"
password="fa8Kdgxxxxx"
application_name="Python Test"
conn = "postgresql+psycopg2://{}:{}@{}:{}/{}?application_name={}".format(user, password, ho
st, port, dbname,application_name)
print(conn)
# 生成dataframe
data = [('1','1','1'),('2','2','2')]
cols = ('col1','col2','col3')
pd_data = pd.DataFrame(data, columns=cols)
# 定义callable函数
import csv
from io import StringIO
def psql_insert_copy(table, conn, keys, data_iter):
    """
    Execute SQL statement inserting data
    Parameters
    ----------
    table : pandas.io.sql.SQLTable
    conn : sqlalchemy.engine.Engine or sqlalchemy.engine.Connection
    keys : list of str
        Column names
    data_iter : Iterable that iterates the values to be inserted
    """
    # gets a DBAPI connection that can provide a cursor
```

```
    dbapi_conn = conn.connection
    with dbapi_conn.cursor() as cur:
        s_buf = StringIO()
        writer = csv.writer(s_buf)
        writer.writerows(data_iter)
        s_buf.seek(0)
        columns = ', '.join('"{}"'.format(k) for k in keys)
        if table.schema:
            table_name = '{}.{}'.format(table.schema, table.name)
        else:
            table_name = table.name
        sql = 'COPY {} ({}) FROM STDIN WITH CSV'.format(
            table_name, columns)
        cur.copy_expert(sql=sql, file=s_buf)
# 导入数据
pd_data.to_sql(
    name="pd_data",
    con=conn,
    if_exists="replace",
    index=False,
    method=psql_insert_copy
)
```

查看历史查询，验证已经使用COPY方式写入数据至Hologres。



3.8. DataGrip

DataGrip是一个多连接数据库开发工具，可让您在一个应用程序中连接多个数据库，帮助您快速轻松地创建、管理和维护数据库。本文为您介绍DataGrip相关产品如何连接Hologres。

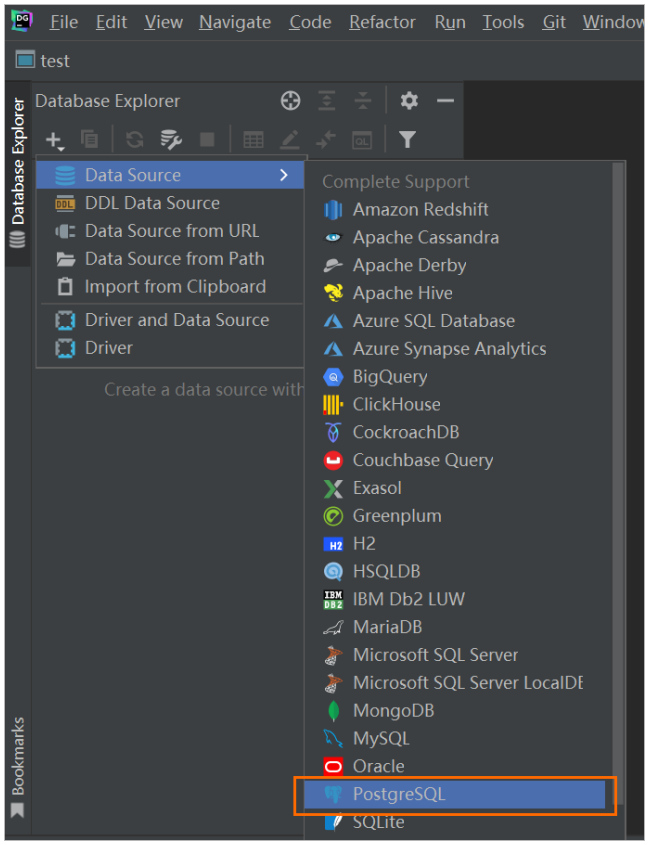
前提条件

- 已安装DataGrip客户端，如果未安装请单击DataGrip进行安装。
- 开通Hologres实例，详情请参见购买Hologres。

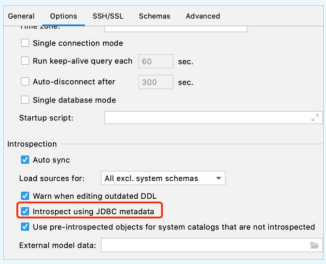
操作步骤

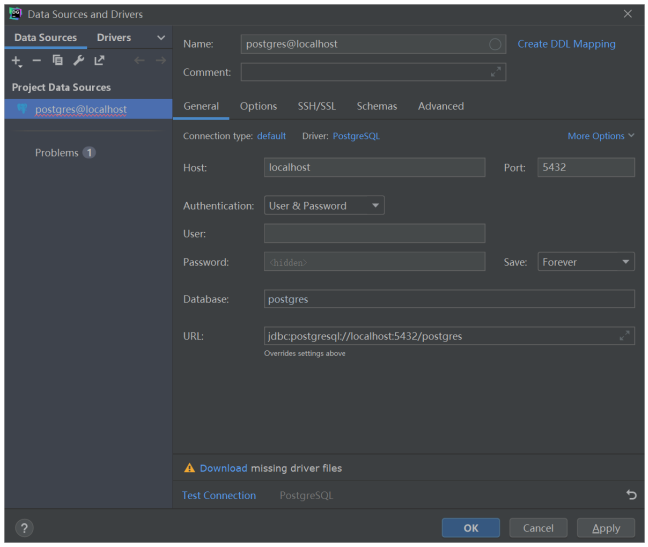
Datagrip客户端连接Hologres步骤如下。

1. 打开DataGrip客户端，单击客户端**Database Explorer**菜单栏的 图标。
2. 选择**DataSource > PostgreSQL**。



3. 在Data Sources and Drivers对话框，配置如下参数。

> **说明**
>
> - 出现 `missing driver files` 的提示，请先点击Download下载驱动。
> - 使用老版本的Datagrip客户端，请确保勾选Options页签下Introspection选项中的Introspect using JDBC metadata选项。
>
> 
>



参数	描述	示例
Name	自定义的连接名称。	无
Host	连接的Hologres公共网络地址。进入Hologres管理控制台的**实例详情**页，在**网络信息**获取网络地址。	holodemo-cn-hangzhou.hologres.aliyuncs.com
Port	连接的Hologres公共网络端口。进入Hologres管理控制台的**实例详情**页，在**网络信息**获取端口号。	80
Database	需要连接的Hologres数据库名称。进入Hologres管理控制台的**实例列表**页，在**数据库管理**页面获取数据库名称。	postgres
User	当前Hologres账号的AccessKey ID。您可以单击AccessKey 管理，获取用户的AccessKey ID。	无
Password	当前Hologres账号的AccessKey Secret。您可以单击AccessKey 管理管理，获取AccessKey Secret。	无
Test Connection	单击**Test Connection**，弹出连接成功对话框，则表示DataGrip成功连接Hologres。	无

4. 单击**OK**，完成DataGrip客户端连接Hologres。

   DataGrip成功连接Hologres后，您可以进行数据开发，详情请参见DataGrip帮助文档。

   Hologres的典型应用场景如下：

   - 直接加速查询离线数据，详情请参见通过创建外部表加速查询MaxCompute数据。
   - 实时写入数据，详情请参见Hologres结果表。

3.9. Navicat

Navicat可以在一个应用程序中连接多个数据库，帮助您轻松创建、管理和维护数据库。本文以Navicat for PostgreSQL为例，为您介绍Navicat如何连接Hologres并进行数据开发。

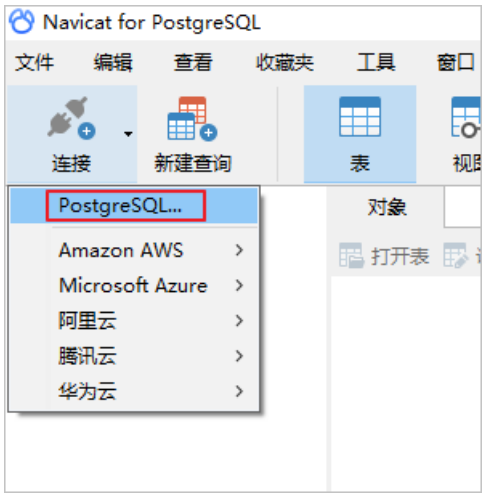
前提条件

- 单击Navicat，下载并安装Navicat客户端。本次示例使用Navicat 15 for PostgreSQL。
- 开通Hologres实例，详情请参见购买Hologres。

操作步骤

1. 新建至Hologres的连接。

   在Navicat客户端，单击顶部菜单栏的**连接**，选择**PostgreSQL**。



2. 配置**PostgreSQL-新建连接**对话框的参数。

参数说明如下表所示。

参数	描述	示例
连接名	自定义的连接名称。	无
主机	连接的Hologres公共网络地址。 进入Hologres管理控制台的实例详情页，从**实例配置**获取主机。	*holodemo-cn-hangzhou.hologres.aliyuncs.com*
端口	连接的Hologres公共网络端口。 进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。	*80*
初始数据库	需要连接的Hologres数据库名称。 进入Hologres管理控制台的实例详情页，从**DB管理**获取初始化数据库。	*postgres*

参数	描述	示例
用户名	当前Hologres账号的AccessKey ID。 您可以单击AccessKey 管理，获取用户的AccessKey ID。	无
密码	当前Hologres账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。	无
测试连接	单击**测试连接**，弹出**连接成功**对话框，则表示Navicat成功连接Hologres。	无

3. 单击**确定**。
4. 进行数据开发。

   Navicat成功连接Hologres后，您可以进行数据开发，详情请参见Navicat帮助文档。Hologres的典型应用场景如下：

   - 直接加速查询离线数据，详情请参见通过创建外部表加速查询MaxCompute数据。
   - 实时写入实时数据，详情请参见Hologres结果表。

3.10. Apache Nifi

本文以一个示例为您介绍Apache Nifi如何连接Hologres。

背景信息

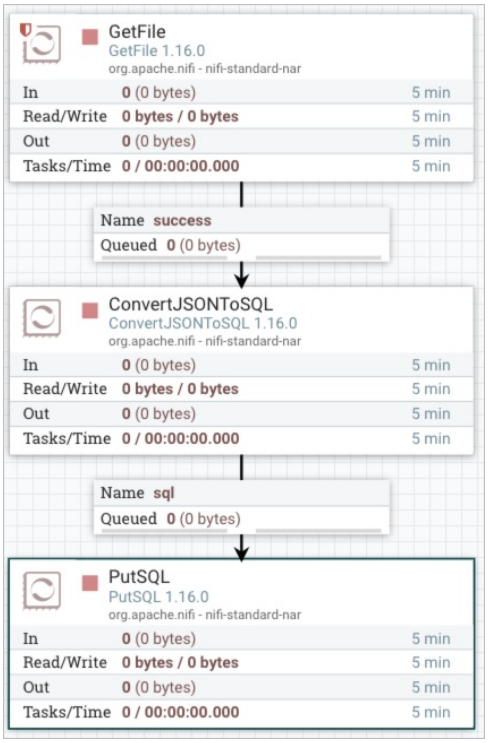
Apache NiFi是一个易用、可靠的数据处理与分发系统，Apache NiFi的设计目标是自动化管理系统间的数据流。Apache Nifi是一个基于WEB-UI用户界面，具有很强交互性和易用性，为不同系统间或系统内提供数据流管理与处理的系统。

前提条件

- 开通Hologres，详情请参见购买Hologres。
- 安装Apache Nifi，详情请参见How to install and start NiFi。

获取本地JSON文件写入Hologres

获取本地JSON文件写入Hologres的流程如下图所示。



1. GetFile：读取JSON格式的文件。
2. ConvertJSONToSQL：将JSON中的元素转化为SQL中的Insert语句。
3. PutSQL：执行上一个Processor生成的SQL语句，将JSON中的元素插入到数据库中。

1. 创建数据库和表
   i. 登录Hologres实例，并且建立数据库，命名为demo，详情请参见创建数据库。
   ii. 创建数据表

   使用如下SQL创建数据表，之后会将数据写入该表中。

```
DROP TABLE IF EXISTS user_info;
CREATE TABLE IF NOT EXISTS user_info (
    id int,
    first_name text,
    last_name text,
    email text
);
```

2. 配置GetFile Processor
   i. 添加一个GetFile Processor

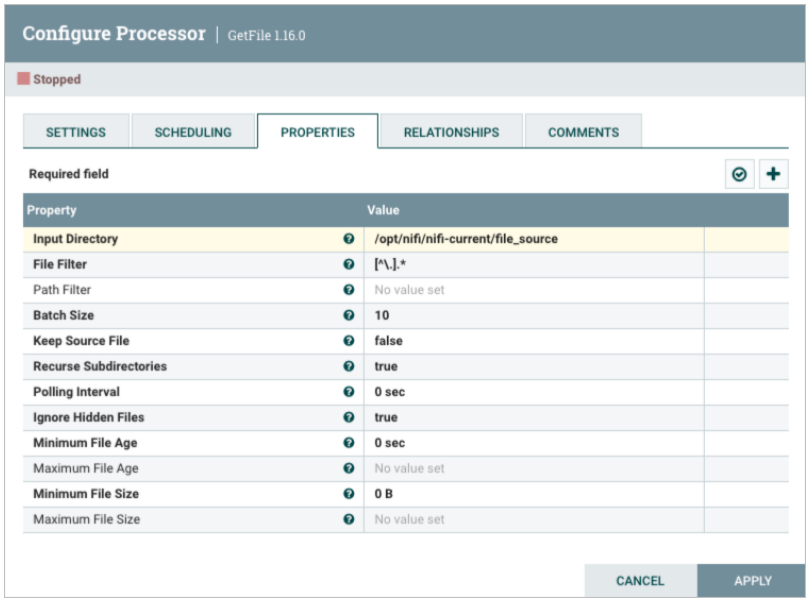
   详情请参见Adding a Processor。

ii. 填写JSON文件存放的地址

在PROPERTIES选项卡中的**Input Directory**中填写JSON文件存放的地址，例如我们将JSON文件存放在Apache Nifi服务器的 `/opt/nifi/nifi-current/file_source` 位置，文件名为user_info.json文件内容如下。

```
{
    "id": 1,
    "first_name": "Sig",
    "last_name": "Olivo",
    "email": "solivo0@blinklist.com"
}
```

配置样例如下所示。

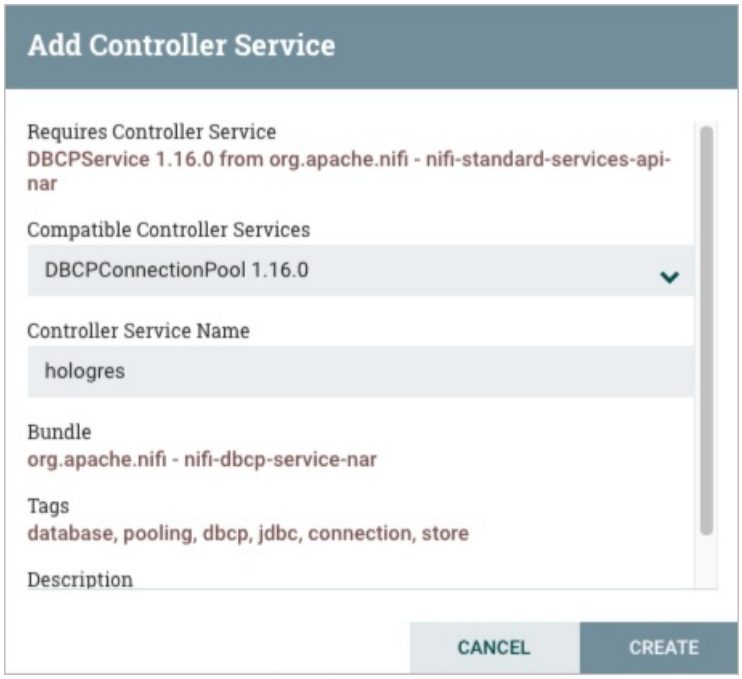


iii. 单击**APPLY**保存配置。

3. 配置ConvertJSONToSQL Processor

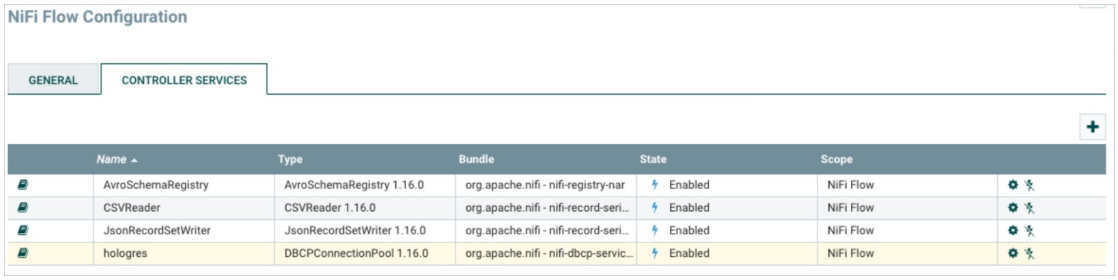
i. 添加一个ConvertJSONToSQL Processor。

ii. 在JDBC Connection Pool中增加一个Service，选择**Compatible Controller Services**为**DBCPConnectionPool**，并且设置**Controller Service Name**，此处设置为hologres。

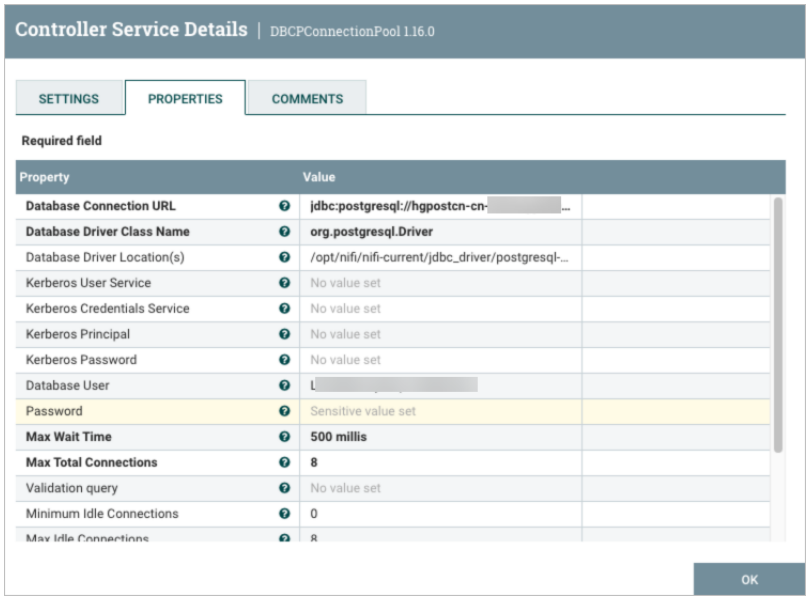


iii. 单击**JDBC Connection Pool**行最右侧的向右箭头按钮，开始配置连接字符串。

iv. 从中找到刚才创建的DBCPConnectionPool，单击设置按钮。



v. 在设置页面的**PROPERTIES**选项卡中配置如下参数。



参数名称	描述	说明
Database Connection URL	连接Hologres实例的JDBC连接字符串格式为：`jdbc:postgresql://<endpoint>/<database name>`，例如`jdbc:postgresql://hgpostcn-cn-xxxxxxxxxxx-cn-shanghai.hologres.aliyuncs.com:80/demo`。	其中Endpoint需要使用公网或者VPC的Endpoint，进入Hologres管理控制台的实例详情页获取Endpoint。
Database Driver Class Name	org.postgresql.Driver	不涉及
Database Driver Location(s)	PostgreSQL的JDBC存放地址，例如`/opt/nifi/nifi-current/jdbc_driver/postgresql-42.3.4.jar`。	您可以从PostgreSQL官网下载JDBC，推荐使用42.2.25以上版本的JDBC。
Database User	当前阿里云账号的AccessKey ID。	您可以单击AccessKey 管理，获取AccessKey ID。
Password	当前账号的AccessKey Secret。	您可以单击AccessKey 管理，获取AccessKey Secret。

vi. 单击**OK**完成配置。

vii. 单击**ENABLE**启动该服务。

viii. 回到最初的ConvertJSONToSQL Processor，选择修改如下参数，更多说明请参见官方手册。

参数名称	描述
Statement Type	需要执行的SQL语句，该示例中使用`INSERT`。
Table Name	需要写入数据的表名称，该示例中使用`user_info`。
Schema Name	需要写入数据的表的Schema名称，该示例中使用`public`。

ix. 单击**APPLY**完成配置。

4. 配置PutSQL Processor

   i. 添加一个PutSQL Processor。

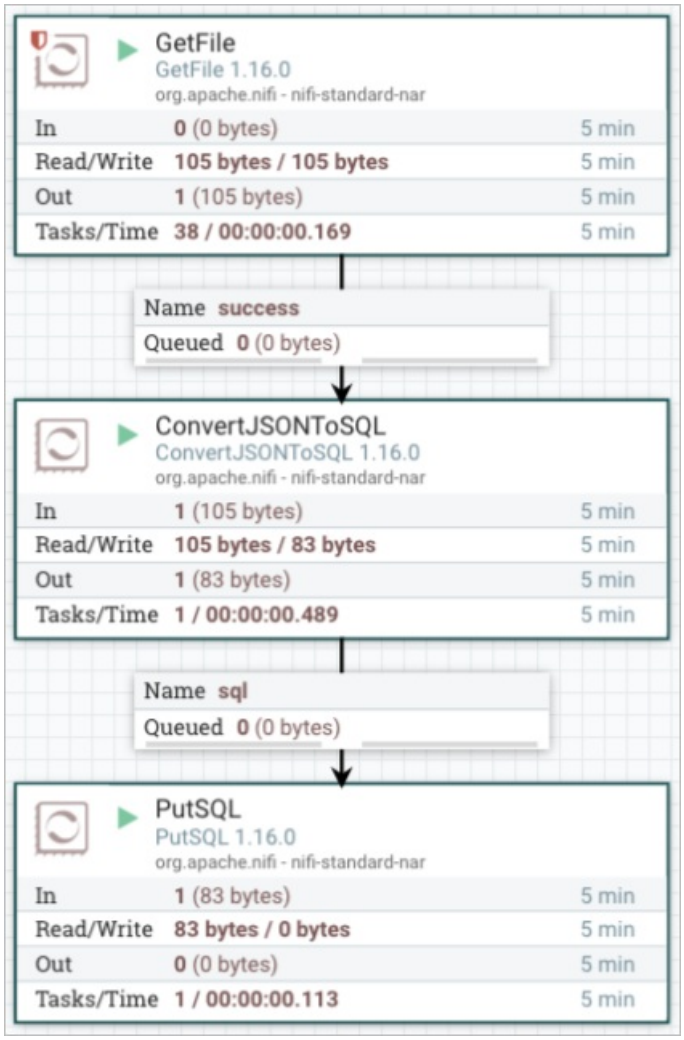
   ii. 将JDBC Connection Pool设置为上一步配置的DBCPConnectionPool，该案例中DBCPConnectionPool名称为 `hologres` 。

   iii. 将**Support Fragmented Transactions**置为false。

   iv. 单击**APPLY**完成配置。

5. 开始写入数据

   至此，您就完成了所有配置，将所有节点置为运行状态，Nifi即开始读取JSON文件写入Hologres。



6. 查询数据

   使用如下命令在Hologres中查询user_info表，即可看到写入的数据。

```
SELECT * FROM user_info;
```

   查询结果如下。

3.11. SQL Workbench/J

本文为您介绍如何使用交互式分析Hologres连接SQL Workbench/J实现数据的可视化分析。

前提条件

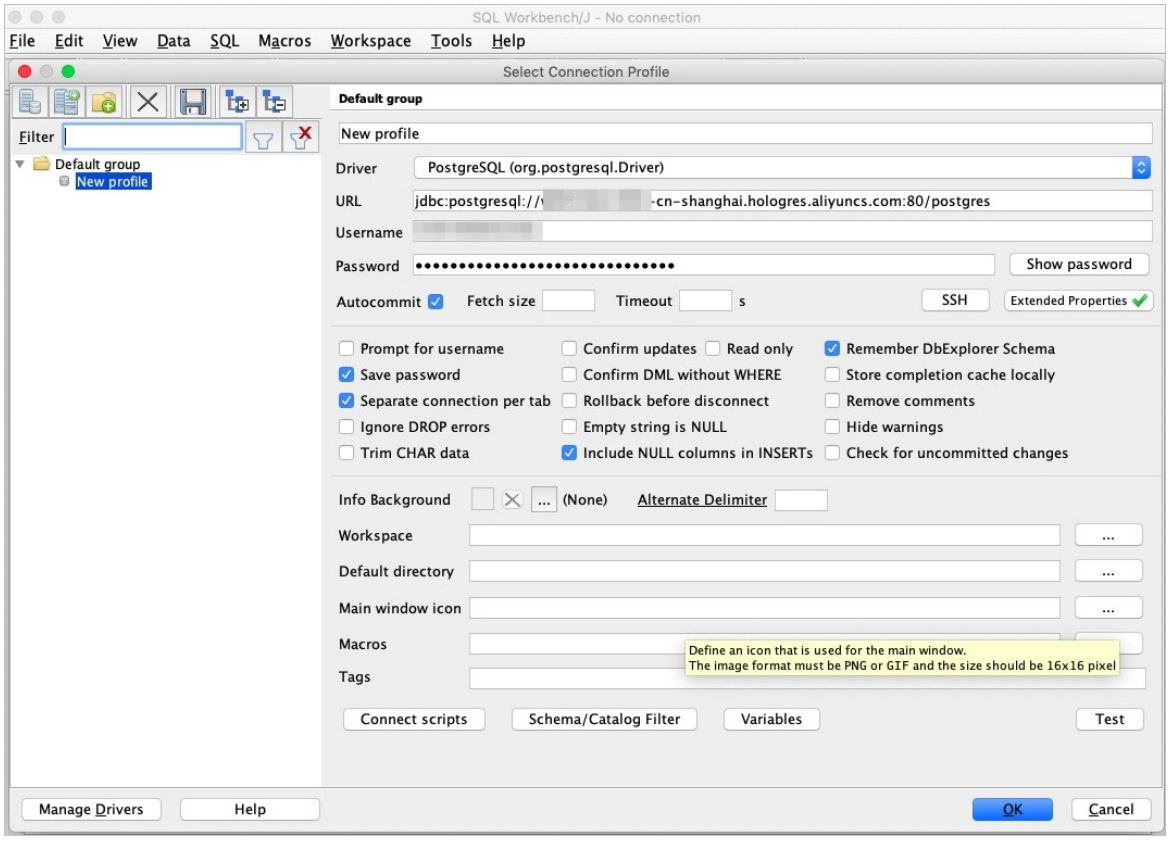
- 单击SQL Workbench/J，下载并安装SQL Workbench。
- 开通Hologres实例，详情请参见购买Hologres。

背景信息

SQL Workbench/J是一款免费的、跨平台的SQL查询分析工具，您可以使用SQL Workbench/J通过PostgreSQL驱动连接Hologres进行数据开发。

操作步骤

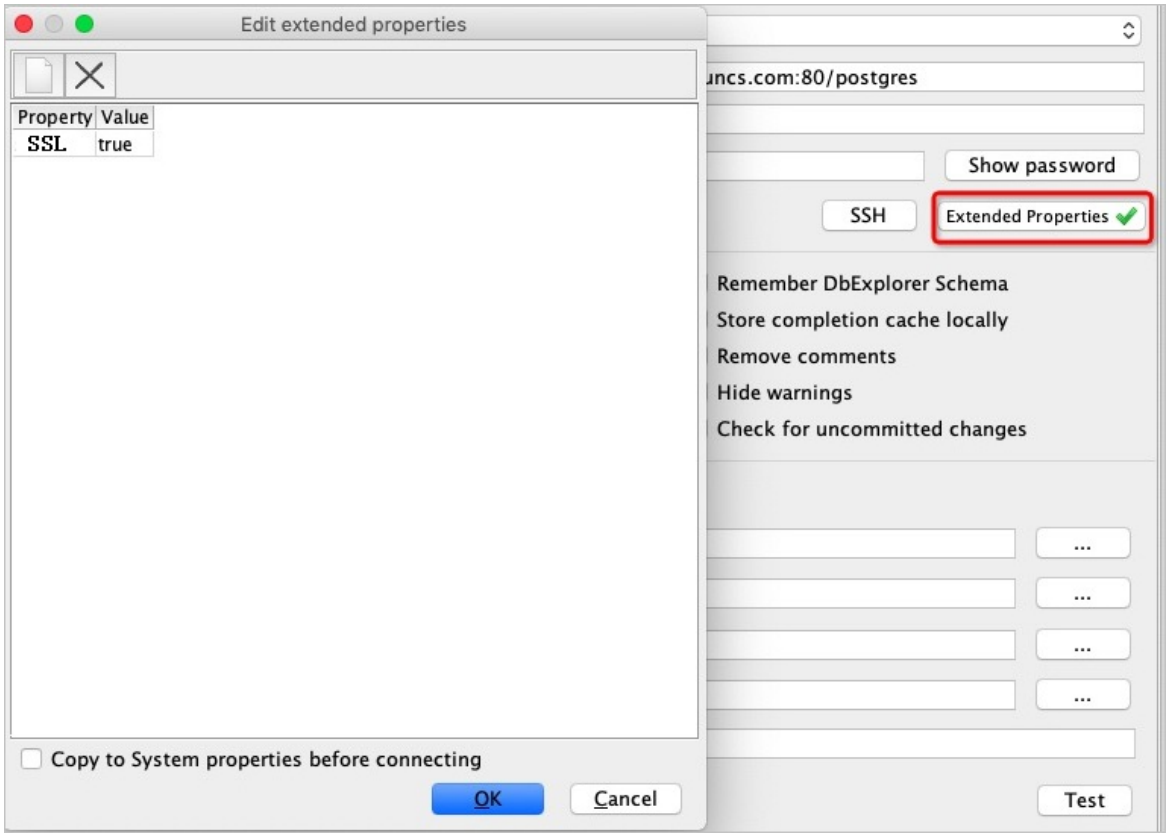
1. 启动SQL Workbench/J，新建至Hologres的连接。
2. 配置连接信息。



参数说明如下表所示。

参数	描述	示例
Driver	PostgreSQL 连接Hologres使用PostgreSQL驱动。	无
URL	`jdbc:postgresql://endpoint:port/database` ○ Endpoint：Hologres实例的公共网络地址。 进入Hologres管理控制台的实例详情页，从**实例配置**获取Endpoint。 ○ Port：Hologres实例的公共网络端口。 进入Hologres管理控制台的实例详情页，从**实例配置**获取Port。 ○ Database：需要连接的Hologres数据库名称。 进入Hologres管理控制台的实例详情页，从**DB管理**获取数据库名称。	`jdbc:postgresql://holodemo-cn-hangzhou.aliyun.com:80/postgres` 示例仅供参考，实际连接时需要更换为对应项目的实际参数。
Username	当前Hologres账号的AccessKey ID。 您可以单击AccessKey 管理，获取用户的AccessKey ID。	无
Password	当前Hologres账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。	无

3. 设置扩展属性。



i. 单击**Extended Properties**，配置SSL为*true*。

ii. 单击**OK**。

4. 单击**OK**，成功连接Hologres至SQL Workbench/J。

   您可以使用SQL Workbench/J查询分析Hologres的数据。更多关于SQL Workbench/J的驱动配置以及软件的安装，请参见SQL Workbench。

4.账号与权限管理

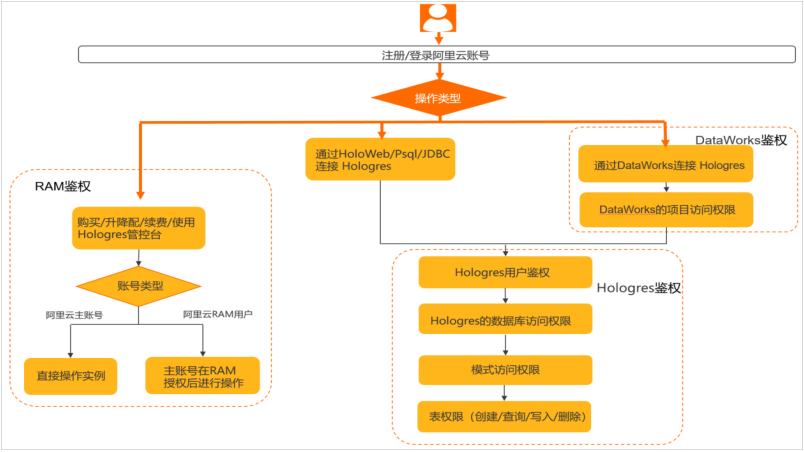
4.1. 权限管理概述

在使用Hologres之前，必须拥有相关的权限才能使用Hologres进行开发。本文为您介绍Hologres相关的权限策略，方便您根据使用场景对不同用户授权，精细化管理用户权限。

Hologres用户鉴权流程

Hologres用户在使用过程中会根据使用场景的不同进行相关权限的鉴权。例如，当用户需要购买实例，则需要进行RAM鉴权，判断该用户是否具有Hologres管理控制台的购买操作权限，否则需要授权才可以进行购买操作。

一个用户从进入阿里云到使用Hologres的完整鉴权流程如下图所示。



- 管理Hologres实例：当您需要在Hologres管理控制台进行实例的购买、管理、升降配、续费和停机等操作时，阿里云会对您的账号进行RAM鉴权，是否具备相关权限，否则不能进行操作。RAM鉴权说明请参见RAM鉴权。
- 连接Hologres进行开发：当您连接Hologres进行相关操作时，会进行Hologres鉴权，您需要具备相关权限才能正常执行操作。Hologres鉴权说明请参见Hologres鉴权。
- 使用DataWorks：如果您需要使用DataWorks进行Hologres开发时，除了Hologres鉴权外也会需要进行DataWorks的操作鉴权确保您具有操作权限。DataWorks鉴权说明请参见DataWorks鉴权。
- 使用MaxCompute：您需要使用Hologres加速查询MaxCompute的表数据，还需要当前Hologres的账号具有访问对应MaxCompute的项目以及表的权限。MaxCompute鉴权说明请参见MaxCompute鉴权。

RAM鉴权

访问控制RAM（Resource Access Management）是阿里云提供的权限管理系统。RAM主要的作用是控制账号系统的权限，您可以通过RAM授权为不同的子账号分配不同的权限，包括实例购买、删除、升配、降配、修改网络类型、查看实例信息等权限，从而达到实例管理的目的。

如果不授予子账号RAM权限，子账号无法在Hologres管理控制台台查看实例详情，但不会影响子账号对Hologres的连接访问。关于RAM的授权流程请参见授予RAM用户权限。

Hologres鉴权

Hologres实时数仓兼容PostgreSQL 11，在使用Hologres实例进行开发之前，会经过如下几个层级的鉴权：

1. 账号鉴权
   - 您可以使用阿里云账号（包括阿里云账号和RAM用户）登录Hologres管理控制台

- 您可以使用阿里云账号（包括阿里云账号和RAM用户）登录Hologres管理控制台。
- 当您使用psql或者JDBC等工具连接Hologres实例时，需要用到AccessKey ID及AccessKey Secret分别对应用户名和用户密码。

更多关于账号的详细描述，请参见账号概述。

2. 用户鉴权

   账号鉴权成功后，当用户使用账号连接Hologres时，系统将会判断当前账号是否为Hologres用户。只有管理员执行了 `create user "xxx"` 命令，用户才会被创建进实例中。关于用户的描述以及创建用户，请参见用户概念。

3. 实例鉴权

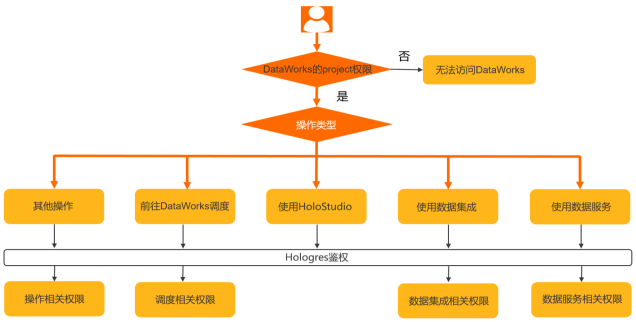
   用户被创建进实例后，还需要被授予相关的操作权限，才能在权限范围内进行操作。对用户的授权操作，请参见Hologres权限模型概述。

DataWorks鉴权

Hologres与DataWorks深度集成，使用DataWorks进行Hologres开发时，其权限管理内容也有部分兼容。在使用过程中请您按照如下几点说明检查您的授权情况：

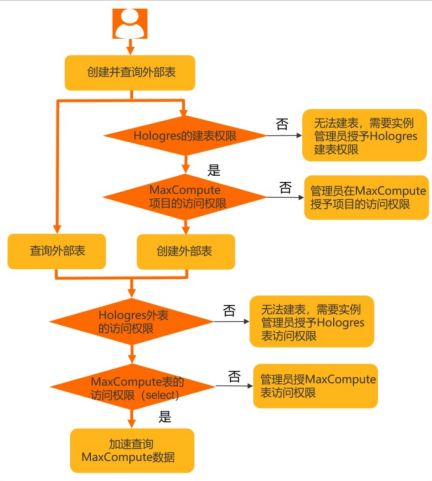
- 进入DataWorks需要具备项目的访问权限。
- 使用HoloStudio开发时需要具备Hologres实例的相关权限。
- 涉及到DataWorks的其他操作，例如数据集成、数据服务等，除了Hologres的鉴权之外，还需要进行DataWorks的操作鉴权，关于DataWorks的权限，请参见附录：预设角色权限列表（空间级）。

DataWorks鉴权流程如下图所示：



MaxCompute鉴权

当使用Hologres加速查询MaxCompute表数据时，需要当前Hologres的账号有访问对应MaxCompute的项目以及表的访问权限。当某个用户想要在Hologres中加速查询MaxCompute时，其鉴权流程如下：



4.2. 账号概述

交互式分析Hologres和阿里云账号体系深度集成，Hologres的用户是通过阿里云账号来进行认证的。本章节内容将为您介绍在Hologres中使用到的阿里云账号体系。

账号

Hologres和阿里云账号体系深度集成，账号主要包括阿里云账号和阿里云RAM用户，具体说明如下表所示。在Hologres实例中，当您需要为阿里云账号和RAM用户授予权限时，需要使用其对应的登录账号或者账号ID信息。具体内容请参见登录账号和账号ID。

账号类型	说明
阿里云账号	阿里云账号用于创建和管理Hologres实例，例如登录Hologres控制台、创建数据库、按量付费转包年包月、授权对象等。
阿里云RAM用户	阿里云账号在RAM创建用户并授予一定的权限后，RAM用户也可以在权限范围内创建和管理实例，例如登录Hologres控制台、创建数据库、按量付费转包年包月、授权对象等。 RAM用户从属于阿里云主账号并且本身不拥有实际的资源，所有资源都属于阿里云主账号。
自定义账号	自定义账号在权限范围内用于对实例进行操作，例如创建/删除数据库、连接数据库、创建/删除表、创建/删除视图等。

登录账号

当您为阿里云账号和RAM用户授予权限时，需要使用阿里云账号的登录账号信息。登录账号信息，请通过用户信息页面获取。

- 当您需要为阿里云账户授权时，阿里云账号的完整表达方式会包含登录账号的信息，具体如下表所示：

账号格式	说明	示例
ALIYUN$<Login Account>@aliyun.com	<Login Account>为阿里云账号的登录账号	ALIYUN$company@aliyun.com
<Login Account>@aliyun.com		company@aliyun.com

- 当您需要为RAM用户授权时，其用户名称的完整表达方式会包含登录账号的信息，具体如下表所示：

账号格式	说明	示例
<subUserName>@<Login Account>.onaliyun.com	账号格式的参数说明如下： ◦ <subUserName>：阿里云RAM用户的名称 ◦ <Login Account>：阿里云主账号的登录账号 ◦ <AccountID>：阿里云账号ID	holouser@company.onaliyun.com
<subUserName>@<Login Account>		holouser@company
<subUserName>@<Account ID>.onaliyun.com		holouser@123456789xxxx
RAM$<subUserName>		RAM$holo_test
RAM$<Login Account>:<subUserName>		RAM$company:holouser
RAM$<Account ID>:<subUserName>		RAM$123456789xxxx:holouser
<subUserName>@<Account ID>		holouser@123456789xxxx

账号ID

账号ID又称为Account ID，是一串数字，例如 `189813715xxxx` 。账号ID信息，请通过用户信息页面获取。

RAM用户的Account ID为对应的UID，请通过访问控制页面的用户页面获取。在Hologres中，要使用RAM用户UID授权时，格式必须为 `p4_UID` ，例如 `p4_12333388xxx` 。

Hologres可以使用Account ID为用户授权，示例语句如下：

```
create USER "189813715xxxx"; //将阿里云账号189813715xxxx创建至Hologres。
create USER "p4_12333388xxx" superuser;//将RAM用户12333388xxx授权为实例的Superuser。
```

在Hologres中执行如下语句，查看当前账号的Account ID。

```
SELECT current_user;
```

AccessKey ID及AccessKey Secret

阿里云AccessKey ID和AccessKey Secret用于连接访问Hologres实例。AccessKey ID类似登录账号，AccessKey Secret类似登录密码，可以在阿里云官网Access Key管理查看。

AccessKey ID及AccessKey Secret是阿里云颁发的访问凭证，有一定的使用期限。如果AccessKey ID及AccessKey Secret过期，您可以参见准备阿里云账号重新创建。

当您使用psql或者JDBC等工具连接Hologres实例时，其中填写的用户名和用户密码就是AccessKey ID和AccessKey Secret。

4.3. 授予RAM用户权限

本文为您介绍主账号如何为RAM用户授予权限，从而让RAM用户实现在Hologres管理控制台执行查看、购买或删除实例等操作。

背景信息

访问控制（Resource Access Management，简称RAM）是阿里云提供的权限管理系统。

RAM主要用于控制账号系统的权限。

您可以使用RAM在主账号的权限范围内创建RAM用户，并为不同的RAM用户授予不同的权限，例如购买或删除实例、升降配实例资源、修改网络类型以及查看实例信息等。

RAM用户实例开发权限的权限控制如下：

- 如果主账号没有授予RAM用户权限，则RAM用户无法在管理控制台查看或操作实例。
- 主账号可以直接授予RAM用户实例的开发权限。RAM用户即使无法在管理控制台操作实例，也可以正常连接开发工具进行数据开发。详情请参见授予RAM用户实例的开发权限。

授予RAM用户权限

1. 主账号登录RAM管理控制台。
2. 选择需要授权的RAM用户。
   i. 在左侧导航栏的**人员管理**菜单下，单击**用户**。
   ii. 在**用户**页面，单击目标RAM用户**操作**列的**添加权限**。
3. 新增授权。

   配置**添加权限**对话框的各项参数。

参数	描述
授权范围	取值如下： ○ 云账号全部资源 ○ 指定资源组
被授权主体	需要授权的RAM用户。

参数	描述
选择权限	取值如下： ○ 系统策略 ○ 自定义策略 **说明** ○ 您也可以根据业务需求自定义新建权限策略。 ○ 每次最多添加5条策略。如果您需要添加的策略数量大于5条，请分多次执行。

系统策略和**自定义策略**的权限说明如下：

- **系统策略**

  为RAM用户授予以下权限策略，RAM用户就会拥有所有操作的权限，如下表所示。

权限策略	描述
AliyunHologresFullAccess	管理Hologres服务的权限。 设置该权限后，RAM用户可以查看管理控制台所有实例的信息，以及购买实例。 **说明** 您需要设置**AliyunRAMReadOnlyAccess**权限后，才可以在管理控制台的**用户管理**查看用户信息。
AliyunBSSOrderAccess	在费用中心（BSS）查看、支付以及取消订单的权限。 设置该权限后，子账号可以在管理控制台升级或降级实例的配置，以及为实例续费。
AliyunRAMReadOnlyAccess	只读访问控制（RAM）的权限。 设置该权限后，RAM用户可以在管理控制台的**用户管理**查看当前实例的用户、组以及授权信息。
AliyunHologresReadOnlyAccess	只读管理Hologres的权限。 设置该权限后，RAM用户可以查看Hologres管理控制台所有实例的信息，但是无法操作实例，例如修改网络类型。

说明

- RAM用户购买的实例，主账号和RAM用户默认均为Superuser。
- 主账号购买的实例，RAM用户需要被主账号授权后，才能使用实例。

- **自定义策略**

您可以根据业务需求，在**新建权限策略**中自定义不同的权限。

在**新建自定义权限策略**页面，您可以使用**脚本配置**自定义权限。

示例语句如下。

```
{
    "Statement": [
        {    //授予RAM用户所有操作的权限。设置该权限后，则无需再设置下文的其他权限。
            "Effect": "Allow",
            "Action": "hologram:*", //表示拥有所有操作的权限。
            "Resource": "acs:hologram:*:<主账号ID>:instance/*" //可以配置所有地域的实例。
        },
        {    //购买实例或为实例续费。
            "Effect": "Allow",
```

```
            "Action": "hologram:*",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //删除实例。
            "Effect": "Allow",
            "Action": "hologram:DeleteInstance",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //RAM用户购买权限。RAM用户配置该权限后才可以购买实例。
            "Effect": "Allow",
            "Action": "bss:PayOrder",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //显示实例详情。
            "Effect": "Allow",
            "Action": "hologram:DescribeInstance",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //显示实例列表。
            "Effect": "Allow",
            "Action": "hologram:ListInstances",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //暂停实例。
            "Effect": "Allow",
            "Action": "hologram:StopInstance",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //恢复实例。
            "Effect": "Allow",
            "Action": "hologram:ResumeInstance",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //显示实例监控告警。
            "Effect": "Allow",
            "Action": "hologram:GetInstanceMetrics",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        },
        {   //修改网络类型。
            "Effect": "Allow",
            "Action": "hologram:ModifyInstanceNetworkType",
            "Resource": "acs:hologram:cn-<region>:<主账号ID>:instance/*"
        }
    ],
    "Version": "1"
}
```

参数说明如下表所示。

参数	描述
<region>	地域，例如**beijing**。
<主账号ID>	阿里云主账号的ID。

参数	描述
*	表示该主账号中所有实例的ID。 您也可以替换*为具体的一个实例ID。

示例语句如下。

```
acs:hologram:cn-beijing:4322xxxxx:instance/hhhgggxxxx
```

4. 单击**确定**。

与控制台相关的RAM用户权限问题

Hologres管理控制台主要集成了RAM鉴权和实例的部分开发权限，与Hologres管理控制台相关的权限问题如下所示：

- RAM用户无法查看实例列表及实例ID。
  - 问题现象

    RAM用户选择了正确的地域，但无法看到已购买的实例，并提示**暂无权限查看所有实例，请让主账号前往RAM中心授予当前用户“xxx/*”资源的“hologram:ListInstances”权限**报错。

  - 问题原因

    当前RAM用户没有查看实例列表的权限。

  - 解决方法

    主账号登录RAM控制台，授予RAM用户显示实例列表的权限**AliyunHologresReadOnlyAccess**。

- 被授权为Superuser的RAM用户无法新增用户。

- 问题现象

  RAM用户已经被授权为Superuser，但是在Hologres管理控制台的**用户管理**模块无法看到用户列表并新增用户。提示**暂无权限列出所有子账号，请让主账号前往RAM中心授予当前用户ram:ListUsers权限**报错。

- 问题原因

  当前RAM用户在Hologres管理控制台没有查看用户列表的权限。

- 解决方法

  主账号登录RAM控制台，授予RAM用户显示用户列表的权限**AliyunRAMReadOnlyAccess权限策略**。

- 无法查看**用户管理**和**DB管理**模块的信息。
  - 问题现象

    RAM用户登录Hologres管理控制台后，无法查看**用户管理**和**DB管理**模块的信息。提示**暂无权限，请找Superuser将当前账户添加到实例中**报错。

  - 问题原因

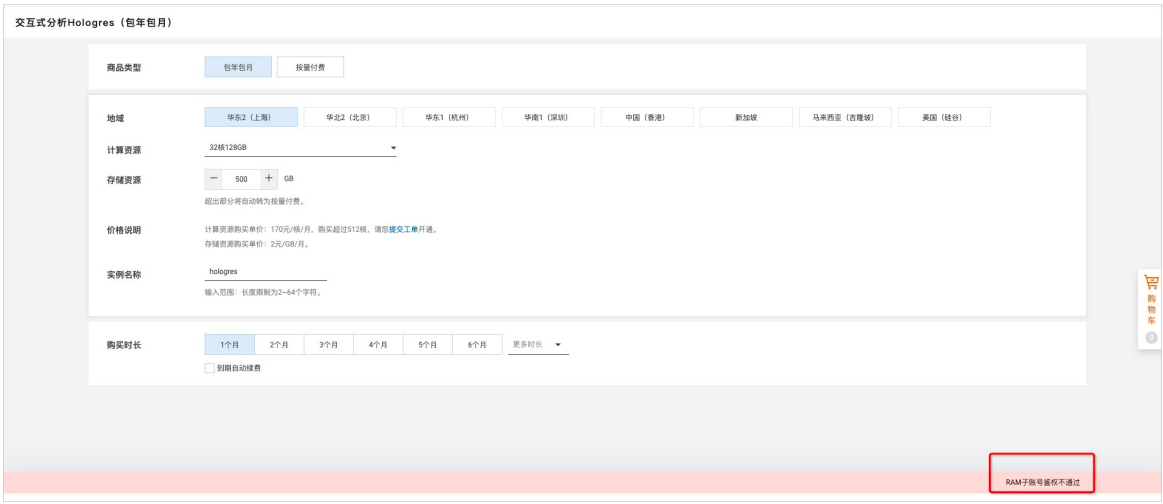
    当前RAM用户没有实例的开发权限。

  - 解决方法

    主账号或具有Superuser权限的账号为RAM用户授予实例的开发权限，详情请参见授予RAM用户实例的开发权限。

- RAM用户无实例的操作权限。
  - 问题现象

    被授权为Superuser的RAM用户无法购买、升配和降配实例，以及转换实例的付费方式由**按量付费**为**包年包月**。提示**RAM子账号鉴权不通过**报错。



  - 问题原因

    购买、升配、降配及转换付费方式等操作涉及费用账单，由主账号统一控制，当前RAM用户没有相关权限。

  - 解决方法

    主账号登录RAM控制台，授予RAM用户与实例费用相关的权限AliyunHologresFullAccess及AliyunBSSOrderAccess。

4.4. RAM角色授权模式

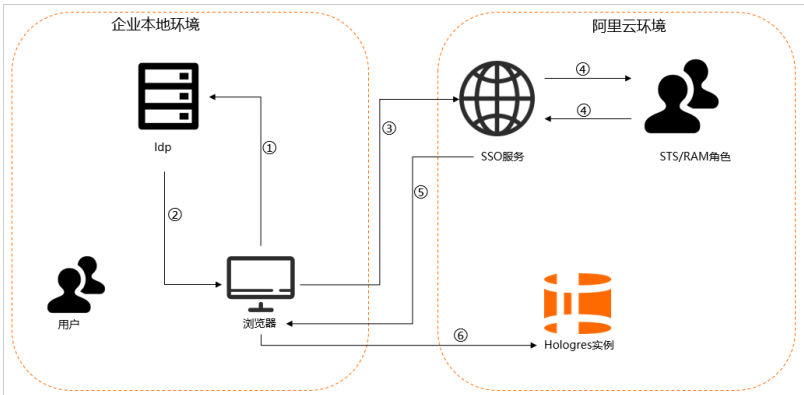
本文为您介绍如何使用角色SSO的方式访问Hologres。

背景信息

阿里云支持企业用户通过在阿里云控制台输入账号、密码后登录阿里云来管理和使用云资源。随着企业安全监管要求的日益严格，部分企业更愿意通过角色登录（Role Base_SSO）的方式登录阿里云。详情请参见SAML角色SSO概览。

适用场景

一般情况下，对于阿里云企业用户，需要用户通过在阿里云控制台输入账号、密码后登录阿里云来管理和使用云资源。但是随着企业安全监管要求的日益严格，希望集中管理登录鉴权信息，企业往往会选择单点登录（SSO）的方式登录应用。SSO是指在多个应用系统中，用户只需要登录一次，就可以访问所有相互信任的应用系统。现在Hologres支持基于角色的SSO登录模式，详情请参见SAML角色SSO概览。基于该功能，您可以使用企业账号，扮演RAM角色访问Hologres实例。具体的访问权限由RAM角色控制。一个调用的样例如下。



1. 用户使用浏览器在Idp的登录页面中选择阿里云作为目标服务。

   例如：如果企业IdP使用AD FS（Microsoft Active Directory Federation Service），则登录URL为：https://ADFSServiceName/adfs/ls/IdpInitiatedSignOn.aspx。

   > **说明** 有些IdP会要求用户先登录，再选择代表阿里云的SSO应用。

2. IdP生成一个SAML响应并返回给浏览器。
3. 浏览器重定向到SSO服务页面，并转发SAML响应给SSO服务。
4. SSO服务使用SAML响应向阿里云STS服务请求临时安全凭证，并生成一个可以使用临时安全凭证登录阿里云控制台的URL。

   > **说明** 如果SAML响应中包含映射到多个RAM角色的属性，系统将会首先提示用户选择一个用于访问阿里云的角色。

5. SSO服务将URL返回给浏览器。
6. 浏览器重定向到该URL，以指定RAM角色登录到阿里云控制台，用户使用企业账户扮演RAM角色登录Hologres实例。

Hologres支持的用户访问方式

Hologres支持如下两种访问方式：

- 通过云账号（阿里云账号或RAM用户）访问Hologres。

  您可以通过输入账号、密码的方式登录阿里云管理控制台，并以当前登录账号的身份使用Hologres。此时，阿里云云账号将成为Hologres某个实例的成员，拥有Hologres产品的使用权限。

- 通过RAM角色SSO的方式登录Hologres。

  您也可以通过角色SSO的方式（SAML角色SSO概览）登录阿里云，并使用Hologres。此时，阿里云访问控制角色（RAM Role）将成为Hologres某个实例的成员，扮演该RAM Role的使用者将拥有和云账号类成员同样的产品使用权限。RAM角色的描述，详情请参见RAM角色概览。

对于Hologres来说，RAM角色（RAM Role）和阿里云账号（包括主账号和RAM用户）是同等地位的账号。因此，对于Hologres来说，RAM角色就是一个普通的可登录账号。Superuser需要对这个RAM角色（而不是背后的云账号）进行授权来赋予这个RAM角色权限（SELECT/INSERT/UPDATE等），这个RAM角色才能在权限范围内使用Hologres。

RAM角色SSO（STS）介绍

通过RAM角色SSO的方式登录并访问Hologres是基于阿里云STS服务实现的。阿里云STS（Security Token Service）是为阿里云账号（或RAM用户）提供短期访问权限管理的云服务。通过STS，您可以为用户（您的本地账号系统所管理的用户）颁发一个自定义时效和访问权限的访问凭证。用户可以使用STS短期访问凭证直接连接Hologres并访问被授权的资源。

使用STS Token有以下优势：

- 减少了账号AccessKey ID和AccessKey Secret泄露的风险，只需要生成一个临时访问凭证给用户使用即可。
- 能使用灵活的权限控制，STS Token有一定的时间期限，不需要关心权限撤销问题，临时访问凭证过期后会自动失效。

如下操作步骤将指导您通过创建RAM角色并授权访问Hologres。

- 步骤一：创建RAM角色
- 步骤二：添加RAM角色至Hologres实例并授权
- 步骤三：登录阿里云并使用Hologres

步骤一：创建RAM角色

登录访问控制，创建RAM角色。当前可信实体类型，可以选择阿里云账号或者身份提供商。

- 如果需要通过阿里云管理控制台切换身份方式扮演该角色，选择阿里云账号类型的角色。具体操作请参见通过RAM用户扮演角色并添加权限。
- 如果需要通过本地IDP身份提供商账号登录至阿里云扮演该角色，选择身份提供商类型的角色。具体操作请参见通过IDP身份提供商账号来扮演角色并添加权限。

通过RAM用户扮演角色并添加权限

如果需要通过RAM用户来扮演RAM角色并基于阿里云控制台切换身份方式扮演该角色，请登录访问控制，创建RAM角色。当前可信实体类型，可以选择阿里云账号。

1. 创建可信实体类型为阿里云账号的RAM角色。
    i. 登录访问控制在左侧导航栏，单击RAM角色管理。
    ii. 在**RAM角色管理**页面，单击**创建RAM角色**，当前可信实体类型选择阿里云账号。
    iii. 单击**下一步**，配置角色名称，并选择当前云账号。
    iv. 单击**完成**，页面提示角色创建成功完成创建。
2. 创建并添加权限策略。
    i. 在RAM角色管理列表页，单击目标角色名称，进入角色信息详情页。

ii. 单击进入**信任策略管理**页签，修改新任策略为如下脚本内容。

- 参数说明

  在策略配置时，您需要将如下脚本中的 `acs:ram::主账号ID:root` 信息中的主账号ID，替换为需要授权的账号信息。请前往用户信息页面，获取账号ID。

- 脚本配置

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "RAM": [
                    "acs:ram::主账号ID:root"
                ]
            }
        },
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "dataworks.aliyuncs.com"
                ]
            }
        }
    ],
    "Version": "1"
}
```

iii. 单击**确定**，完成权限策略配置。

3. 创建RAM用户并授予角色权限。

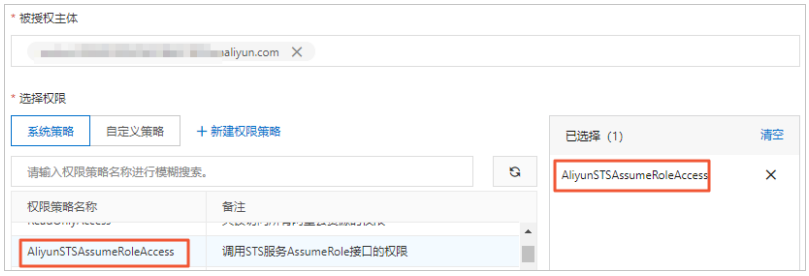
通过RAM用户来扮演RAM角色，需要创建一个RAM用户并授予扮演角色的权限。

i. 登录访问控制，在左侧导航栏选择**人员管理 > 用户**。

ii. （可选，如果您已经存在RAM用户，可跳过该步骤。）单击**创建用户**，可一次性创建多个RAM用户。配置参数，其中访问方式选择编程访问。单击**确定**。

iii. 单击目标RAM用户操作列的**添加权限**。

iv. 在**添加权限**页面，为已创建的RAM用户添加AliyunSTSAssumeRoleAccess权限，获取调用STS服务AssumeRole接口的权限。



v. 单击**确定**，完成角色设置。

通过IDP身份提供商账号来扮演角色并添加权限

如果需要通过本地IDP身份提供商账号登录至阿里云扮演RAM角色，请登录访问控制，创建RAM角色。当前可信实体类型，可以选择身份提供商。

1. 创建可信实体类型为身份提供商的RAM角色。
   i. 登录访问控制在左侧导航栏，单击RAM角色管理。
   ii. 在**RAM角色管理**页面，单击**创建RAM角色**，当前可信实体类型选择身份提供商。
   iii. 单击**下一步**，配置角色名称和备注。
   iv. 选择身份提供商并查看限制条件后，单击**完成**，页面提示角色创建成功完成创建。
2. 创建并添加权限策略。
   i. 在RAM角色管理列表页，单击目标角色名称，进入角色信息详情页。
   ii. 单击进入**信任策略管理**页签，修改新任策略为如下脚本内容。
   - 参数说明

     在策略配置时，您需要将如下脚本中的`acs:ram::主账号ID:saml-provider/IDP`信息中的主账号ID，替换为需要授权的账号信息。请前往用户信息页面，获取账号ID。
   - 脚本配置

```
"Statement": [
        {
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "saml:recipient": "https://signin.aliyun.com/saml-role/sso"
                }
            },
            "Effect": "Allow",
            "Principal": {
                "Federated": [
                    "acs:ram::主账号ID:saml-provider/IDP"
                ]
            }
        },
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "dataworks.aliyuncs.com"
                ]
            }
        }
    ],
    "Version": "1"
}
}
```

   iii. 单击**确定**，完成权限策略配置。

步骤二：添加RAM角色至Hologres实例并授权

RAM角色需要有Hologres实例的开发权限，才能在权限范围内使用Hologres。由于RAM角色默认没有Hologres管控台的查看和操作实例的权限，因此需要主账号完成RAM相关权限授予才能进行后续操作。具体操作，请参见授予RAM用户权限。添加RAM角色至Hologres实例，您可以通过如下方式进行授权。

1. 登录Hologres管理控制台进行授权。
   i. 在左侧导航栏选择进入**实例列表**页面，单击需要授权的实例，在**用户管理**页签找到RAM角色并新增用户至实例。
   ii. 在**DB管理**页签，为该RAM用户授予具体的实例开发权限。
2. 通过SQL方式授权。

   您可以通过SQL方式进行授权，具体请参见权限管理概述。
3. 若是通过RAM用户扮演RAM角色，RAM角色默认没有Hologres管理控制台的权限，需要主账号给RAM用户在访问控制页面授予**AliyunRAMReadOnlyAccess**权限，否则RAM角色无法在Hologres管理控制台进行任何操作。详情请参见文档授予RAM用户权限。

步骤三：登录阿里云并使用Hologres

当您完成授权之后，使用者就可以扮演user-role角色登录并使用Hologres。

1. 登录阿里云。使用者可以扮演RAM角色user-role，登录至阿里云控制台。
2. 登录Hologres管理控制台，进行实例管理和监控。
3. 在Hologres管理控制台单击**登录Hologres数据库**，进入HoloWeb，进行Hologres结构设计和数据开发。具体操作请参见连接HoloWeb。

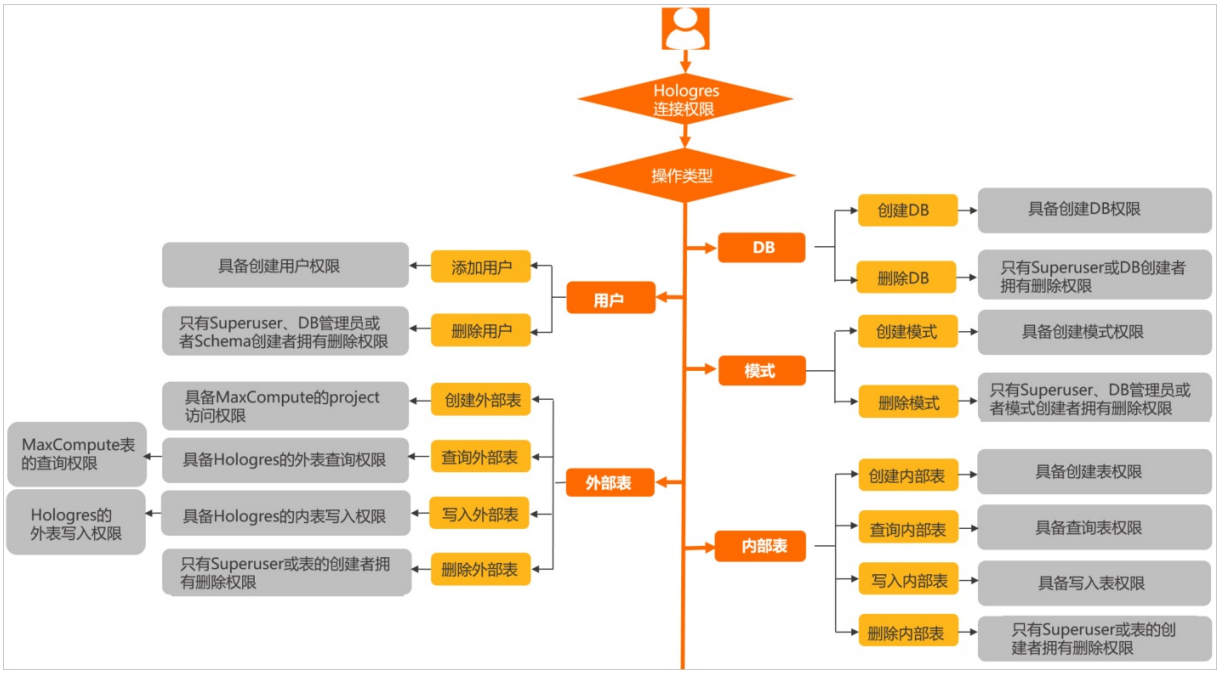
4.5. Hologres权限模型

4.5.1. Hologres权限模型概述

当一个用户登录Hologres实例后，必须有实例相关的操作权限才能在权限范围内进行操作。本文为您介绍使用Hologres实例开发需要的权限。

Hologres鉴权流程

一个用户从连接Hologres到使用Hologres的完整鉴权流程如下：



用户概念

当一个阿里云账号连接Hologres时，需要被创建成为Hologres的用户（管理员需要执行 `create user "xxx"` ，否则会报错 `role "xxx" does not exist` ），才能连接成功。用户属于实例级别，所以添加用户和删除用户相当于把用户创建进实例中或者从实例中删除。在实例内进行的具体操作（例如创建表等），则是DB级别的权限，授予权限时需要在某一个DB中执行，只享有这个DB内对象的权限，不能跨DB使用权限。

角色类型	说明
超级管理员（Superuser）	系统默认将购买Hologres实例的主账号设置为Superuser，拥有实例的所有权限。可以创建、销毁数据库，也可创建角色以及为角色授权等。
普通用户（Normal）	除Superuser之外的用户都被称为普通用户，需要Superuser授予权限才能访问Hologres实例，并在权限范围内进行数据开发。普通用户也能被授权成为Superuser，享有实例的所有权限。
用户组	为了方便用户的管理，将零个或者多个用户设置成用户组，用来表示用户角色。例如管理员角色、开发者角色等，一个用户组内用户的拥有相同的权限。设置用户组并授权相关角色的详情内容，请参见Postgres 数据库角色。

权限模型

Hologres兼容PostgreSQL 11，为用户授予实例开发权限时，可以使用标准的PostgreSQL授权语句（专家权限模型），由于PostgresQL的权限粒度较细，且授权语句比较复杂难懂，Hologres在此基础上又提供简单权限模型，提供更加便捷的权限控制。

关于简单权限模型和专家权限模型的区别和使用场景如下：

权限类型	适用场景	说明
专家权限模型（PostgreSQL）	适用于需要非常严格权限管控的场景。例如，精确到某个人用某个表。	专家权限模型的权限授予粒度小且灵活，可以为用户授予具体某个表的权限，但是授权语法比较复杂。具体权限授予操作，请参见专家权限模型。
简单权限模型（Simple Permission Model，SPM）	DB级别的权限管控，适用于粗粒度的权限管理场景。	简单权限模型是封装好的权限模型，以DB为维度，每个用户组都有对应的权限，不可修改，能满足大部分授权场景，且授权操作比较简单。具体权限授予操作，请参见简单权限模型。
Schema级别的简单权限模型（Schema-level Permission Model，SLPM）	精确到Schema级别的权限管控，使用于对权限粒度较为细致且又希望简化授权流程的场景。	Schema级别的简单权限模型是已经封装好的权限模型，以Schema为维度，每个用户组都有对应的权限，不可修改，满足对于权限较为细粒度的管控，且授权操作比较简单。具体权限授予操作，请参见基于Schema级别的简单权限模型的使用。

权限授予

用户想要使用Hologres进行开发，需要授予具体的权限才能在权限范围内操作，专家权限模型和简单权限模型授权的具体操作和权限关系如下两个表格所示，您可以根据需要的操作配合所使用的权限模型来为用户授权。

> **说明** Superuser默认具有表格中的所有权限。

专家权限模型操作	需要的权限	授权语句
CREATE USER(ROLE) DROP USER(ROLE)	CREATEROLE	如下示例为授予用户A创建角色的权限。 `alter user A createrole;`
CREATE TABLE VIEW TABLE FOREIGN TABLE	DB的CONNECT权限以及对应Schema的CREATE权限	如下示例为授权用户A有xx schema的建表权限。 `GRANT CREATE ON SCHEMA xx TO A;` 任何用户默认有public schema下的建表权限。

SELECT	对应Schema的USAGE权限以及SELECT权限	您可以按照如下方式进行授权：
		`GRANT USAGE ON SCHEMA xx to A;`
		`GRANT SELECT ON TABLE xx TO A;`
		`GRANT SELECT ON ALL TABLES IN SCHEMA xx TO A;`

专家权限模型操作	需要的权限	授权语句

INSERT
UPDATE
DELETE
TRUNCATE
	对应Schema的USAGE权限以及INSERT/UPDATE /DELETE /TRUNCATE权限	您可以按照如下方式进行授权：
		`GRANT USAGE ON SCHEMA xx to A;`
		`GRANT 【INSERT/UPDATE/DELETE/TRUNCATE】 ON TABLE xx TO A;`
		`GRANT 【INSERT/UPDATE/DELETE/TRUNCATE】 ON ALL TABLES IN SCHEMA xx TO A;`

ALTER TABLE DROP TABLE	表的owner（表的owner可以用ALTER OWNER改变）	删除表不能使用GRANT，只能执行命令 `ALTER TABLE xx OWNER TO A;` 将表的owner修改为A后，由A删除该表。
CREATE DATABASE	CREATEDB	如下示例为授予用户A创建数据库的权限： `ALTER USER A CREATEDB;`
DROP DATABASE	DB owner	删除数据库不能使用GRANT，只能执行命令 `ALTER DATABASE xx OWNER TO A;` 将数据库的owner修改为A后，由A删除该DB。
CREATE EXTENSION	DB owner	-
GRANT REVOKE	具有相应权限，并具备GRANT OPTION的用户	如下示例为授予用户GRANT操作权限： `GRANT 【Privilege】 TO A WITH GRANT OPTION;`

权限模式授权	简单权限模型（SPM）		Schema级别的简单权限模型（SLPM）	
操作	需要的权限	授权语句	需要的权限	授权语句

权限模式授权	简单权限模型（SPM）		Schema级别的简单权限模型（SLPM）	

CREATE USER(ROLE)
DROP USER(ROLE)
	DB admin	您可以选择如下两种方式为用户授予admin权限：
		`call spm_grant('A', '<dbname>_admin');`
		`call spm_create_user('A', '<dbname>_admin');`
			DB admin	您可以选择如下两种方式为用户授予admin权限：
				`call slpm_create_user('A', '<dbname>.<schema>.admin');`
				`call slpm_grant('<dbname>.<schema>.admin', 'A');`

CREATE TABLE VIEW TABLE FOREIGN TABLE	拥有superuser、admin或者developer用户组权限	您可以参照本表格内SPM的授权语句为用户授予admin或者developer权限。	拥有superuser、schema的admin或者developer用户组权限	您可以参照本表格内SLPM的授权语句为用户授予schema的admin或者developer权限。
SELECT	拥有superuser、admin、developer、writer或viewer用户组权限	您可以参照本表格内SPM的授权语句为用户授予admin或者developer、writer、viewer权限。	拥有superuser、schema的admin、developer writer或viewer用户组权限	您可以参照本表格内SLPM的授权语句为用户授予schema的admin、developer、writer或viewer权限。
INSERT UPDATE DELETE TRUNCATE	拥有superuser、admin、developer或writer用户组权限	您可以参照本表格内SPM的授权语句为用户授予admin、developer或者writer权限。	拥有superuser、schema的admin、developer或writer用户组权限	您可以参照本表格内SLPM的授权语句为用户授予schema的admin、developer、writer或viewer权限。
ALTER TABLE DROP TABLE	拥有superuser、admin或者developer用户组权限	您可以参照本表格内SPM的授权语句为用户授予admin或者developer权限。	拥有superuser、schema的admin或者developer用户组权限	您可以参照本表格内SLPM的授权语句为用户授予schema的admin或者developer权限。

权限模式授权	简单权限模型（SPM）		Schema级别的简单权限模型（SLPM）	
CREATE DATABASE DROP DATABASE CREATE EXTENSION	DB admin	您可以参照本表格内SPM的授权语句为用户授予admin权限。	DB admin	您可以参照本表格内SLPM的授权语句为用户授予schema的admin权限。

GRANT
REVOKE
	DB admin	如下示例分别为用户A授予GRANT和REVOKE权限：
		`call spm_grant('A', 'role');`
		`call spm_revoke('A', 'role');`
			DB admin	如下示例分别为用户A授予GRANT和REVOKE权限：
				`call slpm_grant('A', 'role');`
				`call slpm_revoke('A', 'role');`

4.5.2. 简单权限模型（SPM）

4.5.2.1. 简单权限模型概述

本文为您介绍基于实时数仓Hologres的简单权限模型。

背景

Hologres兼容Postgres，使用与Postgres完全一致的权限系统（简称专家模式，详情请参见专家权限模型）。典型的Postgres权限系统划分非常严格，在实际业务场景中使用时，操作较为复杂，通常存在如下痛点：

- 如果您想要给某个用户授权，需要执行大量的授权语句。
- 不同的角色有不同的权限，在操作上非常繁琐，有时也会出现某条授权语句遗漏导致某个权限缺失的情况。
- 每新增一个用户就需要执行相同的批量授权语句，浪费大量时间。
- 尽管我们提供了Postgres的标准授权语句作为参考，但当使用习惯不同时，面对不同的权限，常常难以准确执行正确的授权语句，在权限管理方面容易混乱。这在一定程度上也会给业务带来安全风险，同时也加大了您的管理成本、时间成本和使用成本。

为解决以上业务痛点，Hologres在PostgreSQL权限的基础上，提供了一种粗粒度的简单权限模型（Simple Permission Model，SPM）。简单权限模型以DB作为维度，划分admin（管理员）、developer（开发者）、writer（读写者）以及viewer（分析师）四种角色，您可以通过少量的权限管理函数，即可对DB中的对象进行方便且安全的权限管理。

简单权限模型介绍

在简单权限模型中，每个DB有如下几种权限等级：

- DB管理员：{db}_admin
- 开发者：{db}_developer
- 读写者：{db}_writer
- 分析师：{db}_viewer

每个角色对应的权限如下表所示。

角色	权限
{db}_admin	• 某个DB的管理员（admin）。 • {db}_admin组的权限是{db}_developer、{db}_writer和{db}_viewer组权限的合集。 • 某个DB的owner，可以删除DB。 • 可管理当前DB的{db}_admin、{db}_developer、{db}_writer和{db}_viewer四个用户组的成员，包括新增及移除用户组成员。 • 可以创建用户，并将用户加入某个用户组。 • 可以在该DB创建对象，例如Schema，并可以对对象进行增删改查。 • 可以在DB级别修改DB的配置项。
{db}_developer	• DB的开发者（developer）。 • {db}_developer组的权限是{db}_writer和{db}_viewer组权限的合集。 • 该DB所有Schema中，除系统对象以外的所有表、外表、类表对象（如视图等）、Function、Procedure、Foreign Server、FDW、Type及Language的owner，可以增删查改所有Schema中的所有表。 • 所有Schema的USAGE及CREATE权限，可以在任意非系统的Schema中进行创建表，创建视图及创建外表等DDL操作。
{db}_writer	• DB下的读写者（writer）。 • {db}_writer组的权限是{db}_viewer权限的合集。 • Schema中所有表、外表及类表对象（如视图等）的数据，即拥有SELECT、INSERT、UPDATE及DELETE等权限。 • 可以增删查改所有Schema。 • 可以访问或使用developer所拥有的Function、Procedure、Foreign Server、FDW、Type及Language等对象。 • 所有Schema的USAGE权限，不可进行DDL操作。
{db}_viewer	• DB的分析师（viewer）。 • 可以读取所有schema下所有表、外表及类表对象（如视图等）的数据，即拥有SELECT权限。 • 可以访问或使用developer所拥有的Function、Procedure、Foreign Server、FDW、Type及Language等对象。 • 所有Schema的USAGE权限。

4.5.2.2. 简单权限模型的使用

本文为您介绍在Hologres中简单权限模型（Simple Permission Model，SPM）的使用方法。

使用简单权限模型进行授权

在Hologres中，使用简单权限模型对单个实例进行授权可以使用如下方式：

- 使用管理控制台进行授权

您可以在Hologres管理控制台使用可视化方式创建用户并授权，详情请参见使用简单权限模型为RAM用户授权（推荐）。

- 使用SQL语句进行授权

  在Hologres实例连接开发工具后，可以使用SQL语句通过简单模式对用户进行授权，使该用户具有实例的相关权限。具体步骤如下所示：

  1. 开启函数调用。

     开启SPM前，您需要执行如下命令，开启调用函数的开关。

     ```
     create extension spm;
     ```

  2. 开启简单权限模型。

     简单权限模型默认不开启，Superuser需要在目标DB里执行如下语句，开启简单权限模型。更多关于函数的说明请参见spm_enable。

     ```
     call spm_enable();  // 开启当前DB的简单权限模型。
     ```

     > **说明** 开启简单权限模型之后，developer用户组拥有DB中所有schema的所有表、类表的权限默认。

  3. （可选）专家权限模型迁移。

     如果您的DB使用的是专家权限模型，并且DB中包含一定数量的表、视图或外表等对象。此时，您必须将原对象迁移至简单权限模型，否则会出现表权限缺失的情况，从而影响业务运行。在该DB中执行如下语句迁移专家权限模型。

     ```
     call spm_migrate();  // 将DB中已有的对象change owner到developer，使用SPM管理。
     ```

     如果您是新创建的DB，DB内无任何对象，则不需要执行此步骤。

     > **说明** 在开启简单权限模型时，需要确保当前DB没有正在运行的SQL，否则可能导致开启失败，并对服务产生影响。
     >
     > 由于migrate可能涉及到将大量表进行Alter Owner操作，触发PostgreSQL对此操作的限制，则spm_migrate每次仅对不超过max_locks_per_transaction的对象进行Change Owner操作。您可能需要多次执行spm_migrate，直到所有对象完成迁移为止。更多关于函数的说明请参见spm_migrate。

  4. 创建用户。

     在为新用户授权之前，您需要将新用户创建至当前实例中。如果新用户已经被创建进实例，则直接进行用户授权。更多关于函数的说明请参见spm_create_user。

     ```
     call spm_create_user('云账号ID/云邮箱/RAM账号'); // 如果创建的用户是RAM用户，则需要在云账号前加"p4_"，即"p4_id"。
     call spm_create_user('云账号ID/云邮箱/RAM账号', '<dbname>_[admin|developer|writer|viewer]'); // 创建用户的同时把用户加入对应的用户组，如果是RAM用户则需要加"p4_"，即"p4_uid"。
     ```

> 说明
> - 如果是RAM用户，执行 `spm_create_user` 时需要在账号UID前加p4_，即“p4_UID”。
> - 在Hologres中，RAM用户的表达格式请参见账号。

5. 授权新用户。

   成功将新用户创建至实例后，必须在对应的DB内将新用户加入相应的用户组，以完成授权操作。授权后，RAM用户就能使用开发工具连接当前DB并在权限范围内进行开发。如果是在创建用户的同时已经加入对应的用户组则不需要再次授权。更多关于函数的说明请参见spm_grant。

   如下命令中，**{dbname}_[admin|developer|writer|viewer]**是将用户加入当前DB中可供选择的用户组名称，更多描述请参见用户组说明。

```
call spm_grant('{dbname}_[admin|developer|writer|viewer]', '云账号id/云邮箱/RAM账号'); // 将某个用户加入某个用户组。
```

   您可以参照如下示例，将用户加入不同权限的用户组。

```
// 加入某个DB的admin用户组。
call spm_grant('mydb_admin', 'p4_564306222995xxx', ); // 将564306222995xxx（RAM用户）加入数据库mydb的admin用户组。
call spm_grant('mydb_admin', '197006222995xxx', ); // 将197006222995xxx（云账号）加入数据库mydb的admin用户组。
call spm_grant('mydb_admin', 'ALIYUN$xxx'); // 将xxx@aliyun.com加入数据库mydb的admin用户组。
// 加入某个DB的developer用户组。
call spm_grant('mydb_developer', 'p4_564306222995xxx'); // 将564306222995xxx（RAM用户）加入数据库mydb的developer用户组。
call spm_grant('mydb_developer', '197006222995xxx'); // 将197006222995xxx（云账号）加入数据库mydb的developer用户组。
call spm_grant('mydb_developer', 'RAM$mainaccount:subuser');// 将云账号mainaccount的RAM用户subuser加入数据库mydb的developer用户组。
// 加入某个DB的viewer用户组。
call spm_grant('"MYDB_viewer"', 'p4_564306222995xxx'); // 将564306222995xxx（RAM用户）加入数据库"MYDB"的viewer用户组。
call spm_grant('"MYDB_viewer"', '197006222995xxx'); // 将197006222995xxx（云账号）加入数据库"MYDB"的viewer用户组。
call spm_grant('mydb_viewer', '"xxx@aliyun.com"'); // 将账号xxx@aliyun.com加入数据库mydb的viewer用户组。
```

移除用户组

在Hologres中，如果您需要将某个用户从某个DB的某个用户组中移除，使用简单权限模型可以使用如下方式：

- 使用管理控制台移除用户组

  您可以在Hologres管理控制台使用可视化方式将用户从某个用户组移除，详情请参见DB管理。

- 使用SQL语句移除用户组

  如果您需要将某个用户从某个DB的某个用户组中移除，可以执行如下命令。更多关于函数的说明请参见spm_revoke。

```
call spm_revoke('<dbname>_[admin|developer|writer|viewer]', '云账号id/云邮箱/RAM账号'); // 移除某用户的权限。
// 示例：
// 将用户从某DB的admin用户组移除。
call spm_revoke('dbname_admin', 'p4_564306222995xxx');//将564306222995xxx（RAM用户）移除admin用户组。
call spm_revoke('dbname_admin', '197006222995xxx');//将197006222995xxx（云账号）移除admin用户组。
call spm_revoke('dbname_admin', '"xxx@aliyun.com"');
// 将用户从某DB的developer用户组移除。
call spm_revoke('mydb_developer', 'RAM$mainaccount:subuser'); // 将RAM用户subuser移出数据库mydb的developer用户组。
call spm_revoke('mydb_developer', 'p4_564306222995xxx');//将564306222995xxx（RAM用户）移除developer用户组。
// 将用户从某个DB的viewer用户组移除。
call spm_revoke('"MYDB_viewer"', 'p4_564306222995xxx'); // 将564306222995xxx（RAM用户）移出数据库"MYDB"的viewer用户组。
```

删除用户

如果您需要从实例中删除某个用户，可以执行如下语句。

> **注意** 成功删除用户后，该用户将会从当前实例删除，用户将无实例任何权限，请谨慎执行该操作。

```
DROP ROLE "云账号ID/云邮箱/RAM账号"; // 直接将该用户从实例中删除。
```

（可选）切换专家模型至简单权限模型

如果您的DB使用的是专家权限模型，并且DB中包含一定数量的表、视图或外表等对象，您可以开启简单权限模型更好的管理权限。您可以通过调用spm_migrate函数将已有对象迁移至简单权限模型，在该DB中执行如下语句。

```
call spm_migrate();  // 将DB已有的对象，change owner到developer，通过SPM管理。
```

> **说明** 在开启简单权限模型时，需要确保当前DB没有正在运行的SQL，否则可能导致开启失败，并对服务产生影响。

切换时可能涉及到将大量表进行Alter Owner的操作，但spm_migrate语句每次仅对不超过max_locks_per_transaction的对象进行Change Owner，您可能需要多次执行spm_migrate，直到所有对象切换完成为止。

关闭简单权限模型

1. 关闭简单权限模型。

   Superuser可以根据业务需求，在该DB中执行如下语句关闭简单权限模型。更多关于函数的说明请参见spm_disable。

   ```
   call spm_disable();
   ```

   关闭简单权限模型之后，对应用户组将不会被删除，用户组内的用户拥有的权限请参见简单权限模型函

数说明。

2. 清除用户组。

   关闭简单权限模型后，您可以根据业务需求，通过调用**spm_cleanup**函数清除用户组。场景如下：

   > **说明** 通常情况下，为了方便管理不建议删除用户组。

   - 场景1：删除用户组保留DB。

     如果您需要删除DB内的用户组，但同时又希望当前DB可以继续使用，Superuser可以执行如下语句。更多关于函数的说明请参见spm_cleanup。

     ```
     call spm_cleanup('{dbname}');
     ```

     > **说明** 调用**spm_cleanup**时，请确保该DB上没有正在运行的SQL语句，否则可能会失败，并可能对服务产生影响。

     由于可能涉及对大量业务表进行Alter Owner操作，**spm_cleanup**每次仅对不超过`max_locks_per_transaction`的对象进行Alter Owner。因此，您可能需要多次执行**spm_cleanup**，直到所有对象完成迁移，并删除四个用户组为止。

   - 场景2：先删除DB再删除用户组。

     如果您已经将原有DB删除，但用户组并未删除，Superuser可以在其他的DB（例如postgres）执行如下语句删除对应用户组。

     ```
     postgres=# call spm_cleanup('mydb');
     ```

关闭简单权限模型的注意事项：

- 只能由Superuser执行关闭操作。
- Public拥有Public Schema的USAGE及CREATE权限。
- Public拥有DB的CONNECT及TEMPORARY权限。
- Public拥有functions及procedures的EXECUTE权限。
- Public拥有`language, data types (include domains)`的USAGE权限。
- Public不拥有其他对象如table、view、materialized view、table column、sequence、foreign data wrapper、foreign server及schema（除public schema）等的权限。
- 关闭简单权限模型之后，用户组的权限如下：
  - admin：保留对当前已有对象的权限，但对新建数据库对象不生效。
  - developer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。
  - writer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。
  - viewer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。

再次开启简单权限模型

如果您的DB已经开启过简单权限模型，后来关闭并使用了专家权限模型。当您因为业务需求，需要再次开启简单权限模型时，您可以通过执行如下语句再次开启

```
call spm_enable('t');  // 开启当前DB的简单权限模型。
call spm_migrate();  // 将DB中已有的对象，change owner到developer，使用SPM管理。
```

> **说明** 在开启简单权限模型时，需要确保当前DB没有正在运行的SQL，否则可能导致开启失败，并对服务产生影响。

切换时可能涉及到将大量表进行Alter Owner的操作，但**spm_migrate**语句每次仅对不超过max_locks_per_transaction的对象进行Change Owner，您可能需要多次执行**spm_migrate**，直到所有对象切换完成为止。

4.5.2.3. 简单权限模型函数说明

本文为您介绍在实时数仓Hologres中简单权限模型（Simple Permission Model，SPM）的函数调用说明，您可以通过调用相关函数管理简单权限模型。

函数概述

简单权限模型的函数及其功能说明如下所示。

- spm_enable：开启简单权限模型。
- spm_migrate：将已有实例对象（包括表、视图及外表等）切换至SPM权限模型。
- spm_create_user：创建仅具有登录权限的用户，您需要授予其具体的权限才能进行开发。
- spm_grant：将某个用户加入到某个用户组。
- spm_revoke：将某个用户从某个用户组中移除。
- spm_disable：关闭当前DB简单权限模型。
- spm_cleanup：清除DB的SPM保留用户组。

spm_enable

- 函数介绍

  spm_enable()函数用于开启简单权限模型。

  调用**spm_enable()**函数后，系统将会自动创建{db}_admin、{db}_developer、{db}_writer和{db}_viewer 4个用户组。

- 命令语法

```
CALL spm_enable();
```

> **说明** 仅实例的Superuser可以执行**spm_enable()**函数。

- 使用说明

  调用**spm_enable()**函数成功开启SPM的使用说明如下：

  - DB的所有权限从PUBLIC用户组收回。因此，不相关的用户将不能连接到目标DB。
  - DB下所有Schema的所有权限从PUBLIC用户组收回。
  - {db}_admin、{db}_developer、{db}_writer和{db}_viewer都具有DB的连接权限。
  - {db}_admin将作为DB的Owner以及DB下所有Schema的Owner。
  - {db}_developer、{db}_writer和{db}_viewer具有所有Schema的USAGE权限，{db}_developer具有所有Schema的CREATE权限。
  - {db}_admin和{db}_developer未来创建的对象，Owner都是{db}_developer。{db}_writer具有读写（readwrite）权限，{db}_viewer具有只读（readonly）权限。

spm_migrate

- 函数介绍

  spm_migrate()函数用于将已有实例对象（包括表、视图及外表等）切换至SPM权限模型。

- 命令语法

```
CALL spm_migrate( [ batch_size ] );
```

  参数说明如下表所示。

参数	描述	取值范围
batch_size	单次批量迁移对象的大小。 取值为*0*代表采用当前max_locks_per_transaction作为batch_size。	*[0, max_locks_per_transaction]*，超出该范围则不合法。其中 `max_locks_per_transaction` 最大取值为64. 如果您的实例中有大量的对象（超过数千甚至数万）需要切换至简单权限模型，需要您多次执行**spm_migrate()**函数，直到全部对象切换完成为止。 同时，建议您提交工单将max_locks_per_transaction参数调大，再执行操作。

- 使用说明
  - 调用**spm_migrate()**函数时，当遇到 `DONE BUT NOT COMPLETED` 提示，说明全部对象还没有迁移完毕，您需要继续执行**spm_migrate()**迁移。
  - 当遇到 `COMPLETED` 提示，说明全部对象已经迁移完毕。

- 使用示例

```
CALL spm_migrate(); //切换最多max_locks_per_transaction个对象至SPM管理。
CALL spm_migrate(128); // 切换128个对象至SPM管理。
```

spm_create_user

- 函数介绍

  spm_create_user()函数用于在SPM中创建用户。创建的用户仅具有登录权限，您需要授予其具体的权限才能进行开发。

> **说明** 仅Superuser和{db}_admin的成员可以调用该函数。

- 命令语句

```
CALL spm_create_user( user_name [, role_name] );
```

  参数说明如下表所示。

参数	描述

参数	描述
user_name	需要创建的用户名。格式如下： ○ 阿里云账号，例如13432193xxxx或者xx@aliyun.com。 ○ RAM用户账号，例如RAM$mainaccount:subuser或者p4_202338382183xxx。
role_name	创建用户时可以根据业务需求选择将用户加入如下用户组： ○ {db}_admin ○ {db}_developer ○ {db}_writer ○ {db}_viewer

- 使用示例

```
CALL spm_create_user('my_test@aliyun.com');
CALL spm_create_user('RAM$my_test:mysubuser', 'mydb_developer');
CALL spm_create_user('13532313103042xxx');
CALL spm_create_user('p4_23319103042xxx', 'mydb_admin');
```

spm_grant

- 函数介绍

  spm_grant()函数用于将某个用户加入到{db}_admin、{db}_developer、{db}_writer或{db}_viewer用户组。

  > 说明　仅Superuser和{db}_admin可以调用该函数。

- 命令语法

```
CALL spm_grant( role_name, user_name );
```

  参数说明如下表所示。

参数	描述
role_name	可以根据业务需求选择将用户加入如下用户组： ○ {db}_admin ○ {db}_developer ○ {db}_writer ○ {db}_viewer 关于用户组的权限说明，请参见简单权限模型概述。
user_name	需要被添加进用户组的用户名。格式如下： ○ 阿里云账号，例如13432193xxxx或者xx@aliyun.com。 ○ RAM用户账号，例如RAM$mainaccount:subuser或者p4_202338382183xxx。

- 使用说明
  - 只有开启简单权限模型后才能调用该函数。

- user_name必须是云账号ID或者云账号。
- role_name必须是{db}_admin、{db}_developer、{db}_writer或{db}_viewer用户组。

- 使用示例

```
CALL spm_grant('mydb_developer', 'p4_202338382183xxx');//授予RAM用户mydb的developer权限。
CALL spm_grant('mydb_admin', 'RAM$my_test:xxx');//授予RAM用户mydb的admin权限。
CALL spm_gran('otherdb_admin', '13532313103042xxx'); // 错误，加入其他DB的角色请连接到其他DB执行spm_grant函数。
```

spm_revoke

- 函数介绍

  spm_revoke()函数用于将某个用户从{db}_admin、{db}_developer、{db}_writer或{db}_viewer用户组中移除。

  > **说明** 仅Superuser和{db}_admin可以调用该函数。

- 命令语法

```
CALL spm_revoke( role_name, user_name );
```

- 使用说明
  - 只有开启简单权限模型后才能调用该函数。
  - user_name必须是云账号ID或者云账号。
  - role_name必须是{db}_admin、{db}_developer、{db}_writer或{db}_viewer用户组。

- 使用示例

```
CALL spm_revoke('mydb_developer', 'p4_202338382183xxx');//将RAM用户从mydb的developer用户组移除。
CALL spm_revoke('mydb_admin', 'RAM$my_test:xxx');//将RAM用户从mydb的admin用户组移除。
CALL spm_revoke('otherdb_admin', '13532313103042xxx'); // 错误，移出其他DB的用户组请连接到其他DB执行spm_revoke。
```

spm_disable

- 函数介绍

  spm_disable()函数用于为当前DB关闭简单权限模型。

  > **说明** 仅Superuser可以调用该函数。

- 命令语句

```
CALL spm_disable();
```

- 使用说明

  关闭SPM的说明如下：

  - {db}_admin、{db}_developer、{db}_writer和{db}_viewer用户组继续保留，可以使用专家模式授权语句将用户加入对用户组来做权限管理，数据库对象Owner保持不变（仍然是{db}_developer）。
  - DB的CONNECT，TEMPORARY权限向PUBLIC开放。

- DB中public schema的USAGE，CREATE权限向PUBLIC开放。
- DB中public schema中function，procedure的EXECUTE权限向PUBLIC开放。
- 用户定义的LANGUAGE的USAGE权限向PUBLIC开放。
- 用户定义的TYPE的USAGE权限向PUBLIC开放。

- 使用示例

```
CALL spm_disable();
```

spm_cleanup

- 函数介绍

  spm_cleanup()函数用于清除DB的SPM保留用户组，包括{db}_admin、{db}_developer、{db}_writer和{db}_viewer。

  > **说明** 仅Superuser可以调用此函数。

- 命令语句

```
CALL spm_cleanup( db_name [, batch_size ] );
```

参数说明如下表所示。

参数	描述	取值范围
db_name	需要清除用户组的DB。包含特殊字符或大写字母的db_name用双引号引起来。例如 `"MYDB"` 。	无
batch_size	单次批量迁移对象的大小。 取值为*0*代表采用当前max_locks_per_transaction作为batch_size。	*[0, max_locks_per_transaction]*，超出该范围则不合法。 如果您的实例中有大量的对象（超过数千甚至数万）需要迁移，可能会导致锁溢出，需要您多次执行**spm_migrate()**函数，直到全部对象迁移完成为止。 同时，建议您提交工单将max_locks_per_transaction参数调大，再执行迁移操作。

- 使用说明
  - 当遇到 `DONE BUT NOT COMPLETED` 提示，说明全部对象还没有迁移完毕，并且未删除保留用户组，您需要继续执行**spm_cleanup()**函数。
  - 当遇到 `COMPLETED` 提示，说明全部对象已经迁移完毕，并已删除保留用户组，无需再调用此函数。

- 使用示例

```
CALL spm_cleanup('mydb'); //单次最多将max_locks_per_transaction个对象转移Owner到current_user。
CALL spm_cleanup('mydb', 128);//单次转移128个对象的Owner到current_user。
```

- 场景1：删除对应DB，再清除用户组。

```
drop database mydb;
CALL spm_cleanup('mydb');  // 无需重试，一次成功。
```

- 场景2：在DB仍然存在的情况下，需要连接到对应DB进行清除。

```
 CALL spm_cleanup('otherdb');
ERROR:  Permission Denied. execute in database otherdb, or drop database before call spm_cleanup.
```

功能函数调用权限

用户组对功能函数调用的权限如下表所示。

功能函数	作用	Superuser	db_admin	db_developer | db_writer | db_viewer
spm_enable	开启简单权限模型。	是	否	否
spm_disable	关闭简单权限模型。	是	否	否
spm_grant	将user加入组。	是	是	否
spm_revoke	将user移出组。	是	是	否
spm_alter_database_rename	修改DB名称，并修改所有Schema的保留用户组名称。	是	是	否
spm_migrate	将已有的表或类表对象等迁移至SPM管理。	是	是	否
spm_cleanup	删除DB所有保留用户组。	是	否	否
spm_create_user	创建简单用户并该用户仅拥有登录权限。	是	是	否

相关权限命令限制

开启简单权限模型之后，部分权限相关命令会受到限制，不能在简单模型中使用的命令如下表所示。

命令语句	说明
alter table owner to xx	所有table owner都自动是对应Schema的developer用户组，不能更改，无需手动更改。
grant	通过 `spm_grant` 将用户加入某个用户组后，无需再执行grant语句。
revoke	通过 `spm_revoke` 将用户加入某个用户组后，无需再执行revoke语句。

命令语句	说明
alter default privileges	在原权限模型下，授权只对当前以及过去的表有权限，未来表还需要重新授权。在简单模型下，无需考虑建表的时态，只需要加入某个用户组就能拥有对应的权限，因此无需再给未来表授权。
create role / drop role / alter role / alter role set默认用户组	db_admin、db_developer及db_viewer为系统默认用户组，在开启简单权限模型后，会默认生成，不允许用户（包括Superuser）对默认用户组进行创建或修改。
rename to/from默认用户组	不可以对保留用户组（`*_admin`, `*_developer`, `*_writer`, `*_viewer`）进行更名。

4.5.3. 专家权限模型

Hologres兼容PostgreSQL，采用与标准PostgreSQL语句相同的授权体系（简称专家模式）。本文为您介绍Hologres如何使用专家权限模型对用户授权及撤销授权。

专家权限模型授权

在Hologres实例连接开发工具后，可以使用SQL语句通过专家权限模型授权，使该用户具有实例的相关权限。

1. 创建用户。

   一个账号必须先被创建成为Hologres的用户，才能访问Hologres并进行开发。

   创建用户的语句格式如下：

   ```
   --创建具有登录Hologres实例权限的用户，如果是为RAM用户授权，账号格式请使用RAM用户的表达格式。
   CREATE USER "云账号ID/云邮箱";
   --创建用户并授予Superuser权限。
   CREATE USER "云账号ID/云邮箱" SUPERUSER;
   ```

   您可以参照如下示例创建用户，其中更多关于阿里云账号和RAM用户的格式说明，请参见账号概述。

   ```
   --使用阿里云账号ID创建用户。
   CREATE USER "11822780xxx";
   --授予RAM用户Superuser权限。
   CREATE USER "p4_1822780xxx" SUPERUSER;
   ```

   更多关于创建角色的说明，请参见CREATE ROLE。

2. 授予权限。

   将账号创建为Hologres的用户后，您需要授予用户一定的权限，该用户才能在权限范围内使用Hologres。Hologres中常用的授权操作如下表所示。

   > **说明** 目前专家模式只能对现有实例对象进行授权，对授权后创建的内容不生效。例如，用户A对用户B授予了public schema中所有表的查看权限。当用户A创建一张新表，则用户B不具有对这张表的查看权限，需要重新授权。

权限描述	语法示例	是否必须
创建具有登录Hologres实例权限的用户	`CREATE USER "云账号/云邮箱";`	是
创建用户并授予用户Superuser的权限	`CREATE USER "云账号/云邮箱" SUPERUSER ;`	可选
授予在Schema下创建表的权限	`GRANT CREATE ON SCHEMA schema_name TO "云账号/云邮箱";`	可选
授予Schema访问权限	`GRANT USAGE ON SCHEMA schema_name TO "云账号/云邮箱";`	必须 **说明** 必须授予Schema的访问权限才能有表的查询权限。
授予所有用户public schema中所有表的查看、写入、及修改权限	`GRANT SELECT,INSERT,UPDATE ON ALL TABLES IN SCHEMA public to PUBLIC;`	可选
授予用户某个表的SELECT权限	`GRANT SELECT ON TABLE <tablename> TO "云账号/云邮箱";`	可选
授予用户某个表的SELECT权限，并允许该用户授予此权限给其他用户	`GRANT SELECT ON TABLE <tablename> TO "云账号/云邮箱" WITH GRANT OPTION;`	可选
授予用户public schema中所有表的SELECT权限	`GRANT SELECT ON ALL TABLES IN SCHEMA public TO "云账号/云邮箱";`	可选
默认所有用户具有public schema中所有表（包含未来表）的读权限	`ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO PUBLIC;`	可选
修改普通用户为Superuser	`ALTER USER "云账号/云邮箱" SUPERUSER;`	可选

权限描述	语法示例	是否必须
修改Superuser为普通用户	`ALTER USER "云账号/云邮箱" NOSUPERUSER;`	可选
授予其他用户表的Owner权限	`ALTER TABLE <tablename> OWNER TO "云账号/云邮箱";`	可选
创建没有登录Hologres实例权限的角色	`CREATE ROLE "云账号/云邮箱";`	可选
授予某个用户某个角色的权限	`GRANT <rolename> TO "云账号/云邮箱" ;`	可选

在专家模型下，您可以参照以下示例给一个新用户授予某张表的查询权限。

```
CREATE USER "云账号/云邮箱";
GRANT USAGE ON SCHEMA <schema_name>  TO "云账号/云邮箱";
GRANT SELECT ON TABLE <tablename> TO "云账号/云邮箱";
```

CREATE ROLE用于创建没有登录Hologres实例权限的角色，例如代表一类具体用户的用户组或虚拟角色等。更多关于权限的授予的说明，请参见GRANT。

3. 删除表。

   只有Superuser或表Owner才可以删除表。您可以使用如下几种方法授予某个用户或多个用户删除表的权限：

   - 替换新用户为表的Owner。

     ```
     ALTER TABLE TABLENAME OWNER TO "云账号/云邮箱";
     ```

   - 授予新用户Superuser权限。

     ```
     ALTER USER "云账号/云邮箱" SUPERUSER;
     ```

   - 添加多个用户至用户组并授予表Owner权限。

     ```
     CREATE USER "云账号ID/云邮箱";
     CREATE ROLE <rolename>;
     GRANT <rolename> TO "云账号/云邮箱";
     ALTER TABLE <tablename> OWNER TO <rolename>;
     ```

未来表授权

由于专家模式授权不包含对未来表的授权，因此需要使用ALTER DEFAULT PRIVILEGES语句对未来表进行授权。具体操作步骤如下：

> **说明**
>
> - 该命令语句不影响已有的逻辑对象。
> - 该命令语句只能设置TABLE、SCHEMA、FUNCTION、SEQUENCE或TYPE的默认权限。

1. 授权。
   - 默认执行授权语句的人，在某个Schema下新建的表对所有人或指定人进行查询。示例语句如下。
     - 执行授权的人在public schema下，新建的表可以被子账号p4_123xxx查询。

       ```
       ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO "p4_123xxx";
       ```

     - 执行授权的人在test schema下，新建的表可以被子账号p4_123xxx查询。

       ```
       ALTER DEFAULT PRIVILEGES IN SCHEMA test GRANT SELECT ON TABLES TO "p4_123xxx";
       ```

     - 执行授权的人在public schema下，新建的表可以被所有人查询。

       ```
       ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO PUBLIC;
       ```

   - 默认授权后，其他人新建表时可被指定所有人或指定人进行查询。示例语句如下。
     - 授权后，p4_id1新建的表可以被public schema下的所有用户访问。

       ```
       ALTER DEFAULT PRIVILEGES FOR ROLE "p4_id1" IN SCHEMA public GRANT SELECT ON TABLES
       TO PUBLIC;
       ```

     - 授权后，p4_id1新建的表可以被public schema下的子账号p4_id2访问。

       ```
       ALTER DEFAULT PRIVILEGES FOR ROLE "p4_id1" IN SCHEMA public GRANT SELECT ON TABLES
       TO "p4_id2";
       ```

     - 授权后，p4_id1新建的表可以被test schema下的所有用户访问。

       ```
       ALTER DEFAULT PRIVILEGES FOR ROLE "p4_id1" IN SCHEMA test GRANT SELECT ON TABLES TO
       PUBLIC;
       ```

   - 撤销设置的默认授权，有的场景您需要撤销您设置的默认授权，示例语句如下。

     ```
     --撤销执行授权的人在public schema下未来新建的表可以被子账号p4_123xxx查询的默认授权
     ALTER DEFAULT PRIVILEGES IN SCHEMA public REVOKE SELECT ON TABLES FROM "p4_123xxx";
     --撤销执行授权的人在test schema下未来新建的表可以被子账号p4_123xxx查询的默认授权
     ALTER DEFAULT PRIVILEGES IN SCHEMA test REVOKE SELECT ON TABLES FROM "p4_123xxx";
     --撤销执行授权的人在public schema下未来新建的表可以被所有人查询的默认授权
     ALTER DEFAULT PRIVILEGES IN SCHEMA public REVOKE SELECT ON TABLES FROM PUBLIC;
     ```

2. 查看默认权限是否设置成功。
   - 使用\ddp命令在Psql客户端查看ALTER DEFAULT PRIVILEGES是否设置成功。
   - 使用如下SQL命令在Hologres中直接查询。

     ```
     SELECT pg_catalog.pg_get_userbyid(d.defaclrole) AS "Owner",
       n.nspname AS "Schema",
       CASE d.defaclobjtype WHEN 'r' THEN 'table' WHEN 'S' THEN 'sequence' WHEN 'f' THEN '
     function' WHEN 'T' THEN 'type' WHEN 'n' THEN 'schema' END AS "Type",
       pg_catalog.array_to_string(d.defaclacl, E'\n') AS "Access privileges"
     FROM pg_catalog.pg_default_acl d
          LEFT JOIN pg_catalog.pg_namespace n ON n.oid = d.defaclnamespace
     ORDER BY 1, 2, 3;
     ```

   创建新表时，Hologres会使用当前用户和模式去匹配系统表*pg_catalog.pg_default_acl*。如果检查到匹配项ALTER DEFAULT PRIVILEGES，则为用户添加匹配项规则。当前用户说明如下：

- 如果当前用户是User，则创建表时使用User进行匹配。
- 如果用户User创建表之前执行了 `SET SESSION ROLE GROUP1;` 语句，此时当前用户就变为GROUP1，则创建表时使用GROUP1进行匹配。

匹配规则只在创建表时执行，在创建表之后执行 `ALTER TABLE SET OWNER TO` 语句修改表Owner，不会触发对应匹配项规则。

专家模式撤销授权

使用REVOKE语句撤销用户权限的示例如下，更多关于权限的撤销操作，请参见REVOKE。

```
REVOKE SELECT ON TABLE tablename FROM "云账号ID/云邮箱" ; --如果是RAM用户，账号格式请使用RAM用户的表达格式。
```

查看权限

通过以下SQL命令查看用户的角色及权限。

```
SELECT ROLNAME FROM pg_roles;
SELECT user_display_name(ROLNAME) FROM pg_roles;
```

删除用户

若您的实例已经连接开发工具，您可以使用SQL语句进行删除子账号，分为如下两种情况。

- 删除普通用户

  如果是删除普通用户，并且该账号没有创建其他对象（如表、视图、extension等），可以执行以下命令语句或者直接在HoloWeb删除用户。

  ```
  drop user "云账号ID/云邮箱";
  ```

- 删除Superuser等管理员

  若是要删除Superuser、Admin等管理员，但是该账号创建过表、视图、extension等实例内对象，并且是这些对象的管理员（尤其是专家权限模型下），若是直接删除用户会报错，需要先将该账号下的对象进行转移，执行以下命令语句。

  ```
  -- 将A账号下的对象转给B
  reassign owned by "A云账号ID" to "B云账号ID";
  -- 删除A账号
  drop user "A云账号ID";
  ```

您可以使用以下方式删除实例中的RAM用户：

```
DROP USER "云账号ID/云邮箱";
```

> **注意** RAM用户被删除后，将不能连接实例并访问实例内的任何对象，请您谨慎操作。

标准的PostgreSQL对于权限有着非常严格的划分，对此我们提供最佳实践供您根据业务需求选择和参考，详情请参见基于PostgreSQL标准权限模型授权。

4.5.4. Schema级别的简单权限模型（SLPM）

4.5.4.1. 基于Schema级别的简单权限模型概述

本章节内容将会为您介绍在Hologres中基于Schema级别的简单权限模型。

背景信息

实时数仓Hologres兼容PostgreSQL，使用与Postgres完全一致的权限系统（简称专家模式，详情请参见专家权限模型）。专家权限模型授权较为细致，在此基础上，Hologres提供两种简单权限模型供业务使用：

- 简单权限模型（Simple Permission Model）：简称SPM，该权限模型授权是以DB为粒度，提供简单方便的授权操作，详情请参见简单权限模型。
- 基于Schema级别的简单权限模型(Schema-Level Permission Model)：简称SLPM，该权限模型基于Schema划分，相比于简单权限模型更为细粒度，若是对权限有严格划分且又希望授权操作简便可以使用该权限模型。

> **说明** Hologres管理控制台暂未提供SLPM授权操作，您需要使用连接工具对接Hologres实例后使用SQL命令进行授权操作。

基于Schema级别的简单权限模型介绍

当您开启了基于Schema级别的简单权限模型时，每个DB会默认产生如下几种权限等级的用户组：

- 超级管理员：Superuser
- DB管理员：{db}.admin
- 开发者：{db}.{schema}.developer
- 读写者：{db}.{schema}.writer
- 分析师：{db}.{schema}.viewer

每个角色对应的权限如下表所示：

角色	权限
Superuser	整个实例的管理员，拥有实例的所有权限。
{db}.admin	• 某个DB的管理员（admin）。 • admin组的权限是developer组、writer组及viewer组权限的合集。 • 某个DB的owner，可以删除DB。 • 可管理当前DB的{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer和{db}.{schema}.viewer四个用户组的成员，包括新增及移除用户组成员。 • 可以创建用户，并将用户加入某个用户组。 • 可以在该DB创建对象，例如Schema，并可以对对象进行增删改查。 • 可以在DB级别修改DB的配置项。

角色	权限
{db}.{schema}.developer	• DB下某个Schema的开发者（developer）。 • {db}.{schema}.developer组的权限是{db}.{schema}.writer组及{db}.{schema}.viewer组权限的合集。 • 该DB对应Schema中，除系统对象以外的所有表、外表、类表对象（如视图等）、Function、Procedure、Foreign Server、FDW、Type及Language的owner，可以增删查改该Schema中的所有表。 • 可使用DB级别的Foreign Server、FDW、Type及Language等对象。 • 该Schema的USAGE及CREATE权限，可以在任意非系统的Schema中进行创建表，创建视图及创建外表等DDL操作。
{db}.{schema}.writer	• DB下某个Schema的读写者（writer）。 • {db}.{schema}.writer组的权限是{db}.{schema}.viewer组权限的合集。 • Schema中所有表、外表及类表对象（如视图等）的数据，即拥有SELECT、INSERT、UPDATE及DELETE等权限。 • 可以增删查改该Schema下的对象。 • 可以访问或使用Schema的Function、Procedure。 • 可使用DB级别的Foreign Server、FDW、Type及Language等对象。 • 该Schema的USAGE权限，不可进行DDL操作。
{db}.{schema}.viewer	• DB的分析师（viewer）。 • 可以读取该Schema下所有表、外表及类表对象（如视图等）的数据，即拥有SELECT权限。 • 可以访问或使用Schema下的Function、Procedure。 • 可使用DB级别的Foreign Server、FDW、Type及Language等对象。 • 该Schema的USAGE权限。

4.5.4.2. 基于Schema级别的简单权限模型的使用

本文为您介绍在交互式分析Hologres中使用SQL语句实现基于Schema级别的简单权限模型（Schema-level Permission Model，SLPM）的使用方法。

SLPM的使用限制

由于SLPM是基于Schema级别严格进行权限管控的，因此在授权和使用时需要您注意如下的限制条件：

- **当您创建一个View或者Rule，引用了跨Schema的两个或多个表时，由于Schema间权限不互通的原因，此View或者Rule将无法被访问。因此不建议您在SLPM管理的DB中创建跨Schema的View或Rule对象。**
- 开启SLPM之后，只会开放特定的权限，开放权限请参见基于Schema级别的简单权限模型授权。以下表格中的DDL功能将不能在SLPM使用，请您使用对应的SLPM语句执行相关操作，具体函数说明请参见基于Schema级别的简单权限模型函数说明。

命令语句	说明	SLPM语句

命令语句	说明	SLPM语句
alter table owner to xx	所有table owner都是对应Schema的developer用户组，不支持修改。	无需手动执行。
grant	将用户加入某个用户组后，会被授予对应用户组的权限，无需再执行grant授权语句。	slpm_grant
revoke	若要移除某个用户的某个权限，只能通过将该用户移除某个用户组的方法来实现，不能单独执行revoke语句。	slpm_revoke
alter default privileges	在原有权限模型下，授权只对当前以及过去的表有权限，未来表需要重新授权。在简单模型下，无需考虑建表的时态，只需要加入某个用户组就能拥有对应的权限，因此无需再给未来表授权。	无需手动执行。
create / drop / alter / rename默认用户组	开启SLPM后，会默认生成admin/developer/writer/viewer等用户组，不允许用户（包括Superuser）对默认用户组进行创建、修改和删除。	不涉及。
rename schema	Schema重命名不支持在某个DB直接执行`alter rename schema`命令，需要调用`slpm_rename_schema`命令实现。	slpm_rename_schema
rename database	DB重命名不支持直接执行`alter rename database`命令，需要调用`slpm_rename_database`命令实现。	slpm_rename_database
drop database	删除DB需要在执行`drop database`之后，调用`slpm_cleanup('<dbname>')`来清除默认用户。	需要执行`drop database`之后，调用`slpm_cleanup('<dbname>')`来清除默认用户。

基于Schema级别的简单权限模型授权

在Hologres实例连接开发工具后，可以使用SQL语句通过基于Schema级别的简单模式对用户进行授权，使该用户具有对应Schema的相关权限。

1. 开启函数调用。

   开启SLPM前，您需要执行如下命令，开启调用函数的开关。示例中的extension是DB维度的，在同一个DB只需执行一次。

   ```
   create extension slpm;
   ```

2. 开启SLPM。

   SLPM默认不开启，需要Superuser在目标DB里执行如下语句进行开启。同时，在开启SLPM时，需要确保当前DB没有正在运行的SQL，否则可能导致开启失败，并对服务产生影响。

   ```
   call slpm_enable ();  // 开启当前DB的SLPM。
   ```

> **说明** 为当前DB开启SLPM之后，您必须把使用该DB的所有用户加入至对应的用户组，否则这些用户将无实例的连接权限，可能对业务造成影响。具体操作请参见授权新用户。

3. （可选）专家权限模型切换为SLPM。

   您可以进入Hologres管理控制台，从**DB授权**页面查看当前使用的权限模型。

   如果您的DB使用的是专家权限模型，并且DB中包含一定数量的表、视图或外表等对象。此时，如果您需要开启SLPM进行权限管理，可以通过调用**slpm_migrate**函数将原用户的权限由专家模型权限切换为SLPM。

   ```
   call slpm_migrate ();  // 将DB中已有的对象change owner到develoepr，使用SLPM管理。
   ```

   在将专家权限模型切换为SLPM时，需要您注意如下内容：

   slpm_migrate函数默认每次仅对64个（参数可调）用户进行权限切换。如果涉及用户对象数量过多，需要多次执行该函数，直到全部用户权限切换完毕。更多关于该函数的说明，请参见slpm_migrate。

4. 创建用户至当前实例。

   在为新用户授权之前，您需要将新用户创建至当前实例中。如果新用户已经被创建进实例，可跳过该步骤。

   如下命令中，**{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]**是将用户加入当前DB中可供选择的用户组名称，更多描述请参见用户组说明。

   ```
   call slpm_create_user ('云账号ID/云邮箱/RAM账号'); // 创建用户，使用云邮箱时还需要加双引号。
   call slpm_create_user ('云账号ID/云邮箱/RAM账号', '{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]');  // 创建用户的同时把用户加入对应的用户组。
   ```

   > **说明**
   > - 如果使用RAM用户UID账号进行授权，执行 `slpm_create_user` 时需要在UID前添加 `p4_` ，即 `p4_UID` 。在Hologres中，请前往用户页面获取UID账号。更多RAM账号表达格式请参见账号概述。
   > - 如果您需要批量创建用户至当前实例，请前往Hologres管理控制台，通过**用户管理**页签批量新增用户。请参见新增用户。

5. 授权新用户。

   成功将新用户创建至实例后，必须在对应的DB内将新用户加入相应的用户组，以完成授权操作。如果是在创建用户的同时已经加入对应的用户组则不需要再次授权。

   如下命令中，**{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]**是将用户加入当前DB中可供选择的用户组名称，更多描述请参见用户组说明。

   ```
   call slpm_grant ('{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]', '云账号id/云邮箱/RAM账号'); // 将某个用户加入某个用户组
   ```

   您可以参照如下示例，将用户加入不同权限的用户组。

```
// 加入某个DB的admin用户组
call slpm_grant ('mydb.admin', '197006222995xxx', ); // 将197006222995xxx加入数据库mydb的admin用户组
call slpm_grant ('mydb.admin', 'ALIYUN$xxx'); // 将xxx@aliyun.com加入数据库mydb的admin用户组
// 加入某个DB的developer用户组
call slpm_grant ('mydb.public.developer', '197006222995xxx'); // 将197006222995xxx加入数据库mydb的developer用户组
call slpm_grant ('mydb.public.developer', 'RAM$mainaccount:subuser');// 将云账号mainaccount的RAM用户subuser加入数据库mydb的developer用户组
// 加入某个DB的viewer用户组
call slpm_grant ('"MYDB.lisa.viewer"', '197006222995xxx'); // 将197006222995xxx加入数据库"MYDB"的viewer用户组
call slpm_grant ('mydb.lisa.viewer', '"xxx@aliyun.com"'); // 将账号xxx@aliyun.com加入数据库mydb的viewer用户组
```

移除用户组

如果您需要将某个用户从某个DB的某个用户组中移除，可以执行如下命令。移除之后，该用户将不具备该用户组具有的权限。

如下命令中，{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]是将用户加入当前DB中可供选择的用户组名称，更多描述请参见用户组说明。

```
call slpm_revoke ('{dbname}.[admin|{schemaname}.developer|{schemaname}.writer|{schemaname}.viewer]', '云账号id/云邮箱/RAM账号'); // 移除某用户的权限
```

您可以参照如下示例，将用户从不同权限的用户组移除。

```
// 将用户从某DB的admin用户组移除
call slpm_revoke ('dbname.admin', 'p4_564306222995xxx');//将564306222995xxx（RAM用户）移除admin用户组
call slpm_revoke ('dbname.admin', '197006222995xxx');//将197006222995xxx（云账号）移除admin用户组
call slpm_revoke ('dbname.admin', '"xxx@aliyun.com"');
// 将用户从某DB的developer用户组移除
call slpm_revoke ('mydb.lisa.developer', 'RAM$mainaccount:subuser'); // 将RAM用户subuser移出数据库mydb的developer用户组
call slpm_revoke ('mydb.public.developer', 'p4_564306222995xxx');//将564306222995xxx（RAM用户）移除developer用户组
// 将用户从某个DB的viewer用户组移除
call slpm_revoke ('"MYDB.SCHEMA1.viewer"', 'p4_564306222995xxx'); // 将564306222995xxx（RAM用户）移出数据库"MYDB"的viewer用户组
```

删除用户

您也可以根据需要对某个用户进行删除，成功删除用户后，该用户将会从当前实例删除，用户将无实例任何权限，请谨慎执行该操作。

```
DROP ROLE "云账号ID/云邮箱/RAM账号"; // 直接将该用户从实例中删除。
```

基于Schema级别的简单权限模型关闭

当您不需要使用基于Schema级别的简单权限模型时，您可以根据业务需求执行如下操作，关闭此功能。

1. 关闭SLPM。

   开启SLPM后，Superuser也可以根据业务需要执行如下示例内容关闭SLPM。关闭SLPM之后，对应用户组将不会被删除，对应用户组内用户拥有的权限请参见基于Schema级别的简单权限模型函数说明。

   ```
   call slpm_disable ();
   ```

   关闭SLPM注意事项：

   - 只有Superuser可以执行关闭操作。
   - 将对PUBLIC开放public schema的USAGE和CREATE权限、DB的CONNECT和TEMPORARY权限、functions和procedures的EXECUTE权限以及language和data types (include domains) 的USAGE权限。
   - 对于其他对象如table、view、materialized view、table column、sequence、foreign data wrapper、foreign server、schema（除public schema）等不会将权限开放给PUBLIC。如果您需要开放此权限，请联系Superuser授权。
   - 关闭SLPM之后，{db}.admin、{db}.{schemaname}.developer、{db}.{schemaname}.writer和{db}.{schemaname}.viewer将保留对当前已有对象的权限，对新建数据库对象不生效。

2. 删除用户组（通常情况下，为了方便管理不建议删除用户组）。

   关闭SLPM后，您可以根据业务需求，通过调用**slpm_cleanup**函数删除用户组。场景如下：

   - 场景1：删除用户组保留DB。

     如果您需要删除DB内的用户组，但同时又希望当前DB可以继续使用，Superuser可以执行如下语句。

     ```
     call slpm_cleanup ( '<dbname>' );
     ```

     > **说明** 调用**slpm_cleanup**时，请确保该DB上没有正在运行的SQL语句，否则可能会失败，并可能对服务产生影响。

     由于**slpm_cleanup**需要将现有对象owner转移给当前用户，但**slpm_cleanup**默认每次仅对64个（参数可调）对象转移owner。因此，您可能需要多次执行**slpm_cleanup**，直到所有对象完成迁移（建议累计执行次数小于5次），并删除所有保留用户组为止。更多关于该函数的说明，请参见slpm_cleanup。

   - 场景2：先删除DB再删除用户组。

     如果您已经将原有DB删除，但用户组并未删除，Superuser可以在其他的DB（例如postgres）执行如下语句删除原有DB对应的所有用户组。

     ```
     postgres=# call slpm_cleanup ( 'mydb' );
     ```

重新开启SLPM

如果您因为某些原因需要重新开启SLPM，步骤如下。

1. 清除用户权限

   为了避免用户权限混乱，在开启SLPM之前，建议执行如下命令将数据库中用户现有的权限全部清除。

   ```
   call slpm_cleanup ( '<dbname>' );
   ```

2. 重新开启SLPM权限模型

清除用户权限后，执行以下命令开启SLPM权限模型。

```
call slpm_enable ('t');
call slpm_migrate ();
```

3. 用户授权

   重新开启SLPM之后，可以在Hologres管理控制台或者使用SQL命令为用户重新授权，详情请参见用户授权。

4.5.4.3. 基于Schema级别的简单权限模型函数说明

本文为您介绍在交互式分析Hologres中基于Schema级别的简单权限模型（Schema-level Permission Model，SLPM）的函数调用说明，您可以通过调用相关函数管理Schema级别的简单权限模型。

函数概述

基于Schema级别的简单权限模型的函数及其功能说明如下所示。

- slpm_enable：开启基于Schema级别的简单权限模型。
- slpm_migrate：将已有实例对象（包括表、视图及外表等）迁移至SLPM权限模型。
- slpm_create_user：在SLPM中创建用户。
- slpm_grant：将某个用户加入到{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组。
- slpm_revoke：将某个用户从{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组中移除。
- slpm_disable：为当前DB关闭SLPM。
- slpm_cleanup：清除DB的SLPM保留用户组，包括{db}.admin，{db}.{schema}.developer，{db}.{schema}.writer和{db}.{schema}.viewer。
- slpm_rename_schema：重命名Schema。

slpm_enable

- 函数介绍

  slpm_enable()函数用于开启基于Schema级别的简单权限模型。

  调用**slpm_enable()**函数后，系统将会自动创建{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer和{db}.{schema}.viewer 4个用户组。

- 命令语法

  ```
  CALL slpm_enable ();
  ```

  > **说明** 仅实例的Superuser可以执行**slpm_enable()**函数。

- 使用说明

  调用**slpm_enable()**函数成功开启SLPM的使用说明如下：

  - DB的所有权限从PUBLIC用户组收回。因此，不相关的用户将不能连接到目标DB。
  - DB下所有Schema的所有权限从PUBLIC用户组收回。
  - {db}.admin，{db}.{schema}.developer，{db}.{schema}.writer和{db}.{schema}.viewer都具有DB的连接权限。

- <db>_admin将作为DB的Owner以及DB下所有Schema的Owner。
- {db}.{schema}.developer，{db}.{schema}.writer和{db}.{schema}.viewer具有对应Schema的USAGE权限，{db}.{schema}.developer具有该Schema的CREATE权限。
- {db}.admin和{db}.{schema}.developer未来创建的Schema对象，Owner都是对应Schema的{db}.{schema}.developer。{db}.{schema}.writer具有读写（readwrite）权限，{db}.{schema}.viewer具有只读（readonly）权限。
- {db}.admin和{db}.{schema}.developer未来创建的无schema的对象（例如Foreign Server、FDW、Language等），Owner都是对应Schema的{db}.admin。{schema}.developer、{db}.{schema}.writer和{db}.{schema}.viewer具有使用（USAGE）权限。

slpm_migrate

- 函数介绍

  slpm_migrate()函数用于将已有实例对象（包括表、视图及外表等）迁移至SLPM权限模型。

- 命令语法

```
CALL slpm_migrate ( [ batch_size ] );
```

  参数说明如下表所示。

参数	描述	取值范围
batch_size	单次批量迁移对象的大小。 取值为*0*代表采用当前max_locks_per_transaction作为batch_size。	*[1, 64]*，超出该范围则不合法。 如果您的DB中有较多的对象（数百个）需要迁移，需要您多次执行**slpm_migrate()**函数，直到全部对象迁移完成为止。 如果您的DB内有大量表对象（超过数千甚至数万），建议您提交工单将max_locks_per_transaction参数调大，再执行迁移操作。

- 使用说明
  - 调用**slpm_migrate()**函数时，如果出现 `DONE BUT NOT COMPLETED` 提示，说明全部对象还没有迁移完毕，您需要继续执行**slpm_migrate()**迁移。
  - 如果出现 `COMPLETED` 提示，说明全部对象已经迁移完毕。

- 使用示例

```
CALL slpm_migrate (); //迁移最多max_locks_per_transaction个对象至SLPM管理。
CALL slpm_migrate (64); // 迁移64个对象至SLPM管理。
```

slpm_create_user

- 函数介绍

  slpm_create_user()函数用于在SLPM中创建用户。创建的用户仅具有登录权限，您需要授予其具体的权限才能进行开发。

  > **说明** 仅Superuser和{db}.admin的成员可以调用该函数。

- 命令语句

```
CALL slpm_create_user ( user_name [, role_name] );
```

参数说明如下表所示。

参数	描述
user_name	需要创建的用户名。格式如下： ○ 阿里云云账号ID，例如13432193xxxx或者xx@aliyun.com。 ○ RAM用户账号ID，例如RAM$mainaccount:subuser或者p4_202338382183xxx。
role_name	创建用户时可以根据业务需求选择加入如下用户组： ○ {db}.admin ○ {db}.{schema}.developer ○ {db}.{schema}.writer ○ {db}.{schema}.viewer 关于用户组的权限说明，请参见基于Schema级别的简单权限模型介绍。

- 使用示例

```
CALL slpm_create_user ('my_test@aliyun.com');
CALL slpm_create_user ('RAM$my_test:mysubuser', 'mydb.public.developer');
CALL slpm_create_user ('13532313103042xxx');
CALL slpm_create_user ('p4_23319103042xxx', 'mydb.admin');
```

slpm_grant

- 函数介绍

  slpm_grant()函数用于将某个用户加入到{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组。

  > 说明　仅Superuser和{db}.admin可以调用该函数。

- 命令语法

```
CALL slpm_grant ( role_name, user_name );
```

参数说明如下表所示。

参数	描述
user_name	需要授权的用户名。格式如下： ○ 阿里云云账号，例如13432193xxxx或者xx@aliyun.com。 ○ RAM用户账号，例如RAM$mainaccount:subuser或者p4_202338382183xxx。

参数	描述
role_name	可以根据业务需求选择加入如下用户组： ○ {db}.admin ○ {db}.{schema}.developer ○ {db}.{schema}.writer ○ {db}.{schema}.viewer 关于用户组的权限说明，请参见基于Schema级别的简单权限模型介绍。

- 使用说明
  - 只有开启SLPM后才能调用该函数。
  - user_name必须是云账号ID或者云账号。
  - role_name必须是{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组。
- 使用示例

```
CALL slpm_grant ('mydb.public.developer', 'p4_202338382183xxx');//授予RAM用户mydb的developer权限。
CALL slpm_grant ('mydb.admin', 'RAM$my_test:xxx');//授予RAM用户mydb的admin权限。
CALL slpm_grant ('otherdb.admin', '13532313103042xxx'); // 错误，加入其他DB的角色请连接到其他DB执行slpm_grant函数。
```

slpm_revoke

- 函数介绍

  slpm_revoke()函数用于将某个用户从{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组中移除。

  > 说明　仅Superuser和{db}.admin可以调用该函数。

- 命令语法

```
CALL slpm_revoke ( role_name, user_name );
```

- 使用说明
  - 只有开启SLPM后才能调用该函数。
  - user_name必须是云账号ID或者云账号。
  - role_name必须是{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer或{db}.{schema}.viewer用户组。
- 使用示例

```
CALL slpm_revoke ('mydb.public.developer', 'p4_202338382183xxx');//将RAM用户从mydb的developer用户组移除。
CALL slpm_revoke ('mydb.admin', 'RAM$my_test:xxx');//将RAM用户从mydb的admin用户组移除。
CALL slpm_revoke ('otherdb.admin', '13532313103042xxx'); // 错误，移出其他DB的用户组请连接到其他DB执行slpm_grant。
```

slpm_disable

- 函数介绍

  slpm_disable()函数用于为当前DB关闭SLPM。

  > **说明** 仅Superuser可以调用该函数。

- 命令语句

```
CALL slpm_disable ();
```

- 使用说明

  关闭slpm的说明如下：

  - {db}.admin、{db}.{schema}.developer、{db}.{schema}.writer和{db}.{schema}.viewer用户组继续保留，可以使用专家模式授权语句将用户加入对用户组来做权限管理，数据库对象Owner保持不变（仍然是{db}.{schema}.devleoper）。
  - DB的CONNECT，TEMPORARY权限向PUBLIC开放。
  - DB中public schema的USAGE，CREATE权限向PUBLIC开放。
  - DB中public schema中function，procedure的EXECUTE权限向PUBLIC开放。
  - 用户定义的LANGUAGE的USAGE权限向PUBLIC开放。
  - 用户定义的TYPE的USAGE权限向PUBLIC开放。

- 使用示例

```
CALL slpm_disable ();
```

slpm_cleanup

- 函数介绍

  slpm_cleanup()函数用于清除DB的SLPM保留用户组，包括{db}.admin、{db}.{schema}.developer、{db}.{schema}.writer和{db}.{schema}.viewer。

  > **说明** 仅Superuser可以调用此函数。

- 命令语句

```
CALL slpm_cleanup ( db_name [, batch_size ] );
```

  参数说明如下表所示。

参数	描述	取值范围
db_name	需要清除用户组的DB。	如果名称中包含特殊字符或大写字母的db_name用双引号（""）引起来。例如 `"MYDB"` 。

参数	描述	取值范围
batch_size	单次批量迁移对象的大小。 取值为*0*代表默认采用[0,64]max_locks_per_transaction作为当前batch_size。	*[0, 64]*，超出该范围则不合法。 如果您的DB中有大量的对象（数百个）需要迁移，可能会导致锁溢出，需要您多次执行**slpm_migrate()**函数，直到全部对象迁移完成为止。 如果您的DB中有大量的对象（超过数千甚至数万）需要迁移，建议您提交工单将batch_size参数调大，再执行迁移操作。

- 使用说明
  - 调用**slpm_cleanup()**函数时，如果出现 `DONE BUT NOT COMPLETED` 提示，说明全部对象还没有迁移完毕，并且未删除保留用户组，您需要继续执行**slpm_cleanup()**函数。
  - 如果出现 `COMPLETED` 提示，说明全部对象已经迁移完毕，并已删除保留用户组，无需再调用此函数。
- 使用示例

```
CALL slpm_cleanup ('mydb'); //单次最多将max_locks_per_transaction个对象转移Owner到current_user。
CALL slpm_cleanup ('mydb', 64);//单次转移64个对象的Owner到current_user。
```

  - 场景1：删除对应DB，再清除用户组。

```
drop database mydb;
CALL slpm_cleanup ('mydb');  // 无需重试，一次成功。
```

  - 场景2：在DB仍然存在的情况下，需要连接到对应DB进行清除。

```
 CALL slpm_cleanup ('otherdb');
ERROR:  Permission Denied. execute in database otherdb, or drop database before call slpm_cleanup.
```

slpm_rename_schema

- 函数介绍

  slpm_rename_schema()函数用于重命名Schema。

  开启SLPM时，重命名DB需要同步修改4个用户组的命名。因此，需要调用**slpm_rename_schema()**函数来重命名DB。

  > **说明** 仅Superuser和{db}.admin成员可以调用该函数。

- 命令语法

```
CALL slpm_rename_schema ( old_name, new_name );
```

  参数说明如下表所示。

参数	描述

参数	描述
old_name	需要重命名的目标Schema名。包含特殊字符或大写字母的schema_name需要使用双引号（""）引起来。例如 `"MYDB"` 。
new_name	将Schema重命名为new_name。包含特殊字符或大写字母的schema_name需要使用双引号（""）引起来。例如 `"MYDB"` 。

- 使用示例

```
CALL slpm_rename_schema ('oldschema', 'newschema');
```

4.6. 授权操作

4.6.1. 授予RAM用户实例的开发权限

本文为您介绍主账号如何使用简单权限模型和专家权限模型，授予RAM用户实例的开发权限。

背景信息

系统默认设置购买实例的主账号为超级管理员Superuser。Superuser拥有该实例的所有权限，例如创建数据库、删除数据库、创建角色以及为角色授权等。

RAM用户的权限说明如下：

- RAM用户需要被主账号授权后才能访问实例。RAM用户也可以被授权为Superuser。
- 即使RAM用户拥有实例的购买权限，也必须经过主账号授予实例的开发权限后，才能在Hologres实例中进行数据开发。

RAM权限与实例的开发权限的权限控制不同，详情请参见授予RAM用户权限。

使用简单权限模型为RAM用户授权（推荐）

您可以在Hologres管理控制台使用可视化方式为RAM用户授权。具体操作如下：

1. 新建用户。
   i. 使用主账号登录阿里云官网。
   ii. 登录Hologres管理控制台，单击目标实例名称，进入实例详情页。
   iii. 在实例详情页的左侧导航栏，单击**用户管理**。
   iv. 在**用户管理**页面，单击**新增用户**。

v. 配置**新增用户**弹框的各项参数。

参数	描述
选择组织成员	选择需要授权的RAM用户，将其创建至实例中。
用户类型	■ 实例超级管理员（Superuser）：拥有实例内所有操作的权限。 ■ 普通用户：默认无实例的操作权限。 您需要授予RAM用户具体的操作权限后，RAM用户才能连接Hologres实例并使用。

vi. 单击**确定**。

2. （可选）为用户授权。

如果新增的用户是普通用户，则需要执行该步骤。具体操作如下：

i. 在实例详情页的左侧导航栏，单击**DB管理**。

ii. 在**DB管理**页面，单击目标数据库操作列的**用户授权**。

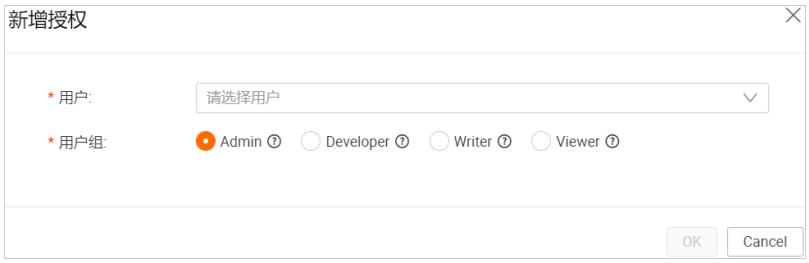
如果**DB管理**页面没有数据库，您需要执行如下操作：

a. 单击**新增Database**。

b. 配置**Database名称**，并选择**简单权限模型**为**开启**。

iii. 进入数据库的权限管理页面，单击**新增授权**。

iv. 配置**新增授权**的各项参数。



参数	描述
用户	需要授权的RAM用户。
用户组	■ Admin：可以访问或操作数据库的所有对象，以及管理数据库的用户组。 ■ Developer：可以使用DDL语句创建、删除或修改数据库的对象，以及读写数据库对象中的数据。 ■ Writer：读写数据库对象中的数据。 ■ Viewer：拥有所有数据库对象的只读权限。

v. 单击**确定**。

简单权限模型也支持使用SQL语句为RAM用户授权，详情请参见简单权限模型的使用。

使用专家权限模型为RAM用户授权

使用专家权限模型为RAM用户授权的操作步骤如下：

1. 新建用户。

   创建RAM用户至Hologres实例。示例语句如下。

   ```
   CREATE USER "p4_账号ID"; //ID为阿里云RAM用户的UID。
   CREATE USER "p4_账号ID" SUPERUSER; //授予RAM用户Superuser权限。
   ```

2. 为用户授权。

   RAM用户需要被授予相应的权限后，才能访问权限范围内的对象。示例授权语句如下。

```
GRANT SELECT ON TABLE TABLENAME TO "账号ID"; //授予RAM用户表的查看权限。
GRANT SELECT,INSERT,UPDATE ON ALL TABLES IN SCHEMA PUBLIC TO "p4_账号ID"; //授予RAM用户所有表的新增、修改和查看权限。
```

> **说明** 只有Superuser和表Owner才可以删除表。

专家权限模型的更多授权操作请参见专家权限模型。

RAM用户使用Hologres

完成授权的RAM用户可以在Psql客户端连接Hologres实例并使用。详情请参见PSQL客户端。

示例语句如下。

```
PGUSER=<AccessID> PGPASSWORD=<AccessKey> psql -p <Port> -h <Endpoint> -d <Database>
```

查看RAM用户的权限

您可以通过如下方式查看RAM用户的权限：

- 使用管理控制台查看RAM用户的权限。
  i. 进入Hologres管理控制台的实例详情页。
  ii. 在实例详情页左侧导航栏，单击**用户管理**。
  iii. 查看RAM用户的权限。

  如果您的数据库已经开启简单权限模型，您可以在**DB管理**页面，单击目标数据库的**用户授权**，查看RAM用户已加入的用户组。
- 使用SQL语句查看RAM用户的权限。

  连接开发工具至Hologres实例后，您可以使用如下SQL语句查看RAM用户的权限。

```
SELECT * FROM pg_roles WHERE rolname = 'p4_ID'; //查看某个成员拥有的角色。
SELECT rolname FROM pg_roles;
SELECT user_display_name(rolname) FROM pg_roles;
```

4.6.2. 权限模型转换

Hologres提供三种权限模型：专家权限模型、简单权限模型（Simple Permission Model，SPM）和基于Schema的简单权限模型（Schema-Level Permission Model，SLPM），可以根据业务情况选择合适的权限模型，但是在实例使用中往往会有权限模型切换的需求。本文为您介绍不同的权限模型之间如何快速简单切换。

查看当前数据库的权限模型

查看当前数据库的权限模型有如下两种方式。

- HoloWeb

  在HoloWeb**安全中心**模块的**DB授权**页面查看当前数据库的权限模型，详情请参见DB管理。
- SQL语句

  可以使用以下命令语句查看数据库是否开启了SPM或者SLPM。

```
--查看是否开启SPM
show hg_experimental_enable_spm;
--查看是否开启SLPM
show hg_enable_slpm;
```

简单权限模型（SPM）切换为专家权限模型

- 注意事项
  - 只能由Superuser执行关闭简单权限模型操作。
  - Public拥有Public Schema的USAGE及CREATE权限。
  - Public拥有DB的CONNECT及TEMPORARY权限。
  - Public拥有functions及procedures的EXECUTE权限。
  - Public拥有 `language, data types (include domains)` 的USAGE权限。
  - Public不拥有其他对象如table、view、materialized view、table column、sequence、foreign data wrapper、foreign server及schema（除public schema）等的权限。
  - 关闭简单权限模型之后，用户组的权限如下。
    - admin：保留对当前已有对象的权限，但对新建数据库对象不生效。
    - developer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。
    - writer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。
    - viewer用户组：保留对当前已有对象的权限，但对新建数据库对象不生效。
- 切换权限模型

  执行以下SQL命令关闭简单权限模型，简单权限模型关闭后就会变成专家权限模型。

```
--关闭简单模型
call spm_disable ();
--清理用户组（可选）
call spm_cleanup ('dbname');
```

> **说明** 关闭简单权限模型之后，对应用户组将不会被删除，通常情况下，为了方便管理不建议删除用户组，如需要清理用户组，请确保该DB上没有正在运行的SQL语句，否则可能会失败，并可能对服务产生影响。

专家权限模型切换为简单权限模型（SPM）

专家权限模型切换成简单权限模型命令语句如下。

```
-- 开启当前DB的简单权限模型
call spm_enable ();
-- 将DB中已有的对象owner更新为developer，使用SPM管理。
call spm_migrate ();
```

如果报错： `ERROR: cannot enable Simple Privilege Model for db=[xxxxxx] because roles conflict` ，说明之前已经开启过SPM，需要执行以下命令语句。

```
--再次开启简单权限模型
call spm_enable ('t');
--将DB中已有的对象owner更新为developer，使用SPM管理。
call spm_migrate ();
```

> **说明** 若是切换成SPM后，新的账号可能会存在即使有权限也会报错没有权限的情况，说明之前的对象迁移不够完整，需要多次执行 `call spm_migrate ();` 命令。

简单权限模型（SPM）切换为Schema级别的权限模型（SLPM）

从SPM切换到SLPM没有直接的方法，需要先从SPM切换成专家权限模型，再从专家权限模型切换为SLPM，切换命令如下。

```
-- 关闭spm
call spm_disable ();
-- 开启Schema级别的权限
call slpm_enable ();
-- 将DB中已有的对象owner更新为developer，使用SLPM管理
call slpm_migrate ();
```

如果报错：`cannot enable slpm for database xxxxx because roles conflict.`，说明之前已经开启过SLPM，需要执行以下命令。

```
-- 关闭spm
call spm_disable ();
-- 开启Schema级别的权限，之前开启过SLPM时执行
call slpm_enable ('t')
-- 将DB中已有的对象owner更新为developer，使用SLPM管理
call slpm_migrate ();
```

> **说明** 若是切换成SLPM后，新的账号可能会存在即使有权限也会报错没有权限的情况，说明之前的对象迁移不够完整，需要多次执行 `call slpm_migrate ();` 命令。

Schema级别的权限模型（SLPM）切换为简单权限模型（SPM）

从SLPM切换到SPM没有直接的方法，需要先从SLPM切换成专家权限模型，再从专家权限模型切换为SPM，切换命令如下。

```
-- 关闭slpm
call slpm_disable ();
-- 开启当前DB的简单权限模型
call spm_enable ();
--将DB中已有的对象owner更新为develoepr，使用SPM管理。
call spm_migrate ();
```

如果报错：`cannot enable spm for database xxxxx because roles conflict.`，说明之前已经开启过SPM，需要执行以下命令。

```
-- 关闭slpm
call slpm_disable ();
-- 开启Schema级别的权限，之前开启过SPM时执行
call spm_enable ('t')
-- 将DB中已有的对象owner更新为developer，使用SPM管理
call spm_migrate ();
```

说明 若是切换成SPM后，新的账号可能会存在即使有权限也会报错没有权限的情况，说明之前的对象迁移不够完整，需要多次执行 `call spm_migrate ();` 命令。

5.数据同步

5.1. 数据同步概述

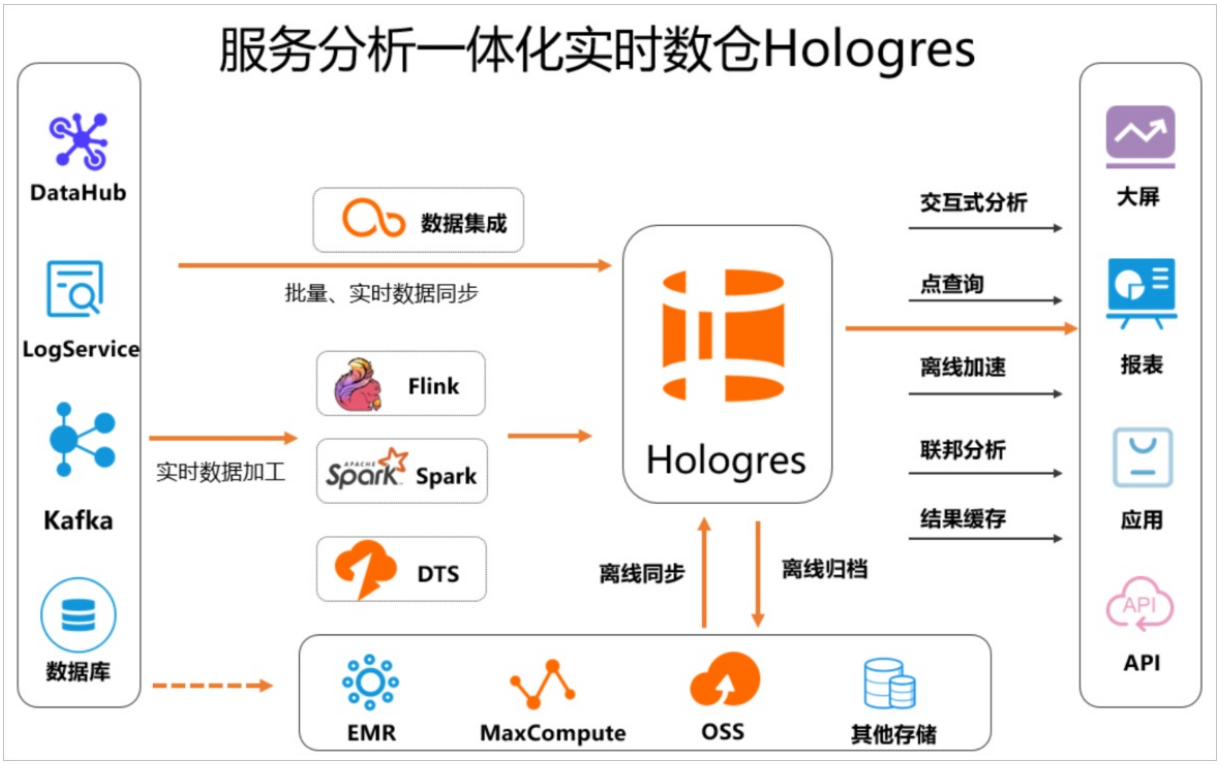
交互式分析Hologres是一款兼容PostgreSQL 11协议的实时数仓，与大数据生态无缝连接，支持高并发地实时写入，数据写入即可查，同时也支持离线数据的加速查询、实时数据和离线数据联邦分析，助力快速搭建企业级实时数仓。

Hologres数据同步说明

Hologres有着非常庞大的生态家族，支持多种异构数据源的离线、实时写入。

- 对于开源大数据：Hologres支持当下最流行的大数据开源组件，其中包括Flink、Blink和Spark等，通过内置的Hologres Connector实现高并发实时写入。
- 对于数据库类数据：Hologres与DataWorks数据集成（DataX和StreamX）深度集成，支持通过Hologres Writer和Hologres Reader，实现方便高效地将多种数据库数据离线、实时、整库同步至Hologres中，满足各类企业数据同步迁移的需求。

无论是实时数据，还是离线数据，同步至Hologres之后就能使用Hologres对数据进行多维分析，例如通过JDBC或者ODBC对数据进行查询、分析、监控，然后直接承接上游的业务例如大屏、报表、应用等可视化展现，实现数据从写入到服务分析一体化。具体使用流程如下所示：



常见同步方案

常见数据源同步数据至Hologres的同步方式支持情况如下表所示，您可以根据业务情况选择合适的同步方式。

常见数据源	Hologres内置同步方案	DataWorks数据集成方式同步数据	Flink方式同步数据
MaxCompute	支持（推荐，SQL命令）	支持	支持
OSS	支持（推荐，SQL命令）	支持	不支持
本地文件	支持（Copy命令）	不支持	不支持
MySQL等数据库	不支持	支持（推荐）	支持
Kafka	不支持	支持	支持
DataHub	支持（Hologres数据源直接写入）	支持	支持

开源Connector支持

Hologres支持丰富的同步Connector如下表所示，并且这些Connector已经开源，请您根据业务情况自行选用。

Connector名称	适用场景
Holo Client	适用于大批量数据写入（批量、实时同步至Hologres）和高QPS点查（维表关联）场景，基于JDBC实现，也提供C语言和GO语言版本。
Holo Shipper	将实例部分表导入导出的备份工具，适用于实例迁移或者数据库数据迁移的场景，也可以dump至中间存储再恢复。
Holo-datax-writer	适配开源DataX，依赖DataX框架，适用开源DataX将多种数据源写入Hologres，相比PostgreSQL Writer性能更好。
Holo-flink-connector	对接开源Flink，Flink版本包括1.11、1.12、1.13以及后续版本，实现高性能实时写入。 **说明** 阿里云Flink支持Hologres数据源，可以直接写入，无需引用connector。
Holo-Kafka-connector	适用于Kafka直接写入Hologres的场景。
Holo-Spark-connector	适用于Spark（社区版以及阿里云EMR Spark版）写入Hologres的场景，支持Spark2.x、3.x及以上版本，提供高性能的写入。
Holo-Hive-connector	适用于Hive写入Hologres的场景，支持Hive2.x、3.x及以上版本，提供高性能的写入。

5.2. 离线同步

5.2.1. MySQL等数据库

5.2.1.1. 数据库中的数据离线同步至Hologres

Hologres支持通过DataWorks平台将数据库中的数据离线同步至Hologres，进行高并发低延时的查询分析处理操作。本文将为您介绍使用DataWorks将各类数据库数据离线同步至Hologres的操作方法。

前提条件

- 开通DataWorks，详情请参见入门概述。
- 开通Hologres实例，并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。
- 开通对应版本数据库。

> **说明** 跨地域是否可以同步数据，详情请参见配置资源组与网络连通。

背景信息

Hologres是一款实时交互式分析产品，与大数据生态无缝打通，与大数据智能研发平台DataWorks深度融合。您可以通过DataWorks数据集成同步将数据库中的数据离线同步至Hologres，再进行高并发低延时的查询分析处理。

常见的支持离线数据同步的数据库包括：RDS for MySQL、Oracle、Polar DB、SQL Server等。

> **说明** 如需查看更多支持的数据库，请参见支持的数据源与读写插件。

相关原理：MySQL Reader、Oracle Reader、PolarDB Reader、SQL Server Reader、Hologres Writer。

操作步骤

1. 配置数据源。

   离线同步前，需先配置输入源数据库和Hologres数据源。

   - 配置输入源数据库：
     - 配置MySQL数据源
     - 配置Oracle数据源
     - 配置PolarDB数据源
     - 配置SQLServer数据源
     - 更多数据源配置请参见配置数据源。
   - 配置Hologres数据源：配置Hologres数据源

2. 配置离线同步任务。

   配置数据源成功之后，可以配置同步任务将数据库中的数据离线同步至Hologres。您可以根据业务情况选择**向导模式**或者**脚本模式**。

   - 通过向导模式配置离线同步任务
   - 通过脚本模式配置离线同步任务

3. 查询数据。

   任务同步成功之后，您可以在Hologres中查询到已同步的数据。

   ```
   SELECT * FROM rds_test;
   ```

5.2.2. MaxCompute

5.2.2.1. 通过创建外部表加速查询MaxCompute数据

本文为您介绍在Hologres中如何通过创建外部表的方式，实现查询，帮助您快速查看MaxCompute的数据。

背景信息

大数据计算服务（MaxCompute，原名ODPS）是一种快速、完全托管的EB级数据仓库，致力于批量结构化数据的存储和计算，提供海量数据仓库的解决方案及分析建模服务。

Hologres是兼容PostgreSQL协议的实时交互式分析引擎，与MaxCompute存储原生对接，支持使用创建外部表的方式实现查询，无冗余存储，无需导入导出数据，即可快速获取查询结果，采用标准PostgreSQL协议，无缝对接几乎所有主流BI工具。

您也可以导入数据至Hologres后，再进行查询。相比其他非大数据生态产品，Hologres导入导出数据的速度性能更佳。

您可以根据业务特性和场景，选择查询方式：

- 在Hologres中直接查询MaxCompute的数据。

  该方式适用于单次查询所需扫描底层数据量小于200 GB，且一次查询命中的分区数少于512个分区的场景。

  > **说明** 数据量小于200 GB指经过分区过滤后，命中分区的数据量大小，与查询字段的大小无关。设定扫描数据量的限制，是为了保障查询的稳定性。外表查询的原理是在运行时将MaxCompute数据的特定查询分区加载到Hologres的内存和缓存中完成计算，如果加载数据过多，会消耗更多网络带宽和计算资源，进而影响查询并发体验。

- 导入MaxCompute的数据至Hologres后再进行查询。

  该方式无扫描数据量限制，支持复杂查询，支持索引，支持UPDATE、INSERT、DELETE等操作。

注意事项

通过创建外部表加速查询MaxCompute数据时，您需要注意如下内容：

- Hologres只能加速查询MaxCompute的内部表，不能查询MaxCompute的外部表和VIEW。
- 通过外部表方式加速查询MaxCompute数据，一次Query命中的数据量大小不超过200 GB，一次Query命中的分区数不超过512个。通过导入数据至Hologres内部表的方式则没有此限制。
- MaxCompute的表数据更新之后，在Hologres存在缓存（一般为5分钟内）才能加速更新后的数据，如果您需要实时查询更新后的数据，可以使用IMPORT FOREIGN SCHEMA语法更新外部表元数据，就能实时查询更新后的数据。如果是数据导入场景，在V1.1.25+版本，无需手动重新刷新外部表元数据，导入语句会自动获取最新元数据。
- MaxCompute的Schema更新之后，Hologres不会自动更新，需要手动更新。
- MaxCompute的分区与Hologres无强映射关系，映射至Hologres之后均为普通字段。
- MaxCompute与Hologres数据类型一一映射，建表时您可以查看映射关系，详情请参见数据类型汇总。
- 可以跨区域查询MaxCompute的数据，但是国内区域不能加速查询中国以外区域MaxCompute的数据，中国以外区域间不能相互加速查询。不建议使用跨区域查询加速，由于跨区域存在较多网络不可靠因素，查询稳定性无法保证，请保持Hologres和MaxCompute处于同一区域。
- 外部表不存储数据，数据存储在MaxCompute中。

- 暂不支持MaxCompute Transaction Table；暂不支持处于Schema Evolution状态的表。
- 当使用Streaming Tunnel写入MaxCompute表时，表会首先处于streaming状态，异步进行compaction到orc格式，Hologres 1.1.45+版本可以读取streaming状态的表，早期版本暂不支持读取未完成compaction状态的表。

查询MaxCompute非分区表数据

1. 准备MaxCompute非分区表数据。

   创建MaxCompute非分区表并导入数据，详情请参见创建表。您也可以选择已创建的MaxCompute非分区表。

   本实验选用已创建的MaxCompute表，示例SQL语句如下。

```
CREATE TABLE weather (
    city            STRING ,
    temp_lo         int,           --最低温度
    temp_hi         int           --最高温度
);
INSERT INTO weather VALUES
('beijing',40,50),
('hangzhou',46,55);
```

2. Hologres创建外部表。

   在Hologres中创建一张用于映射MaxCompute数据的外部表。您可以选择查询部分字段或全部字段。示例语句如下。

```
CREATE FOREIGN TABLE weather1 (
 city text,
 temp_lo int8,
 temp_hi int8
)
SERVER odps_server
OPTIONS (project_name '<projectname>',table_name 'weather');
```

   参数说明如下表所示。

参数	描述
SERVER	外部表服务器。 您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
project_name	MaxCompute表所在的项目名称。
table_name	需要查询的MaxCompute表名称。

> **说明** ：
> - Hologres的字段类型必须与MaxCompute的字段类型保持一致，数据类型的映射关系请参见MaxCompute与Hologres的数据类型映射。
> - Hologres支持使用 `IMPORT FOREIGN SCHEMA` 语句批量创建外部表，详情请参见IMPORT FOREIGN SCHEMA。您也可以在**数据开发**中执行该语句，并配置调度，实现更新MaxCompute表时，Hologres外部表也同步更新，详情请参见Hologres开发：周期性调度。
> - Hologres仅支持加速查询MaxCompute的内部表数据，不支持加速查询MaxCompute的外部表和视图。

3. 查询外部表数据。

   成功创建外部表后，您可以直接查询外部表，即可查询到MaxCompute的数据。示例语句如下。

```
SELECT * FROM weather1;
```

查询MaxCompute分区表数据

1. 准备MaxCompute分区表数据。

   创建一张MaxCompute分区表并导入数据，详情请参见分区和列操作。您也可以选择已创建的MaxCompute分区表。

   本实验选用数据地图已创建的分区表，示例语句如下。

```
create table odps_test
(
    shop_name     string,
    customer_id   string,
    total_price   INT
)
partitioned by (sale_date string);
```

2. Hologres创建外部表。

   在Hologres中创建一张用于映射MaxCompute源数据表的外部表。示例语句如下。

```
CREATE FOREIGN TABLE table_odps (
 shop_name text,
 customer_id text,
 total_price int8,
 sale_date text
)
SERVER odps_server
OPTIONS (project_name '<projectname>', table_name 'odps_test');
```

> **说明** Hologres的外部表仅用于字段映射，不存储数据。MaxCompute中的分区字段映射为Hologres的普通字段。

3. 查询分区表数据。

   查询整张表数据，示例SQL语句如下。

```
SELECT * FROM table_odps;
```

查询分区表数据，示例SQL语句如下。

```
SELECT * FROM table_odps
WHERE sale_date = '2013';
```

批量创建外部表

如果您需要加速查询大批量的MaxCompute表，可以通过批量创建外部表的方式来实现。在Hologres您可以使用SQL语句或者管理控制台可视化的方式批量创建外部表。

- Hologres支持使用 `IMPORT FOREIGN SCHEMA` 语句批量创建外部表，详情请参见IMPORT FOREIGN SCHEMA。
- 通过HoloWeb批量创建外部表，详情请参见批量创建外部表。
- 通过HoloStudio批量创建外部表，详情请参见一键同步MaxCompute表结构。

HoloWeb可视化创建外部表

HoloWeb提供可视化一键创建外部表功能，无需写SQL命令就能创建外部表和查看数据，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**元数据管理 > MaxCompute加速**，单击**创建外部表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**新建外部表**。

3. 在**新建外部表**页面，配置各项参数。单击**提交表**。

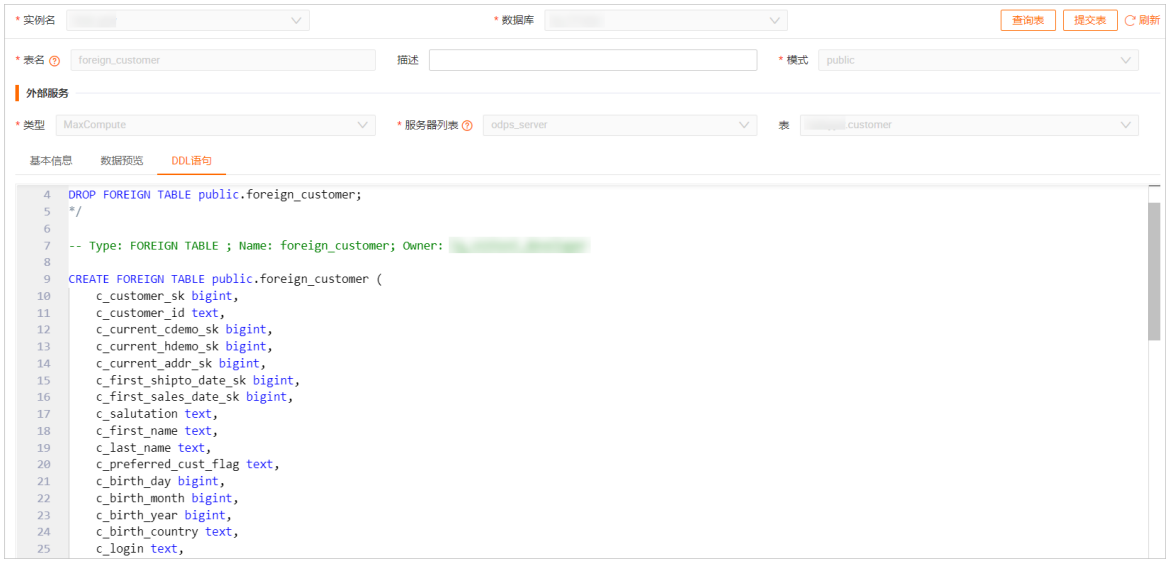
参数	描述
实例名	已登录的实例名称。
数据库	Hologres当前已登录实例的数据库名称。
表名	新建的Hologres外部表名称。 输入目标MaxCompute表名后，将会自动创建同名外部表。在创建时不支持更改表名，如果您需要更改表名，可以在外部表创建成功后，在已登录实例列表中右键单击目标表进行修改。
描述	新建的Hologres外部表描述。
模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。
类型	外部表类型。 目前仅支持MaxCompute。
服务器列表	您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
表	MaxCompute的项目名和表名。 格式为**project.table_name**。 **说明** ○ 目前暂不支持跨地域查询外部表数据。 ○ 输入表名称后，会显示外部源表的所有字段，创建外部表时也将会默认创建所有字段。如果您需要创建部分字段，请使用SQL语句创建外部表，请参见CREATE FOREIGN TABLE。

> **说明** 创建外部表同步MaxCompute表的数据时，会将数据库中表字段的Comment和列的Comment一并同步至Hologres。

4. 单击**提交表**，完成外部表的创建。提交之后，您可以在左侧对应模式下，刷新出新建的外部表。
5. （可选）表数据预览
   i. 在**已登录实例**列表，双击目标表。
   ii. 进入表信息页签，单击**数据预览**，则可以预览表数据。

6. （可选）DDL预览

   在目标表信息页签，单击**DDL语句**，则可以预览DDL语句。



外部表查询性能优化

当外部表查询性能不满足当前查询时，您可以通过MaxCompute合并小文件，优化Hologres SQL等标准手段进行优化，以提升查询性能。从Hologres V0.10版本开始，Hologres采用全新外部表加速引擎，相比低于V0.10版本实例，查询MaxCompute表性能提升30%~100%左右。详情请参见优化MaxCompute外部表的查询性能。

常见问题

通过外部表查询MaxCompute数据的相关常见问题请参见对接MaxCompute常见问题与诊断。

5.2.2.2. 使用SQL导入MaxCompute的数据至Hologres

本文为您介绍如何使用SQL语句导入MaxCompute的数据至Hologres。

背景信息

外部表不存储数据，无索引能力，针对MaxCompute的业务数据大于200 GB，查询复杂度高且要求查询响应达到秒级的场景，Hologres支持直接导入数据并查询，查询数据速度比通过外部表方式更快。

注意事项

通过SQL导入MaxCompute数据至Hologres时，您需要注意如下内容：

- MaxCompute的分区与Hologres无强映射关系，MaxCompute的分区字段映射至Hologres为普通字段，因此支持MaxCompute分区数据导入Hologres非分区或者分区。
- Hologres仅支持一级分区，MaxCompute多级分区导入Hologres分区表时，只需要映射一个分区，其余分区映射成Hologres的普通字段。
- 如果导入数据时需要更新覆盖原有数据，您需要使用INSERT ON CONFLICT(UPSERT)语法。
- MaxCompute与Hologres的数据类型映射，请参见数据类型汇总。
- MaxCompute的表数据更新之后，在Hologres存在缓存延迟（一般为10分钟内），建议在导入数据前使用IMPORT FOREIGN SCHEMA语法更新外部表以获取最新数据。
- 导入MaxCompute数据至Hologres时，建议使用SQL导入，不建议使用数据集成导入，因为使用SQL导入性能表现更优。

导入MaxCompute的非分区表数据至Hologres并查询

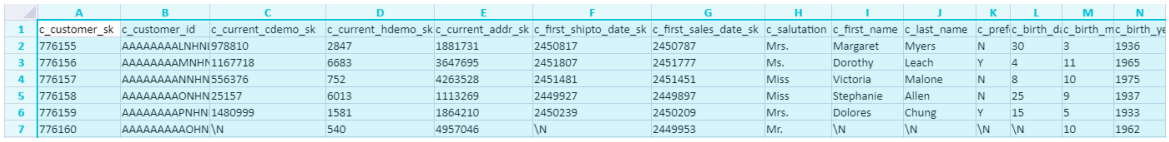
1. 准备MaxCompute的非分区表数据。

   在MaxCompute中创建一张非分区源数据表，或直接选用已创建的非分区表。

   示例选取MaxCompute公共数据集**public_data**的customer表，您可以参照使用公开数据集描述，登录并查询数据集。其表DDL以及数据如下。

   ```
   --MaxCompute公共数据集的表DDL
   CREATE TABLE IF NOT EXISTS public_data.customer(
     c_customer_sk BIGINT,
     c_customer_id STRING,
     c_current_cdemo_sk BIGINT,
     c_current_hdemo_sk BIGINT,
     c_current_addr_sk BIGINT,
     c_first_shipto_date_sk BIGINT,
     c_first_sales_date_sk BIGINT,
     c_salutation STRING,
     c_first_name STRING,
     c_last_name STRING,
     c_preferred_cust_flag STRING,
     c_birth_day BIGINT,
     c_birth_month BIGINT,
     c_birth_year BIGINT,
     c_birth_country STRING,
     c_login STRING,
     c_email_address STRING,
     c_last_review_date STRING,
     useless STRING);
   --在MaxCompute中查询表是否有数据
   SELECT * FROM public_data.customer;
   ```

   部分数据内容显示如下图所示。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	c_customer_sk	c_customer_id	c_current_cdemo_sk	c_current_hdemo_sk	c_current_addr_sk	c_first_shipto_date_sk	c_first_sales_date_sk	c_salutation	c_first_name	c_last_name	c_prefi	c_birth_d	c_birth_m	c_birth_ye
2	776155	AAAAAAAALNHN	978810	2847	1881731	2450817	2450787	Mrs.	Margaret	Myers	N	30	3	1936
3	776156	AAAAAAAAMNHN	1167718	6683	3647695	2451807	2451777	Ms.	Dorothy	Leach	Y	4	11	1965
4	776157	AAAAAAAANNHN	556376	752	4263528	2451481	2451451	Miss	Victoria	Malone	N	8	10	1975
5	776158	AAAAAAAAONHN	25157	6013	1113269	2449927	2449897	Miss	Stephanie	Allen	N	25	9	1937
6	776159	AAAAAAAAPNHN	1480999	1581	1864210	2450239	2450209	Mrs.	Dolores	Chung	Y	15	5	1933
7	776160	AAAAAAAAAOHN	\N	540	4957046	\N	2449953	Mr.	\N	\N	\N	\N	10	1962

2. 在Hologres中创建外部表。

   在Hologres中创建一张用于映射MaxCompute源数据表的外部表。示例SQL语句如下。

```
CREATE FOREIGN TABLE foreign_customer (
    "c_customer_sk" int8,
    "c_customer_id" text,
    "c_current_cdemo_sk" int8,
    "c_current_hdemo_sk" int8,
    "c_current_addr_sk" int8,
    "c_first_shipto_date_sk" int8,
    "c_first_sales_date_sk" int8,
    "c_salutation" text,
    "c_first_name" text,
    "c_last_name" text,
    "c_preferred_cust_flag" text,
    "c_birth_day" int8,
    "c_birth_month" int8,
    "c_birth_year" int8,
    "c_birth_country" text,
    "c_login" text,
    "c_email_address" text,
    "c_last_review_date" text,
    "useless" text
)
SERVER odps_server
OPTIONS (project_name 'public_data', table_name 'customer');
```

参数	描述
Server	您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
Project_Name	MaxCompute表所在的项目名称。
Table_Name	需要查询的MaxCompute表名称。

   外部表字段的数据类型与MaxCompute表字段的数据类型保持一致，数据类型的映射关系请参见MaxCompute与Hologres的数据类型映射。

3. 在Hologres中创建存储表。

   在Hologres中创建一张用于接收MaxCompute源表数据的存储表。

   本示例是初步的DDL示例，实际导入数据时创建表请根据业务情况设置表结构并给表设置合适的索引，以达到更优的查询性能，更多关于表属性的描述，请参见CREATE TABLE。

```
--示例建一张列存表
BEGIN;
CREATE TABLE public.holo_customer (
 "c_customer_sk" int8,
 "c_customer_id" text,
 "c_current_cdemo_sk" int8,
 "c_current_hdemo_sk" int8,
 "c_current_addr_sk" int8,
 "c_first_shipto_date_sk" int8,
 "c_first_sales_date_sk" int8,
 "c_salutation" text,
 "c_first_name" text,
 "c_last_name" text,
 "c_preferred_cust_flag" text,
 "c_birth_day" int8,
 "c_birth_month" int8,
 "c_birth_year" int8,
 "c_birth_country" text,
 "c_login" text,
 "c_email_address" text,
 "c_last_review_date" text,
 "useless" text
);
CALL SET_TABLE_PROPERTY('public.holo_customer', 'orientation', 'column');
CALL SET_TABLE_PROPERTY('public.holo_customer', 'bitmap_columns', 'c_customer_id,c_salu
tation,c_first_name,c_last_name,c_preferred_cust_flag,c_birth_country,c_login,c_email_a
ddress,c_last_review_date,useless');
CALL SET_TABLE_PROPERTY('public.holo_customer', 'dictionary_encoding_columns', 'c_custo
mer_id:auto,c_salutation:auto,c_first_name:auto,c_last_name:auto,c_preferred_cust_flag:
auto,c_birth_country:auto,c_login:auto,c_email_address:auto,c_last_review_date:auto,use
less:auto');
CALL SET_TABLE_PROPERTY('public.holo_customer', 'time_to_live_in_seconds', '3153600000'
);
CALL SET_TABLE_PROPERTY('public.holo_customer', 'storage_format', 'segment');
COMMIT;
```

4. 导入数据至Hologres。

   使用 `INSERT` 语句将MaxCompute源头表中的数据导入至Hologres，您可以选择部分字段导入（字段顺序需要一一对应），也可以选择全部字段导入，示例DDL如下。

```
--部分字段数据导入
INSERT INTO holo_customer (c_customer_sk,c_customer_id,c_email_address,c_last_review_da
te,useless)
SELECT
    c_customer_sk,
    c_customer_id,
    c_email_address,
    c_last_review_date,
    useless
FROM foreign_customer;
--全部字段数据导入
INSERT INTO holo_customer
SELECT * FROM foreign_customer;
```

5. Hologres查询MaxCompute表数据。

   在Hologres中查询导入的MaxCompute表数据。示例SQL语句如下。

```
SELECT * FROM holo_customer;
```

导入MaxCompute的分区表数据至Hologres并查询

详情请参见MaxCompute分区表数据导入。

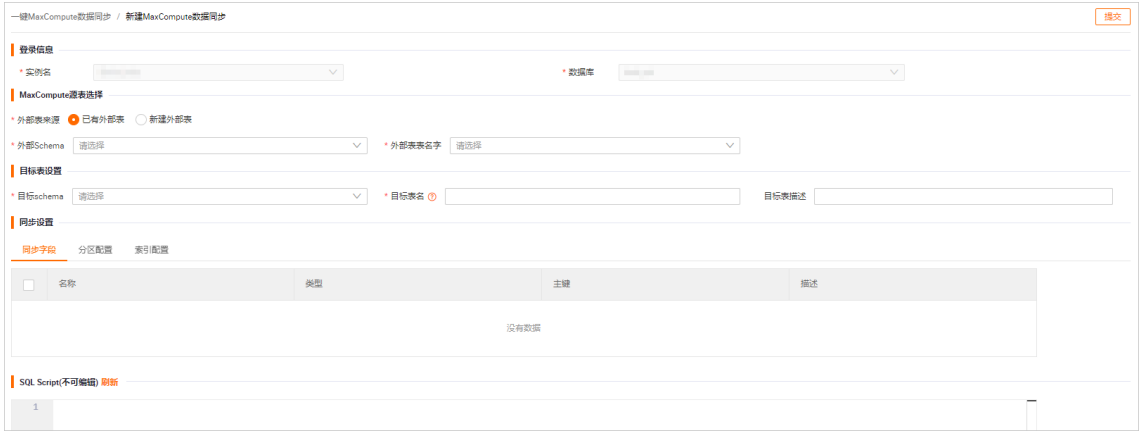
INSERT OVERWRITE最佳实践

详情请参见INSERT OVERWRITE。

通过可视化工具或周期性调度同步数据

如果您需要一次性同步大数据量，可以通过可视化工具或者调度任务来实现。

- 通过可视化工具HoloWeb一键同步MaxCompute数据操作步骤如下。
  i. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
  ii. 在HoloWeb开发页面的顶部菜单栏，选择**元数据管理 > MaxCompute加速**，单击**一键MaxCompute数据同步**。
  iii. 配置**新建MaxCompute数据同步**页面的各项参数。



参数说明如下表所示。

> **说明** SQL Script自动解析当前可视化操作对应的SQL语句。SQL Script内的SQL语句不支持修改，如果需要修改，请将SQL复制，手动更改后，使用SQL同步。

类别	参数	描述
选择连接	实例名	已登录的实例名称。
	数据库	Hologres已登录实例的数据库名称。
	外部表来源	■ **已有外部表**：表示在Hologres中已经建立映射MaxCompute数据的外部表。 ■ **新建外部表**：表示在Hologres中未建立MaxCompute数据映射的外部表。

类别	参数	描述
MaxCompute源表选择	外部Schema	Hologres中已创建的MaxCompute外部表所在的Schema。 当**外部表来源**选择**已有外部表**时，需要配置该参数。
	外部表表名字	Hologres中已创建的MaxCompute外部表的名称。 当**外部表来源**选择**已有外部表**时，需要配置该参数。
	服务器列表	Hologres存放新建外部表的服务器。您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。 当**外部表来源**选择**新建外部表**时，需要配置该参数。
	表名	Hologres新建的外部表的项目名和表名。 格式为**project.table_name**。 当**外部表来源**选择**新建外部表**时，需要配置该参数。
目标表设置	目标Schema	当前表所在的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public** Schema。如果有新建的Schema，您也可以选择新建的Schema。
	目标表名	接收MaxCompute表数据的Hologres内部表名称，在相同Schema下不能与外部表同名。
	目标表描述	目标表的信息描述。
同步设置	同步字段	需要导入的MaxCompute表字段。 只能选择导入全部字段，不支持导入部分字段。
分区配置	分区字段	选择分区字段，Hologres将会默认将表创建为分区表。 Hologres仅支持一级分区。如果您需要导入MaxCompute的多级分区，则在Hologres中设置一级分区即可，其余分区自动映射为Hologres的普通字段。

类别	参数	描述
	业务日期	如果MaxCompute表使用日期进行分区，则您可以选择具体的分区日期，系统将会导入指定日期的数据至MaxCompute表。
索引配置	存储模式	■ **列存**，适用于各种复杂查询。 ■ **行存**，适用于基于主键的点查询和Scan。 如果不指定存储模式，则默认为**列存**。
	生命周期（秒）	表数据的生命周期。默认为**永久**存储。 指定生命周期后，如果数据在指定时间内未被修改，则引擎将会在到期后的某一个时间段删除数据。
	聚簇索引	排序索引Clustering_key。 索引的类型和列的顺序密切相关。聚簇索引帮助您加速执行索引列的Range和Filter查询。
	分段键	您可以指定部分列作为分段键Segment_key。当查询条件包含分段列时，您可以通过分段键快速查找相应数据的存储位置。
	字典编码列	Hologres支持为指定列的值构建字典映射。 字典编码可以将字符串的比较转换为数字的比较，加速Group By和Filter查询。 默认设置所有**text**列至字典编码列中。
	位图列	Hologres支持在位图列构建比特编码。 位图列可以根据设置的条件快速过滤字段内部的数据。 默认设置所有**text**列至位图列中。
	分布列索引	Hologres会按照分布列指定的列将数据shuffle到各个Shard，同样的数值会在同样的Shard中。以分布列做过滤条件时，可以大大提高执行效率。

iv. 单击**提交**，数据导入完成后，可以在Hologres中查询内部表数据。

- HoloWeb一键同步不支持周期性调度，若是需要同步大量历史数据，或者周期性调度导入，需要通过DataWorks的DataStudio，详情请参见通过DataWorks周期性导入MaxCompute数据最佳实践。

常见问题

导入MaxCompute的数据至Hologres时发生OOM（Out Of Memory，内存溢出），提示超出内存限制异常。通常会产生`Query executor exceeded total memory limitation xxxxx: yyyy bytes used`报错。导致报错的四种可能原因及其对应的解决方案如下所示。

- 排查步骤一

- 可能原因：

  当导入query包含查询，但部分table没有 `analyze` ；或者进行过 `analyze` ，但数据又有更新导致不准确，最终导致查询优化器决策join order有误，引起内存开销过高。

- 解决方案：

  对所有参与的内表、外表执行 `analyze` 命令，更新表的统计元信息，可以帮助查询优化器生成更优的执行计划，解决出现OMM的情况。

- 排查步骤二
  - 可能原因：

    当表的列数较多，单行数据量较大时，单次读取的数据量会更大，引起内存开销过高。

  - 解决方案：

    在sql语句前加以下参数来控制单次读取数据行数，可以有效减少OOM情况。

    ```
    set hg_experimental_query_batch_size = 1024;--默认为8192
    insert into holo_table select * from mc_table;
    ```

- 排查步骤三
  - 可能原因：

    导入数据的过程中，并发度高，CPU消耗大，影响内表查询。

  - 解决方案：

    在V1.1之前版本，可以通过并发度通过参数hg_experimental_foreign_table_executor_max_dop控制，默认为实例的Core数，在导入时设置更小的hg_experimental_foreign_table_executor_max_dop参数，降低导入的内存使用，解决出现OMM的情况。该参数对外表执行的所有作业有效。代码示例如下所示。

    ```
    set hg_experimental_foreign_table_executor_max_dop = 8;
    insert into holo_table select * from mc_table;
    ```

- 排查步骤四
  - 可能原因：

    导入数据的过程中，并发度高，CPU消耗大，影响内表查询。

  - 解决方案：

    从V1.1版本开始，可以通过并发度通过参数hg_experimental_foreign_table_executor_dml_max_dop控制，默认为32，在导入时设置更小的hg_experimental_foreign_table_executor_dml_max_dop参数，降低执行DML语句的并发度（主要是数据导入导出场景），避免执行DML语句占用过多资源。代码示例如下所示。

    ```
    set hg_experimental_foreign_table_executor_dml_max_dop = 8;
    insert into holo_table select * from mc_table;
    ```

5.2.2.3. MaxCompute分区表数据导入

本文为您介绍如何将MaxCompute分区表数据导入到Hologres分区表。

前提条件

- 已购买并开通Hologres实例，开通方法请参见购买Hologres。
- 已开通MaxCompute并创建项目，详情请参见开通MaxCompute和DataWorks。
- 已开通DataWorks服务并创建DataWorks工作空间，详情请参见创建工作空间。

背景信息

通过Hologres中的MaxCompute外表方式向Hologres导入数据是非常常见的数据导入模式。在日常工作中会经常需要进行数据导入，此时可以借助DataWorks的强大调度和作业编排能力，实现周期性调度，配置一个调度作业覆盖数据导入两个场景，详情请参见DataWorks作业案例。

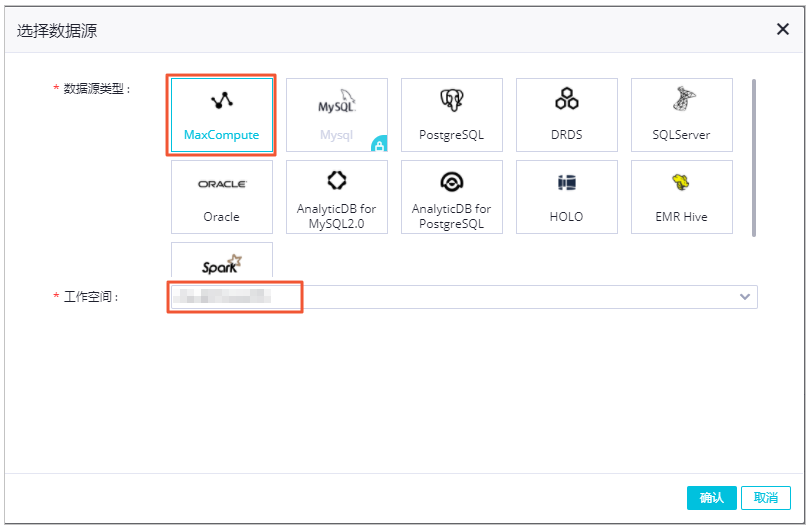
考虑到作业较为复杂，所以可以利用DataWorks的迁移助手功能，将Data作业案例文件导入您的项目中，您即可获得Data作业案例，之后按照您的具体业务需求更改部分参数或脚本即可，详情请参见使用迁移工具导入DataWorks作业。

注意事项

- 使用临时表的原因是为了保证原子性，只有在导入完成后才绑定至分区表，为了避免导入任务失败时还需要重新删除表等操作。
- 对于更新子表分区数据场景，需要删除子表和重新绑定临时表放入一个事务过程中，保证该过程的事务性。
- 使用迁移工具导入DataWorks作业时需满足以下条件：
  - DataWorks需标准版及以上版本，详情请参见DataWorks各版本详解。
  - DataWorks工作空间需绑定MaxCompute和Hologres计算引擎服务，详情请参见配置工作空间。

详细操作步骤如下。

1. MaxCompute数据准备
   i. 登录MaxCompute控制台，在左上角选择区域，单击**查询编辑**，即可进入查询编辑器界面。
   ii. 在**选择数据源**对话框，选择**数据源类型**为**MaxCompute**，**工作空间**为已创建好的项目空间。



> **说明** 如果您选择的**工作空间**模式为标准模式，在查询编辑器中提交作业实际是在开发项目（带dev标识）中提交。

   iii. 单击**确认**，即可进入查询编辑器界面。

iv. 在**SQL查询页面**，输入如下SQL语句用于创建分区表，单击**运行**。

```
DROP TABLE IF EXISTS odps_sale_detail;
--创建一张分区表sale_detail。
CREATE TABLE IF NOT EXISTS odps_sale_detail
(
    shop_name STRING
    ,customer_id STRING
    ,total_price DOUBLE
)
PARTITIONED BY
(
    sale_date STRING
)
;
```

v. 在**SQL查询页面**，输入如下SQL语句用于向分区表中导入数据，单击**运行**。

```
-- 向源表增加分区20210815
ALTER TABLE odps_sale_detail ADD IF NOT EXISTS PARTITION(sale_date='20210815')
;
-- 向分区写入数据
INSERT OVERWRITE TABLE odps_sale_detail PARTITION(sale_date='20210815') VALUES
('s1','c1',100.1),
('s2','c2',100.2),
('s3','c3',100.3)
;
-- 向源表增加分区20210816
ALTER TABLE odps_sale_detail ADD IF NOT EXISTS PARTITION(sale_date='20210816')
;
-- 向分区写入数据
INSERT OVERWRITE TABLE odps_sale_detail PARTITION(sale_date='20210816') VALUES
('s1','c1',100.1),
('s2','c2',100.2),
('s3','c3',100.3)
;
-- 向源表增加分区20210817
ALTER TABLE odps_sale_detail ADD IF NOT EXISTS PARTITION(sale_date='20210817')
;
-- 向分区写入数据
INSERT OVERWRITE TABLE odps_sale_detail PARTITION(sale_date='20210817') VALUES
('s1','c1',100.1),
('s2','c2',100.2),
('s3','c3',100.3)
;
-- 向源表增加分区20210818
ALTER TABLE odps_sale_detail ADD IF NOT EXISTS PARTITION(sale_date='20210818')
;
-- 向分区写入数据
INSERT OVERWRITE TABLE odps_sale_detail PARTITION(sale_date='20210818') VALUES
('s1','c1',100.1),
('s2','c2',100.2),
('s3','c3',100.3)
;
```

2. Hologres中建表
   - 创建外部表
     a. 登录数据库
        a. 在**HoloWeb控制台DB授权**页面，单击**元数据管理**。
        b. 在**元数据管理**页面，双击左侧目录树中已创建成功的数据库名称，单击**确认**。

     b. 创建外部表
        a. 在**SQL编辑器**页面，单击左上角的**新建SQL窗口**。
        b. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

```
DROP FOREIGN TABLE IF EXISTS odps_sale_detail;
-- 创建外部表
IMPORT FOREIGN SCHEMA maxcompute_project LIMIT to
(
    odps_sale_detail
)
FROM SERVER odps_server INTO public
OPTIONS(if_table_exist 'error',if_unsupported_type 'error');
```

   - 创建分区表（内部表）

a. 登录数据库

a. 在**HoloWeb控制台DB授权**页面，单击**元数据管理**。

b. 在**元数据管理**页面，双击左侧目录树中已创建成功的数据库名称，单击**确认**。

b. 创建分区表

a. 在**SQL编辑器**页面，单击左上角的**新建SQL窗口**。

b. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

```
DROP TABLE IF EXISTS holo_sale_detail;
-- 创建Hologres分区表（内部表）
BEGIN ;
CREATE TABLE IF NOT EXISTS holo_sale_detail
(
    shop_name TEXT
    ,customer_id TEXT
    ,total_price FLOAT8
    ,sale_date TEXT
)
PARTITION BY LIST(sale_date);
COMMIT;
```

3. 分区数据导入Hologres临时表

在**临时Query查询**页面，请您在SQL查询的编辑框输入如下语句，单击**运行**。

此SQL语句将MaxCompute的hologres_test项目中的odps_sale_detail分区表的20210816分区导入Hologres中的holo_sale_detail分区表的20210816分区。

```
-- 刷新外表的Schema
IMPORT FOREIGN SCHEMA hologres_test LIMIT to
(
    odps_sale_detail
)
FROM SERVER odps_server INTO public
OPTIONS(if_table_exist 'update',if_unsupported_type 'error');
-- 清理潜在的临时表
BEGIN ;
DROP TABLE IF EXISTS holo_sale_detail_tmp_20210816;
COMMIT ;
-- 创建临时表
SET hg_experimental_enable_create_table_like_properties=on;
BEGIN ;
CALL HG_CREATE_TABLE_LIKE ('holo_sale_detail_tmp_20210816', 'select * from holo_sale_de
tail');
COMMIT;
-- 向临时表插入数据
INSERT INTO holo_sale_detail_tmp_20210816
SELECT *
FROM public.odps_sale_detail
WHERE sale_date='20210816';
```

4. 临时表绑定至Hologres分区表

   在**临时Query查询**页面，请您在SQL查询的编辑框输入如下语句，单击**运行**。

   - 存在旧的子表，则需要先删除旧子表，再将临时表绑定至Hologres分区表。

     此SQL语句用于删除子表holo_sale_detail_20210816并将临时表holo_sale_detail_tmp_20210816绑定至holo_sale_detail分区表的20210816分区。

     ```
     -- 已有子表时替换子表
     BEGIN ;
     -- 删除旧子表
     DROP TABLE IF EXISTS holo_sale_detail_20210816;
     -- 将临时表改名
     ALTER TABLE holo_sale_detail_tmp_20210816 RENAME TO holo_sale_detail_20210816;
     -- 将临时表绑定至指定分区表
     ALTER TABLE holo_sale_detail ATTACH PARTITION holo_sale_detail_20210816
     FOR VALUES IN ('20210816')
     ;
     COMMIT ;
     ```

   - 不存在旧子表，直接将临时表绑定至Hologres分区表。

     此SQL语句用于将临时表holo_sale_detail_tmp_20210816绑定至holo_sale_detail分区表的20210816分区。

```
BEGIN ;
-- 将临时表改名
ALTER TABLE holo_sale_detail_tmp_20210816 RENAME TO holo_sale_detail_20210816;
-- 将临时表绑定至指定分区表
ALTER TABLE holo_sale_detail ATTACH PARTITION holo_sale_detail_20210816
FOR VALUES IN ('20210816');
COMMIT ;
```

5. ANALYZE Hologres分区表

   在**临时Query查询**页面，请您在SQL查询的编辑框输入如下语句，单击**运行**。

   此SQL语句用于ANALYZE holo_sale_detail分区表，验证分区表执行计划。ANALYZE分区表时，仅需ANALYZE父表。

```
-- 大量数据导入后执行ANALYZE分区表父表操作
ANALYZE holo_sale_detail;
```

6. 清理过期的分区子表（按需）

   生产环境中，数据具备生命周期，对于超期的分区需要清理。

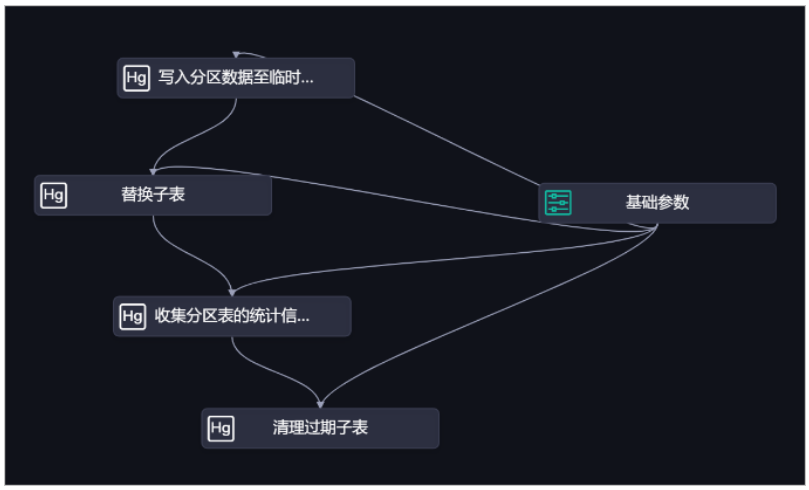
   在**临时Query查询**页面，请您在SQL查询的编辑框输入如下语句，单击**运行**。

   此SQL语句清理20210631的分区。

```
DROP TABLE IF EXISTS holo_sale_detail_20210631;
```

DataWorks作业案例

日常工作中往往需要周期性的调度以上的SQL，此时可以借助DataWorks的强大调度和作业编排能力，实现周期性调度，且使用一个调度作业覆盖以上两个场景。请仔细阅读以下内容，便于您使用迁移工具导入DataWorks作业时按照您的具体业务需求更改部分参数或脚本。业务流程总览如下。



业务流程模块详解

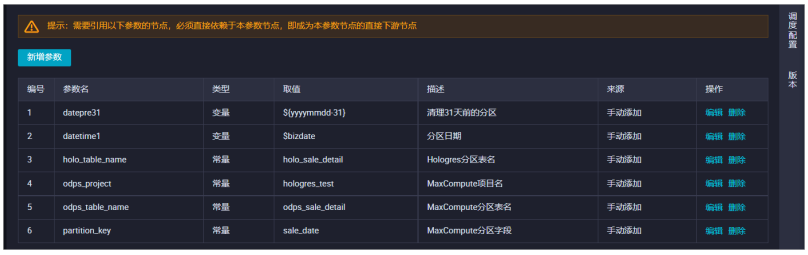
- 基础参数

  基础参数用于管理整个业务流程中用到的所有参数，主要用到的参数如下。

编号	参数名	类型	取值	描述

编号	参数名	类型	取值	描述
1	datepre31	变量	${yyyymmdd-31}	用于控制清理过期分区的参数，此处含义为清理31天前的分区。
2	datetime1	变量	$bizdate	用于控制创建分区的参数。
3	holo_table_name	常量	holo_sale_detail	Hologres分区表名。
4	odps_project	常量	hologres_test	MaxCompute项目名。
5	odps_table_name	常量	odps_sale_detail	MaxCompute分区表名。
6	partition_key	常量	sale_date	MaxCompute分区字段。

系统配置图如下。



- 写入分区数据至临时表

  该步骤是一个Hologres SQL模块，其中SQL代码如下。

```
-- 刷新外表的Schema
IMPORT FOREIGN SCHEMA ${odps_project} LIMIT to
(
    ${odps_table_name}
)
FROM SERVER odps_server INTO public
OPTIONS(if_table_exist 'update',if_unsupported_type 'error');
-- 清理潜在的临时表
BEGIN ;
DROP TABLE IF EXISTS ${holo_table_name}_tmp_${datetime1};
COMMIT ;
-- 创建临时表
SET hg_experimental_enable_create_table_like_properties=on;
BEGIN ;
CALL HG_CREATE_TABLE_LIKE ('${holo_table_name}_tmp_${datetime1}', 'select * from ${holo_table_name}');
COMMIT;
-- 向临时表插入数据
INSERT INTO ${holo_table_name}_tmp_${datetime1}
SELECT *
FROM public.${odps_table_name}
WHERE ${partition_key}='${datetime1}';
```

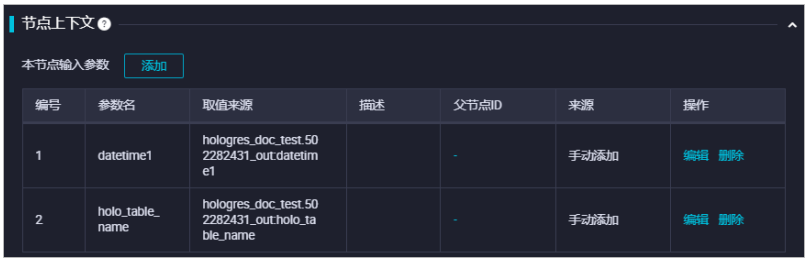
需要将基础参数绑定至该模块上游，用于控制其中的参数变量，系统配置如下：

- 替换子表

  该步骤是一个Hologres SQL模块，用于替换已有子表。将替换子表相关过程放在一个事务中，保证执行的事务性，SQL代码如下。

```
-- 已有子表时替换子表
BEGIN ;
-- 删除已经存在的子表
DROP TABLE IF EXISTS ${holo_table_name}_${datetime1};
-- 将临时表改名
ALTER TABLE ${holo_table_name}_tmp_${datetime1} RENAME TO ${holo_table_name}_${datetime1}
;
-- 将临时表绑定至指定分区表
ALTER TABLE ${holo_table_name} ATTACH PARTITION ${holo_table_name}_${datetime1}
FOR VALUES IN ('${datetime1}');
COMMIT ;
```

需要将基础参数绑定至该模块上游，用于控制其中的参数变量，系统配置如下。



- 收集分区表的统计信息

  该步骤是一个Hologres SQL模块，收集父表的统计信息，SQL代码如下。

```
-- 大量数据导入后执行ANALYZE分区表父表操作
ANALYZE ${holo_table_name};
```

需要将基础参数绑定至该模块上游，用于控制其中的参数变量，系统配置如下。

- 清理过期子表

  生产环境中，数据具备生命周期，对于超期的分区需要清理。

  现以仅在Hologres中存储最近31天的分区为例，由于之前设置的参数为datepre31=${yyyymmdd-31}，所以清理过期子表的SQL代码如下。

```
-- 清理过期子表
BEGIN ;
DROP TABLE IF EXISTS ${holo_table_name}_${datepre31};
COMMIT ;
```

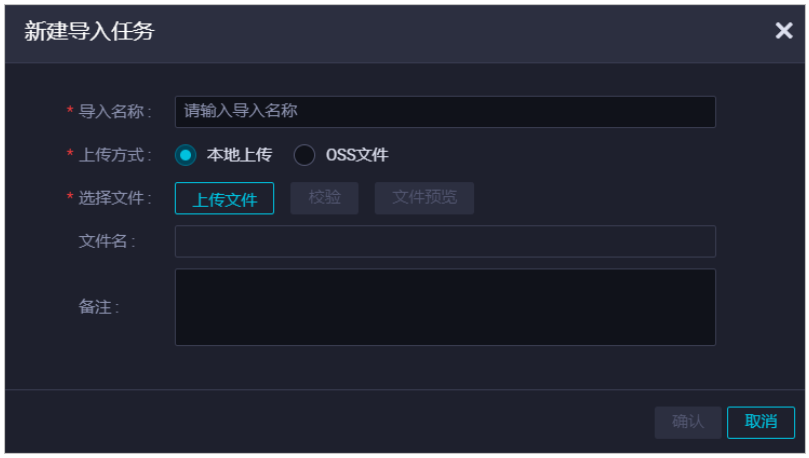
所以在作业运行时，如果bizdate=20200309，则datepre31=20200207，这样即可达到清理分区的目的。

同时需要将基础参数绑定至该模块上游，用于控制其中的参数变量，系统配置如下。

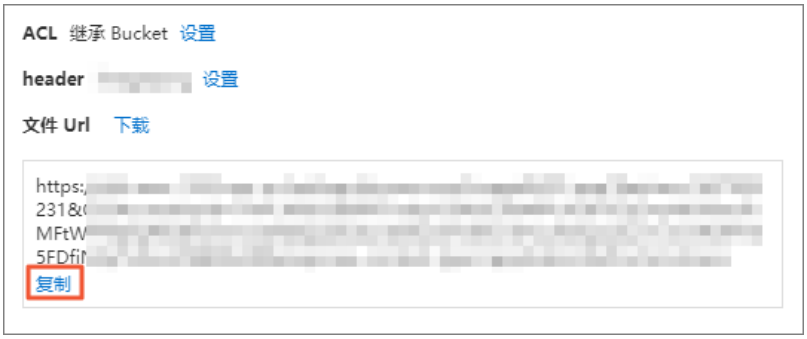
使用迁移工具导入DataWorks作业

- 考虑到作业较为复杂，所以可以利用DataWorks的迁移助手功能，将以下文件导入您的项目中，您即可获得以上说明的DataWorks的作业，之后按照您的具体业务需求更改部分参数或脚本即可。
- DataWorks迁移助手的详细介绍，请参见DataWorks迁移助手介绍及实践。
- 下载如下作业包：DataWorks作业包。

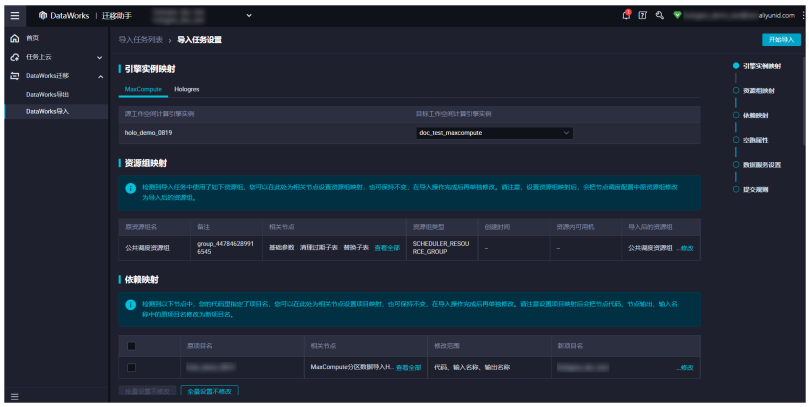
1. 进入DataWorks迁移助手，详情请参见进入迁移助手。
2. 在迁移助手的左侧导航栏，单击**DataWorks迁移 > DataWorks导入**。
3. 在**DataWorks导入**页面，单击右上方的**新建导入任务**。
4. 在**新建导入任务**对话框中，配置各项参数。



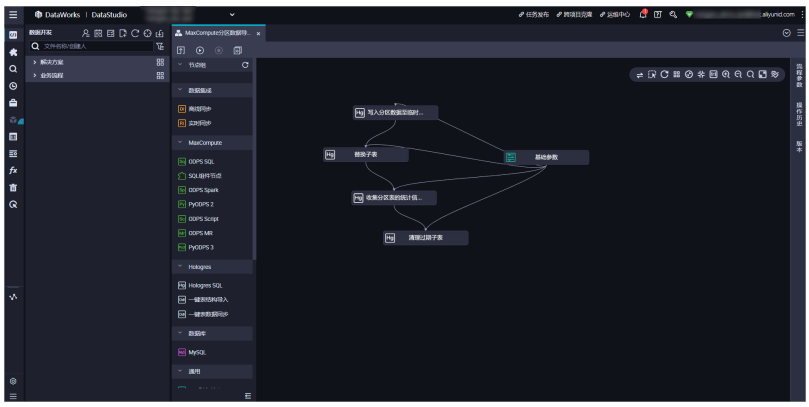
参数	描述
导入名称	自定义名称。导入名称仅支持大小写字母、中文、数字、下划线（_）和英文句号（.）。

参数	描述
上传方式	上传文件的方式。 ○ **本地上传**：上传导出包文件小于或等于30 MB时，您可使用本方式上传导出包文件到DataWorks工作空间中。 ○ **OSS链接**：上传导出包文件大于30 MB时，请将导出包文件上传至OSS存储。在OSS存储控制台文件详情页面复制URL链接，将获取到的OSS链接上传至DataWorks工作空间中。OSS上传操作请参见上传文件，获取OSS下载链接请参见分享文件。 
备注	对导入任务进行简单描述。

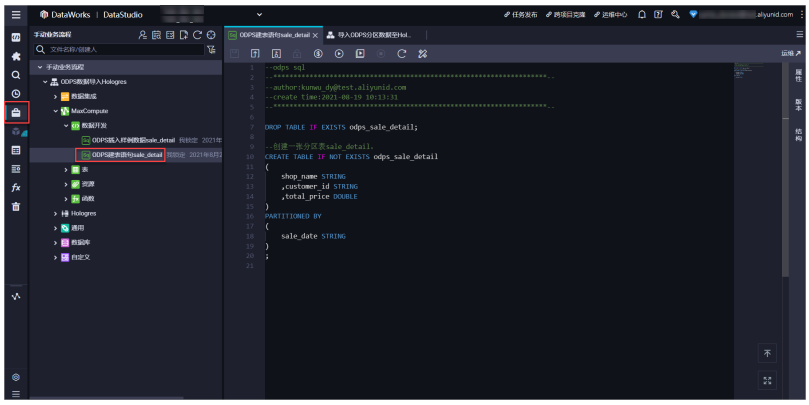
5. 单击**确认**，进入**导入任务设置**页面，设置匹配关系。



6. 单击右上方的**开始导入**，在**请确认**对话框中，单击**确认**。
   i. 导入成功后，在您的数据开发模块中则会出现以上提及的周期性作业。



ii. 同时在手工作业流程中会出现相关的DDL语句。



5.2.2.4. 将Hologres作为MaxCompute的外部表进行访问

MaxCompute与Hologres在底层无缝打通，您可以在MaxCompute中，将Hologres的表定义为MaxCompute的外部表直接进行访问。通过MaxCompute的JDBC驱动方式查询Hologres数据源的数据，该方式无数据冗余，无需数据导入导出。

使用限制

在MaxCompute中创建Hologres外部表的使用限制如下：

- MaxCompute不支持对创建的Hologres外部表进行更新（Update）、删除（Delete）操作。
- Hologres的分区表和MaxCompute的分区表没有对应关系。MaxCompute的外部表不支持分区。
- 在运行时，MaxCompute会通过JDBC接口，批量拉取Hologres对应表的所有数据到MaxCompute的查询作业中，会占用较多连接和计算资源，应避免在高并发场景使用外表查询。
- MaxCompute的Hologres外部表对Array类型的支持尚不完善，会因为底层序列化问题导致数据无法插入到Hologres外部表。一般报错：`Can't infer the SQL type to use for an instance of java.util.ArrayList`。

应用场景

在MaxCompute中创建Hologres外部表的应用场景如下：

- Hologres作为加工结果表：在MaxCompute中进行数据加工ETL，将加工后的数据直接写入Hologres外部表，再通过Hologres对外提供分析查询、在线服务，减少数据集成的调度和配置工作，开发效率更高。
- Hologres作为加工维度表：Hologres具备实时更新的能力，因此适合作为维度表的统一存储，提供实时动态更新能力，供MaxCompute、Flink等数据处理系统访问。只需要一份维度数据存储，无冗余，保证了维度数据的一致性。事实表保存在MaxCompute中，维度表保存在Hologres中，MaxCompute在ETL加工过程中关联维度表数据进行关联分析。

以上场景的具体使用操作请参见MaxCompute文档Hologres外部表。

5.2.2.5. 通过SQL导出数据至MaxCompute

本文将为您介绍如何在Hologres中通过SQL方式将数据导出至MaxCompute。

前提条件

- 开通MaxCompute并连接开发工具，请参见通过查询编辑器使用MaxCompute。
- 开通Hologres并连接开发工具，请参见连接HoloWeb。

注意事项

在Hologres中使用SQL导出数据至MaxCompute需要注意如下事项：

- 仅Hologres V0.9及以上版本支持使用SQL导出数据至MaxCompute，如果您的实例是V0.9以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 支持跨区域导出至MaxCompute，但因为网络原因，同一个区域导出的性能会更好。
- 当前Hologres仅支持一级分区，但是可以导出至MaxCompute的二级分区，但是需要MaxCompute的分区值与Hologres字段逐一对应。同时Hologres的分区表也可以导入至MaxCompute的非分区表。
- Hologres的数据类型与MaxCompute的数据类型逐一对应，但Hologres当前不支持DATE、ARRAY、MAP、STRUCT等复杂数据类型，其余数据类型映射请参见数据类型汇总。
- ODPS写入服务器数量有限，建议您避开凌晨生产作业高峰期，以获得更佳性能。
- 如果存在TIMESTAMPTZ类型的字段，字段的取值范围为1677-09-21 00:00:00到2262-04-12 00:00:00。
- Hologres从V1.1版本开始，默认支持导出至MaxCompute，如果您的Hologres实例版本低于V1.1，请在`insert`语句前添加以下语句。

```
--V0.9和V0.10版本需要使用以下参数
set hg_experimental_enable_write_maxcompute = on;
```

- 从Hologres V1.3版本开始，支持将ARRAY类型的数据导出至MaxCompute。
- 暂不支持使用SQL将数据导出至MaxCompute的Transactional表。

操作流程

1. 在Hologres准备一张Hologres内部表（例如：holo_source），用于导出数据至MaxCompute。
2. 在MaxCompute准备一张MaxComute表用于接收数据（例如mc_sink）。
3. 在Hologres新建一张外部表，用于映射MaxCompute表（例如 mapping_foreign_table）。
4. 在Hologres通过SQL语句导出数据至MaxCompute。

```
--导出部分字段
insert into mapping_foreign_table
select x,x,x from holo_soruce;--x,x,x可以替换为您需要导出的字段名
--导出全部字段
insert into mapping_foreign_table
select * from holo_soruce;
```

下面将会分非分区表和分区表两种情况来说明具体操作。

非分区表导出至MaxCompute非分区表

1. 准备Hologres数据源表。

   在Hologres中准备一张数据表，用于将数据导出至MaxCompute。示例选择已有表其DDL如下：

```
BEGIN;
CREATE  TABLE "public"."bank" (
 "age" int8,
 "job" text,
 "marital" text,
 "education" text,
 "card" text,
 "housing" text,
 "loan" text,
 "contact" text,
 "month" text,
 "day_of_week" text,
 "duration" text,
 "campaign" int8,
 "pdays" float8,
 "previous" float8,
 "poutcome" text,
 "emp_var_rate" float8,
 "cons_price_idx" float8,
 "cons_conf_idx" float8,
 "euribor3m" float8,
 "nr_employed" float8,
 "y" int8
);
COMMIT;
```

2. 创建MaxCompute目标表。

   在MaxCompute中创建一张目标表，用于接收数据，字段顺序与字段类型需要与Hologres表逐一对应。其建表语法，请参见表操作。

```
CREATE TABLE IF NOT EXISTS mc_bank
(
 age             BIGINT COMMENT '年龄',
 job             STRING COMMENT '工作类型',
 marital         STRING COMMENT '婚否',
 education       STRING COMMENT '教育程度',
 card            STRING COMMENT '是否有信用卡',
 housing         STRING COMMENT '房贷',
 loan            STRING COMMENT '贷款',
 contact         STRING COMMENT '联系途径',
 month           STRING COMMENT '月份',
 day_of_week     STRING COMMENT '星期几',
 duration        STRING COMMENT '持续时间',
 campaign        BIGINT COMMENT '本次活动联系的次数',
 pdays           DOUBLE COMMENT '与上一次联系的时间间隔',
 previous        DOUBLE COMMENT '之前与客户联系的次数',
 poutcome        STRING COMMENT '之前市场活动的结果',
 emp_var_rate    DOUBLE COMMENT '就业变化速率',
 cons_price_idx  DOUBLE COMMENT '消费者物价指数',
 cons_conf_idx   DOUBLE COMMENT '消费者信心指数',
 euribor3m       DOUBLE COMMENT '欧元存款利率',
 nr_employed     DOUBLE COMMENT '职工人数',
 y               BIGINT COMMENT '是否有定期存款'
);
```

3. 准备一张映射到MaxCompute的Hologres外部表。

   在Hologres中准备一张外表，用于将字段映射至MaxCompute，也可以通过Import Foreign Table导入MaxCompute外表。创建外表示例DDL如下：

```
BEGIN;
CREATE FOREIGN TABLE "public"."mapping_bank" (
 "age" int8,
 "job" text,
 "marital" text,
 "education" text,
 "card" text,
 "housing" text,
 "loan" text,
 "contact" text,
 "month" text,
 "day_of_week" text,
 "duration" text,
 "campaign" int8,
 "pdays" float8,
 "previous" float8,
 "poutcome" text,
 "emp_var_rate" float8,
 "cons_price_idx" float8,
 "cons_conf_idx" float8,
 "euribor3m" float8,
 "nr_employed" float8,
 "y" int8
)
SERVER odps_server
OPTIONS (project_name 'xxx',table_name 'mc_bank');--project_name为MaxCompute的project名
，table_name为MaxCompute接收数据的表名
COMMIT;
```

4. 导出Hologres数据至MaxCompute。

   在Hologres中执行SQL语句将数据导出至MaxCompute，您可以选择导出部分字段，也可以选择导出全部字段，导出部分字段时需要保证字段顺序一致。

```
--导出部分字段数据
insert into mapping_bank
select age,job from bank;
--导出全部字段数据
insert into mapping_bank
select *from bank;
```

分区表导出至MaxCompute分区表

1. 准备Hologres数据源表。

   在Hologres中准备一张分区源表，用于将数据导出至MaxCompute。示例选择已有表其DDL如下：

```
BEGIN;
CREATE TABLE "public"."par_bank" (
 "age" int8,
 "job" text,
 "marital" text,
 "education" text,
 "default" text,
 "housing" text,
 "loan" text,
 "contact" text,
 "month" text,
 "day_of_week" text,
 "duration" text,
 "campaign" int8,
 "pdays" float8,
 "previous" float8,
 "poutcome" text,
 "emp_var_rate" float8,
 "cons_price_idx" float8,
 "cons_conf_idx" float8,
 "euribor3m" float8,
 "nr_employed" float8,
 "y" int8,
 "ds" text
)
PARTITION BY list (ds);
COMMIT;
--需要有分区子表
CREATE TABLE "public"."par_bank_20190830" PARTITION OF "public"."par_bank" FOR VALUES I
N ('20190830');
CREATE TABLE "public"."par_bank_20190901" PARTITION OF "public"."par_bank" FOR VALUES I
N ('20190901');
```

2. 创建MaxCompute目标表。

   在MaxCompute中创建一张目标表，用于接收数据，表可以是一级分区，也可以是二级分区。字段顺序与字段类型需要与Hologres表逐一对应。其建表语法，请参见表操作。

```
--情况1:MaxCompute表是一级分区
CREATE TABLE IF NOT EXISTS mc_par_bank
(
    age             BIGINT COMMENT '年龄',
    job             STRING COMMENT '工作类型',
    marital         STRING COMMENT '婚否',
    education       STRING COMMENT '教育程度',
    default         STRING COMMENT '是否有信用卡',
    housing         STRING COMMENT '房贷',
    loan            STRING COMMENT '贷款',
    contact         STRING COMMENT '联系途径',
    month           STRING COMMENT '月份',
    day_of_week     STRING COMMENT '星期几',
    duration        STRING COMMENT '持续时间',
    campaign        BIGINT COMMENT '本次活动联系的次数',
    pdays           DOUBLE COMMENT '与上一次联系的时间间隔',
    previous        DOUBLE COMMENT '之前与客户联系的次数',
```

```
    previous        DOUBLE COMMENT '之前与客户联系的次数',
    poutcome        STRING COMMENT '之前市场活动的结果',
    emp_var_rate    DOUBLE COMMENT '就业变化速率',
    cons_price_idx DOUBLE COMMENT '消费者物价指数',
    cons_conf_idx   DOUBLE COMMENT '消费者信心指数',
    euribor3m       DOUBLE COMMENT '欧元存款利率',
    nr_employed     DOUBLE COMMENT '职工人数',
    y               BIGINT COMMENT '是否有定期存款'
)
PARTITIONED BY
(
    ds              STRING
);
--为一级分区添加分区
alter table mc_par_bank add if not exists partition (ds='20190830');
alter table mc_par_bank add if not exists partition (ds='20190901');
--情况二：MaxCompute表是二级分区
CREATE TABLE IF NOT EXISTS mc_par_bank_2
(
    age             BIGINT COMMENT '年龄',
    job             STRING COMMENT '工作类型',
    marital         STRING COMMENT '婚否',
    education       STRING COMMENT '教育程度',
    default         STRING COMMENT '是否有信用卡',
    housing         STRING COMMENT '房贷',
    loan            STRING COMMENT '贷款',
    contact         STRING COMMENT '联系途径',
    month           STRING COMMENT '月份',
    day_of_week     STRING COMMENT '星期几',
    duration        STRING COMMENT '持续时间',
    campaign        BIGINT COMMENT '本次活动联系的次数',
    pdays           DOUBLE COMMENT '与上一次联系的时间间隔',
    previous        DOUBLE COMMENT '之前与客户联系的次数',
    poutcome        STRING COMMENT '之前市场活动的结果',
    emp_var_rate    DOUBLE COMMENT '就业变化速率',
    cons_price_idx DOUBLE COMMENT '消费者物价指数',
    cons_conf_idx   DOUBLE COMMENT '消费者信心指数',
    euribor3m       DOUBLE COMMENT '欧元存款利率',
    nr_employed     DOUBLE COMMENT '职工人数'
)
PARTITIONED BY
(
    y               BIGINT,
    ds              STRING
);
alter table mc_par_bank_2 add if not exists partition (y='1',ds='20190830');
alter table mc_par_bank_2 add if not exists partition (y='1',ds='20190901');
```

3. 准备一张映射到MaxCompute的Hologres外部表。

   在Hologres中准备一张外表，用于将字段映射至MaxCompute。创建外表示例DDL如下：

```
--映射MaxCompute的一级分区表
BEGIN;
CREATE FOREIGN TABLE "public"."mapping_par_bank" (
```

```
 "age" int8,
 "job" text,
 "marital" text,
 "education" text,
 "default" text,
 "housing" text,
 "loan" text,
 "contact" text,
 "month" text,
 "day_of_week" text,
 "duration" text,
 "campaign" int8,
 "pdays" float8,
 "previous" float8,
 "poutcome" text,
 "emp_var_rate" float8,
 "cons_price_idx" float8,
 "cons_conf_idx" float8,
 "euribor3m" float8,
 "nr_employed" float8,
 "y" int8,
 "ds" text
)
SERVER odps_server
OPTIONS (project_name 'xxx',table_name 'mc_par_bank');
COMMIT;
--映射MaxCompute的二级分区表
BEGIN;
CREATE FOREIGN TABLE "public"."mapping_par_bank_2" (
 "age" int8,
 "job" text,
 "marital" text,
 "education" text,
 "default" text,
 "housing" text,
 "loan" text,
 "contact" text,
 "month" text,
 "day_of_week" text,
 "duration" text,
 "campaign" int8,
 "pdays" float8,
 "previous" float8,
 "poutcome" text,
 "emp_var_rate" float8,
 "cons_price_idx" float8,
 "cons_conf_idx" float8,
 "euribor3m" float8,
 "nr_employed" float8,
 "y" int8,
 "ds" text
)
SERVER odps_server
OPTIONS (project_name 'xxx',table_name 'mc_par_bank_2');//project_name为MaxCompute的project名，table_name为MaxCompute接收数据的表名
```

```
COMMIT;
```

4. 导出Hologres数据至MaxCompute。

   在Hologres中执行SQL语句将数据导出至MaxCompute，您可以选择导出部分字段，也可以选择导出全部字段，导出部分字段时需要保证字段顺序一致。

```
--导出至MaxCompute一级分区表
--示例1：加where条件的分区父表
INSERT INTO mapping_par_bank SELECT * FROM "public"."par_bank" WHERE ds='20190830';
--示例2：直接通过分区子表导出
insert into mapping_par_bank select * from "public"."par_bank_20190901";
--导出至MaxCompute二级分区表
--示例1：加where条件的分区父表
INSERT INTO mapping_par_bank_2 SELECT * FROM "public"."par_bank" WHERE y='1' and ds='20
190830';
--示例2：直接通过Hologres分区子表导出
INSERT INTO mapping_par_bank_2 SELECT * FROM "public"."par_bank_20190901" where y='1';
```

5.2.2.6. 执行MaxCompute SQL语句

从Hologres V0.10版本开始，Hologres与MaxCompute集成性进一步提升，支持在Hologres中执行MaxCompute的SQL语句，方便您直接快速操作MaxCompute。本文将为您介绍，在Hologres中如何执行MaxCompute SQL语句。

前提条件

- 已开通Hologres并连接开发工具，本文使用HoloWeb，详情请参见连接HoloWeb。
- 已开通MaxCompute，详情请参见通过查询编辑器使用MaxCompute。
- 在Hologres中执行MaxCompute SQL需要当前用户具有在MaxCompute中执行SQL的权限。关于MaxCompute的权限，详情请参见用户规划与管理。

使用限制

- 仅Hologres V0.10 及以上版本支持在Hologres中执行MaxCompute SQL语句。请前往Hologres管控台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- 目前建议您在Hologres中仅执行MaxCompute的DDL语句，如果需要执行DML语句，请前往MaxCompute进行操作。
- 每次调用只能执行一条SQL语句。

命令语法

- **语法示例**

```
select exec_external_sql(
  'server',
  'database',
  'sql' ,
  timeout_ms,
  'options'
);
```

如上参数必须要按照顺序填写，若是需要省略某些参数，需要显示指定参数名，如下所示：

```
exec_external_sql(
            server:='odps_server' ,
            database:='odps_project_name' ,
            sql:='sql' ,
            timeout_ms:=timeout_ms ,
            options:='options'
)
```

- **参数说明**

参数名	参数含义	说明	示例
server	外部服务器名称。您可以直接调用Hologres底层已创建的名为odps_server的外部表服务器。此函数目前仅支持odps server，外部server详细原理请参见Postgres FDW。	如果为空字符串，会取odps_server作为外部server。	`'odps_server'`
database	MaxCompute的project名。	无	`'seahawks'`
sql	需要执行的MaxCompute SQL（建议只执行DDL语句）： ○ create table ○ alter table ○ desc table ○ drop table	SQL语句需要符合MaxCompute的语法，建议只执行DDL语句。如果SQL中有单引号，需要在SQL语句前后添加双美元符号（`$$SQL$$`）实现单引号转义。 **说明** 在HoloWeb和HoloStudio的临时查询页面暂不支持转义字符。	`'CREATE TABLE IF EXISTS MC_TBL ;'`
timeout_ms	执行超时时间，单位ms。	缺省或者小于0的情况下会将该参数值设置为60000ms。超过超时时间将会退出，同时向MaxCompute发送一个取消指令。	50000

参数名	参数含义	说明	示例
options	使用DataWorks或者MaxCompute客户端提交SQL时，通常需要设置的SQL Flag。	具体的配置项请参见SET操作，若是有多个flag需要设置，需要将字段类型改成JSON格式。	{"odps.sql.type.system.odps2":"true"} 或者 { "odps.sql.type.system.odps2": "true", "odps.sql.decimal.odps2=":"true"}

使用示例

您可以在Hologres中执行MaxCompute SQL，如果执行建表语句则会在MaxCompute对应的project中创建一张表。当前版本建议仅执行DDL语句。

- **示例一：创建MaxCompute非分区表**

```
select exec_external_sql(
            'odps_server',
            'mc_project' , --project名
            'create table par_mc_table (id int,name string);' ,--在MaxCompute中创建一张表
            5000  --超时时间为5000ms
            );
```

- **示例二：创建MaxCompute分区表并指定分区**

```
--创建分区表
select exec_external_sql(
          server:='odps_server',
          database:='mc_project', --maxcompute的project名
          sql:='create table par_mc_table (id int,name string) partitioned by (pt string)
;', --创建分区表
          timeout_ms:=10000--超时时间为10000ms
                  );
--指定分区表的分区
select exec_external_sql(
           'odps_server',
           'mc_project', --maxcompute的project名
           $$ALTER TABLE par_mc_table ADD IF NOT EXISTS partition(pt='202102');$$--指定分
区
                  );
```

- **示例三：跨Region创建MaxCompute表**

```
select exec_external_sql(
            'hangzhou_odps_server' ,
            'hologres_test' ,
            'create table mc_test(id int,create_time datetime,decimal_column decimal(38,
10));' ,
            50000,    --超时时间
            '{
            "odps.sql.type.system.odps2":"true",
            "odps.sql.decimal.odps2":"true"
  }'                  --MaxCompute的flag设置，分别代表开启新数据类型和开启decimal类型
);
```

- **示例四：删除MaxCompute中的表**

```
select exec_external_sql(
            'odps_server',
            'mc_project',  --MaxCompute的project名
            'drop table if exists mc_table;' ,--删除MaxCompute的表
            50000
            );
```

- **示例五：配合“通过SQL方式导出MaxCompute”功能使用**

  从Hologres V0.9版本开始，Hologres支持通过SQL导出数据至MaxCompute，但是该方法需要在MaxCompute中提前创建接收数据的表，操作比较麻烦。从Hologres V0.10版本开始，通过在Hologres中执行MaxCompute SQL，即可创建表，再将数据导出，支持一站式开发。

  示例操作将Hologres中非分区数据导入至MaxCompute非分区表。具体如下：

  i. 在Hologres准备一张Hologres内部表（例如：holo_table），用于导出数据至MaxCompute，示例DDL和数据如下：

  ```
  create table "public"."holo_table" (
   "id" int4,
   "name" text
  );
  insert into "public"."holo_table" values
  (1,'a'),
  (2,'b'),
  (3,'c');
  ```

  ii. 创建一张MaxCompute表，用于在MaxCompute中接收数据。

  ```
  select exec_external_sql(
              'odps_server',
              'mc_project' , --project名
              'create table mc_sink_table (id int,name string);' ,--在MaxCompute中创建一
  张接收数据表
              5000--超时时间为5000ms
              );
  ```

  iii. 在Hologres新建一张外部表，用于映射MaxCompute表。

```
begin;
create foreign table "public"."mc_mapping_foreign_table" (
 "id" int4,
 "name" text
)
server odps_server_bj
options (project_name 'default_project_2361b62', table_name 'mc_sink_table');
commit;
```

iv. 在Hologres通过SQL语句导出数据至MaxCompute。

- 全部字段数据导出

```
set hg_experimental_enable_write_maxcompute = on;-- 由于是beta功能，需要打开GUC参数
insert into mc_mapping_foreign_table
select * from holo_table;
```

- 部分字段数据导出

```
set hg_experimental_enable_write_maxcompute = on;-- 由于是beta功能，需要打开GUC参数
insert into mc_mapping_foreign_table (name)
select name from holo_table;
```

更多关于数据导出的操作说明请参见通过SQL导出数据至MaxCompute。

5.2.2.7. MaxCompute外部表自动加载

Hologres作为和MaxCompute深度结合的产品，无需数据搬迁，即可通过外部表查询MaxCompute的数据。Hologres从1.1.43版本开始支持MaxCompute外部表自动加载功能。本文将为您介绍如何使用MaxCompute外部表自动加载功能。

技术原理

Hologres支持将MaxCompute表信息映射为Hologres的外部表，表信息的自动加载通过两种手段共同实现。

- 周期性同步元数据，这种方式定时同步MaxCompute表的元信息，并更新外部表定义语句，这些需要自动同步的MaxCompute Project需要显式定义。
- 访问时增加重试机制，当实际访问时遇到尚未创建的外部表时，会通过自动重试的机制，即刻创建相关外部表。重试对应用层透明，当前一个SQL中最多自动重试创建六张外部表，超出六张时，SQL会失败，应用层重新提交SQL可以自动创建全部外部表。

使用限制

- 仅Hologres V1.1.43及以上版本支持MaxCompute外部表自动加载，如果您的实例是V1.1.43以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 打开MaxCompute外部表自动加载后，进行查询，若发现外部表不存在时，Hologres会自动根据查询中的MaxCompute中的Project Name和Table Name创建外部表（MaxCompute中的Project Name作为Hologres外部表所在的Schema名称，以Table Name作为Hologres外部表名称）。
  - 如果已经存在相同的Schema和Table名称的内部表，实际查询的是Hologres的内部表，此时并不会触发MaxCompute外部表自动加载功能。
  - 由于自动加载时会创建外部表，需要查询账号拥有该数据库的创建、删除Schema和Table的权限。若使用自动加载已经创建出了外部表，查询账号仅需要查询权限即可。
- 使用MaxCompute外部表自动加载时，MaxCompute的Project Name不能以hg_或者holo_开头，这些是

Hologres的保留关键字。

- 自动创建外部表时，对应MaxCompute表中存在Hologres外部表目前不支持的数据类型，则该外部表无法自动创建，需要您手动选择支持的字段创建外部表。
- 不建议将表超过1000张的MaxCompute的项目设置为默认Project。
- 不建议设置自动加载时间小于5分钟。

使用命令

- 打开MaxCompute外部表自动加载功能

  使用如下SQL，数据库打开自动加载的功能。

  ```
  ALTER DATABASE <database name> SET hg_experimental_enable_auto_load_foreign_table = on;
  ```

  database name为需要开启MaxCompute外部表自动加载的数据库名称。MaxCompute外部表自动加载默认为关闭状态，即默认值为off。

- 关闭MaxCompute外部表自动加载功能

  使用如下SQL，数据库关闭自动加载功能。

  ```
  ALTER DATABASE <database name> SET hg_experimental_enable_auto_load_foreign_table = off;
  ```

  database name为需要关闭MaxCompute外部表自动加载的数据库名称。

- 设置默认加载的MaxCompute的Project

  若您需要默认自动加载某些MaxCompute的Project，使用如下SQL设置默认MaxCompute的Project，以便对接BI时可以自动加载这些Project中的表。

  ```
  ALTER DATABASE <database name>
  SET hg_experimental_default_odps_project_list='<odps_project_name_1>,<odps_project_name_2>...';
  ```

  参数说明如下。

参数	说明
database name	需要设置MaxCompute外部表自动加载的数据库名称。
odps_project_name_1	MaxCompute的项目名称。
odps_project_name_2	MaxCompute的项目名称。

  > **说明**
  > - 如果需要更新清单，只需要重新设置即可，系统会全量覆盖默认加载清单。
  > - 默认值为空，即不周期性的加载任何Project中表的元数据。

  配置默认MaxCompute的Project后，系统周期性的同步Project清单中表的元数据，您可以使用如下参数修改同步的时间间隔。

```
-- 查看数据库的自动加载间隔时间
show hg_experimental_load_all_foreign_table_interval_time;
-- 修改数据库的自动加载间隔时间
ALTER DATABASE <database name> SET hg_experimental_load_all_foreign_table_interval_time =
'5min';
```

database name为需要设置MaxCompute外部表自动加载的数据库名称。默认间隔时间5分钟。

- 查看默认加载的MaxCompute的Project

  您可以使用如下SQL查看设置的默认加载的MaxCompute的Project。

  ```
  show hg_experimental_default_odps_project_list;
  ```

- 删除默认加载的MaxCompute的Project

  您可以使用如下SQL删除默认加载的MaxCompute的Project。

  ```
  ALTER DATABASE <database name> SET hg_experimental_default_odps_project_list='';
  ```

  database name是需要删除MaxCompute默认Project的数据库名称。

使用示例

- 打开数据库demo的MaxCompute外部表自动加载。

  ```
  ALTER DATABASE demo SET hg_experimental_enable_auto_load_foreign_table = on;
  ```

- 将MaxCompute的Project：odps_hologres设置为默认加载的Project，系统会根据 `hg_experimental_load_all_foreign_table_interval_time` 的配置自动周期性的加载odps_hologres这个Project中表的元数据。

  ```
  ALTER DATABASE demo SET hg_experimental_default_odps_project_list='odps_hologres';
  ```

- 设置自动加载间隔时间为10分钟。

  ```
  ALTER DATABASE demo SET hg_experimental_load_all_foreign_table_interval_time = '10min';
  ```

5.2.3. OSS

5.2.3.1. 使用COPY命令导出Hologres的数据至OSS

阿里云对象存储（Object Storage Service，简称 OSS）是阿里云提供的安全、低成本及高可靠的云存储服务。本文指导您使用 `copy to` 命令语句和 `hg_dump_to_oss` 命令语句将查询的数据导出到指定的OSS。

使用限制

- 仅当前Hologres实例的Superuser或拥有 `pg_execute_server_program` 权限的用户，才可以使用 `hg_dump_to_oss` 导出Hologres的数据至OSS。Superuser可以授予其他用户 `pg_execute_server_program` 权限，命令如下。

  ```
  --DB开启简单权限模型，执行以下语句
  call spm_grant('pg_execute_server_program','云账号ID/云邮箱/RAM账号');
  --DB使用的是专家权限模型，执行以下语句
  grant pg_execute_server_program to 云账号ID/云邮箱/RAM账号;
  ```

- 单次导入至OSS的数据量不能超过5GB。

命令介绍

- COPY命令：`COPY`命令是PostgreSQL表和标准文件系统之间移动数据的工具，Hologres支持`COPY`命令。`COPY TO`语句将SELECT查询的结果内容复制到一个文件或者其他输出介质中。`COPY FROM`语句用于从一个文件复制数据到一个表。
- hg_dump_to_oss命令：用于将在Hologres中查询的结果dump到指定的OSS。

COPY命令

如下内容将为您介绍`COPY TO`命令的具体语法格式和参数说明：

- 命令格式

```
COPY ( query ) TO { PROGRAM 'command' | STDOUT }
    [ [ WITH ] ( option [, ...] ) ]
```

其中，option可以是下列之一：

```
    FORMAT [format_name]
    DELIMITER 'delimiter_character'
    NULL 'null_string'
    HEADER
    QUOTE 'quote_character'
    ESCAPE 'escape_character'
    ENCODING 'encoding_name'
```

- 参数说明

参数	描述
query	输入的查询语句。查询语句前后的圆括号必需保留。
PROGRAM	一个要执行的命令，输出会写入到该命令的标准输入。
STDOUT	指定输出，该输出将同步到到客户端应用。
FORMAT	选择数据格式，其格式可以为text，csv 或者binary。默认为text。
DELIMITER	指定分隔文件每行中各列的字符，这必须是一个单一的单字节字符。各数据格式的字符说明如下： ○ text格式中默认是一个制表符（如\t）。 ○ csv格式默认是一个逗号（,）。 ○ binary格式时不支持使用这个选项。
NULL	指定表示一个空值的字符串。各数据格式的字符说明如下： ○ text格式中默认是\N。 ○ csv格式默认是一个未加引用的空串。 ○ binary格式时不支持使用这个选项。
HEADER	指定文件包含标题行，其中有每一列的名称。只有csv格式支持该选项。

参数	描述
QUOTE	指定应该出现在一个匹配QUOTE值的数据字符之前的字符，这必须是一个单一的单字节字符。 默认和QUOTE值一样。只有csv格式支持该选项。
ENCODING	指定文件按照encoding_name编码。如果省略，将使用当前的客户端编码。

hg_dump_to_oss命令

如下内容将为您介绍 `hg_dump_to_oss` 命令的具体语法格式和参数说明，在Hologres中 `hg_dump_to_oss` 命令与 `COPY TO` 命令组合在一起使用。

- 命令格式

```
COPY (query) TO PROGRAM 'hg_dump_to_oss --AccessKeyId <accessid> --AccessKeySecret <accesskey> --Endpoint <ossendpoint> --BucketName <bucketname> --DirName <dirname> --FileName <filename> --BatchSize <xxx> ' (DELIMITER ',', HEADER true, FORMAT CSV);
```

> 说明 <dirname>前请不要添加 `/` 、 `\` 等字符。

- 参数说明

参数	描述	示例
query	输入的查询语句。	`select * from dual;`
AccessKeyId	当前账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。	无
AccessKeySecret	当前账号的AccessKey SECRET。 您可以单击AccessKey 管理，获取AccessKey Secret。	无
Endpoint	OSS的经典网络访问域名。 您可以单击Bucket列表页面的目标Bucket名称，进入Bucket详情页查看。	*oss-cn-beijing-internal.aliyuncs.com*
BucketName	OSS对应的Bucket名称。	*dummy_bucket*
DirName	OSS存放输出结果的目录。	*testdemo/*
FileName	（可选）OSS对应的文件名称。	*file_name*
BatchSize	每次执行 `hg_dump_to_oss` 的行数，默认为*1000*。	*5000*

参数	描述	示例
DELIMITER	结果列之间的分隔符，默认为制表符（Tab-separated Values，简称TSV）。	','

使用示例

在Hologres中 `hg_dump_to_oss` 命令与 `COPY TO` 命令的使用示例如下。

```
-- 将Hologres内部表数据dump到指定OSS
COPY (select * from holo_test LIMIT 2) TO PROGRAM 'hg_dump_to_oss --AccessKeyId <access id>
--AccessKeySecret <access key> --Endpoint oss-cn-hangzhou-internal.aliyuncs.com --BucketNam
e hologres-demo --DirName holotest/ --FileName file_name --BatchSize 3000' DELIMITER ',';
-- 将Hologres外部表数据dump到指定OSS
COPY (select * from foreign_holo_test LIMIT 20) TO PROGRAM 'hg_dump_to_oss --AccessKeyId <a
ccess id> --AccessKeySecret <access key> --Endpoint oss-cn-hangzhou-internal.aliyuncs.com -
-BucketName hologres-demo --DirName holotest/ --FileName file_name --BatchSize 3000' (DELIM
ITER ',', HEADER true);
-- 跨region dump到指定OSS
COPY (select * from holo_test_1 LIMIT 20) TO PROGRAM 'hg_dump_to_oss --AccessKeyId <access
id> --AccessKeySecret <access key> --Endpoint oss-cn-beijing-internal.aliyuncs.com --Bucket
Name hologres-demo --DirName holotest/ --FileName file_name --BatchSize 3000' (DELIMITER ',
', HEADER true, FORMAT CSV);
```

> **说明** Hologres支持跨地域导出数据至指定的OSS。例如，可以导出杭州地域的实例查询出的数据至北京地域的OSS。

常见问题

常见的报错内容及解决方法如下：

- `ERROR: syntax error at or near ")"LINE 1: COPY (select 1,2,3 from ) TO PROGRAM 'hg_dump_to_oss2 --Acce...`

  输入的query有误，请检查对应的查询语句。

- `DETAIL: child process exited with exit code 255`

  选择的OSS网络类型有误。如果您使用的是公共云，请选择**经典网络**。

- `DETAIL: command not found`

  您需要配置 `DUMP TO OSS` 的**program**为*hg_dump_to_oss*，否则会出现该报错。

- `ERROR: program "hg_dump_to_oss ...” failed DETAIL: child process exited with exit code 101`

  输入的AccessKeyId不合法，请使用当前账号的AccessKey ID。

- `ERROR: program "hg_dump_to_oss ...” failed DETAIL: child process exited with exit code 102`

  输入的AccessKeySecret不合法，请使用当前账号的AccessKey Secret。

- `ERROR: program "hg_dump_to_oss ...” failed DETAIL: child process exited with exit code 103`

输入的Endpoint不合法，请确认对应OSS经典网络的Endpoint。

- ```
  ERROR: program "hg_dump_to_oss ...” failed DETAIL: child process exited with exit code 104
  ```

  输入的BucketName不合法，请确认对应的Bucket名称。

- ```
  ERROR: program "hg_dump_to_oss ..." failed DETAIL: child process exited with exit code 105
  ```

  缺少参数，请对照参数说明，检查必选参数是否均已配置。

- ```
  ERROR: program "hg_dump_to_oss ...” failed DETAIL: child process exited with exit code 255
  ```

  一般情况下是由于holo server与指定的OSS网络不通导致该报错，可以更换OSS域名（例如：OSS网络类型选择经典网络）。更多关于OSS的域名信息，请参见访问域名和数据中心。

5.2.3.2. 通过外部表直接访问OSS

本文将为您介绍如何通过创建外部表的方式加速查询OSS数据。

前提条件

- 已开通Hologres并连接开发工具，本文使用HoloWeb，请参见连接HoloWeb。
- 已开通OSS并准备好数据，请参见开始使用OSS。
- 已进行OSS授权操作，通过外部表方式访问OSS数据，需要访问的账号有OSS的相关访问权限，否则即使创建外表成功了，也无法查询数据。OSS授权请参见授权策略。

背景信息

对象存储服务（Object Storage Service，OSS）是一种海量、安全、低成本、高可靠的云存储服务，适合存放任意类型的文件。Hologres通过与OSS打通，支持通过创建外部表的方式，无需导入导出就能直接加速查询OSS的数据（数据仍然存储在OSS，外部表只做字段映射）。

Hologres兼容Postgres，通过创建外部表的方式访问OSS数据的原理同Postgres FDW。更多关于原理的描述，请参见Postgres FDW。

使用限制

通过创建外部表直接访问OSS的使用限制具体如下：

- 该函数仅Hologres V0.10及以上版本支持，请前往Hologres管控台实例详情页查看当前版本，如果您是V0.10以下的版本，请您提交工单升级实例。
- 在Hologres创建外部表之前，需要执行以下语句创建extension才可以使用。创建extension需要实例的Superuser执行，该操作针对整个DB生效，一个DB只需执行一次。

  ```
  create extension oss_fdw;
  ```

- 目前只支持部分数据类型，具体包括：text、bigint、int、float、double precision、boolean、date。
- 不支持访问OSS分区表，请使用非分区表。
- 目前查询数据不支持跳过头部N行。
- 如果您需要将Hologres数据导出到OSS，详情请参见使用COPY命令导出Hologres的数据至OSS。
  ```

数据类型映射

PARQUET文件、ORC文件分别与Hologres的数据类型映射关系如下表所示。

PARQUET文件数据类型	Hologres数据类型
INT	INT
BIGINT	BIGINT
STRING	TEXT
BOOLEAN	BOOLEAN
DATE	DATE
TIMESTAMP	若原文件中包含TIMESTAMP类型的列，建表时可以设置为TEXT类型，建表可以创建成功，但是无法查询。
FLOAT	FLOAT4
DOUBLE	FLOAT8
DECIMAL	若原文件中包含DECIMAL类型的列，建表时可以设置为TEXT类型，建表可以创建成功，但是无法查询。
CHAR(n)	TEXT
VARCHAR(n)	TEXT

ORC文件数据类型	Hologres数据类型
INT	INT
BIGINT	BIGINT
STRING	TEXT
BOOLEAN	BOOLEAN
DATE	DATE
TIMESTAMP	若原文件中包含TIMESTAMP类型的列，建表时会报错。
FLOAT	FLOAT4
DOUBLE	FLOAT8
DECIMAL	若原文件中包含DECIMAL类型的列，建表时会报错。
CHAR(n)	TEXT

ORC文件数据类型	Hologres数据类型
VARCHAR(n)	TEXT

创建外部表直接访问OSS数据

通过在Hologres中创建外部表的方式直接访问OSS操作步骤如下：

1. 创建extension

   在Hologres中创建外部表之前，需要Superuser在DB中执行以下语句创建extension。创建extension需要实例的Superuser执行，该操作针对整个DB生效，一个DB只需执行一次。

   ```
   create extension oss_fdw;
   ```

   > **说明** 如需卸载extension请执行如下命令。
   >
   > ```
   > DROP extension oss_fdw;
   > ```

2. 创建server

   创建完extension之后，您需要创建server用来连接OSS。

   - 语法示例

     ```
     CREATE SERVER <server_name> FOREIGN DATA WRAPPER oss_fdw
     OPTIONS (
       endpoint 'https://<oss经典网络>'
     );
     ```

   - 参数说明

参数	说明	示例
server_name	自定义的server名称。	oss_server
endpoint	OSS的endpoint需要以 `http://` 或者 `https://` 开头，并且使用OSS的经典网络（内网）。更多关于OSS的endpoint说明，请参见获取OSS经典网络（内网）地址。	http://oss-cn-shanghai-internal.aliyuncs.com

   - 使用示例

     ```
     CREATE SERVER oss_server FOREIGN DATA WRAPPER oss_fdw
     OPTIONS (
       endpoint 'https://oss-cn-shanghai-internal.aliyuncs.com'
     );
     ```

3. 创建外部表

   server创建完成后，您可以在Hologres中创建外部表直接查询OSS中的数据。您可以根据业务场景需求创建parquet类型和text类型的外部表。创建的表必须和实际存储的格式一致，否则会导致读取异常。

- 语法示例

  parquet类型具体语句如下：

```
create foreign table <tablename>(
  col type,
  col type
) server <server_name>
 options (
  dir './<dir_url>/',
  format 'parquet',
)
```

  text类型具体语句如下：

```
create foreign table <tablename>(
  col type,
  col type
) server <server_name>
 options (
  dir './<dir_url>/',
  format 'text',
  delimiter ','
)
```

- 参数说明

参数	说明	示例
tablename	外部表名称。	oss_table
col	字段名称。	id
type	字段类型。	int
server_name	创建的server名称。	oss_server
dir_url	OSS表的存储URL，需要指定到目录名称。您需要注意如下事项： ■ 不支持指定到具体文件，只支持到目录。 ■ Hologres会读取orc目录下的所有文件解析，请您保证单独目录下的格式一致，否则读取数据会出现乱码。 ■ 目录名称不能指定为 `./oss-test/orc/region_zlib_dict.orc` ，否则虽然创建表成功，但是读取的内容为空表。	orc表的存储位置具体如下： `oss://oss-test/orc/region_zlib_dict.orc` 那么，指定当前表的dir为 `./oss-test/orc/` 。 dir_url：bucket名称+目录路径。 例如： `./oss-holo-tmp/holotest/` 。 

参数	说明	示例
format	当前OSS存储的表的格式。当前支持的format包括： ■ parquet ■ orc ■ text	parquet
delimiter	text类型的分隔符。 **说明** 当format格式为 text类型时候，此项为必填项。	英文逗号（,）

- 使用示例

  在OSS中，名为oss-holo-tmp的bucket下有一个*holotest*目录。在Hologres中创建一个外表读取该目录中TEXT类型文件的数据。

```
create foreign table oostest (
    a int,
    b float,
    c text
    )
    server oss_server
 options (
  dir './oss-holo-tmp/holotest/',
  format 'text',
delimiter ','
);
```

4. 查询外部表

   创建外表成功后，可以直接查询外部表。

```
SELECT COUNT (*) FROM osstest;--查询表osstest中记录的行数。
SELECT * FROM osstest limit 10;--查询表osstest中的数据并返回10条记录。
```

性能优化

目前Hologres通过外部表直接读取OSS数据，即将某个目录下的所有文件读取出来作为一个表，在做表切分的时候，建议直接将单个文件作为一个切分单元。建议您尽量控制每个文件的大小一致，以防止某个表过大导致计算资源倾斜影响整体性能。

5.2.3.3. 通过DLF读取OSS数据

本文将为您介绍在Hologres中如何通过DLF读取OSS数据。

前提条件

- 您已开通Hologres实例并连接开发工具，并保证Hologres实例为V0.10及以上版本。本文以连接HoloWeb为例，详情请参见连接HoloWeb。

- 您已开通OSS并上传数据，详情请参见开始使用OSS。
- 您已开通DLF，详情请参见快速入门。

背景信息

阿里云数据湖构建（Data Lake Formation，DLF）是一款全托管的快速帮助用户构建云上数据湖的服务，产品提供了云上数据湖统一的权限管理、数据湖元数据管理和元数据自动抽取能力，更多关于DLF介绍请参见DLF产品简介。

DLF使用阿里云对象存储（Object Storage Service，OSS）作为云上数据湖的统一存储，从Hologres V0.10版本开始，支持以外表方式通过DLF读取OSS中的数据，方便快捷，降低开发的复杂度和运维成本。

以外部表方式通过DLF访问OSS的数据，外部表只做字段映射，不真正存储数据，数据还是存储在OSS，相关原理请参见PostgreSQL FDW。

使用限制

通过DLF读取OSS数据的使用限制具体如下：

- 该功能仅Hologres V0.10及以上版本支持，请前往Hologres管控台实例详情页查看当前版本，如果您是V0.10以下的版本，请您提交工单升级实例。
- 使用该功能需要后台配置，请您在使用之前提交工单添加后台配置。
- 使用该功能之前，需要实例的Superuser执行以下语句创建extension才可以使用。extension针对整个DB生效，一个DB只需执行一次，新建DB需要再次执行。

```
--创建extension
create extension dlf_fdw;
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP extension dlf_fdw;
> ```

- 当前仅支持读取同一个地域的数据，暂不支持跨地域读取。
- 目前支持的OSS文件格式为：Text（CSV）、SquenceFile、Parquet和ORC格式。
- 创建server时，DLF和OSS的网络类型都需要使用经典网络。
- 支持读取DLF中的分区表，但只有TEXT、VARCHAR和INT类型的数据可以作为分区键。
- 仅Hologres V1.3及以上版本支持读取Hudi和Delta格式的数据，请在Hologres管控台查看当前实例版本，若有需要请提交工单升级实例。
- 由于Spark（Delta）生成的Decimal格式与Hive不一致，需要在生成Delta生成数据的任务前加上参数，打开开关强制生成Hive可读的格式。相关社区的ISSUE。

通过DLF读取OSS数据

在开始之前，您需要在DLF中准备元数据表，并保证该表中已抽取数据，详情请参见元数据管理。在Hologres中以外部表方式通过DLF访问OSS的数据操作步骤如下：

1. 创建extension

   在Hologres中由Superuser在DB中执行以下语句创建extension，用于开启通过DLF读取OSS数据的功能。该操作针对整个DB生效，一个DB只需执行一次。

   ```
   create extension if not exists dlf_fdw;
   ```

2. 创建server

   创建完extension之后，您需要创建server用于Hologres连接DLF和OSS，需要使用经典网络连接。

   - 语法示例

```
create server if not exists <servername> foreign data wrapper dlf_fdw options
(
    dlf_region 'cn-<region>',
    dlf_endpoint '<DLF经典网络>',
    oss_endpoint '<OSS经典网络>'
);
```

   - 参数说明

参数	说明	示例
server_name	自定义的server名称。	oss_server

endpoint	■ DLF的endpoint经典网络，具体格式如下：`dlf-share.cn-<region>.aliyuncs.com`。
	■ OSS的endpoint需要使用OSS的经典网络。更多关于OSS的endpoint说明，请参见获取OSS内网地址。
		■ DLF
		`dlf-share.cn-shanghai.aliyuncs.com`
		■ OSS
		`oss-cn-shanghai-internal.aliyuncs.com`

   - 使用示例

     示例在上海区域创建一个名为dlf_server的server：

```
create server if not exists dlf_server foreign data wrapper dlf_fdw options
(
    dlf_region 'cn-shanghai',
    dlf_endpoint 'dlf-share.cn-shanghai.aliyuncs.com', --上海DLF经典网络地址
    oss_endpoint 'oss-cn-shanghai-internal.aliyuncs.com'--OSS经典网络地址
);
```

3. 创建用户映射

   默认情况下，使用当前用户的AccessKey ID和AccessKey Secret访问DLF元数据和读取表数据的操作。当前阿里云账号的AccessKey ID和AccessKey Secret，获取方式请参见创建访问密钥。如果不希望使用默认账号，也可以通过如下语句指定用户映射，查询时将使用该账号访问数据，请确保该账号有对应数据的查询权限。详细原理请参见postgres create user mapping。

```
create user mapping for <账号uid> server hm_dlf_server options
(
    dlf_access_id 'accessid', dlf_access_key 'accesskey',
    oss_access_id 'accessid', oss_access_key 'accesskey'
);
```

使用示例：

```
--为当前用户创建用户映射
create user mapping for current_user server hm_dlf_server options
(
    dlf_access_id 'LTAI5txxx', dlf_access_key 'y8LUUyyy',
    oss_access_id 'LTAI5txxx', oss_access_key 'y8LUUyyy'
);
--为RAM用户123xxx创建用户映射
create user mapping for "p4_123xxx" server hm_dlf_server options
(
    dlf_access_id 'LIlY5txxx', dlf_access_key 'KsjkXKyyy',
    oss_access_id 'LIlY5txxx', oss_access_key 'KsjkXKyyy'
);
--删除用户映射
Drop USER MAPPING for CURRENT_USER server hm_dlf_server options;
Drop USER MAPPING for "p4_123xxx" server hm_dlf_server options;
```

4. 创建外部表

server创建完成后，您可以在Hologres中使用IMPORT FOREIGN SCHEMA语法创建外部表，用于读取DLF抽取的OSS数据。

- 语法示例

```
import foreign schema remote_schema
    [ { limit to | except } ( table_name [, ...] ) ]
    from server server_name
    into local_schema
    [ options ( option 'value' [, ... ] ) ]
```

- 参数说明

参数	说明
remote_schema	DLF中创建的元数据库名。
table_name	DLF中创建的元数据表名。
server_name	Hologres中创建的server名。
local_schema	Hologres中的schema名。
options	IMPORT FOREIGN SCHEMA中的option参数取值，详情请参见IMPORT FOREIGN SCHEMA。

- 使用示例

创建一张外部表映射DLF元数据库dlfpro中元数据表dlf_oss_test的数据，该表位于Hologres中的public schema，并且检验是否存在该外部表，若存在，则对已有表更新。

```
import foreign schema dlfpro limit to
(dlf_oss_test)
from server dlf_server  into public options(if_table_exist 'update');
```

将DLF元数据库dlfpro中所有的表都映射至Hologres的public schema，将会在Hologres中批量创建同

名外部表。

```
import foreign schema dlfpro
from server dlf_server  into public options(if_table_exist 'update');
```

5. 数据查询

创建外部表成功后，可以直接查询外部表读取OSS中的数据。

```
select * from dlf_oss_test;
```

数据类型映射

PARQUET		ORC		HUDI		DELTA	
DLF数据类型	Hologres数据类型	DLF数据类型	Hologres数据类型	DLF数据类型	Hologres数据类型	DLF数据类型	Hologres数据类型
INT	INT	INT	INT	INT	INT	INT	INT
BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
STRING	text	STRING	text	STRING	text	STRING	text
BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN
DATE	DATE	DATE	DATE	DATE	DATE	DATE	DATE
TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP
FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	暂不支持
CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)	VARCHAR(n)

采用全新外部表查询引擎（Beta）

Hologres 从V1.3版本开始，将会采用全新的OSS查询引擎，相比之前版本的实例，查询性能约有40%的提升。

- 使用限制
  - 该功能仅Hologres V1.3及以上版本支持，请在Hologres管理控制台的实例详情页查看当前实例版本，如果您的实例是V1.3以下版本，请您提交工单升级实例。
  - 当前仅对OSS ORC类型的表有加速效果，暂不支持对Parquet等文件进行加速。
  - 请确保数据类型正确，否则加速效果不明显。
- 使用方式

Hologres实例升级到V1.3版本之后，您可以使用如下开关参数开启全新外表查询引擎。

```
--session级别
set hg_experimental_enable_access_hive_orc_via_holo = on;
--DB级别
alter database <dbname> set hg_experimental_enable_access_hive_orc_via_holo = on;
```

常见问题

创建外部表报错

- 问题现象

  创建DLF外部表时，提示报错 `ERROR: babysitter not ready,req:name:"HiveAccess"` 。

- 可能原因

  未添加后台配置。

- 解决方法

  请您提交工单添加后台配置。

5.2.3.4. Hologres数据导出至OSS

Hologres从V1.1版本开始，与OSS深度打通，支持将Hologres表中数据导出至OSS，快速实现Hologres数据统一存储，进一步降低存储成本。本文以一个示例为您介绍如何将Hologres数据导出至OSS。

前提条件

- 已开通Hologres并连接HoloWeb开发工具，详情请参见连接HoloWeb。
- 已开通OSS，请参见开始使用OSS。

使用限制

- 仅Hologres V1.1及以上版本支持导出数据至OSS，如果您的Hologres版本较低，请提交工单进行升级。
- Hologres支持将数据类型为BOOLEAN、INTEGER、BIGINT、FLOAT、DOUBLE PRECISION、TEXT的数据导出至OSS，其他数据类型暂不支持。
- Hologres数据导出至OSS时不支持执行 `INSERT ON CONFLICT` 、 `UPDATE` 和 `DELETE` 命令。
- Hologres数据导出至OSS时，用于给OSS写入数据的SQL语句不支持事务处理，但支持原子性。

操作步骤

将Hologres数据导出至OSS的完整步骤如下。

1. 创建extension

   Hologres数据导出至OSS是扩展功能，需要Superuser在数据库中执行以下语句创建extension。创建extension需要实例的Superuser执行，该操作针对整个数据库生效，一个数据库只需执行一次。

   ```
   CREATE EXTENSION oss_fdw;
   ```

2. 创建server

   创建extension后，执行以下语句创建server，用于连接Hologres与OSS。

   > **说明** 同一个数据库，可以创建多个server。

- 语法示例

```
CREATE SERVER IF NOT EXISTS <server_name> FOREIGN DATA WRAPPER oss_fdw OPTIONS (endpo
int 'https://<oss_internal>');
```

- 参数说明

参数	说明	示例
server_name	自定义的server名称。	oss_server
endpoint	OSS的endpoint需要以 `http://` 或者 `https://` 开头，并且使用OSS的经典网络（内网）。更多关于OSS的endpoint说明，请参见获取OSS经典网络（内网）地址。	http://oss-cn-shanghai-internal.aliyuncs.com

3. 创建用户映射

创建完server之后，使用如下语句创建用户映射，用于查询数据，需要保证创建的用户映射有对应的原数据查询权限。

> **说明** 同一个数据库，可以创建多个用户映射。

- 语法示例

```
-- 创建用户映射
CREATE USER MAPPING FOR CURRENT_USER SERVER <server_name>
OPTIONS (
  access_id '<access_id>',
  access_key '<access_key>'
);
-- 删除用户映射
DROP USER MAPPING FOR CURRENT_USER SERVER <server_name>;
```

- 参数说明

access_id和access_key两个参数，分别代表当前访问账号的AccessKey ID和AccessKey Secret，您可以单击用户中心，获取AccessKey ID和AccessKey Secret。

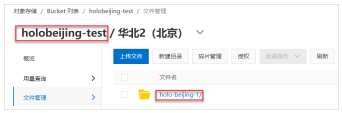
4. 创建外部表

创建完用户映射后，使用如下语句创建外部表，通过外部表将数据导出至OSS中。

- 语法示例

```
CREATE FOREIGN TABLE IF NOT EXISTS <foreign_table_name>(
  <col_name> <type>,
  ......
) SERVER <server_name>
OPTIONS(
  dir './<dir_url>/',
  format '<format_type>',
 delimiter '<delimiter_type>',
  line_delimiter <line_delimiter>,
  null_indicator <null_indicator>'
);
```

- 参数说明

参数	说明	示例
foreign_table_name	创建的外部表名称，若是需要指定schema，需要在表名前加schema名，例如 `schema.table` 。	lineitem
col_name	表中的数据列名称。	l_orderkey
type	列的数据类型。	int8
server_name	创建的server名称。	oss_server
dir	OSS表的存储URL，需要指定到目录名称。您需要注意如下事项： ■ 不支持指定到具体文件，只支持到目录。 ■ 目录名称不能指定为./oss-holotmp/holotest/region_zlib_dict.text。	dir_url：bucket名称+目录路径。例如： `./holobeijing-test/holo-beijing-1/` 。 
format	当前OSS存储的表的格式，当前仅支持 `text` 类型。	text
delimiter	TEXT类型的分隔符。	半角逗号（,）
line_delimiter	指定文件的行终止符，配置时需要用E避免转义带来的问题。	E'\n'
null_indicator	对NULL字段的处理，配置时需要用E避免转义带来的问题。	E'\\N'

5. 导出数据至OSS

   创建外部表成功后，使用如下语句您就可以将Hologres内部表中数据导出至指定OSS目录下。您可以根据业务需求选择Hologres内部表全部字段导出，或导出部分字段。

   - 语法示例

- 导出全部字段数据

```
insert into <foreign_table_name> select * from <holo_table_name>;
```

- 导出部分字段数据

```
insert into <foreign_table_name>(<col_name>,.....) select <col_name>,..... from <
holo_table_name>;
```

- 参数说明

参数	说明
foreign_table_name	创建的外部表名称。
holo_table_name	导出数据的Hologres内部表名称。
col_name	表中的数据列名称。

6. 查看导出结果

   导出成功后，您可登录OSS管理控制台，在对应OSS存储空间的目录下，查看到导出的数据文件。

使用示例

在使用示例之前，需要准备好Hologres内部表及其数据和OSS如下。

- Hologres的内部表DDL如下。

```
BEGIN;
CREATE TABLE public.holo_lineitem_100g(
 "l_orderkey" int8 NOT NULL,
 "l_linenumber" int8 NOT NULL,
 "l_suppkey" int8 NOT NULL,
 "l_partkey" int8 NOT NULL,
 "l_quantity" int8 NOT NULL,
 "l_extendedprice" int8 NOT NULL,
 "l_discount" int8 NOT NULL,
 "l_tax" int8 NOT NULL,
 "l_returnflag" text NOT NULL,
 "l_linestatus" text NOT NULL
);
COMMIT;
```

- 为Hologres内部表插入测试数据如下。

```
INSERT INTO public.holo_lineitem_100g VALUES ('8195','1333365','33366','2','14','19576','
23','22','RAIL','ic excuses? express de');
INSERT INTO public.holo_lineitem_100g VALUES ('8195','1855029','5066','3','39','38373','2
4','25','SHIP','requests cajole fluffily toward the');
```

- OSS的相关信息如下。

参数	内容
region	华北2（北京）

参数	内容
endpoint	https://oss-cn-beijing-internal.aliyuncs.com 说明 获取OSS的endpoint请参见获取OSS经典网络（内网）地址。
bucket名	holobeijing
目录名	holo-beijing-1

将Hologres内部表数据导出至OSS的完整示例语句如下。

```
CREATE EXTENSION oss_fdw;
CREATE SERVER IF NOT EXISTS oss_server_beijing FOREIGN DATA WRAPPER oss_fdw
OPTIONS(endpoint 'https://oss-cn-beijing-internal.aliyuncs.com');
CREATE USER MAPPING FOR CURRENT_USER SERVER oss_server_beijing
OPTIONS (access_id 'LTAI5XXX', access_key 'TrjgMHYYY');
BEGIN;
CREATE FOREIGN TABLE public.oss_lineitem_100g(
 "l_orderkey" int8 NOT NULL,
 "l_linenumber" int8 NOT NULL,
 "l_suppkey" int8 NOT NULL,
 "l_partkey" int8 NOT NULL,
 "l_quantity" int8 NOT NULL,
 "l_extendedprice" int8 NOT NULL,
 "l_discount" int8 NOT NULL,
 "l_tax" int8 NOT NULL,
 "l_returnflag" text NOT NULL,
 "l_linestatus" text NOT NULL
)
SERVER oss_server_beijing
OPTIONS (
  dir './holobeijing-test/holo-beijing-1/',
  format 'text',
  delimiter ',',
  line_delimiter E'\n',
  null_indicator E'\\N');
COMMIT;
INSERT INTO public.oss_lineitem_100g SELECT * FROM public.holo_lineitem_100g;
```

5.2.4. 本地文件

5.2.4.1. 使用COPY命令导入或导出本地数据

本文为您介绍如何使用COPY命令导入本地的数据至Hologres或从Hologres中导出数据至本地。

使用限制

使用COPY命令的限制说明如下：

- 当前COPY命令支持的数据类型与Hologres引擎支持的数据类型一致，详情参见数据类型汇总。
- 如果导入的是分区表数据，则Hologres只支持导入数据至分区表子表，不支持导入数据至分区表父表。
- Hologres仅支持使用 `COPY FROM STDIN` 命令导入数据和 `COPY ( query ) TO STDOUT` 命令导出数据。
- 在v1.1.43+版本中，当COPY FROM STDIN时，支持表中有DEFAULT关键字以及serial类型字段，早期版本不支持。

更多关于 `COPY` 命令的用法请参见PostgreSQL官网文档COPY。

命令格式

`COPY FROM` 命令用于从客户端的标准输入导入数据至Hologres； `COPY TO` 命令用于将Hologres数据导出至本地文件。

Hologres支持的COPY语句格式如下。

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM STDIN
    [ [ WITH ] ( option [, ...] ) ]
COPY { ( query ) }
    TO STDOUT
    [ [ WITH ] ( option [, ...] ) ]
where option can be one of:
    FORMAT format_name
    DELIMITER 'delimiter_character'
    NULL 'null_string'
    HEADER [ boolean ]
    QUOTE 'quote_character'
    ESCAPE 'escape_character'
    FORCE_QUOTE { ( column_name [, ...] ) | * }
    FORCE_NOT_NULL ( column_name [, ...] )
```

参数说明

参数说明如下表所示。

参数	描述
table_name	Hologres接收数据的表名称。
query	查询语句。
STDIN	指定从客户端使用标准输入。
STDOUT	导出至指定客户端。
FORMAT	支持*TEXT*和*CSV*格式。 默认为*TEXT*格式。
DELIMITER	指定的字段分隔符。 文本格式默认为制表符，CSV格式默认为逗号。例如 `DELIMITER AS ','` 。

使用示例

- 使用COPY命令导入本地数据
  - 使用STDIN导入数据至Hologres，命令如下。

```
--创建Hologres表。
CREATE TABLE copy_test (
  id    int,
  age   int,
  name  text
) ;
--导入数据至Hologres表。
COPY copy_test FROM STDIN WITH DELIMITER AS ',' NULL AS '';
53444,24,wangming
55444,38,ligang
55444,38,luyong
\.
--查询表中的数据。
SELECT * FROM copy_test;
```

> **说明** psql客户端支持使用STDIN导入数据，HoloStudio及HoloWeb暂不支持使用命令行方式导入数据。

  - 使用STDIN方式导入CSV格式的文件至Hologres。

```
--创建Hologres表。
CREATE TABLE partsupp ( ps_partkey          integer not null,
                        ps_suppkey     integer not null,
                        ps_availqty    integer not null,
                        ps_supplycost  float  not null,
                        ps_comment     text not null );
--导入CSV格式的文件至Hologres表。
COPY partsupp FROM STDIN WITH DELIMITER '|' CSV;
1|2|3325|771.64|final theodolites
1|25002|8076|993.49|ven ideas
\.
--查询表中的数据。
SELECT * FROM partsupp;
```

> **说明** psql客户端支持使用STDIN导入数据，HoloStudio及HoloWeb暂不支持使用命令行方式通过STDIN导入CSV格式的文件。

- 导入本地文件至Hologres，命令如下。

```
psql -U <username> -p <port> -h <endpoint> -d <databasename> -c "COPY <table> from stdi
n with delimiter '|' csv;" <<filename>;
```

> **说明** psql客户端支持使用STDIN导入数据，HoloStudio及HoloWeb暂不支持使用命令行方式通过STDIN导入本地文件。由于psql客户端仅支持STDIN（标准输入）方式导入数据，需要将文件数据转换为标准输入格式。

如下示例将指导您在终端执行命令导入本地文件至Hologres。

- 输入命令导入本地文件copy_test至Hologres。

```
C:\Users\nan>psql -U LTAI4GEnfGNu8[redacted] -p 80 -h hg[redacted].hologres.aliyuncs.com -d mydb
 -c "COPY copy_test from stdin with delimiter ',' ;" <D:\tmp\copy_test.csv
用户 LTAI4[redacted] 的口令:
COPY 3
```

其中，插入的标准文件内容如下：

```
01,01,name1
02,01,name2
03,01,name3
04,01,name4
```

- 执行完成后，回到psql客户端可以查询新插入的数据，如下图所示。

```
mydb=# select * from copy_test;
 id | age | name
----+-----+-------
  1 |   1 | 01
  1 |   1 | 01
  1 |   1 | 01
  1 |   1 | 01
  1 |   1 | name1
  2 |   1 | name2
  3 |   1 | name3
  4 |   1 | name4
```

- 使用COPY命令导出数据至本地

- 使用 `\copy` 导出Hologres的数据至本地文件。

  > **说明** 仅支持psql客户端使用该方式导出数据。

```
-- 建表
CREATE  TABLE copy_to_local (
  id    int,
  age   int,
  name  text
) ;
-- 写入数据
INSERT INTO copy_to_local VALUES
(1,1,'a'),
(1,2,'b'),
(1,3,'c'),
(1,4,'d');
-- 查数据
select * from  copy_to_local;
-- 导出数据至本地文件
\copy (select * from copy_to_local) to '/root/localfile.txt';
```

- 导入Hologres数据至本地文件。

  > **说明** 仅支持psql客户端使用该方式导出数据。

```
psql -U <username> -p <port> -h <endpoint> -d <databasename> -c "COPY (select * from <t
ablename>) to stdout with delimiter '|' csv;" ><filename>;
```

- 使用CopyManager导入导出
  - 使用CopyManager导入JDBC客户端的文件至Hologres，命令如下。

```
package com.aliyun.hologram.test.jdbc;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.*;
import java.util.Properties;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
public class jdbcCopyFile {
    public static void main(String args[]) throws Exception {
        System.out.println(copyFromFile(getConnection(), "/Users/feng/Workspace/region.
tbl", "region"));
    }
    public static Connection getConnection() throws Exception {
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://endpoint:port/dbname";
        Properties props = new Properties();
    //set db user
        props.setProperty("user", "AAA");//当前账号的AccessKey ID。
    //set db password
        props.setProperty("password", "BBB");//当前账号的AccessKey SECRET。
```

```
        return DriverManager.getConnection(url, props);
    }
    /**
     * 导入文件至数据库。
     *
     * @param connection
     * @param filePath
     * @param tableName
     * @return
     * @throws SQLException
     * @throws IOException
     */
    public static long copyFromFile(Connection connection, String filePath, String tabl
eName)
            throws SQLException, IOException {
        long count = 0;
        FileInputStream fileInputStream = null;
        try {
            CopyManager copyManager = new CopyManager((BaseConnection) connection);
            fileInputStream = new FileInputStream(filePath);
            count = copyManager.copyIn("COPY " + tableName + " FROM STDIN delimiter '|'
csv", fileInputStream);
        } finally {
            if (fileInputStream != null) {
                try {
                    fileInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return count;
    }
}
```

- 使用CopyManager导出Hologres的数据至JDBC客户端的文件，命令如下。

```
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
public class copy_to_local_file {
    public static void main(String args[]) throws Exception {
        System.out.println(copyToFile(getConnection(), "/Users/feng/Workspace/region.tb
l", "select * from region"));
    }
    public static Connection getConnection() throws Exception {
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://endpoint:port/dbname";
        Properties props = new Properties();
    //set db user
```

```
        props.setProperty("user", "AAA");//当前账号的AccessKey ID。
    //set db password
        props.setProperty("password", "BBB");//当前账号的AccessKey SECRET。
        return DriverManager.getConnection(url, props);
    }
    /**
     * 导入文件至数据库。
     *
     * @param connection
     * @param filePath
     * @param SQL_Query
     * @return
     * @throws SQLException
     * @throws IOException
     */
    public static String copyToFile(Connection connection, String filePath, String SQL_
Query)
            throws SQLException, IOException {
        FileOutputStream fileOutputStream = null;
        try {
            CopyManager copyManager = new CopyManager((BaseConnection)connection);
            fileOutputStream = new FileOutputStream(filePath);
            copyManager.copyOut("COPY " + "(" + SQL_Query + ")" + " TO STDOUT DELIMITER
'|' csv ", fileOutputStream);
        } finally {
            if (fileOutputStream != null) {
                try {
                    fileOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return filePath;
    }
}
```

HoloWeb可视化一键导入

HoloWeb支持可视化一键上传本地文件，具体操作如下。

1. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，单击**数据方案**。
3. 在左侧导航栏，选择**一键本地文件导入** > **新建数据导入**。
4. 配置**一键本地文件上传**对话框中**选择目标表**页面的各项参数。

参数	描述
作业名称	新建的作业名称。
实例名	选择已登录的实例名称。
目标库	Hologres对应实例中已创建的数据库名称。
目标Schema	Hologres中已创建的Schema名称。 如果您没有新建Schema，则只能选择默认创建的**public**。如果有新建的Schema，您也可以选择新建的Schema。
选择要导入的数据表	用于存储本地文件的表名称。 导入本地文件前，您需要在目标数据库中创建一张用于存储本地文件的表。

5. 单击**下一步**，配置**选择数据源表**页面的各项参数。

参数	描述
选择文件	需要上传的本地文件。 仅支持上传TXT、CSV和LOG类型的文件。 **说明** 数据文件列的顺序必须和表定义的列顺序一致，且列的个数相同。

参数	描述
选择分隔符	○ 逗号 ○ Tab ○ 分号 ○ 空格 ○ | ○ # ○ & 您也可以自定义分隔符。
原始字符集	○ GBK ○ UTF-8 ○ CP936 ○ ISO-8859
首行为标题	勾选则设置首行数据为标题。

6. 单击**下一步**，单击**导入总览**页面的**执行**，完成本地文件一键导入。

   在**导入总览**页面您可以查看本地文件导入的Schema、数据库、表等信息。

5.2.5. Hologres

5.2.5.1. 跨库查询

本文为您介绍如何在Hologres中使用跨库查询数据以及相关的使用示例。

背景信息

Hologres从V1.1版本开始，支持在不同地域、不同实例和不同数据库之间，通过创建外部表的方式查询数据，操作方便简单。Hologres兼容Postgres，通过外部表的方式跨库查询原理同Postgres，详情请参见FDW。

使用限制

- 仅Hologres V1.1及以上版本支持跨库查询数据，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅支持跨库查询Hologres V1.1及以上版本的实例，例如您的Hologres实例是V1.1版本，想要跨库查询V0.10版本实例中的数据，执行会报错，需要将跨库查询的实例升级至V1.1及以上版本。

- 仅支持跨库查询Hologres内部表中的数据，不支持查询Hologres外部表和Hologres的视图View。
- 仅支持跨库查询Hologres分区父表，不支持查询Hologres分区子表。

操作步骤

1. 创建extension

   在使用之前，需要Superuser在数据库中执行以下语句创建extension。创建extension需要实例的Superuser执行，该操作针对整个数据库生效，一个数据库只需执行一次。

   ```
   --创建extension
   CREATE EXTENSION hologres_fdw;
   ```

   > **说明** 如需卸载extension请执行如下命令。
   >
   > ```
   > DROP EXTENSION hologres_fdw;
   > ```

2. 创建server

   创建extension成功后，执行以下语句创建server，用于连接跨库查询的实例。

   > **说明** 同一个数据库，可以创建多个server。

   ```
   CREATE SERVER <server_name> FOREIGN DATA WRAPPER hologres_fdw OPTIONS (
       host '<endpoint>',
       port '<port>',
       dbname '<dbname>'
   );
   ```

参数	说明	示例
server_name	server的名称，自定义设置。	holo_fdw_server
host	Hologres实例的经典网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取经典网络（内网）地址。	hgpostcn-cn-xxx-cn-hangzhou-internal.hologres.aliyuncs.com
port	Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。	80
dbname	需要跨库查询的源数据库名称。	testdb

3. 创建用户映射

   创建完server之后，使用如下语句创建用户映射，用于查询数据，需要保证创建的用户映射有对应的原数据查询权限。

   > **说明** 同一个数据库，可以创建多个用户映射。

```
CREATE USER MAPPING FOR <账号uid> SERVER <server_name>
OPTIONS (
  access_id '<access_id>',
  access_key '<access_key>'
);
```

参数	说明
server_name	server的名称，上一步骤自定义设置的名称。
access_id	当前访问账号的AccessKey ID，您可以单击用户中心，获取AccessKey ID。
access_key	当前访问账号的AccessKey Secret，您可以单击用户中心，获取AccessKey Secret。

使用示例

```
--为当前用户创建用户映射
create user mapping for current_user server holo_fdw_server options
(
    access_id 'LTAI5txxx', access_key 'y8LUUyyy'
);
--为RAM用户123xxx创建用户映射
create user mapping for "p4_123xxx" server holo_fdw_server options
(
    access_id 'LIlY5txxx', access_key 'KsjkXKyyy'
);
--删除用户映射
Drop USER MAPPING for CURRENT_USER server holo_fdw_server;
Drop USER MAPPING for "p4_123xxx" server holo_fdw_server;
```

4. 创建外部表

   创建外部表有两种方式，分别如下。

   - （推荐）使用 `IMPORT FOREIGN SCHEMA` 语句创建外部表

     使用 `IMPORT FOREIGN SCHEMA` 语句创建外部表，操作更加简单方便，SQL语句如下。

     ```
     IMPORT FOREIGN SCHEMA <holo_remote_schema>
     [{ LIMIT TO EXCEPT }| (remote_table [, ...])]
     FROM SERVER <server_name>
     INTO <holo_local_schema>
     [ OPTIONS ( OPTION 'values' [, ...])];
     ```

     > **说明** 导入外表元数据需要读取较多外部数据库元数据，建议通过LIMIT TO参数限制只导入需要的表，避免整库导入，可以保障外表创建效率。

参数	说明	示例
holo_remote_schema	要跨库查询的源表所在的schema名称。	remote

参数	说明	示例
remote_table	要跨库查询的源表名称，外部表创建成功，将会在新的数据库创建与源表同名的外部表。	lineitem
server_name	创建的server的名称。	holo_fdw_server
holo_local_schema	创建的外部表所在的schema名称。	local
OPTION 'values'	创建外部表时的冲突策略，参数包含如下内容。 ■ import_collate：列是否包括collate配置，默认为*true*。 ■ import_default：列是否包括default值，默认为*false*。 ■ import_not_null：列是否包含not null约束，默认为*true*。	import_not_null 'true'

- 使用 `CREATE FOREIGN TABLE` 语句创建外部表

  SQL语句如下。

```
CREATE FOREIGN TABLE <local_table> (
 col_name type,
  ......
)SERVER <server_name>
OPTIONS (schema_name '<remote_schema_name>', table_name '<remote_table>');
```

参数	说明	示例
local_table	创建的外部表名称。默认表放在public schema，若是有自定义schema，需要在表名前加schema名，为 `schema.table` 。	public.lineitem
server_name	创建的server的名称。	holo_fdw_server
remote_schema_name	要跨库查询的源表所在的schema名称。	public
remote_table	要跨库查询的源表名称。	holo_lineitem

5. 查询外部表数据

   外部表创建完成后，可以直接查询外部表数据实现跨库查询，SQL语句如下。

```
select * from <holo_local_table> limit 10;
```

6. 数据导入内部表

   若是想要实现跨库、跨实例导入数据，或者外部表查询性能不满足预期，可以通过如下SQL语句，将数据导入至Hologres内部表。

> 说明 在使用之前，需要先创建一张内部表用于接收数据，创建内部表请参见表。

```
insert into <holo_table> select * from <holo_local_table>;
```

使用示例

本小节为您介绍跨库查询数据使用示例前的预置配置以及三个相关的完整示例。

- 预置配置

  在示例之前，需要在Hologres里面已准备好一个实例，以及创建好数据库，以及准备好相关的内部表数据，具体内容如下。

  - 实例相关配置

配置	说明
源Hologres实例ID	hgpostcn-cn-i7mxxxxxxxxx
源Hologres数据库名称	remote_db
源Hologres实例schema名称	remote
源Hologres内部表名称	lineitem
源Hologres分区父表名称	holo_dwd_product_movie_basic_info

  - 源Hologres内部表DDL

```
BEGIN;
CREATE SCHEMA remote;
CREATE TABLE "remote"."lineitem" (
 "l_orderkey" int8 NOT NULL,
 "l_linenumber" int8 NOT NULL,
 "l_suppkey" int8 NOT NULL,
 "l_partkey" int8 NOT NULL,
 "l_quantity" int8 NOT NULL,
 "l_extendedprice" int8 NOT NULL,
 "l_discount" int8 NOT NULL,
 "l_tax" int8 NOT NULL,
 "l_returnflag" text NOT NULL,
 "l_linestatus" text NOT NULL,
 "l_shipdate" timestamptz NOT NULL,
 "l_commitdate" timestamptz NOT NULL,
 "l_receiptdate" timestamptz NOT NULL,
 "l_shipinstruct" text NOT NULL,
 "l_shipmode" text NOT NULL,
 "l_comment" text NOT NULL
);
COMMIT;
```

- 源Hologres分区表DDL

```
-- 分区父表
BEGIN;
CREATE TABLE "remote"."holo_dwd_product_movie_basic_info" (
 "movie_name" text,
 "director" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
)
PARTITION BY LIST (ds);
comment on column "remote"."holo_dwd_product_movie_basic_info"."movie_name" is '电影名称
';
comment on column "remote"."holo_dwd_product_movie_basic_info"."director" is '导演';
comment on column "remote"."holo_dwd_product_movie_basic_info"."scriptwriter" is '编剧'
;
comment on column "remote"."holo_dwd_product_movie_basic_info"."area" is '制片地区/国家'
;
comment on column "remote"."holo_dwd_product_movie_basic_info"."actors" is '主演';
comment on column "remote"."holo_dwd_product_movie_basic_info"."type" is '类型';
comment on column "remote"."holo_dwd_product_movie_basic_info"."movie_length" is '电影长
度';
comment on column "remote"."holo_dwd_product_movie_basic_info"."movie_date" is '上映日期
';
comment on column "remote"."holo_dwd_product_movie_basic_info"."movie_language" is '语
言';
comment on column "remote"."holo_dwd_product_movie_basic_info"."imdb_url" is 'imdb号';
COMMIT;
--创建20170122为分区的分区子表
CREATE TABLE IF NOT EXISTS "remote".holo_dwd_product_movie_basic_info_20170122 PARTITIO
N OF "remote".holo_dwd_product_movie_basic_info FOR VALUES IN ('20170122');
```

- 示例1：跨库查询非分区表

> **说明** 以下所有代码实例，需要在跨库查询的数据库中执行。

```
-- superuser创建extension
CREATE EXTENSION hologres_fdw;
--superuser创建server
CREATE SERVER holo_fdw_server FOREIGN DATA WRAPPER hologres_fdw OPTIONS (
    host 'hgpostcn-cn-i7mxxxxxxxxx-cn-hangzhou-internal.hologres.aliyuncs.com',
    port '80',
    dbname 'remote_db'
);
-- 为当前用户创建授权映射
CREATE USER MAPPING FOR CURRENT_USER SERVER holo_fdw_server
OPTIONS (access_id 'LTAIxxxxxx', access_key 'Trjgyyyyyyy');
--创建schema（使用fdw功能的实例，local这个schema是非必需创建，可以换成业务的schema）
CREATE SCHEMA local;
-- 创建外部表
IMPORT FOREIGN SCHEMA remote
LIMIT to (lineitem)
FROM SERVER holo_fdw_server
INTO local
OPTIONS (
  import_not_null 'true'
);
SELECT * FROM local.lineitem limit 10;
```

- 示例2：跨库查询分区表

```
CREATE EXTENSION hologres_fdw;
CREATE SERVER holo_fdw_server FOREIGN DATA WRAPPER hologres_fdw OPTIONS (
    host 'hgpostcn-cn-i7mxxxxxxxxx-cn-hangzhou-internal.hologres.aliyuncs.com',
    port '80',
    dbname 'remote_db'
);
-- 为当前用户创建授权映射
CREATE USER MAPPING FOR CURRENT_USER SERVER holo_fdw_server
OPTIONS (access_id 'LTAIxxxxxx', access_key 'Trjgyyyyyyy');
-- 创建schema（使用fdw功能的实例，local这个schema是非必需创建，可以换成业务的schema）
CREATE SCHEMA local;
-- 切换至本地实例（使用fdw功能的实例）
IMPORT FOREIGN SCHEMA remote
LIMIT to (holo_dwd_product_movie_basic_info)
FROM SERVER holo_fdw_server
INTO local
OPTIONS (
  import_not_null 'true'
);
-- 直接查全表数据
SELECT * FROM local.holo_dwd_product_movie_basic_info limit 10;
```

- 示例3：将外部表数据导入内部表

```
-- 创建schema（使用fdw功能的实例，local这个schema是非必需创建，可以换成业务的schema）
CREATE SCHEMA local;
-- 创建内部表
BEGIN;
CREATE TABLE "local"."dwd_product_movie_basic_info" (
 "movie_name" text,
 "director" text,
 "scriptwriter" text,
 "area" text,
 "actors" text,
 "type" text,
 "movie_length" text,
 "movie_date" text,
 "movie_language" text,
 "imdb_url" text,
 "ds" text
);
COMMIT;
--导入内部表
insert into local.dwd_product_movie_basic_info select * from local.holo_dwd_product_movie
_basic_info;
```

5.3. 实时写入

5.3.1. Flink

5.3.1.1. 概览

实时计算Flink版（Alibaba Cloud Realtime Compute for Apache Flink，Powered by Ververica）是阿里云基于Apache Flink构建的企业级高性能的实时大数据处理系统。Hologres与Flink深度连通，支持实时写入Flink的数据，可以实时查询写入的数据，帮助您快速搭建实时数仓。

阿里云实时计算Flink版不进行业务存储，所有的数据均来自于外部存储系统持有的数据。阿里云实时计算Flink版支持的数据存储类型如下：

- 源表

  源表指输入至Flink的数据输入源。Flink的源表指定为Hologres时，使用的是批量导入而非流式导入，Hologres会将全表的数据统一扫描一次后再输出至下游，扫描完成后本次作业结束。

- 维表

  维表一般适用于点查询场景（Lookup by Key），因此在Hologres中，维表建议使用行存储，并且JOIN的字段必须是完整的主键字段。

- 结果表

  结果表用于接收并存放经过Flink计算的结果数据，为下游数据继续消费提供各类读写接口。

阿里云实时计算Flink版还支持很多企业级高级能力，通过与Hologres深度集成，提供以下创新能力：

- Hologres Binlog消费

  使用消息队列的模式消费Hologres表的Change Log。

- Flink Catalog

  Flink支持导入Hologres元数据为Catalog，在Flink全托管控制台直接读取Hologres元数据，不用再手动注册Hologres表，可以提高开发效率且能保证表结构的正确性。

- Schema Evolution

  Flink全托管支持Schema Evolution，在Flink读取JSON数据时，可以自动解析类型，自动创建对应表的列，支持数据模型的动态演化。

Hologres支持的Flink产品形态及功能如下表所示。

产品形态	数据存储类型			企业级高级能力			描述
	源表	结果表	维表	Hologres Binlog消费	Flink Catalog	Schema Evolution	
Flink全托管	支持行存储及列存储	支持行存储及列存储	建议使用行存储	支持	支持	支持	使用VVP开发平台。
Flink半托管	支持行存储及列存储	支持行存储及列存储	建议使用行存储	支持	支持	支持	使用EMR Studio开发平台。
Blink独享（已停售）	支持行存储及列存储	支持行存储及列存储	建议使用行存储	Hologres V0.8版本只支持行存储，V0.9及以上版本支持行存储及列存储。建议使用行存储。	不支持	不支持	使用Bayes开发平台。 推荐使用阿里云Flink全托管。
开源Flink1.10	不支持	支持行存储及列存储	不支持	不支持	不支持	不支持	-
开源Flink1.11及以上版本	不支持	支持行存储及列存储	建议使用行存储	不支持	不支持	不支持	从开源Flink1.11版本开始，Hologres代码已开源。详细内容请参见GitHub。

5.3.1.2. Flink全托管

5.3.1.2.1. Hologres源表

Flink全托管产品（Flink Serverless）是基于Apache Flink构建的全托管产品，为您提供全托管的实时计算服务。本文为您介绍Flink全托管如何创建Hologres源表。

使用说明

Hologres源表默认使用批模式读取数据，即对全表数据仅扫描一次。扫描结束，消费即结束，扫描结束后输入的数据将不会被Hologres源表读取。

DDL定义

- DDL语句

  创建Hologres源表的DDL语句如下。

```
create table hologres_source(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>', --Hologres的数据库名称。
  'tablename'='<yourTablename>',  --Hologres用于接收数据的表名称。
  'username'='<yourAccessID>',  --当前阿里云账号的AccessKey ID。
  'password'='<yourAccessSecret>',  --当前阿里云账号的AccessKey Secret。
  'endpoint'='<vpc_ip:vpc_port>',  --当前Hologres实例VPC网络的Endpoint。
  'field_delimiter'='|' --该参数为可选参数。
);
```

> **说明** Flink全托管不支持在源表中定义计算列。

- 参数说明

  With参数的描述如下表所示。

参数	描述	是否必填
connector	源表类型。 固定值为*hologres*。	是
dbname	Hologres的数据库名称。	是
tablename	Hologres用于接收数据的表名称。	是
username	当前阿里云账号的AccessKey ID。 您可以登录AccessKey 管理，获取AccessKey ID。	是
password	当前阿里云账号的AccessKey Secret。 您可以登录AccessKey 管理，获取AccessKey Secret。	是

参数	描述	是否必填
endpoint	Hologres的VPC网络地址。 您可以登录Hologres管控台，进入目标实例的详情页，在**实例配置**中获取Endpoint。Endpoint需包含端口号，格式为ip:port。 **说明** 如果Flink与Hologres实例部署在同一个地域，请使用VPC网络的网络地址。如果在不同地域，请使用公共网络的网络地址，并确保Flink集群能正常访问公网（公网网络延迟较高）。	是
connectionSize	表示单个Flink维表Task所创建的JDBC连接池大小。 默认值为*3*，和吞吐成正比。	否
connectionPoolName	连接池名称，同一个TaskManager中，表配置同名的连接池名称可以共享连接池。 无默认值，每个表默认使用自己的连接池。如果设置连接池名称，则所有表的connectionSize需要相同，需要实时计算版本大于1.13-vvr-4.1.21.13。	否
jdbcScanFetchSize	扫描时攒批大小。 默认值为256。	否
jdbcScanTimeoutSeconds	扫描操作超时时间。 默认值为*60s*。	否
field_delimiter	导出数据时，不同行之间使用的分隔符。 默认值为*"\u0002"*。	否
binlog	是否为Binlog source，默认为false。如果需要消费，需要将binlog设置为true。	否
binlogMaxRetryTimes	读取Binlog出错重试次数，默认为60次。	否
binlogRetryIntervalMs	读取Binlog出错重试间隔，默认为2000ms。	否
binlogBatchReadSize	读取Binlog批量大小，默认为16个。	否
cdcMode	读取Binlog时是否采用CDC模式，默认为false。详情请参见Flink实时消费Binlog。	否
startTime	启动位点的时间，格式为yyyy-MM-dd hh:mm:ss。如果没有设置该参数，且作业没有从状态恢复，则从最早的Binlog开始消费Hologres数据。	否

> **说明** `jdbc` 开头的参数在Flink引擎1.13-vvr-4.0.11版本开始支持，其他JDBC参数请参见Hologres结果表。

- 使用示例

  创建Hologres的源表并导入Flink的数据，示例SQL语句如下。

```
CREATE TEMPORARY TABLE hologres_source (
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'field_delimiter'='|'  --该参数可选。
);
CREATE TEMPORARY TABLE blackhole_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='blackhole'
);
INSERT INTO blackhole_sink
SELECT
   name, age, birthday
from hologres_source;
```

Flink实时消费Binlog

Hologres Connector支持实时消费Binlog，详情请参见Flink实时消费Binlog。

Hologres Catalog

Flink全托管支持Hologres Catalog，在Flink全托管控制台直接读取Hologres元数据，不用再手动注册Hologres表，可以提高作业开发的效率且能保证表结构的正确性，详情请参见管理Hologres Catalog。

基于Hologres Catalog，目前全托管的Flink已经支持模型演进（schema evolution）以及整库同步能力，详情请参见CREATE TABLE AS（CTAS）语句和CREATE DATABASE AS（CDAS）语句。

Hologres DataStream Connector

如果您通过DataStream的方式读写Hologres数据，则需要使用Hologres DataStream Connector连接Flink全托管，详情请参见Hologres DataStream Connector。

数据类型映射

Flink全托管与Hologres的数据类型映射，请参见数据类型汇总。

5.3.1.2.2. Hologres维表

Flink全托管产品（Flink Serverless）是基于Apache Flink构建的全托管产品，为您提供全托管的实时计算服务。本文为您介绍在Flink全托管中如何使用Hologres维表。

使用限制

1. 建议使用Hologres的行存表或者行列共存表，列存表对于点查场景性能开销较大。

   创建行存储表时必须设置主键，并且将主键配置为*clustering key*时性能较好，示例语句如下。

   ```
   begin;
   create table test(a int primary key, b text, c text, d float8, e int8);
   call set_table_property('test', 'orientation', 'row');
   call set_table_property('test', 'clustering_key', 'a');
   commit;
   ```

2. 表的主键必须是Flink Join ON的字段，Flink Join ON的字段也必须是表的完整主键，两者必须完全匹配。

DDL定义

创建Hologres维表的DDL语句如下。

```
CREATE TABLE hologres_dim(
 id INT,
 len INT,
 content VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',  --Hologres的数据库名称。
  'tablename'='<yourTablename>',  --Hologres用于接收数据的表名称。
  'username'='<yourUsername>',  --当前阿里云账号的AccessKey ID。
  'password'='<yourPassword>',  --当前阿里云账号的AccessKey Secret。
  'endpoint'='<yourEndpoint>'  --当前Hologres实例VPC网络的Endpoint。
);
```

With参数的描述如下表所示。

参数	描述	是否必填
connector	维表类型。 固定值为*hologres*。	是
dbname	Hologres的数据库名称。	是
tablename	Hologres用于接收数据的表名称。	是
username	当前阿里云账号的AccessKey ID。 您可以登录AccessKey 管理，获取AccessKey ID。	是

参数	描述	是否必填
password	当前阿里云账号的AccessKey Secret。 您可以登录AccessKey 管理，获取AccessKey Secret。	否
endpoint	Hologres的VPC网络地址。 您可以登录Hologres管控台，进入目标实例的详情页，在**实例配置**中获取Endpoint。Endpoint需包含端口号，格式为ip:port。 **说明** 如果Flink与Hologres实例部署在同一个地域，请使用VPC网络的网络地址。如果在不同地域，请使用公共网络的网络地址，并确保Flink集群能正常访问公网（公网网络延迟较高）。	是
useRpcMode	Hologres Connector默认使用JDBC实现，可以通过该选项切换至老版本的Connector实现方式，即RPC模式。未来版本将不再支持RPC模式，因此不推荐使用该参数。 默认值为*false*。	否
connectionSize	表示单个Flink维表Task所创建的JDBC连接池大小。 默认值为*3*，和吞吐成正比。	否
connectionPoolName	连接池名称，同一个TaskManager中，表配置同名的连接池名称可以共享连接池。 无默认值，每个表默认使用自己的连接池。如果设置连接池名称，则所有表的connectionSize需要相同，需要实时计算版本大于1.13-vvr-4.1.2。	否

参数	描述	是否必填
async	表示是否采用异步方式同步数据。 异步模式可以并发地处理多个请求和响应，从而连续的请求之间不需要阻塞等待，提高查询的吞吐。但在异步模式下，无法保证请求的绝对顺序。 取值如下 • *true*：表示异步同步数据。 • *false*：默认值，表示不进行异步同步数据 **说明** 关异步请求的原理请参见维表 JOIN 与异步优化	否
jdbcReadBatchSize	表示维表点查最大批次大小。 默认值为*128*	否
jdbcReadBatchQueueSize	表示维表点查请求缓冲队列大小。 默认值为*256*	否

说明 `jdbc` 开头的参数在Flink引擎1.13-vvr-4.0.11版本开始支持，其他JDBC参数请参见Hologres结果表。

Cache参数

如果Hologres维表包含Cache参数，则可以参考如下参数描述。

参数	描述	是否必填
cache	缓存策略。 Hologres仅支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果未找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cachesize）和缓存更新时间间隔（cachettlms）。	否

参数	描述	是否必填
cachesize	缓存大小。 选择 `LRU` 缓存策略后，可以设置缓存大小，默认值为*10000*行。	否
cachettlms	更新缓存的时间间隔，单位为毫秒。 当选择 `LRU` 缓存策略后，可以设置缓存失效的超时时间，默认不过期。	否
cacheempty	是否缓存join结果为空的数据。 取值如下： • true（默认值）：表示缓存join结果为空的数据。 • false：表示不缓存join结果为空的数据。	否

Hologres Catalog

Flink全托管支持Hologres Catalog，在Flink全托管控制台直接读取Hologres元数据，不用再手动注册Hologres表，可以提高作业开发的效率且能保证表结构的正确性，详情请参见管理Hologres Catalog。

基于Hologres Catalog，目前全托管的Flink已经支持模型演进（schema evolution）以及整库同步能力，详情请参见CREATE TABLE AS（CTAS）语句和CREATE DATABASE AS（CDAS）语句。

使用示例

创建Hologres维表并接收Flink的数据，示例语句如下。

```
CREATE TEMPORARY TABLE datagen_source (
   a INT,
   b BIGINT,
   c STRING,
   proctime AS PROCTIME()
) with (
   'connector' = 'datagen'
);
CREATE TEMPORARY TABLE hologres_dim (
   a INT,
   b VARCHAR,
   c VARCHAR
) with (
   'connector' = 'hologres',
   ...
);
CREATE TEMPORARY TABLE blackhole_sink (
   a INT,
   b STRING
) with (
   'connector' = 'blackhole'
);
insert into blackhole_sink select T.a,H.b
FROM datagen_source AS T JOIN hologres_dim FOR SYSTEM_TIME AS OF T.proctime AS H ON T.a = H
.a;
```

Hologres DataStream Connector

如果您通过DataStream的方式读写Hologres数据，则需要使用Hologres DataStream Connector连接Flink全托管，详情请参见Hologres DataStream Connector。

类型映射

Flink全托管与Hologres的数据类型映射，请参见数据类型汇总。

5.3.1.2.3. Hologres结果表

Flink全托管产品（Flink Serverless）是基于Apache Flink构建的全托管产品，为您提供全托管的实时计算服务。本文为您介绍Flink全托管如何实时写入数据至Hologres结果表。

DDL定义

创建Hologres结果表的DDL语句如下。

```
create table hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',  --Hologres的数据库名称。
  'tablename'='<yourTablename>',  --Hologres用于接收数据的表名称。
  'username'='<yourUsername>',  --当前阿里云账号的AccessKey ID。
  'password'='<yourPassword>',  --当前阿里云账号的AccessKey Secret。
  'endpoint'='<yourEndpoint>',  --当前Hologres实例VPC网络的Endpoint。
);
```

> **说明** Flink全托管不支持在结果表中定义计算列。

With参数的描述如下表所示。

参数	描述	是否必填
connector	结果表类型。 固定值为*hologres*。	是
dbname	Hologres的数据库名称。	是
tablename	Hologres用于接收数据的表名称。	是
username	当前阿里云账号的AccessKey ID。 您可以登录AccessKey 管理，获取AccessKey ID。	是
password	当前阿里云账号的AccessKey Secret。 您可以登录AccessKey 管理，获取AccessKey Secret。	是
endpoint	Hologres的VPC网络地址。 您可以登录Hologres管控台，进入目标实例的详情页，在**实例配置**中获取Endpoint。Endpoint需包含端口号，格式为ip:port。 > **说明** 如果Flink与Hologres实例部署在同一个地域，请使用VPC网络的网络地址。如果在不同地域，请使用公共网络的网络地址，并确保Flink集群能正常访问公网（公网网络延迟较高）。	是

参数	描述	是否必填
field_delimiter	Hologres Sink支持将一个STRING字段按照field_delimiter切分为数组导入Hologres。 默认值为*"\u0002"*。	否
mutatetype	数据写入模式，详情请参见流式语义。 默认值为*insertorignore*。	否
ignoredelete	是否忽略回撤消息。取值如下： • *false*，不忽略。 • *true*，忽略。 通常Flink的Group By会产生回撤消息，回撤消息传输到Hologres Connector时会产生Delete请求。 默认值为*false*。 **说明** 仅在使用流式语义时生效。	否
partitionrouter	是否写入分区表。取值如下： • *false*，不写入。 • *true*，写入。 默认值为*false*。	否
createparttable	当写入分区表时，是否根据分区值自动创建分区表。仅Flink全托管2.1.X及以上版本支持自动创建分区表。取值如下： • *false*，不自动创建。 • *true*，自动创建。 默认值为*false*。 **注意** 分区值中不能含有减号。请您谨慎使用该功能，确保分区值不会出现脏数据，导致创建了错误的分区表。	否
ignoreNullWhenUpdate	当`mutatetype='insertOrUpdate'`时，是否忽略更新写入数据中的Null值。 默认值为*false*。	否

参数	描述	是否必填
useRpcMode	Hologres Connector默认使用JDBC实现，可以通过该选项切换至老版本的Connector的RPC模式。 默认值为*false*。	否
connectionSize	表示单个Flink结果表Task所创建的JDBC连接池大小。 默认值为*3*，和吞吐成正比。	否
connectionPoolName	连接池名称，同一个TaskManager中，表配置同名的连接池名称可以共享连接池。 无默认值，每个表默认使用自己的连接池。如果设置连接池名称，则所有表的connectionSize需要相同，需要实时计算版本大于1.13，Flink（VVR）版本大于4.1.2。	否
jdbcWriteBatchSize	表示Hologres Sink节点数据攒批的最大批大小。 默认值为*256*。	否
jdbcWriteBatchByteSize	表示Hologres Sink节点数据攒批的最大字节大小。 默认值为*20971520（2 * 1024 * 1024）*，即2MB。	否
jdbcWriteFlushInterval	表示Hologres Sink节点数据攒批的最长Flush等待时间。 默认值为 *10000*，即10秒	否
jdbcReWriteBatchedDeletes	表示将多条delete请求合并为一条SQL语句以提升性能。 默认值为*true*	否
jdbcRewriteSqlMaxBatchSize	表示单条SQL进行INSERT/DELETE操作的最大批次大小，比如写入操作，所攒的批会通过writeBatchSize/rewriteSqlMaxBatchSize条INSERT语句完成插入。 默认值为 *1024*	否

参数	描述	是否必填
jdbcEnableDefaultForNotNullColumn	设置为true时，not null且未在表上设置default的字段传入null时，将以默认值写入。 默认值为*true* String 默认为 ""，Number 默认为*0*，Date/timestamp/timestamptz 默认为*1970-01-01 00:00:00*。	否

说明 jdbcWriteBatchSize和jdbcWriteFlushInterval之间为或的关系，即当同时设置了两个参数，满足其中一个，就进行写入。

更多JDBC相关的参数

参数	描述	是否必填
jdbcRetryCount	表示当连接故障时，写入和查询的重试次数。 默认值为*10*。	否
jdbcRetrySleepInitMs	表示每次重试的等待时间，公式为*retrySleepInitMs + retry * retrySleepStepMs*。 默认值为*1000ms*。	否
jdbcRetrySleepStepMs	表示每次重试的等待时间，公式为*retrySleepInitMs + retry * retrySleepStepMs*。 默认值为*5000ms*。	否
jdbcConnectionMaxIdleMs	表示写入线程和点查线程数据库连接的最大Idle时间，超过连接将被释放。 默认值为*60000*，即60秒。	否
jdbcMetaCacheTTL	表示元信息（TableSchema）的本地缓存时间。 默认值为*60000*，即60秒。	否
jdbcMetaAutoRefreshFactor	表示当元信息缓存剩余存活时间短于metaCacheTTL/metaAutoRefreshFactor 将自动刷新缓存。 默认值为*-1*，表示不自动刷新。	否

> **说明** 以jdbc开头的相关参数自1.13-vvr-4.0.11版本开始支持。

流式语义

流处理，也称为流数据或流事件处理，即对一系列无界数据或事件连续处理。执行流数据或流事件处理的系统通常允许您指定一种可靠性模式或处理语义，保证整个系统处理数据的准确性，因为网络或设备故障等可能会导致数据丢失。

根据Hologres Sink的配置和Hologres表的属性，流式语义分为以下两种：

- Exactly-once（仅一次）：即使在发生各种故障的情况下，系统只处理一次数据或事件。
- At-least-once（至少一次）：如果在系统完全处理之前丢失了数据或事件，则从源头重新传输，因此可以多次处理数据或事件。如果第一次重试成功，则不进行后续重试。

在Hologres结果表中使用流式语义，您需要注意以下几点：

- 如果Hologres物理表未设置主键，则Hologres Sink使用At-least-once语义。
- 如果Hologres物理表已设置主键，则Hologres Sink通过主键确保使用Exactly-once语义。当同主键数据出现多次时，您需要设置mutatetype参数确定更新结果表的方式，mutatetype取值如下：
  - insertorignore（默认值）：保留首次出现的数据，忽略后续所有数据。
  - insertorreplace：整行替换已有数据。
  - insertorupdate：替换部分已有数据。例如一张表有a、b、c和d四个字段，a是主键PK（Primary Key），写入Hologres时只写入a和b两个字段，在主键重复的情况下，系统只会更新b字段，c和d保持不变。

  > **说明**
  > - 当mutatetype设置为insertorupdate或insertorreplace时，系统根据主键更新数据。
  > - Flink全托管定义的结果表中的数据列数不一定要和Hologres物理表的列数一致，您需要保证缺失的列没有非空约束，即列值可以为Null，否则会报错。

- 默认情况下，Hologres Sink只能向一张表导入数据。如果导入数据至分区表的父表，即使导入成功，也会查询数据失败。您可以设置参数partitionrouter为*true*，开启自动将数据路由到对应分区表的功能。注意事项如下：
  - tablename参数需要填写为分区表父表的表名称。
  - 如果没有提前创建分区表，需要设置createparttable参数为*true*，从而支持自动创建分区表，否则会导入失败。

如何使用宽表Merge/局部更新功能

对于常见的多个流的数据写入至一张Hologres宽表的场景，具体使用方法如下：

假设Hologres有一张宽表**WIDE_TABLE**，有A、B、C、D、E几列，其中A字段是主键，Flink一个流包含数据A、B、C，另一个流包含数据A、D、E。

1. 使用Flink SQL声明两张Hologres结果表，其中一张表只声明字段A、B、C，另一张表只声明字段A、D、E，这两张表都映射至**WIDE_TABLE**。
2. 两张结果表的mutatetype属性都设置成insertorupdate。
3. 两张结果表的ignoredelete属性都设置成true，防止回撤消息产生Delete请求。
4. 将两个流的数据分别Insert至两张结果表中。

该场景的具体使用限制如下：

1. 宽表必须有主键。
2. 每个流的数据都必须包含完整主键字段。
3. 列存表的宽表Merge场景在高RPS的情况下，CPU使用率会偏高，建议关闭表中字段的Dictionary encoding。

Hologres Catalog

Flink全托管支持Hologres Catalog，在Flink全托管控制台直接读取Hologres元数据，不用再手动注册Hologres表，可以提高作业开发的效率且能保证表结构的正确性，详情请参见管理Hologres Catalog。

基于Hologres Catalog，目前全托管的Flink已经支持模型演进（schema evolution）以及整库同步能力，详情请参见CREATE TABLE AS（CTAS）语句和CREATE DATABASE AS（CDAS）语句。

Hologres DataStream Connector

如果您通过DataStream的方式读写Hologres数据，则需要使用Hologres DataStream Connector连接Flink全托管，详情请参见Hologres DataStream Connector。

类型映射

Flink全托管与Hologres的数据类型映射，请参见数据类型汇总。

5.3.1.3. Blink独享

5.3.1.3.1. 实时数据API

本文为您介绍什么是实时数据API，以及实时数据API的作用。

- 实时数据业务流程。

  在实时数据业务场景中，最常见的链路是将实时采集的数据，通过实时计算初步清洗，实时写入数据至数据库，再对接BI工具实现数据的可视化分析。数据处理流程如下图所示。

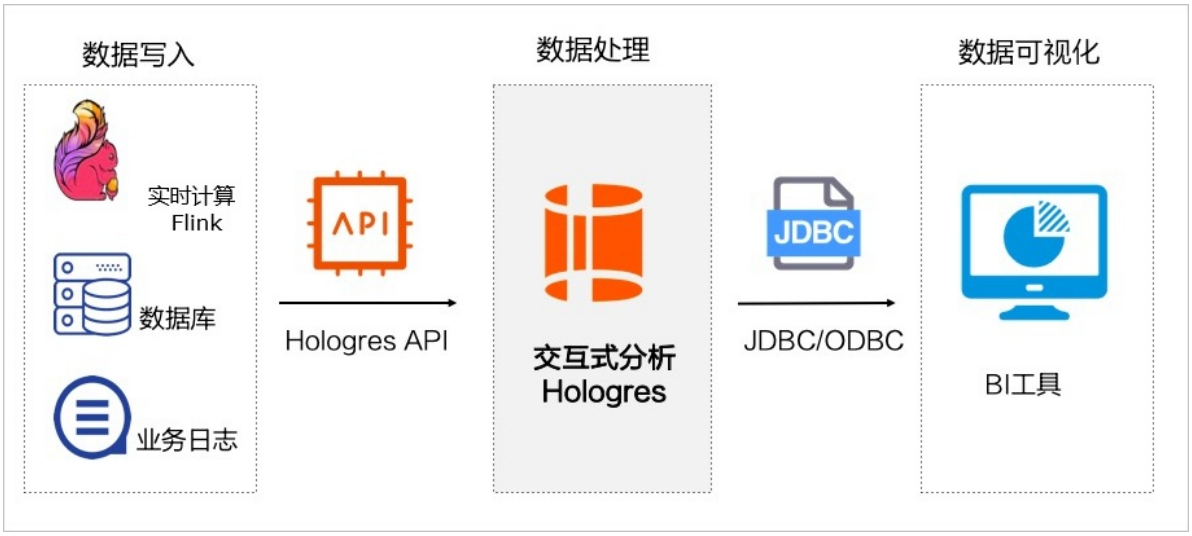
- 实时数据业务痛点。

  处理实时数据业务的整个链路中，要求数据库提供高性能的计算服务，存储海量数据，同时对接多种BI分析工具。单一的数据库很难实现以上所有功能，您必须借助其他数据库的相关能力完成业务流程。

借助其他数据库在导入导出数据时会产生冗余存储，浪费存储资源。同时需要维护多套系统，给开发及运维增加了一定难度。

- 实时数据API接口的引入。

  为解决实时场景的业务痛点，Hologres提供了实时数据API接口。业务数据及日志数据可以直接调用实时数据API接口，实时写入数据，再由Hologres提供高性能的计算服务和海量数据的存储服务。数据处理流程如下图所示。



  整个业务链路中，您无需导入导出数据，写入的数据统一存储在Hologres中，无冗余存储，节省计算及存储资源。一套系统就能满足您的多种需求，节省了开发和运维的成本。

5.3.1.3.2. Hologres源表

本文为您介绍在独享Blink中如何创建交互式分析Hologres源表。

使用说明

Hologres源表默认使用批模式读取数据，即对全表数据仅扫描一次。扫描结束，消费即结束，扫描结束后输入的数据将不会被Hologres源表读取。如果您需要使用实时消费Hologres的数据，请参见订阅Hologres Binlog。

使用限制

Hologres支持使用Holo-blink Connector将Hologres表作为源表读取数据并进行维表Join，其使用限制如下：

- 默认只能读取**行存储格式**的表数据。如果要读取列存储格式表的数据，需要配置 `bulkread='true'` 。
- 创建行存储表时，如果该表设置了主键，必须将主键配置为*clustering key*才能工作。Hologres创建源表的示例语句如下。

```
begin;
create table test(a int primary key, b text, c text[], d float8, e int8);
call set_table_property('test', 'orientation', 'row');
call set_table_property('test', 'clustering_key', 'a');
commit;
```

- Blink独享模式3.6以下的版本未支持Hologres数据源，您需要使用Hologres Connector，并引用相关JAR文件，才可以进行作业。请提交工单或者联系对应的技术支持获取JAR文件，并在作业中使用如下DDL语句创建Hologres源表。

DDL定义

- Blink独享模式3.6及以上版本支持Hologres数据源，其创建Hologres源表的DDL语句如下所示。

```
CREATE TABLE holo_dim_table
(
    pk VARCHAR
    ,seller_id VARCHAR
    ,seller_bc_type VARCHAR
    ,seller_tag  VARCHAR
    ,PRIMARY KEY (pk)
) with (
    type = 'hologres',
    `endpoint` = '<yourEndpoint>',  --当前Hologres实例VPC网络的Endpoint。
    `username` = '<yourUsername>',  --当前阿里云账号的AccessKey ID。
    `password` = '<yourPassword>',  --当前阿里云账号的AccessKey Secret。
    `dbname` = '<yourDbname>',  --Hologres的数据库名称。
    `tablename` = '<yourTablename>',  --Hologres用于接收数据的表名称。
    `bulkread`='true'
);
```

With参数的描述如下表所示。

参数	描述	是否必填
type	数据源类型。 默认为*hologres*。	是
dbname	Hologres的数据库名称。	是
tablename	Hologres用于接收数据的表名称。	是
username	当前阿里云账号的AccessKey ID。 您可以登录AccessKey 管理，获取AccessKey ID。	是
password	当前阿里云账号的AccessKey Secret。 您可以登录AccessKey 管理，获取AccessKey Secret。	是
endpoint	Hologres的VPC网络地址。 您可以登录Hologres管控台，进入目标实例的详情页，在**实例配置**中获取Endpoint。Endpoint需包含端口号，格式为ip:port。	是

参数	描述	是否必填
bulkread	参数取值如下： • true：表示可以使用Hologres的列存储表作为源表。 • false：表示只能使用Hologres的行存储表作为源表。 **说明** 如果您需要使用Hologres的列存储表作为源表进行作业，则需要配置该参数为*true*。	是

数据类型映射

Blink独享的数据类型与Hologres的数据类型映射，请参见数据类型汇总。

Blink实时消费Hologres Binlog

实时计算Blink 3.7及以上版本，支持Hologres Connector实时消费Binlog，请参见Blink实时消费Binlog。

5.3.1.3.3. Hologres维表

本文为您介绍Blink独享模式如何创建Hologres维表。

使用限制

- 建议使用**Hologres的行存表**，列存表对于点查场景性能开销较大。

  创建行存储表时必须设置主键，并且将主键配置为*clustering key*才可以工作，Hologres创建表的示例语句如下。

  ```
  begin;
  create table test(a int primary key, b text, c text, d float8, e int8);
  call set_table_property('test', 'orientation', 'row');
  call set_table_property('test', 'clustering_key', 'a');
  commit;
  ```

- 表的主键必须是Flink Join ON的字段，Flink Join ON的字段也必须是表的完整主键，两者必须完全匹配。
- Hologres Blink Connector的维表功能不支持一对多的输出。
- 不支持Blink写入时带有数组类型。

DDL语义

创建Hologres维表的DDL语句如下。

```
CREATE TABLE rds_dim_table(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME --定义维表的变化周期，表明该表是一张会变化的表。
) with (
  type='hologres',
  'dbname'='<yourDbname>',  --Hologres的数据库名称。
  'tablename'='<yourTablename>', --Hologres用于接收数据的表名称。
  'username'='<yourUsername>', --当前阿里云账号的AccessKey ID。
  'password'='<yourPassword>', --当前阿里云账号的AccessKey Secret。
  'endpoint'='<yourEndpoint>' --当前Hologres实例VPC网络的Endpoint。
);
```

With参数的描述如下表所示。

参数	描述	是否必选
type	维表类型。 固定值为*hologres*。	是
dbname	Hologres的数据库名称。	是
tablename	Hologres用于接收数据的表名称。	是
username	当前阿里云账号的AccessKey ID。 您可以登录AccessKey 管理，获取AccessKey ID。	是
password	当前阿里云账号的AccessKey Secret。 您可以登录AccessKey 管理，获取AccessKey Secret。	是
endpoint	Hologres的VPC网络地址。 您可以登录Hologres管控台，进入目标实例的详情页，在**实例配置**中获取Endpoint。Endpoint需包含端口号，格式为ip:port。	是

Cache参数

如果Hologres维表包含Cache参数，则可以参考如下参数描述。

参数	描述	是否必选	示例值

参数	描述	是否必选	示例值
cache	缓存策略	否	目前交互式分析Hologres维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果未找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cachesize）和缓存更新时间间隔（cachettlms）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新全量加载Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）较多的场景。 需要配置相关参数：缓存更新时间间隔（cachettlms），更新时间黑名单（cachereloadtimeblackList）。 **说明** • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。 • 对于数据量比较大的维表，选择CACHE ALL时，可能会出现OOM或者Full GC耗时很久的情况，针对这个问题，可以选择以下两种解决方式： ◦ 对于支持Cache All策略的维表，开启Partitionedjoin优化。 ▪ 3.6.0之前的版本，每个并发默认加载维表全量数据。 ▪ 3.6.0及之后的版本，CACHE ALL策略支持Partitionedjoin优化。 开启Partitionjoin优化后，每个并发只缓存自己并发所需要的数据。
cachesize	缓存大小	否	当缓存策略选择*LRU*时，可以设置缓存大小，默认为*10000*行。

参数	描述	是否必选	示例值
cachettlms	缓存失效时间，单位为毫秒。	否	• 当缓存策略选择*LRU*时，可以设置缓存失效时间，默认不过期。 • 当缓存策略选择*ALL*时，缓存失效时间为缓存重新加载的间隔时间，默认不重新加载。
partitionedjoin	根据分区查找数据。	否	partitionedjoin参数的取值如下： • *false*（默认值） • *true* **说明** 此处的分区指在缓存中按照并发对维表的key进行分区，并非支持Hologres分区表做维表。
async	是否异步读取数据。 异步模式可以并发地处理多个请求和响应，从而连续的请求之间不需要阻塞等待，提高查询的吞吐。但在异步模式下，无法保证请求的绝对顺序。	否	async参数的取值如下： • *false*（默认值） • *true* **说明** 有关异步请求的原理请参见维表 JOIN与异步优化
cacheempty	是否缓存join结果为空的数据。	否	cacheempty取值如下： • true（默认值）：表示缓存join结果为空的数据。 • false：表示不缓存join结果为空的数据。

使用示例

创建Hologres维表并导入数据的示例语句如下。

```
create table randomSource (a int, b VARCHAR, c VARCHAR) with (type = 'random');
create table dim_test (
      a int,
    b VARCHAR,
    c VARCHAR,
    PRIMARY KEY (a, b),
    PERIOD FOR SYSTEM_TIME
) with (
  type = 'hologres',
  'dbname'='<yourDbname>', --Hologres的数据库名称。
  'tablename'='<yourTablename>', --Hologres用于接收数据的表名称。
  'username'='<yourUsername>', --当前阿里云账号的AccessKey ID。
  'password'='<yourPassword>', --当前阿里云账号的AccessKey Secret。
  'endpoint'='<yourEndpoint>' --当前Hologres实例VPC网络的Endpoint。
);
create table print_sink (
  a int,
  b VARCHAR
) with (type = 'print', `ignoreWrite` = 'false');
insert into print_sink
select randomSource.a, dim_test.b from randomSource
LEFT JOIN dim_test FOR SYSTEM_TIME AS OF PROCTIME()
on randomSource.a = dim_test.a and randomSource.b = dim_test.b;
```

数据类型映射

Blink独享与Hologres的数据类型映射，请参见数据类型汇总。

5.3.1.3.4. Hologres结果表

Hologres与实时计算Blink独享模式（原产品线）深度融合，支持使用Connector的方式写入数据至Hologres结果表，您可以立即查询写入的数据。本文为您介绍实时计算Blink独享模式（原产品线）如何写入数据至Hologres结果表。

使用限制

- 不同Blink独享模式的版本开发语义不同，在使用之前，请先确定Blink独享模式的版本，并根据版本示例使用。
- 请确保开通的实时计算与Hologres地域一致，以免连接失败。
- Blink独享模式3.6之前的版本未内置Hologres Connector，实时写入数据至Hologres需要引用JAR文件，您可以提交工单或通过Hologres交流群（钉钉群号：32314975）获取。

  > **说明** 建议您升级至3.6及以上的版本进行作业。

- Blink独享模式3.7版本支持自动创建Hologres分区表，但是您需要在作业中配置`createparttable='true'`。同时，使用分区表的注意事项如下：
  - Hologres当前仅支持List分区。
  - 创建分区表时，需要显示指定的分区列。目前仅支持*text*和*int4*类型的分区列，并且分区的值不能包含短划线（-），例如`2020-09-12`。
  - 如果分区表配置了主键（pk），则分区列必须是pk的一部分。

- 创建分区子表时，子表分区列的值必须为固定值。
- 写入分区子表的数据对应的分区列值，必须与子表创建时定义的值严格匹配，否则会报错。
- 当前不支持*DEFAULT*分区功能。

- 当导入数据的Hologres目标表设置了主键，实时写入的默认语义不会按照主键进行更新，后续导入的主键数据如果重复，则会被丢弃。
- Hologres为异步写入数据，您在进行作业时需要添加 `blink.checkpoint.fail_on_checkpoint_error=true` 配置，当作业发生异常时才会触发Failover。Blink3.7.6及以上版本不需要添加该参数。

DDL语义

创建Hologres结果表的语句如下。

```
create table Hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='<yourDbname>', --Hologres的数据库名称。
  tablename='<yourTablename>', --Hologres用于接收数据的表名称。
  username='<yourUsername>', --当前阿里云账号的AccessKey ID。
  password='<yourPassword>', --当前阿里云账号的AccessKey Secret。
  endpoint='<yourEndpoint>'); --当前Hologres实例VPC网络的Endpoint。
```

WITH参数

参数	描述	示例
type	结果表的类型。 固定值为hologres。	*hologres*
endpoint	Hologres实例的VPC网络地址。 进入Hologres管理控制台，在目标实例详情页的**实例配置**获取Endpoint。Endpoint需包含端口号，格式为ip:port。	*demo-cn-hangzhou-vpc.hologres.aliyuncs.com:80*
username	AccessKey ID 您可以单击AccessKey 管理，获取AccessKey ID。	*xxxxm3FMWaxxxx*
password	AccessKey Secret 您可以单击AccessKey 管理，获取AccessKey Secret。	*xxxxm355fffaxxxx*
dbname	当前Hologres的数据库名称。	*Holodb*

参数	描述	示例
tablename	当前Hologres数据库的表名称。	*blink_test*
arraydelimiter	Hologres Sink支持将一个STRING字段按照field_delimiter切分为数组导入Hologres。 默认值为 *\u0002*。	*\u0002*
mutatetype	数据写入模式，详情请参见Hologres结果表。 默认值为*insertorignore*。	*insertorignore*
ignoredelete	是否忽略回撤消息。 • true：忽略回撤消息。 • false：不忽略回撤消息。 **说明** 该参数仅在使用流式语义时生效。 默认为*false*。 通常Flink的Groupby会产生回撤消息，回撤消息传输到Hologres Connecor时会产生Delete请求。	*false*
partitionrouter	是否写入分区表。 • true：写入分区表。 • false：不写入分区表。 默认为*false*。	*false*
createparttable	当写入分区表时，是否根据分区值自动创建分区表。Blink独享 3.7及以上版本支持该功能。 默认值为*false*。 **注意** 请您谨慎使用该功能，确保分区值不会出现脏数据，从而导致创建了错误的分区表。	*false*

> **说明** arraydelimiter、mutatetype、ignoredelete、partitionrouter及createparttable参数未在DDL示例语句中展示，如果您在实际应用中需要使用相应参数，可参考上述表格中的参数描述。

实时写入数据至Hologres普通结果表

1. Hologres创建表。

   在Hologres中创建一张用于接收数据的表。示例建表SQL语句如下。

   ```
   create table blink_test (a int, b text, c text, d float8, e bigint);
   ```

2. 创建实时计算作业。

   i. 登录实时计算控制台。

   ii. 创建实时计算作业。

   - 实时计算Blink 3.6及以上版本已支持Hologres数据源，您可以直接调用，示例SQL语句如下。

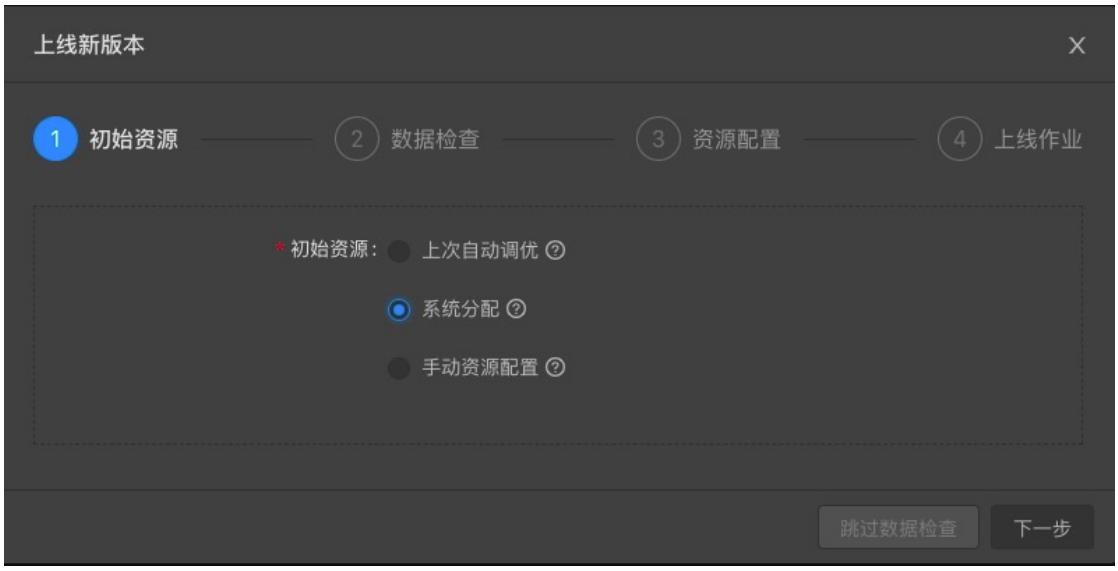
     ```
     create table randomSource (a int, b VARCHAR, c VARCHAR, d DOUBLE, e BIGINT) with
     (type = 'random');
     create table test (
       a int,
       b VARCHAR,
       c VARCHAR,
       PRIMARY KEY (a)
     ) with (
       type = 'hologres',
       `endpoint` = '$ip:$port', --当前Hologres的VPC网络地址以及端口号。
       `username` = '当前阿里云账号的AccessKey ID',
       `password` = '当前阿里云账号的AccessKey Secret',
       `dbname` = '当前Hologres的数据库名称',
       `tablename` = 'blink_test'--当前Hologres接收数据的表名称。
     );
     insert
       into test
     select
       a,b,c
     from
       randomSource;
     ```

3. 上线作业。

   i. 完成新建作业后，单击编辑框的**语法检查**，如果显示**成功**，则表明语法正确。

   ii. 单击**保存**保存作业。

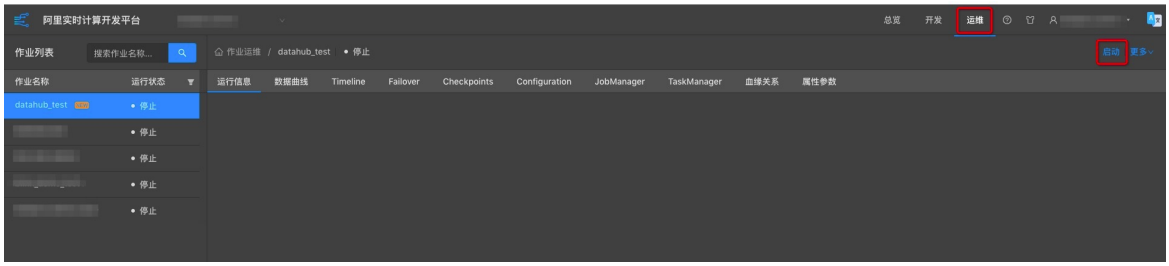
iii. 单击**上线**，提交作业至生产环境。根据业务情况填写作业的上线配置。



4. 启动作业。

   提交作业至生产环境后，您需要手动启动作业。

   在**阿里实时计算开发平台**页面顶部菜单栏右侧，单击**运维**，跳转至运维界面，选择需要启动的作业，单击右上角的**启动**。



5. Hologres实时查询数据。

   查询Hologres中用于接收数据的表，就可以实时获取到已写入的数据。示例查询SQL语句如下。

```
select * from blink_test;
```

如何使用宽表Merge/局部更新功能

对于常见的多个流的数据写入至一张Hologres宽表的场景，具体使用方法如下：

假设Hologres有一张宽表WIDE_TABLE，有A、B、C、D、E几列，其中A字段是主键，Flink一个流包含数据A、B、C，另一个流包含数据A、D、E。

1. 使用Flink SQL声明两张Hologres结果表，其中一张表只声明字段A、B、C，另一张表只声明字段A、D、E，这两张表都映射至*WIDE_TABLE*。
2. 两张结果表的mutatetype属性都设置成insertorupdate。
3. 两张结果表的ignoredelete属性都设置成true，防止回撤消息产生Delete请求。
4. 将两个流的数据分别Insert至两张结果表中。

该场景的具体使用限制如下：

1. 宽表必须有主键。

2. 每个流的数据都必须包含完整主键字段。
3. 列存表的宽表Merge场景在高RPS的情况下，CPU使用率会偏高，建议关闭表中字段的Dictionary encoding。

实时写入数据至Hologres的分区结果表

Hologres支持通过调用实时数据API接口，直接将数据写入分区父表中，对应的分区数据将会自动路由至分区子表。您可以直接写入数据至分区表。实时数据API的描述，详情请参见实时数据API。

使用限制如下：

- Hologres当前版本仅支持List分区。
- 创建分区表时，需要显示指定的分区列，分区列的类型仅支持*text*和 *int4*。
- 如果设置了主键，分区列必须为主键的一部分。
- 创建分区子表时，子表分区列的值必须为固定值。
- 写入分区子表的数据对应的分区列值，必须严格与创建子表时定义的值匹配，否则会报错。
- Hologres当前不支持默认分区。

1. Hologres创建分区表。

   在Hologres中创建一张用于接收数据的分区表，并创建对应的分区子表。示例建表SQL语句如下。

```
--创建分区父表test_message和对应的分区子表。
drop table if exists test_message;
begin;
create table test_message (
 "bizdate" text NOT NULL,
 "tag" text NOT NULL,
 "id" int4 NOT NULL,
 "title" text NOT NULL,
 "body" text,
PRIMARY KEY (bizdate,tag,id)
)
PARTITION BY LIST (bizdate);
commit;
```

   > **说明**
   > - 执行命令时，`${bizdate}`参数需要替换为实际值。
   > - Blink独享模式3.7版本才支持自动创建分区，如果您使用的是Blink独享模式3.7以下的版本，需要在Hologres中提前创建分区子表，否则会导入数据失败。

2. Blink独享创建作业。

   在Blink独享模式中创建作业的示例语句如下。

   > **说明** 以下示例适用于独享在Blink独享模式3.7及以上版本。如果您使用的是在Blink独享模式3.7以下版本，请升级至3.7及以上版本，或者删除自动创建分区子表的配置`` `createparttable` = 'true' ``。

```
create table test_message_src(
  tag VARCHAR,
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  type = 'random',
  `interval` = '10',
  `count` = '100'
);
create table test_message_sink (
  bizdate VARCHAR,
  tag VARCHAR,
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  type = 'hologres',
  `endpoint` = '$ip:$port', --Hologres实例的VPC网络地址。
  `username` ='<AccessID>', --当前阿里云账号的AccessKey ID。
  `password` = '<AccessKey>', --当前阿里云账号的AccessKey Secret。
  `dbname` = '<DBname>', --当前Hologres的数据库名称。
  `tablename` = '<Tablename>', --当前Hologres数据库的表名称。
  `partitionrouter` = 'true', --写入数据至Hologres的分区表。
  `createparttable` = 'true', --自动创建Hologres的分区子表。
);
insert into test_message_sink select "20200327",* from test_message_src;
insert into test_message_sink select "20200328",* from test_message_src;
```

3. 上线并启动作业。

   请参考*实时写入数据至Hologres结果表*章节中的*上线作业*和*启动作业*步骤。

4. Hologres实时查询数据。

   查询Hologres中用于接收数据的表，就可以实时获取到已写入的数据。示例查询SQL语句如下。

```
select * from test_message;
select * from test_message where bizdate = '20200327';
```

数据类型映射

Blink独享与Hologres的数据类型映射，请参见数据类型汇总。

5.3.1.4. 开源Flink

5.3.1.4.1. 开源Flink 1.10实时导入数据至Hologres

本文以一个示例为您演示开源Flink如何实时写入数据至Hologres。

前提条件

- 开通Hologres实例，并连接开发工具。本次示例使用psql客户端连接Hologres，详情请参见PSQL客户端。

- 创建Flink集群。您可以进入Flink官网下载二进制文件，启动一个Standalone集群，详细操作步骤请参见集群搭建。本次示例使用Flink1.10集群。
- 请您根据实际业务准备Flink的输入数据源。

操作步骤

1. Hologres创建结果表。

   Holoogres实例连接开发工具后，您需要创建一张结果表，用于接收实时写入的数据。示例语句如下。

   ```
   begin;
   create table order_details(user_id bigint, user_name text, item_id bigint, item_name te
   xt, price numeric(38, 2), province text, city text, ip text, longitude text, latitude t
   ext, sale_timestamp timestamptz not null, primary key(user_id, item_id));
   call set_table_property ('order_details', 'segment_key', 'sale_timestamp');
   commit;
   ```

2. 下载并编译Flink的JAR文件。

   i. 下载并安装Hologres Connector依赖的JAR文件hologres-flink-connector-1.10-jar-with-dependencies.jar，示例语句如下。

   ```
   mvn install:install-file -Dfile=hologres-flink-connector-1.10-jar-with-dependencies
   .jar -DgroupId=org.apache.flink -DartifactId=hologres-flink-connector -Dversion=1.1
   0 -Dpackaging=jar -DgeneratePom=true
   ```

   ii. 进入Hologres的Git官方示例库，下载并编译JAR文件，示例语句如下。

   ```
   git clone https://github.com/hologres/hologres-flink-examples.git
   cd hologres-flink-examples
   git checkout -b example
   mvn package -DskipTests
   ```

3. 提交Flink作业。

   编译完JAR文件后，配置作业参数，提交Flink作业，示例语句如下。

   > **说明** 示例使用命令行方式提交Flink作业，您也可以选择使用Flink Web页面提交作业。

   ```
   flink run -c io.hologres.flink.example.HologresSinkExample ../hologres-flink-example/ta
   rget/hologres-flink-examples-1.0.0-jar-with-dependencies.jar --endpoint $ENDPOINT --use
   rname $USERNAME --password $PASSWORD --database $DATABASE --tablename order_details
   ```

   参数说明如下表所示。

参数	描述	示例

参数	描述	示例
endpoint	Hologres的Endpoint地址。 进入Hologres管理控制台的实例详情页，从**实例配置**获取Endpoint。 **说明** 本地Flink请使用Hologres的公共网络地址，阿里云VPC网络请使用Hologres的VPC网络地址。	`ssseeee-cn-hangzhou.hologres.aliyuncs.com:80`
username	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。	无
password	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。	无
database	连接的Hologres数据库名称。	`hologres_demo`
tablename	Hologres接收数据的表名称。	`order_details`

4. Hologres查询数据。

   成功启动任务后，您可以在Hologres中实时查询写入的数据。示例语句如下。

```
select count(1) from order_details;
select item_id, sum(price) as total from order_details group by item_id limit 10;
```

5.3.1.4.2. 开源Flink 1.11及以上版本实时写入

本文为您介绍开源Flink 1.11如何实时写入数据至Hologres。

前提条件

- 开通Hologres实例，并连接开发工具，详情请参见连接HoloWeb。
- 搭建Flink集群（本次示例使用的是1.13版本），可以前往Flink官网下载二进制包，启动一个Standalone集群，详情请参见文档集群搭建

背景信息

从开源Flink1.11版本开始，Hologres代码已开源，相应版本的Connector已经在中央仓库发布Release包（1.0.1），可在项目中参照如下pom文件进行配置。详细内容请参见Hologres GitHub官方库。

```
<dependency>
    <groupId>com.alibaba.hologres</groupId>
    <artifactId>hologres-connector-flink-1.13</artifactId>
    <version>1.0.1</version>
    <classifier>jar-with-dependencies</classifier>
</dependency>
```

Flink SQL写入数据至Hologres代码示例

您可以参照如下代码示例，通过将Flink SQL将数据写入Hologres。其中，更多详细的代码示例请参见Hologres GitHub官方库。

```
        String createHologresTable =
                String.format(
                        "create table sink("
                                + "  user_id bigint,"
                                + "  user_name string,"
                                + "  price decimal(38,2),"
                                + "  sale_timestamp timestamp"
                                + ") with ("
                                + "  'connector'='hologres',"
                                + "  'dbname' = '%s',"
                                + "  'tablename' = '%s',"
                                + "  'username' = '%s',"
                                + "  'password' = '%s',"
                                + "  'endpoint' = '%s'"
                                + ")",
                        database, tableName, userName, password, endPoint);
        tEnv.executeSql(createHologresTable);
        createScanTable(tEnv);
        tEnv.executeSql("insert into sink select * from source");
```

更多详尽的代码示例请参见hologres-connector-flink-examples，包括如下示例。

- FlinkSQLToHoloExample：一个使用纯Flink SQL接口实现的应用，将数据写入至Hologres。
- FlinkDSAndSQLToHoloExample：一个混合Flink DataStream以及SQL 接口实现的应用，写入Hologres前，将DataStream转换成Table，之后再用Flink SQL写入Hologres。
- FlinkDataStreamToHoloExample：一个使用纯Flink DataStream接口实现的应用，将数据写入至Hologres。
- FlinkRoaringBitmapAggJob：一个使用Flink及RoaringBitmap，结合Hologres维表，实现实时去重统计UV的应用，并将统计结果写入Hologres。

Hologres Flink Connector参数说明

您可以将Flink数据写入Hologres，Hologres Flink Connector相关参数具体内容如下：

参数	是否必填	说明
connector	是	结果表类型，固定值为hologres。
dbname	是	Hologres的数据库名称。

参数	是否必填	说明
tablename	是	Hologres接收数据的表名称。
username	是	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
password	是	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
endpoint	是	Hologres的VPC网络地址。进入Hologres管理控制台的实例详情页，从**实例配置**获取Endpoint。 **说明** endpoint需包含端口号，格式为 `ip:port` 同一个区域使用VPC网络地址，跨区域请使用公共网络。

- 连接参数

参数	是否必填	说明
connectionSize	否	单个Flink Hologres Task所创建的JDBC连接池大小。 默认值：3，和吞吐成正比。
jdbcRetryCount	否	当连接故障时，写入和查询的重试次数。 默认值：10。
jdbcRetrySleepInitMs	否	每次重试的等待时间=retrySleepInitMs+retry*retrySleepStepMs。 默认值：1000ms。
jdbcRetrySleepStepMs	否	每次重试的等待时间=retrySleepInitMs+retry*retrySleepStepMs。 默认值：5000ms。
jdbcConnectionMaxIdleMs	否	写入线程和点查线程数据库连接的最大Idle时间，超过连接将被释放。 默认值：60000ms。
jdbcMetaCacheTTL	否	TableSchema信息的本地缓存时间。 默认值：60000ms。

参数	是否必填	说明
jdbcMetaAutoRefreshFactor	否	当TableSchema cache剩余存活时间短于 `metaCacheTTL/metaAutoRefreshFactor` 将自动刷新cache。 默认值：-1，表示不自动刷新。

- 写入参数

参数	是否必填	说明
mutatetype	否	数据写入模式，详情请参见流式语义。 默认值：insertorignore。
ignoredelete	否	是否忽略撤回消息。通常Flink的Group By会产生回撤消息，回撤消息到Hologres connector会产生Delete请求。 默认值：false，仅在使用流式语义时生效。
partitionrouter	否	是否写入分区表。 默认值：false。
createparttable	否	当写入分区表时，是否自动根据分区值自动创建分区表。Flink 全托管2.1.x及以上版本支持自动创建分区表。 ◦ false（默认值）：不会自动创建。 ◦ true：自动创建。 建议慎用该功能，确保分区值不会出现脏数据导致创建错误的分区表。
jdbcWriteBatchSize	否	Hologres Sink节点数据攒批的最大批大小。 默认值：256
jdbcWriteBatchByteSize	否	Hologres Sink节点单个线程数据攒批的最大字节大小。 默认值：2097152（2 * 1024 * 1024），2MB。
jdbcWriteBatchTotalByteSize	否	Hologres Sink节点所有数据攒批的最大字节大小。 默认值：20971520（20 * 1024 * 1024），20MB。

参数	是否必填	说明
jdbcWriteFlushInterval	否	Hologres Sink节点数据攒批的最长Flush等待时间。 默认值：10000，即10秒。
jdbcReWriteBatchedDeletes	否	将多条Delete请求合并为一条SQL语句提升性能。 默认值：true。
jdbcRewriteSqlMaxBatchSize	否	单条SQL进行INSERT/DELETE操作的最大批次大小，比如写入操作，所攒的批会通过 `writeBatchSize/rewriteSqlMaxBatchSize` 条INSERT语句完成插入。 默认值：1024。
jdbcEnableDefaultForNotNullColumn	否	设置为true时，not null且未在表上设置default的字段传入null时，将以默认值写入。String类型默认为""，Number类型默认为0，Date、timestamp、timestamptz 类型默认为1970-01-01 00:00:00。 默认值：true。
ignoreNullWhenUpdate	否	当 `mutatetype='insertOrUpdate'` 时，是否忽略更新写入数据中的Null值。 默认值：false。

- 点查参数

参数	是否必填	说明
jdbcReadBatchSize	否	维表点查最大批次大小。 默认值：128。
jdbcReadBatchQueueSize	否	维表点查请求缓冲队列大小。 默认值：256。

- 数据类型映射

  当前Flink全托管与Hologres的数据类型映射请参见实时计算与Hologres的数据类型映射。

5.3.2. Spark

5.3.2.1. Spark的数据写入至Hologres

本文为您介绍如何通过Spark读取或写入数据至Hologres的操作方法。

背景信息

Spark是用于大规模数据处理的统一分析引擎，Hologres已经与Spark（社区版以及EMR Spark版）高效打通，快速助力企业搭建数据仓库。Hologres提供的Spark Connector，支持Spark以批处理的方式将数据写入Hologres，同时Spark支持读取多种数据源（例如文件、Hive、MySQL、PostgreSQL等）。

Hologres兼容PostgreSQL，因此Spark也可以用读取PostgreSQL的方式直接读取Hologres数据，进行ETL处理，再写入Hologres及其他数据源，完成大数据开发抽取、处理、加载的完整闭环。

前提条件

- 实例版本需为V0.9及以上版本。请在Hologres管控台的实例详情页查看当前实例版本，如实例是V0.9以下版本，请您提交工单升级实例。
- 需要安装对应版本的Spark环境，能够运行 `spark-shell` 命令。

通过Spark Connector写入（推荐使用）

Hologres当前支持使用内置的Spark Connector将Spark数据写入Hologres，相比其他写入方式，调用基于Holo Client实现Connector写入的方式性能更优。具体操作步骤如下，阿里云也为您提供了相关的使用示例，详情请参见通过Spark Connector写入使用示例。

1. 获取JAR包。

   Spark2和Spark3上均已支持Connector写入，Spark写入Hologres时需要引用connector的JAR包，当前已经发布到Maven中央仓库，在项目中参照如下pom文件进行配置。

   > **说明** 相关Connector也已开源，详情请参见hologres-connectors。

   ```
   <dependency>
       <groupId>com.alibaba.hologres</groupId>
       <artifactId>hologres-connector-spark-2.x</artifactId>
       <version>1.0.1</version>
       <classifier>jar-with-dependencies</classifier>
   </dependency>
   ```

   当前Hologres已自动生成JAR文件，下载链接如下。

   - Spark2
   - Spark3

2. 使用JAR包。

   执行如下命令，启动Spark。

   ```
   spark-shell --jars hologres-connector-spark-2.x-1.0-SNAPSHOT-jar-with-dependencies.jar
   ```

3. 配置Spark。

   在Spark中执行如下命令，以连接Hologres将数据加载至目标表中。

   - 代码：

```
df.write
  .format("hologres")
  .option(SourceProvider.USERNAME, "AccessKey ID") //阿里云账号的AccessKey ID。
  .option(SourceProvider.PASSWORD, "Accesskey SECRET") //阿里云账号的Accesskey SECRET。
  .option(SourceProvider.ENDPOINT, "Ip:Port") //Hologres的Ip和Port。
  .option(SourceProvider.DATABASE, "Database")
  .option(SourceProvider.TABLE, "Table")
  .save()
```

说明 其中如下代码可以进行替换。

- 原代码：

```
.option(SourceProvider.endpoint, "Ip:Port")
.option(SourceProvider.database, "Database")
```

- 替换代码：

```
.option(SourceProvider.jdbcUrl, "jdbc:postgresql://Ip:Port/Database")
```

- 参数释义：

参数名	默认值	是否必填	参数描述
USERNAME	无	是	登录Hologres账号的AccessKey ID。您可以单击AccessKey 管理
PASSWORD	无	是	登录Hologres账号的AccessKey Secret。您可以单击AccessKey 管理
TABLE	无	是	Hologres用于接收数据的表名称。
ENDPOINT	无	与JDBCURL二选一	Hologres实例的网络域名。 您可以进入Hologres管理控制台实例详情页，从**实例配置**获取主机和端口号。
DATABASE	无	与JDBCURL二选一	Hologres接收数据的表所在数据库名称。
JDBCURL	无	与ENDPOINT+DATABASE组合设置二选一	Hologres的JDBCURL。
INPUT_DATA_SCHEMA_DDL	无	spark3.x必填，值为`<your_DataFrame>.schema.toDDL`。	Spark中DataFrame的DDL。

参数名	默认值	是否必填	参数描述
WRITE_MODE	INSERT_OR_REPLACE	否	当INSERT目标表为有主键的表时采用不同策略。 ■ INSERT_OR_IGNORE：当主键冲突时，不写入。 ■ INSERT_OR_UPDATE：当主键冲突时，更新相应列。 ■ INSERT_OR_REPLACE：当主键冲突时，更新所有列。
WRITE_BATCH_SIZE	512	否	每个写入线程的最大批次大小，在经过WRITE_MODE合并后的Put数量达到WRITE_BATCH_SIZE时进行一次批量提交。
WRITE_BATCH_BYTE_SIZE	2 MB	否	每个写入线程的最大批次Byte大小，在经过WRITE_MODE合并后的Put数据字节数达到WRITE_BATCH_BYTE_SIZE时进行一次批量提交。
WRITE_MAX_INTERVAL_MS	10000 ms	否	距离上次提交超过WRITE_MAX_INTERVAL_MS会触发一次批量提交。
WRITE_FAIL_STRATEGY	TYR_ONE_BY_ONE	否	当某一批次提交失败时，会将批次内的记录逐条提交（保序），单条提交失败的记录将会跟随异常被抛出。
WRITE_THREAD_SIZE	1	否	写入并发线程数（每个并发占用1个数据库连接）。 在一个Spark作业中，占用的总连接数与Spark并发相关，关系为 **总连接数**= `spark.default.parallelism * WRITE_THREAD_SIZE` 。
DYNAMIC_PARTITION	false	否	若为true，写入分区表父表时，当分区不存在时自动创建分区。
RETRY_COUNT	3	否	当连接故障时，写入和查询的重试次数。
RETRY_SLEEP_INIT_MS	1000 ms	否	每次重试的等待时间=RETRY_SLEEP_INIT_MS+RETRY_COUNT*RETRY_SLEEP_STEP_MS。
RETRY_SLEEP_STEP_MS	10*1000 ms	否	每次重试的等待时间=RETRY_SLEEP_INIT_MS+RETRY_COUNT*RETRY_SLEEP_STEP_MS。
CONNECTION_MAX_IDLE_MS	60000 ms	否	写入线程和点查线程数据库连接的最大IDLE时间，超过此时间的连接将被释放。

通过Spark Connector写入使用示例

根据如下示例步骤为您介绍，如何通过Spark Connector将数据写入Hologres。

1. 创建Hologres表。

   在Hologres中执行如下SQL命令创建目标表，用来接受数据。

```
CREATE TABLE tb008 (
  id BIGINT primary key,
  counts INT,
  name TEXT,
  price NUMERIC(38, 18),
  out_of_stock BOOL,
  weight DOUBLE PRECISION,
  thick FLOAT,
  time TIMESTAMPTZ,
  dt DATE,
  by bytea,
  inta int4[],
  longa int8[],
  floata float4[],
  doublea float8[],
  boola boolean[],
  stringa text[]
);
```

2. Spark准备数据并写入Hologres。

   i. 在命令行运行命令开启Spark。

```
spark-shell --jars hologres-connector-spark-2.x-1.0-SNAPSHOT-jar-with-dependencies.
jar
```

   ii. 在spark-shell里使用命令 `load spark-test.scala` 执行测试文件，加载测试示例。

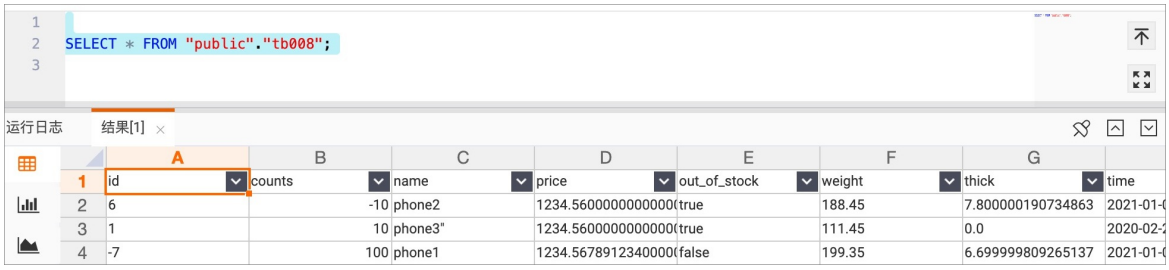
   *spark-test.scala*文件示例如下。

```
import java.sql.{Timestamp, Date}
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import com.alibaba.hologres.spark2.sink.SourceProvider
val byteArray = Array(1.toByte, 2.toByte, 3.toByte, 'b'.toByte, 'a'.toByte)
val intArray = Array(1, 2, 3)
val longArray = Array(1L, 2L, 3L)
val floatArray = Array(1.2F, 2.44F, 3.77F)
val doubleArray = Array(1.222, 2.333, 3.444)
val booleanArray = Array(true, false, false)
val stringArray = Array("abcd", "bcde", "defg")
val data = Seq(
  Row(-7L, 100, "phone1", BigDecimal(1234.567891234), false, 199.35, 6.7F, Timestam
p.valueOf("2021-01-01 00:00:00"), Date.valueOf("2021-01-01"), byteArray, intArray,
longArray, floatArray, doubleArray, booleanArray, stringArray),
  Row(6L, -10, "phone2", BigDecimal(1234.56), true, 188.45, 7.8F, Timestamp.valueOf
("2021-01-01 00:00:00"), Date.valueOf("1970-01-01"), byteArray, intArray, longArray
, floatArray, doubleArray, booleanArray, stringArray),
  Row(1L, 10, "phone3\"", BigDecimal(1234.56), true, 111.45, null, Timestamp.valueO
f("2020-02-29 00:12:33"), Date.valueOf("2020-07-23"), byteArray, intArray, longArra
```

```
y, floatArray, doubleArray, booleanArray, stringArray)
)
val schema = StructType(Array(
  StructField("id", LongType),
  StructField("counts", IntegerType),
  StructField("name", StringType, false), //false表示此Field不允许为null
  StructField("price", DecimalType(38, 12)),
  StructField("out_of_stock", BooleanType),
  StructField("weight", DoubleType),
  StructField("thick", FloatType),
  StructField("time", TimestampType),
  StructField("dt", DateType),
  StructField("by", BinaryType),
  StructField("inta", ArrayType(IntegerType)),
  StructField("longa", ArrayType(LongType)),
  StructField("floata", ArrayType(FloatType)),
  StructField("doublea", ArrayType(DoubleType)),
  StructField("boola", ArrayType(BooleanType)),
  StructField("stringa", ArrayType(StringType))
))
val df = spark.createDataFrame(
  spark.sparkContext.parallelize(data),
  schema
)
df.show()
//配置导入数据至Hologres的信息。
df.write.format("hologres") //必须配置为hologres
  .option(SourceProvider.USERNAME, "your_username") //阿里云账号的AccessKey ID。
  .option(SourceProvider.PASSWORD, "your_password") //阿里云账号的Accesskey SECRET。
  .option(SourceProvider.ENDPOINT, "Ip:Port") //Hologres的Ip和Port。
  .option(SourceProvider.DATABASE, "test_database") //Hologres的数据库名称,示例为test
_database。
  .option(SourceProvider.TABLE, "tb008") //Hologres用于接收数据的表名称，示例为tb008。
  .option(SourceProvider.WRITE_BATCH_SIZE, 512) // 写入攒批大小
  .option(SourceProvider.INPUT_DATA_SCHEMA_DDL, df.schema.toDDL) // Dataframe对应的D
DL，仅spark3.x需要
  .mode(SaveMode.Append) // 仅spark3.x需要
  .save()
```

3. 查询写入的数据。

   在Hologres侧查询目标表，即可确认写入的数据，示例如下图所示。



	A	B	C	D	E	F	G	
1	id	counts	name	price	out_of_stock	weight	thick	time
2	6	-10	phone2	1234.5600000000000	true	188.45	7.800000190734863	2021-01-(
3	1	10	phone3"	1234.5600000000000	true	111.45	0.0	2020-02-1
4	-7	100	phone1	1234.5678912340000	false	199.35	6.699999809265137	2021-01-(

通过Spark读取数据源数据并写入Hologres

1. Spark从数据源读取数据。

Spark支持从不同数据源读取数据，具体数据源分类如下。

- Spark支持以Hologres为数据源。

  Hologres兼容PostgreSQL，因为Spark可以用读取PostgreSQL的方式读取Hologres中的数据。读取代码如下。

  > **说明** 在使用JDBC方式进行读取前，请前往官网下载Postgresql JDBC Jar，在spark-shell启动时执行 `./bin/spark-shell --jars /path/to/postgresql-42.2.18.jar` 加载该jar，可以与hologres-connector的jar包一同加载。

```
// Read from some table, for example: tb008
val readDf = spark.read
  .format("jdbc") //使用postgresql jdbc driver读取holo
  .option("driver","org.postgresql.Driver")
  .option("url", "jdbc:postgresql://Ip:Por/test_database")
  .option("dbtable", "tb008")
  .option("user", "your_username")
  .option("password", "your_password")
  .load()
```

- Spark支持其他数据源（如Parquet格式的文件）。

  Spark支持从其他数据源中读取数据写入Hologres中，例如使用Spark从Hive中读取数据，代码如下。

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.sql.hive.HiveContext
val sparkConf = new SparkConf()
val sc = new SparkContext(sparkConf)
val hiveContext = new HiveContext(sc)
// Read from some table, for example: phone
val readDf = hiveContext.sql("select * from hive_database.phone")
```

2. Spark将读到的数据写入Hologres。

```
import com.alibaba.hologres.spark2.sink.SourceProvider
-- 函数实现见测试用例,也可以手动创建数据表
val table = createTableSql(readDf.schema, "tb009")
val df = spark.createDataFrame(
  readDf.rdd,
  readDf.schema
)
-- Write to hologres table, for example: tb009
df.write
  .format("hologres")
  .option(SourceProvider.USERNAME, "your_username")
  .option(SourceProvider.PASSWORD, "your_password")
  .option(SourceProvider.ENDPOINT, "Ip:Port")
  .option(SourceProvider.DATABASE, "test_database")
  .option(SourceProvider.TABLE, table)
  .option(SourceProvider.WRITE_BATCH_SIZE, 512) -- 写入攒批大小
  .option(SourceProvider.INPUT_DATA_SCHEMA_DDL, df.schema.toDDL) -- 仅spark3.x需要
  .mode(SaveMode.Append) // 仅spark3.x需要
  .save()
```

通过Spark实时写入数据至Hologres

1. 在Hologres创建一张表，用于接收数据，创建代码如下。

```
CREATE TABLE test_table_stream
(
    value text,
    count bigint
);
```

2. 读取本地端口输入行，进行词频统计并实时写入Hologres中，相关示例代码如下。

```
 val spark = SparkSession
      .builder
      .appName("StreamToHologres")
      .master("local[*]")
      .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    import spark.implicits._
    val lines = spark.readStream
      .format("socket")
      .option("host", "localhost")
      .option("port", 9999)
      .load()
    -- Split the lines into words
    val words = lines.as[String].flatMap(_.split(" "))
    -- Generate running word count
    val wordCounts = words.groupBy("value").count()
    wordCounts.writeStream
        .outputMode(OutputMode.Complete())
        .format("hologres")
        .option(SourceProvider.USERNAME, "your_username")
        .option(SourceProvider.PASSWORD, "your_password")
        .option(SourceProvider.JDBCURL, "jdbc:postgresql://Ip:Port/dbname")
        .option(SourceProvider.TABLE, "test_table_stream")
        .option("batchsize", 1)
        .option("isolationLevel", "NONE")
        .option("checkpointLocation", checkpointLocation)
        .start()
        .awaitTermination()
```

数据类型映射

Spark与Hologres的数据类型映射如下表所示。

Spark类型	Hologres类型
IntegerType	INT
LongType	BIGINT
StringType	TEXT
DecimalType	NUMERIC(38, 18)

Spark类型	Hologres类型
BooleanType	BOOL
DoubleType	DOUBLE PRECISION
FloatType	FLOAT
TimestampType	TIMESTAMPTZ
DateType	DATE
BinaryType	BYTEA
ArrayType(IntegerType)	int4[]
ArrayType(LongType)	int8[]
ArrayType(FloatType	float4[]
ArrayType(DoubleType)	float8[]
ArrayType(BooleanType)	boolean[]
ArrayType(StringType)	text[]

5.3.3. MySQL/PostgreSQL等数据库

5.3.3.1. 实时同步数据库的数据至Hologres

本文为您介绍如何通过DataWorks数据集成实时同步数据库中的数据至Hologres。

前提条件

- 开通DataWorks，详情请参见入门概述。
- 开通Hologres实例并绑定至DataWorks工作空间，详情请参见DataWorks快速入门。
- 已开通云数据库。

> **说明** 跨地域是否可以同步数据，详情请参见配置资源组与网络连通。

背景信息

Hologres是实时交互式分析产品，与大数据生态无缝打通，深度集成智能研发平台DataWorks，支持高并发和低延时地查询分析数据。您可以通过DataWorks数据集成同步将数据库中的数据实时同步至Hologres，再进行高并发低延时的查询分析处理。

常见的支持实时数据同步的数据库包括：Oracle、Polar DB、PolarDB MySQL等。

> **说明** 如需查看更多支持的数据库，请参见实时同步支持的数据源。

相关原理：MySQL Reader、Oracle Reader、PolarDB Reader、SQL Server Reader、Hologres Writer。

操作流程

通过DataWorks数据集成将多种数据库数据稳定、高效的实时同步至Hologres，请参见以下操作步骤进行操作。

1. 配置输入数据源

   在同步数据之前，需要配置数据来源的数据源。例如，您需要将MySQL数据实时同步至Hologres，就需要配置MySQL数据源。您可以根据业务场景选择数据源并配置，详情请参见配置数据源。

2. 配置输出Hologres数据源

   > **说明** Hologres数据源必须使用数据集成独享资源组。

   在同步之前，需要配置输出数据源Hologres，详情请参见配置Hologres数据源。

3. 配置任务

   输入数据源与输出Hologres数据源配置成功后，需要开始配置同步方式并运行任务，DataWorks数据集成提供三种实时同步方式，您可以根据业务需求选择，详情见下表。

同步类型	适用场景	支持的数据来源	数据源配置指导	同步任务配置指导
单表实时同步	适用于将源端部分表数据的变化实时同步至目标数据库中，实现目标库实时保持和源库的数据对应。	◦ MySQL Binlog ◦ DataHub ◦ LogHub ◦ Kafka ◦ PolarDB ◦ SQL Server	◦ 配置数据源（输入为PolarDB） ◦ 配置数据源（输入为MySQL）	通用配置流程
整库实时同步	适用于将源端全部表的数据变化实时同步至目标数据库中，实现目标库实时保持和源库的数据对应。	◦ PolarDB MySQL ◦ PolarDB ◦ MySQL	◦ 配置数据源（来源为PolarDB） ◦ 配置数据源（来源为Oracle） ◦ 配置数据源（来源为MySQL）	配置并管理实时同步任务

同步类型	适用场景	支持的数据来源	数据源配置指导	同步任务配置指导
同步解决方案	提供多种数据源之间进行不同数据同步场景的同步解决方案，包括实时数据同步、离线全量同步、离线增量同步等同步场景，助力企业数据更高效、更便捷的一键上云。主要包括： ○ 全量数据初始化。 ○ 增量数据实时写入。 ○ 增量数据和全量数据定时自动合并写入新的全量表分区。	○ PolarDB MySQL ○ Oracle ○ MySQL ○ DRDS ○ PostgreSQL	○ 配置数据源（来源为PolarDB） ○ 配置数据源（来源为Oracle） ○ 配置数据源（来源为MySQL） ○ 配置数据源（来源为DRDS） ○ 配置数据源（来源为PostgreSQL）	○ 配置查看数据同步任务 ○ 增加或删除已运行任务的同步表 ○ 查看同步任务运行状态

5.3.3.2. MySQL分库分表实践

本文为您介绍如何通过DataWorker数据集成和Flink两种方式将MySQL分库分表的数据写入至Hologres同一个表中，通过本实践您可以根据业务场景选择合适的方式将MySQL分库分表数据写入Hologres。

背景信息

实际业务场景下数据同步通常不能通过一个或多个简单离线同步或者实时同步任务完成，而是由多个离线同步、实时同步和数据处理等任务组合完成，这就会导致数据同步场景下的配置复杂度非常高。尤其是在MySQL分库分表的场景下，上游的数据库和表非常多，都需要同时写入一张Hologres表，如果要同时配置多个任务则会导致配置非常复杂且运维困难。

针对以上痛点，阿里云DataWorks数据集成一键同步解决方案提供了面向业务场景的同步任务配置化方案，支持不同数据源的一键同步功能，方便业务简单快速的进行数据同步。同时阿里云Flink也提供了丰富强大的数据实时入仓入湖能力，支持将多种数据源方便快捷的写入至Hologres。

前提条件

- 开通Hologres，详情请参见购买Hologres。
- 开通RDS MySQL数据库，详情请参见创建RDS MySQL实例。
- 如果选择使用DataWorks同步，请开通DataWorks，详情请参见开通DataWorks；同时购买DataWorks独享数据集成资源组，并完成资源配置，详情请参见新增和使用独享数据集成资源组。
- 如果选择使用Flink同步，请开通Flink，详情请参见开通流程。

> **说明** 开通的云服务请确保在同一个地域内。

MySQL分库分表通过DataWorks同步至Hologres

Hologres与DataWorks深度集成，支持一键同步解决方案将多种数据源数据同步至Hologres中，将MySQL分库分表数据通过一键同步解决方案同步至Hologres步骤如下。

1. 准备MySQL数据

   在同步之前需要准备好MySQL分库分表数据。本示例准备了两个库和三张表分别如下。

数据库名称	表名称	数据量
hmtest1	product_20220420	6301
hmtest1	product_20220421	6331
hmtest2	product_20220422	6227

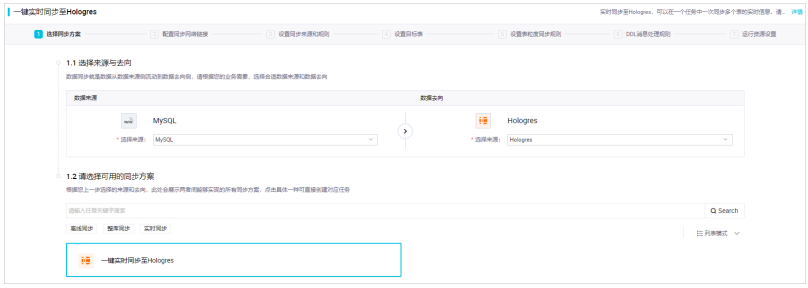
   表的DDL定义如下，三个表的Schema一致，但不同的表中会有部分数据重复。

```
CREATE TABLE product_20220420 (
 value_id int8,
 attribute_id int8 not null,
 id_card int8,
 name text,
 potion text,
    ds text,
PRIMARY KEY (`value_id`)
);
```

2. 配置一键实时同步至Hologres任务

   i. 创建一键实时同步任务

      a. 前往DataWorks数据集成创建一键实时同步至Hologres任务，详情请参见选择同步解决方案。数据来源选择为**MySQL**，数据去向选择为**Hologres**。



      b. 单击**下一步**。

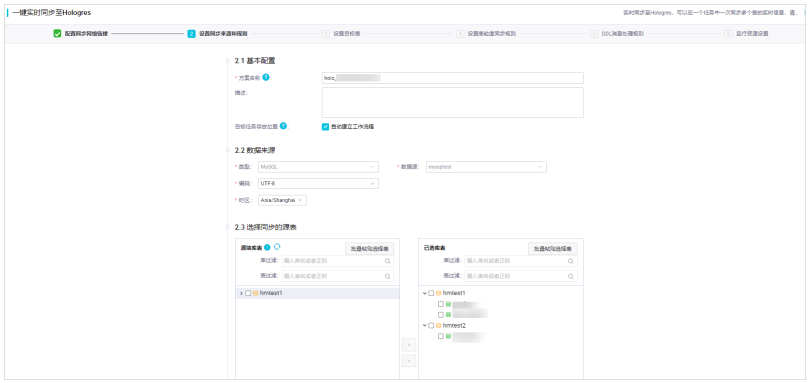
   ii. 配置同步网络链接

      a. 分别选择数据来源和数据去向的数据源名称后并单击**测试连通性**。

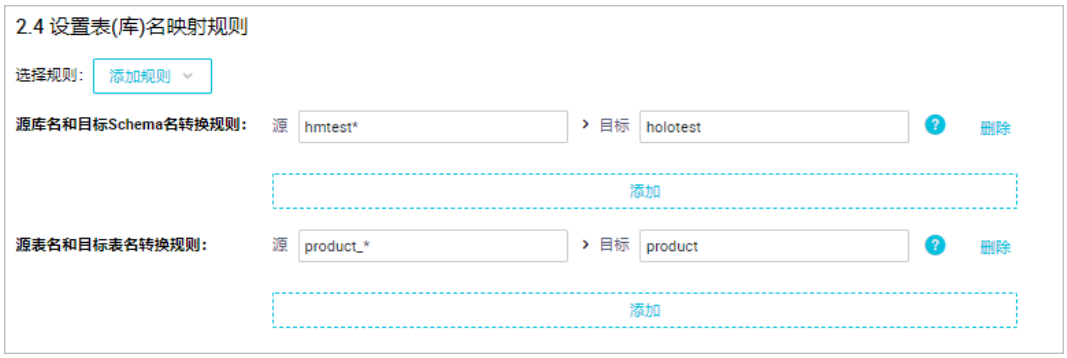
      b. 数据来源和数据去向都为可连通状态后，单击**下一步**。

iii. 设置同步来源和规则

a. 配置方案名称以及来源的基本信息，包括时区以及来源表。



b. 设置表（库）名的映射规则。如下示例通过正则匹配法选择出上游的库和表，实现分库分表写入同一个目标表。



c. 单击**下一步**。

iv. 设置目标表

a. 单击**刷新源表和Hologres表映射**。

> **说明** 映射关系里会展示每一个上游表与目标表的关系，只要目标表为同一个表即表示都映射至同一个目标表。

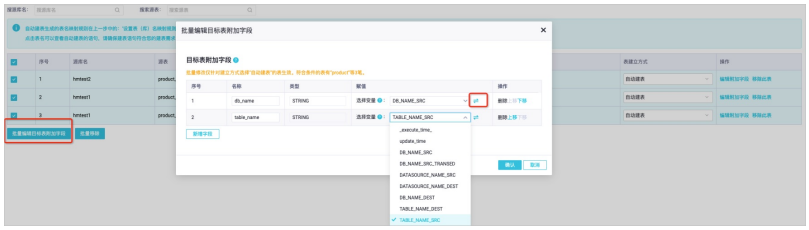
b. 添加附加字段

为了更好的区分上游表的来源，需要为目标表添加附加字段。

a. 勾选所有的任务，并单击**批量编辑目标表附加字段**。

b. 在**批量编辑目标表附加字段**弹窗，单击**新增字段**，分别新增db_name和table_name两个字段。

c. 新增字段后单击⇌为目标表添加附加字段。



本示例选择DB_NAME_SRC表示来源数据库名称；TABLE_NAME_SRC表示来源数据表名称。

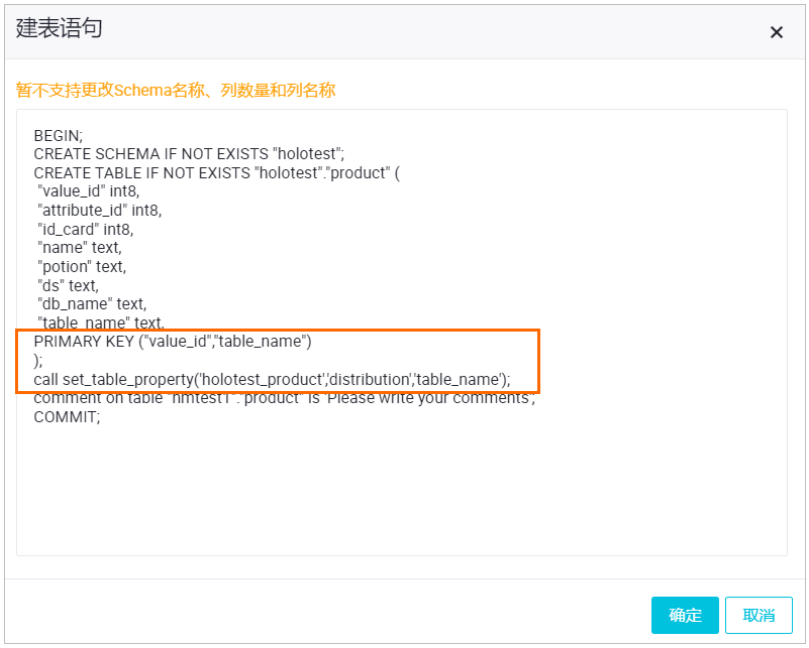
d. （可选）将附加字段设置为主键（PK）

若是上游数据量比较大，且表比较多，建议将附加字段设置成PK，与源表主键做联合主键，防止多源表主键数据互相冲突，同时将附加字段设置为Distribution Key，能保证将相同的数据写入至同一个Shard，实现更好的性能。

a. 单击表名称，进入**建表语句**弹窗。

b. 修改建表语句，附加列table_name添加为PK和Distribution Key。

> **说明**
>
> - 建议添加tablename为联合主键，可根据业务场景适当添加。
> - 也可以根据业务需求为表设置更多的索引，以实现更好的性能，详情请参见CREATE TABLE。



```
BEGIN;
CREATE SCHEMA IF NOT EXISTS "holotest";
CREATE TABLE IF NOT EXISTS "holotest"."product" (
 "value_id" int8,
 "attribute_id" int8,
 "id_card" int8,
 "name" text,
 "potion" text,
 "ds" text,
 "db_name" text,
 "table_name" text,
PRIMARY KEY ("value_id","table_name")
);
call set_table_property('holotest_product','distribution','table_name')
;
comment on table "hmtest1"."product" is 'Please write your comments';
COMMIT;
```

e. 单击**下一步**。

v. 设置DML策略

目标表设置完成之后，为任务配置DML策略。根据业务情况进行单表设置或者批量设置。

a. 勾选所有的任务，并单击**批量设置DML规则**。

b. 在**DML规则配置**弹窗，选择处理策略为**正常处理**。

c. 单击**确认**完成DML规格配置。

d. 单击**下一步**。

vi. 实时同步DDL消息处理策略

a. 根据业务情况为任务设置DDL消息处理策略。本示例DDL消息处理策略设置如下所示。

b. 单击**下一步**。

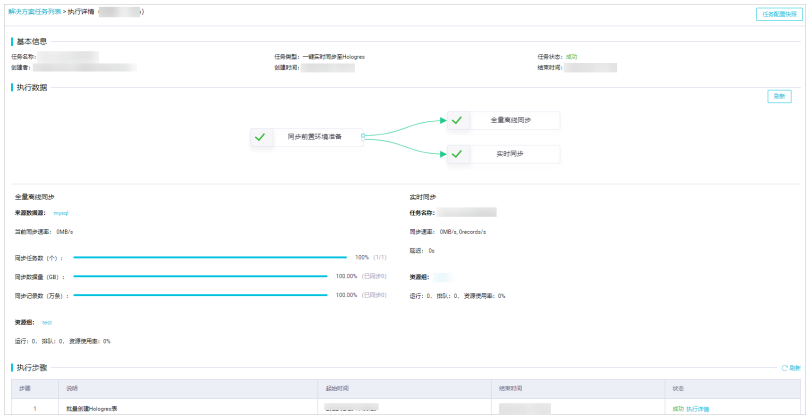
vii. 运行资源设置

a. 根据业务情况进行运行资源设置，包括资源组、连接数，并发数等。

b. 单击**完成配置**。

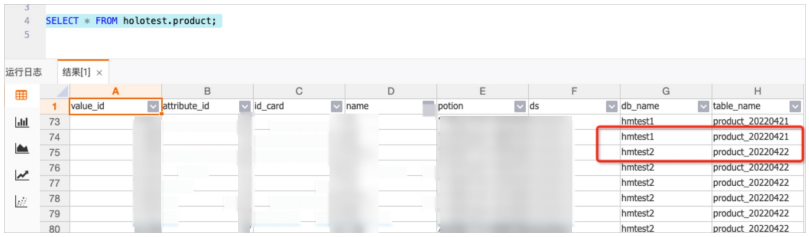
3. 运行任务

配置完成之后，提交任务执行，单击**执行详情**可以查看任务运行详情。



4. 查询数据

   一键解决方案会先运行全量数据，再运行实时同步数据。当全量离线任务运行完成后，可以前往Hologres中查询数据。



   如上示例可以看到附加列有对应的数据表示数据的来源库和表名，说明上游分库分表已经写入至Hologres的同一个表中。

业务上游有实时数据也会启动实时任务，如上游增加数据下游将会自动触发实时任务写入至Hologres中。本示例仅展示如何通过一键同步解决方案实现MySQL分库分表写入至Hologres一张表中，实现更多功能请根据业务逻辑自行配置任务。

MySQL分库分表通过Flink同步至Hologres

通过Flink将MySQL分库分表的数据同步至Hologres的具体操作请参见数据实时入仓入湖快速入门。

5.3.4. Kafka

5.3.4.1. Kafka通过DataWorks实时同步

本文为您介绍如何通过DataWorks数据集成将Kafka数据实时同步至Hologres。

前提条件

- 开通Hologres，并连接HoloWeb，详情请参见连接HoloWeb。
- 开通DataWorks，详情请参见入门概述。
- 准备好Kafka环境以及数据，详情请参见概述。

背景信息

Kafka 是一款高吞吐量、高可扩展性的分布式消息队列服务，广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等场景。Hologres与大数据生态无缝打通，与大数据智能研发平台DataWorks深度融合，您可以通过DataWorks数据集成将Kafka数据实时同步至Hologres，再进行高并发低延时的查询分析处理。相关原理请参见Kafka Reader和Hologres Writer。

单表实时同步

DataWorks数据集成通过Kafka服务的Java SDK从Kafka读取数据，再实时同步至Hologres。

1. 配置数据源

   在同步数据之前需要先配置输入数据源Kafka和输出数据源Hologres，详情请参见：

   - 配置Kafka数据源。
   - 配置Hologres数据源。

2. 配置同步任务

   配置数据源成功之后，可以配置同步任务将Kafka的数据实时同步至Hologres，详情请参见：

   - 配置Kafka输入。
   - 配置Hologres输出。

3. 查询数据

   任务同步成功之后，可以在Hologres中查询到已同步的数据。

Kafka通过Flink实时同步

Kafka也可以通过Flink初步清洗汇总指标再实时同步至Hologres，详情请参见Hologres结果表。

Kafka通过Hologres Connector写入

还可以通过Hologres Connector直接将Kafka数据写入Hologres，详情请参见Kafka写入Hologres。

5.3.5. DataHub

5.3.5.1. 实时同步DataHub的数据至Hologres

本文为您介绍如何使用Connector同步DataHub中的数据至Hologres。

前提条件

- 开通Hologres并连接开发工具，详情请参见PSQL客户端。
- 开通DataHub，详情请参见快速入门。
- 目前仅支持同步DataHub的TUPLE数据至Hologres中。

背景信息

阿里云流数据处理平台DataHub是流式数据（Streaming Data）的处理平台，提供对流式数据的发布（Publish）、订阅（Subscribe）和分发功能，让您可以轻松构建基于流式数据的分析和应用。

DataHub提供数据Sink/Source功能（数据同步），支持对Topic中的数据通过Hologres Connector实时同步到Hologres，对数据进行多维分析和实时探索。

DataHub与Hologres的概念映射如下表所示。

DataHub	Hologres
Project	Database
Topic	Table

使用限制

- Hologres实例开启白名单功能，则无法同步DataHub的数据至Hologres，请不要开启白名单功能。
- 数据写入分区表必须先在Hologres中创建分区子表，详情请参见CREATE PARTITION TABLE。
- 不支持写入带有Default值的表。

同步介绍

同步DataHub中的数据至Hologres，有两种同步模式和两种同步策略，同步模式与同步策略还可以分别进行组合，实现不同的效果。

> **说明**
> - 以下的两种同步模式和两种策略，不是DataHub的任务级别配置，而是在Hologres建表时的表属性，必须在建Hologres表时指定。
> - 同步DataHub中的数据至Hologres与DataWorks数据集成批量同步至Hologres SDK写入模式冲突，关于SDK写入模式的介绍请参见Hologres Writer。

- 同步模式
  - 逐条插入

    逐条插入是指将DataHub数据逐条写入Hologres，需要在Hologres建表时指定表属性如下。

    ```
    call set_table_property('<table_name>', 'datahub_sync_mode', 'none');
    ```

  - 回放

    回放是指回放上游数据库的DML操作，DataHub相当于是一个binlog，若是使用`dts-datahub-hologres`是否启用新的附加列规则，需要在Hologres建表时指定表属性如下。

    - 是否启用新的附加列规则选择为*是*时，需要在Hologres中建表时配置如下表属性。

      ```
      call set_table_property('<table_name>', 'datahub_sync_mode', 'dts');
      ```

    - 是否启用新的附加列规则选择为*否*时，需要在Hologres中建表时配置如下表属性。

      ```
      call set_table_property('<table_name>', 'datahub_sync_mode', 'dts_old');
      ```

    DTS在同步数据到DataHub时，会在数据列的基础上附加如下8列，用于描述回放数据信息（INSERT/UPDATE/DELETE），字段的主要说明如下。

    - 附加列命名方式

旧版数据列名称	新版数据列名称
dts_${原始列名}	new_dts_sync_dts_${原始列名}

- 附加列说明

旧版附加列名称	新版附加列名称	数据类型	说明
dts_record_id	new_dts_sync_dts_record_id	String	增量日志的记录ID，为该日志唯一标识。
dts_operation_flag	new_dts_sync_dts_operation_flag	String	操作类型，取值： ■ I：INSERT操作。 ■ D：DELETE操作。 ■ U：UPDATE操作。
dts_instance_id	new_dts_sync_dts_instance_id	String	数据库的server ID。暂不支持显示实际的值，目前固定为NULL。
dts_db_name	new_dts_sync_dts_db_name	String	数据库名称。
dts_table_name	new_dts_sync_dts_table_name	String	表名。
dts_utc_timestamp	new_dts_sync_dts_utc_timestamp	String	操作时间戳，即Binlog的时间戳（UTC时间）。
dts_before_flag	new_dts_sync_dts_before_flag	String	所有列的值是否更新前的值，取值：Y或N。
dts_after_flag	new_dts_sync_dts_after_flag	String	所有列的值是否更新后的值，取值：Y或N。

- 同步策略（主键冲突策略）

  当Hologres表设置主键时，从DataHub写入的数据有如下两种主键冲突策略。

  - 覆盖

    覆盖是指当写入发生主键冲突时，新的数据覆盖老数据，这个时候需要在Hologres建表时指定表属性如下。

    ```
    call set_table_property('<table_name>', 'datahub_upsert_mode', 'insert_or_replace');
    ```

  - 忽略

    忽略是指写入时发生主键冲突，忽略新数据，即数据不更新，仍然使用老数据，这个时候需要在Hologres建表时指定表属性如下。

    ```
    call set_table_property('<table_name>', 'datahub_upsert_mode', 'insert_or_ignore');
    ```

- 同步模式与同步策略组合

  以上通过DataHub写入Hologres的几种模式，不同组合之间实现的效果不同，具体请参见以下。

  - 插入模式与覆盖策略组合

    相当于在Hologres中执行以下SQL。

    ```
    INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO UPDATE
    ```

- 插入模式与忽略策略组合

  相当于在Hologres中执行以下SQL。

  ```
  INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO NOTHING
  ```

- 回放模式与覆盖策略组合
  - dts_operation_flag=I，相当于在Hologres中执行以下SQL。

    ```
    INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO UPDATE
    ```

  - dts_operation_flag=D，相当于在Hologres中执行以下SQL。

    ```
    DELETE FROM target_table where pk=?
    ```

  - dts_operation_flag=U AND dts_before_flag=Y，相当于在Hologres中执行以下SQL。

    ```
    DELETE FROM target_table where pk=?
    ```

  - dts_operation_flag=U AND dts_after_flag=Y，相当于在Hologres中执行以下SQL。

    ```
    INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO UPDATE
    ```

- 回放模式与忽略策略组合
  - dts_operation_flag=I，相当于在Hologres中执行以下SQL。

    ```
    INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO NOTHIN
    G
    ```

  - dts_operation_flag=D，相当于在Hologres中执行以下SQL。

    ```
    DELETE FROM target_table where pk=?
    ```

  - dts_operation_flag=U AND dts_before_flag=Y，相当于在Hologres中执行以下SQL。

    ```
    DELETE FROM target_table where pk=?
    ```

  - dts_operation_flag=U AND dts_after_flag=Y，相当于在Hologres中执行以下SQL。

    ```
    INSERT INTO target_table (column0,…,columnN) values (?,…,?) ON CONFLICT(PK) DO NOTHIN
    G
    ```

操作步骤

1. 准备DataHub数据源。

i. 创建项目。

a. 登录DataHub控制台，单击左侧导航栏的**项目管理**。

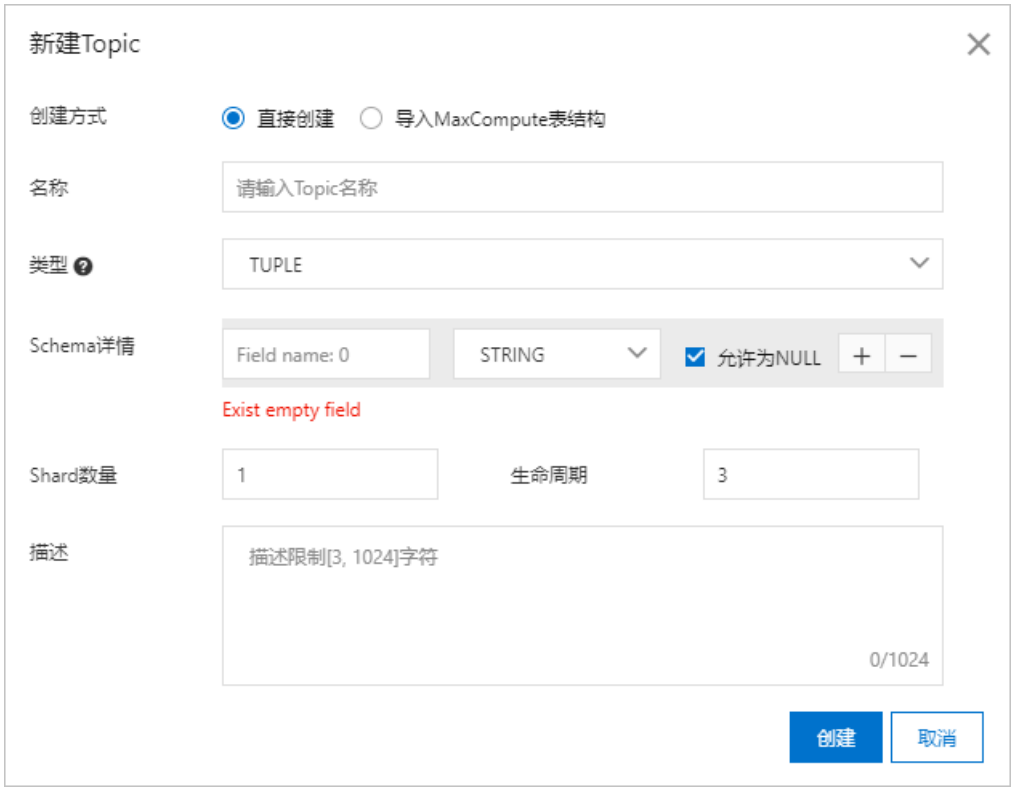
b. 在**项目列表**页面单击**新建项目**。

c. 在**新建项目**弹框，配置参数后，单击**创建**。

ii. 新建Topic。

a. 成功创建项目后，在**项目列表**页面单击项目名称，进入项目详情页。

b. 单击项目详情页右上角的**新建Topic**，进入**新建Topic**页面，填写配置参数。



参数	描述

参数	描述
创建方式	■ 直接创建：创建新的Topic。 ■ 导入MaxCompute表结构：选择MaxCompute中已有的表结构创建Topic。
名称	自定义Topic名称。
类型	Topic类型，分为以下两种： ■ TUPLE：结构化数据。 ■ BLOB：非结构化数据。
Schema详情	选择TUPLE类型会出现Schema详情，根据自己需求创建字段，允许为NULL代表如果上游没有该字段值自动置为NULL；不允许为NULL则会严格检验，字段类型不匹配写入报错。
Shard数量	Shard表示对一个Topic进行数据传输的并发通道，每个Shard会有对应的ID。每个Shard会有多种状态：Opening - 启动中，Active - 启动完成可服务。每个Shard启用以后会占用一定的服务端资源，建议按需申请Shard数量。
生命周期	Topic中写入数据在系统中可以保存的最长时间，以天为单位，最小值为1，最大值为7。
描述	Topic的描述信息。

iii. 写入数据。

成功创建Topic后，您需要使用工具（例如Blink）或者程序写入数据至Topic中。

2. Hologres创建数据接收表。

在Hologres中创建一张用于接收数据的表，表的字段类型与DataHub中Topic的字段类型相互映射。

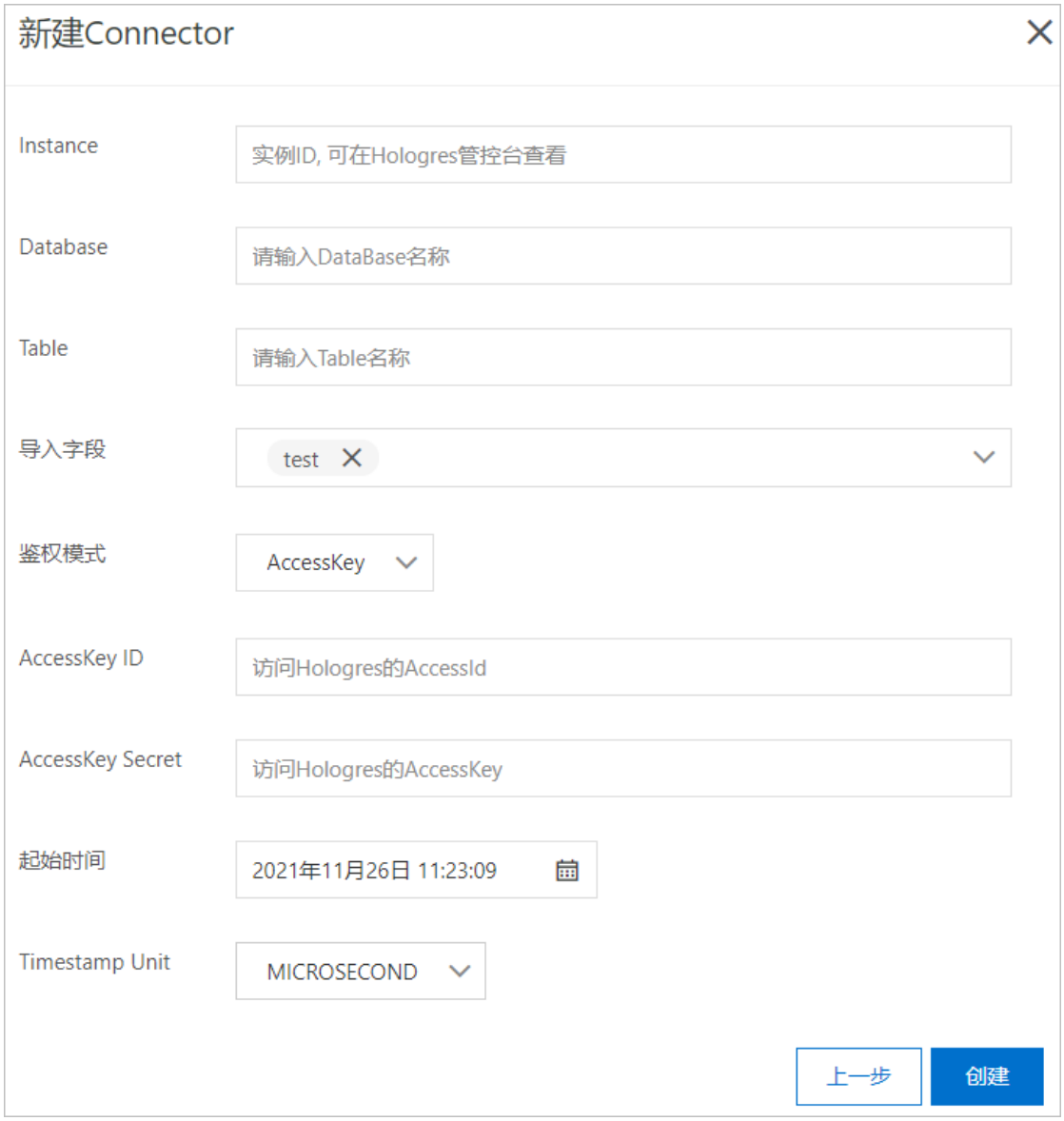
DataHub与Hologres的数据类型映射如下表所示。

DataHub	Hologres
BIGINT	BIGINT
STRING	TEXT
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE PRECISION
TIMESTAMP	TIMESTAMPTZ
DECIMAL	DECIMAL

示例建表语句如下。

```
BEGIN;
CREATE TABLE lineitem (
L_ORDERKEY BIGINT NOT NULL,
L_PARTKEY BIGINT NOT NULL,
L_SUPPKEY BIGINT NOT NULL,
L_LINENUMBER BIGINT NOT NULL,
L_QUANTITY DECIMAL(20,10),
L_EXTENDEDPRICE DECIMAL(20,10),
L_DISCOUNT DECIMAL(20,10),
L_TAX DECIMAL(20,10),
L_RETURNFLAG TEXT,
L_LINESTATUS TEXT,
L_SHIPDATE TIMESTAMPTZ,
L_COMMITDATE TIMESTAMPTZ,
L_RECEIPTDATE TIMESTAMPTZ,
L_SHIPINSTRUCT TEXT,
L_SHIPMODE TEXT,
L_COMMENT TEXT
);
CALL set_table_property('lineitem', 'orientation', 'column');
CALL set_table_property('lineitem', 'datahub_sync_mode', 'none');
CALL set_table_property('lineitem', 'datahub_upsert_mode', 'insert_or_ignore');
COMMIT;
```

3. 在DataHub中创建Hologres Connector。
   i. 单击DataHub中已创建的Topic，进入Topic详情页。
   ii. 单击Topic详情页右上角的**+同步**。
   iii. 在**新建Connector**界面单击**Hologres**，在**新建Connector**页面配置参数后，单击**创建**。

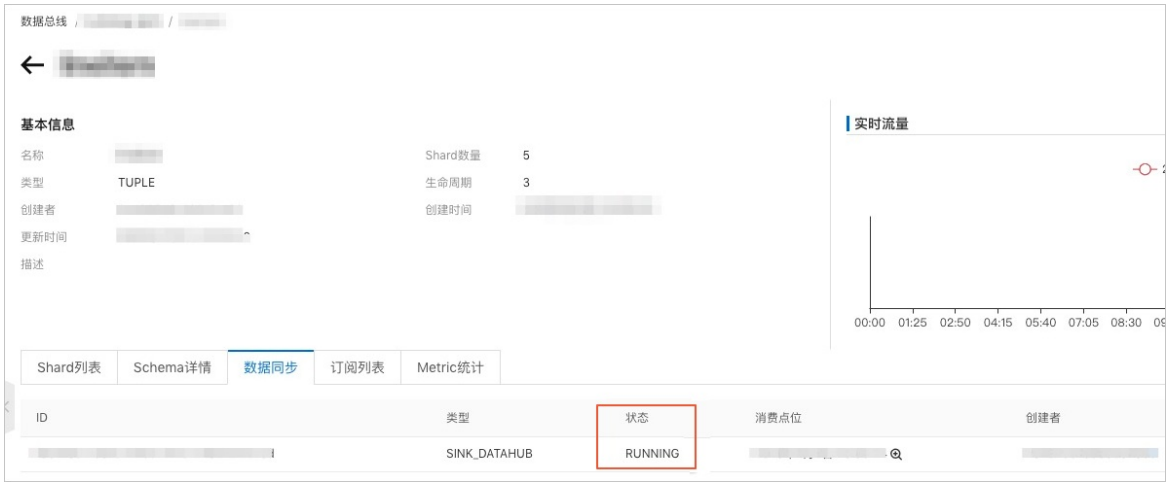


参数	说明
Instance	Hologres实例的ID。进入Hologres管理控制台，获取**实例ID**。
Project	Hologres的数据库名称。
Topic	Hologres用于接收数据的表名称。
导入字段	需要导入Hologres的字段。可以根据实际业务需求选择导入部分或全部字段。
鉴权模式	默认为AccessKey。
AccessKey ID	访问Hologres实例的AccessKey ID。您可以单击AccessKey 管理，获取用户的AccessKey ID。
AccessKey Secret	访问Hologres实例的AccessKey Secret。您可以单击AccessKey 管理，获取AccessKey Secret。

参数	说明
Timestamp Unit	同步时间单位，可选择如下。 ■ MICROSECOND：微秒，为默认值。 ■ MILLISECOND：毫秒。 ■ SECOND：秒。

4. 同步DataHub的数据至Hologres。

   成功创建Connector后，您可以在Topic详情页的**数据同步**中查看实时同步数据的状态。



5. Hologres查询数据。

   您可以连接Hologres实例至开发工具，实时查询同步至Hologres中的数据，详情请参见概述。示例查询语句如下。

```
SELECT COUNT(*) FROM lineitem;
```

常见报错

为您介绍在使用Hologres过程中的常见报错，以便于您能自行排查并解决问题。

- **场景1:**查询数据时，出现如下报错。

```
ErrorMessage [Import field not found in dest schema;
```

  可能原因：未设置datahub_sync_mode参数值为 `dts` 。

  解决办法：重新创建Hologres表，并设置表属性datahub_sync_mode为 `dts` 。

- **场景2:**查询数据时，出现如下报错。

```
ErrorCode=InternalServerError; ErrorMessage =Field already exists
```

  可能原因：Hologres列datahub_sync_mode设置为 `dts` ，并且建表时包含了8列附加列。

  解决办法：重新创建Hologres表，设置datahub_sync_mode为 `dts` 时，字段只需要跟上游保持一致，无需多增加8列附加列。

6.开发指南

6.1. Hologres开发规范

本文将为您介绍，Hologres在开发过程中的相关规范，帮助您快速了解Hologres的开发要求，避免进行错误的操作。

数据域规范

- **数仓分层**

  数据仓库以分层建设为主包含如下几类分层，其中CDM包含DWD、DWS、DIM。在Hologres中通过Schema隔离不同的分层。

  - ODS：操作数据层
  - CDM：公共维度模型层
    - DWD：明细数据层
    - DWS：汇总数据层
    - DIM：维度数据层
  - ADS：应用数据层

  不同公司可根据业务的复杂程度选择适当的颗粒度，如一个公司存在多个BU，则以BU缩写作为Schema的前缀。

```
create schema ${bu}_ads;
create schema ${bu}_ads_dev;
create schema ${bu}_dwd;
create schema ${bu}_dwd_dev;
create schema ${bu}_dws;
create schema ${bu}_dws_dev;
create schema ${bu}_dim;
create schema ${bu}_dim_dev;
create schema ${bu}_ods;
create schema ${bu}_ods_dev;
```

- **数据域缩写**

  不同的数据域需要定义共享的数据域代码，形成公司共识规范。示例如下。

数据域的名称	数据域对应的缩写示例
交易域	trd
商品域	itm
日志域	log
会员和店铺域	mbr
供销存管理域	dst
销售和客服域	crm

数据域的名称	数据域对应的缩写示例
信用和风控域	rsk
工具和服务域	tls
物流和快递域	lgt

命名规范

- **任务命名规范**

  任务命令时要区分内部任务还是同步任务，命名规则如下。

  - 内部SQL任务（非同步任务）：`holo_{target_table_name}`（与外部表任务区分）。
  - 数据导入Hologres：`{source}2holo_{target_table_name}`。
  - Hologres数据导出：`holo2{target}_{target_table_name}`。

- **表命名规范**

分层名称	当前分层中表的命名规则	示例
DWD	`${bu}_dwd.数据域_业务过程_[自定义词根]_后缀`	`taobao_dwd._trd_ord_flow`
DWS	`${bu}_dws.数据域_数据粒度缩写_业务过程_[{自定义词根}]_统计时间周期`	`taobao_dws.trd_all_dtr，taobao_dws.log_slr_pv_dtr`
DIM	`${bu}_dim.{维度定义}[_{自定义词根}]`	`taobao_cdm.dim_itm`
ADS	`${bu}_ads.业务域_维度_[{自定义词根}]_{刷新周期标识}` **说明** 刷新周期标识如下。 *d*：按天刷新。 *r*：实时刷新。 *h*：准实时刷新。	`taobao_ads.trd_cate_d`

- **Table Group命名规范**

  若是业务需要创建多个Table Group，命名可以采取`${bu}_{数仓分层名}_{业务定义}_tg`的规范。

- **视图命名规范**

  持久化视图命名规则与示例如下。

  - 规则
    - DWS：`${bu}_dws.数据域_数据粒度缩写_业务过程_[{自定义词根}]_统计时间周期_v`。
    - ADS：`${bu}_ads.业务域_维度_[{自定义词根}]_{刷新周期标识}_v`。
  - 示例

    ```
    taobao_dws.trd_byr_itm_ord_cm_v
    ```

- **外表命名规范**

  在原有MaxCompute表名基础下加 `ext` 后缀，具体示例如下。

  ```
  taobao_dim.camp_ext
  ```

- **临时表命名规范**

  在原有表名基础下加 `tmp` 前缀及数字序号后缀，具体示例如下。

  ```
  taobao_dim.tmp_camp_01
  ```

- **常用缩写词**

统计周期	缩写
最近一天	1d
最近多天	nd
累计	td
自然周	cw
自然月	cm
截止当前累计	dtr
截止当前小时累计	dhr

表开发规范

- **内表规范**

  创建表之前必须按照数据模型规范确定表和字段的命名，并根据需求确认表的生命周期，为表和字段添加完整注释，相关规范如下。

  - 强控规范（如不符合则不允许发布）：
    - 输出的表与字段需包含Comment，适用于全平台数据研发操作场景，表的Comment应确保描述信息简练、清晰。
    - 建表语句中需包含有表的生命周期（time_to_live_in_seconds）。
    - 建表语句需带有分布键索引（distribute_key），分部键选择原则如下。

      足够分散、最常JOIN或者GROUP BY的字段。例如买家商品表，可以设置user_id和item_id，但如果常关联的KEY为user_id，则分布键设置user_id而非user_id和item_id。
    - 进行关联查询的表需要创建在同一个Table Group中。
    - 同一个实体ID，在所有事实表和维表中保持名称和数据类型一致，比如交易表中用户ID为user_id，在维表中也为user_id，而不能是uid，同时数据类型保持一致，减少数据类型的转换。
    - 所有物理表的分区字段默认使用 `ds` 表示。

- 建议规范：
  - 建表语句应当带有bitmap_columns、segment_key、cluster_key任意之一。
  - 在不明确字段基数情况下，不建议设置建表属性`dictionary_encoding_columns`（字典索引），您可调用如下内容将属性置空。

    ```
    call set_table_property('table_name', 'dictionary_encoding_columns','')
    ```

  - 建表属性*orientation*（数据存储格式）建议使用*column*，可以设置为*row*。

    > **说明** 除非明确该表的查询能够始终指定所有的primary key（等于或者*in*），否则尽量不要使用*row*，不设置时默认是用*column*存储。

  - 建表属性*bitmap_columns*（比特编码），*bitmap*可以对存储文件内部的数据进行快速过滤。
    - 建议把filter条件的数据建成bitmap_columns，默认情况下会将所有的TEXT字段设置。
    - 不建议枚举值过多的字段，比如user_id，建议活动ID这类指标设置为*bitmap_columns*。
  - 建表属性event_time_column需用在与实时写入有关的字段上，例如事件时间戳。
  - 建表属性*clustering_key*聚簇索引，Hologres会在聚簇索引上对数据进行排序，建立聚簇索引能够加速在索引列上的*range*和*filter*查询，仅能设置一组。针对对于范围过滤适用，比如gmv分档时。

- **MaxCompute外表规范**

  Hologres支持通过外表对MaxCompute进行加速查询，可简化数据同步的流程。为了提升计算性能，非必要场景不建议您使用内表与外表关联。为更好的管理和维护外表，请遵循如下规范。

  - 强控规范：您需严格按照外表命名规范，在原有的MaxCompute表名基础下增加`ext`后缀。
  - 建议规范：
    - 保留外表的DDL，做好版本之间的管理。
    - 不建议内表与外表关联使用，建议采用外表同步至内表的方式。

- **视图规范**
  - 强控规范：您需严格按照视图命名规范。
  - 建议规范：
    - 建议您开启任务调度，保障后续开发作业依赖链路完整。
    - 建议不同粒度的视图单独创建，避免综合请求计算量过大。

      例如，cw、cm、nd、1d等，可分别建立4个视图。如存在分端，则建立pc、wap、app。如分采集方式，可以分ut和非ut。

- **生命周期（仅限内表）规范**

数仓分层	对应的生命周期规则描述
DWD	天级增量明细，建议不超过2年。
DWS	天级增量明细，建议不超过2年。
DIM	大维表建议进行极限存储建模后永久保存，小维表与MaxCompute表保持一致。 大、小维表的界定标准：单分区不可超过1 TB。

建议规范：

若是有分区表，建议按照实时任务写入当天分区，并且按照数仓分层设置合适的TTL，且更新的历史数据不应该写入已经超过TTL设置的分区。

- **Table Group规范（可选）**

  每个数据库都会有默认的Table Group和Shard数，您可以根据业务需要新建Table Group或者修改Shard数，以此达到更好的性能，建议规范如下。

  - 如无必要不要新建Table Group。
  - 数据量较大的表，可独立新建Shard数较大的Table Group。
  - 有大量数据量较小的表，可适当创建一个Shard数较小的Table Group。
  - 需要Join关联查询的表，必须放在同一个Table Group。

字段开发规范

- **字段类型规范**

  字段类型需严格按照如下要求进行创建。

字段/字段后缀	字段注释	示例	缩写
user_id	自增会员ID	`user_id=232442843`	`int8`
item_id	商品ID	`item_id=63283278784383`	`int8`
member_id	注册会员ID	`member_id=b2b-dsajk2343821b`	TEXT
amt	金额类	`pay_ord_amt_1d_001=923.23`	NUMERIC
fee	费用类	`post_fee=923.23`	NUMERIC
cnt	数量类	`pay_ord_byr_cnt_1d_001=923`	`int4/int8`
is_*	是否类	`is_pm=Y/is_pm=true`	TEXT/BOOL
ds	分区	`ds=20210120`	YYYYMMDD

- **基本数据类型参考**

  目前，Hologres数据类型与PostgreSQL数据类型兼容，但支持的数据类型是PostgreSQL的一个子集。详细字段类型及MaxCompute映射，请参见数据类型汇总。

- **货币单位及精度**

  中国内地地域货币单位统一为人民币，其他地域货币单位统一为美元。除非模型有特殊说明，否则金额相关的数据不做任何四舍五入操作，以免后续汇总计算中造成不同口径的汇总结果不一致。

SQL规范

- 强控规范：
  - SQL最外层及子查询内层不需要计算的字段禁止使用 `select *` 操作，所有操作必须明确指定列名。
  - Where条件中空字段和空字符串要进行必要的Coalesce处理。

- 建议规范：
  - 常采用count distinct字段作为distribution_key，对于多个count distinct的组合需要手动的改写。

```
select count(distinct userid)
     , count(distinct case when stat_date = '20201111' then userid end)
from t group by cate_id;
--改写为如下
select count(1), sum(c) from
(
  select userid
       , cate_id
       , cast(count(case when stat_date = '20201111' then 1 end) > 0) as c
  from t
  group by cate_id, userid
) t1
group by cate_id;
```

  - 离线调度任务增加analyze table操作分区表。
  - 针对长周期使用场景，批操作时采用ATTACH/DETACH操作历史分区，避免数据指标大起大落。

6.2. 数据类型

6.2.1. 数据类型汇总

Hologres的数据类型与PostgreSQL的数据类型兼容。本文为您介绍Hologres支持的数据类型及数组类型。

数据类型

当前Hologres版本支持的数据类型是PostgreSQL的一个子集。Hologres已支持的数据类型列表如下。

名称	支持的版本	存储大小	说明	取值范围	示例
INTEGER（别名INT或INT4）	Hologres所有版本	4字节	常用的整数	-2147483648 ~ +2147483647	2147483647
BIGINT（别名INT8）	Hologres所有版本	8字节	大范围整数	-9223372036854775808 ~ +9223372036854775807	9223372036854775807
BOOLEAN（别名BOOL）	Hologres所有版本	1字节	布尔类型	• True • False	True

名称	支持的版本	存储大小	说明	取值范围	示例
REAL（别名FLOAT4）	Hologres所有版本	4字节	可变精度，不精确。 **说明** 在PostgreSQL生态中FLOAT不设置精度默认对应的是DOUBLE PRECISION（FLOAT8）。	精度为6位的十进制数字。	123.123
DOUBLE PRECISION（别名FLOAT8）	Hologres所有版本	8字节	可变精度，不精确。	精度为15位的十进制数字。	123.123456789123
TEXT（别名VARCHAR）	Hologres所有版本	可变长	可变长度字符串。	无	abcdefg
TIMESTAMP WITH TIME ZONE（别名TIMESTAMPTZ）	Hologres所有版本	8字节	带时区的时间戳。存储精度为毫秒。 **说明** 标准PostgreSQL使用 `TIMESTAMPTZ` 的+或者-符号，以及符号后的时区偏移来识别 `TIMESTAMP WITH TIME ZONE` 的时区。如果未表明时区偏移，则系统自动给数据添加默认时区。	4713 BC ~ 294276 AD	2004-10-19 10:23:54+02
DECIMAL（别名NUMERIC）	Hologres所有版本	可变长	需要指定PRECISION和SCALE： • *PRECISION*：表示数字的位数，取值范围为0~38。 • *SCALE*：表示小数部分的位数，取值范围为0~PRECISION。	可以指定数据到小数点前的38位和小数点后的38位。	DECIMAL(38, 10)
DATE	Hologres 0.8版本新增	4字节	单位为一天。	4713 BC ~ 5874897 AD	2004-10-19

名称	支持的版本	存储大小	说明	取值范围	示例
TIMESTAMP	Hologres 0.8版本新增	8字节	不包含时区的时间戳，存储精度为微秒。	4713 BC ~ 5874897 AD	2020-01-01 01:01:01.123456
CHAR(n)	Hologres 0.8版本新增	固定字符长度，最长n个字符。	存储字节不大于1 GB。	固定字符长度的字符串。	• abcd • 中国人民
VARCHAR(n)	Hologres 0.8版本新增	可变长，不超过n个字符。	存储字节不大于1 GB。	有限字符长度的可变字符串。	abcdefg
SERIAL（自增序列）	Hologres 0.8版本新增	详情请参见自增序列Serial（Beta）。	无	无	无
SMALLINT	Hologres 0.9版本新增	2字节	小范围整数	-32768~+32767	32767
JSON和JSONB	Hologres 0.9版本新增	详情请参见JSON和JSONB类型。	无	无	无
BYTEA	Hologres 0.9版本新增	可变长，详情请参见Binary Data Types。	可变长的二进制串。	存储字节不超过1 GB。	无
RoaringBitmap	Hologres 0.10版本新增	可变长，详情请参加Roaring Bitmap函数。	高效的INT类型数组，支持常量数组位图计算。	无	无
BIT(n)	Hologres 0.9版本新增	长度为n个bit的二进制串	固定长度的二进制串。	存储字节不超过1 GB。	无
VARBIT(n)	Hologres 0.9版本	可变长，长度不超过n个bit的二进制串	有限bit长度的二进制串	存储字节不超过1GB。	无
INTERVAL	Hologres 所有版本	16字节	无	-178000000 years~178000000 years	interval '1 year'

名称	支持的版本	存储大小	说明	取值范围	示例
TIMETZ	Hologres 0.9版本新增	12字节	一天中的时间（带时区），解析度为微秒。	00:00:00~24:00:00	12:00:00+08
TIME	Hologres 0.9版本新增	8字节	一天中的时间（不带时区），解析度为微秒。	00:00:00~24:00:00	12:00:00
INET	Hologres 0.9版本新增	详情请参见网络地址类型。	INE在一个数据域里保存一个IPv4 或IPv6主机地址。	无	192.168.100.128/25
MONEY	Hologres 0.9版本新增	8字节，详情请参见货币类型。	money类型存储固定小数精度的货币数字。	-922337203685477758.08~+922337203685477758.07	12.34美元
OID	Hologres 0.9版本新增	4字节	数字形式的对象标识符。	无	1024
UUID	Hologres 0.9版本新增	16字节	通用唯一标识符，定长128位。 说明 目前还不支持uuid-ossp中实现的算法。详情请参见uuid。	00000000-0000-0000-0000-000000000000~ffffffff-ffff-ffff-ffff-ffffffffffff	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11

TIMESTAMP WITH TIME ZONE、DATE和DECIMAL的示例SQL语句如下。

```
CREATE TABLE test_data_type (
  tswtz_column TIMESTAMP WITH TIME ZONE,
  date_column date,
  decimal_column decimal(38, 10),
  char_column char(20),
  varchar_volumn varchar(225)
);
INSERT INTO test_data_type
VALUES ('2004-10-19 08:08:08', '2004-10-19', 123.456, 'abcd', 'a');
SELECT * FROM test_data_type;
      tswtz_column      | date_column | decimal_column |     char_column      | varchar_volumn
------------------------+-------------+----------------+----------------------+----------------
 2004-10-19 08:08:08+08 | 2004-10-19  | 123.4560000000 | abcd                 | a
(1 row)
```

BIT、VARBIT和BYTEA的示例SQL语句如下。

```
//BIT、VARBIT
CREATE TABLE test (a BIT(3), b BIT VARYING(5));
INSERT INTO test VALUES (B'101', B'00');
INSERT INTO test VALUES (B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
INSERT INTO test VALUES (B'10'::bit(3), B'101');
SELECT * FROM test;
  a  |  b
-----+-----
 101 | 00
 100 | 101
//BYTEA
SET bytea_output = 'escape';
SELECT 'abc \153\154\155 \052\251\124'::bytea;
     bytea
----------------
 abc klm *\251T
RESET bytea_output;  -- 'hex' by default
SELECT 'abc \153\154\155 \052\251\124'::bytea;
          bytea
--------------------------
 \x616263206b6c6d202aa954
(1 row)
```

数组类型

Hologres当前版本仅支持如下一维数组：

- int4[]
- int8[]
- float4[]
- float8[]
- boolean[]
- text[]

使用示例：

- 声明数组。

  ```
  CREATE TABLE array_example(
  int4_array int4[],
  int8_array int8[],
  float4_array float4[],
  float8_array float8[],
  boolean_array boolean[],
  text_array text[]);
  ```

- 插入数组。

- 使用ARRAY关键字。

```
INSERT INTO array_example(
int4_array,
int8_array,
float4_array,
float8_array,
boolean_array,
text_array)
VALUES (ARRAY[1, 2, 3, 4],
ARRAY[1, 2, 3, 4],
ARRAY[1.0, 2.0],
ARRAY[1.0, 2.0, 3.0],
ARRAY[true, true, false],
ARRAY['foo1', 'foo2', 'foo3']);
```

- 使用 `{}` 表达式。

```
INSERT INTO array_example(
int4_array,
int8_array,
float4_array,
float8_array,
boolean_array,
text_array)
VALUES ('{1, 2, 3, 4}',
'{1, 2, 3, 4}',
'{1.0, 2.0}',
'{1.0, 2.0, 3.0}',
'{true, true, false}',
'{"foo1", "foo2", "foo3"}');
```

- 查询数组。
  - 查询数组中单个元素。

```
SELECT int4_array[3] FROM array_example;
```

  - 查询数组中多个元素。

```
SELECT int4_array[1:2] FROM array_example;
```

MaxCompute与Hologres的数据类型映射

创建MaxCompute外部表时，MaxCompute与Hologres的数据类型映射如下表所示。

MaxCompute数据类型	Hologres数据类型	支持映射的版本	说明
• STRING • VARCHAR	TEXT	Hologres所有版本	无
BIGINT	INT8	Hologres所有版本	无

MaxCompute数据类型	Hologres数据类型	支持映射的版本	说明
INT	• INT4 • INT	Hologres所有版本	无
FLOAT	• FLOAT4 • REAL	Hologres所有版本	无
DOUBLE	• FLOAT • FLOAT8	Hologres所有版本	无
BOOLEAN	BOOL	Hologres所有版本	无
DATETIME	TIMESTAMP WITH TIME ZONE	Hologres所有版本	MaxCompute的DATETIME是日期时间类型，使用东八区时间作为系统标准时间。范围从0000年1月1日到9999年12月31日，精确到毫秒 。
DECIMAL	NUMERIC	Hologres所有版本	MaxCompute的DECIMAL如果未指定精度，则默认为（38，18），使用IMPORT FOREIGN SCHEMA创建表时系统会自动转换精度。
TIMESTAMP	TIMESTAMP WITH TIME ZONE	Hologres 0.8新增	• MaxCompute的TIMESTAMP取值范围为0000-01-01 00:00:00.000000000~9999-12-31 23.59:59.999999999，精确到纳秒。 • Hologres支持的TIMESTAMPTZ精确到毫秒。 MaxCompute与Hologres在底层已经进行了精度转换，您无需再关心转换问题。

MaxCompute数据类型	Hologres数据类型	支持映射的版本	说明
CHAR(n)	默认为CHAR(n)。 Hologres也支持映射MaxCompute的CHAR(n)为TEXT类型。您需要设置参数 `set hg_enable_convert_type_for_foreign_table = true` ，并在建表时将字段类型修改为TEXT。	Hologres 0.8新增	MaxCompute的CHAR(n)为固定长度字符类型，n为长度。最大取值为255。长度不足则使用空格填充。
VARCHAR(n)	默认为VARCHAR(n)。 Hologres也支持映射MaxCompute的VARCHAR(n)为TEXT类型。您需要设置参数 `set hg_enable_convert_type_for_foreign_table = true` ，并在建表时将字段类型修改为TEXT。	Hologres 0.8新增	MaxCompute的VARCHAR(n)为变长字符类型，n为长度。取值范围为1~65535。
DATE	DATE	Hologres 0.8新增	无
SMALLINT	默认为INT2。 Hologres也支持映射MaxCompute的SMALLINT为INT8类型。您需要设置参数 `set hg_enable_convert_type_for_foreign_table = true` ，并在建表时将字段类型修改为INT8。	Hologres所有版本（0.8版本为 int4, 0.9版本为int2）	无
TINYINT	默认为INT2。 Hologres也支持映射MaxCompute的TINYINT为INT8类型。您需要设置参数 `set hg_enable_convert_type_for_foreign_table = true` ，并在建表时将字段类型修改为INT8。	Hologres所有版本（0.8版本为 int4, 0.9版本为int2）	无
CHAR	不支持	不支持	无
ARRAY<INT>	INT4[]	Hologres 0.8新增	无
ARRAY<BIGINT>	INT8[]	Hologres 0.8新增	无
ARRAY<FLOAT>	FLOAT4[]	Hologres 0.8新增	无

MaxCompute数据类型	Hologres数据类型	支持映射的版本	说明
ARRAY<DOUBLE>	FLOAT8[]	Hologres 0.8新增	无
ARRAY<BOOLEAN>	BOOLEAN[]	Hologres 0.8新增	无
ARRAY<STRING>	TEXT[]	Hologres 0.8新增	无
BINARY	BYTEA	Hologres 0.9新增	无
ARRAY<TINYINT>	不支持	不支持	无
ARRAY<SMALLINT>	不支持	不支持	无

说明 当MaxCompute数据表中含有Hologres不支持的类型字段时，如果Hologres不访问该字段，则可以正常查询所支持的类型字段。

实时计算与Hologres的数据类型映射

实时计算与Hologres的数据类型映射如下表所示。

实时计算数据类型	Hologres数据类型	支持映射的版本
VARCHAR	TEXT	Hologres所有版本
BIGINT	INT8	Hologres所有版本
INT	• INT4 • INT	Hologres所有版本
SMALLINT	• INT4 • INT	Hologres所有版本
TINYINT	• INT4 • INT	Hologres所有版本
FLOAT	• FLOAT4 • REAL	Hologres所有版本
DOUBLE	• FLOAT • FLOAT8	Hologres所有版本
BOOLEAN	BOOL	Hologres所有版本
TIMESTAMP	TIMESTAMP WITH TIME ZONE	Hologres所有版本
DATE	DATE	Hologres 0.8新增

实时计算数据类型	Hologres数据类型	支持映射的版本
DECIMAL	NUMERIC	Hologres所有版本
TIME	不支持	不支持
CHAR	不支持	不支持
BINARY	不支持	不支持
VARCHAR	JSON	Hologres 0.9新增
VARCHAR	JSONB	Hologres 0.10新增
BYTES	RoaringBitmap	Hologres 0.10新增，在实时计算引擎vvr-4.0.12-flink-1.13以上版本支持
ARRAY<INT>	INT[]	Hologres 0.8新增
ARRAY<BIGINT>	BIGINT[]	Hologres 0.8新增
ARRAY<REAL>	REAL[]	Hologres 0.8新增
ARRAY<DOUBLE PRECISION>	DOUBLE PRECISION[]	Hologres 0.8新增
ARRAY<BOOLEAN>	BOOLEAN[]	Hologres 0.8新增
ARRAY<varchar>	TEXT[]	Hologres 0.8新增

MySQL与Hologres数据类型映射

MySQL与Hologres数据类型映射如下表所示，关于MySQL迁移至Hologres的详情请参见迁移MySQL至Hologres。

MySQL数据类型	Hologres数据类型
BIGINT	BIGINT
BINARY(n)	BYTEA
BIT	BOOLEAN
• CHAR(n) • CHARACTER(n)	• CHAR(n) • CHARACTER(n)
DATE	DATE
DATETIME	TIMESTAMP [WITHOUT TIME ZONE]
• DECIMAL(p,s) • DEC(p,s)	• DECIMAL(p,s) • DEC(p,s)

MySQL数据类型	Hologres数据类型
DOUBLE	DOUBLE PRECISION
FLOAT	REAL
• INT • INTEGER	• INT • INTEGER
MEDIUMINT	INTEGER
NUMERIC(p,s)	NUMERIC(p,s)
SMALLINT	SMALLINT
• TINYBLOB • BLOB • MEDIUMBLOB • LONGBLOB	BYTEA
TINYINT	SMALLINT
• TINYTEXT • TEXT • MEDIUMTEXT • LONGTEXT	TEXT
TIME	TIME [WITHOUT TIME ZONE]
TIMESTAMP	TIMESTAMP [WITH TIME ZONE]
• VARBINARY(n) • VARBINARY(max)	BYTEA
VARCHAR(n)	VARCHAR(n)
VARCHAR(max)	TEXT

6.2.2. 自增序列Serial（Beta）

实时数仓Hologres兼容Postgres生态，支持在建表时将字段的类型定义为Serial和Bigserial来定义自增的字段。

数据类型简介

Hologres在建表时可以将字段的数据类型定义为Serial和Bigserial。其中，Serial表示INT4类型的自增字段，Bigserial表示INT8类型的自增字段。

配置Serial类型的自增序列。如下以Serial类型为例，对表格中的colname字段实现自增，Bigserial类型同样适用。

```
CREATE TABLE tablename (
    colname serial
);
```

参数说明

参数	说明	存储大小	取值范围
Serial	INT4类型的自增字段	4字节	-2147483648～2147483647
Bigserial	INT8类型的自增字段	8字节	-9223372036854775808～9223372036854775807

使用限制

- Hologres不能指定Serial和Bigserial的额外参数，包括递增步长及初始值参数，递增步长为1，初始值默认设置为 *1*。
- Hologres暂不支持Smallserial类型的数据。
- 首次创建Serial和Bigserial类型的表，需要Superuser在当前DB创建。例如Superuser执行 `create table test(a serial);` ，然后其余用户就能按照业务逻辑创建有关Seria和Bigserial类型的表。创建Serial和Bigserial类型的表是DB级别的，如果切换数据库， Superuser需要再次执行该操作。
- 仅支持使用 `insert into xxx select` 方式写入Serial和Bigserial类型。若是有Flink或者数据集成写入，仅Flink的JDBC模式和数据集成 `insert into` 模式支持写入Serial和Bigserial类型，DataHub暂不支持写入。
- 仅Hologres V0.10及以上版本支持修改Serial参数，且仅支持 `restart with` 参数的修改。
- Serial写入会有额外全局锁的开销，对写入性能影响较大，在性能敏感场景有限使用。

示例一：使用SQL语句实现自增序列

如下以Serial类型为例，使用SQL语句实现自增序列，Bigserial类型同样适用。

```
--创建表，包含id和f1两个字段。
create table if not exists test_tb(id serial primary key, f1 text);
--在insert语句中，给f1字段插入数据。
insert into test_tb(f1) values('1');
insert into test_tb(f1) values('2');
insert into test_tb(f1) values('3');
--查询test_tb表，按照id增序排列。
select * from test_tb order by id asc;
```

示例二：使用JDBC连接Hologres，实现自增序列

如下以Serial类型为例，使用JDBC连接Hologres实现自增序列，Bigserial类型同样适用。

```
package test;
import java.sql.*;
public class HoloSerial {
//创建表，包含id和f1两个字段。
    private static void Init(Connection conn) throws Exception {
        try (Statement stmt = conn.createStatement()) {
            stmt.execute("drop table if exists test_tb;");
            stmt.execute("create table if not exists test_tb(id serial primary key, f1 text
);");
        }
    }
//给f1字段插入数据。
    private static void TestSerial(Connection conn) throws Exception {
        try (PreparedStatement stmt = conn.prepareStatement("insert into test_tb(f1) values
(?)")) {
            for (int i = 0; i < 100; ++i) {
                stmt.setString(1, String.valueOf(i + 1));
                int affected_rows = stmt.executeUpdate();
                System.out.println("affected rows => " + affected_rows);
            }
        }
//查询test_tb表，按照id增序排列。
        try (PreparedStatement stmt = conn.prepareStatement("select * from test_tb order by
id asc")) {
            try(ResultSet rs = stmt.executeQuery()) {
                while(rs.next()) {
                    String res = rs.getObject(1).toString() + "\t" + rs.getObject(2).toStri
ng();
                    System.out.println(res);
                }
            }
        }
    }
//使用JDBC连接Hologres。
    public static void main(String[] args) throws Exception {
        Class.forName("org.postgresql.Driver").newInstance();
        String host = "127.0.0.1:13737";
        String db = "postgres";
        String user = "xx";
        String password = "xx";
        String url = "jdbc:postgresql://" + host + "/" + db;
        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            Init(conn);
            TestSerial(conn);
        }
    }
}
```

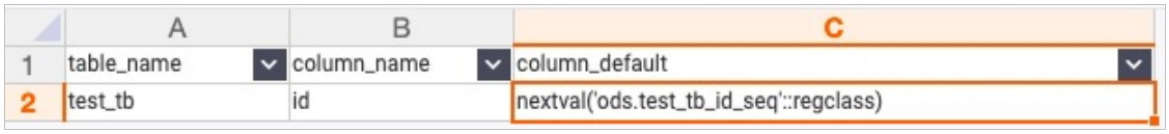
示例三：修改Serial参数

当您在创建表时使用Serial参数时，创建完成会自动生成一个名为**schema_name.tablename_columnname_seq**的Sequence，您可以通过修改Sequence参数控制Serial的生成方式。具体操作方法如下所示：

1. 执行如下代码查看生成的Sequence，其中，table_schematable_name和column_name的取值需要修改为实际的名称。本示例以示例一创建的表信息为例进行说明。

   ```
   select table_name,column_name,column_default  from information_schema.columns where table_schema='ods' and table_name = 'test_tb' and column_name = 'id';
   ```

   查询结果如下图所示：

	A	B	C
1	table_name	column_name	column_default
2	test_tb	id	nextval('ods.test_tb_id_seq'::regclass)

   查询结果中，引号部分的 `ods.test_tb_id_seq` 为Sequence名称。

2. 在获取到Sequence名称之后，您可以执行如下示例语句，对Serial的 `restart with` 参数进行修改。

   ```
   alter sequence ods.test_tb_id_seq restart with 100
   ```

   其中， `ods.test_tb_id_seq` 名称和具体数量您可以按照实际业务内容进行修改。修改完成后，您可以给表中插入数据进行验证。

常见问题

创建带Serial类型的表报错

- 问题现象

  创建带Serial类型的表时，出现如下报错提示。

  ```
  ERROR: permission denied schema hologres
  ```

- 问题原因

  superuser未创建一个Serial类型的示例表。

- 解决方法

  第一次在数据库创建Serial类型的表时，需要superuser先创建一个示例表，例如superuser执行 `create table test(a serial);` ，其余用户就能按照业务逻辑创建有关Seria和Bigserial类型的表。

6.2.3. JSON和JSONB类型

文将为你介绍Hologres中JSON和JSONB数据类型的语法和使用方法。

使用限制

Hologres支持JSON和JSONB两种JSON数据类型，在使用时需要注意的事项如下：

- 仅Hologres V0.9及以上版本支持JSON类型，如果您的实例是V0.9以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅Hologres V1.1及以上版本支持JSONB类型创建GIN索引，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- Hologres暂不支持的函数包括如下函数：json_each、jsonb_each、json_each_text、jsonb_each_text、json_extract_path、jsonb_extract_path、jsonb_each、jsonb_to_record。
- 仅Hologres V1.3及以上版本支持对JSONB类型进行列存优化，若您的实例是V1.3以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- JSONB的列存优化仅能用于列存表，行存表暂不支持，且至少1000条数据才会触发列存优化。

JSON和JSONB类型概述

JSON和JSONB区别如下。

- JSON储存的是文本格式的数据，JSONB储存的是Binary格式的数据。
- JSON插入速度快，查询速度慢，原因是处理函数必须在每次执行时重新解析该数据。JSONB插入速度慢，而查询速度快，原因是JSONB数据被存储在一种分解好的二进制格式中，因为需要做附加的转换，它在输入时要稍慢一些。但是JSONB在查询数据时快很多，因为不需要重新解析。
- JSON储存的数据是对数据的完整拷贝，会保留源数据的空格、重复键和顺序等，如果一个值中的JSON对象包含同一个键超过一次，所有的键、值对都会被保留。而JSONB在解析时会删除掉不必要的空格、重复键和数据的顺序等，如果在输入中指定了重复的键，只有最后一个值会被保留。

> **说明** 键值对的键必须使用双引号。

JSON和JSONB的操作符

JSON数据类型用来存储JSON数据，这种数据也可以被存储为TEXT类型。JSON数据类型的优势在于能强制要求每个被存储的值符合JSON规则，JSON支持的操作符使得其操作更为方便。

操作符	右操作数类型	描述	操作示例	执行结果
->	int	获得JSON数组元素（索引从0开始，负整数从末尾开始计）。	`select '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2`	`{"c":"baz"}`
->	text	通过键获得JSON对象域。	`select '{"a":{"b":"foo"}}'::json->'a'`	`{"b":"foo"}`
->>	int	以TEXT形式获得JSON数组元素。	`select '[1,2,3]'::json->>2`	`3`
->>	text	以TEXT形式获得JSON对象域。	`select '{"a":1,"b":2}'::json->>'b'`	`2`
#>	text[]	获取在指定路径的JSON对象。	`select '{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'`	`{"c":"foo"}`
#>>	text[]	以TEXT形式获取在指定路径的JSON对象。	`select '{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'`	`3`

额外的JSONB操作符

如下表格中为JSONB数据类型支持的操作符。

操作符	右操作数类型	描述	操作示例	执行结果
@>	jsonb	左侧的JSON值是否在顶层包含右侧的JSON路径或值。	`select '{"a":1,"b":2}'::jsonb @>'{"b":2}'::jsonb`	`true`

操作符	右操作数类型	描述	操作示例	执行结果
<@	jsonb	左侧的JSO 路径或值项是否被包含在右侧的JSON 值的顶层。	`select '{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb`	`true`
?	text	键或元素字符串是否存在于JSON值的顶层。	`select '{"a":1, "b":2}'::jsonb ? 'b'`	`true`
?|	text[]	数组字符串中的任何一个是否做为顶层键存在。	`select '{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']`	`true`
?&	text[]	是否所有数组字符串都作为顶层键存在。	`select '["a", "b"]'::jsonb ?& array['a', 'b']`	`true`
||	jsonb	将两个jsonb值串接成一个新的jsonb值。	`select '["a", "b"]'::jsonb || '["c", "d"]'::jsonb`	`["a", "b", "c", "d"]`
-	text	从左操作数删除键/值对或者string元素。键/值对基于它们的键值来匹配。	`select '{"a": "b"}'::jsonb - 'a'`	`{}`
-	text[]	从左操作数中删除多个键/值对或者string元素。键/值对基于它们的键值来匹配。	`select '{"a": "b", "c": "d"}'::jsonb - '{a,c}'::text[]`	`{}`
-	integer	删除具有指定索引（负值表示倒数）的数组元素。如果顶层容器不是数组则抛出一个错误。	`select '["a", "b"]'::jsonb - 1`	`["a"]`
#-	text[]	删除具有指定路径的域或者元素（对于JSON数组，负值表示倒数）。	`select '["a", {"b":1}]'::jsonb #- '{1,b}'`	`["a", {}]`

JSON创建函数

如下为可以用于创建JSON值的函数描述及操作示例。

函数	描述	操作示例	执行结果

函数	描述	操作示例	执行结果
to_json(anyelement)	此函数可以将该值返回为JSON。数组和组合会被（递归）转换成数组和对象，对于不是数组和组合的值，如果有从该类型到JSON的造型，造型函数将被用来执行该转换，否则将产生一个标量值。对于任何不是数字、布尔、空值的标量类型，将使用文本表达，使其是一个有效的JSON值。	`select to_json('Fred said "Hi."'::text)`	`"Fred said \"Hi.\""`
to_jsonb(anyelement)			
array_to_json(anyarray [, pretty_bool])	此函数可以将数组作为一个JSON数组返回。一个PostgreSQL多维数组会成为一个数组的JSON数组。如果*pretty_bool*为真，将在第1维度的元素之间增加换行。	`select array_to_json('{{1,5},{99,100}}'::int[])`	`[[1,5],[99,100]]`
json_build_array(VARIADIC "any")	此函数可以从一个可变参数列表构造一个可能包含异质类型的JSON数组。	`select json_build_array(1,2,'3',4,5)`	`[1, 2, "3", 4, 5]`
jsonb_build_array(VARIADIC "any")			
json_build_object(VARIADIC "any")	此函数可以从一个可变参数列表构造一个JSON对象。通过转换，该参数列表由交替出现的键和值构成。	`select json_build_object('foo',1,'bar',2)`	`{"foo": 1, "bar": 2}`
jsonb_build_object(VARIADIC "any")			
json_object(text[])	此函数可以从一个文本数组构造一个JSON对象。该数组必须可以是具有偶数个成员的一维数组（成员被当做交替出现的键/值对），或者是一个二维数组（每一个内部数组刚好有2个元素，可以被看做是键/值对）。	`select json_object('{a, 1, b, "def", c, 3.5}');`	`{"a": "1", "b": "def", "c": "3.5"}`
jsonb_object(text[])		`select jsonb_object('{a, 1, b, "def", c, 3.5}');`	`{"a": "1", "b": "def", "c": "3.5"}`
json_object(keys text[], values text[])	json_object的这种形式从两个独立的数组得到键/值对。在其他方面和一个参数的形式相同。	`select json_object('{a, b}', '{1,2}')`	`{"a": "1", "b": "2"}`
jsonb_object(keys text[], values text[])			

JSON处理函数

如下为可以用于处理JSON值的函数描述及操作示例。

函数	返回值	描述	操作示例	执行结果
json_array_length(json) jsonb_array_length(jsonb)	int	返回最外层JSON数组中的元素数量。	`select json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]')`	`5`

json_object_keys(json)
jsonb_object_keys(jsonb)
	setof text	返回最外层JSON对象中的键集合。	`select json_object_keys('{"f1":"abc","f2":{"f3":"a","f4":"b"}}')`	`json_object_keys`
				`---`
				`f1`
				`f2`

json_populate_record(base anyelement, from_json json)
jsonb_populate_record(base anyelement, from_json jsonb)
	anyelement	扩展from_json中的对象成一个行，它的列匹配由base定义的记录类型。	`begin;`
			`create table myrowtype( a text, b text, c text);commit;`
			`select * from json_populate_record(null::myrowtype, '{"a": 1, "b": ["2", "a b"], "c": {"d": 4, "e": "a b c"}}')`
				`a | b | c`
				`---+---+---`
				`1 | {2,"a b"} | {"d": 4, "e": "a b c"}`

json_populate_recordset(base anyelement, from_json json)
jsonb_populate_recordset(base anyelement, from_json jsonb)
	setof anyelement	扩展from_json中最外的对象数组为一个集合，该集合的列匹配由base定义的记录类型。	`begin;`
			`create table myrowtype(a text,b text);`
			`commit;`
			`select * from json_populate_recordset(null::myrowtype, '[{"a":1,"b":2},{"a":3,"b":4}]')`
				`a | b`
				`---+---`
				`1 | 2`
				`3 | 4`

函数	返回值	描述	操作示例	执行结果
json_array_elements(json)	setof json	把一个JSON数组扩展成一个JSON值的集合。	select * from json_array_elements('[1,true, [2,false]]')	value ------- 1 true [2,false]
jsonb_array_elements(jsonb)	setof jsonb			
json_array_elements_text(json)	setof text	把一个JSON数组扩展成一个text值集合。	select * from json_array_elements_text('["foo", "bar"]')	value ------- foo bar
jsonb_array_elements_text(jsonb)				
json_typeof(json)	text	把最外层的JSON值的类型作为一个文本字符串返回。可能的类型是：object、array、string、number、 boolean以及null。	select json_typeof('-123.4')	number
jsonb_typeof(jsonb)				
json_strip_nulls(from_json json)	json	返回from_json，其中所有具有空值的对象域都被省略。其他空值不动。	select json_strip_nulls('[{"f1":1,"f2":null},2,null,3]')	[{"f1":1},2,null,3]
jsonb_strip_nulls(from_json jsonb)	jsonb			
jsonb_set(target jsonb, path text[], new_value jsonb[,create_missing boolean])	jsonb	返回target，其中由path指定的节用new_value替换，如果path指定的项不存在并且create_missing为真（默认为 true）则加上new_value。正如面向路径的操作符一样，出现在path中的负整数表示从JSON数组的末尾开始数。	select jsonb_set('[{"f1":1,"f2":null},2,null,3]', '{0,f1}','[2,3,4]', false);	[{"f1": [2,3,4],"f2":null},2,null,3]
			select jsonb_set('[{"f1":1,"f2":null},2]', '{0,f3}','[2,3,4]')	[{"f1": 1, "f2": null, "f3": [2, 3, 4]}, 2]

函数	返回值	描述	操作示例	执行结果
jsonb_insert(target jsonb, path text[], new_value jsonb, [insert_after boolean])	jsonb	返回被插入了new_value的target。如果path指定的target节在一个JSONB数组中，new_value将被插入到目标之前（insert_after为false，默认情况）或者之后（insert_after为真）。如果path指定的target节在一个JSONB对象内，则只有当target不存在时才插入new_value。对于面向路径的操作符来说，出现在path中的负整数表示从JSON数组的末尾开始计数。	`select jsonb_insert('{"a": [0,1,2]}', '{a, 1}', '"new_value"')`	`{"a": [0, "new_value", 1, 2]}`
			`select jsonb_insert('{"a": [0,1,2]}', '{a, 1}', '"new_value"', true)`	`{"a": [0, 1, "new_value", 2]}`
jsonb_pretty(from_json jsonb)	text	把from_json返回成一段缩进后的JSON文本。	`select jsonb_pretty('[{"f1":1,"f2":null},2,null,3]')`	`[ { "f1": 1, "f2": null }, 2, null, 3 ]`

函数	返回值	描述	操作示例	执行结果

jsonb_agg	jsonb	将值（包括空值）聚合为JSON数组。	DROP TABLE IF EXISTS t;
			CREATE TABLE t (
			k int PRIMARY KEY,
			class int NOT NULL,
			v text NOT NULL
			);
			INSERT INTO t (k, class, v)
			SELECT
			(1 + s.v),
			CASE (s.v) < 3
			WHEN TRUE THEN
			1
			ELSE
			2
			END,
			chr(97 + s.v)
			FROM
			generate_series(0, 5) AS s (v);
			SELECT
			class,
			jsonb_agg(v ORDER BY v DESC) FILTER (WHERE v <> 'b') AS "jsonb_agg",
			jsonb_object_agg(v, k ORDER BY v DESC) FILTER (WHERE v <> 'e') AS "jsonb_object_agg(v, k)"
			FROM
			t
			GROUP BY
			class;
				class | jsonb_agg | jsonb_object_agg(v, k)
				---+---+---
				1 | ["c", "a"] | {"a": 1, "b": 2, "c": 3}
				2 | ["f", "e", "d"] | {"d": 4, "f": 6}

函数	返回值	描述	操作示例	执行结果

函数	返回值	描述	操作示例	执行结果

jsonb_object_agg	jsonb	将Key/Value对聚合为JSON对象，值可以为空，但名称不能为空。	DROP TABLE IF EXISTS t;
			CREATE TABLE t (
			k int PRIMARY KEY,
			class int NOT NULL,
			v text NOT NULL
			);
			INSERT INTO t (k, class, v)
			SELECT
			(1 + s.v),
			CASE (s.v) < 3
			WHEN TRUE THEN
			1
			ELSE
			2
			END,
			chr(97 + s.v)
			FROM
			generate_series(0, 5) AS s (v);
			SELECT
			class,
			jsonb_agg(v ORDER BY v DESC) FILTER (WHERE v <> 'b') AS "jsonb_agg",
			jsonb_object_agg(v, k ORDER BY v DESC) FILTER (WHERE v <> 'e') AS "jsonb_object_agg(v, k)"
			FROM
			t
			GROUP BY
			class;
				class | jsonb_agg | jsonb_object_agg(v, k)
				-------+-------------+-----------------------
				1 | ["c", "a"] | {"a": 1, "b": 2, "c": 3}
				2 | ["f", "e", "d"] | {"d": 4, "f": 6}

更多关于JSON类型的用法，请参见JSON函数和操作符。

JSONB索引

从Hologres V1.1版本开始，支持JSONB索引，如下为您介绍在JSONB类型字段上建GIN索引的语法解释和使用示例。

- 语法解释

  JSONB类型支持GIN，BTree索引。一般情况下，会在JSONB类型字段上建GIN索引，建GIN索引存在三种方式，语法分别如下。

  - 使用默认的jsonb_ops操作符创建索引

    ```
    CREATE INDEX idx_name ON table_name USING gin (idx_col);
    ```

  - 使用jsonb_path_ops操作符创建索引

    ```
    CREATE INDEX idx_name ON table_name USING gin (idx_col jsonb_path_ops);
    ```

  - 使用jsonb_holo_path_ops操作符创建索引

    ```
    CREATE INDEX idx_name ON table_name USING gin (idx_col jsonb_holo_path_ops);
    ```

  三种方式的区别为：在jsonb_ops的GIN索引中，JSONB数据中的每个key和value都是作为一个单独的索引项的；而jsonb_path_ops则只为每个value创建一个索引项；jsonb_holo_path_ops为Hologres全新的操作符号，可以省去检索数据后recheck的动作。

- 使用示例
  - 原生PostgreSQL操作符
    - 创建jsonb_ops操作符号索引

      a. 创建表的SQL语句如下。

      ```
      DROP TABLE IF EXISTS json_table;
      CREATE TABLE IF NOT EXISTS json_table
      (
          id INT
          ,j jsonb
      );
      ```

      b. 创建jsonb_ops操作符号索引的SQL语句如下。

      ```
      CREATE INDEX index_json on json_table USING GIN(j);
      ```

      c. 在表中插入数据的SQL语句如下。

      ```
      INSERT INTO json_table VALUES
      (1, '{"key1": 1, "key2": [1, 2], "key3": {"a": "b"}}') ,
      (1, '{"key1": 1}'),
      (2, '{"key2": [1, 2], "key3": {"a": "b"}}') ;
      ```

      d. 筛选包含数据的SQL语句如下。

      ```
      SELECT  *
      FROM    json_table
      WHERE   j ? 'key1'
      ;
      ```

e. 执行结果如下。

```
 id |                         j
----+-----------------------------------------------
  1 | {"key1": 1, "key2": [1, 2], "key3": {"a": "b"}}
  1 | "key1"
(2 rows)
```

f. 使用 `explain` 命令查看执行计划如下。

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------
----------
 Gather  (cost=0.00..12.36 rows=1 width=66)
   ->  Exchange (Gather Exchange)  (cost=0.00..12.36 rows=1 width=66)
         ->  Decode  (cost=0.00..12.36 rows=1 width=66)
               ->  Bitmap Heap Scan on json_table  (cost=0.00..12.26 rows=1 width
=66)
                     Recheck Cond: (j ? 'key1'::text)
                     ->  Bitmap Index Scan on index_json  (cost=0.00..0.00 rows=0
width=0)
                           Index Cond: (j ? 'key1'::text)
 Optimizer: HQO version 1.1.0
(8 rows)
```

执行计划中出现了 `Index Scan` 步骤，表明查询过程使用了索引。

- 创建jsonb_path_ops操作符号索引
  a.
  b. 创建jsonb_path_ops操作符号索引的SQL语句如下。

```
CREATE INDEX index_json on json_table USING GIN(j jsonb_path_ops);
```

  c. 插入数据的SQL语句如下。

```
INSERT INTO json_table (
    SELECT
        i,
        ('{
            "key1": "'||i||'"
            ,"key2": "'||i%100||'"
            ,"key3": "'||i%1000 ||'"
            ,"key4": "'||i%10000||'"
            ,"key5": "'||i%100000||'"
        }')::jsonb
    FROM generate_series(1, 1000000) i
) ;
```

  d. 筛选包含'{"key1": "10"}'数据的SQL语句如下。

```
SELECT  *
FROM    json_table
WHERE   j @> '{"key1": "10"}'::JSONB
;
```

  e. 执行结果如下。

```
 id |                                   j
----+-------------------------------------------------------------------------
 10 | {"key1": "10", "key2": "10", "key3": "10", "key4": "10", "key5": "10"}
(1 row)
```

  f. 使用 `explain` 命令查看执行计划如下。

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------
----------
 Gather  (cost=0.00..0.26 rows=1000 width=12)
   ->  Exchange (Gather Exchange)  (cost=0.00..0.23 rows=1000 width=12)
         ->  Decode  (cost=0.00..0.23 rows=1000 width=12)
               ->  Bitmap Heap Scan on json_table  (cost=0.00..0.13 rows=1000 wid
th=12)
                     Recheck Cond: (j @> '{"key1": "10"}'::jsonb)
                     ->  Bitmap Index Scan on index_json  (cost=0.00..0.00 rows=0
width=0)
                           Index Cond: (j @> '{"key1": "10"}'::jsonb)
 Optimizer: HQO version 1.1.0
(8 rows)
```

  执行计划中出现了 `Index Scan` 步骤，表明查询过程使用了索引。

- Hologres操作符

由于原生PostgreSQL的JSOB的GIN索引是非精确的索引，所以检索数据后需要进行recheck动作。最终导致创建索引后性能不一定提升。针对上述情况，Hologres实现了一种新的ops_class，可以省去recheck的动作，且若不指定索引操作符，系统会默认使用该操作符，具体使用方式如下。

其中jsonb_holo_ops对应jsonb_ops，支持 `?, ?|, ?&, @>` 的过滤操作。其中jsonb_holo_path_ops对应jsonb_path_ops，仅支持 `@>` 的过滤操作。

- 创建jsonb_holo_ops操作符号索引
  a. 创建表的SQL语句如下。

```
DROP TABLE IF EXISTS json_table;
CREATE TABLE IF NOT EXISTS json_table
(
    id INT
    ,j jsonb
);
```

  b. 创建jsonb_holo_ops操作符号索引的SQL语句如下。

```
-- 创建索引，使用jsonb_holo_ops操作符
CREATE INDEX index_json on json_table USING GIN(j jsonb_holo_ops);
```

  c. 插入数据的SQL语句如下。

```
INSERT INTO json_table VALUES
(1, '{"key1": 1, "key2": [1, 2], "key3": {"a": "b"}}') ,
(1, '{"key1": 1}'),
(2, '{"key2": [1, 2], "key3": {"a": "b"}}') ;
```

  d. 筛选包含数据的SQL语句如下。

```
SELECT  *
FROM    json_table
WHERE   j ? 'key1'
;
```

  e. 执行结果如下。

```
 id |                         j
----+---------------------------------------------------
  1 | {"key1": 1, "key2": [1, 2], "key3": {"a": "b"}}
  1 | {"key1": 1}
(2 rows)
```

- 创建jsonb_holo_path_ops操作符号索引
  a.
  b. 创建jsonb_holo_path_ops操作符号索引的SQL语句如下。

```
CREATE INDEX index_json on json_table USING GIN(j jsonb_holo_path_ops);
```

c. 插入数据的SQL语句如下。

```
INSERT INTO json_table (
    SELECT
        i,
        ('{
            "key1": "'||i||'"
            ,"key2": "'||i%100||'"
            ,"key3": "'||i%1000 ||'"
            ,"key4": "'||i%10000||'"
            ,"key5": "'||i%100000||'"
        }')::jsonb
    FROM generate_series(1, 1000000) i
) ;
```

d. 筛选包含'{"key1": "10"}'数据的SQL语句如下。

```
SELECT  *
FROM    json_table
WHERE   j @> '{"key1": "10"}'::JSONB
;
```

e. 执行结果如下。

```
 id |                                   j
----+-----------------------------------------------------------------------
 10 | {"key1": "10", "key2": "10", "key3": "10", "key4": "10", "key5": "10"}
(1 row)
```

f. 使用 `explain` 命令查看执行计划如下。

```
QUERY PLAN
Gather  (cost=0.00..39038928.99 rows=400000 width=88)
  ->  Exchange (Gather Exchange)  (cost=0.00..39038843.49 rows=400000 width=88)
        ->  Decode  (cost=0.00..39038843.37 rows=400000 width=88)
              ->  Bitmap Heap Scan on json_table  (cost=0.00..39038840.00 rows=40
0000 width=88)
"                    Recheck Cond: (j @> '{"key1": "10"}'::jsonb)"
                   ->  Bitmap Index Scan on index_json  (cost=0.00..0.00 rows=0
width=0)
"                          Index Cond: (j @> '{"key1": "10"}'::jsonb)"
Optimizer: HQO version 0.10.0
```

执行计划中出现了 `Index Scan` 步骤，表明查询过程使用了索引。

JSONB列式存储优化

从Hologres V1.3版本开始，支持对于JSONB类型开启列存优化，能够减少JSONB的存储并加速查询，具体使用方法如下。

- 语法解释

- 打开JSONB列存优化

```
-- 打开xx表的xx列的JSONB列式存储优化
ALTER TABLE <table_name> ALTER COLUMN <column_name> SET (enable_columnar_type = ON);
```

table_name为表名称；column_name为列名称。

- 关闭JSONB列存优化

```
-- 关闭xx表的xx列的JSONB列式存储优化
ALTER TABLE <table_name> ALTER COLUMN <column_name> SET (enable_columnar_type = OFF);
```

table_name为表名称；column_name为列名称。

- 查看某张表的配置情况

```
SELECT DISTINCT
    a.attnum as num,
    a.attname as name,
    format_type(a.atttypid, a.atttypmod) as type,
    a.attnotnull as notnull,
    com.description as comment,
    coalesce(i.indisprimary,false) as primary_key,
    def.adsrc as default,
    a.attoptions
FROM pg_attribute a
JOIN pg_class pgc ON pgc.oid = a.attrelid
LEFT JOIN pg_index i ON
    (pgc.oid = i.indrelid AND i.indkey[0] = a.attnum)
LEFT JOIN pg_description com on
    (pgc.oid = com.objoid AND a.attnum = com.objsubid)
LEFT JOIN pg_attrdef def ON
    (a.attrelid = def.adrelid AND a.attnum = def.adnum)
WHERE a.attnum > 0 AND pgc.oid = a.attrelid
AND pg_table_is_visible(pgc.oid)
AND NOT a.attisdropped
AND pgc.relname = '<table_name>'  -- 表名称
ORDER BY a.attnum;
```

table_name为表名称。

- 使用示例

i. 创建表SQL命令如下。

```
DROP TABLE IF EXISTS user_tags;
-- 创建数据表
BEGIN;
CREATE TABLE IF NOT EXISTS user_tags (
    ds timestamptz,
    tags jsonb
);
COMMIT;
```

ii. 打开JSONB列存优化SQL命令如下。

```
-- 打开tags列的JSONB列存优化
ALTER TABLE user_tags ALTER COLUMN tags SET (enable_columnar_type = ON);
```

iii. 查看是否设置成功

- 查看配置命令如下。

```
SELECT DISTINCT
    a.attnum as num,
    a.attname as name,
    format_type(a.atttypid, a.atttypmod) as type,
    a.attnotnull as notnull,
    com.description as comment,
    coalesce(i.indisprimary,false) as primary_key,
    def.adsrc as default,
    a.attoptions
FROM pg_attribute a
JOIN pg_class pgc ON pgc.oid = a.attrelid
LEFT JOIN pg_index i ON
    (pgc.oid = i.indrelid AND i.indkey[0] = a.attnum)
LEFT JOIN pg_description com on
    (pgc.oid = com.objoid AND a.attnum = com.objsubid)
LEFT JOIN pg_attrdef def ON
    (a.attrelid = def.adrelid AND a.attnum = def.adnum)
WHERE a.attnum > 0 AND pgc.oid = a.attrelid
AND pg_table_is_visible(pgc.oid)
AND NOT a.attisdropped
AND pgc.relname = 'user_tags'  -- 表名
ORDER BY a.attnum;
```

- 结果

  可以看到tags列的attoptions属性是 `enable_columnar_type = on` ，表示已经配置成功。

num	name	type	notnull	comment	primary_key	default	attoptions
1	ds	timestamp with time z	f	\N	f	\N	\N
2	tags	jsonb	f	\N	f	\N	{enable_columnar_type=on}

iv. 导入数据SQL命令如下。

```
INSERT INTO user_tags (ds, tags)
SELECT
    '2022-01-01 00:00:00+08'
    , ('{"id":' || i || ',"first_name" :"Sig",  "gender" :"Male"}')::jsonb
FROM
    generate_series(1, 1001) i;
```

v. （可选）强制触发数据落盘

写入数据后，系统会在数据落盘时进行JSONB的列存优化，为了尽快看到效果，此处使用如下后台命令，强制触发数据落盘。

```
SELECT hg_admin_command ('flush', 'table_name=user_tags');
VACUUM user_tags;
```

vi. 样例查询

使用如下SQL查询id为10的first_name。

```
SELECT
    (tags -> 'first_name')::text AS first_name
FROM
    user_tags
WHERE (tags -> 'id')::int = 10;
```

vii. 查看列存优化是否生效

- 通过如下命令查看执行计划。

```
-- 显示详细的统计信息
SET hg_experimental_show_execution_statistics_in_explain = ON;
-- 查看执行计划
EXPLAIN ANALYZE
SELECT
    (tags -> 'first_name')::text AS first_name
FROM
    user_tags
WHERE (tags -> 'id')::int = 10;
```

- 结果

  执行计划如下所示，其中使用了 `lazy_evaluated` ，表示JSONB列存优化生效。

```
QUERY PLAN
  "table_bytes_read": "71396",
  "table_bytes_accessed": "71396",
  "count_cache_lookup_cost": "2",
  "avg_cache_lookup_cost": 1.5,
  "min_cache_lookup_cost": "1",
  "max_cache_lookup_cost": "2",
  "min_cache_insert_cost": "18446744073709551615",
  "start_open_timestamp": "1649990120816",
  "end_open_timestamp": "1649990120819",
  "operator_case": 101,
  "scanned_rows": 1001,
  "lazy_evaluated": 2
 },
 {
  "id": "3",
  "count": 20,
```

6.3. PostgreSQL兼容函数

6.3.1. 查看表和DB的存储大小

Hologres兼容PostgreSQL，当前支持查看表或者DB的存储大小。本文将会为您介绍如何使用SQL语句查看表和DB的存储大小。

注意事项

在Hologres中查看表和DB的存储大小需要注意如下事项：

- 在查看表和DB的存储规格之前，您需要开通Hologres实例并连接开发工具，操作示例请参见连接HoloWeb。
- 仅Hologres V0.9及以上版本支持查看对象的存储大小，如果您的实例是V0.9以下版本，请您提交工单升级实例。

查看表的存储大小

- 使用说明

  当前仅支持查看内部表的存储规格大小，不支持直接查看分区父表（查看分区父表返回结果是0），需要指定分区子表进行查看。如果查询到表的规格大小是0，则返回空值。

- 函数语法

```
select pg_relation_size('table_name');--返回单位是Byte
```

- 参数说明

参数	说明
table_name	表示待查询的当前数据库下的表名称。

- 返回值：返回值的单位是 Byte，类型为字符串。返回的数据为该表此刻的内存所占空间和物理磁盘空间。

  如果您需要提高可读性，可以使用pg_size_pretty函数进行查询，具体语法如下：

```
--查看单表存储
SELECT pg_size_pretty(pg_relation_size('table_name'));
--查看所有表大小
SELECT table_schema || '.' || table_name AS table_full_name,
pg_size_pretty(pg_relation_size('"' || table_schema || '"."' || table_name || '"')) AS ta
ble_size,
pg_relation_size('"' || table_schema || '"."' || table_name || '"') AS  order_size
FROM information_schema.tables
WHERE table_schema NOT IN ('pg_catalog','information_schema','hologres')
ORDER BY order_size DESC;
```

查看schema存储大小

- 使用说明

  您可以通过执行SQL语句查看对应schema下面的所有表的大小。

- 函数语法

```
SELECT table_schema, pg_size_pretty(SUM(pg_relation_size( table_schema  || '.' || table_n
ame)::decimal)) AS schema_size
FROM information_schema.tables
WHERE table_schema = 'schema_name'
GROUP BY table_schema;
```

- 参数说明

参数	说明
schema_name	表示当前表所对应的schema名称。

- 返回值：返回值的单位是 Byte。

查看DB的存储大小

- 使用说明

  仅支持查看当前连接DB和该DB下内部表的存储规格大小。

- 函数语法

```
select pg_database_size(current_database()); --返回单位是Byte
```

- 参数说明

参数	说明
current_database	指代当前DB。您无需替换参数，直接执行函数命令语句即可查询当前DB的存储规格大小。

- 返回值：返回值的单位是 Byte。返回的数据为指定DB下面所有Hologres表的大小和DB下面产生的WAL（Write-Ahead Log）日志大小。

  如果您需要提高可读性，可以使用pg_size_pretty函数进行查询，具体语法如下：

```
select pg_size_pretty(pg_database_size(current_database())); --返回单位是KB或者MB等单位
```

6.3.2. 有序聚集函数

交互式分析（Hologres）兼容PostgreSQL，使用标准的PostgreSQL语法开发。

当前Hologres版本支持的函数是PostgreSQL的一个子集，Hologres已支持的有序聚集函数列表如下。

函数	描述	直接参数类型	聚集参数类型	示例	返回类型

`mode() WITHIN GROUP (ORDER BY sort_expression)`	返回最频繁的输入值。如果有多个频度相同的值，则返回第一个。	无	任何可排序类型。	• 示例
				`select mode() WITHIN GROUP (ORDER BY user_id ) from testtable;`
				• 结果
				`293890`
					与排序表达式相同。

函数	描述	直接参数类型	聚集参数类型	示例	返回类型

`percentile_cont(fraction) WITHIN GROUP (ORDER BY sort_expression)`	连续百分率：返回一个对应于排序中指定分数的值，如有必要就在相邻的输入项之间插值。	`double precision`	`double precision` 或者 `interval`	• 示例
				`select percentile_cont(0.5) WITHIN GROUP (ORDER BY cust_id) from testtable;`
				• 结果
				`1105639996.5`
					与排序表达式相同

`percentile_cont(fractions) WITHIN GROUP (ORDER BY sort_expression)`	多重连续百分率：返回一个匹配fractions参数形状的结果数组，其中每一个非空元素都用对应于那个百分率的值替换。	`double precision[]`	`double precision` 或者 `interval`	• 示例
				`select percentile_cont(0.5) WITHIN GROUP (ORDER BY member_id) from testtable;`
				• 结果
				`96727903.5`
					排序表达式类型的数组。

`percentile_disc(fraction) WITHIN GROUP (ORDER BY sort_expression)`	离散百分率：返回第一个在排序中位置等于或者超过指定分数的输入值。	`double precision`	一种可排序类型。	• 示例
				`select percentile_disc(0.6) WITHIN GROUP (ORDER BY (impression_id) from testtable;`
				• 结果
				`0.0`
					与排序表达式相同

`percentile_disc(fractions) WITHIN GROUP (ORDER BY sort_expression)`	多重离散百分率：返回一个匹配fractions参数形状的结果数组，其中每一个非空元素都用对应于那个百分率的输入值替换。	`double precision[]`	任何可排序类型。	• 示例
				`select percentile_disc(0.6) WITHIN GROUP (ORDER BY (impr_id) from testtable;`
				• 结果
				`0.0`
					排序表达式类型的数组。

6.3.3. 数学函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。本文为您介绍Hologres已支持的数学函数列表及使用用例。

Hologres已支持的数学函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见数学函数。

函数名	描述	用例	结果
abs(bigint)	返回BIGINT类型表达式的绝对值。	abs(-17)	17
abs(int)	返回INT类型表达式的绝对值。	abs(-17)	17
abs(float8)	返回FLOAT8类型表达式的绝对值。	abs(-17.4)	17.4
abs(float4)	返回FLOAT4类型表达式的绝对值。	abs(-17.4)	17.4
abs(numeric)	返回NUMERIC类型表达式的绝对值。	abs(-17.4)	17.4
cbrt(dp)	返回DP类型表达式的立方根。	cbrt(27.0)	3.0000000000000004
ceil(dp)	对DP类型的表达式向上取整。	ceil(-42.8)	-42.0
ceil(numeric)	对NUMERIC类型的表达式向上取整。	ceil(-42.8)	-42.0
ceiling(dp)	对DP类型的表达式向上取整。	ceil(-42.8)	-42.0
ceiling(numeric)	对NUMERIC类型的表达式向上取整。	ceil(-42.8)	-42.0
degrees(dp)	将DP类型表达式的弧度转换为角度。	degrees(0.5)	28.64788975654116
exp(dp)	返回DP类型表达式的指数。	exp(1.0)	2.718281828459045
exp(numeric)	返回NUMERIC类型表达式的指数。	exp(1.0)	2.718281828459045
floor(dp)	对DP类型表达式向下取整。	floor(-42.8)	-43.0
floor(numeric)	对NUMERIC类型表达式向下取整。	floor(-42.8)	-43.0
ln(dp)	返回DP类型表达式的自然对数。	ln(2.0)	0.6931471805599453

函数名	描述	用例	结果
ln(numeric)	返回NUMERIC类型表达式的自然对数。	ln(2.0)	0.6931471805599453
log(dp)	返回DP类型表达式的常用对数。	log(100.0)	2.0
log(numeric)	返回NUMERIC类型表达式的常用对数。	log(100.0)	2.0
log(b numeric, x numeric)	返回NUMERIC类型表达式的对数。	log(2.0, 64.0)	6.0
mod(bigint, x)	求BIGINT类型表达式除以x的余数。	mod(9,4)	1
mod(int, x)	求INT类型表达式除以x的余数。	mod(9,4)	1
pi()	返回π常量。	pi()	3.141592653589793
power(a dp, b dp)	求a的b次幂，a和b使用DP类型的表达式。	power(9.0, 3.0)	729.0
power(a numeric, b numeric)	求a的b次幂，a和b使用NUMERIC类型的表达式。	power(9.0, 3.0)	729.0
radians(dp)	将DP类型表达式的角度转换为弧度。	radians(45.0)	0.7853981633974483
round(dp)	返回DP类型表达式四舍五入后的整数值。	round(42.4)	42.0
round(numeric)	返回NUMERIC类型表达式四舍五入后的整数值。	round(42.4)	42.0
round(v numeric, s int)	保留小数位数字到s位。	round(42.4382, 2)	42.44
sign(dp)	返回DP类型表达式的符号。参数值大于0返回*1*，小于0返回 *-1* ，等于0返回*0*。	sign(-8.4)	-1
sign(numeric)	返回NUMERIC类型表达式的符号。参数值大于0返回*1*，小于0返回 *-1* ，等于0返回*0*。	sign(-8.4)	-1
sqrt(dp)	返回DP类型表达式的平方根。	sqrt(2.0)	1.414213562373095

函数名	描述	用例	结果
sqrt(numeric)	返回NUMERIC类型表达式的平方根。	sqrt(2.0)	1.414213562373095
trunc(dp)	去掉DP类型表达式的小数位。	trunc(42.8)	42.0
trunc(numeric)	去掉NUMERIC类型表达式的小数位。	trunc(42.8)	42.0
trunc(v numeric, s int)	截断NUMERIC类型表达式的小数位置到s位。	trunc(42.4382, 2)	42.43
width_bucket(operand numeric, b1 numeric, ...)	返回OPERAND在BUCKET中的位置。	width_bucket(5.35, 0.024, 10.06, 5)	3
random()	获取一个随机数，返回值范围为[0.0,1.0)。	random()	0.3977345246821642

6.3.4. 三角函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的三角函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见三角函数。

函数名	描述	用例	结果
acos(bigint)	返回BIGINT类型表达式的反余弦值。	acos(1)	0.0
acos(int)	返回INT类型表达式的反余弦值。	acos(1)	0.0
acos(float8)	返回FLOAT8类型表达式的反余弦值。	acos(0.9)	0.45102681179626236
acos(float4)	返回FLOAT4类型表达式的反余弦值。	acos(0.9)	0.45102681179626236
acos(numeric)	返回NUMERIC类型表达式的反余弦值。	acos(0.9)	0.45102681179626236
asin(bigint)	返回BIGINT类型表达式的反正弦值。	asin(1)	1.5707963267948966
asin(int)	返回INT类型表达式的反正弦值。	asin(1)	1.5707963267948966
asin(float8)	返回FLOAT8类型表达式的反正弦值。	asin(1.0)	1.5707963267948966

函数名	描述	用例	结果
asin(float4)	返回FLOAT4类型表达式的反正弦值。	asin(1.0)	1.5707963267948966
asin(numeric)	返回NUMERIC类型表达式的反正弦值。	asin(1.0)	1.5707963267948966
atan(bigint)	返回BIGINT类型表达式的反正切值。	atan(2)	1.1071487177940904
atan(int)	返回INT类型表达式的反正切值。	atan(2)	1.1071487177940904
atan(float8)	返回FLOAT8类型表达式的反正切值。	atan(2.0)	1.1071487177940904
atan(float4)	返回FLOAT4类型表达式的反正切值。	atan(2.0)	1.1071487177940904
atan(numeric)	返回NUMERIC类型表达式的反正切值。	atan(2.0)	1.1071487177940904
atan2(bigint y, bigint x)	取y/x的反正切值。y和x的数据类表达式为BIGINT。	atan2(2, 1)	1.1071487177940904
atan2(int y, int x)	取y/x的反正切值。y和x的数据类表达式为INT。	atan2(2, 1)	1.1071487177940904
atan2(float8 y, float8 x)	取y/x的反正切值。y和x的数据类表达式为FLOAT8。	atan2(2.0, 1.0)	1.1071487177940904
atan2(float4 y, float4 x)	取y/x的反正切值。y和x的数据类表达式为FLOAT4。	atan2(2.0, 1.0)	1.1071487177940904
atan2(numeric y, numeric x)	取y/x的反正切值。y和x的数据类表达式为NUMERIC。	atan2(2.0, 1.0)	1.1071487177940904
cos(bigint)	返回BIGINT类型表达式的余弦值。	cos(2)	-0.4161468365471424
cos(int)	返回INT类型表达式的余弦值。	cos(2)	-0.4161468365471424
cos(float8)	返回FLOAT8类型表达式的余弦值。	cos(2.0)	-0.4161468365471424
cos(float4)	返回FLOAT4类型表达式的余弦值。	cos(2.0)	-0.4161468365471424

函数名	描述	用例	结果
cos(numeric)	返回NUMERIC类型表达式的余弦值。	cos(2.0)	-0.4161468365471424
cot(bigint)	返回BIGINT类型表达式的余切值。	cot(2)	-0.45765755436028577
cot(int)	返回INT类型表达式的余切值。	cot(2)	-0.45765755436028577
cot(float8)	返回FLOAT8类型表达式的余切值。	cot(2.0)	-0.45765755436028577
cot(float4)	返回FLOAT4类型表达式的余切值。	cot(2.0)	-0.45765755436028577
cot(numeric)	返回NUMERIC类型表达式的余切值。	cot(2.0)	-0.45765755436028577
sin(bigint)	返回BIGINT类型表达式的正弦值。	sin(2)	0.9092974268256817
sin(int)	返回INT类型表达式的正弦值。	sin(2)	0.9092974268256817
sin(float8)	返回FLOAT8类型表达式的正弦值。	sin(2.0)	0.9092974268256817
sin(float4)	返回FLOAT4类型表达式的正弦值。	sin(2.0)	0.9092974268256817
sin(numeric)	返回NUMERIC类型表达式的正弦值。	sin(2.0)	0.9092974268256817
tan(bigint)	返回BIGINT类型表达式的正切值。	tan(2)	-2.185039863261519
tan(int)	返回INT类型表达式的正切值。	tan(2)	-2.185039863261519
tan(float8)	返回FLOAT8类型表达式的正切值。	tan(2.0)	-2.185039863261519
tan(float4)	返回FLOAT4类型表达式的正切值。	tan(2.0)	-2.185039863261519
tan(numeric)	返回NUMERIC类型表达式的正切值。	tan(2.0)	-2.185039863261519

6.3.5. 字符串函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。本文为您介绍Hologres已支持的字符串函数列表及使用用例。

常见字符串函数

Hologres已支持的字符串函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见字符串函数。

函数名	描述	用例	结果
string || string	连接两个字符串。	'Holo' || 'greSQL'	HologreSQL
bit_length(string)	获取字符串的位长度。	bit_length('jose')	32
char_length(string)	获取字符串的字符长度。	char_length('jose')	4
lower(string)	转换字符串为小写格式。	lower('TOM')	tom
octet_length(string)	返回字符串的字节数。	octet_length('jose')	4
position(substring in string)	查找子字符串在字符串中的位置。	position('om' in 'Thomas')	3
substring(string [from int] [for int])	从字符串中找出指定的子字符串。	substring('Thomas' from 2 for 3)	hom
substring(string from pattern)	从字符串中找出与POSIX正则表达式匹配的子字符串。	substring('Thomas' from '...$')	mas
substring(string from pattern for escape)	从字符串中找出与SQL正则表达式匹配的子字符串。	substring('Thomas' from '%#"o_a#"_' for '#')	oma
trim([leading | trailing | both] [characters] from string)	从字符串String的开始、结尾或两端删除仅包含Characters中字符的最长字符串。 **说明** • 默认从*两端*删除。 • 默认仅包含Characters中字符的最长字符串为*空格*。	trim(both 'xyz' from 'yxTomxx')	Tom
trim([lea| tra| both] [from] string [, char])	从字符串String的开始、结尾或两端删除仅包含Characters中字符的最长字符串。 **说明** • 默认从*两端*删除。 • 默认仅包含Characters中字符的最长字符串为*空格*。	trim(both from 'yxTomxx', 'xyz')	Tom

函数名	描述	用例	结果
upper(string)	转换字符串为大写格式。	upper('tom')	TOM

其他字符串函数

Hologres已支持的其他字符串函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见其他字符串函数。

函数名	描述	用例	结果
ascii(string)	返回参数第一个字符的ASCII码。	ascii('x')	120
btrim(string text [, characters text])	从字符串String的开始和结尾删除仅包含Characters中字符的最长字符串。 **说明** 默认仅包含Characters中字符的最长字符串为*空格*。	btrim('xyxtrimyyx', 'xyz')	trim
chr(int)	返回指定编码值对应的字符。 **说明** 参数必须是合法的ASCII或UTF8编码值，并且参数值不能为*0*。	chr(65)	A
concat(str "any" [, str "any" [, ...]])	连接所有参数。忽略*NULL*参数。	concat('abcde', 2, NULL, 22)	abcde222
concat_ws(sep text, str "any" [, str "any" [, ...]])	使用分隔符连接除第一个参数外的所有参数。 **说明** 第一个参数用作分隔符字符串。忽略*NULL*参数。	concat_ws(',', 'abcde', 2, NULL, 22)	abcde,2,22
initcap(string)	将每个单词的第一个字母转换为大写，其余字母转换为小写。 **说明** 单词是由一系列字母和数字组成的字符，使用非字母或数字分隔。	initcap('hi THOMAS')	Hi Thomas
length(string)	返回字符串中字符的个数。	length('jose')	4

函数名	描述	用例	结果
lpad(string text, length int [, fill text])	用Fill填充在String头部，将String填充为长度是Length的字符串。 **说明** • 如果String的长度已经超过Length，则从右侧将String截断为长度是Length的字符串。 • 如果没有指定Fill的值，则Fill默认为*空格*。	lpad('hi', 5, 'xy')	xyxhi
ltrim(string text [, characters text])	从字符串String的开始删除只包含Characters 中字符的最长的字符串。 **说明** 如果没有指定Characters的值，则Characters默认是空格。	ltrim('zzzytest', 'xyz')	test
md5(string)	计算String的MD5哈希值。结果表示为十六进制的形式。	md5('abc')	900150983cd24fb0d6963f7d28e17f72
parse_ident(quali_iden text [,...])	解析字符串。	parse_ident('"SomeSchema".someTable')	{SomeSchema,sometable}
quote_ident(string text)	使用String作为合法的SQL标识符。 **说明** 当字符串包含非标识符字符或者字符串会转换大小写时，需要添加引号。	quote_ident('Foo bar')	"Foo bar"
quote_literal(string text)	将String转换为合法的SQL语句字符串的常量形式。	quote_literal(E'O\'Reilly')	'O''Reilly'
regexp_matches(string text, pattern text)	返回与POSIX正则表达式匹配的子字符串。	regexp_match('foobarbequebaz', '(bar)(beque)')	{bar,beque}
regexp_replace(str text, pat text, replace text)	使用Replacement替换与POSIX正则表达式匹配的子字符串。	regexp_replace('Thomas', '.[mN]a.', 'M')	ThM
regexp_split_to_array(string text, pattern text)	使用POSIX正则表达式分割字符串。	regexp_split_to_array('hello world', '\s+')	{hello,world}

函数名	描述	用例	结果
regexp_split_to_table(string text, pattern text)	使用POSIX正则表达式分割字符串。	regexp_split_to_table('hello world', '\s+')	hello world (2 rows)
repeat(string text, number int)	将String重复指定Nnumber次。	repeat('Pg', 4)	PgPgPgPg
replace(string text, from text, to text)	替换String中所有From子字符串为To。	replace('abcdefabcdef', 'cd', 'XX')	abXXefabXXef
rpad(string text, length int [, fill text])	用Fill填充在String尾部，将String填充为长度是Length的字符串。 **说明** • 如果String的长度已经超过Length，则从左侧将String截断为长度是Length的字符串。 • 如果没有指定Fill的值，则Fill默认为*空格*。	rpad('hi', 5, 'xy')	hixyx
rtrim(string text [, characters text])	从字符串String的末尾删除只包含Characters中字符的最长的字符串。 **说明** 如果没有指定Characters的值，则Characters默认是空格。	rtrim('testxxzx', 'xyz')	test
strpos(string, substring)	返回Substring在String中的位置。	strpos('high', 'ig')	2
substr(string, from [, count])	从字符串中找出指定的子字符串。	substr('alphabet', 3, 2)	ph
starts_with(string, prefix)	如果字符串以前缀开头，则返回*true*。	starts_with('alphabet', 'alph')	true
to_hex(number int or bigint)	将数字转换为十六进制的表示形式。	to_hex(2147483647)	7fffffff
translate(string text, from text, to text)	使用字符串To中的字符替换From中的字符。	translate('12345', '143', 'ax')	a2x5

6.3.6. 模式匹配函数

Hologres兼容PostgreSQL，支持用标准的PostgreSQL语法进行开发。

Hologres已支持的模式匹配函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见模式匹配函数。

函数名	描述	用例	结果
like	使用LIKE操作符判断字符串与模式是否匹配： • 匹配则返回true。 • 不匹配则返回false。	'abc' LIKE 'a%'	true
not like	使用NOT LIKE操作符判断字符串与模式是否匹配： • 不匹配则返回true。 • 匹配则返回false。	'abc' NOT LIKE 'c'	true
similar to	使用SIMILAR TO操作符判断字符串与模式是否匹配： • 匹配则返回true。 • 不匹配则返回false。	'abc' SIMILAR TO '%(b|d)%'	true
not similar to	使用NOT SIMILAR TO操作符判断字符串与模式是否匹配： • 不匹配则返回true。 • 匹配则返回false。	'abc' NOT SIMILAR TO '(b|c)%'	true
rlike ~	使用正则表达式判断字符串与模式是否匹配： • 匹配则返回true。 • 不匹配则返回false。 说明 区分大小写。	'abc' ~ '(b|d)'	true
rlike !~*	使用正则表达式判断字符串与模式是否不匹配： • 不匹配则返回true。 • 匹配则返回false。 说明 不区分大小写。	'abc' !~* '(B|D)'	false

函数名	描述	用例	结果
rlike ~*	使用正则表达式判断字符串与模式是否匹配： • 匹配则返回true。 • 不匹配则返回false。 说明 不区分大小写。	'abc' ~* '(B|D)'	true
rlike !~	使用正则表达式判断字符串与模式是否不匹配： • 不匹配则返回true。 • 匹配则返回false。 说明 区分大小写。	'abc' !~ '(b|d)'	false

6.3.7. 类型转换函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。本文为您介绍Hologres已支持的类型转换函数列表及使用用例。

Hologres已支持的类型转换函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见类型转换函数。

函数名	返回类型	描述	用例	结果	备注

函数名	返回类型	描述	用例	结果	备注
to_char(timestamp, text)	TEXT	将时间戳转换为字符串，支持时间范围为1925~2282年。	to_char(current_timestamp, 'HH12:MI:SS')	06:26:33	从1.1.31版本开始，在SQL前执行 `set hg_experimental_functions_use_pg_implementation = 'to_char';` 或者 `set hg_experimental_functions_use_pg implementation = 'to_char,to_date,to_timestamp';` 可支持所有时间。 **说明** 使用该GUC参数后，查询性能约有50%的损失，升级至Hologres V1.1.42及以上版本后，约有20%的损失。
to_char(int, text)	TEXT	将整数转换为字符串。	to_char(125, '999')	125	无
to_char(double precision, text)	TEXT	将实数或双精度数转换为字符串。	to_char(125.8::real, '999D9')	125.8	无

函数名	返回类型	描述	用例	结果	备注
to_date(text, text)	DATE	将字符串转换为日期，支持时间范围为1925~2282年。	• to_date('05 Dec 2000', 'DD Mon YYYY') • to_date('2000 12 05', 'YYYY MM DD')	2000-12-05	从1.1.31版本开始，在SQL前执行 `set hg_experimental_functions_use_pg_implementation = 'to_date';` 或者 `set hg_experimental_functions_use_pg_implementation = 'to_char,to_date,to_timestamp';` 可支持所有时间。 **说明** 使用该GUC参数后，查询性能约有50%的损失，升级至Hologres V1.1.42及以上版本后，约有20%的损失。
to_number(text, text)	NUMERIC	将字符串转换为数字。	to_number('12,454.8-', '99G999D9S')	-12454.8	无

函数名	返回类型	描述	用例	结果	备注
to_timestamp(text, text)	TIMESTAMP	将字符串转换为时间戳，支持时间范围为1925~2282年。	to_timestamp('05 Dec 2000', 'DD Mon YYYY')	2000-12-05 00:00:00	• 执行结果不包含 `+08` 。 • 从1.1.31版本开始，在SQL前执行 `set hg_experimental_functions_use_pg_implementation = 'to_timestamp';` 可支持所有时间。 **说明** 使用该GUC参数后，查询性能约有50%的损失，升级至Hologres V1.1.42及以上版本后，约有20%的损失。
array_to_string(anyarray, text [, text])	TEXT	将数组转换为字符串。	array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*')	1,2,3,*,5	无
array_agg(expression)	ARRAY	将值串联到数组中，可作为字符串转数组、列转行使用。	• array_agg(123) • array_agg(colname)	• {123} • {1，2}	无
string_agg(expression)	TEXT	使用指定分隔符将字段的非空值串联成字符串，可作为列转行。	string_agg(colname, '-')	a-b-c	无
regexp_split_to_table(string text, pattern text)	TEXT	使用POSIX正则表达式分割字符串，可作为行转列。	regexp_split_to_table('hello world', '\s+')	hello world	无
isnumeric(text)	BOOLEAN	判断输入字符串是否是有效数字类型。	isnumeric('95.5')	true	从V1.1版本开始支持。

6.3.8. 时间和日期函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。本文为您介绍Hologres已支持的时间和日期函数列表及使用用例。

Hologres已支持的时间和日期函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见时间和日期函数和时间格式化函数。

格式化函数

函数名	返回类型	描述	用例	结果
to_char(timestamp, text)	TEXT	将时间戳转换为字符串。 说明 可以使用to_char函数进行24小时制和12小时制的转换。	• to_char(current_timestamp, 'HH24:MI:SS') • to_char(current_timestamp,'HH12:MI:SS AM') • to_char(time '00:30:00','HH12:MI:SS AM')	• 18:26:33 • 18:26:33 PM • 12:30:00 AM
to_date(text, text)	DATE	将字符串转换为日期。	• to_date('05 Dec 2000', 'DD Mon YYYY') • to_date('2000 12 05', 'YYYY MM DD')	2000-12-05
to_timestamp(text, text)	TIMESTAMP	将字符串转换为时间类型。	to_timestamp('05 Dec 2000', 'DD Mon YYYY')	2000-12-05 00:00:00

to_timestamp(double precision)	TIMESTAMP	将时间戳转换为日期。
		说明 从1970-01-01 00:00:00+00的秒数开始转换。
			• 时间戳为秒的转换
			`SELECT TO_TIMESTAMP(163280296)`
			• 时间戳为毫秒的转换
			`SELECT TO_TIMESTAMP(1632802961000/1000)`
				• 1975-03-06 03:38:16+08
				• 2021-09-28 12:22:41+08

操作符

操作符	返回类型	用例	结果
	DATE	date '2001-09-28' + integer '7'	2001-10-05

操作符	返回类型	用例	结果
+	TIMESTAMP	date '2001-09-28' + time '03:00'	2001-09-28 03:00:00
		date '2001-09-28' + interval '1 hour'	2001-09-28 01:00:00
		timestamp '2001-09-28 01:00' + interval '23 hours'	2001-09-29 00:00:00
-	INTEGER	date '2001-10-01' - date '2001-09-28'	3
	DATE	date '2001-10-01' - integer '7'	2001-09-24
	TIMESTAMP	date '2001-09-28' - time '03:00'	2001-09-27 21:00:00
		date '2001-09-28' - interval '1 hour'	2001-09-27 23:00:00
		timestamp '2001-09-28 23:00' - interval '23 hours'	2001-09-28 00:00:00
*	INTERVAL	21 * interval '1 day'	21 days
/	INTERVAL	interval '1 hour' / double precision '1.5'	00:40:00

当前日期/时间

函数名	返回类型	描述	用例	结果
current_date	DATE	获取当前日期。	current_date	2020-05-03
current_timestamp	TIMESTAMP WITH TIME ZONE	获取当前事务的开始时刻。 说明 在事务的整个运行周期内不改变。	current_timestamp	2020-05-03 06:33:36.113682+08
clock_timestamp()	TIMESTAMP WITH TIME ZONE	获取当前时刻。 说明 在同一条命令中也会发生变化。	clock_timestamp()	2020-05-03 06:32:28.814918+08

函数名	返回类型	描述	用例	结果
localtimestamp	TIMESTAMP	获取不包含时区的当前时间。	localtimestamp	2020-08-21 12:02:21.178031
now()	TIMESTAMP WITH TIME ZONE	获取当前事务的开始时刻，等效于transaction_timestamp()。 **说明** 在事务的整个运行周期内不改变。	now()	2020-05-03 06:38:48.492168+08
statement_timestamp()	TIMESTAMP WITH TIME ZONE	获取当前语句的开始时刻。 **说明** 在事务的不同命令中返回值不同。	statement_timestamp()	2020-05-05 06:39:11.125957+08
timeofday()	TEXT	获取当前时刻。 **说明** 与clock_timestamp()类似，但时间使用格式化文本字符串格式。	timeofday()	Tue May 03 06:39:43.195368 2020 CST
transaction_timestamp()	TIMESTAMP WITH TIME ZONE	获取当前事务的开始时刻，等效于current_timestamp。 **说明** 在事务的整个运行周期内不改变。	transaction_timestamp()	2020-05-03 06:40:08.023623+08

其他函数

函数名	返回类型	描述	用例	结果

函数名	返回类型	描述	用例	结果
date_part(text, timestamp)	DOUBLE PRECISION	从时间戳中获取子字段，等效于extract(field from timestamp)。 **说明** 输入的text常量值包括year、month、day、hour、minute、second。	date_part('hour', timestamp '2001-02-16 20:38:40')	20
date_trunc(text, timestamp)	TIMESTAMP	截断时间戳到指定精度。 **说明** 输入的text常量值包括year、month、day、hour、minute、second。	date_trunc('hour', timestamp '2001-02-16 20:38:40')	2001-02-16 20:00:00
extract(field from timestamp)	DOUBLE PRECISION	从时间戳中获取子字段。 **说明** 输入的field常量值包括year、month、day、hour、minute、second。	extract(hour from timestamp '2001-02-16 20:38:40')	20
isfinite(date)	BOOLEAN	判断日期是否为有限值，有限值返回 `true` ，无限值返回 `false` 。	isfinite(date '2001-02-16')	true
isfinite(timestamp)	BOOLEAN	判断时间戳是否为有限值，有限值返回 `true` ，无限值返回 `false` 。	isfinite(timestamp '2001-02-16 21:28:30')	true
make_date(year int, month int, day int)	DATE	使用年、月、日创建日期。	make_date(2013, 7, 15)	2013-07-15

6.3.9. 条件函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发，当前Hologres版本支持的函数是PostgreSQL的一个子集。本文为您介绍Hologres已支持的条件函数列表及使用用例。

条件函数的使用方法请参见条件函数。

> **说明** 以下表格中的用例和结果是在名称为test的表中执行和获取到的，test表中数据如下。

```
 a
---
 1
 2
 3
```

函数名	描述	用例	结果

case	在指定的条件表达式为TRUE时执行一组单个或多个语句。	`SELECT a,`
		`  CASE WHEN a=1 THEN 'one'`
		`  WHEN a=2 THEN 'two'`
		`  ELSE 'other'`
		`  END`
		`FROM test;`
			`a  case`
			`----+-----------`
			`1  one`
			`2  two`
			`3  other`

coalesce	返回参数列表中第一个非空表达式的值。
	说明 仅当所有表达式为NULL时，结果返回NULL。
		`SELECT a,`
		`  COALESCE(null, 'a', 'b')`
		`FROM test;`
			`a  coalesce`
			`----+-----------`
			`1  a`
			`2  a`
			`3  a`

nullif	如果两个表达式的值相等，则结果返回NULL，反之返回第一个表达式的值。	`SELECT a,`
		`  nullif('a','a')`
		`FROM test;`
			`a  nullif`
			`----+-----------`
			`1`
			`2`
			`3`

greatest	选择表达式列表中的最大值。	`SELECT a,`
		
		`greatest('a','b','c')`
		`FROM test;`
			`a  greatest`
			`----+-----------`
			`1  c`
			`2  c`
			`3  c`

函数名	描述	用例	结果
least	选择表达式列表中的最小值。	SELECT a, least('a','b','c') FROM test;	a least ----+----------- 1 a 2 a 3 a

6.3.10. 数组函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的数组函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见数组函数。

函数名	描述	用例	结果
array_append(anyarray, anyelement)	添加元素至数组的尾部。	array_append(ARRAY[1, 2], 3)	{1,2,3}
array_cat(anyarray,anyarray)	连接两个数组。	array_cat(ARRAY[1,2,3], ARRAY[4,5])	{1,2,3,4,5}
array_ndims(anyarray)	返回数组的维度数。	array_ndims(ARRAY[[1,2,3], [4,5,6]])	2
array_dims(anyarray)	使用文本形式表示数组的维度。	array_dims(ARRAY[[1,2,3], [4,5,6]])	[1:2][1:3]
array_length(anyarray, int)	返回被请求的数组维度的长度。	array_length(ARRAY[1,2,3], 1)	3
array_lower(anyarray, int)	返回ANYARRAY数组维度的下限。	array_lower('[0:2]={1,2,3}'::int[], 1)	0
array_positions(anyarray, anyelement)	返回在第一个参数给定的数组（数组必须是一维的）中， 第二个参数所有出现位置的下标组成的数组。	array_positions(ARRAY['A','A','B','A'], 'A')	{1,2,4}
array_prepend(anyelement, anyarray)	添加元素至数组的头部。	array_prepend(1, ARRAY[2,3])	{1,2,3}
array_remove(anyarray, anyelement)	从数组中移除所有等于给定值的所有元素。	array_remove(ARRAY[1,2,3,2], 2)	{1,3}

函数名	描述	用例	结果
array_sort(anyarray)	支持text类型数组，且text数组将会被转为int8数组进行排序，返回排序后的text数组。 说明 1.1.46+版本开始支持	array_sort(ARRAY['1','3','2','1'])	{'1','1','2','3'}
array_to_string(anyarray,text[,text])	使用提供的定界符和可选的空串连接数组元素。	array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*')	1,2,3,*,5
array_upper(anyarray, int)	返回ANYARRAY数组维度的上限。	array_upper(ARRAY[1,8,3,7], 1)	4
unnest(anyarray)	将数组的每个元素扩展为单独行。	unnest(ARRAY[1,2])	1 2 (2 rows)

6.3.11. 通用聚合函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的通用聚合函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见通用聚合函数。

函数名	描述	用例	结果
array_agg(bigint)	将BIGINT类型表达式的值串联到数组中。	array_agg(c1)	{1,2}
array_agg(bool)	将BOOL类型表达式的值串联到数组中。	array_agg(c1)	{true,false}
array_agg(text)	将TEXT类型表达式的值串联到数组中。	array_agg(c1)	{a,b}
array_agg(float8)	将FLOAT8类型表达式的值串联到数组中。	array_agg(c1)	{1.1,2.2}
array_agg(float4)	将FLOAT4类型表达式的值串联到数组中。	array_agg(c1)	{1.1,2.2}
array_agg(int)	将INT类型表达式的值串联到数组中。	array_agg(c1)	{1, 2}
avg(bigint)	求BIGINT类型表达式中非空值的平均值。	avg(c1)	2.000000
avg(float8)	求FLOAT8类型表达式中非空值的平均值。	avg(c1)	2.000000

函数名	描述	用例	结果
avg(float4)	求FLOAT4类型表达式中非空值的平均值。	avg(c1)	2.000000
avg(int)	求INT类型表达式中非空值的平均值。	avg(c1)	2.000000
bit_and(bigint)	对BIGINT类型表达式中的非空值执行按位与运算。	bit_and(c1)	0
bit_and(int)	对INT类型表达式中的非空值执行按位与运算。	bit_and(c1)	0
bit_or(bigint)	对BIGINT类型表达式中的非空值执行按位或运算。	bit_or(c1)	3
bit_or(int)	对INT类型表达式中的非空值执行按位或运算。	bit_or(c1)	3
bool_and(bool)	如果BOOL表达式的值均为TRUE，则函数结果返回TRUE，否则返回FALSE。	bool_and(c1)	f
bool_or(bool)	如果BOOL表达式的值包含TRUE，则函数结果返回TRUE，否则返回FALSE。	bool_or(c1)	t
count(*)	返回指定表的行数。	count(*)	3
count(bigint)	求BIGINT类型表达式的输入行数。 **说明** BIGINT类型表达式的值不为NULL。	count(c1)	3
count(numeric)	求NUMERIC类型表达式的输入行数。 **说明** NUMERIC类型表达式的值不为NULL。	count(c1)	3
every(bool)	如果BOOL表达式的值均为TRUE，则函数结果返回TRUE，否则返回FALSE。	无	无
max(bigint)	求BIGINT类型表达式的最大值。	max(c1)	3
max(float8)	求FLOAT8类型表达式的最大值。	max(c1)	3.0
max(float4)	求FLOAT4类型表达式的最大值。	max(c1)	3.0
max(int)	求INT类型表达式的最大值。	max(c1)	3
max(numeric)	求NUMERIC类型表达式的最大值。	max(c1)	3.0
min(bigint)	求BIGINT类型表达式的最小值。	min(c1)	1

函数名	描述	用例	结果
min(float8)	求FLOAT8类型表达式的最小值。	min(c1)	1.0
min(float4)	求FLOAT4类型表达式的最小值。	min(c1)	1.0
min(int)	求INT类型表达式的最小值。	min(c1)	1
min(numeric)	求NUMERIC类型表达式的最小值。	min(c1)	1.0
sum(bigint)	求BIGINT类型表达式所有值的总和。	sum(c1)	6
sum(float8)	求FLOAT8类型表达式所有值的总和。	sum(c1)	6.0
sum(float4)	求FLOAT4类型表达式所有值的总和。	sum(c1)	6.0
sum(int)	求INT类型表达式所有值的总和。	sum(c1)	6
sum(numeric)	求NUMERIC类型表达式所有值的总和。	sum(c1)	6.0
string_agg(expression, delimiter)	使用指定分隔符将指定表达式的非空值串联成字符串。	string_agg(c1, '-')	a-b-c
corr(Y, X)	求相关系数。	corr(c1, c2)	无
covar_pop(Y, X)	求总体协方差。	covar_pop(c1, c2)	无
covar_samp(Y, X)	求样本协方差。	covar_samp(c1, c2)	无
regr_avgx(Y, X)	求自变量的平均值。	reg_avgx(c1, c2)	无
regr_avgy(Y, X)	求因变量的平均值。	reg_avgy(c1, c2)	无
regr_count(Y, X)	求两个输入参数中都不为空的行数。	regr_count(c1, c2)	无
regr_intercept(Y, X)	求由(X,Y)确定的最小方差拟合的纵轴截距。	reg_intercept(c1, c2)	无
regr_r2(Y, X)	求相关系数的平方。	regr_r2(c1, c2)	无
regr_slope(Y, X)	求由(X,Y)确定的最小方差拟合的斜率。	regr_slope(c1, c2)	无
regr_sxx(Y, X)	求自变量的平方和 `sum(X^2) - sum(X)^2/N` 。	regr_sxx(c1, c2)	无
regr_sxy(Y, X)	求自变量和因变量的乘积和 `sum(X*Y) - sum(X) * sum(Y)/N` 。	regr_sxy(c1, c2)	无
regr_syy(Y, X)	求因变量的平方和 `sum(Y^2) - sum(Y)^2/N` 。	regr_syy(c1, c2)	无

函数名	描述	用例	结果
stddev(int)	求INT类型表达式的样本标准差。	stddev(c1)	无
stddev(numeric)	求NUMERIC类型表达式的样本标准差。	stddev(c1)	无
stddev(float8)	求FLOAT8类型表达式的样本标准差。	stddev(c1)	无
stddev_pop(int)	求INT类型表达式的总体标准差。	stddev_pop(c1)	无
stddev_pop(numeric)	求NUMERIC类型表达式的总体标准差。	stddev_pop(c1)	无
stddev_pop(float8)	求FLOAT8类型表达式的总体标准差。	stddev_pop(c1)	无
stddev_samp(int)	求INT类型表达式的样本标准差。	stddev_samp(c1)	无
stddev_samp(numeric)	求NUMERIC类型表达式的样本标准差。	stddev_samp(c1)	无
stddev_samp(float8)	求FLOAT8类型表达式的样本标准差。	stddev_samp(c1)	无
variance(int)	求INT类型表达式的样本方差。	variance(c1)	无
variance(numeric)	求NUMERIC类型表达式的样本方差。	variance(c1)	无
var_pop(float8)	求FLOAT8类型表达式的总体方差。	var_pop(c1)	无
var_pop(int)	求INT类型表达式的总体方差。	var_pop(c1)	无
var_pop(numeric)	求NUMERIC类型表达式的总体方差。	var_pop(c1)	无
var_samp(float8)	求FLOAT8类型表达式的样本方差。	var_samp(c1)	无
var_samp(int)	求INT类型表达式的样本方差。	var_samp(c1)	无
var_samp(numeric)	求NUMERIC类型表达式的样本方差。	var_samp(c1)	无

6.3.12. 有序集合函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

命令语法

当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见有序集合函数。

Hologres已支持的有序集合函数表达式如下。

```
GROUPING( <expr1> [ , <expr2> , ... ] )
```

参数说明

GROUPING不是聚合函数，而是可以与聚合一起使用的实用程序函数，用于确定针对如下行生成行的聚合级别：

- GROUPING(`expr`) 若返回0，表示按照 `expr` 行进行了聚合，若返回1，表示未按照 `expr` 行进行聚合。
- GROUPING(`expr1，expr2,...exprN`) 将会返回包含 GROUPING(`expr1`)，GROUPING( )，...，GROUPING(`exprN`)的位向量的整数表示。

使用示例

- 创建表并插入数据

```
CREATE TABLE t1
(
    col_x INT
    ,col_y INT
    ,col_z INT
)
;
INSERT INTO t1 VALUES(1, 2, 1), (1, 2, 3) ;
INSERT INTO t1 VALUES(2, 1, 10), (2, 2, 11), (2, 2, 3) ;
```

生成的数据如下所示：

```
select * from t1 order by col_x, col_y, col_z;
 col_x | col_y | col_z
-------+-------+-------
     1 |     2 |     1
     1 |     2 |     3
     2 |     1 |    10
     2 |     2 |     3
     2 |     2 |    11
(5 rows)
```

- 使用样例

```
SELECT  col_x
        ,col_y
        ,SUM(col_z)
        ,grouping(col_x)
        ,grouping(col_y)
        ,grouping(col_x, col_y)
FROM    t1
GROUP BY grouping sets ((col_x), (col_y), ())
ORDER BY 1,2;
```

如下内容为输出结果，其中，3表示二进制数11转换为的整数。

```
 col_x | col_y | sum | grouping | grouping | grouping
-------+-------+-----+----------+----------+----------
     1 |       |   4 |        0 |        1 |        1
     2 |       |  24 |        0 |        1 |        1
       |     1 |  10 |        1 |        0 |        2
       |     2 |  18 |        1 |        0 |        2
       |       |  28 |        1 |        1 |        3
(5 rows)
```

6.3.13. 窗口函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

函数列表

Hologres已支持的窗口函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见窗口函数。

函数名	描述	用例	结果	支持的引擎
row_number()	返回当前行在分区中的编号，返回类型为BIGINT。 说明 从1开始计数。	row_number() over (order by c1)	c1 | row_number ---+--- a | 1 a | 2 b | 3 c | 4 (4 rows)	PQE
rank()	返回当前行在分区中的排名，返回类型为BIGINT。 说明 Rank函数是跳跃排序，生成的序号有可能不连续。	rank() over (order by c1)	c1 | rank ---+--- a | 1 a | 1 b | 3 c | 4 (4 rows)	PQE

函数名	描述	用例	结果	支持的引擎
dense_rank()	返回当前行在分区中的排名，返回类型为BIGINT。 **说明** Dense_Rank函数生成的序号是连续的。	dense_rank() over (order by c1)	c1 | dense_rank ---+--- a | 1 a | 1 b | 2 c | 3 (4 rows)	PQE
percent_rank()	求当前行在分区中的百分比排名 （rank-1）/（总分区行-1） ，返回类型为DOUBLE PRECISION。	percent_rank() over (order by c1)	c1 | percent_rank ---+--- a | 0 a | 0 b | 0.6666666666667 c | 1 (4 rows)	PQE

函数名	描述	用例	结果	支持的引擎
lag(value anyelement)	返回value的当前行在分区中的值，返回类型与value相同。	c1, lag(c1) over (order by c1)	c1 | lag ---+--- a | a b | a c | b (4 rows)	Hologres从V1.1.71版本开始HQE支持。
lead(value anyelement)	返回value的当前行在分区中的值，返回类型与value相同。	c1, lead(c1) over (order by c1)	c1 | lead ---+---- a | a a | b b | c c | (4 rows)	Hologres从V1.1.71版本开始HQE支持。
first_value(value anyelement)	返回在窗口框架的第一行计算出的值，返回类型与value相同。	c1, first_value(c1) over (order by c1)	c1 | first_value ---+------ a | a a | a b | a c | a (4 rows)	Hologres从V1.1.71版本开始HQE支持。

函数名	描述	用例	结果	支持的引擎
last_value(value anyelement)	返回在窗口框架的最后一行计算出的值，返回类型与value相同。	c1, last_value(c1) over (order by c1)	c1 | last_value ---+--- a | a a | a b | b c | c (4 rows)	Hologres从V1.1.71版本开始HQE支持。

窗口函数特性

调用窗口函数时，需要在窗口函数及其参数后增加一个OVER子句。OVER子句决定了查询结果中的哪些行需要被分离出来，由窗口函数处理。

- 当OVER子句使用分区子句 `PARTITION BY` 时，分区列值相同的行归属同一分区。对于每一行，窗口函数都会针对其所在分区进行计算。
- 当OVER子句使用排序子句 `ORDER BY` 时，窗口函数会按其定义的顺序处理行。

对于每一行，在其分区中的行集被称为窗口帧。针对部分窗口函数，只会作用于当前行所在的窗口帧，而非整个分区。在使用排序子句ORDER BY的情况下，窗口帧默认为从分区开始到当前行的范围；如果没有定义排序，则窗口帧默认为当前分区的所有行。

以SUM函数为例，是否使用排序子句查询得到的结果如下表。

用例	结果
id, c1, sum(id) over (order by id)	id | c1 | sum ---+----+----- 1 | a | 2 1 | a | 2 3 | b | 5 4 | c | 9 (4 rows)

用例	结果
id, c1, sum(id) over ()	id | c1 | sum ---+----+----- 1 | a | 9 1 | a | 9 3 | b | 9 4 | c | 9 (4 rows)

可以看出，当使用排序子句时，求和值为第一行（最小值）到当前行的和，并且包括与当前行具有相同值的行；当不使用排序子句时，每次求和都针对整个表。

6.3.14. 子查询函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的子查询函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见子查询函数。

函数名	描述
EXISTS (subquery)	判断子查询结果是否返回行： • 如果至少返回一行，则EXISTS的结果为*t*，代表true。 • 如果没有返回行，则EXISTS的结果为*f*，代表false。 **说明** Subquery参数为任意的SELECT语句。
IN (subquery)	逐行比较指定表达式的值与子查询结果是否相等： • 如果存在相等行，则IN的结果返回*t*，代表true。 • 如果不存在相等行，则IN的结果返回*f*，代表false。 **说明** 指定表达式的值与子查询结果的列数目必须相同。
NOT IN (subquery)	逐行比较指定表达式的值与子查询结果是否相等： • 如果不存在相等行，则NOT IN的结果返回*t*，代表true。 • 如果存在相等行，则NOT IN的结果返回*f*，代表false。 **说明** 指定表达式的值与子查询结果的列数目必须相同。

函数名	描述
ANY (subquery)	使用指定操作符逐行运算指定表达式的值与子查询结果： • 如果运算结果存在*t*，则ANY的结果返回*t*，代表true。 • 如果运算结果均为*f*，则ANY的结果返回*f*，代表false。 **说明** 指定表达式的值与子查询结果的列数目必须相同。
SOME (subquery)	使用给定的操作符计算指定表达式并与子查询结果的每一行进行比较： • 如果运算结果存在*t*，则SOME的结果返回*t*，代表true。 • 如果运算结果均为*f*，则SOME的结果返回*f*，代表false。 **说明** 指定表达式的值与子查询结果的列数目必须相同。

6.3.15. 比较函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的比较函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见比较函数。

函数名	描述
expression NOT IN (values,...)	如果Expression表达式的值与所有Value的值都不相等，则结果返回*TRUE*，反之返回*FALSE*。
expression IN (values,...)	如果Expression表达式的值与任意Value的值相等，则结果返回*TRUE*，反之返回*FALSE*。

6.3.16. 设置返回函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的设置返回函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见设置返回函数。

函数名	描述	用例	结果

函数名	描述	用例	结果
generate_series(start, stop)	从Start到Stop生成一个步长为1的数值序列。	generate_series(2,4)	2 3 4 (3 rows)
generate_series(start, stop, step)	从Start到Stop生成一个步长为Step的数值序列。 说明 参数类型为*INT*、*BIGINT*或*NUMERIC*。	generate_series(5,1,-2)	5 3 1 (3 rows)
generate_series(start, stop, step interval)	从Start到Stop生成一个步长为Step的数值序列。 说明 参数类型为*TIMESTAMP*或*TIMESTAMP WITH TIME ZONE*。	generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours')	2008-03-01 00:00:00 2008-03-01 10:00:00 2008-03-01 20:00:00 2008-03-02 06:00:00 2008-03-02 16:00:00 2008-03-03 02:00:00 2008-03-03 12:00:00 2008-03-03 22:00:00 2008-03-04 08:00:00 (9 rows)
generate_subscripts(array anyarray, dim int)	为给定数组的指定维度生成有效下标集。	generate_subscripts('{NULL,1,NULL,2}'::int[], 1)	1 2 3 4 (4 rows)

6.3.17. 权限查询函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的权限查询函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见权限查询函数。

函数名	描述
has_any_column_privilege(user, table, privilege)	指定的用户是否有表中任意列的权限。

函数名	描述
has_any_column_privilege(table, privilege)	当前用户是否有表中任意列的权限。
has_column_privilege(user, table, column, privilege)	指定的用户是否有列权限。
has_column_privilege(table, column, privilege)	当前用户是否有列权限。
has_database_privilege(user, database, privilege)	指定的用户是否有数据库权限。
has_database_privilege(database, privilege)	当前用户是否有数据库权限。
has_foreign_data_wrapper_privilege(user, fdw, privilege)	指定的用户是否有外部数据包装器的权限。
has_foreign_data_wrapper_privilege(fdw, privilege)	当前用户是否有外部数据包装器的权限。
has_function_privilege(user, function, privilege)	指定的用户是否有函数权限。
has_function_privilege(function, privilege)	当前用户是否有函数权限。
has_language_privilege(user, language, privilege)	指定的用户是否有语言权限。
has_language_privilege(language, privilege)	当前用户是否有语言权限。
has_schema_privilege(user, schema, privilege)	指定的用户是否有模式权限。
has_schema_privilege(schema, privilege)	当前用户是否有模式权限。
has_server_privilege(user, server, privilege)	指定的用户是否有外部服务器的权限。
has_server_privilege(server, privilege)	当前用户是否有外部服务器的权限。
has_table_privilege(user, table, privilege)	指定的用户是否有表权限。
has_table_privilege(table, privilege)	当前用户是否有表权限。
has_tablespace_privilege(user, tablespace, privilege)	指定的用户是否有表空间权限。
has_tablespace_privilege(tablespace, privilege)	当前用户是否有表空间权限。
has_type_privilege(user, type, privilege)	指定的用户是否有类型的权限。
has_type_privilege(type, privilege)	当前用户是否有类型的权限。
pg_has_role(user, role, privilege)	指定的用户是否有角色权限。
pg_has_role(role, privilege)	当前用户是否有角色权限。

6.3.18. 连接函数

Hologres兼容PostgreSQL，支持使用标准的PostgreSQL语法进行开发。

Hologres已支持的连接函数列表如下。当前Hologres版本支持的函数是PostgreSQL的一个子集，函数的使用方法请参见连接函数。

函数名	描述
current_catalog	当前数据库
current_database()	返回当前数据库名称。
current_query()	执行当前查询操作。
current_role	当前身份
current_schema[()]	返回当前模式的名称。
current_schemas(boolean)	返回搜索路径中出现的所有模式名称的数组。
current_user	当前用户
inet_client_addr()	返回当前用户的远程连接地址。
inet_client_port()	返回当前用户的远程连接端口。
inet_server_addr()	返回当前服务器的地址。
inet_server_port()	返回当前服务器的端口。
pg_backend_pid()	返回当前会话所在服务器进程的进程ID。
pg_blocking_pids(int)	阻止指定服务器进程ID获取锁定的进程ID。
session_user	当前会话用户
user	用户
version()	返回PostgreSQL的版本信息。

6.4. 扩展函数

6.4.1. 概览

本文为您介绍交互式分析Hologres支持的扩展函数。

Hologres兼容PostgreSQL，支持的函数以及用法同标准的PostgreSQL。

Hologres支持的函数是PostgreSQL的子集。在实际业务场景中，标准的PostgreSQL函数并不能完全满足您的业务需求。因此，Hologres丰富了支持的函数，您可以根据业务需求通过调用函数来简化开发流程。

Hologres支持的扩展函数如下表所示。

函数分类	函数名称	应用场景	是否支持内部表	是否支持外部表	支持的版本
空间地理函数	PostGIS (Beta)	计算空间对象、空间索引、空间操作函数和空间操作符等空间信息。	是	否	V0.10及以上版本。
向量计算	Proxima向量计算	提供高性能的向量查询功能。	是	否	V0.10及以上版本。
聚合函数	聚合视图（Beta）	提供单表固定维度列的预聚合能力。	是	否	V0.10及以上版本。
	APPROX_COUNT_DISTINCT	count distinct近似结果函数，性能更优。	是	是	V0.10及以上版本。
流量分析函数	漏斗分析函数和留存函数	提供漏斗和留存分析，计算行为转化率。	是	否	V0.9及以上版本。
	明细圈人函数	找出明细表中满足某些条件组合的用户列表，减少Join开销。	是	否	V0.10及以上版本。
	Roaring Bitmap函数	高效Bitmap压缩算法，常用于去重（UV计算）、标签筛选、近实时用户画像等场景。	是	否	V0.9及以上版本。
账号转换函数	USER_DISPLAY_NAME	将账号ID转换为用户名。	不涉及	不涉及	所有版本。
	HG_USER_DISPLAY_NAME_TO ID	将用户名转换为用户ID。	不涉及	不涉及	所有版本。
Hive兼容函数	GET_JSON_OBJECT	解析JSON对象。	是	否	V0.9及以上版本。
MaxCompute兼容函数	MAX_PT	计算最大分区表。	是	是	V0.9及以上版本。
工具函数	HG_VERSION	查看Hologres版本。	不涉及	不涉及	所有版本。
	HG_SHARD_ID_FOR_DISTRIBUTION_KEY	查看数据所在的数据分片（Shard ID）。	不涉及	不涉及	所有版本。
	HG_UPDATE_DATABASE_PROPERTY	更新数据库属性。	不涉及	不涉及	所有版本。
	SET_TABLE_PROPERTY	设置表属性，与建表语句一起执行。	是	不涉及	所有版本。

6.4.2. PostGIS (Beta)

PostGIS是数据库PostgreSQL的空间扩展，PostGIS可以提供空间对象、空间索引、空间操作函数和空间操作符等空间信息服务功能。本文为您介绍PostGIS在Hologres中的使用方法。

使用限制

在Hologres中使用PostGIS，目前该功能为Beta版本，在使用过程中不能100%保证其稳定性。目前也暂不支持PostGIS索引。如果您在使用过程中有任何问题，请您提交工单进行反馈。

安装PostGIS

在使用PostGIS之前，需要Superuser在DB内执行以下语句安装扩展包才可以正常使用。一个DB只需执行一次即可，如果创建新的DB，还需要再次执行如下语句。

```
-- 加载PostGIS插件
create extension if not exists postgis;
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP extension  postgis;
> ```

在安装完成后，您可以执行如下SQL语句查看当前PostGIS版本。

```
select postgis_full_version();
```

创建并查询包含空间数据类型的表

目前在Hologres中使用PostGIS，支持两类空间数据类型，即几何类型（Geometry Type）和地理类型（Geography Type）。

在实际使用过程中，几何类型（Geometry Type）是使用较频繁的类型，如下内容将以几何类型为例，指导您创建包含Geometry数据类型的表并进行典型空间查询。更多关于地理类型（Geography Type）的参数和使用说明，请参见PostGIS Geography Type。

1. **创建包含Geometry数据类型的表**

   > **说明** 创建包含几何类型的表时，您可以指定创建的几何体类型。包括Point、MultiPoint、Linestring、MultiLinestring、Polygon、MultiPolygon等。

   - 创建不指定几何类型的表。

     ```
     CREATE TABLE holo_gis_1 ( id int, geom geometry, PRIMARY KEY (id)) ;
     ```

     以上示例中，创建了包含几何类型的表，但是该表中不指定具体的几何类型。

   - 创建指定几何体类型和SRID的表。

     ```
     CREATE TABLE holo_gis_2 ( id int, geom geometry(point, 4326), PRIMARY KEY (id)) ;
     ```

     以上示例中，Geometry类型指定为Point，SRID为4326，SRID不指定默认为0。更多关于SRID释义，请参见PostGIS官方文档。

2. **向表中插入数据**

   您可以通过如下方式向表中插入数据，其中，关于空间函数的使用说明，请参见空间函数。关于SRID释义，请参见PostGIS官方文档。

   - 不指定SRID值。

     ```
     insert into holo_gis_1 values (1, ST_GeomFromText('point(116 39)'));
     ```

   - 指定SRID值。

     ```
     insert into holo_gis_2 values (1, ST_GeomFromText('point(116 39)', 4326));
     ```

3. **执行数据查询**

   您可以在创建表并插入数据后，执行矩形范围查询和多边形相交判定两种典型场景的查询。如下示例中，更多关于空间函数的使用说明，请参见空间函数。

   - 矩形范围查询
     - 不指定SRID值。

       ```
       select st_astext(geom) from holo_gis_1
       where ST_Contains(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), geom);
       ```

     - 指定SRID值。

       ```
       select st_astext(geom) from holo_gis_2
       where ST_Contains(ST_SetSRID(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), 432
       6), geom);
       ```

   - 多边形相交判定（在内部或在边界上）
     - 不指定SRID值。

       ```
       select st_astext(geom) from holo_gis_1
       where ST_Contains(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), geom);
       ```

     - 指定SRID值。

       ```
       select st_astext(geom) from holo_gis_2
       where ST_Contains(ST_SetSRID(ST_MakeBox2D(ST_Point(116, 39),ST_Point(117, 40)), 432
       6), geom);
       ```

空间函数

PostGIS为您提供了一些空间函数， 每种函数可以将一种数据类型的值转换为另一种类型。空间函数说明具体见下表，关于函数语法，部分参数说明如下：

- geom：表格中涉及的geom均表示一个GEOMETRY类型的值，或一个计算结果为GEOMETRY类型的表达式。
- precision：表格中涉及的precision均表示一个INTEGER类型的值。坐标系geom将使用指定的精度1~20显示。如果未指定精度，则默认值为15。
- index：表格中涉及的index为索引，均表示一个INTEGER类型的值。
- srid：表格中涉及srid均表示一个INTEGER类型的值，它是空间参考标识符SRID。

更多关于空间函数的说明，请参见PostGIS官方文档。

函数	函数语法	返回类型	说明
GeometryType	GeometryType(geom)	VARCHAR	GeometryType以字符串形式返回输入几何体的子类型。例如，geom为POINT子类型时返回的字符串为POINT。
ST_AddPoint	ST_AddPoint(geom1, geom2) geom1子类型必须为LINESTRING，geom2子类型必须是POINT。	GEOMETRY	ST_AddPoint将一个坐标点添加到LineString之中。
ST_Angle	ST_Angle(geom1, geom2, geom3) ST_Angle(geom1, geom2, geom3, geom4) geom子类型必须是POINT。	DOUBLE	ST_Angle返回顺时针方向测量的点之间的角度（返回值以弧度为单位且在[0, 2π）范围内）。例如： • 如果输入三个点，则测量P1P2P3三个点顺时针组成的角度。 • 如果输入四个点，则测量有向线P1P2和P3P4顺时针形成的角度。如果输入为两条线平行（即，P1等于P2，或P3等于P4），则返回 null。
ST_Area	ST_Area(geom)	DOUBLE	ST_Area返回多边形几何体的笛卡尔面积。 • 对于点、线串、多点和多线串，返回0。 • 对于几何体集合，返回集合中几何体的面积之和。
ST_AsBinary	ST_AsBinary(geom)	BYTEA	ST_AsBinary使用ASCII十六进制字符（0~9，A~F）返回几何体的WKB（Well-Known Binary）表示形式，不包含SRID数据。
ST_AsEWKB	ST_AsEWKB(geom)	BYTEA	ST_AsEWKB使用ASCII十六进制字符（0~9，A~F）返回几何体的EWKB（Well-Known Binary）表示形式，包含SRID数据。
ST_AsEWKT	ST_AsEWKT(geom)	VARCHAR	ST_AsEWKT返回几何体的EWKT（Well-Known Text）表示形式，包含SRID数据。
ST_AsGeoJSON	ST_AsGeoJSON(geom) ST_AsGeoJSON(geom, precision)	VARCHAR	ST_AsGeoJSON返回几何体的GeoJSON 表示形式。
ST_AsText	ST_AsText(geom) ST_AsText(geom, precision)	VARCHAR	ST_AsText返回几何体的WKT（Well-Known Text）表示形式，不包含SRID数据。

函数	函数语法	返回类型	说明
ST_Azimuth	ST_Azimuth(point1, point2) point1和point2为GEOMETRY类型的值。其两者的空间参考系统标识SRID必须相互匹配。	DOUBLE	ST_Azimuth返回两个输入点的基于北向的笛卡尔方位。方位角以北为参考方向，顺时针为正，例如北为0，则东为π/2、南为π。如果两点重合则返回null。
ST_Boundary	ST_Boundary(geom)	GEOMETRY	ST_Boundary返回输入几何体的边界。 • 如果输入几何为空（即不包含点），则按原样返回。 • 如果输入几何是点或非空多点，则返回空几何集合。 • 如果输入是线串或多线串，则返回包含边界上所有点的多点（多点可能为空）。 • 如果输入是一个没有任何内环的面，则返回一个表示其边界的闭合线串。 • 如果输入是具有内环的面，或者是多面，则返回多线串。多线串包含面积几何中所有环的所有边界作为闭合线串。
ST_Buffer	ST_Buffer(geography,float8)	GEOMETRY	用于指定与围绕其计算缓冲区的几何图形实例的距离。返回一个几何图形，该几何图形覆盖与输入参数geography的距离小于等于给定值的所有点。
ST_Contains	ST_Contains(geom1, geom2)	BOOLEAN	如果第一个输入几何体包含第二个输入几何体，则ST_Contains返回true。如果B中的每个点均为A中的一个点，并且其内部有非空相交区域，则几何体A包含几何体B。ST_Contains(A, B)与ST_Within(B, A)等效。
ST_ContainsProperly	ST_ContainsProperly(geom1, geom2) geom的子类型不能是GEOMETRYCOLLECTION。	BOOLEAN	如果两个输入几何体都是非空的，并且第二个几何体的所有点都是第一个几何的内部点，则ST_ContainsProperly返回true。
ST_Convexhull	ST_ConvexHull(geom)	GEOMETRY	ST_convexhull返回一个几何体，该几何体表示输入几何体中包含的非空点的凸壳。

函数	函数语法	返回类型	说明
ST_CoveredBy	ST_CoveredBy(geom1, geom2)	BOOLEAN	如果第一个输入几何体被第二个输入几何体覆盖，则ST_CoveredBy返回true。即，如果几何体A和几何体B都是非空的，并且A的每个点均为B中的一个点，则前者被后者覆盖。 ST_CoveredBy(A, B)与 ST_Covers(B, A)等效。
ST_Covers	ST_Covers(geom1, geom2)	BOOLEAN	如果第一个输入几何体被第二个输入几何体覆盖，则ST_Covers返回true。即，如果几何体A和几何体B都是非空的，并且B的每个点均为A中的一个点，则前者覆盖了后者。 ST_Covers(A, B)与ST_CoveredBy(B, A) 等效。
ST_Crosses	ST_Crosses(geom1, geom2)	BOOLEAN	如果两个输入几何体部分交叉，则ST_Crosses返回true。
ST_Dimension	ST_Dimension(geom)	INTEGER	ST_Dimension返回输入几何体的固有维度。固有维度是几何体中定义的子类型的维度值。
ST_Disjoint	ST_Disjoint(geom1, geom2)	BOOLEAN	如果两个输入几何体没有共同点（不相交），则ST_Disjoint返回true。
ST_Distance	ST_Distance(geom1, geom2)	DOUBLE	ST_Distance返回两个输入几何体之间的欧氏距离。
ST_DWithin	ST_DWithin(geom1, geom2, threshold)	BOOLEAN	如果两个输入几何体值之间的欧氏距离在给定的阈值内，则ST_DWithin返回true。
ST_Envelope	ST_Envelope(geom)	GEOMETRY	ST_Envelope返回输入几何体的最小边界框。 • 如果输入几何体为空，则返回的几何体也为空。 • 如果输入几何体的最小边界框退化为一个点，则返回的几何体是一个点。 • 如果输入几何体的最小边界框是一维的，则返回两点线串。 • 如果上述条件都不成立，则函数将返回一个顺时针方向的多边形，其顶点为最小边界框的角。返回几何体的SRID与输入几何体的相同。

函数	函数语法	返回类型	说明
ST_Equals	ST_Equals(geom1, geom2)	BOOLEAN	如果两个输入几何体在几何上相等，则ST_Equals 返回true。即，如果几何体具有相等的点集且其内部具有非空相交区域，则将几何体视为在几何上相等。
ST_ExteriorRing	ST_ExteriorRing(geom)	子类型LINESTRING的GEOMETRY。	ST_ExteriorRing返回一个表示多边形几何体外环的闭合线串。如果输入不是多边形，则返回null。
ST_GeometryN	ST_GeometryN(geom, index)	GEOMETRY	ST_GeometryN返回由输入几何体的输入索引指向的几何体。 • 当输入是点、线串或多边形时，如果索引等于1，则按原样返回几何体；如果索引不是1，则返回 null。 • 如果输入是多点、多线串、多边形或几何体集合，则返回由输入索引（从1开始）指向的点、线串、多边形或几何体集合。返回几何体的SRID与输入几何体的相同。
ST_GeometryType	ST_GeometryType(geom)	VARCHAR	ST_GeometryType以字符串形式返回输入几何体的子类型。例如，当geom为POINT子类型时返回字符串ST_Point。
ST_GeomFromText	ST_GeomFromText(wkt_string) ST_GeomFromText(wkt_string, srid) wkt_string为VARCHAR数据类型的值，它是几何体的WKT表示形式。	GEOMETRY	ST_GeomFromText从输入几何体的WKT（Well-Known Text）表示形式构造几何体对象。该函数有两种形式： • 第一种不包含SRID，并返回SRID=0的几何图形。 • 第二种采用SRID作为第二个参数，并返回一个几何图形，该几何图形将该SRID作为元数据的一部分。
ST_InteriorRingN	ST_InteriorRingN(geom, index)	子类型LINESTRING的GEOMETRY。	ST_InteriorRingN返回与索引位置处输入多边形的内环相对应的闭合线串。
ST_Intersects	ST_Intersects(geom1, geom2)	BOOLEAN	如果两个输入几何体至少有一个共同点，则ST_Intersects返回true。
ST_Intersection	ST_Intersection(geom1, geom2)	GEOMETRY	返回两个输入几何体之间的交集。

函数	函数语法	返回类型	说明
ST_IspolyGonCW	ST_IsPolygonCW(geom)	BOOLEAN	测试多边形是否具有顺时针方向的外环和逆时针方向的内环。 • 如果输入多边形是逆时针的，则ST_IspolyGonCW返回true。 • 如果输入几何是点、线串、多点或多线串，则返回true。 • 对于几何体集合，如果集合中的所有几何体均为逆时针方向，ST_IspolygonCW将返回true。
ST_IsClosed	ST_IsClosed(geom)	BOOLEAN	如果输入几何体已闭合，则ST_IsClosed返回 true。如下内容定义闭合的几何体： • 输入的几何体是一个点或一个多点。 • 输入几何体是一个线串，并且该线串的起点和终点是重合的。 • 输入几何体是一个非空的多线串，并且其所有线串均已闭合。 • 输入几何体是一个非空多边形，所有多边形的环都是非空的，并且所有环的起点和终点都是重合的。 • 输入几何体是一个非空的多边形集合，并且其所有多边形均已闭合。 • 输入几何体是一个非空几何体集合，并且其所有组件均已闭合。
ST_IsCollection	ST_IsCollection(geom)	BOOLEAN	如果输入几何体为下列子类型之一，则ST_IsCollection返回true： • GEOMETRYCOLLECTION • MULTIPOINT • MULTILINESTRING • MULTIPOLYGON
ST_IsEmpty	ST_IsEmpty(geom)	BOOLEAN	如果输入几何体为空，则ST_IsEmpty返回true。
ST_IsSimple	ST_IsSimple(geom)	BOOLEAN	如果输入几何体没有异常几何点，如自交或自切线，则ST_IsSimple返回true。
ST_IsValid	ST_IsValid(geom)	BOOLEAN	如果输入几何体有效（结构稳定），则ST_IsVALID返回 true。

函数	函数语法	返回类型	说明
ST_Length	ST_Length(geom)	DOUBLE	ST_Length返回输入线性几何体的笛卡尔长度。长度单位与用于表示输入几何体坐标的单位相同。 • 对于点、多点和平面几何体，此函数返回零。 • 当输入为几何体集合时，此函数返回集合中的几何体长度之和。
ST_LineFromMultiPoint	ST_LineFromMultiPoint(geom)	GEOMETRY	ST_LineFromMultiPoint返回输入多点几何体中的线串。点的顺序将保留。返回几何体的SRID与输入几何体的相同。
ST_LineInterpolatePoint	ST_LineInterpolatePoint(geom, fraction) geom子类型必须为LINESTRING，fraction是一个DOUBLE数据类型的值，介于0和1之间。	子类型POINT的GEOMETRY。	ST_LineInterpolatePoint为插值点，即返回一个距离线起点为小数距离的点。
ST_MakeEnvelope	ST_MakeEnvelope(xmin, ymin, xmax, ymax) ST_MakeEnvelope(xmin, ymin, xmax, ymax, srid) x和y均为一个DOUBLE类型的值。	GEOMETRY子类型的EMR群集POINT、LINESTRING或者POLYGON。	从X和y的最小值和最大值创建一个矩形多边形。 • 如果输入坐标指定一个点，则返回的几何体是一个点。 • 如果输入坐标指定一条线，则返回的几何为线串。 • 如果输入坐标指定框的左下角和右上角，则返回的几何为面。 输入的值必须在SRID指定的空间参考系统中，如果没有指定SRID，则SRID默认为0。
ST_MakeLine	ST_MakeLine(geom1, geom2)	子类型LINESTRING的GEOMETRY。	ST_MakeLine从输入几何体创建线串。
ST_MakePoint	ST_MakePoint(x, y)	子类型 POINT的GEOMETRY。	ST_MakePoint返回其坐标值为输入值的点。

函数	函数语法	返回类型	说明
ST_Multi	ST_Multi(geom)	GEOMETRY带子类型的MULTIPOINT、MULTILINESTRING、MULTIPOLYGON，或者GEOMETRYCOLLECTION。	ST_multi将几何体转换为相应的多类型。 • 如果输入几何已经是多类型或几何集合，则其返回值不变。 • 如果输入几何体是点、线串或多边形，则分别返回包含输入几何体的多点、多线串或多边形。
ST_NPoints	ST_NPoints(geom)	INTEGER	ST_NPoints返回输入几何体中点的数量。
ST_NRings	ST_NRings(geom)	INTEGER	ST_NRings返回输入几何体中环的数量。
ST_NumGeometries	ST_NumGeometries(geom)	INTEGER	ST_NumGeometries返回输入几何体集合中的几何体数量。
ST_NumInteriorRings	ST_NumInteriorRings(geom)	INTEGER	ST_NumInteriorRings返回输入多边形几何体中的环的数量。
ST_NumPoints	ST_NumPoints(geom)	INTEGER	ST_NumPoints返回LineString或CircularString中点的数量。
ST_Perimeter	ST_Perimeter(geom)	DOUBLE	ST_Perimeter返回输入平面几何体的笛卡尔周长（边界长度）。周长单位与用于表示输入几何体坐标的单位相同。 • 当输入为点、多点和线性几何体，此函数返回0。 • 当输入为几何体集合时，此函数返回集合中的几何体周长之和。
ST_Point	ST_Point(x, y)	子类型 POINT 的GEOMETRY。	ST_Point从输入坐标值返回点。
ST_PointN	ST_PointN(geom, index)	子类型 POINT 的GEOMETRY。	ST_PointN返回由索引值指定的线串中的点。负索引值从线串的末尾开始倒计数，因此-1是最后一个点。
ST_Points	ST_Points(geom)	子类型MULTIPOINT的GEOMETRY。	ST_Points返回包含输入几何体中所有非空点的多点。ST_Points不会移除输入中重复的点，包括环形几何的起点和终点。

函数	函数语法	返回类型	说明
ST_Polygon	ST_Polygon(linestring, srid) linestring为一个GEOMETRY类型的值，或一个计算结果为GEOMETRY类型的表达式。子类型必须是表示线串的LINESTRING。	子类型POLYGON的GEOMETRY。	ST_Polygon返回一个多边形，其外部环形是输入线串，其值是SRID的输入值。
ST_RemovePoint	ST_RemovePoint(geom, index)	GEOMETRY	ST_RemovePoint返回一个线串，该几何体已删除输入几何体在索引位置的点。索引是从零开始的。返回线串的SRID与输入几何体的相同。
ST_Reverse	ST_Reverse(geom)	GEOMETRY	ST_Revert可反转几何体的顶点顺序。 • 对于点或多点，则其返回值不变。 • 对于几何体集合，ST_Revert将反转集合中每个几何体的顶点顺序。
ST_setPoint	ST_SetPoint(geom1, index, geom2)	GEOMETRY	用给定的点替换索引指定的输入线串中的点。
ST_SetSRID	ST_SetSRID(geom, srid)	GEOMETRY	ST_SetSRID返回一个与输入几何体相同的几何体，但是其SRID值进行了更新。
ST_Simplify	ST_Simplify(geom, tolerance) tolerance为一个DOUBLE类型的值，表示Ramer-Douglas-Peucker算法的容差水平。如果公差是负数，则取值为零。	GEOMETRY	ST_Simplify使用Ramer-Douglas-Peucker算法返回输入几何体的简化版本。输入几何的拓扑可能不会保留。
ST_SRID	ST_SRID(geom)	INTEGER	ST_SRID返回输入几何体的SRID值。
ST_StartPoint	ST_StartPoint(geom)	GEOMETRY	ST_StartPoint返回输入线串的第一个点。返回几何体的SRID值与输入几何体相同。
ST_Touches	ST_Touches(geom1, geom2)	BOOLEAN	如果两个输入几何体接触，则ST_Touches返回true。即，如果两个几何体是非空的、相交并且没有共同的内部点，则它们是接触的。
ST_Within	ST_Within(geom1, geom2)	BOOLEAN	如果第一个输入几何体在第二个输入几何体中，则ST_Within返回true。例如，如果几何体A中的每个点均为几何体B中的一个点，并且其内部有非空相交区域，则几何体A在几何体B中。ST_Within(A, B)与ST_Contains(B, A)等效。

函数	函数语法	返回类型	说明
ST_X	ST_X(point) point为GEOMETRY数据类型的值。	DOUBLE	ST_X返回输入点的X坐标。
ST_XMax	ST_XMax(geom)	DOUBLE	ST_XMax返回输入几何体最大的X坐标。
ST_XMin	ST_XMin(geom)	DOUBLE	ST_XMin返回输入几何体最小的X坐标。
ST_Y	ST_Y(point)	DOUBLE	ST_Y返回输入点的Y坐标。
ST_YMax	ST_YMax(geom)	DOUBLE	ST_YMax返回输入几何体最大的Y坐标。
ST_YMin	ST_YMin(geom)	DOUBLE	ST_YMin返回输入几何体最小的Y坐标。

空间函数最佳实践

阿里云为您提供了空间函数使用方法的最佳实践，详情请参见使用空间函数查询数据方法。

6.4.3. 聚合视图（Beta）

聚合视图实现了对单张表在固定维度列上的预聚合能力，通过预聚合和自动的查询重写，使得统计类查询可以利用预聚合的结果加速查询。本文将会为您介绍在hologres中聚合视图的用法。

使用限制

在Hologres中使用聚合视图，具体限制如下：

- 该功能仅Hologres V0.10及以上版本支持，请在Hologres管控台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- 在使用聚合视图功能之前，需要Superuser在DB内执行以下语句安装扩展包才可以使用聚合视图。一个DB只需执行一次即可，如果创建新的DB，还需要再次执行如下语句。

```
--创建extension
create extension  aggregate_view;
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP extension aggregate_view;
> ```

创建聚合视图

聚合视图通过函数hg_aggregate_view_create创建，具体内容如下：

- **语法说明**

```
call hg_aggregate_view_create(
    aggregate_dst_view ,--目标view名
    aggregate_src_table ,--待聚合表名
    aggregate_keys_non_time ,--非时间粒度待聚合key
    aggregate_key_time , --时间粒度聚合key
    aggregate_values ,--待聚合的列名。按照顺序排列，不可重复
    aggregate_funcs ,--聚合函数，支持sum,min,max,count,avg,approx_count_distinct
    aggregate_timestamp_column ,--刷新时间列，精确时间列，类型须是timestamptz
    aggregate_allowed_lateness ,--允许晚到时间
    aggregate_base_table_ttl --聚合数据保存时间 second
);
```

- **参数说明**

参数	类型	说明
aggregate_dst_view	text	目标View名。
aggregate_src_table	text	待聚合表名。
aggregate_keys_non_time	text[]	非时间粒度待聚合key。
aggregate_key_time	text	时间粒度待聚合key。时间维度，查询时展示的时间维度，同时也是查询指定的时间范围。
aggregate_values	text[]	待聚合的列名。按照顺序排列，不可重复。
aggregate_funcs	text[]	聚合函数。支持的聚合函数具体如下： ○ sum：求和函数。 ○ min：最小值函数。 ○ max：最大值函数。 ○ count：计数函数。 ○ avg：平均值函数。 ○ approx_count_distinct：计算某一列去重后的行数，但为近似值，结果为二进制类型，需调用approx_count_distinct_final返回数值，支持使用approx_count_distinct_merge聚合。 ○ rb_count_distinct：计算某一列去重后行数值，只支持int32类型，结果为二进制类型，需调用rb_cardinality返回数值，支持用rb_or_agg聚合，使用前需要创建roaringbitmap extention。如下所示： `create extension roaringbitmap_hqe;`
aggregate_timestamp_column	text	精确时间列，按照该列定时刷新。这一列在源表中必须非空。

参数	类型	说明
aggregate_allowed_lateness	interval	允许数据晚到的时间，数据最晚刷新的时间不超过该时间。需要和精确时间搭配使用，当不指定截止时间调用refresh时，用源表max(aggregate_timestamp_column)-aggregate_allowed_lateness做截止时间。
aggregate_base_table_ttl	int	聚合视图内数据保存的时间，单位为秒（s），超过该时间，数据将会被清除。

- **使用示例**

  本样例提供的样本数据具体如下：

event_id	uid	user_id	event_time	province	city	PV	charge	event_count
1002	User10011	10011	2020-11-11 00:01:11	hebei	tangshan	1	10	10
1002	User10012	10012	2020-11-11 00:01:12	shandong	qingdao	2	11	11
1003	User10013	10013	2020-11-11 00:01:13	hunan	changsha	2	11.9	11
1002	User10012	10012	2020-11-11 00:01:14	shandong	qingdao	1	1.1	9
1002	User10012	10012	2020-11-11 00:01:14	shandong	qingdao	1	1.1	9

  对应数据创建如下表：

```
--建表并插入样本数据
CREATE TABLE agg_src_table
(
    event_id INT
    ,user_id TEXT
    ,uid INT
    ,event_time timestamptz NOT NULL
    ,province TEXT
    ,city TEXT
    ,pv INT
    ,charge NUMERIC(15,2)
    ,event_count INT
);
insert into agg_src_table values
(1002, 'User10011', 10011, '2020-11-11 00:01:11', 'hebei', 'tangshan', 1, 10,10 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:12', 'shandong', 'qingdao', 2, 11,11 ),
(1003, 'User10013', 10013, '2020-11-11 00:01:13', 'hunan', 'changsha', 2, 11.9, 11 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:14', 'shandong', 'qingdao', 1, 1.1, 9 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:14', 'shandong', 'qingdao', 1, 1.1, 9 );
--对表初步加工，主要是创建一个view并将时间事件截取到小时粒度。
CREATE view agg_src_view AS
SELECT  province
        ,city
        ,event_time
        ,date_trunc('hour', event_time) AS event_date
        ,pv
        ,charge
        ,event_count
        ,user_id
        ,uid
FROM    agg_src_table ;
```

对应创建聚合视图示例如下：

```
--创建聚合视图扩展
create extension  aggregate_view;
--创建聚合视图
call hg_aggregate_view_create(
  'agg_dst_view', --目标view名
  'agg_src_view', --待聚合的表名，此处以初步加工后的view代替
  ARRAY['province', 'city'], --按照省份province和城市city聚合
  'event_date', --按照事件时间粒度聚合
  ARRAY['event_count', 'charge', 'pv', 'user_id'],
  ARRAY['count','avg', 'sum', 'approx_count_distinct'],--计算event_count列的count，charge列的avg，pv列的sum，user_id列的近似去重，
  'event_time', --按照该列进行定时刷新
  '600 sec', --允许数据晚到600s
  '2592000'--聚合数据保存时间
    );
 --查询视图
 SELECT * FROM agg_dst_view;
```

刷新聚合视图

刷新聚合视图可以将实时写入的新数据更新到聚合视图中，主要包括截止时间刷新和区间刷新。同一时间同一个View，只可以有一个刷新任务。具体内容如下：

- **截止时间刷新**
  - 功能说明

    截止时间刷新可以将指定的截止时间数据刷新。如果是增量刷新（默认），上次截止时间到本次截止时间的数据会被刷新，如果是全量刷新，截止时间之前的数据都会被刷新。具体语法如下：

    ```
    call hg_aggregate_view_refresh('aggregate_dst_view', 'refresh_until_timestamp','delta')
    ;
    ```

  - 参数说明

参数	类型	说明
aggregate_dst_view	text	目标View名。
refresh_until_timestamp	timestamptz	刷新截止时间戳。 aggregate_timestamp_column小于此值的记录都会被聚合写入basetable。缺省情况下refresh_until_timestamp取max(aggregate_timestamp_column)-aggregate_allowed_lateness的值。
delta	bool	是否增量更新。 ■ true：默认为true，增量更新，上次截止时间到这次截止时间的数据会被刷新。 ■ false：全量刷新，截止时间之前的数据都会被刷新，会将basetable中的数据清空再写入。

  - 使用示例

    如下示例将刷新截止到2020-11-11 01:00:00之前写入的数据。

    ```
    --插入新的数据
    insert into agg_src_table values
    (1002, 'User10998', 10320, '2020-11-11 00:01:15', 'sichuan', 'chengdu', 2, 3,1 ),
    (1003, 'User10007', 10343, '2020-11-12 00:01:15', 'zhejiang', 'hangzhou', 7, 2,4 ),
    (1003, 'User10073', 10221, '2020-11-12 00:01:16', 'anhui', 'huangshan', 3, 8,4 );
    --增量刷新
    call hg_aggregate_view_refresh('agg_dst_view', '2020-11-11 01:00:00'::timestamptz,'true
    ');
    --全量刷新
    call hg_aggregate_view_refresh('agg_dst_view', '2020-11-11 00:00:00'::timestamptz,'fals
    e');
    ```

- **区间刷新**
  - 功能说明

    区间刷新可以将指定某个时间区间内的数据更新到聚合视图中。具体语法如下：

    ```
    call hg_aggregate_view_range_refresh('aggregate_dst_view','refresh_start_timestamp','re
    fresh_until_timestamp');
    ```

- 参数说明

参数	类型	说明
aggregate_dst_view	text	目标View名。
refresh_until_timestamp	timestamptz	刷新截止时间戳。
refresh_start_timestamp	timestamptz	刷新开始时间戳。

- 使用示例

  如下示例指定将2020-11-11 01:00:00到2020-11-12 01:00:00期间写入的数据进行刷新。

```
--刷新截止到（当前时间-允许晚到区间）
call hg_aggregate_view_range_refresh('agg_dst_view','2020-11-11 01:00:00'::timestamptz,
'2020-11-12 01:00:00'::timestamptz);
```

删除聚合视图

当您不需要聚合视图的时候可以对其进行删除。

- 语法示例

```
call hg_aggregate_view_delete('aggregate_dst_view');
```

- 参数说明

  aggregate_dst_view：表示需要删除的聚合视图名。

- 使用示例

  如下示例将删除名称为agg_dst_view的聚合视图。

```
call hg_aggregate_view_delete('agg_dst_view');
```

使用示例

本次示例以某网站访问数据为例，通过聚合视图定期计算，对用户ID做近似去重和精确去重，计算事件类型event_count的访问量、平均收费情况（charge列的avg）、总PV（pv列的sum）。该聚合视图通过事件时间event_time列刷新，累计600秒之前的汇聚结果，视图的数据保存2592000秒，过期将会被删除。

- 建表并插入样本数据。

```
CREATE TABLE agg_src_table
(
    event_id INT
    ,user_id TEXT
    ,uid INT
    ,event_time timestamptz NOT NULL
    ,province TEXT
    ,city TEXT
    ,pv INT
    ,charge NUMERIC(15,2)
    ,event_count INT
);
insert into agg_src_table values
(1002, 'User10011', 10011, '2020-11-11 00:01:11', 'hebei', 'tangshan', 1, 10,10 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:12', 'shandong', 'qingdao', 2, 11,11 ),
(1003, 'User10013', 10013, '2020-11-11 00:01:13', 'hunan', 'changsha', 2, 11.9, 11 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:14', 'shandong', 'qingdao', 1, 1.1, 9 ),
(1002, 'User10012', 10012, '2020-11-11 00:01:14', 'shandong', 'qingdao', 1, 1.1, 9 );
```

- 对表初步加工，主要是创建一个view并将时间事件截取到小时粒度。

```
CREATE view agg_src_view AS
SELECT  province
        ,city
        ,event_time
        ,date_trunc('hour', event_time) AS event_date
        ,pv
        ,charge
        ,event_count
        ,user_id
        ,uid
FROM    agg_src_table ;
```

- 创建聚合视图。

> **说明** 对用户ID精确去重，需要开启RoaringBitmap扩展使用rb_count_distinct函数。更多关于RoaringBitmap函数的说明，请参见Roaring Bitmap函数。

```
--创建聚合视图扩展
create extension  aggregate_view;
--创建Roaringbitmap扩展
create extension  roaringbitmap;
--创建聚合视图，在province，city维度聚合，基于event_date日期粒度，计算event_count列的count、charge列的avg，pv列的sum，user_id列的approx_count_distinct、uid列的rb_count_distinct算子，刷新时间戳为event_time列，累计600秒之前的汇聚结果，数据保存2592000秒
call hg_aggregate_view_create(
  'agg_dst_view',
  'agg_src_view',
  ARRAY['province', 'city'],
  'event_date',
  ARRAY['event_count', 'charge', 'pv', 'user_id', 'uid'],
  ARRAY['count','avg', 'sum', 'approx_count_distinct', 'rb_count_distinct'],
  'event_time',
  '600 sec',
  '2592000');
```

- 数据查询和插入处理。

```
--业务查询语句
SELECT  province
        ,city
        ,event_date
        ,event_count
        ,charge
        ,pv
        ,approx_count_distinct_final(user_id) AS user_id
        ,rb_cardinality(uid) AS uid
FROM    agg_dst_view
WHERE   event_date > '2020-11-01'::timestamptz
AND     event_date < '2020-11-12'::timestamptz;
--插入新的数据
insert into agg_src_table values
(1002, 'User10998', 10320, '2020-11-11 00:01:15', 'sichuan', 'chengdu', 2, 3,1 ),
(1003, 'User10007', 10343, '2020-11-12 00:01:15', 'zhejiang', 'hangzhou', 7, 2,4 ),
(1003, 'User10073', 10221, '2020-11-12 00:01:16', 'anhui', 'huangshan', 3, 8,4 );
--刷新截止到(当前表最大时间-允许晚到区间)  '2020-11-12 00:01:16' - '600 sec' = '2020-11-11 23:
51:16'
call hg_aggregate_view_refresh('agg_dst_view');
--刷新截止到2020-11-12 01:20:00之前的数据
call hg_aggregate_view_refresh('agg_dst_view', '2020-11-12 01:20:00'::timestamptz,'true')
;
--带有近似去重函数的查询语句
SELECT  city
        ,approx_count_distinct_final(approx_count_distinct_merge(user_id)) AS user_id
FROM    agg_dst_view
WHERE   event_date > '2020-11-11'::timestamptz
AND     event_date < '2020-11-13'::timestamptz
GROUP BY city;
--带有精确去重函数的查询语句
  SELECT  city
        ,SUM(pv)
        ,rb_cardinality(rb_or_agg(uid)) AS uid
FROM    agg_dst_view
WHERE   event_date > '2020-11-10'::timestamptz
AND     event_date < '2020-11-12'::timestamptz
GROUP BY city;
--再聚合查询
SELECT  province
        ,SUM(event_count)
        ,SUM(pv)
        ,approx_count_distinct_final(approx_count_distinct_merge(user_id)) AS user_id
        ,rb_cardinality(rb_or_agg(uid)) AS uid
FROM    agg_dst_view
WHERE   event_date > '2020-11-10'::timestamptz
AND     event_date < '2020-11-15'::timestamptz
GROUP BY province;
```

- 删除聚合视图。

```
--删除聚合视图
call hg_aggregate_view_delete('agg_dst_view');
```

6.4.4. Proxima向量计算

本文为您介绍在Hologres中如何使用向量计算功能。

背景信息

Proxima是一款来自于阿里达摩院的实现向量近邻搜索的高性能软件库，相比于Faiss等开源的同类产品，Proxima在稳定性、性能等方面都要更为出色，能够提供业内性能和效果领先的基础方法模块，支持图像、视频、人脸等各种应用场景。Hologres向量查询功能与Proxima深度整合，提供高性能的向量查询服务。

Proxima简介

- 名词解释
  - 特征向量：向量是一种将实体和应用代数化的一种表示，其将实体间的关系抽象成向量空间中的距离，而距离的远近代表着相似程度。例如：身高、年龄、性别、地域。
  - 向量检索：在特征向量数据集合中进行快速搜索和匹配的方法，常涉及到的问题有KNN和RNN。
  - KNN（K-Nearest Neighbor）：查找离查询点最近的 K 个点。
  - RNN（Radius Nearest Neighbor）：查找离查询点某半径范围内的所有点。
- Proxima的基本模型

  分为索引构建和在线检索两部分：
  - 索引构建：索引构建从原始向量数据中构建出相关索引文件，并传予在线检索模块加载使用。支持Brute Force、KD-Tree、Product Quantization、KNN Graph、LSH 等。
  - 在线检索：在线检索加载完索引文件后，为向量检索提供查询服务。在聚类后的数据集上进行KNN和RNN搜索，用户设定检索过程中的参数。
- Proxima与Hologres概念对照

Proxima概念	Hologres中的概念
特征向量	数组类型 Array，仅支持固定长度数组
向量索引	一种特殊类型的Index，当前仅支持KNN/RNN的Graph索引
距离计算	◦ 一种类型的UDF：`proxima_distance()` ◦ 每种距离计算对应一个UDF
KNN查询	order by distance(x, [x1, x2]) asc limit k
RNN查询	where distance(x, [x1,x2]) < r

使用Proxima进行向量计算

Hologres中使用Proxima进行向量计算的操作步骤如下：

1. 连接Hologres

   通过开发工具连接Hologres，详情请参见连接开发工具。如果是JDBC连接，请使用Prepare Statement模式。
2. 安装Proxima插件。

Proxima是以一个extension的方式与Hologres连通，因此在使用之前需要Superuser执行如下命令安装Proxima插件。

```
--安装extension
create extension proxima;
```

Proxima插件是针对数据库级别使用的，一个数据库仅需要安装一次即可。

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP EXTENSION proxima;
> ```

3. 创建向量表

向量在Hologres中一般用FLOAT4数组表示，创建向量表的语法如下。

> **说明** 仅列存表支持向量索引。

```
begin;
create table feature_tb (
    id bigint,
    <feature_col> float4[] check(array_ndims(<feature_col>) = <value> and
array_length(feature_col, 1) = <value>) --定义向量
);
-- 构建向量索引
call set_table_property('feature_tb', 'proxima_vectors', '{"<feature_col>":
{"algorithm":"<value>","distance_method":"<value>"}}');
commit;
```

参数说明如下。

分类	参数	描述	示例
向量基本属性	feature_col	向量列名称。	feature。
	array_ndims	向量的维度。	构建一维且长度为4的向量示例如下。 `feature float4[] check(array_ndims(feature) = 1 and array_length(feature, 1) = 4)`
	array_length	向量的长度。	

分类	参数	描述	示例
向量索引	proxima_vectors	代表构建向量索引，其中： ○ algorithm：用于指定构建向量索引的算法，目前仅支持 `Graph` 。 ○ distance_method：用于定义构建向量索引使用的距离计算方法，目前支持三种距离计算函数： ■ （推荐使用）SquaredEuclidean：平方欧式距离， 查询效率最高。适合查询时使用 `pm_approx_squared_euclidean_distance` 。 ■ Euclidean：开方的欧式距离，仅适合查询时使用 `pm_approx_euclidean_distance` ，如果使用其他距离函数会利用不上索引。 ■ InnerProduct（避免使用）：内积距离，会在底层转换为开方的欧式距离的计算，所以构建索引和查询索引都会多一层计算开销，比较低效，尽量避免使用，除非业务有强需求。仅适合查询时使用 `pm_approx_inner_product_distance` 。 ○ min_flush_proxima_row_count：用于控制开始构建向量索引的数量量，默认为20000，一般场景无需修改。 **说明** 索引需要在一定的场景下使用才能发挥更好的作用。	--使用平方欧式距离查询，构建对应的向量索引示例如下。 `call set_table_property('feature_tb', 'proxima_vectors', '{"feature": {"algorithm":"Graph", "distance_method":"SquaredEuclidean", "builder_params": {"min_flush_proxima_row_count" : 20000} }}');`

4. 向量导入

   可以通过离线或者实时的方式将数据导入至向量表，详情请参见数据同步概述，可以根据业务需求选择合适的同步方式。但需要注意的是，在批量导入后，执行VACUUM和Analyze命令以提升查询效率。

   ○ VACUMM会让后端的文件compaction成更大的文件，对查询更高效。 但是VACUMM需要耗费一定的CPU资源，表的数据量越大，执行VACUMM的时间越久，当VACUMM还在执行中时，请耐心等待执行

结果。

```
VACUMM <tablename>;
```

- Analyze是收集统计信息， 用于优化器QO（Query Optimizer）生成较优的执行计划，提高查询性能。

```
analyze <tablename>;
```

5. 向量查询。

Hologres支持精确和近似向量查询，其中以 `pm_` 开头的UDF都为精准查询，以 `pm_approx_` 开头的UDF都为近似查询。

> **说明** 对于构建向量索引的场景，建议您使用 `pm_approx_` 开头的UDF近似查询，查询效率会更高。

使用如下的方式查询Top N，近似查询中的第二个参数必须是常量值。

```
-- 计算内积的TOPK，此时建表里面的proxima_vector参数的distance_method需要为SquaredEuclidean
select pm_approx_squared_euclidean_distance(feature, '{0.1,0.2,0.3,0.4}') as distance f
rom feature_tb order by distance asc limit 10 ;
-- 计算内积的TOPK，此时建表里面的proxima_vector参数的distance_method需要为Euclidean
select pm_approx_euclidean_distance(feature, '{0.1,0.2,0.3,0.4}') as distance from feat
ure_tb order by distance asc limit 10 ;
-- 计算内积的TOPK，此时建表里面的proxima_vector参数的distance_method需要为InnerProduct
select pm_approx_inner_product_distance(feature, '{0.1,0.2,0.3,0.4}') as distance from
feature_tb order by distance desc limit 10 ;
```

向量计算UDF列表

Hologres当前版本支持的向量计算UDF如下。

- 精准查询

```
float4 pm_squared_euclidean_distance(float4[], float4[])
float4 pm_euclidean_distance(float4[], float4[])
float4 pm_inner_product_distance(float4[], float4[])
```

- 近似查询

```
float4 pm_approx_squared_euclidean_distance(float4[], float4[])
float4 pm_approx_euclidean_distance(float4[], float4[])
float4 pm_approx_inner_product_distance(float4[], float4[])
```

完整使用示例

使用Proxima进行向量计算的示例语句如下。

```
create extension proxima;
begin;
create table feature_tb (
    id bigint,
    feature float4[] check(array_ndims(feature) = 1 and array_length(feature, 1) = 4)
);
call set_table_property('feature_tb', 'proxima_vectors', '{"feature":{"algorithm":"Graph","
distance_method":"SquaredEuclidean","builder_params":
{"min_flush_proxima_row_count" : 20000}}}');
end;
insert into feature_tb select i, array[random(), random(), random(), random()]::float4[] fr
om generate_series(1, 10000) i;
analyze feature_tb;
select pm_approx_squared_euclidean_distance(feature, '{0.1,0.2,0.3,0.4}') as distance from
feature_tb order by distance desc limit 10 ;
```

性能调优

- 设置向量索引的场景

  数据量较小，比如几万条数据情况下，建议不设置索引直接计算即可。 或者实例资源较多的情况下但查询的数据量较少，也可以直接计算。 当直接计算满足不了需求（如延迟、吞吐等要求）时，可以考虑使用Proxima索引， 原因如下。

  - Proxima本身是有损索引，不保证结果的准确性，即计算出来的距离可能是有偏差的。
  - Proxima索引有可能导致召回的条数不足，如 `limit 1000` 情况下，只返回了500条。
  - Proxima索引使用有一定难度。

- 设置合适的Shard Count

  Shard Count越多， 实际构建Proxima索引的文件就越多， 查询吞吐就越差。所以在实际使用中，根据实例资源建议设置合理的Shard Count，最极致的场景是设置Shard Count为1，查询性能最好。但是Shard Count数越少，写入性能将会越差，在实际业务中建议综合评估。

  经验证明Proxima查询使用的计算资源与Shard Count成正比，但是如果没有设置Proxima索引，即为暴力计算，不建议修改Shard Count。

  ```
  --创建向量表，并放于shard count为1的table group中
  begin;
  create table prixima_test (
  group_id text not null,
  user_id text not null,
  model_version text not null,
  feature float4[] check(array_ndims(feature) = 1 and array_length(feature, 1) = 4),    pri
  mary key(group_id,user_id,model_version));
  call set_table_property('prixima_test', 'proxima_vectors', '{"feature":{"algorithm":"Grap
  h","distance_method":"SquaredEuclidean"}}');
  call set_table_property('prixima_test', 'shard_count', '1');
  end;
  ```

- （推荐）不带过滤条件的查询场景

在有 `where` 过滤条件的情况下，会影响索引使用，性能可能更差，所以推荐不带过滤条件的查询场景。对于不带过滤条件的向量检索，最极致的状态就是一个Shard上只有一个向量索引文件，这样查询就能直接落在一个Shard上。

因此对于不带过滤条件的查询场景通常建表如下。

```
begin;
CREATE TABLE feature_tb(
    uuid text,
    feature FLOAT4[] not null CHECK (array_ndims(feature)=1 and array_length(feature,1)=N)--定义向量
) ;
call set_table_property('feature_tb', 'shard_count', '?');--指定shard count，根据业务情况合理设置，若有可以不设置
call set_table_property('feature_tb', 'proxima_vectors', '{"feature":{"algorithm":"Graph","distance_method":"InnerProduct"}}');--构建向量索引
end;
```

- 带过滤条件的查询场景

  对于带过滤条件的向量检索，情况细分为如下常见的过滤场景。

  - 查询场景1：字符串列为过滤条件

    示例查询如下，常见的场景为在某个组织内查找对应的向量数据，例如查找班级内的人脸数据。

    ```
    select pm_xx_distance(feature, '{1,2,3,4}') as d from feature_tb where uuid = 'x' order by d limit 10;
    ```

    建议进行如下优化。

    - 将uuid设置为Distribution Key，这样相同的过滤数据会保存在同一个Shard，查询时一次查询只会落到一个Shard上。
    - 将uuid设置为表的Clustering Key，数据将会在文件内根据Clustering Key排序。

  - 查询场景2：时间字段为过滤条件

    示例查询如下，一般是根据时间字段过滤出对应的向量数据。建议将时间字段time_field设置为表的segment_key，可以快速的定位到数据所在的文件。

    ```
    select pm_xx_distance(feature, '{1,2,3,4}') as d from feature_tb where time_field between '2020-08-30 00:00:00' and '2020-08-30 12:00:00' order by d limit 10;
    ```

  因此对于带过滤条件的向量检索而言，其建表语句通常如下。

```
begin;
CREATE TABLE feature_tb(
time_field timestamptz,
uuid text,
feature FLOAT4[] not null CHECK (array_ndims(feature)=1 and array_length(feature,1)= N)
) ;
call set_table_property('feature_tb', 'distribution_key', 'uuid');
call set_table_property('feature_tb', 'segment_key', 'time_field');
call set_table_property('feature_tb', 'clusering_key', 'uuid');
call set_table_property('feature_tb', 'proxima_vectors', '{"feature":{"algorithm":"Graph"
,"distance_method":"InnerProduct"}}');
end;
-- 如果没有按照时间过滤的话，则time_field相关的索引可以删除。
```

常见问题

- 报错 `ERROR: function pm_approx_inner_product_distance(real[], unknown) does not exist` 。

  原因：通常是因为未在数据库中执行 `create extension proxima;` 语句来初始化Proxima插件。

  解决方法：执行 `create extension proxima;` 语句初始化Proxima插件。

- 报错 `Writing column: feature with array size: 5 violates fixed size list (4) constraint declared in schema` 。

  原因：由于写入到特征向量列的数据维度与表中定义的维度数不一致，导致出现该报错。

  解决方法：可以排查下是否有脏数据。

- 报错 `The size of two array must be the same in DistanceFunction, size of left array: 4, size of right array:` 。

  原因：由于 `pm_xx_distance(left, right)` 中，left的维度与right的维度不一致所致。

  调整 `pm_xx_distance(left, right)` 中，left的维度与right的维度一致。

- 实时写入报错 `BackPresure Exceed Reject Limit ctxId: XXXXXXXX, tableId: YY, shardId: ZZ` 。

  原因：实时写入作业遇到了瓶颈，产生了反压的异常，说明写入作业开销大，写入慢，通常是由于min_flush_proxima_row_count较小，而实时写入速度较大，造成写入作业实时构建索引开销大，阻塞了实时写入。

  调整min_flush_proxima_row_count到更大值。

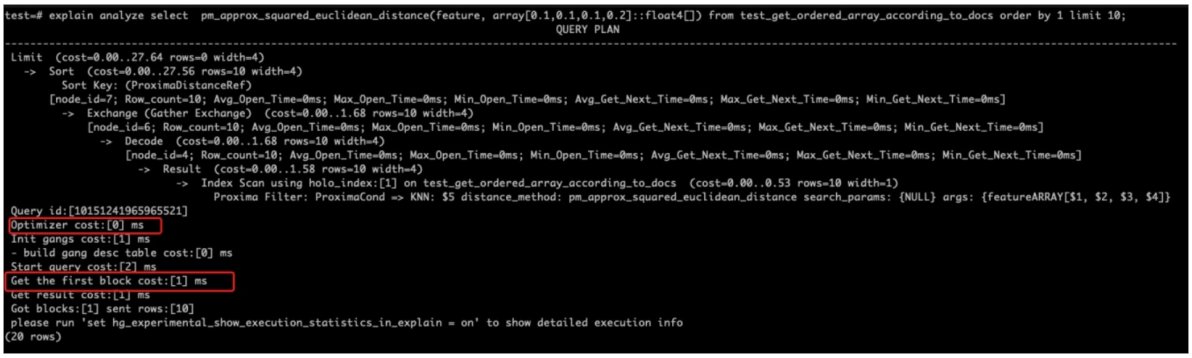
- 如何通过Java写入向量？

  通过Java写入向量的示例如下。

```
private static void insertIntoVector(Connection conn) throws Exception {
    try (PreparedStatement stmt = conn.prepareStatement("insert into feature_tb values(?,
?);")) {
        for (int i = 0; i < 100; ++i) {
           stmt.setInt(1, i);
           Float[] featureVector = {0.1f,0.2f,0.3f,0.4f};
           Array array = conn.createArrayOf("FLOAT4", featureVector);
           stmt.setArray(2, array);
           stmt.execute();
        }
    }
}
```

- 如何通过执行计划检查是否利用上Proxima索引？

  如果执行计划中存在 `Proxima filter: xxxx` 表明使用了索引，如下图所示；如果没有，则索引没有使用上，一般是建表语句与查询语句不匹配。



距离函数说明

Hologres支持的三种向量距离评估函数如下：

- 不开方的欧式距离，计算公式如下。

$$\sum_{i=0}^{n}(u_i - v_i)^2$$

- 开方的欧氏距离，计算公式如下。

$$\sqrt{\sum_{i=0}^{n}(u_i - v_i)^2}$$

- 内积距离，计算公式如下。

$$-\sum_{i=0}^{n} u_i v_i$$

> **说明** 如果您选用欧式距离进行向量计算，不开方的欧式距离与开方的欧式距离相比，可以少一个开方的计算，并且计算出的Top K记录一致。因此，不开方的欧式距离性能更好，在满足功能需求的情况下，一般建议您使用不开方的欧式距离。

6.4.5. 流量分析函数

6.4.5.1. 漏斗分析函数

漏斗分析是常见的转化分析方法，它用于反映用户各个阶段行为的转化率。漏斗分析被广泛应用于用户行为分析和App数据分析的流量分析、产品目标转化等数据运营与数据分析。本文将为您介绍在Hologres中漏斗分析相关函数的使用。

使用限制

Hologres针对各场景的漏斗分析提供了漏斗函数（windowFunnel）、留存函数（retention）、range_retention_count和range_retention_sum四个函数，用于帮助用户进行漏斗分析。具体使用限制如下：

- 仅Hologres V0.9 及以上版本支持漏斗函数（windowFunnel）和留存函数（retention）函数，请前往Hologres管控台的实例详情页查看当前实例版本。
- 仅Hologres V0.10 及以上版本支持range_retention_count和range_retention_sum函数。如果您的实例是V0.9以下版本，请您提交工单升级实例。
- 在使用之前，需要执行以下语句才可以通过语句调用函数。extension是DB级别的函数，一个DB只需执行一次即可。

```
create extension flow_analysis; --开启extension
```

场景说明

本文以一个简化的购买场景为例，介绍漏斗分析相关的函数用法。一般用户完整的购买流程如下：

- 浏览商品
- 收藏商品
- 加入购物车
- 购买商品

其对应的样例数据具体如下：

用户ID（user_id）	事件类型（event_type）	发生事件时间（event_time）
4913	浏览商品	2014-11-18 00:00:00
501286	浏览商品	2014-11-18 00:00:00
501286	加入购物车	2014-11-18 00:00:00
632347	浏览商品	2014-11-18 00:00:00
814199	浏览商品	2014-11-18 00:00:00
814199	收藏商品	2014-11-18 00:00:00
1259845	浏览商品	2014-11-18 00:00:00
1259845	购买商品	2014-11-18 00:00:00

用户ID（user_id）	事件类型（event_type）	发生事件时间（event_time）
1498131	浏览商品	2014-11-18 00:00:00
1498131	收藏商品	2014-11-18 00:00:00
1857066	浏览商品	2014-11-18 00:00:00
1926899	浏览商品	2014-11-18 00:00:00
3318666	浏览商品	2014-11-18 00:00:00
3324323	浏览商品	2014-11-18 00:00:00
3442818	浏览商品	2014-11-18 00:00:00

对应的建表语句如下所示：

```
BEGIN;
DROP TABLE IF EXISTS public.user_analysis;
CREATE TABLE IF NOT EXISTS public.user_analysis
(
    user_id INT NOT NULL ,
      event_type    TEXT ,
    event_time TIMESTAMP NOT NULL
);
call set_table_property('user_analysis', 'distribution_key', 'user_id');
call set_table_property('user_analysis', 'segment_key', 'event_time');
call set_table_property('user_analysis', 'clustering_key', 'event_time');
call set_table_property('user_analysis', 'bitmap_columns', 'event_type');
COMMIT;
```

漏斗函数（windowFunnel）

- **函数说明**

  漏斗函数（windowFunnel）可以搜索滑动时间窗口中的事件列表，并计算条件匹配的事件列表的最大长度。

  搜索事件列表，从第一个事件开始匹配，依次做最长、有序匹配，返回匹配的最大长度。一旦匹配失败，结束整个匹配。

  假设在窗口足够大的条件下：

  - 条件事件为c1，c2，c3，而用户数据为c1，c2，c3，c4，最终匹配到c1，c2，c3，函数返回值为3。
  - 条件事件为c1，c2，c3，而用户数据为c4，c3，c2，c1，最终匹配到c1，函数返回值为1。
  - 条件事件为c1，c2，c3，而用户数据为c4，c3，最终没有匹配到事件，函数返回值为0。

- **函数语法**

  ```
  windowFunnel(window, mode, timestamp, cond1, cond2, ..., condN)
  ```

- **参数说明**

参数	说明
window	窗口大小，即从指定的第一个事件开始，往后推移一个窗口来提取相关事件数据。
mode	模式。支持default和strict两种模式 ，默认为default。当设置strict时，`windowFunnel()` 仅对唯一值应用匹配条件。
timestamp	包含时间的列，支持timestamp、int、bigint类型。
cond	事件的每个步骤。

- **使用示例**

  如果您希望分析一段时间内，用户按照固定转化路径的转化漏斗情况，可以参照如下SQL进行分析，SQL中的各个条件如下：

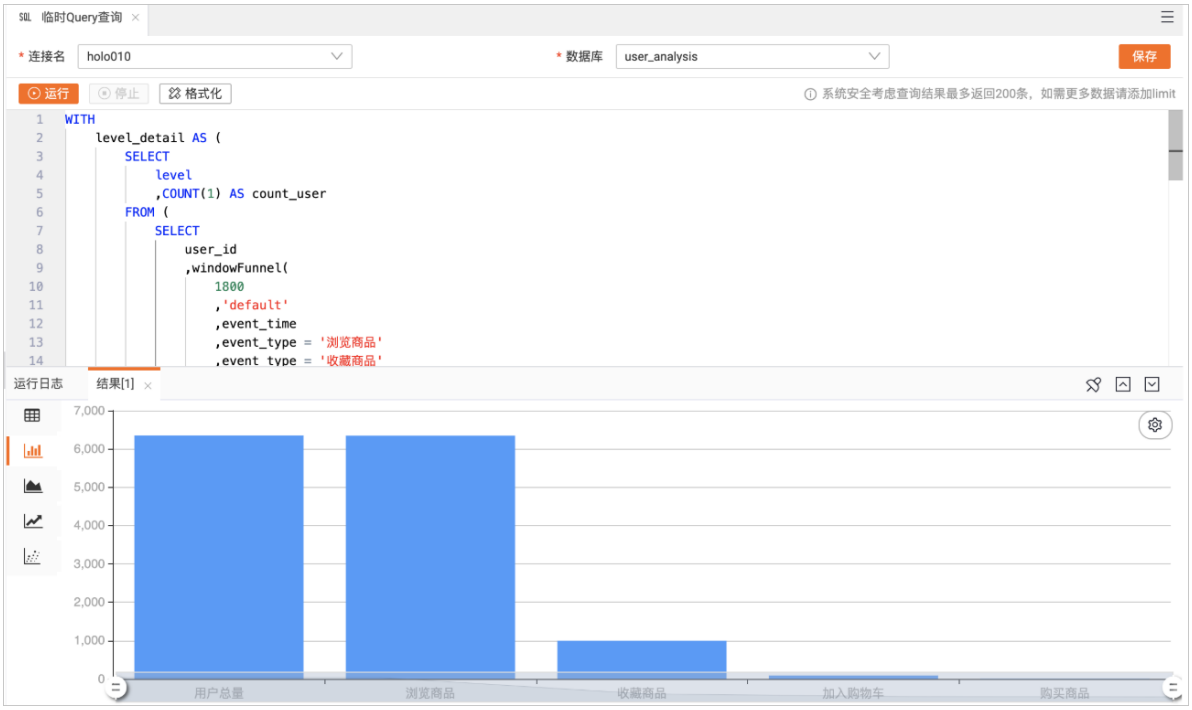
  - 统计间隔：30分钟（即1800秒）
  - 统计时间段：2014-11-25 00:00:00至2014-11-26 00:00:00
  - 转化路径：浏览商品>收藏商品>加入购物车>购买商品

```
WITH
    level_detail AS (
        SELECT
            level
            ,COUNT(1) AS count_user
        FROM (
            SELECT
                user_id
                ,windowFunnel(
                    1800
                    ,'default'
                    ,event_time
                    ,event_type = '浏览商品'
                    ,event_type = '收藏商品'
                    ,event_type = '加入购物车'
                    ,event_type = '购买商品'
                    ) AS level
            FROM public.user_analysis
            WHERE event_time >= TIMESTAMP '2014-11-25 00:00:00'
                AND event_time < TIMESTAMP '2014-11-26 00:00:00'
            GROUP BY user_id
            ) AS basic_table
        GROUP BY level
        ORDER BY level ASC )
SELECT  CASE level    WHEN 0 THEN '用户总量'
                      WHEN 1 THEN '浏览商品'
                      WHEN 2 THEN '收藏商品'
                      WHEN 3 THEN '加入购物车'
                      WHEN 4 THEN '购买商品'
              END
        ,SUM(count_user) over ( ORDER BY level DESC )
FROM    level_detail
GROUP BY level
         ,count_user
ORDER BY level ASC
;
```

显示结果如下所示：

```
    case    | sum
------------+------
 用户总量    | 6351
 浏览商品    | 6346
 收藏商品    |  997
 加入购物车  |   88
 购买商品    |    9
(5 rows)
```

如果您使用HoloWeb执行查询，即可直接得到一张漏斗图。



留存函数（retention）

- **函数说明**

  该函数将一组条件作为参数，类型为1到32个UInt8类型的参数，用来表示事件是否满足特定条件。

- **函数语法**

```
retention(cond1, cond2, ..., cond32);
```

- **参数说明**

参数	说明
cond	返回结果的表达式。返回值包括： ○ 1，条件满足。 ○ 0，条件不满足。

- **使用示例**

  如果您希望分析从某一天开始，用户的留存情况，可以使用如下SQL进行分析，SQL中的场景如下：

  ○ 开始分析日期是2014年11月25日。
  ○ 分别统计了第一天活跃数、次日留存、3日留存和7日留存。

```
-- 根据计算数组中SUM(r[index])获取2014-11-25活跃用户在第2、3、7日的留存数，以此计算留存率
SELECT
    DATE '2014-11-25 00:00:00' AS "访问日期",
    SUM(r[1])::NUMERIC AS "第1天活跃用户",
    SUM(r[2])::NUMERIC/SUM(r[1])::NUMERIC AS "次日留存",
    SUM(r[3])::NUMERIC/SUM(r[1])::NUMERIC AS "3日留存",
    SUM(r[4])::NUMERIC/SUM(r[1])::NUMERIC AS "7日留存"
FROM
-- 计算2014-11-25活跃用户在第2、3、7日的登录情况。r数组表示每天登录情况，1表示登录，0 表示未登录。
    (
        WITH
            first_day_table AS ( SELECT TIMESTAMP '2014-11-25 00:00:00' AS first_day)
        SELECT
            user_id,
            retention(
                DATE(event_time) = (SELECT DATE(first_day) FROM first_day_table),
                DATE(event_time) = (SELECT DATE(first_day + INTERVAL '1 day') FROM first_
day_table),
                DATE(event_time) = (SELECT DATE(first_day + INTERVAL '2 day') FROM first_
day_table),
                DATE(event_time) = (SELECT DATE(first_day + INTERVAL '6 day') FROM first_
day_table)
                ) AS r
-- 过滤2014-11-25活跃用户在后续1～7日登录数据
            FROM    public.user_analysis
            WHERE   (event_time >= TIMESTAMP '2014-11-25 00:00:00')
            AND     (event_time <= TIMESTAMP '2014-11-25 00:00:00' + INTERVAL '6 day')
            GROUP BY user_id
        ) AS basic_table
GROUP BY "访问日期"
;
```

显示结果如下所示：

```
  访问日期   | 第1天活跃用户 | 次日留存 | 3日留存  | 7日留存
-------------+---------------+----------+----------+----------
 2014-11-25 |          6351 | 0.796410 | 0.769013 | 0.088647
(1 row)
```

留存场景扩展函数

- **函数说明**

留存分析是最常见的典型用户增长分析场景，用户经常需要绘制数据可视化图形，分析用户的留存情况。

时间	新增用户	1天后	2天后	3天后	4天后	5天后	6天后	7天后	14天后	30天后
2021-03-26	606	1.49%	0.99%	0.99%	0.83%	0.83%	0.5%			
2021-03-27	256	2.34%	1.95%	1.17%	0.78%	1.95%				
2021-03-28	286	2.8%	1.05%	0.7%	0.35%					
2021-03-29	719	0.56%	0.28%	0.42%						
2021-03-30	555	1.08%	1.44%							
2021-03-31	617	0.97%								

基于该场景，Hologres构造了range_retention_count和range_retention_sum两个函数用于服务该场景。range_retention_count返回值为bigint数组。`output_format='normal'` 不支持直接读取，但可以作为range_retention_sum的输入，range_retention_sum返回值为text数组，示例如下：

```
 {
  "20210306,562574,413024,343126,291178,215240,0,0,0,0,0,0,0,0,0,0",
  "20210309,502617,376019,332303,273917,208421,0,0,0,0,0,0,0,0,0,0",
   "20210305,550118,412264,357542,297458,216549,0,0,0,0,0,0,0,0,0,0",
   "20210307,560659,402810,344581,291851,214705,0,0,0,0,0,0,0,0,0,0",
   "20210308,527392,383356,346084,282764,212309,0,0,0,0,0,0,0,0,0,0"
 }
```

当`output_format= 'expand'` 时，`range_retention_count` 可单独使用，可以直接读取明细，示例如下：

```
{
  7038619894114387000,
  7038619894114560000
}
```

数组中的数字高32位代表初始日期，低32位代表留存日期，日期表达方式是距离1970-01-01的秒数，您可以进行位运算取得这两个数字。

- **函数语法**

```
-- Hologres V0.10及以上版本支持
range_retention_count(is_first,
                      is_active,
                      dt,
                      retention_interval,
                      retention_granularity)
-- Hologres V1.1及以上版本支持，如不支持请升级至最新版本
range_retention_count(is_first,
                      is_active,
                      dt,
                      retention_interval,
                      retention_granularity,
                      output_format)
```

- **参数说明**

参数	类型	说明
is_first	bool	是否符合初始行为。 ○ true：符合初始行为。 ○ false：不符合初始行为。
is_active	bool	是否符合后续留存行为。 ○ true：符合后续留存行为。 ○ false：不符合后续留存行为。
dt	date	发生行为日期。如2020-12-12
retention_interval	int[]	留存间隔，最多支持15个留存间隔。如ARRAY[1,3,5,7,15,30]
retention_granularity	text	留存粒度，支持如下三种： ○ day ○ week ○ month
output_format	text	输出格式，支持如下两种： ○ normal（默认） ○ expand 说明 ○ Hologres V1.1.38及以上版本支持此参数。 ○ expand可取得每日留存明细。

- **使用示例**
  - 如果您需要分析一段时间内用户的留存情况，且希望将浏览商品作为用户的初始行为，您可以使用如下SQL：

```
WITH tbl_detail AS (
        SELECT range_retention_count(is_first, is_active, dt, ARRAY[1, 3, 7],'day' ) AS
detail
        FROM (
            SELECT user_id, event_time::DATE AS dt
                , CASE
                    WHEN
                    event_time >= timestamp '2014-11-25 00:00:00'
                    AND event_time < timestamp '2014-11-25 00:00:00' + INTERVAL '7' day
                    and event_type = '浏览商品'  --将浏览作为初始行为
                    THEN true
                    ELSE false
                END AS is_first
                , CASE
                    WHEN event_time >= timestamp '2014-11-25 00:00:00' + INTERVAL '1' d
ay
                    AND event_time < timestamp '2014-11-25 00:00:00' + INTERVAL '7' day
+ INTERVAL '7' day   --此处为从'2014-11-25 00:00:00'开始的第7天的7日留存，INTERVAL需要加2次
                    THEN true
                    ELSE false
                END AS is_active
            FROM public.user_analysis
        ) tbl
        GROUP BY user_id
    ),
    tbl_sum AS (
        SELECT regexp_split_to_array(unnest(range_retention_sum(detail)), ',') AS s
        FROM tbl_detail
    )
SELECT s[1] AS 访问日期
        ,s[3]::numeric / s[2]::numeric AS 第1天
        ,s[4]::numeric / s[2]::numeric AS 第3天
        ,s[5]::numeric / s[2]::numeric AS 第7天
FROM tbl_sum
ORDER BY s[1];
```

显示结果如下所示：

```
  访问日期|     第1天      |     第3天      |     第7天
----------+--------------+--------------+--------------
 20141125 | 0.7962496060 | 0.7533879609 | 0.7666246454
 20141126 | 0.7904926806 | 0.7547615299 | 0.7684558476
 20141127 | 0.7711184521 | 0.7651407896 | 0.7564889098
 20141128 | 0.7799514955 | 0.7801131770 | 0.7414713015
 20141129 | 0.7946945337 | 0.7770096463 | 0.7500000000
 20141130 | 0.7944496707 | 0.7754782063 | 0.7483537158
 20141201 | 0.8017728870 | 0.7780834479 | 0.7545468439
(7 rows)
```

- 如果您需要分析从2021年2月1日开始至2021年2月7日的每日的1日留存用户、3日留存用户和7日留存用户留存在15日的付费次数，且希望将浏览商品作为用户的初一段时间内用户的留存付费情况，且将浏览作为统计留存率的初始行为，即可使用如下SQL：

```
WITH pay_table AS (
        SELECT event_date, user_id
        FROM retention_demo
        WHERE event_date BETWEEN '2021-02-01' AND '2021-02-15'
            AND event_type = '付款'
    )
SELECT tbl3.init_dt, tbl3.retention_1, tbl3.retention_3, tbl3.retention_7
FROM (
    SELECT to_timestamp(tbl2.init_ds * 86400)::date AS init_dt, COUNT(CASE
            WHEN tbl2.ds_reten - tbl2.init_ds = 1 THEN 1
            ELSE NULL
        END) AS retention_1, COUNT(CASE
            WHEN tbl2.ds_reten - tbl2.init_ds = 3 THEN 1
            ELSE NULL
        END) AS retention_3
        , COUNT(CASE
            WHEN tbl2.ds_reten - tbl2.init_ds = 7 THEN 1
            ELSE NULL
        END) AS retention_7
    FROM (
        SELECT user_id, unnest(r) >> 32 AS init_ds
            , unnest(r) & 4294967295 AS ds_reten
        FROM (
            SELECT user_id
                , range_retention_count(event_date >= '2021-02-01'     AND event_date <
'2021-02-08' AND "event_type" = '浏览',
          event_date >= date '2021-02-01' + INTERVAL '1' day AND event_date < date '202
1-02-08' + INTERVAL '7' day,
                    event_date, ARRAY[1, 3, 7], 'day', 'expand') AS r
            FROM retention_demo
            WHERE event_date BETWEEN '2021-02-01' AND '2021-02-15'
            GROUP BY user_id
        ) tbl1
    ) tbl2
        JOIN pay_table
        ON tbl2.user_id = pay_table.user_id
            AND to_timestamp(tbl2.ds_reten * 86400)::date = pay_table.event_date::date
    GROUP BY tbl2.init_ds
) tbl3;
```

显示结果如下所示。

```
  init_dt   | retention_1 | retention_3 | retention_7
------------+-------------+-------------+-------------
 2021-02-05 |           1 |           0 |           1
 2021-02-03 |           0 |           0 |           1
 2021-02-01 |           3 |           0 |           1
 2021-02-06 |           1 |           0 |           1
 2021-02-07 |           0 |           1 |           1
(5 rows)
```

6.4.5.2. 明细圈人函数

本文将为您介绍明细圈人函数bit_construct和bit_match的具体使用语法。

背景信息

在圈人场景中，通常会有通过明细表圈人的需求，例如一个用户有多条记录，不同的记录满足不同的条件，明细圈人通常需要找出满足某些条件组合的用户列表。

示例明细表如下，需要在明细表中找出有过 `click购物车` 和 `view收藏页` 行为的user列表（此处满足条件的是user A）。

user	action	page
A	click	购物车
B	click	首页
A	view	收藏页
B	click	购物车
A	click	收藏页

传统的查询方法是采用多轮条件过滤并使用JOIN语句来筛选出符合条件的结果，这种方法需要写复杂的SQL，同时多个JOIN语句也会消耗大量的资源。从Hologres V0.10版本开始，针对明细表圈人场景提供bit_construct和bit_match函数，只需要通过一轮过滤筛选和函数计算即可得到结果人群数据，降低明细圈人场景开发复杂度，快速得到结果。

使用限制

在Hologres中使用明细圈人函数，具体限制如下：

- 该函数仅Hologres V0.10及以上版本支持，请在Hologres管控台查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- 明细圈人函数在使用之前，需要执行以下语句开启extension。extension是DB级别的函数，一个DB只需执行一次即可，新建DB需要重新执行。

```
--创建extension
CREATE EXTENSION flow_analysis;
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP EXTENSION flow_analysis;
> ```

bit_construct

Hologres在V0.10及以上版本提供了bit_construct函数用于明细表圈人场景。

- **函数说明**

  bit_construct通过设置筛选条件，返回一个最多32位integer类型的bitmap。

- **语法示例**

```
bit_construct(
  a ,
  b ,
  ....,
  a6
)
```

- **参数说明**
  - 参数a和b等，代表筛选条件，为bool类型，最多支持32个条件，取值范围为a~z,a1~a6。
  - 函数返回值类型为int。

bit_match

Hologres在V0.10及以上版本提供了bit_match函数用于明细表圈人场景。

- **函数说明**

  bit_match用于对bit_construct筛选的结果进行进一步计算

- **语法示例**

```
bit_match('expression', bitmask)
```

- **参数说明**

参数	说明	示例
expression	bit_construct函数中的条件表达式，支持 `&` （and）、 `|` （or）、 `!` （not）和 `^` （XOR）。	a&b
bitmask	bit_construct构造的bitmap名。	无

使用示例

使用明细圈人函数，完整语法示例如下：

1. 开启extension。

   Superuser在DB中执行如下语句开启extension：

```
CREATE EXTENSION flow_analysis;
```

2. 准备表和数据。

   如下为一张记录用户购物行为的明细表，该表记录了用户的uid以及购物路径的行为，同时对该表插入部分数据。

```
create table ods_app_dwd(
event_time timestamptz,
uid bigint,
action text,
page text,
product_code text,
from_days int
);
insert into ods_app_dwd values('2021-04-03 10:01:30', 274649163, 'click', '购物车', 'MDS
', 1);
insert into ods_app_dwd values('2021-04-03 10:04:30', 274649163, 'view', '收藏页', 'MDS'
, 4);
insert into ods_app_dwd values('2021-04-03 10:06:30', 274649165, 'click', '购物车', 'MMS
', 8);
insert into ods_app_dwd values('2021-04-03 10:09:30', 274649165, 'view', '购物车', 'MDS'
, 10);
```

3. 数据查询。

   查询出满足“某个时间范围内将商品既加入购物车又添加至收藏页”条件的精准用户。

   您可以使用如下两种方式进行数据查询：

   - 使用WHERE语句查询。

     > **说明** 在查询语句中WHERE条件筛选后的数据越少，查询性能越好。

     ```
     WITH tbl as (
     SELECT uid, bit_or(bit_construct(
       a := (action='click' and page='购物车'),
       b := (action='view' and page='收藏页'))) as uid_mask
       FROM ods_app_dwd
     WHERE event_time > '2021-04-03 10:00:00' AND event_time < '2021-04-04 10:00:00'
     GROUP BY uid )
     SELECT uid from tbl where bit_match('a&b', uid_mask);
     ```

     其中关于SQL语句的释义如下：

     - bit_construct：筛选出加入购物车的用户为条件a，筛选出添加至收藏夹的用户为条件b。
     - bit_or：将筛选出的条件a和b取或，在该SQL语句示例中，同一个用户只要有一行符合条件，即认为符合条件。
     - bit_match：a&b取满足条件a和b的交集。

   - 使用HAVING子句调用查询。

```
SELECT uid FROM (
    SELECT uid, bit_or(bit_construct(
      a := (action='click' AND page='购物车'),
      b := (action='view' AND page='收藏页'))) as uid_mask
    FROM ods_app_dwd
    WHERE event_time > '2021-04-03 10:00:00' AND event_time < '2021-04-04 10:00:00'
    GROUP BY uid
    HAVING bit_match('a&b', bit_or(bit_construct(
      a := (action='click' and page='购物车'),
      b := (action='view' and page='收藏页'))))
) t
```

6.4.5.3. Roaring Bitmap函数

本文将为您介绍在Hologres中Roaring Bitmap函数的具体参数和使用。

背景信息

Roaring Bitmap是一种高效的Bitmap压缩算法，目前已被广泛应用在各种语言和各种大数据平台。适合计算超高基维的，常用于去重、标签筛选、时间序列等计算中。

Roaring Bitmap算法是将32位的INT类型数据划分为2^{16}个数据块（Chunk），每一个数据块对应整数的高16位，并使用一个容器（Container）来存放一个数值的低16位。Roaring Bitmap将这些容器保存在一个动态数组中，作为一级索引。容器使用两种不同的结构：数组容器（Array Container）和位图容器（Bitmap Container）。数组容器存放稀疏的数据，位图容器存放稠密的数据。如果一个容器里面的整数数量小于4096，就用数组容器来存储值。若大于4096，就用位图容器来存储值。

采用这种存储结构，Roaring Bitmap可以快速检索一个特定的值。在做位图计算（AND、OR、XOR）时，Roaring Bitmap提供了相应的算法来高效地实现在两种容器之间的运算。使得Roaring Bitmap无论在存储和计算性能上都表现优秀。

使用限制

在Hologres中使用Roaring Bitmap函数，具体限制如下：

- 该函数仅Hologres V0.10及以上版本支持，请在Hologres管控台查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- Roaring Bitmap函数在使用之前，需要执行以下语句开启extension才可以调用。extension是DB级别的函数，一个DB只需执行一次即可，新建DB需要重新执行。

  ```
  --创建extension
  CREATE EXTENSION roaringbitmap;
  ```

  > **说明** 如需卸载extension请执行如下命令。
  >
  > ```
  > DROP EXTENSION roaringbitmap;
  > ```

- 不支持将Roaring Bitmap设置为Bitmap或者Dictionary。

Bitmap函数列表

Bitmap函数主要包括的函数及说明如所示：

函数名	输入	输出	描述	示例
rb_build	integer[]	roaringbitmap	通过数组创建一个Bitmap。	`rb_build('{1,2,3,4,5}')`
rb_and	roaringbitmap，roaringbitmap	roaringbitmap	And计算。	`rb_and(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_and_null2empty	roaringbitmap，roaringbitmap	roaringbitmap	And计算。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_and_null2empty(rb_build(null),rb_build('{3,4,5}'))`
rb_or	roaringbitmap，roaringbitmap	roaringbitmap	Or计算。	`rb_or(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_or_null2empty	roaringbitmap，roaringbitmap	roaringbitmap	Or计算。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_or_null2empty(rb_build(null),rb_build('{3,4,5}'))`
rb_xor	roaringbitmap，roaringbitmap	roaringbitmap	Xor计算。	`rb_xor(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot	roaringbitmap，roaringbitmap	roaringbitmap	AndNot计算。	`rb_andnot(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot_null2empty	roaringbitmap，roaringbitmap	roaringbitmap	AndNot计算。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_andnot_null2empty(rb_build(null),rb_build('{3,4,5}'))`

函数名	输入	输出	描述	示例
rb_cardinality	roaringbitmap	integer	统计基数。	`rb_cardinality(rb_build('{1,2,3,4,5}'))`
rb_and_cardinality	roaringbitmap，roaringbitmap	integer	And计算并返回基数。	`rb_and_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_and_null2empty_cardinality	roaringbitmap，roaringbitmap	integer	And计算并返回基数。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_and_null2empty_cardinality(rb_build(null),rb_build('{3,4,5}'))`
rb_or_cardinality	roaringbitmap，roaringbitmap	integer	Or计算并返回基数。	`rb_or_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_or_null2empty_cardinality	roaringbitmap，roaringbitmap	integer	Or计算并返回基数。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_or_null2empty_cardinality(rb_build(null),rb_build('{3,4,5}'))`
rb_xor_cardinality	roaringbitmap，roaringbitmap	integer	Xor计算并返回基数。	`rb_xor_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot_cardinality	roaringbitmap，roaringbitmap	integer	AndNot计算并返回基数。	`rb_andnot_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`

函数名	输入	输出	描述	示例
rb_andnot_null2empty_cardinality	roaringbitmap，roaringbitmap	integer	AndNot计算并返回基数。当输入为Null时，Bitmap会按空（{}）来处理。 **说明** Hologres V1.1.42及以上版本支持。	`rb_andnot_null2empty_cardinality(rb_build(null),rb_build('{3,4,5}'))`
rb_is_empty	roaringbitmap	boolean	判断是否为空的Bitmap。	`rb_is_empty(rb_build('{1,2,3,4,5}'))`
rb_equals	roaringbitmap，roaringbitmap	boolean	判断两个Bitmap是否相等。	`rb_equals(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_intersect	roaringbitmap，roaringbitmap	boolean	判断两个Bitmap是否相交。	`rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_contains	roaringbitmap,roaringbitmap	boolean	判断第一个Bitmap是否包含第二个Bitmap	`rb_contains(rb_build('{1,2,3}'),rb_build('{3}'))`
rb_remove	roaringbitmap,integer	roaringbitmap	从Bitmap移除特定的Offset。	`rb_remove(rb_build('{1,2,3}'),3)`
rb_flip	roaringbitmap,integer,integer	roaringbitmap	翻转Bitmap中特定的Offset段。	`rb_flip(rb_build('{1,2,3}'),2,3)`
rb_minimum	roaringbitmap	integer	返回Bitmap中最小的Offset，如果Bitmap为空则返回-1。	`rb_minimum(rb_build('{1,2,3}'))`
rb_maximum	roaringbitmap	integer	返回Bitmap中最大的Offset，如果Bitmap为空则返回0。	`rb_maximum(rb_build('{1,2,3}'))`
rb_rank	roaringbitmap,integer	integer	返回Bitmap中小于等于指定Offset的基数。	`rb_rank(rb_build('{1,2,3}'),3)`

函数名	输入	输出	描述	示例
rb_iterate	roaringbitmap	set of integer	返回Offset List。	`rb_iterate(rb_build('{1,2,3}'))`
rb_to_array	roaringbitmap	integer[]	返回Bitmap对应整型数组。	`rb_to_array(rb_build('{1,2,3}'))`
rb_to_array_string	roaringbitmap, text	text	返回Bitmap对应整型数组拼接的字符串。	`rb_to_array_string(rb_build('{1,2,3}'),',')`

Bitmap聚合函数列表

Bitmap聚合函数主要包括的函数及说明如所示：

函数名	输入数据类型	输出数据类型	描述	示例
rb_build_agg	integer	roaringbitmap	将Offset聚合成bitmap。	`rb_build_agg(1)`
rb_or_agg	roaringbitmap	roaringbitmap	Or聚合计算。	`rb_or_agg(rb_build('{1,2,3}'))`
rb_and_agg	roaringbitmap	roaringbitmap	And聚合计算。	`rb_and_agg(rb_build('{1,2,3}'))`
rb_xor_agg	roaringbitmap	roaringbitmap	Xor聚合计算。	`rb_xor_agg(rb_build('{1,2,3}'))`
rb_or_cardinality_agg	roaringbitmap	integer	Or聚合计算并返回其基数。	`rb_or_cardinality_agg(rb_build('{1,2,3}'))`
rb_and_cardinality_agg	roaringbitmap	integer	And聚合计算并返回其基数。	`rb_and_cardinality_agg(rb_build('{1,2,3}'))`
rb_xor_cardinality_agg	roaringbitmap	integer	Xor聚合计算并返回其基数。	`rb_xor_cardinality_agg(rb_build('{1,2,3}'))`

使用示例

如下内容将为您介绍Roaring Bitmap函数完整的使用示例。

1. 加载RoaringBitmap函数插件。

```
CREATE EXTENSION roaringbitmap;
```

2. 创建带有RoaringBitmap数据类型的表。

```
--创建名称为t1的表
CREATE TABLE public.t1 (id integer, bitmap roaringbitmap);
```

3. 使用rb_build函数插入RoaringBitmap的数据。

```
--数组位置对应的BIT值为1
INSERT INTO public.t1 SELECT 1,RB_BUILD(ARRAY[1,2,3,4,5,6,7,8,9,200]);
--将输入的多条记录的值对应位置的BIT值设置为1，最后聚合为一个RoaringBitmap
INSERT INTO public.t1 SELECT 2,RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

4. 进行Bitmap计算（OR、AND、XOR、ANDNOT）。

```
SELECT  RB_OR(a.bitmap,b.bitmap)
FROM    (
            SELECT  bitmap
            FROM    public.t1
            WHERE   id = 1
        ) AS a
        ,(
            SELECT  bitmap
            FROM    public.t1
            WHERE   id = 2
        ) AS b
;
```

5. 进行Bitmap聚合计算（OR、AND、XOR、BUILD），并生成新的RoaringBitmap类型。

```
SELECT RB_OR_AGG(bitmap) FROM public.t1;
SELECT RB_AND_AGG(bitmap) FROM public.t1;
SELECT RB_XOR_AGG(bitmap) FROM public.t1;
SELECT RB_BUILD_AGG(id) FROM public.t1;
```

6. 统计基数（Cardinality），即统计RoaringBitmap中包含多少个位置为1的BIT位。

```
SELECT RB_CARDINALITY(bitmap) FROM public.t1;
```

7. 从RoaringBitmap中返回位置为1的BIT下标（即位置值）。

```
SELECT RB_ITERATE(bitmap) FROM public.t1 WHERE id = 1;
```

8. 将RoaringBitmap转换为数组结构。

```
SELECT RB_TO_ARRAY(bitmap) FROM public.t1 WHERE id = 1;
```

6.4.6. 聚合函数

6.4.6.1. APPROX_COUNT_DISTINCT

APPROX_COUNT_DISTINCT是聚合函数。本文为您介绍在交互式分析Hologres中APPROX_COUNT_DISTINCT函数的用法。

语法

`APPROX_COUNT_DISTINCT` 函数用于计算某一列去重后的行数，结果只能返回一个值，并且该值为近似值。

```
APPROX_COUNT_DISTINCT ( <column> )
```

参数说明如下表所示。

参数	描述
column	需要近似计算去重后行数的列。

`APPROX_COUNT_DISTINCT` 采用HyperLogLog基数估计的方式进行非精确的COUNT DISTINCT计算。非精确的COUNT DISTINCT计算能提升查询性能，尤其是对于column的离散值比较大的情况，误差率平均可以控制在 `0.1%-1%` 以内。该函数适用于对性能敏感并且可以接受误差的场景。

同时，您也可以通过 `COUNT DISTINCT ( column )` 的方式进行精确的COUNT DISTINCT计算，使用时资源开销会比较大。

调整误差率

通过使用以下参数调整误差率。

```
set hg_experimental_approx_count_distinct_precision = 20;
```

- 支持取值范围为[12,20]，默认值为17。
- 精度参数含义为HyperLogLog算法的分桶bit位个数，参数越大，代表分桶越多，理论精度越高。
- 精度参数取值越高，计算时间和内存开销也会相应增大，但都远远小于精确的 `COUNT DISTINCT ( column )` 语句带来的开销，因此，推荐选用APPROX_COUNT_DISTINCT替换 `COUNT DISTINCT ( column )` 。
- 当精度参数设置为17以上时，Hologres采用HyperLogLog++算法，会对返回值做误差修正，以进一步降低误差、稳定误差。例如hg_experimental_approx_count_distinct_precision取值为20时，多数情况下，可以降低到0.01-0.2%不等的误差率。

示例

计算O_CUSTKEY列去重后行数的近似值，示例语句如下。

```
SELECT APPROX_COUNT_DISTINCT ( O_CUSTKEY ) FROM ORDERS;
--全局设置精度20，计算O_CUSTKEY列去重后行数的近似值
ALTER DATABASE dbname SET hg_experimental_approx_count_distinct_precision = 20;
SELECT APPROX_COUNT_DISTINCT ( O_CUSTKEY ) FROM ORDERS;
--在Session级别设置精度20
SET hg_experimental_approx_count_distinct_precision = 20;
SELECT APPROX_COUNT_DISTINCT ( O_CUSTKEY ) FROM ORDERS;
```

6.4.7. 账号与授权函数

6.4.7.1. USER_DISPLAY_NAME

USER_DISPLAY_NAME用于转换Hologres账号为邮箱形式或RAM形式的阿里云账号。本文为您介绍在交互式分析Hologres中USER_DISPLAY_NAME的用法。

命令语法

USER_DISPLAY_NAME的命令语法如下所示：

```
SELECT user_display_name ( user_name )
```

参数说明

参数说明如下表所示。

参数	描述
user_name	阿里云账号ID。通常为数字形式，例如 `13532241323xxx` 或RAM用户 `p4_23402030200xxx` 。

返回值为邮箱形式或RAM形式的阿里云账号，例如 `xx@aliyun.com` ， `RAM$mainaccount:subuser` 或 `RAM$public` 。

使用示例

转换Hologres账号为邮箱形式或RAM形式的阿里云账号的示例语句如下。

```
SELECT user_display_name ('13532241323xxx');
SELECT rolname, user_display_name(rolname) FROM pg_roles;
```

6.4.7.2. HG_USER_DISPLAY_NAME_TO ID

HG_USER_DISPLAY_NAME_TO ID用于转换阿里云账号ID为Hologres账号。本文为您介绍在交互式分析Hologres中HG_USER_DISPLAY_NAME_TO ID的用法。

语法

```
SELECT hg_user_diplay_name_to_id (aliyun_id);
```

参数说明如下表所示。

参数	描述
aliyun_id	阿里云账号。通常为邮箱形式或RAM形式，例如 `xx@aliyun.com` ， `RAM$mainaccount:subuser` 或 `RAM$public` 。

返回值*user_name*为阿里云账号ID，通常为数字形式，例如 `13532241323xxx` 或 `p4_23402030200xxx` 。

示例

转换阿里云账号ID为Hologres账号的示例语句如下。

```
SELECT hg_user_display_name_to_id ('RAM$main:subuser');
```

6.4.7.3. APPLY_PRIVILEGES

APPLY_PRIVILEGES函数用于在专家权限模型下，将源表给用户授予的权限复制到目标表，目标表保持与源表一致的权限。本文为您介绍在Hologres中APPLY_PRIVILEGES函数的用法。

使用场景

专家权限模型下，当使用 `CREATE TABLE LIKE` 、 `INSERT OVERWRITE` 、自动分区等创建新表时，新表的权限还需要重新再授予给用户，通过 `APPLY_PRIVILEGES` 函数可以复制源表的所有用户权限至新表，简化授权操作。

使用限制

- 仅Hologres V1.1.48及以上版本支持 `APPLY_PRIVILEGES` 函数，如果您的实例是V1.1.48以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 需要数据库开启专家权限模型，如果您开启的是SPM或者SLPM请转换为专家权限模型，详情请参见权限模型转换。
- 调用函数需要具有目标表的GRANT、REVOKE权限。
- `APPLY_PRIVILEGES` 的目标表是分区表时，只对父表复制权限，不影响子表访问控制权限。

命令语法

```
-- 将源表的权限复制给目标表，并且不回收目标表的权限，即不删除目标表之前具有的权限规则
CALL apply_privileges('<old_table>','<new_table>', false);
-- 将源表的权限复制给目标表，并且回收目标表的权限，即删除目标表之前具有的权限规则
CALL apply_privileges('<old_table>','<new_table>');
```

参数说明如下表所示。

参数	说明
old_table	源表名称，即被复制权限的表名称。
new_table	目标表名称，即获得权限的表名称。

使用示例

使用 `APPLY_PRIVILEGES` 函数进行复制权限操作的示例如下。

```
-- 将test_table1 ACL授权信息、owner信息应用到test_table2上， false代表不对test_table2之前具有的权限规则进行删除
CALL apply_privileges('test_table1','test_table2', false);
-- 将test_table1 ACL授权信息、owner信息应用到test_table2上，对test_table2之前具有的权限规则进行删除，保持两表的权限完全一致
CALL apply_privileges('test_table1','test_table2');
```

6.4.8. Hive兼容函数

6.4.8.1. GET_JSON_OBJECT

GET_JSON_OBJECT用于解析JSON对象。本文为您介绍在交互式分析Hologres中GET_JSON_OBJECT的用法。

使用限制

- 一个数据库只能在一个Schema下创建一次extension。例如在某数据库的public Schema下创建了extension，则此数据库下其余的Schema不能再创建extension。
- 可以将extension直接创建在pg_catalog系统Schema下，就会默认所有数据库的所有Schema都能使用该功能。
- 需要Superuser创建extension。

命令语法

GET_JSON_OBJECT的命令语法如下，在调用之前需要先创建extension。

```
--创建extension
CREATE extension if not exists $(extension_name) schema $(schema_name);
SELECT get_json_object ( json_string, path );
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP extension hive_compatible;
> ```

参数说明

GET_JSON_OBJECT的参数说明如下表所示：

参数	描述
json_string	JSON对象变量，TEXT类型。格式为合法JSON格式字符串。
path	JSON内层对象访问变量。使用 `$` 表示JSON变量标识，通过 `.` 或 `[]` 读取JSON内层对象或数组。 如果您输入的JSON字符串无效，则系统返回*NULL*。

使用示例

```
create extension if not exists hive_compatible schema pg_catalog;
begin;
create table hive_json_example(
    col_json text
);
commit;
insert into hive_json_example values('{"store":{"fruit":[{"weight":8,"type":"apple"}, {"wei
ght":9,"type":"pear"}],"bicycle":{"price":19.95,"color":"red"}},"email":"amy@only_for_json_
udf_test.net","owner":"amy"}');
select get_json_object(col_json, '$.owner') from hive_json_example;
--Result: amy
select get_json_object(col_json, '$.store.bicycle.price') from hive_json_example;
--Result: 19.95
select get_json_object(col_json, '$.store.fruit[0]') from hive_json_example;
--Result: {"weight":8, "type":"apple"}
select get_json_object(col_json, '$.no_key') from hive_json_example;
--Result: NULL
```

常见报错

- 报错：`ERROR: function get_json_object (text, unknown) does not exist` 。
  - 可能原因一

    在SLPM模式下RAM用户没有创建extension所在Schema的查询权限（例如extension指定创建在名称为public的Schema下，RAM用户没有public的查询权限）。
  - 解决方法一
    - 授予RAM用户Schema的查询权限。
    - 使用如下命令重新创建extension在pg_catalog下，所有账号都可查询。

      ```
      drop extension hive_compatible;
      create extension hive_compatible schema pg_catalog;
      ```
  - 可能原因二

    GET_JSON_OBJECT的第一个参数不是TEXT类型。
  - 解决方法二

    将第一个参数转换为TEXT类型。
- 报错：`ERROR: get_json_object for fe, should not be evaluated` 。
  - 可能原因一

    GET_JSON_OBJECT的第一个参数是常量。
  - 解决方法一

    第一个参数使用表的列。
  - 可能原因二

    GET_JSON_OBJECT的第一个参数含有为NULL的值。
  - 解决方法二

    将第一个参数为NULL的值删除。

6.4.9. MaxCompute兼容函数

6.4.9.1. MAX_PT

MAX_PT用于查询交互式分析Hologres表最大的分区表名称。本文为您介绍在Hologres中MaxCompute兼容函数MAX_PT的用法。

命令语法

```
SELECT MAX_PT('<table_name>');
```

参数说明如下表所示。

参数	描述
table_name	Hologres的表名称。

使用示例

```
SELECT MAX_PT('holo_table');
```

6.4.10. 工具函数

6.4.10.1. HG_VERSION

HG_VERSION用于获取当前Hologres的版本。本文将为您介绍在Hologres中HG_VERSION函数的具体用法。

使用限制

仅Hologres V0.10 及以上版本支持HG_VERSION函数。

命令语法

使用HG_VERSION函数获取当前Hologres的版本的语法如下：

```
select hg_version()
```

输出结果会显示当前Hologres实例版本、兼容的Psql版本，当前操作系统信息和编译器等信息。

使用示例

您可以在当前连接输入如下命令：

```
select hg_version()
```

输出结果如下所示：

```
                                                          hg_version
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------
 Hologres 0.9.29 (tag: 0.9.29 build: Release,Skylake,clang), compatible with PostgreSQL 11.
3 (Release-build@1b725dxxxxx on origin/ralease-0.9.x) on x86_64-pcplinux-gnu, compiled by cl
ang version 8.0.1 (Alibaba 8.0.1-14.alios7),64-bit
(1 row)
```

其中，输出结果显示当前Hologres实例版本为0.9.29、兼容PostgreSQL的11.3版本，当前操作系统是Linux下64位操作系统。

6.4.10.2. HG_SHARD_ID_FOR_DISTRIBUTION_KEY

HG_SHARD_ID_FOR_DISTRIBUTION_KEY用于获取指定键或组合键的数据片ID（Shard ID）。本文为您介绍在交互式分析Hologres中HG_SHARD_ID_FOR_DISTRIBUTION_KEY的用法。

命令语法

```
SELECT HG_SHARD_ID_FOR_DISTRIBUTION_KEY ( 'schemaname.tablename' , columnname [...] ] )
FROM schemaname.tablename
```

参数说明

参数说明如下表所示。

参数	描述
schemaname	对应schema名称。
tablename	需要获取Shard ID的表名称。
columnname	单column或组合column的表名，作为计算所落到的数据分片ID（Shard ID）上。

使用示例

您可以使用如下示例步骤获取指定键或组合键的数据片ID（Shard ID）。

1. 安装插件。您可以执行以下语句安装插件。该命令是DB级别的，在一个DB内执行一次即可。切换到新的DB需要再次执行。

   ```
   create extension hg_toolkit;
   ```

2. 获取lineitem表中l_shipmode列的Shard ID。

   ```
   SELECT  HG_SHARD_ID_FOR_DISTRIBUTION_KEY ( 'public.lineitem', l_shipmode)
   FROM public.lineitem ;
   ```

3. 获取执行结果。

```
 hg_shard_id_for_distribution_key
---------------------------------
                               24
                               24
                               24
                               24
                               31
                               24
                               12
                               25
                               25
                               25
(10 rows)
```

6.4.10.3. HG_UPDATE_DATABASE_PROPERTY

HG_UPDATE_DATABASE_PROPERTY用于更新数据库的属性。本文为您介绍在交互式分析Hologres中HG_UPDATE_DATABASE_PROPERTY的用法。

使用限制

仅实例的Superuser和数据库的Owner可以调用 `HG_UPDATE_DATABASE_PROPERTY` 函数。

介绍

HG_UPDATE_DATABASE_PROPERTY用于更新数据库的两个属性，如下表所示。

参数	说明
default_table_group	设置某个Table Group为默认Table Group。 **说明** 仅Hologres V0.10 及以上版本支持使用此参数。
shard_count	设置默认Table Group的Shard Count（不建议使用）。

语法

```
CALL HG_UPDATE_DATABASE_PROPERTY ( 'property', 'value' );
```

参数说明如下表所示。

参数	描述
property	属性名称
value	属性值

示例

更改默认Table Group为名称为 `TG120` 的Table Group，语句如下。

```
CALL HG_UPDATE_DATABASE_PROPERTY ( 'default_table_group', 'TG120' );
```

6.4.10.4. SET_TABLE_PROPERTY

本文为您介绍在交互式分析Hologres中SET_TABLE_PROPERTY的用法。

函数概述

SET_TABLE_PROPERTY用于设置表的属性，包括索引、分布列、行存储、列存储以及生命周期等属性。如果您需要对创建的表进行修改、更新和删除，具体操作请参见ALTER TABLE或DROP TABLE。

命令格式

```
CALL SET_TABLE_PROPERTY ( table_name, property, value )
WHERE PROPERTY IN
    orientation
  clustering_key
  segment_key
  bitmap_columns
  dictionary_encoding_columns
  time_to_live_in_seconds
  distribution_key
```

参数说明

SET_TABLE_PROPERTY参数说明如下表所示。

参数	描述
table_name	表名称。您也可以使用Schema限定表名称。 表名称只能是小写英文字母 `a~z` 、大写英文字母 `A~Z` 、数字以及下划线 `(_)` 的组合，并且以字母开头。 如果表名称有特殊字符，则需要使用双引号 `""` 转义。由于系统对大小写不敏感，大写字母 `A~Z` 会被认为是小写字母 `a~z` 。
property	属性名称。
orientation	用于指定数据库的表在Hologres中是列存储还是行存储。 该参数仅支持和CREATE TABLE在同一事务中执行。
clustering_key	用于在指定的列建立聚簇索引。 该参数仅支持和CREATE TABLE在同一事务中执行。

参数	描述
segment_key	用于指定某些列作为分段键，例如，指定时间列作为分段键。 当查询条件包含分段列时，查询可以通过分段键快速找到相应数据对应的存储位置。 该参数仅支持和CREATE TABLE在同一事务中执行。
bitmap_columns	用于在指定列构建比特编码，快速过滤分段内部的数据。您可以单独使用该参数。
dictionary_encoding_columns	用于为指定列的值构建字典映射。 字典编码可以转换字符串的比较为数字的比较，加速Group By、Filter等查询。您可以单独使用该参数。
distribution_key	用于指定数据库中表的分布策略。 该参数仅支持和CREATE TABLE在同一事务中执行。
time_to_live_in_seconds	表数据的生命周期，单位为秒。 取值为非负数、整数或浮点数。您可以单独使用该参数。
value	属性值。如果该参数包含列名，并且列名包含大写字母，则需要添加双引号 `""` 。

使用示例

```
BEGIN;
CREATE TABLE ORDERS (
  O_ORDERKEY              INTEGER NOT NULL,
  O_CUSTKEY        INTEGER NOT NULL,
  O_ORDERSTATUS    TEXT NOT NULL,
  O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
  O_ORDERDATE      DATE NOT NULL,
  O_ORDERPRIORITY  TEXT NOT NULL,
  O_CLERK          TEXT NOT NULL,
  O_SHIPPRIORITY   INTEGER NOT NULL,
  O_COMMENT        TEXT NOT NULL);
CALL SET_TABLE_PROPERTY ('ORDERS', 'clustering_key', 'O_ORDERKEY:asc,O_CUSTKEY:asc');
CALL SET_TABLE_PROPERTY ('ORDERS', 'segment_key', 'O_ORDERDATE');
CALL SET_TABLE_PROPERTY ('ORDERS', 'bitmap_columns', 'O_ORDERSTATUS,O_ORDERPRIORITY,O_CLERK
,O_SHIPPRIORITY');
CALL SET_TABLE_PROPERTY ('ORDERS', 'dictionary_encoding_columns', 'O_ORDERSTATUS,O_ORDERPRI
ORITY,O_CLERK,O_SHIPPRIORITY');
CALL SET_TABLE_PROPERTY ('ORDERS', 'time_to_live_in_seconds', '172800');
COMMIT;
```

6.5. DDL

6.5.1. DATABASE

6.5.1.1. CREATE DATABASE

CREATE DATABASE语句用于创建数据库。本文为您介绍CREATE DATABASE的用法。

语法

```
CREATE DATABASE db_name
    [ [ WITH ] [ OWNER [=] user_name ]
;
```

参数说明如下表所示。

参数	描述
db_name	创建的数据库名称，遵循SQL标识符的一般规则。
user_name	被授权为数据库管理员的账号名称。 执行该语句的账号默认成为新数据库的Owner，拥有Superuser权限，该账号也可以授予其他用户Superuser权限。 数据库的Owner拥有删除该数据库的权限。删除数据库时，系统会同时删除数据库中的表或数据等对象。

> **说明** 开通Hologres实例后，系统自动创建postgres数据库。该数据库分配到的资源较少，仅用于管理，开发实际业务建议您新建数据库。
>
> Superuser可以为其他用户创建数据库，并授权该用户为新数据库的Owner，方便用户自行管理和配置该数据库。

示例

新建数据库的示例语句如下。

```
CREATE DATABASE testdb;
```

6.5.1.2. ALTER DATABASE

ALTER DATABASE语句用于修改数据库。本文为您介绍ALTER DATABASE的用法。

语法

```
ALTER DATABASE <dbname> SET configuration_parameter { TO | = } { value | DEFAULT }
ALTER DATABASE <dbname> SET configuration_parameter FROM CURRENT
ALTER DATABASE <dbname> RESET configuration_parameter
ALTER DATABASE <dbname> RESET ALL
```

参数说明如下表所示。

参数	描述
configuration_parameter	Hologres的配置参数。
value	配置参数的取值。
DEFAULT	设置配置参数为缺省值。
RESET	重置指定配置参数。
RESET ALL	重置数据库的所有配置参数。
SET FROM CURRENT	设置配置参数为当前Session的配置值。

示例

设置时区为晚格林威治标准时间（北京时区）8个小时的时区，SQL语句如下。

```
ALTER DATABASE postgres SET timezone='GMT-8:00';
```

6.5.1.3. DROP DATABASE

DROP DATABASE语句用于删除数据库。本文为您介绍DROP DATABASE的用法。

使用限制

- 只有数据库的Superuser，或被Superuser授权为数据库Owner的用户，才可以删除该数据库。
- 删除数据库时会同时删除数据库中的所有对象。
- 您不能再连接使用已删除的数据库。

语法

```
DROP DATABASE [ IF EXISTS ] db_name;
```

示例

删除数据库，示例语句如下。

```
DROP DATABASE mydb;
```

6.5.2. SCHEMA

6.5.2.1. CREATE SCHEMA

Hologres支持跨Schema创建表。本文为您介绍如何在Hologres中创建Schema，以及跨Schema创建表。

背景信息

Hologres兼容PostgreSQL，支持的Schema功能与PostgreSQL相同。

Hologres新增Schema功能后，表的存储结构由 `database.table` 变为 `database.schema.table` 。

Hologres当前版本主要支持创建Schema、重命名Schema以及在Schema中创建表等功能。

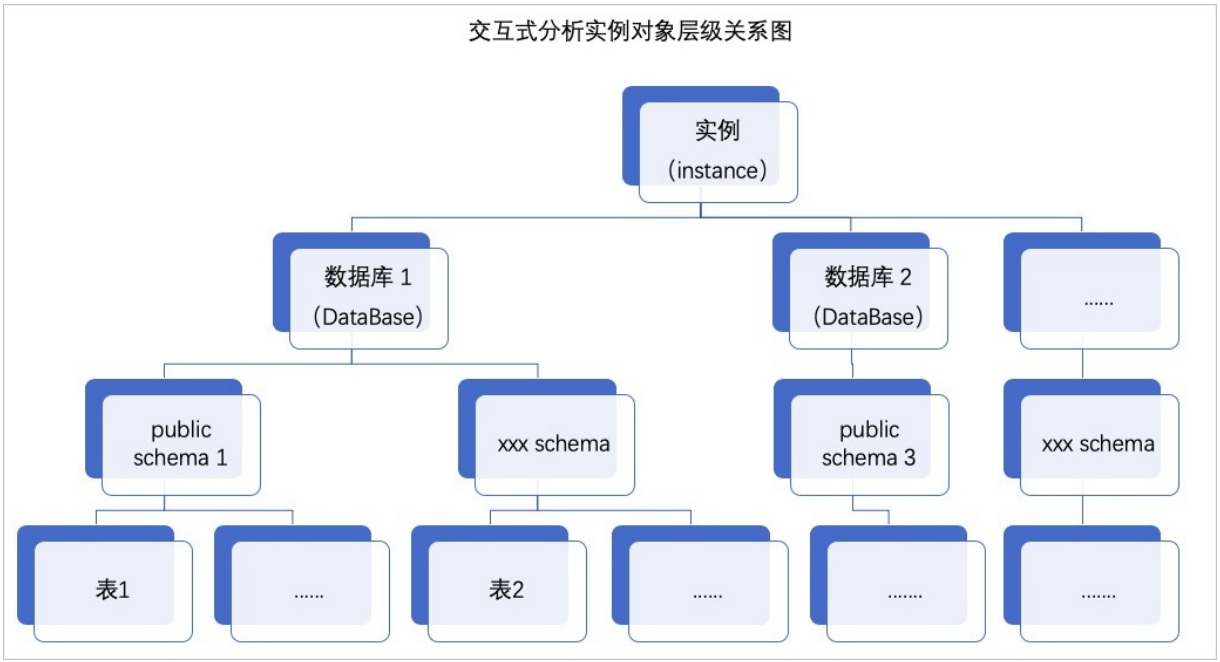
在Hologres中，每张表归属于一个Schema，一个数据库可以包含多个Schema，方便您管理。同时，多个用户使用同一个数据库时不会互相干扰。

不同的Schema可以包含相同的表名称或数据类型。

创建数据库时系统默认创建一个**public** Schema。如果您不创建其它Schema，则在该数据库创建的所有表默认存储在**public** Schema中。您可以执行以下语句查看当前Schema。

```
select current_schema(); //查看当前Schema。
\d tablename; //查看目标表所属的Schema。该语句只适用于终端。
```

Hologres实例对象的层级关系如下图所示。



操作步骤

1. 创建Schema。

   在数据库中创建Schema的示例SQL语句如下。

   ```
   create schema schemaname; //创建Schema。
   set search_path to schemaname; //切换至目标Schema。
   create table blink_demo (id text); //在目标Schema中创建表。
   select current_schema(); //查看当前Schema。
   ```

2. 跨Schema创建表。

   您可以使用 `schema.table` 语句，添加Schema名称至表名称之前，即可跨Schema创建表。具体如下：

   - 在目标Schema中为**public** Schema创建表。示例SQL语句如下。

```
create table public.mytest (
  name text,
  id int);
```

- 在public Schema中为目标Schema创建表。示例SQL语句如下。

```
set search_path to public;
create table my_schema.mytest (
  name text,
  id int,
  age int
);
```

6.5.2.2. ALTER SCHEMA

ALTER SCHEMA语句用于修改数据库中的模式。本文为您介绍ALTER SCHEMA的用法及使用限制。

使用限制

- 交互式分析Hologres仅支持修改模式的名称。
- 使用 `ALTER SCHEMA RENAME` 修改模式名称时，系统将转移原Schema中的所有表至重命名的Schema中，同时会保留原Schema。

语法

修改模式名称的SQL语句如下。

```
ALTER SCHEMA oldschema RENAME TO newschema;
```

示例

```
ALTER SCHEMA my_schema RENAME TO new_schema;
SET search_path TO my_schema;  //进入原Schema。
SET search_path TO new_schema; //进入重命名的Schema。
```

6.5.2.3. DROP SCHEMA

DROP SCHEMA语句用于删除数据库中的模式。本文为您介绍DROP SCHEMA的用法及使用限制。

使用限制

您不能删除系统默认创建的命名为public的Schema。

语法

```
DROP SCHEMA [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明如下表所示。

参数	描述
CASCADE	系统自动删除目标Schema中的所有对象，例如表、函数或数据等。
RESTRICT	如果Schema中包含任意对象，则系统默认不删除该Schema。

示例

```
DROP SCHEMA mystuff CASCADE;
```

6.5.3. TABLE

6.5.3.1. CREATE TABLE

CREATE TABLE语句用于创建表。本文为您介绍在交互式分析Hologres中CREATE TABLE的用法。

使用限制

- 支持将多个字段设置为primary key（即复合主键）。被设置为primary key的字段是唯一且非空的列或者列组合，同时只能在一个语句里设置多列为表的primary key。目前primary key不支持Float、Double、Numeric、Array、Json、Date及其他复杂数据类型，不支持修改主键，如需修改主键请重新建表。如下示例指导您将id和ds设置为表的primary key。

  ```
  --正确示例
  BEGIN;
  CREATE TABLE public.test (
   "id" text NOT NULL,
   "ds" text NOT NULL,
  PRIMARY KEY (id,ds)
  );
  CALL SET_TABLE_PROPERTY('public.test', 'orientation', 'column');
  COMMIT;
  ```

- 表名和列名均对大小写不敏感，如需定义大写表名、大写列名、特殊字符表名或列名，可使用双引号（""）进行转义。示例如下：

```
create table "TBL" (a int);
select relname from pg_class where relname = 'TBL';
insert into "TBL" values (-1977);
select * from "TBL";
----------------------------------------------------------------------
begin;
create table tbl ("C1" int not null);
call set_table_property('tbl', 'clustering_key', '"C1"');
commit;
----------------------------------------------------------------------
begin;
create table tbl ("C1" int not null, c2 text not null);
call set_table_property('tbl', 'clustering_key', '"C1,c2:desc"');  -- set_table_property
见下文
commit;
----------------------------------------------------------------------
create table "Tab_$A%*" (a int);
select relname from pg_class where relname = 'Tab_$A%*';
insert into "Tab_$A%*" values (-1977);
select * from "Tab_$A%*";
```

- `IF NOT EXISTS`：在创建表时，如果不存在同名表且语义正确，表创建都会返回成功。如果不指定 `IF NOT EXISTS` 选项而存在同名表，则返回异常。如果指定 `IF NOT EXISTS` 选项，Hologres会提示信息，跳过表创建步骤，返回成功，直观的规则如下。

配置项	指定if not exists	不指定if not exists
存在同名表	`NOTICE: relation "xx"already exists, skippingSUCCEED`	`ERROR: relation is already exists.`
不存在同名表	`SUCCEED`	`SUCCEED`

- 表名的长度不超过64字节，超过64字节将被截断。
- 不支持修改数据类型，如果必须修改，请重新建表。
- 行存表必须设置主键，行列共存表必须设置主键，列存表不要求有主键。
- orientation、distribution_key、clustering_key、event_time_column属性在建表后不支持更改，如需修改，需要重新建表；bitmap和dictionary属性可以在建表后更改。

建表

- **命令格式**

  目前Hologres语法是PostgreSQL的一个子集，支持的建表语法如下。

  > **说明** 当前Hologres仅DDL支持事务处理，DML不支持事务处理。

```
begin;
create table [if not exists] [schema_name.]table_name ([
  {
   column_name column_type [column_constraints, [...]]
   | table_constraints
   [, ...]
  }
]);
call set_table_property('<table_name>', property, value);
commit;
```

- 参数说明
  - column_type：为字段的数据类型，已支持的数据类型可以参见数据类型汇总。
  - 列约束 `column_constraints` 和表约束 `table_constraints` 的支持情况如下。

参数	column_constraints	table_constraints
primary key	支持	支持
not null	支持	-
null	支持	-
unique	不支持	不支持
check	不支持	不支持
default	支持	不支持

  - set_table_property为表设置属性，详情请参见设置表属性。
  - 声明default约束的表，暂时无法通过Fixed Plan实现高吞吐数据写入和更新，有关Fixed Plan，请参考Fixed Plan加速SQL执行。

设置表属性

在Hologres中，可以通过set_table_property为表设置多种属性，合理的表属性设置可以有助于系统高效地组织和查询数据。与数据存储布局有关的参数需要和建表语句同时执行，其中orientation、distribution_key、clustering_key和event_time_column创建后当前版本不支持修改。

- 命令格式

```
call set_table_property('<table_name>', property, value);
```

> 说明 ： `set_table_property` 的调用需要与 `create table` 在同一事务中执行。

Hologres当前版本支持的设置表属性有以下几种。

```
call set_table_property('table_name', 'orientation', '[column | row | row,column]');
call set_table_property('table_name', 'table_group', '[tableGroupName]');
call set_table_property('table_name', 'distribution_key', '[columnName[,...]]');
call set_table_property('table_name', 'clustering_key', '[columnName{:[desc|asc]} [,...]]
');
call set_table_property('table_name', 'event_time_column', '[columnName [,...]]');
call set_table_property('table_name', 'bitmap_columns', '[columnName [,...]]');
call set_table_property('table_name', 'dictionary_encoding_columns', '[columnName [,...]]
');
call set_table_property('table_name', 'time_to_live_in_seconds', '<non_negative_literal>'
);
```

- **参数说明**

  具体参数和相关内容如下表所示：

参数	列存表	行存表	行列共存表	建议值	建表后是否可修改
orientation	column（默认值）	row	row,column	column	否，如需修改请重新建表。
table_group	默认为default table gorup。	默认为default table gorup。	默认为default table gorup。	默认即可。	否，如需修改请重新建表或者Resharding。
distribution_key	默认为主键，根据业务场景修改。	默认为主键。	默认为主键。	主键的子集，建议只选择一列。	否，如需修改请重新建表。
clustering_key	默认为空。	默认为主键。	默认为空。	建议最多选择一列。	否，如需修改请重新建表。
event_time_column	默认为第一个非空时间戳字段。	不支持。	默认为第一个非空时间戳字段。	建议时间戳字段。	否，如需修改请重新建表。
bitmap_columns	按需使用。	不支持。	按需使用。	建议用于等值比较的列，一般10列以下。	是，详情请参见ALTER TABLE。
dictionary_encoding_columns	按需使用。	不支持。	按需使用。	建议低基数列，一般10列以下。	是，详情请参见ALTER TABLE。
time_to_live_in_seconds	按需使用。	按需使用。	按需使用。	默认即可，无需设置。	是，详情请参见ALTER TABLE。

  - **orientation**

    ```
    call set_table_property('table_name', 'orientation', '[column | row |row,column]');
    ```

    - orientation：指定了数据库表在Hologres中的存储模式是列存还是行存，从HologresV1.1版本开始支持行列共存的格式。

- 在Hologres中表默认为列存（column store）形式，表的存储格式说明如下。

存储格式	适用场景	使用说明
列存	适用于OLAP场景，适合各种复杂查询、数据关联、扫描、过滤和统计。写入和更新效率低于行存表。	列存会默认创建更多的索引，包括对字符串类型创建bitmap索引，这些索引可以显著加速查询过滤和统计，因此列比较多的表，会占用更多的存储空间，您可以通过修改表属性关闭这些默认创建的索引，释放存储空间。
行存	适用于 KV（key-value）场景 ，适合基于primary key的点查和扫描。写入和更新都更友好。	行存默认仅对主键创建索引，仅支持主键的快速查询，因此使用的存储空间更少，但使用场景也受到限制。
行列共存	同时适用列存和行存的场景，既支持高效点查也支持OLAP分析，但也带来了更多的存储开销，以及内部数据状态同步的开销。在部分列局部更新场景优势明显，显著快于列存，但弱于行存。	在行列共存中，必须有主键，其他类型的索引与列存的行为对齐。

- 使用示例

```
--建行存表
begin;
create table public.tbl_row (
    a integer NOT NULL,
    b text NOT NULL
    ,PRIMARY KEY (a)
);
call set_table_property('public.tbl_row', 'orientation', 'row');
commit;
--建列存表
begin;
create table tbl_col (
  a int not null,
  b text not null);
call set_table_property('tbl_col', 'orientation', 'column');
commit;
--建行列共存
begin;
create table tbl_col_row (
    pk  text  not null,
    col1 text,
    col2 text,
    col3 text,
    PRIMARY KEY (pk));
call set_table_property('tbl_col_row', 'orientation', 'row,column');
commit;
```

- table_group

```
call set_table_property('table_name', 'table_group', '[tableGroupName]');
```

  - table_group：指定了表在Hologres中所在的Table Group。
  - Hologres中，表默认会创建在默认的Table Group中（新建数据库后，如果没有创建新的Table Group，那么创建第一个表时，会自动建立一个默认的Table Group，名称为 `<db>_tg_default` ）。一般情况下无需创建Table Group，详情请参见Table Group与Shard Count操作指南。

- distribution_key

```
call set_table_property('table_name', 'distribution_key', '[columnName[,...]]');
```

  - distribution_key：指定了数据库表分布策略。数据根据distribution_key被分配到各个shard上。系统保证distribution_key相同的记录会被分配到同一个shard上，算法为Hash(distribution_key)%shard_count的结果为对应的shard。
  - columnName部分如设置单列，不要有多余空格。如设置多列，则以逗号分隔，同样不要有多余的空格。
  - distribution_key指定的列或列组合不支持Float、Double、Decimal(Numeric)、Date、Timestamp、Timestamptz、Array、Json、Serial、Bytea、INET、MONEY及其他复杂数据类型。
  - 当表中有primary key时，distribution_key默认为primary key。distribution_key必须为primary key或者primary key中的部分字段（不能为空），同一记录的数据只能属于一个shard。当表中没有primary key时，对distribution_key没有限制，可以为空（不指定任何列）。如果distribution_key为空，即随机shuffle，数据随机分布到不同shard上。当distribution_key对应列的值为空时，当作 `""` （空串）看待。
  - Hologres中，distribution_key是非常重要的分布式概念。合理的设置distribution_key可以达到如下效果：
    - 提高性能。不同的Shard可以进行并行计算，从而提高性能。
    - 提高QPS。当您以distribution_key做过滤条件时，Hologres可以直接筛选出数据相关的Shard进行扫描。否则，Hologres需要让所有的Shard参与计算，会影响QPS。
    - 提高Join性能。当两张表在同一个Table Group内，并且Join key是distribution_key时，那么数据分布保证表A一个Shard内的数据和表B同一Shard内的数据对应，只需要直接在本节点Join本节点数据（Local Join）即可，可以大大提高执行效率。

- 使用示例

```
--对a列的值做hash操作，再取模，即hash(a)%shard_count = shard_id，结果相同的一组数据分布在同一个Shard内
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'distribution_key', 'a');
commit;
--对a,b两个列的值做hash操作，再取模，即hash(a,b)%shard_count = shard_id，结果相同的一组数据分布在同一个Shard内
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'distribution_key', 'a,b');
commit;
--tbl1按照a列分布，tbl2按照c列分布，当tbl1与tbl2以a=c关联条件join时，对应的数据分布在同一个Shard内，这种查询可以实现Local Join的加速效果
begin;
create table tbl1(a int not null, b text not null);
call set_table_property('tbl1', 'distribution_key', 'a');
create table tbl2(c int not null, d text not null);
call set_table_property('tbl2', 'distribution_key', 'c');
commit;
select b, count(*) from tbl1 join tbl2 on tbl1.a = tbl2.c group by b;
```

- clustering_key

```
call set_table_property('table_name', 'clustering_key', '[columnName{:asc} [,...]]');
```

- clustering_key：在指定的列上建立聚簇索引。Hologres会在聚簇索引上对数据进行排序，建立聚簇索引能够加速在索引列上的*range*和*filter*查询。clustering_key指定了底层存储文件内部的数据有序关系，因此对范围查询有加速效果。
- 必须为 `not nullable` 的列或者列组合，不支持Float、Double、Decimal(Numeric)、Array、Json及其他复杂数据类型。
- clustering_key指定列时，可在列名后添加 `:asc` （升序）来表明构建索引时的排序方式。
- 行存表的clustering_key默认为主键 （V0.9之前的版本默认不设置）。如果设置为和主键不同的clustering_key，那么Hologres会为这张表生成两个排序（ `primary_key` 排序和 `clustering_key` 排序），造成数据冗余。
- 列存表的clustering_key默认为空。
- 由于clustering_key用于排序，所以clustering_key里的列组合排在前面的优先级更高，clustering_key建议仅保留1～2列。

- clustering_key可以用于在clustering index最开始几列的*range*和*filter*的加速查询，即**查询具备左匹配原则，不匹配则无法利用clustering_key查询加速**。

  假设表table1的clustering_key设置为col1和col2，那么下面的*query*可以被加速：

```
-- 可加速
select * from table1 where col1='abc';
-- 可加速
select * from table1 where col1>'xxx' and col1<'abc';
-- 可加速
select * from table1 where col1 in ('abc','def');
-- 可加速
select * from table1 where col1='abc' and col2='def';
-- 不可加速
select col1,col4 from table1 where col2='def';
```

- 使用示例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'clustering_key', 'a,b');
commit;
------------------------------------------------------------
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'clustering_key', 'a,b:asc');
commit;
```

- **event_time_column**

```
call set_table_property('table_name', 'event_time_column', '[columnName [,...]]');
```

- event_time_column：指定时间列作为分段键，必须为时间类型强相关的列（如果数据有更新的话，需要和update time强相关）。当查询条件包含event_time_column时，查询可以通过event_time_column快速找到相应数据对应的存储位置。适用于日志、流量等和时间强相关的数据，合理设置可极大提升性能。event_time_column指定了底层存储文件之间的边界条件，为每个文件记录了边界最大值、最小值，因此对于文件按范围扫描场景有加速作用。
- event_time_column必须为not nullable的列或者列组合，不支持Float、Double、Decimal(Numeric)、Array、Json及其他复杂数据类型。
- 行存表不可设置event_time_column。
- 设置event_time_column要求orientation为column，即列存表。
- 列存表默认将table schema中的第一个非空的timestamp/timestamptz的字段作为event_time_column，如果不存在这样的字段，则默认将第一个非空的date类型的字段作为event_time_column（V0.9之前的版本默认为空）。
- event_time_column原名为segment_key，在0.9版本默认改名为event_time_column，segment_key依旧向下兼容使用。
- 使用示例

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'event_time_column', 'a,b');
commit;
```

- bitmap_columns

```
call set_table_property('table_name', 'bitmap_columns', '[columnName{:[on|off]}[,...]]'
);
```

其中，参数说明如下表所示：

参数	说明
table_name	表名称。
on	当前字段打开bitmap_columns。
off	当前字段关闭bitmap_columns。

- bitmap_columns：比特编码列。bitmap可以对存储文件内部的数据进行快速等值过滤，所以建议把等值filter条件的数据建成比特编码。bitmap_columns是数据存储之外的独立索引结构，以位图向量结构加速等值比较场景。
- 设置bitmap_columns要求表的存储形式为column，即列存表。
- bitmap_columns适合取值不多的列，对于每个取值构造一个二进制串，表示取值所在位置的bitmap。
- bitmap_columns指定的列可以为空。
- 当前版本默认所有text列都会被隐式地设置到bitmap_columns中。
- **可以在事务之外单独使用，表示修改bitmap_columns列，修改之后非立即生效，比特编码构建和删除在后台异步执行。**详情请参见ALTER TABLE。
- 使用示例

```
--创建tbl并设置bitmap索引
begin;
create table tbl (
  a int not null,
  b text not null);
call set_table_property('tbl', 'bitmap_columns', 'a:on,b:off');
commit;
--修改bitmap索引
call set_table_property('tbl', 'bitmap_columns', 'a:off');//全量修改
call update_table_property('tbl', 'bitmap_columns', 'b:off');//增量修改
```

- dictionary_encoding_columns

```
call set_table_property('table_name', 'dictionary_encoding_columns', '[columnName{:[on|off|auto]}[,...]]');
```

其中，参数说明如下表所示：

参数	说明
table_name	表名称。
on	表示当前字段打开dictionary_encoding_columns。
off	表示当前字段关闭dictionary_encoding_columns。
auto	表示自动。如果设置了auto，Hologres会根据所在列数值的重复程度自动选择是否进行dictionary_encoding_columns，值的重复度越高，字典编码的收益越大。在Hologres V0.8版本及更早版本中默认所有text列都会被设置为dictionary_encoding_columns，在Hologres V0.9版本及之后版本，会根据数据特征自动选择是否创建字典编码。

- dictionary_encoding_columns：字典编码列，为指定列的值构建字典映射。字典编码可以将字符串的比较转成数字的比较，加速group by、filter等查询。dictionary_encoding_columns是一种压缩存储的技术，将原始数值编码为数值类型存储，同时也会维护对应的编码表结构，在数据读取时，会根据编码表进行数据解码操作，解码会带来额外的计算开销。
- 设置dictionary_encoding_columns要求表的存储形式为column，即列存表。
- dictionary_encoding_columns指定的列可以为null。
- 取值较少的列适合设置dictionary_encoding_columns，可以压缩存储。
- V0.8及更早版本中默认所有text列都会被隐式地设置到dictionary_encoding_columns中。V0.9及之后的版本会根据数据特征自动选择是否创建字典编码。
- **可以在事务之外单独使用。表示修改dictionary_encoding_columns列，修改之后非立即生效，字典编码构建和删除在后台异步执行。**详情请参见ALTER TABLE。
- 使用示例

```
--创建表tbl并设置dictionary_encoding_columns索引
begin;
create table tbl (
  a int not null,
  b text not null,
  c text not null
);
call set_table_property('tbl', 'dictionary_encoding_columns', 'a:on,b:off,c:auto');
commit;
--修改dictionary_encoding_columns索引
call set_table_property('tbl', 'dictionary_encoding_columns', 'a:off');--全量修改
call update_table_property('tbl', 'dictionary_encoding_columns', 'b:off');--增量修改
```

- time_to_live_in_seconds

```
call set_table_property('table_name', 'time_to_live_in_seconds', '<non_negative_literal
>');
```

time_to_live_in_seconds：表数据的生存时间TTL，单位为秒，必须是非负数字类型，整数或浮点数均可。

- 注意事项
  - 若是没有显示指定TTL，则默认数据永久保留（100年）。
  - TTL过期时间以数据第一次写入的时间开始计算，而不是数据最后一次修改的时间；当到达TTL后，表数据会在某一段时间内被清除（没有固定时间段），只是数据被清除，表不会被删除。
  - 可以在事务之外单独使用，表示修改表数据生存时间。
  - TTL相比于 `delete from tbl where xxx_time + TTL < now();` 几乎不占资源，也不会锁表，但是对于过期的数据不能保证数据一致性。这意味着：
    - 读取过期数据，可能能读到，可能读不到，也可能读到某个历史版本。
    - 修改或删除过期数据，可能正常工作，也可能出现PK重复之类的意外结果。
    - 使用TTL功能，最好要保证不会读取或修改过期的数据，或者对过期数据的一致性不关心。
- 使用示例

```
--建表时指定TTL
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'time_to_live_in_seconds', '864000');
commit;
--修改TTL
call set_table_property('tbl', 'time_to_live_in_seconds', '86400');
```

> **说明** 表数据的TTL并不是精确的时间，当超过设置的TTL后，系统会在某一个时间自动删除表数据，所以业务逻辑不能强依赖TTL。若是想精确的删除表数据，可以使用DataWorks，进行调度任务配置来删除数据。

增加注释

Hologres支持给表、外表、列等增加注释（comment）的功能。

增加注释的使用示例如下：

- 新建表时添加注释。

```
BEGIN;
CREATE TABLE public."user" (
 "id" text NOT NULL,
 "name" text NOT NULL
);
COMMENT ON TABLE public."user" is '用户属性表';
COMMENT ON COLUMN public."user".id is '身份证号';
COMMENT ON COLUMN public."user".name is '姓名';
COMMIT;
```

- 已有表添加注释。

```
-- 给表增加注释
COMMENT ON TABLE table_name IS 'my comments on table table_name.';
-- 给列增加注释
COMMENT ON COLUMN table_name.col1 IS 'This my first col1';
-- 给外部表增加注释
COMMENT ON FOREIGN TABLE foreign_table IS ' comments on my foreign table';
```

更多关于注释的用法请参见PostgreSQL comment。

查看表结构

您可以执行如下命令查看TABLE的具体DDL。

```
--需要Superuser执行
select hg_dump_script('tablename');
--部分早期版本会提示函数不存在，可以用过以下命令，手动加载相关函数，该命令是DB级别，一个DB执行一次即可
create extension hg_toolkit;
```

> **说明** 也可以通过HoloWeb，在元数据管理模块进行DDL查看。

使用示例

- 新建普通列存表并指定Primary Key。

  > **说明** Distribution Key必须是Primary Key的子集。

  ```
  begin;
  CREATE TABLE tbl (
   "id" bigint NOT NULL,
   "name" text NOT NULL,
   "age" bigint,
   "class" text NOT NULL,
   "reg_timestamp" timestamptz NOT NULL,
  PRIMARY KEY (id,age)
  );
  call set_table_property('tbl', 'orientation', 'column');
  call set_table_property('tbl', 'distribution_key', 'id');
  call set_table_property('tbl', 'clustering_key', 'age');
  call set_table_property('tbl', 'event_time_column', 'reg_timestamp');
  call set_table_property('tbl', 'bitmap_columns', 'name,class');
  call set_table_property('tbl', 'dictionary_encoding_columns', 'class:auto');
  commit;
  ```

- 新建分区表并指定Primary Key。

  > **说明** 分区表有主键时，主键里面必须包含分区字段。

```
begin;
CREATE TABLE www (
 name text NOT NULL,
 ds text NOT NULL,
 age text NOT NULL,
PRIMARY KEY (name,ds)
)
PARTITION BY LIST(ds);
CALL SET_TABLE_PROPERTY('www', 'orientation', 'column');
commit;
```

- 新建普通表并设置默认值。

```
--设置当前时间
CREATE TABLE test(
  id TIMESTAMPTZ DEFAULT now()
);
--设置数据的默认值
CREATE TABLE products (
    product_no integer,
    name text,
    price FLOAT  DEFAULT 1.99
);
```

HoloWeb可视化新建内部表

HoloWeb提供可视化一键建表功能，无需写SQL命令就能创建表，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在**HoloWeb**页面顶部菜单栏，单击**元数据管理 > 表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**新建内部表**。
3. 在**新建内部表**页面，配置各项参数。

类别	参数	描述
基本属性	实例名	已登录的实例名称。
	数据库	当前已登录实例的数据库名称。
	表名	新建的Hologres内部表名称。
	描述	新建的Hologres内部表描述。

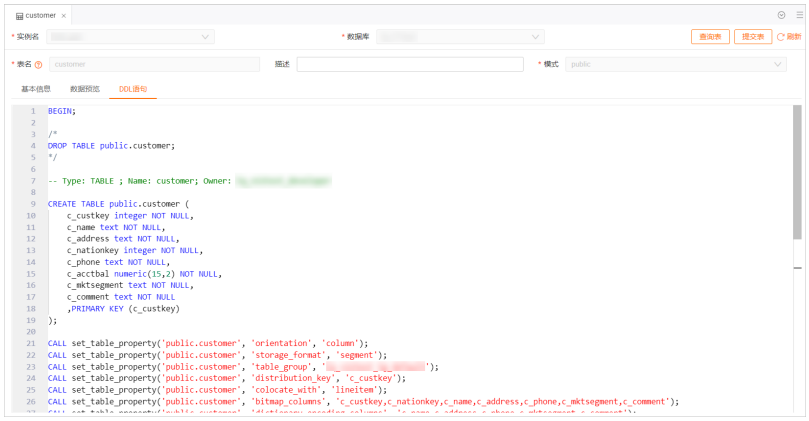
类别	参数	描述
	模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。
字段	字段名	表中每一列的标识。
	数据类型	字段取值的类型。
	主键	表中每条数据的唯一标识。
	可空	字段是否可以设置为空。
	数组	有序的元素序列。
	描述	字段的描述信息。
	操作	包括**删除**、**上移**和**下移**。
属性	存储模式	包括**列存**和**行存**两种存储模式。 默认为**列存**。
	生命周期（秒）	数据第一次写入的时间开始计算，当到达生命周期后，表数据会在某一段时间内被清除（没有固定时间段）。 默认生命周期为**永久**。
	Binlog	表是否开启Binlog，详情请参见订阅Hologres Binlog。
	Binlog生命周期	Binlog的生命周期，详情请参见订阅Hologres Binlog。默认生命周期为**永久**。
	分布列	distribution_key，使用详情请参见设置表属性。
	分段列	event_time_column ，使用详情请参见设置表属性。
	聚簇列	clustering_key，使用详情请参见设置表属性。
	字典编码列	dictionary_encoding_columns，使用详情请参见设置表属性。
	位图列	bitmap_columns，使用详情请参见设置表属性。
分区表	无	选择分区字段。

4. 在页面右上角，单击**提交表**。提交之后，您可以在左侧对应模式下，刷新出新建的内部表。
5. （可选）表数据预览
   i. 在**已登录实例**列表，双击目标内部表。

ii. 进入表信息页签，单击**数据预览**，则可以预览表数据。

6. （可选）DDL预览

在目标表信息页签，单击**DDL语句**，则可以预览DDL语句。



```
BEGIN;

/*
DROP TABLE public.customer;
*/

-- Type: TABLE ; Name: customer; Owner:

CREATE TABLE public.customer (
    c_custkey integer NOT NULL,
    c_name text NOT NULL,
    c_address text NOT NULL,
    c_nationkey integer NOT NULL,
    c_phone text NOT NULL,
    c_acctbal numeric(15,2) NOT NULL,
    c_mktsegment text NOT NULL,
    c_comment text NOT NULL
    ,PRIMARY KEY (c_custkey)
);

CALL set_table_property('public.customer', 'orientation', 'column');
CALL set_table_property('public.customer', 'storage_format', 'segment');
CALL set_table_property('public.customer', 'table_group', '');
CALL set_table_property('public.customer', 'distribution_key', 'c_custkey');
CALL set_table_property('public.customer', 'colocate_with', 'lineitem');
CALL set_table_property('public.customer', 'bitmap_columns', 'c_custkey,c_nationkey,c_name,c_address,c_phone,c_mktsegment,c_comment');
```

6.5.3.2. CREATE TABLE LIKE

CREATE TABLE LIKE语句用于创建一个同Select Query结果相同的表。本文为您介绍CREATE TABLE LIKE的用法。

使用限制

- Hologres V0.9及以下版本，`CREATE TABLE LIKE`语句仅支持复制表结构，不支持复制表属性（主键、索引等）。请在Hologres管理控制台的实例详情页查看当前版本。
- Hologres V0.10版本开始，支持`CREATE TABLE LIKE`语句复制表结构和表属性（主键、索引等）。但仅限于`SELECT * FROM TABLE`语法。通过执行以下命令可以复制表属性：

```
set hg_experimental_enable_create_table_like_properties=true;
```

- 使用`CREATE TABLE LIKE`创建的表不会自动同步源表的数据。
- 查询语句中的每一个目标列都要有一个不重复的别名，否则建表语句就会生成同列名的语句导致执行报错。示例如下：

```
CALL hg_create_table_like('new_table', 'select *, 1 as c, ''a'' as c from src_table');
ERROR:  column "c" specified more than once
CONTEXT:  SQL statement "create table new_table (
"a"     integer,
"b"     text,
"c"     integer,
"c"     text
);"
PL/pgSQL function hg_create_table_like(text,text) line 22 at EXECUTE
```

普通表

1. **命令格式**

在Hologres中，普通表 `CREATE TABLE LIKE` 的命令格式如下：

```
--复制一张普通表，但不能复制表属性
CALL hg_create_table_like('new_table_name', 'select_query');
--复制一张表并复制表属性
set hg_experimental_enable_create_table_like_properties=true;--开关参数，session级别
CALL hg_create_table_like('new_table_name', 'select * from old_table_name');
```

> **说明** 通过调用hg_create_table_like，系统会根据selec_query结果的schema创建一个表名为new_table_name的表，但不会插入任何数据。

2. **参数说明**
   - new_table_name：要创建表的表名（不支持创建外部表），只支持固定字符串，不支持字符拼接或函数生成等。
   - select_query：查询的SQL语句串。当SQL中的内容完全为 `select * from tablename` 时， `CREATE TABLE LIKE` 会自动同步创建原表的所有属性，包括pk、索引等。如果SQL语句中有较多单引号，可以将$$符号置于SQL语句前后，通过 `$$query_sql$$` 改写（推荐使用该用法，操作更简便）自动实现单引号转义，用法如下：

     ```
     CALL HG_CREATE_TABLE_LIKE ('table_name', $$query_sql$$ [, 'partition_clause'])
     ```

   - old_table_name：需要复制的原表名。
3. **使用示例**

   如在Hologres中存在如下源表：

   ```
   BEGIN;
   CREATE TABLE public.src_table (
    "a" int8 NOT NULL,
    "b" text NOT NULL,
   PRIMARY KEY (a)
   );
   CALL SET_TABLE_PROPERTY('public.src_table', 'orientation', 'column');
   CALL SET_TABLE_PROPERTY('public.src_table', 'bitmap_columns', 'b');
   CALL SET_TABLE_PROPERTY('public.src_table', 'dictionary_encoding_columns', 'b:auto');
   CALL SET_TABLE_PROPERTY('public.src_table', 'time_to_live_in_seconds', '3153600000');
   CALL SET_TABLE_PROPERTY('public.src_table', 'distribution_key', 'a');
   CALL SET_TABLE_PROPERTY('public.src_table', 'storage_format', 'segment');
   COMMIT;
   ```

   在Hologres中 `CREATE TABLE LIKE` 的示例用法如下：

   - 创建一个同源表结构和属性相同的表

     ```
     --复制表及属性
     set hg_experimental_enable_create_table_like_properties=true;
     CALL hg_create_table_like('new_table', 'select * from src_table');
     ```

   - 创建一个在源表的基础上再增加字段的表

     ```
     --新增一个同b一样的字段c
     CALL hg_create_table_like('holo_table_1', $$select *, "b" as c from src_table$$);
     ```

分区表

1. **命令格式**

   在Hologres中，分区表的 `CREATE TABLE LIKE` 的命令格式如下：

   ```
   --复制一张分区表，但不能复制表属性
   CALL hg_create_table_like('new_table_name', 'select_query', 'partition_clause');
   ```

2. **参数说明**

   - new_table_name：要创建表的表名（不支持创建外部表），只支持固定字符串，不支持字符拼接或函数生成等。
   - select_query：查询的SQL语句串。当SQL中的内容完全为 `select * from tablename` 时， `CREATE TABLE LIKE` 会自动同步创建原表的所有属性，包括pk、索引等。如果SQL语句中有较多单引号，可以将$$符号置于SQL语句前后，通过 `$$query_sql$$` 改写（推荐使用该用法，操作更简便）自动实现单引号转义，用法如下：

     ```
     CALL HG_CREATE_TABLE_LIKE ('table_name', $$query_sql$$ [, 'partition_clause'])
     ```

   - partition_clause：分区相关的语法定义。用于指定分区键，不会自动创建分区子表，需要手动建分区子表。

3. **使用示例**

   如在Hologres中存在如下源表：

   ```
   BEGIN;
   CREATE TABLE public.src_table (
    "a" int8 NOT NULL,
    "b" text NOT NULL,
   PRIMARY KEY (a)
   );
   CALL SET_TABLE_PROPERTY('public.src_table', 'orientation', 'column');
   CALL SET_TABLE_PROPERTY('public.src_table', 'bitmap_columns', 'b');
   CALL SET_TABLE_PROPERTY('public.src_table', 'dictionary_encoding_columns', 'b:auto');
   CALL SET_TABLE_PROPERTY('public.src_table', 'time_to_live_in_seconds', '3153600000');
   CALL SET_TABLE_PROPERTY('public.src_table', 'distribution_key', 'a');
   CALL SET_TABLE_PROPERTY('public.src_table', 'storage_format', 'segment');
   COMMIT;
   ```

   在Hologres中使用 `CREATE TABLE LIKE` 创建一张分区表示例如下：

   - 创建一个分区表

     ```
     --新增一个字段ds，并将表设置为以ds为分区的分区表
     CALL hg_create_table_like('new_table', $$select *, "b" as ds from src_table$$, 'parti
     tion by list(ds)');
     ```

   - 给对应的分区表创建分区子表

     ```
     create table new_table_child_20201213 partition of new_table for values in('20201213'
     );--以20201213为分区值
     create table new_table_child_20201214 partition of new_table for values in('20201214'
     );--以20201214为分区值
     ```

HoloWeb可视化复制表

HoloWeb提供可视化复制表功能，无需写SQL命令就能复制表，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在**HoloWeb**页面顶部菜单栏，单击**元数据管理**。
3. 在**元数据管理**页面左侧的**已登录实例**列表，鼠标右击要复制的表，选择**复制表结构**。
4. 在**复制表结构**页签，配置如下参数。

类别	参数	说明
目标位置	表名	目标表的表名称，可自定义命名。默认为：源表_copy。
	描述	对目标表的描述，可选择是否配置。
	模式	目标表所在的SCHEMA，默认为：public。
高级选项	是否同步源表属性	选择目标表是否同步源表的属性。 **说明** 仅Hologres V0.10及以上版本实例支持同步源表属性，如果您的实例低于V0.10版本，请选择否或升级实例。

5. 单击右上角**提交**，完成表复制。

6.5.3.3. ALTER TABLE

ALTER TABLE语句用于修改表，其中对分区父表的修改会自动应用到分区子表中。本文为您介绍ALTER TABLE的用法。

使用限制

Hologres当前对修改表的支持情况如下：

- 目前支持对表进行重命名、增加列和修改表数据生存时间的操作。
- 支持修改字段的默认值、dictionary_encoding_columns和bitmap_columns属性。
- 目前不支持修改数据类型。

重命名

ALTER TABLE语句可以对表进行重命名，如果目标表不存在，或者重命名目标表为已存在的表名称，系统均会返回异常。

说明 目前不支持跨Schema对表进行重名命操作。

- 使用语法

```
--内部表重命名
ALTER TABLE [schema_name.]<table_name> RENAME TO <new_table_name>;
--外部表重命名
ALTER FOREIGN TABLE [schema_name.]<foreign_table_name> RENAME TO <new_foreign_table_name>
;
```

- 使用示例

```
--将表holo_test重命名为holo_test_1
ALTER TABLE public.holo_test RENAME TO holo_test_1 ;
--将外部表foreign_holo_test重命名为foreign_holo_test_1
ALTER FOREIGN TABLE public.foreign_holo_test RENAME TO foreign_holo_test_1;
```

增加列

ALTER TABLE语句可以给表增加列，仅支持在表的最后一列之后增加新的列。

- 使用语法

```
--新增一列
ALTER TABLE IF EXISTS [schema_name.]<table_name> ADD COLUMN <new_column> <data_type>;
--新增多列
ALTER TABLE IF EXISTS [schema_name.]<table_name> ADD COLUMN <new_column_1> <data_type>, A
DD COLUMN <new_column_2> <data_type>;
```

- 使用示例

```
--在表holo_test中增加id列
ALTER TABLE IF EXISTS public.holo_test ADD COLUMN id int;
```

重命名列

Hologres从V1.1版本开始，支持重命名列，具体语法如下。

> **说明**
>
> - 如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
> - 若您的表是分区表，由于存在分区子表和父表数据结构一致性的要求，仅支持重命名分区父表的列，不支持单独重命名某个分区子表的列。重命名分区父表的列，所有子表自动生效。
> - 不支持同时重命名多个表的列名称。
> - 仅只有表Owner才能重命名列，若您的数据库使用的是简单权限模型，需要设置为developer用户组权限。

- 使用语法

```
ALTER TABLE [schema_name.]<table_name> RENAME COLUMN <old_column_name> TO <new_column_nam
e>;
```

- 使用示例

```
--将表holo_test的id列重命名为name
ALTER TABLE public.holo_test RENAME COLUMN id TO name;
```

修改默认值

ALTER TABLE语句支持修改默认值设置（常量或常量表达式），该默认值仅对设置之后新写入/更新数据有效，不会更新表中已有数据的默认值。当前仅Hologres V0.9.23及以上版本支持修改默认值。具体修改方式说明如下：

- 使用语法

```
--修改表字段的默认值
ALTER TABLE [schema_name.]<table_name> ALTER COLUMN <column> SET DEFAULT <expression>;
--删除表字段的默认值
ALTER TABLE [schema_name.]<table_name> ALTER COLUMN <column> DROP DEFAULT;
```

- 使用示例

```
--修改表holo_test中id列的默认值为0
ALTER TABLE holo_test ALTER COLUMN id SET DEFAULT 0;
--删除表holo_test中id列的默认值
ALTER TABLE holo_test ALTER COLUMN id DROP DEFAULT;
```

修改表属性

Hologres支持通过执行语句修改参数，达到修改表属性的目的。具体修改方式说明如下：

- 修改dictionary_encoding_columns字典编码列。修改Dictionary Encoding设置，会引起数据文件重新编码存储，会在一段时间内消耗一部分CPU和内存资源，建议在业务低峰期执行变更。
  - 使用语法

```
--修改全量
CALL SET_TABLE_PROPERTY('[schema_name.]<table_name>', 'dictionary_encoding_columns', '[
columnName{:[on|off|auto]}[,...]]');
--修改增量，只修改call里面的指定字段，其余字段不变
CALL UPDATE_TABLE_PROPERTY('[schema_name.]<table_name>', 'dictionary_encoding_columns',
'[columnName{:[on|off|auto]}[,...]]');
```

  - 参数说明

参数	说明
table_name	需要和待修改的表名大小写保持一致，可以携带Schema信息。
on	表示当前字段打开dictionary_encoding_columns。
off	表示当前字段关闭dictionary_encoding_columns。
auto	表示自动。如果是设置了auto，Hologres会根据所在列数值的重复程度自动选择是否进行dictionary_encoding_columns，值的重复度越高，字典编码的收益越大。在Hologres V0.8版本及更早版本中默认所有text列都会被设置为dictionary_encoding_columns，在Hologres V0.9版本及之后版本，会根据数据特征自动选择是否创建字典编码。

- 使用示例
  - 对a列显示创建dictionary，b列自动选择是否创建dictionary，c、d两列不创建dictionary。

```
CREATE TABLE dwd.holo_test (
 a text NOT NULL,
 b text NOT NULL,
 c text NOT NULL,
 d text
);
CALL UPDATE_TABLE_PROPERTY('dwd.holo_test','dictionary_encoding_columns','a:on,b:auto
');
```

  - 对a列显示关闭dictionary，系统也会自动给b、c、d字段加上dictionary索引。

```
CREATE TABLE dwd.holo_test (
 a text NOT NULL,
 b text NOT NULL,
 c text NOT NULL,
 d text
);
CALL SET_TABLE_PROPERTY('dwd.holo_test','dictionary_encoding_columns','a:off');
```

- 修改bitmap_columns比特编码列

  Hologres从V0.9版本开始支持通过执行以下语句修改bitmap_columns，无需再重新建表即可修改表属性。

  - 使用语法

```
--修改全量
CALL SET_TABLE_PROPERTY('[schema_name.]<table_name>', 'bitmap_columns', '[columnName{:[
on|off]}[,...]]');
--修改增量，只修改call里面的指定字段，其余字段不变
CALL UPDATE_TABLE_PROPERTY('[schema_name.]<table_name>', 'bitmap_columns', '[columnName
{:[on|off]}[,...]]');
```

  - 参数说明

参数	说明
table_name	需要和待修改的表名大小写保持一致，可以携带Schema信息。
on	当前字段打开bitmap_columns。
off	当前字段关闭bitmap_columns。

- 使用示例
  - 对a列启动bitmap索引，对b、c、d不启动bitmap索引。

```
CREATE TABLE dwd.holo_test (
 a text NOT NULL,
 b text NOT NULL,
 c text NOT NULL,
 d text
);
CALL UPDATE_TABLE_PROPERTY('dwd.holo_test','bitmap_columns','a:on');
```

  - 对b关闭bitmap索引，系统会自动给a、c、d创建bitmap索引。

```
CREATE TABLE dwd.holo_test_1 (
 a text NOT NULL,
 b text NOT NULL,
 c text NOT NULL,
 d text
);
CALL SET_TABLE_PROPERTY('dwd.holo_test_1','bitmap_columns','b:off');
```

- 修改表数据的生存时间
  - 使用语法

```
call set_table_property('[schema_name.]<table_name>', 'time_to_live_in_seconds', '<non_negative_literal>');
```

  - 参数说明

参数	说明
time_to_live_in_seconds	简称TTL，表数据的生存时间，单位为秒，必须是非负数字类型，整数或浮点数均可。

> **说明** 表数据生存时间是按照数据写入Hologres开始，超过该指定时间，表数据将会在某个时间内被删除，但并不是精准的时间。

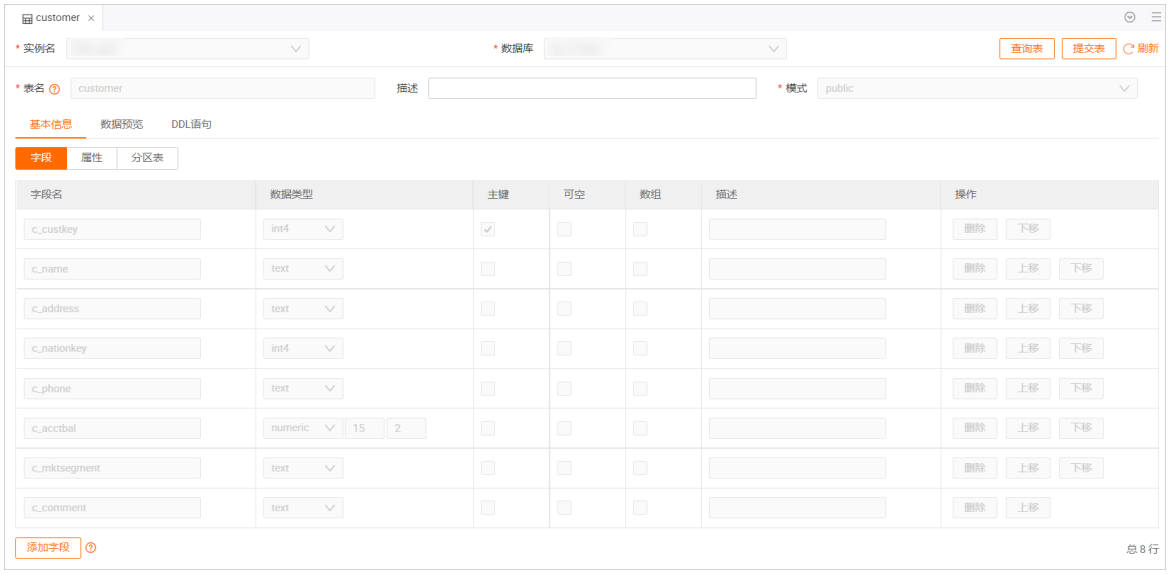
  - 使用示例

```
call set_table_property('dwd.holo_test', 'time_to_live_in_seconds', '600');
```

HoloWeb可视化修改表

HoloWeb提供可视化编辑表功能，无需写SQL命令就能修改表字段和部分表属性，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在**HoloWeb**页面顶部菜单栏，单击**元数据管理**。
3. 在**元数据管理**页面左侧的**已登录实例**列表，双击要修改的目标表。
4. 在表的详情页面，可视化修改表的字段和部分表属性。



5. 单击右上角的**提交表**，完成表修改。

6.5.3.4. DROP TABLE

DROP TABLE语句用于删除表。本文为您介绍DROP TABLE用法。

语法

```
DROP TABLE [ IF EXISTS ] table_name [, ...];
```

> **说明** `DROP TABLE` 支持一次删除多个表。

参数说明如下表所示。

参数	描述
IF EXISTS	• 如果指定 `IF EXISTS` ，无论目标表是否存在，执行删除语句后系统都会返回成功。 • 如果不指定 `IF EXISTS` ，当目标表不存在时，系统会返回 `ERROR: table "non_exist_table" does not exist` 报错。
table_name	需要删除的表名称。

示例

删除表的示例语句如下。

```
DROP TABLE holo_test;
```

HoloWeb可视化删除表

HoloWeb提供可视化删除表功能，无需写SQL命令就能删除表，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在**HoloWeb**页面顶部菜单栏，单击**元数据管理**。
3. 在**元数据管理**页面左侧的**已登录实例**列表，鼠标右击要删除的表，选择**删除表**。

4. 在**删除表**对话框，单击**确认**，完成删除表。

6.5.4. PARTITION TABLE

6.5.4.1. CREATE PARTITION TABLE

CREATE PARTITION TABLE语句用于创建分区表。本文为您介绍CREATE PARTITION TABLE的用法。

命令说明

Partition Table，也叫分区表。父表按分区键（Partition Key）的值划分为不同的子表，子表对外可见。下文无特殊说明的父表和子表皆指分区父表和子表。

分区表在使用时，需要提前创建子表。`create partition table`命令被用于创建分区表。

分区表的不同分区子表采用不同的文件存储，查询时带上分区条件，指定所需查询的分区，避免全表扫描，快速定位存储文件，提高处理效率。通常将事实表按照日期划分为不同的分区表。

使用限制

- Hologres暂不支持插入数据至分区表父表，只支持插入数据至具体的分区表子表。

> **说明** 实时计算Flink版支持实时写入数据至Hologres的分区表父表，详情请参见实时写入数据至Hologres的分区结果表。

- 只有*TEXT*、*VARCHAR*以及*INT*类型的数据才可以作为分区键（Partition Key）。
- 一个分区规则只能创建一个分区表。
- `PARTITION BY`类型仅支持`LIST`分区，切分`PARTITION BY LIST`只能取唯一值。
- 分区父表和子表必须要在同一个Schema。
- 若是表有主键，分区键必须是主键的一个子集。
- 动态分区管理功能仅Hologres V1.3及以上版本支持，请在Hologres管理控制台查看当前实例版本，如果您的实例是V1.3以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。

注意事项

- 如果您的数据来源于数据库，不建议使用分区表。过多的分区表会引起额外的IO资源浪费，为改善此问题并实现索引过滤加速查询的效果，您可以将常用分区条件作为segment_key。
- 如果单日分区数据小于1亿条，不建议使用分区表。分区表太小，查询加速效果不明显，可以选择较大粒度的分区。
- 如果您需要经常对某日数据进行整体替换，执行truncate操作，建议使用分区表。针对该场景，执行truncate效果更高，可以避免大范围的删除操作。

命令格式

创建分区表的命令格式如下。

```
--创建分区父表语句
CREATE TABLE [if not exists] [<schema_name>.]<table_name>  ([
  {
   <column_name> <column_type> [ <column_constraints>, [...]]
   | <table_constraints>
   [, ...]
  }
])
PARTITION BY LIST(<column_name>);
--创建分区子表语句
CREATE TABLE [if not exists] [<schema_name>.]<table_name> PARTITION OF <parent_table>
  FOR VALUES IN (<string_literal>);
```

参数说明

创建分区表的参数说明如下。

参数	说明
if not exists	如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。
schema_name	表所在的schema名称，若是在同一个schema创建父表和子表，可以不需要指定schema名称。若是需要跨schema创建父表和子表，需要指定schema名称。

参数	说明
table_name	需要创建的分区父表或分区子表的名称。
column_name	新表中要创建的字段名。
column_type	字段的数据类型。
column_constraints	列约束的名称。
table_constraints	表约束的名称。
parent_table	子表对应的父表名称。
string_literal	分区键。

使用示例

创建分区表的SQL语句示例如下。

- 示例1：在public schema下创建不带主键的分区父表和对应的分区子表。

```
BEGIN;
CREATE TABLE public.hologres_parent(
  a text,
  b int,
  c timestamp,
  d text
)
  PARTITION BY LIST(a);
CALL set_table_property('public.hologres_parent', 'orientation', 'column');
CREATE TABLE public.hologres_child1 PARTITION OF public.hologres_parent FOR VALUES IN('v1
');
CREATE TABLE public.hologres_child2 PARTITION OF public.hologres_parent FOR VALUES IN('v2
');
CREATE TABLE public.hologres_child3 PARTITION OF public.hologres_parent FOR VALUES IN('v3
');
COMMIT;
```

- 示例2：在public schema下创建带主键的分区父表和对应的分区子表。

```
BEGIN;
CREATE TABLE public.hologres_parent_2(
  a text ,
  b int,
  c timestamp,
  d text,
  ds text,
  primary key(ds,b)
  )
  PARTITION BY LIST(ds);
CALL set_table_property('public.hologres_parent_2', 'orientation', 'column');
CREATE TABLE public.holo_child_1 PARTITION OF public.hologres_parent_2 FOR VALUES IN('202
01215');
CREATE TABLE public.holo_child_2 PARTITION OF public.hologres_parent_2 FOR VALUES IN('202
01216');
CREATE TABLE public.holo_child_3 PARTITION OF public.hologres_parent_2 FOR VALUES IN('202
01217');
COMMIT;
```

查看所有分区子表

您可以通过如下两种方法查看当前分区父表下所有的分区子表：

- 通过HoloWeb可视化查看，HoloWeb会展示分区父表下面的所有分区子表。
- 通过执行如下命令语句，查看当前分区父表下所有的分区子表。其中，您可以将parent_table_name修改为实际的父表名称。

```
SELECT
    nmsp_parent.nspname AS parent_schema,
    parent.relname      AS parent,
    nmsp_child.nspname  AS child_schema,
    child.relname       AS child
FROM pg_inherits
    JOIN pg_class parent            ON pg_inherits.inhparent = parent.oid
    JOIN pg_class child             ON pg_inherits.inhrelid   = child.oid
    JOIN pg_namespace nmsp_parent   ON nmsp_parent.oid  = parent.relnamespace
    JOIN pg_namespace nmsp_child    ON nmsp_child.oid   = child.relnamespace
WHERE parent.relname='parent_table_name';
```

分区子表和父表约束一览表

分区子表在绑定父表时，其约束关系如下表所示。其中：

- 与父表保持一致：即对应的属性分区子表必须和父表保持一致，若不一致，在分区子表绑定（ATTACH）父表时会报错，需要重新创建分区子表。
- 不要求与父表一致：即对应的属性分区子表可以与父表不一致，如果子表没有显式指定属性，则会继承父表的属性，若是子表显式指定属性，则会保留子表的属性。
- 索引列必须包含父表的索引列：即分区子表的索引列必须包含父表的索引列，还能显式再指定其他列。

分类	表属性	描述	create table partition of 时是否继承父表属性	ATTACH时与父表的约束关系
表属性	orientation	表存储格式。	继承	与父表保持一致。
	table_group	Table Group属性包含Shard Count。	继承	与父表保持一致。
	time_to_live_in_seconds	表数据生命周期。	继承	不要求与父表一致。 • 子表属性未赋值，继承父表属性。 • 子表属性已赋值，保留子表属性。
索引	primary key	主键。	继承	与父表保持一致。
	distribution key	分布键。	继承	与父表保持一致。
	clustering_key（包括列和排序）	聚簇索引。	继承	与父表保持一致。
	event_time_column	分段键。	继承	与父表保持一致。
	bitmap_columns	比特编码。	继承	索引列必须包含父表的索引列。
	dictionary_encoding_columns	字段编码。	继承	索引列必须包含父表的索引列。
	binlog_level	是否开启Binlog。	继承	与父表保持一致。
	proxima_vectors	向量检索索引。	继承	索引列必须包含父表的索引列。
列约束	nullable	非空约束。	继承	与父表保持一致。
	default value	默认值。	继承	与父表保持一致。

动态分区管理

从Hologres V1.3版本开始支持根据用户指定的规则自动增删分区，减少用户管理分区生命周期的负担，系统会根据用户预设的规则定期运行任务，提前创建分区子表和删除过期分区子表。

- 配置动态分区

- 语法说明

  动态分区管理配置既支持在创建分区表时配置动态分区管理属性，也支持建表后修改动态分区管理属性，语法如下所示。

```
-- 创建分区表时配置动态分区管理属性
BEGIN;
CREATE TABLE [if not exists] [<schema_name>.]<table_name>  ([
  {
   <column_name> <column_type> [ <column_constraints>, [...]]
   | <table_constraints>
   [, ...]
  }
])
PARTITION BY LIST(<column_name>);
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.enable', 'xx
x');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.time_unit',
'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.time_zone',
'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.num_precreat
e', 'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.num_retentio
n', 'xxx');
COMMIT;
-- 修改动态分区管理属性
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.enable', 'xx
x');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.time_unit',
'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.time_zone',
'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.num_precreat
e', 'xxx');
CALL set_table_property('[<schema_name>.]<table_name>', 'auto_partitioning.num_retentio
n', 'xxx');
```

- 参数说明

参数	是否必选	说明	是否可更新
auto_partitioning.enable	否	是否启用动态分区管理，取值如下。 ■ true：启用动态分区管理。 ■ （默认）false：关闭动态分区管理。	是

auto_partitioning.time_unit	是	动态分区的时间单位，取值如下。 ■ HOUR ■ DAY ■ MONTH ■ QUARTER ■ YEAR 例如配置为DAY则系统将按天进行分区的预创建、删除。	否
auto_partitioning.time_zone	否	动态分区时区设置，配置后将按照对应时区时间点进行动态分区管理。您可以使用如下SQL查看可选的时区和offset等。返回结果中的name列即为timezone值，例如Asia/Shanghai。 `SELECT * FROM pg_timezone_names;` 默认值为当前连接的时区。	否
auto_partitioning.num_precreate	否	预创建分区数量，有如下取值。 ■ 0：不进行预创建。 ■ [1, 512]：以当前时间点为基准创建分区，建议该值需要大于等于2，默认值为4。 例如 `time_unit = DAY，num_precreate = 3` ：如果当前时间为2022-01-10将创建2022-01-10、2022-01-11、2022-01-12三个分区。	是
auto_partitioning.num_retention	否	保留历史分区数量，有如下取值。 ■ 0：不保留历史分区。 ■ （默认）-1：不清理历史分区。 ■ 正数： 保留N个历史分区，最大值为512。 例如 `<time_unit = DAY，num_retention = 3>` ：如果当前时间为2022-01-10，将保留 2022-01-09、2022-01-08 、2022-01-07三个分区；早于2022-01-07的历史分区将被删除。	是

- 分区表名生成规则

  自动分区表时间单位time_unit可以配置为 `day、month、quarter、year` ，自动分区将使用分区父表名加上时间后缀作为新创建分区的表名；分区表名的生成格式形如： `{parent_table}_{time_suffix}` ， 其中时间后缀将依据自动分区的调度时间和时间单位对应的格式模板生成。具体的对应关系如下。

time_unit	时间后缀格式	样例	执行时间
hour	YYYYMMDDHH24	2022030117。	每个整点的开始，例如2022年3月1日 01:00:01执行任务。
day	YYYYMMDD	20220301。	每天00:00:01开始，例如2022年3月1日 00:00:01。
quarter	YYYYQ	20221、20222、20223、20224分别表示2022年的四个季度。	每天00:00:01开始，例如2022年3月1日 00:00:01。
year	YYYY	2022、2023分别表示2022年、2023年的分区。	每天00:00:01开始，例如2022年3月1日 00:00:01。

- 使用样例

  如下SQL将以天为时间单位，预先创建未来3天的分区，保留近2天的历史分区，并将时区设置为'Asia/Shanghai'。

```
-- 创建分区表，并配置动态分区管理：
BEGIN;
CREATE TABLE tbl1 (
    c1 text NOT NULL,
    c2 int
)
PARTITION BY LIST (c2);
CALL set_table_property ('tbl1', 'auto_partitioning.enable', 'true');
CALL set_table_property ('tbl1', 'auto_partitioning.time_unit', 'DAY');
CALL set_table_property ('tbl1', 'auto_partitioning.time_zone', 'Asia/Shanghai');
CALL set_table_property ('tbl1', 'auto_partitioning.num_precreate', '3');
CALL set_table_property ('tbl1', 'auto_partitioning.num_retention', '2');
COMMIT;
```

  执行结果如下。

时间	事件	结果
2022-01-10 09:00:00	执行如上SQL，创建分区表。	系统创建分区父表：tbl1；系统创建分区子表：tbl1_20220110、tbl1_20220111、tbl1_20220112。
2022-01-11 00:00:00	系统自动创建分区子表。	系统创建分区子表：tbl1_20220113。
2022-01-12 00:00:00	系统自动创建分区子表。	系统创建分区子表：tbl1_20220114。
2022-01-13 00:00:00	系统自动创建分区子表，并清理分区子表。	系统创建分区子表：tbl1_20220115；系统清理分区子表：tbl1_20220110。
2022-01-14 00:00:00	系统自动创建分区子表，并清理分区子表。	系统创建分区子表：tbl1_20220116；系统清理分区子表：tbl1_20220111。

- 保留指定分区子表
  - 语法说明

    默认情况下，系统将按动态分区配置自动创建和删除分区子表，不在保留范围内的分区子表将被自动删除。但在一些特殊场景中，可能需要保留重要分区的数据（例如在电商场景中，需要保留历年双11数据进行同环比分析），改场景就可以使用保留指定分区子表的功能，通过给表增加keep_alive属性的方式避免需要保留的分区子表被自动清理。

```
-- 增加保留分区
call set_table_property('[<schema_name>.]<table_name>', 'keep_alive', 'true');
-- 删除保留分区：删除保留属性后，动态分区管理会立即触发过期数据清理
call set_table_property('[<schema_name>.]<table_name>', 'keep_alive', 'false');
```

    schema_name为需要保留的分区子表所在的Schema名称；table_name为需要保留的分区子表的表名称。

- 使用示例
  - 使用如下SQL将tbl1_20220111保留，使其不会被动态分区机制自动清理。

```
-- 增加保留分区
call set_table_property('pulbic.tbl1_20220111', 'keep_alive', 'true');
```

  - 使用如下SQL将tbl1_20220111解除保留，使其会被动态分区机制自动清理。

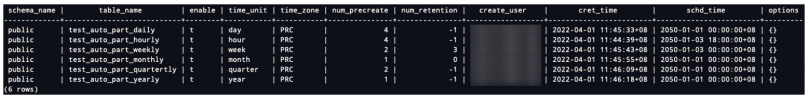
```
-- 删除保留分区：删除保留属性后，动态分区管理会立即触发过期数据清理
call set_table_property('pulbic.tbl1_20220111', 'keep_alive', 'false');
```

- 查看配置动态分区配置和调度情况

  可以使用如下SQL查询当前数据库中配置了分区表和对应的动态分区策略参数。

```
SELECT
    nsp_name AS schema_name,
    tbl_name AS table_name,
    ENABLE,
    time_unit,
    time_zone,
    num_precreate,
    num_retention,
    b.usename AS create_user,
    cret_time,
    schd_time,
    options
FROM
    hologres.hg_partitioning_config AS a
    LEFT JOIN pg_user AS b ON a.cret_user = b.usesysid;
```

  查询结果样例如下。



```
 schema_name |        table_name        | enable | time_unit | time_zone | num_precreate | num_retention |  create_user  |         cret_time          |         schd_time          | options
-------------+--------------------------+--------+-----------+-----------+---------------+---------------+---------------+----------------------------+----------------------------+---------
 public      | test_auto_part_daily     | t      | day       | PRC       |             4 |            -1 | [redacted]    | 2022-04-01 11:45:33+08     | 2050-01-01 00:00:00+08     | {}
 public      | test_auto_part_hourly    | t      | hour      | PRC       |             4 |            -1 | [redacted]    | 2022-04-01 11:44:39+08     | 2050-01-03 18:00:00+08     | {}
 public      | test_auto_part_weekly    | t      | week      | PRC       |             2 |             3 | [redacted]    | 2022-04-01 11:45:43+08     | 2050-01-03 00:00:00+08     | {}
 public      | test_auto_part_monthly   | t      | month     | PRC       |             1 |             0 | [redacted]    | 2022-04-01 11:45:55+08     | 2050-01-01 00:00:00+08     | {}
 public      | test_auto_part_quarterly | t      | quarter   | PRC       |             2 |            -1 | [redacted]    | 2022-04-01 11:46:09+08     | 2050-01-01 00:00:00+08     | {}
 public      | test_auto_part_yearly    | t      | year      | PRC       |             1 |            -1 | [redacted]    | 2022-04-01 11:46:18+08     | 2050-01-01 00:00:00+08     | {}
(6 rows)
```

  样例结果列的含义如下。

列名称	说明
schema_name	Schema名称。
table_name	表名称。
ENABLE	是否启用动态分区管理。
time_unit	动态分区的时间单位。
time_zone	动态分区时区设置。
num_precreate	预创建分区数量。
num_retention	保留历史分区数量。
create_user	创建的用户账号。

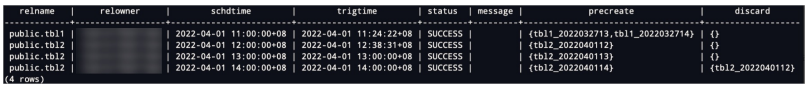
列名称	说明
cret_time	创建时间。
schd_time	最近一次计划调度时间。
options	其他。

- 查看创建和清除分区子表日志

  创建和清除分区子表的日志不会出现在Query Log中，您可以使用如下SQL查询创建和清除分区子表日志。

```
SELECT
    relname,
    relowner,
    schdtime,
    trigtime,
    status,
    message,
    precreate,
    discard
FROM
    hologres.hg_partitioning_log;
```

查询结果样例如下。



样例结果列的含义如下。

列名	说明
relname	`schema.table` 即 `{Schema名称}.{表名称}` 。
relowner	表的Owner。
schdtime	计划调度时间。
trigtime	实际触发时间。
status	状态。
message	备注。
precreate	创建的分区子表名。
discard	清理的分区子表名。

常见问题

- 对于存量分区表如何开启动态分区？

  对于存量分区表您可以使用如下SQL打开动态分区。

> **注意** 由于 `auto_partitioning.time_unit` 和 `auto_partitioning.time_zone` 为动态分区功能的核心配置，所以升级后仅允许设置一次，设置后不能再更改。

```
-- SQL样例
BEGIN;
CALL set_table_property('auto_part_old', 'auto_partitioning.enable', 'true');
CALL set_table_property('auto_part_old', 'auto_partitioning.time_unit', 'HOUR');
CALL set_table_property('auto_part_old', 'auto_partitioning.time_zone', 'PRC');
CALL set_table_property('auto_part_old', 'auto_partitioning.num_precreate', '4');
CALL set_table_property('auto_part_old', 'auto_partitioning.num_retention', '-1');
CALL set_table_property('auto_part_old', 'auto_partitioning.num_hot', '-1');
COMMIT;
```

- 对于存量表开启动态分区后，历史存在的分区子表是否会收到自动清理逻辑的影响？

  系统根据分区子表的名称进行分区子表清理，若分区子表的名称满足 `{parent_table}_{time_suffix}` 此类命名规则，则会被清理，如果不满足则不会被清理。

6.5.4.2. ALTER PARTITION TABLE

ALTER PARTITION TABLE语句用于修改分区表。本文为您介绍ALTER PARTITION TABLE的用法。

命令格式

Hologres支持以下3种修改分区表的操作。

```
ALTER TABLE [IF EXISTS] table_name RENAME to new_table_name;
ALTER TABLE [IF EXISTS] table_name ATTACH PARTITION new_partition_name FOR VALUES in (<stri
ng_literal>);
ALTER TABLE [IF EXISTS] table_name DETACH PARTITION paritition_name;
```

参数说明

修改分区表的示参数说明如下表所示。

参数	描述
`RENAME`	重命名分区表。
`ATTACH PARTITION new_partition_name FOR VALUES in (<string_literal>)`	用于将当前表绑定为目标表的分区。说明如下： • 绑定分区的规范必须与目标表的分区策略和分区键相对应。 • 需要绑定的表与目标表的列数量必须相同，并且列类型必须匹配。 • 需要绑定的表必须具有目标表的所有NOT NULL约束。 如果您添加了一个不接受NULL值的列表分区，除非它是一个表达式， 否则您需要添加NOT NULL约束至分区键列。

参数	描述
`DETACH PARTITION partition_name`	分离目标表的指定分区。 被分离的分区作为独立表继续存在，但不再与目标表相关联。

使用限制

- 分区子表在绑定父表时，其约束关系如下表所示。其中：
  - 与父表保持一致：即对应的属性分区子表必须和父表保持一致，若不一致，在分区子表绑定（ATTACH）父表时会报错，需要重新创建分区子表。
  - 不要求与父表一致：即对应的属性分区子表可以与父表不一致，如果子表没有显式指定属性，则会继承父表的属性，若是子表显式指定属性，则会保留子表的属性。
  - 索引列必须包含父表的索引列：即分区子表的索引列必须包含父表的索引列，还能显式再指定其他列。

分类	表属性	描述	create table partition of 时是否继承父表属性	ATTACH时与父表的约束关系	DETACH时是否继承父表属性
表属性	orientation	表存储格式。	继承	与父表保持一致。	继承
	table_group	Table Group属性包含Shard Count。	继承	与父表保持一致。	继承
	time_to_live_in_seconds	表数据生命周期。	继承	不要求与父表一致。 ◦ 子表属性未赋值，继承父表属性。 ◦ 子表属性已赋值，保留子表属性。	继承
索引	primary key	主键。	继承	与父表保持一致。	继承
	distribution key	分布键。	继承	与父表保持一致。	继承
	clustering_key（包括列和排序）	聚簇索引。	继承	与父表保持一致。	继承
	event_time_column	分段键。	继承	与父表保持一致。	继承
	bitmap_columns	比特编码。	继承	索引列必须包含父表的索引列。	继承

分类	表属性	描述	create table partition of 时是否继承父表属性	ATTACH时与父表的约束关系	DETACH时是否继承父表属性
	dictionary_encoding_columns	字段编码。	继承	索引列必须包含父表的索引列。	继承
	binlog_level	是否开启Binlog。	继承	与父表保持一致。	继承
	proxima_vectors	向量检索索引。	继承	索引列必须包含父表的索引列。	继承
列约束	nullable	非空约束。	继承	与父表保持一致。	继承
	default value	默认值。	继承	与父表保持一致。	继承

- Attach的表必须与目标表具有相同的列，而不能多或者少列。
- Attach时列的类型必须匹配。

使用示例

修改分区表的示例语句如下。

```
--修改分区表的名称
alter table holo_test rename to my_holo_test;
--添加my_table为holo_table的分区表
alter table holo_table attach partition my_table for values in ('2015');
--将holo_test从all_test分区表中解除绑定，分离为独立表
alter table all_test detach partition holo_test;
```

您也可以参见如下完整示例进行分区表替换。

```
-- 创建新的分区表
begin;
drop table if exists "table_20210101_new";
CREATE TABLE "table_20210101_new" (
  "colA" integer NOT NULL,
  "colB" text NOT NULL,
  "colC" numeric(38,10) NOT NULL,
  "ds" text NOT NULL,
  "process_time" timestamptz NOT NULL DEFAULT now()
);
call set_table_property('table_20210101_new', 'orientation','column');
call set_table_property('table_20210101_new', 'distribution_key','"colA"');
call set_table_property('table_20210101_new', 'event_time_column','process_time');
commit;
---导入数据
insert into "table_20210101_new" select * from ...;
---替换子表
begin;
--解除绑定子分区为独立的表
ALTER TABLE table_parent DETACH PARTITION table_20210101;
--重命名解绑定后独立的表
ALTER TABLE table_20210101 RENAME to table_20210101_backup;
--把临时表更名为分区表名
ALTER TABLE table_20210101_new RENAME to table_20210101;
--绑定新的分区表
ALTER TABLE table_parent ATTACH PARTITION table_20210101 FOR VALUES in ("20210101");
commit;
```

6.5.4.3. DROP PARTITION TABLE

DROP PARTITION TABLE语句用于删除分区表。本文为您介绍DROP PARTITION TABLE的用法。

使用限制

删除分区表父表时，默认同时删除分区表子表。

`PARTITION TABLE` 为需要删除的目标分区表。

语法

删除分区表的语法如下。

```
DROP TABLE [ IF EXISTS ] table_name [, ...];
```

> **说明** `DROP TABLE` 支持一次删除多个表。

参数说明如下表所示。

参数	描述

参数	描述
IF EXISTS	• 如果指定 `IF EXISTS` ，无论目标表是否存在，执行删除语句后系统都会返回成功。 • 如果不指定 `IF EXISTS` ，当目标表不存在时，系统会返回 `ERROR: table "non_exist_table" does not exist` 报错。
table_name	需要删除的表名称。

示例

删除分区表的示例SQL语句如下。

```
DROP TABLE hologres_child2;
```

6.5.5. FOREIGN TABLE

6.5.5.1. CREATE FOREIGN TABLE

CREATE FOREIGN TABLE语句用于创建外部表，当前支持创建MaxCompute、OSS、DLF、Hologres类型的外部表。本文为您介绍CREATE FOREIGN TABLE的用法。

语法

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type }
  [, ... ]
  ] )
SERVER odps_server
[ OPTIONS ( option 'value' [, ... ] ) ]
```

参数说明如下表所示。

> **说明** 下表参数说明为创建MaxCompute类型外部表时所需要配置的参数，创建OSS、DLF、Hologres类型的外部表详情请参见创建外部表直接访问OSS数据、创建DLF外部表、创建Hologres外部表。

参数	描述
SERVER	连接外部数据源的服务器。 您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。

参数	描述
OPTIONS	您需要指定project_name和table_name。 • project_name为MaxCompute的项目名称。 • table_name为需要查询的MaxCompute表名称。

> **说明** Hologres创建的外部表的字段类型必须与MaxCompute的字段类型一一对应。

示例

- 直接查询外部表数据。

  Hologres新建外部表后，您可以直接查询MaxCompute外部表数据。示例SQL语句如下。

```
CREATE FOREIGN TABLE src_pt(
  id text,
  pt text)
SERVER odps_server
OPTIONS(project_name '<odps_project>', table_name '<odps_table>');
SELECT * FROM src_pt;
```

  在Hologres中直接查询MaxCompute外部表数据，详情请参见通过创建外部表加速查询MaxCompute数据。

- 导入外部表并查询数据。

  您可以导入MaxCompute的数据至Hologres，再进行查询。示例语句如下。

```
CREATE FOREIGN TABLE src_pt_odps(
  id text,
  pt text)
SERVER odps_server
OPTIONS (project_name'<odps_project>', table_name'<odps_table>');
BEGIN;
CREATE TABLE src_pt(
 id text,
  pt text);
COMMIT;
INSERT INTO src_pt SELECT * FROM src_pt_odps;
```

  导入MaxCompute数据至Hologres后，再进行查询，详情请参见使用SQL导入MaxCompute的数据至Hologres。

MaxCompute与Hologres的数据类型映射

创建MaxCompute外部表时，MaxCompute与Hologres的数据类型映射请参见MaxCompute与Hologres的数据类型映射。

> 说明
> - `DATETIME` 使用东八区时间作为系统的标准时间。取值范围为*0000年1月1日~ 9999年12月31日*，精确到毫秒。
> - `TIMESTAMPTZ` 包含时区，取值范围为*4713BC~ 294276AD*，精确到微秒。
> - 当MaxCompute数据表中含有Hologres不支持的类型字段时，如果Hologres不访问该字段，则可以正常查询所支持的类型字段。

6.5.5.2. IMPORT FOREIGN SCHEMA

IMPORT FOREIGN SCHEMA语句用于批量创建外部表。本文为您介绍IMPORT FOREIGN SCHEMA语句的用法和使用限制。

使用限制

- 使用 `IMPORT FOREIGN SCHEMA` 语句时，建议您添加 `LIMIT TO` 限制，并使用括号将需要添加限制的表名称括起来。如果不添加该限制，系统则将目标MaxCompute工作空间中的所有表批量创建至Hologres中。
- 仅Hologres V1.1.26及以上版本支持对使用 `IMPORT FOREIGN SCHEMA` 创建的外部表名称增加前缀和后缀，如果您的实例是V1.1.26以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。

命令格式

在Hologres中批量创建外部表的命令格式如下。

```
IMPORT FOREIGN SCHEMA remote_schema
    [ { LIMIT TO | EXCEPT } ( table_name [, ...] ) ]
    FROM SERVER odps_server
    INTO local_schema
    [ OPTIONS ( option 'value' [, ... ] ) ]
```

参数说明

参数说明如下表所示。

参数	描述
remote_schema	需要导入的MaxCompute表所在的项目名称。
table_name	需要导入的MaxCompute表名称。
server_name	MaxCompute表所在的外部服务器名称，默认为**odps_server**。 您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
local_schema	Hologres外部表所在的schema名（如public）。

参数	描述

options	Hologres支持如下四个option：
	• *if_table_exist*：表示导入时已经存在该表。取值如下：
	  ○ *error*：默认值，表示已有同名外部表，不再重复创建。
	  ○ *ignore*：忽略该同名表，跳过该表的导入，使导入的表不重复。
	  ○ *update*：更新并重新导入该表。
	• *if_unsupported_type*：表示导入的外部表中存在Hologres不支持的数据类型。取值如下：
	  ○ *error*：报错，导入失败， 并提示哪些表存在不支持的类型。
	  ○ *skip*：默认值，表示跳过导入的存在不支持类型的表，并提示哪些表被跳过。
	• *prefix*：表示导入时生成的Hologres外部表的前缀，自Hologres V1.1.26版本新增。
	• *suffix*：表示导入时生成的Hologres外部表的后缀，自Hologres V1.1.26版本新增。

说明 Hologres仅支持创建MaxCompute外部表。新建的外部表名称需要同MaxCompute表的名称一致。

使用示例

示例选取MaxCompute公共数据集**public_data**中的表，在Hologres中批量创建外部表。您可以参照使用公开数据集描述，登录并查询数据集 。

- 示例1：为public schema新建一张外部表，若表存在则更新表。

```
IMPORT FOREIGN SCHEMA public_data LIMIT to
(customer)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'update');
```

- 示例2：为public schema批量新建外部表。

```
 IMPORT FOREIGN SCHEMA public_data LIMIT to(
  customer,
  customer_address,
  customer_demographics,
  inventory,item,
  date_dim,
  warehouse)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'update');
```

- 示例3：新建一个testdemo schema并批量新建外部表。

```
create schema testdemo;
IMPORT FOREIGN SCHEMA public_data LIMIT to(
  customer,
  customer_address,
  customer_demographics,
  inventory,item,
  date_dim,
  warehouse)
  FROM server odps_server INTO testdemo options(if_table_exist 'update');
set search_path to testdemo;
```

- 示例4：在public schema批创建外部表，已有外表则报错。

```
IMPORT FOREIGN SCHEMA public_data LIMIT to
(customer,
  customer_address)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'error');
```

- 示例5：在public schema批量创建外部表，已有外表则跳过该外部表。

```
IMPORT FOREIGN SCHEMA public_data LIMIT to
(customer,
  customer_address)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'ignore');
```

HoloWeb可视化批量创建外部表

HoloWeb提供可视化批量创建外部表功能，无需写SQL命令就能创建外部表，步骤如下。

1. 进入**HoloWeb**页面，详情请参见连接HoloWeb。
2. 在HoloWeb开发页面的顶部菜单栏，选择**元数据管理 > MaxCompute加速**，单击**批量创建外部表**。

   您也可以在**元数据管理**界面的**已登录实例**列表。单击目标数据库，鼠标右击数据库下已创建的目标模式，选择**批量创建外部表**。

3. 在**批量创建外部表**页面，配置各项参数。

类别	参数	描述
基本属性	实例名	已登录的实例名称。
	数据库	存放新创建外部表的Hologres数据库名称。
目标位置	模式	模式名称。 您可以选择默认创建的**public**模式，也可以选择新建的模式名称。
来源	类型	目前仅支持MaxCompute外部表。
	服务器列表	您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
	远程库	MaxCompute的项目名称。
	选择要直接加速的表	○ **整库加速**：批量创建所选项目下的所有表。 ○ **部分加速**：您可以通过搜索表名称或关键字，选择需要创建的表。 **说明** **部分加速**最多支持显示200张表，超出部分将不显示，但是也会创建成功。 例如，目标项目中共有203个名称中包含*test*的表，搜索*test*时，此处最多只会显示200个相关的表，但实际上203个表都会创建成功。

类别	参数	描述
高级选项	表名冲突	○ **忽略，继续创建其他表**：创建表时，如果数据库中已存在当前创建的表名称，则忽略当前创建的表，继续创建其他表。 ○ **更新，修改同名表**：创建表时，如果数据库中已存在当前创建的表名称，则更新已有表的数据。 ○ **报错，不再重复创建**：创建表时，如果数据库中已存在当前创建的表名称，则发送报错，不再重复创建。
	数据类型不支持	○ **报错，导入失败**：如果创建表时存在不支持的数据类型，则产生报错，导入数据失败。 ○ **忽略，跳过不支持字段**：如果创建表时存在不支持的数据类型，则忽略不支持的字段，继续导入数据。

4. 单击**运行**，批量创建外部表。

数据类型映射

批量创建的MaxCompute外部表与Hologres的数据类型映射，详情请参见批量创建MaxCompute外部表与Hologres的数据类型映射。

6.5.5.3. ALTER FOREIGN TABLE

ALTER FOREIGN TABLE语句用于修改外部表。本文为您介绍如何使用ALTER FOREIGN TABLE，为外部表重命名、增加列及删除列。

使用限制

Hologres仅支持为外部表重命名、增加列及删除列等修改外部表操作。

重命名

语法如下。

```
ALTER FOREIGN TABLE [ IF EXISTS ] name RENAME TO new_name;
```

示例SQL语句如下。

```
ALTER FOREIGN TABLE test RENAME TO new_test_table;
```

增加列

使用Hologres创建外部表查询数据时，当MaxCompute表增加字段后，Hologres不会自动更新Schema，需要您在Hologres中手动增加列。

> 说明
> - 大集群暂不支持使用如下语法增加列，关于大集群增加列的操作请参见IMPORT FOREIGN SCHEMA。
> - Hologres V0.10、V1.1.59及其以上版本支持通过 `ADD COLUMN` 的方式增加多列。

语法如下。

```
ALTER FOREIGN TABLE IF EXISTS table_name ADD COLUMN new_column_name data_type;
```

示例SQL语句如下。

```
ALTER FOREIGN TABLE bank
 ADD COLUMN  cons_conf_idx float8,
 ADD COLUMN  euribor3m float8;
```

删除列

语法如下。

```
ALTER FOREIGN TABLE IF EXISTS table_name DROP COLUMN column_name;
```

示例SQL语句如下。

```
ALTER FOREIGN TABLE bank
 DROP COLUMN  cons_conf_idx,
 DROP COLUMN  euribor3m;
```

6.5.5.4. DROP FOREIGN TABLE

DROP FOREIGN TABLE语句用于删除外部表。本文为您介绍DROP FOREIGN TABLE的用法。

语法

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明如下表所示。

参数	描述
IF EXISTS	如果指定的表不存在，则系统提示一个Notice，而不会提示一个报错。
name	需要删除的表名称。
CASCADE	删除表时，级联删除依赖于表的对象，例如视图。
RESTRICT	如果目标表存在依赖对象，则系统拒绝删除该表。

示例

删除外部表，示例SQL语句如下。

```
DROP FOREIGN TABLE holo_test;
```

6.5.6. CAST

6.5.6.1. CREATE CAST

CREATE CAST用于定义数据类型之间的转换。本文为您介绍CREATE CAST的用法。

语法

```
CREATE CAST (source_type AS target_type)
    WITH INOUT
    [ AS ASSIGNMENT | AS IMPLICIT ]
```

参数说明如下表所示。

参数	描述
source_type	该转换的源数据类型。
target_type	该转换的目标数据类型。
WITH INOUT	表示该转换为I/O转换。 该转换通过调用源数据类型的输出函数来执行，并将结果传给目标数据类型的输入函数。
AS ASSIGNMENT	表示该转换可以在赋值模式下隐含调用。
AS IMPLICIT	表示该转换可以在任意环境里隐含调用。

示例

通常，在Filter中比较字符类型和数值类型时，默认不进行转换。您可以使用`CREATE CAST`语句，创建一个支持比较字符类型和数值类型的CAST，示例语句如下。

```
CREATE TABLE test_cast(id text);
INSERT INTO test_cast VALUES(888);
SELECT * FROM test_cast aa WHERE id > 888;
CREATE CAST (text AS integer) WITH INOUT AS IMPLICIT;
SELECT * FROM test_cast aa WHERE id > 888;
```

更多关于`CREATE CAST`的详情，请参见PostgreSQL CREATE CAST。

6.5.6.2. DROP CAST

DROP CAST语句用于删除已定义的类型转换。本文为您介绍DROP CAST的用法。

使用限制

您必须拥有源或目标数据类型的权限，才可以删除类型转换。

语法

```
DROP CAST [ IF EXISTS ] (source_type AS target_type)
```

参数说明如下表所示。

参数	描述
IF EXISTS	如果指定的转换不存在，则系统提示一个Notice，而不会提示一个报错。
source_type	类型转换的源数据类型。
target_type	类型转换的目标数据类型。

示例

删除类型转换，示例语句如下。

```
DROP CAST IF EXISTS (text AS timestamptz);
DROP CAST IF EXISTS (text AS integer);
```

更多关于 `DROP CAST` 的详情，请参见PostgreSQL DROP CAST。

6.5.7. VIEW

6.5.7.1. VIEW

Hologres支持基于内部表及外部表创建视图VIEW，您可以使用单张表、多张表或者其他视图来创建视图。本文为您介绍在Hologres中如何创建视图。

注意事项

在创建和查看视图时，您需要注意如下事项：

- 如果数据库的权限模型为SLPM，创建一个View引用了跨Schema的两个或多个表时，由于Schema间权限不互通的原因，此View将无法被访问。因此不建议在SLPM管理的数据库中创建跨Schema的View。
- 如果视图中数据来自于单张表时，修改视图中的数据，表数据会随之更新。修改表数据时，对应视图也会更新。在使用单表视图时，建议您谨慎修改视图数据，以避免对应的表数据被修改影响业务。
- 如果视图中的数据来源于多张表时，不支持修改视图数据。

创建视图

创建视图的语法格式如下。

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

创建内部表视图

1. 创建内部表，示例语句如下。

```
create table holo_test (
  amount decimal(10, 2),
  rate decimal(10, 2)
);
insert into holo_test values
(12.12,13.13),
(14.14,15.15),
(16.16,17.17),
(17.1,17),
(18.01,19);
```

2. 基于内部表创建视图并查询表数据，示例语句如下。

```
create view holo_view as select * from holo_test;
select * from holo_view;
 amount | rate
--------+-------
  12.12 | 13.13
  14.14 | 15.15
  16.16 | 17.17
  17.10 | 17.00
  18.01 | 19.00
(5 rows)
```

创建外部表视图

1. 创建外部表，示例语句如下。

```
create foreign table if not exists holo_foreign_test (
  amount decimal(10, 2),
  rate decimal(10, 2))
  server odps_server
  options(project_name '<projectname>', table_name '<odps_name>')
  );
select * from holo_foreign_test limit 2;
```

2. 基于外部表创建视图并查询表数据，示例语句如下。

```
create view foreign_view as select * from holo_foreign_test;
select * from foreign_view limit 2;
 amount | rate
--------+-------
  12.12 | 13.13
  14.14 | 15.15
```

创建内部表及外部表的联合视图

1. 基于内部表和外部表创建联合视图并查询表数据，示例语句如下。

```
create view view1 as select * from holo_view union all  select * from foreign_view;
select * from view1;
 amount | rate
--------+-------
  12.12 | 13.13
  14.14 | 15.15
  16.16 | 17.17
   17.1 |    17
  18.01 |    19
  12.12 | 13.13
  14.14 | 15.15
  16.16 | 17.17
  17.10 | 17.00
  18.01 | 19.00
  12.12 | 13.13
  14.14 | 15.15
  16.16 | 17.17
   17.1 |    17
  18.01 |    19
  12.12 | 13.13
  14.14 | 15.15
  16.16 | 17.17
   17.1 |    17
  18.01 |    19
(20 rows)
```

删除视图

删除视图的语法格式如下。

```
drop view <view_name>;
```

查看所有视图和视图的DDL

您可以执行如下命令查看所有视图，如果您通过psql客户端查看所有视图，也可以执行 `\dv` 进行查看。

```
--sql命令
SELECT n.nspname as "Schema",
  c.relname as "Name",
  CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized vie
w' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'fo
reign table' WHEN 'p' THEN 'table' WHEN 'I' THEN 'index' END as "Type",
  pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('v','')
      AND n.nspname <> 'pg_catalog'
      AND n.nspname <> 'information_schema'
      AND n.nspname !~ '^pg_toast'
  AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
```

您可以执行如下命令查看VIEW的具体DDL:

```
create extension hg_toolkit;
select hg_dump_script('viewname');
```

6.6. DML&DQL

6.6.1. SELECT

本文将会为您介绍在交互式分析（Hologres） 中数据查询SELECT的用法。

命令介绍

SELECT：从零或更多表中以多种形式的数据查询。主要包含的参数如下表所示：

参数列表	
WITH列表	SELECT列表
FROM子句	WHERE子句
GROUP BY子句	CUBE子句
DISTINCT 子句	COUNT DISTINCT子句
UNION子句	INTERSECT子句
EXCEPT子句	ORDER BY子句
LIMIT子句	无

命令格式

SELECT语句的基本语法具体如下。

```
[ WITH with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
  * | expression [[AS] output_name] [, ...]
  [FROM from_item [, ...]]
  [WHERE condition]
  [GROUP BY grouping_element [, ...]]
  [HAVING condition [, ...]]
  [{UNION | INTERSECT | EXCEPT} [ALL] select]
  [ORDER BY expression [ASC | DESC | USING operator] [, ...]]
  [LIMIT {count | ALL}]
```

其中，`grouping_element` 和 `from_item` 包含的内容如下：

- grouping_element包含 `expression` `expression` 可能是列名、常数、函数或者是列名或常数的任意组合，以及以算术运算符或逐位运算符连接的函数。
- from_item包含

```
table_name [[AS] alias [( column_alias [, ...] )]]
(select) [AS] alias [( column_alias [, ...] )]
from_item [NATURAL] join_type from_item
          [ON join_condition | USING ( join_column [, ...] )]
```

命令用法

SELECT的常用方法如下：

1. FROM列表中的所有元素都会被计算（ FROM中的每一个元素都是一个真实表或者虚拟表）。如果在FROM列表中指定了多于一个元素，得到的结果做并集。
2. 如果指定了WHERE子句，所有不满足该条件的行都会被从输出中消除。
3. 如果指定了 `GROUP BY` 子句或者如果有聚集函数，输出会被组合成由在一个或者多个值上匹配的行构成的分组，并且在其上计算聚集函数的结果。如果出现了HAVING子句，它会消除不满足给定条件的分组。
4. 对于每一个被选中的行或者行组，会使用SELECT输出表达式计算实际的输出行。
5. 通过使用操作符UNION、INTERSECT和EXCEPT，多于一个SELECT语句的输出可以被整合形成一个结果集。UNION操作符返回位于一个或者两个结果集中的全部行。INTERSECT操作符返回同时位于两个结果集中的所有行。EXCEPT操作符返回位于第一个结果集但不在第二个结果集中的行。在所有三种情况下，重复行都会被消除（除非指定ALL）。可以增加DISTINCT来显式的消除重复行。注意虽然ALL是SELECT自身的默认行为，但这里DISTINCT是默认行为。
6. 如果指定了 `ORDER BY` 子句，被返回的行会以指定的顺序排序。如果没有给定 `ORDER BY` ，系统会以能最快产生行的顺序返回它们。
7. 如果指定了LIMIT（或FETCH FIRST） 或者OFFSET子句，SELECT语句只返回结果行的一个子集。

参数说明

- **WITH列表**

- 命令简介

  WITH列表是位于SELECT之前或者作为SELECT子句的subquery存在，通常位于SELECT之前，用于定义子查询，并为子查询声明一个名字和返回的列名，定义每一个WITH子句为一个CTE（Common Table Expression），定义语法如下：

  ```
  with query_name [ ( column_name [, ...] ) ] AS ( select )
  ```

- 参数说明

参数	说明
query_name	指定当前CTE的名字，可以是任意有效的标识符。
column_name	列表对应着子查询返回值的列名，类似于SELECT子句中的AS的语义，子查询可以是一个常规的SELECT查询。

  CTE之间通过逗号分隔，后面出现的CTE定义可以引用前面定义的CTE，但是目前暂时不支持递归的CTE调用，在之后的查询中，可以直接将 `query_name` 作为一个视图（view）出现在查询中，如果没有指定 `column_name` 列表，该view的 `column_name` 为对应子查询返回列的列名，如果指定 `column_name` 列表，则需要使用定义 `column_name` ，并且 `column_name` 列表需要和SELECT返回值列表个数相同。

- **SELECT列表**
  - 命令简介

    SELECT列表（位于关键词 SELECT和FROM之间）指定构成SELECT语句输出行的表达式。这些表达式可以（并且通常确实会）引用FROM子句中计算得到的列。

  - 参数说明

    SELECT的每一个输出列都有一个名称。 在一个简单的SELECT中，该名称只用来标记要显示的列，当SELECT为一个大型查询的子查询时，大型查询会把该名称看做子查询产生的虚表的列名。要指定用于输出列的名称，在该列的表达式后面添加 `AS output_name` （可以省略AS，但只能在期望的输出名称不匹配任何PostgreSQL关键词时省略。为了避免和未来增加的关键词冲突， 推荐您添加AS或者用双引号引用输出名称）。如果不指定列名，PostgreSQL会自动选择一个名称。如果列的表达式是一个简单的列引用，那么被选择的名称就和该列的名称相同。在使用函数或者类型名称的更复杂的情况中，系统可能会生成诸如 `?column?` 之类的名称。

    一个输出列的名称可以被用来在 `ORDER BY` 以及 `GROUP BY` 子句中引用该列的值，但是不能用于WHERE和HAVING子句（在其中必须写出表达式）。

    可以在输出列表中写*来取代表达式，它是被选中行的所有列的一种简写方式。还可以写table_name*，它是只来自那个表的所有列的简写形式。在这些情况中无法用AS指定新的名称，输出行的名称将和表列的名称相同。

- **FROM子句**
  - 命令简介

    FROM子句为SELECT 指定一个或者更多源表。如果指定了多个源表，结果将是所有源表的笛卡尔积（交叉连接）。但是通常会增加限定条件（通过 WHERE）来把返回的行限制为该笛卡尔积的一个小子集。

  - 参数说明

    FROM子句可以包含下列元素：

元素	说明
table_name	一个现有表或视图的名称（可以限定表的schma模式）。
alias	一个包含别名的FROM项的替代名称。别名被用于让书写简洁或者消除自连接中的混淆（其中同一个表会被扫描多次）。当提供一个别名时，表或者函数的实际名称会被隐藏。例如，给定 `FROM foo AS f` ，SELECT的剩余部分就必须以f而不是foo来引用这个FROM项。
select	一个子SELECT可以出现在 FROM子句中。这就好像把它的输出创建为一个存在于该SELECT命令期间的临时表。注意子-SELECT必须用圆括号包围，并且必须为它提供一个别名。
function_name	函数调用可以出现在FROM子句中（对于返回结果集合的函数特别有用，但是可以使用任何函数）。这就好像把该函数的输出创建为一个存在于该SELECT命令期间的临时表。 可以用和表一样的方式提供一个别名。如果写了一个别名，还可以写一个列别名列表来为该函数的组合返回类型的一个或者多个属性提供替代名称， 包括由ORDINALITY（如果有）增加的新列。 通过把多个函数调用包围在 `ROWS FROM（。）` 中可以把它们整合在单个FROM-子句项中。这样一个项的输出是把每一个函数的第一行串接起来，然后是每个函数的第二行，以此类推。如果有些函数产生的行比其他函数少，则在缺失数据的地方放上 NULL，这样被返回的总行数总是和产生最多行的函数一样。

元素	说明
join_type	包含如下5种类型： ■ [INNER] JOIN 对于INNER和OUTER连接类型，必须指定一个连接条件，即 `NATURAL`、`ON join_condition`或者 `USING （join_column [, ...]）` 之一（只能有一种）。其含义见下文。对于 `CROSS JOIN` ，上述子句不能出现。一个JOIN子句联合两个FROM项（为了方便我们称之为“表 “，尽管实际上它们可以是任何类型的FROM项）。如有必要可以使用圆括号确定嵌套的顺序。 在没有圆括号时，JOIN会从左至右嵌套。在任何情况下，JOIN的联合比用逗号分隔FROM-列表项更强。 ■ LEFT [OUTER] JOIN `LEFT OUTER JOIN` 返回被限制过的笛卡尔积中的所有行（即所有通过了其连接条件的组合行），以及左表中每行的一个副本，因为没有右行通过了连接条件。通过在右手列中插入空值，这种左手行会被扩展为连接表的完整行。注意在决定哪些行匹配时，只考虑JOIN子句自身的条件。之后才应用外条件。 ■ RIGHT [OUTER] JOIN `RIGHT OUTER JOIN` 返回所有连接行，外加每一个没有匹配上的右手行（在左端用空值扩展）。这只是为了记号上的方便，因为可以通过交换左右表把它转换成一个 `LEFT OUTER JOIN` 。 ■ FULL [OUTER] JOIN `FULL OUTER JOIN` 返回所有连接行，外加每一个没有匹配上的左手行（在右端用空值扩展），再外加每一个没有匹配上的右手行（在左端用空值扩展） ■ CROSS JOIN `CROSS JOIN` 和 `INNER JOIN` 会产生简单的笛卡尔积，也就是与在FROM列出两个表得到的结果相同，但是要用连接条件（如果有）约束该结果。 `CROSS JOIN` 与 `INNER JOIN ON （TRUE）` 等效，也就是说条件不会移除任何行。这些连接类型只是一种记号上的方便，因为没有什么是用纯粹的FROM和 WHERE能做而它们不能做的。
join_condition	`join_condition` 是一个会得到boolean类型值的表达式（类似于一个WHERE子句），它说明一次连接中哪些行被认为相匹配。
USING (a, b, ...)	形式 `USING （a， b...）` 的子句是 `ON left_table.a = right_table.a AND left_table.b = right_table.b ...` 的简写。还有， USING表示每一对相等列中只有一个会被包括在连接输出中。
NATURAL	列出在两个表中所有具有相同名称的列的USING的简写。

- **WHERE子句**
  - 命令简介

    可选的WHERE子句的形式：

```
WHERE condition
```

- 参数说明

参数	说明
condition	condition是任一计算得到布尔类型结果的表达式。任何不满足这个条件的行都会从输出中被消除。如果用一行的实际值替换其中的变量引用后，该表达式返回真，则该行符合条件。

- **GROUP BY子句**
  - 命令简介

    `GROUP BY` 将会把所有被选择的行中共享相同分组表达式值的那些行压缩成一个行。可选的 `GROUP BY` 子句的形式：

    ```
    GROUP BY grouping_element [, ...]
    ```

  - 参数说明

    `grouping_element` 中使用的expression可以是输入列名、输出列（SELECT列表项） 的名称或序号或者由输入列值构成的任意表达式。在出现歧义时， `GROUP BY` 名称将被解释为输入列名而不是输出列名。

    如果任何 `GROUPING SETS`、`ROLLUP` 或者CUBE作为分组元素存在，则 `GROUP BY` 子句整体上定义了数个独立的分组集。其效果等效于在具有独立分组集作为它们的 `GROUP BY` 子句的子查询间构建一个UNION ALL。

    聚集函数（如果使用）会在组成每一个分组的所有行上进行计算，从而为每一个分组产生一个单独的值（如果有聚集函数但是没有 `GROUP BY` 子句，则查询会被当成是由所有选中行构成的一个单一分组）。传递给每一个聚集函数的行集合可以通过在聚集函数调用附加一个FILTER子句来进一步过滤。当存在一个FILTER子句时，只有那些匹配它的行才会被包括在该聚集函数的输入中。

    当存在 `GROUP BY` 子句或者任何聚集函数时， SELECT列表表达式不能引用非分组列（除非它出现在聚集函数中或者它函数依赖于分组列），因为这样做会导致返回非分组列的值时会有多种可能的值。如果分组列是包含非分组列的表的主键（ 或者主键的子集），则存在函数依赖。

    所有的聚集函数都是在HAVING子句或者 SELECT列表中的任何“标量“表达式之前被计算。 这意味着一个CASE表达式不能被用来跳过一个聚集表达式的计算。

- **CUBE子句**
  - CUBE

    CUBE是自动对GROUP BY子句中列出的字段进行分组汇总，结果集将包含维度列中各值的所有可能组合，以及与这些维度值组合相匹配的基础行中的聚合值。它会为每个分组返回一行汇总信息，用户可以使用CUBE来产生交叉表值。例如，在CUBE子句中给出三个表达式 `(n = 3)` ，运算结果为$2^n = 2^3 = 8$组。 以n个表达式的值分组的行称为常规行，其余的行称为超级聚集行。具体表达式如下：

    ```
    CUBE ( { expression | ( expression [, …] ) } [, …] )
    ```

  - ROLLUP

    ROLLUP 函数是聚合函数，它是GROUP BY语句的简单扩展。在数据统计和报表生成过程中，它可以为每个分组返回一个小计，同时为所有分组返回总计。具体表达式如下：

    ```
    ROLLUP ( { expression | ( expression [, …] ) } [, …] )
    ```

- GROUPING SETS

  GROUPING SETS子句是GROUP BY子句的进一步扩展，它可以使用户指定多个GROUP BY选项。 这样做可以通过裁剪用户不需要的数据组来提高效率。 当用户指定了所需的数据组时，数据库不需要执行完整CUBE或ROLLUP生成的聚合集合。具体表达式如下：

  ```
  GROUPING SETS ( grouping_element [, …] )
  ```

- **DISTINCT子句**

  如果指定了SELECT DISTINCT，所有重复的行会被从结果集中移除（为每一组重复的行保留一行）。

  ```
  SELECT DISTINCT accountid FROM table;
  ```

- **COUNT DISTINCT子句**
  - 命令简介

    `COUNT DISTINCT` 支持计算去重之后的某一个column的个数，对于该列中出现多次的值只会被计算一次，和COUNT的计算类似，如果该列包含NULL值，它将不会计算在内。

  - 语法说明
    - 精确计算的语法示例如下：

      ```
      SELECT c1, COUNT(DISTINCT c2) FROM table GROUP BY c1
      ```

    - 由于精确计算的 `COUNT DISTINCT` 需要消耗较大的资源，因此交互式分析还支持非精确的 `COUNT DISTINCT` 计算，语法示例如下：

      ```
      SELECT c1, approx_count_distinct(c2) FROM table GROUP BY c1
      ```

- **UNION子句**
  - 命令简介

    UNION子句语法如下：

    ```
    select_statement UNION [ ALL | DISTINCT ] select_statement
    ```

- 参数说明

参数	说明
select_statement	`select_statement` 是任何没有 `ORDER BY`、`LIMIT`、 `FOR NO KEY UPDATE`、`FOR UPDATE`、 `FOR SHARE` 和 `FOR KEY SHARE` 子句的SELECT语句（如果子表达式被包围在圆括号内， `ORDER BY` 和LIMIT可以被附着到其上。如果没有圆括号，这些子句将被应用到UNION的结果而不是右手边的表达式上）。
UNION	UNION操作符计算所涉及的SELECT语句所返回的行的并集。如果一至少出现在两个结果集中的一个内，它就会在并集中。作为UNION两个操作数的SELECT语句必须产生相同数量的列并且对应位置上的列必须具有兼容的数据类型。 UNION的结果不会包含重复行，除非指定了ALL选项。ALL会阻止消除重复（因此，UNION ALL通常显著地快于UNION，尽量使用ALL）。可以写DISTINCT来显式地指定消除重复行的行为。 除非用圆括号指定计算顺序，同一个SELECT语句中的多个UNION操作符会从左至右计算。

- **INTERSECT子句**
  - 命令简介

    INTERSECT子句具有下面的形式：

    ```
    select_statement INTERSECT [ ALL | DISTINCT ] select_statement
    ```

  - 参数说明

参数	说明
select_statement	`select_statement` 是任何没有 `ORDER BY LIMIT` 子句的SELECT语句。
INTERSECT	INTERSECT操作符计算所涉及的 SELECT语句返回的行的交集。如果一行同时出现在两个结果集中，它就在交集中。 INTERSECT的结果不会包含重复行，除非指定了 ALL选项。如果有ALL，一个在左表中有m次重复并且在右表中有n 次重复的行将会在结果中出现min（m，n）次。DISTINCT可以写DISTINCT来显式地指定消除重复行的行为。 除非用圆括号指定计算顺序， 同一个SELECT语句中的多个 INTERSECT操作符会从左至右计算。 INTERSECT的优先级比UNION更高。也即， `A UNION B INTERSECT C` 将被读成 `A UNION (B INTERSECT C)` 。

- **EXCEPT子句**
  - 命令简介

    EXCEPT子句具有下面的形式：

    ```
    select_statement EXCEPT [ ALL | DISTINCT ] select_statement
    ```

- 参数说明

参数	说明
select_statement	`select_statement` 是任何没有 `ORDER BY LIMIT` 子句的SELECT语句。
EXCEPT	EXCEPT操作符计算位于左SELECT语句的结果中但不在右SELECT语句结果中的行集合。 EXCEPT的结果不会包含重复行，除非指定了 ALL选项。如果有ALL，一个在左表中有 m次重复并且在右表中有n次重复的行将会在结果集中出现max（m-n，0） 次。DISTINCT可以写DISTINCT来显式地指定消除重复行的行为。 除非用圆括号指定计算顺序， 同一个SELECT语句中的多个 EXCEPT操作符会从左至右计算。 EXCEPT的优先级与 UNION相同。 当前， `FOR NO KEY UPDATE`、`FOR UPDATE`、 `FOR SHARE` 和 `FOR KEY SHARE` 不能用于EXCEPT结果或者 EXCEPT的任何输入。

- **ORDER BY子句**
  - 命令简介

    可选的 `ORDER BY` 子句的形式如下：

```
ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...]
```

- 参数说明

  `ORDER BY` 子句导致结果行被按照指定的表达式排序。 如果两行按照最左边的表达式是相等的，则会根据下一个表达式比较它们， 以此类推。如果按照所有指定的表达式它们都是相等的，则它们被返回的顺序取决于实现。

  每一个expression 可以是输出列（SELECT列表项）的名称或者序号，它也可以是由输入列值构成的任意表达式。

  序号指的是输出列的顺序（从左至右）位置。这种特性可以为不具有唯一名称的列定义一个顺序。但这不是必要操作，因为可以使用 AS子句为输出列赋予一个名称。

  也可以在ORDER BY子句中使用任意表达式，包括没有出现在SELECT输出列表中的列。因此， 下面的语句是合法的：

  ```
  SELECT name FROM distributors ORDER BY code;
  ```

  这种特性的一个限制是一个应用在UNION、 INTERSECT或EXCEPT子句结果上的 `ORDER BY` 只能指定输出列名称或序号，但不能指定表达式。

  如果一个ORDER BY表达式是一个既匹配输出列名称又匹配输入列名称的简单名称，ORDER BY将把它解读成输出列名称。这与在同样情况下GROUP BY会做出的选择相反。这种不一致是为了与SQL标准兼容。

  可以为ORDER BY子句中的任何表达式之后增加关键词 ASC（上升）或者DESC（下降）。如果没有指定， ASC被假定为默认值。或者，可以在USING 子句中指定一个特定的排序操作符名称。一个排序操作符必须是某个 B-树操作符族的小于或者大于成员。ASC通常等价于 USING <而DESC通常等价于 USING >（但是一种用户定义数据类型的创建者可以准确地定义默认排序顺序是什么，并且它可能会对应于其他名称的操作符）。

  如果指定 `NULLS LAST` ，空值会排在非空值之后；如果指定 `NULLS FIRST` ，空值会排在非空值之前。如果都没有指定， 在指定或者隐含ASC时的默认行为是 `NULLS LAST` ， 而指定或者隐含DESC时的默认行为是 `NULLS FIRST` （因此，默认行为是空值大于非空值）。 当指定USING时，默认的空值顺序取决于该操作符是否为小于或者大于操作符。

  注意顺序选项只应用到它们所跟随的表达式上。例如 `ORDER BY x， y DESC` 和 `ORDER BY x DESC，y DESC` 是不同的。

  字符串数据会被根据引用到被排序列上的排序规则排序。根据需要可以通过在 expression中包括一个COLLATE子句来覆盖，例如 `ORDER BY mycolumn COLLATE "en_US"` 。

- **LIMIT 子句**
  - 命令简介

    LIMIT子句由两个独立的子句构成：

    ```
    LIMIT { count | ALL }
    OFFSET start
    ```

- 参数说明

  count指定要返回的最大行数，而start指定在返回行之前要跳过的行数。在两者都被指定时，在开始计算要返回的count行之前会跳过 start行。

  如果count表达式计算为NULL，它会被当成 `LIMIT ALL` ，即没有限制。如果 start计算为 NULL，它会被当作OFFSET 0。

  在使用LIMIT时，用一个 `ORDER BY` 子句把结果行约束到一个唯一顺序是个好办法。否则讲得到该查询结果行的一个不可预测的子集 — 可能要求从第 10 到第 20 行，但是在什么顺序下的第10到第20呢？除非指定 `ORDER BY` ，否则是不知道顺序的。

  查询规划器在生成一个查询计划时会考虑LIMIT，因此根据使用的LIMIT和OFFSET，很可能得到不同的计划（得到不同的行序）。所以，使用不同的LIMIT/OFFSET值来选择一个查询结果的不同子集将会给出不一致的结果，除非用ORDER BY强制一种可预测的结果顺序。这不是一个缺陷，它是SQL不承诺以任何特定顺序（除非使用 `ORDER BY` 来约束顺序）给出一个查询结果这一事实造成的必然后果。

  如果没有一个 `ORDER BY` 来强制选择一个确定的子集， 重复执行同样的LIMIT查询甚至可能会返回一个表中行的不同子集。同样，这也不是一种缺陷，再这样一种情况下也无法保证结果的确定性。

使用示例

- 两表JOIN

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM
distributors d, films f WHERE f.did = d.did;
```

- WITH子句

```
WITH distributor_name(name) AS SELECT name from distributors
SELECT name FROM distributor_name ORDER BY name;
```

- GROUP BY分组

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind;
```

- HAVING过滤

```
SELECT kind, sum(length) AS total FROM films GROUP BY kind
HAVING sum(length) < interval '5 hours';
```

- GROUP BY CUBE

```
SELECT  l_returnflag
        ,l_shipmode
        ,SUM(l_quantity)
FROM    public.lineitem
GROUP BY cube((l_returnflag),(l_shipmode))
ORDER BY l_returnflag
         ,l_shipmode;
```

- GROUP BY ROLLUP

```
SELECT  l_returnflag
        ,l_shipmode
        ,SUM(l_quantity)
FROM    public.lineitem
GROUP BY ROLLUP ((l_returnflag),(l_shipmode))
ORDER BY l_returnflag
         ,l_shipmode;
```

- GROUP BY GROUPING SETS

```
SELECT  l_returnflag
        ,l_shipmode
        ,SUM(l_quantity)
FROM    public.lineitem
GROUP BY GROUPING SETS ((l_returnflag,l_shipmode),())
ORDER BY l_returnflag
         ,l_shipmode;
```

- ORDER BY

```
SELECT * FROM distributors ORDER BY name;
```

6.6.2. INSERT

INSERT语句用于插入新的行数据至表中。本文为您介绍在交互式分析Hologres中如何使用INSERT插入数据。

命令格式

您可以插入一个或多个由表达式指定的行，以及插入来自一个查询的零行或多行数据至Hologres。语句如下。

```
INSERT INTO <table> [( <column> [, ...] )]
   VALUES ( {<expression>}  [, ...] )
   [, ...] | <query>}
```

参数说明

参数说明如下表所示。

参数	描述
table	已创建的表名称。 表名称也可以使用`Schema.Table`格式。若是分区表，必须插入具体的子表。通过Flink写入时可以插入父表并自动路由到子表。从Hologres V1.3版本起，支持符合FixedPlan的Insert语句直接写入分区表父表，详情请参见Fixed Plan加速SQL执行。
column	*table*表中的某个列的名称。 您也可以使用子域名或者数组下标限定列名称。（指向一个组合列的某些列中插入会让其他域为空）。

参数	描述
expression	赋予相应列的表达式或值。
query	需要被插入行的SELECT查询语句。 语法详情请参见SELECT语句。

目标列的名称可以以任意顺序列出。如果没有给出列名列表，则有两种确定目标列的可能性。第一种是以被声明的顺序列出该表的所有列。另一种可能性是，如果VALUES子句或者query只提供N个列，则以被声明的顺序列出该表的前N列。VALUES子句或者query提供的值会被从左至右关联到这些显式或者隐式给出的目标列。

每一个没有出现在显式或者隐式列列表中的列都将被默认填充，如果为该列声明过默认值则用默认值填充，否则用空值填充。如果任意列的表达式不是正确的数据类型，将会尝试自动转换类型。

Hologres针对高吞吐实时写入场景进行优化，提供Fixed Plan写入路径，性能远高于没有进入Fixed Plan的Insert语句，有关Fixed Plan的使用方式，请参考INSERT场景。

使用示例

Hologres支持使用如下INSERT语句写入数据：

- `INSERT INTO VALUES`

```
INSERT INTO rh_holo2mysqltest (cate_id, cate_name) VALUES
    (3, 'true'),
    (3, 'fale'),
    (3, 'trxxue'),
    (3, 'x'),
    (4, 'The Dinner Game');
```

- `INSERT INTO SELECT`

```
INSERT INTO test2
SELECT 2,'two';
```

- 插入分区表数据

  插入分区表数据，您需要将数据直接插入已经建好的子表中，插入时对应的分区字段值要和子表分区字段值一致。

```
--在public schema下创建不带主键的分区父表和对应的分区子表
BEGIN;
CREATE TABLE public.hologres_parent(
  a text,
  b int,
  c timestamp,
  d text
)
  PARTITION BY LIST(a);
CALL set_table_property('public.hologres_parent', 'orientation', 'column');
CREATE TABLE public.hologres_child1 partition of public.hologres_parent for values in('v1
');
CREATE TABLE public.hologres_child2 partition of public.hologres_parent for values in('v2
');
CREATE TABLE public.hologres_child3 partition of public.hologres_parent for values in('v3
');
COMMIT;
--插入分区表数据
INSERT INTO public.hologres_child1 values('v1',1,now(),'a');
```

6.6.3. INSERT ON CONFLICT(UPSERT)

INSERT ON CONFLICT语句用于在指定列插入某行数据时，如果主键存在重复的行数据，则对该数据执行更新或跳过操作，实现UPSERT(INSERT OR UPDATE)的效果。INSERT ON CONFLICT的执行开销要小于UPDATE语句，推荐将UPDATE语句改写为INSERT ON CONFLICT，语义一致。本文为您介绍在Hologres中INSERT ON CONFLICT语句的用法。

使用限制

- `INSERT ON CONFLICT`语句的条件必须包含所有主键。
- 推荐使用Fixed Plan优化Insert on Conflict执行效率，参考INSERT场景。
- 如果系统提示实例版本过低不支持该功能。您可以执行如下命令或提交工单升级实例至最新版本。

```
set hg_experimental_enable_insert_on_conflict = on;
```

应用场景

`INSERT ON CONFLICT`命令适用于通过SQL方式导入数据的场景。

使用数据集成或Flink写入数据时，如果需要对主键重复的行数据执行更新或跳过操作，则需进行如下配置：

- 通过DataWorks的数据集成导入数据。

  数据集成已内置`INSERT ON CONFLICT`功能，该功能的实现原理请参见Hologres Writer。同时，您需要进行如下配置：

  - 离线同步数据时，**写入冲突策略**选择**忽略（Ignore）**或者**更新（Replace）**。
  - 实时同步数据时，**写入冲突策略**选择**忽略（Ignore）**或者**更新（Replace）**。

  > **说明** 同步数据时，Hologres的表均需要设置主键，才能更新数据。

- 通过Flink写入数据。

通过Flink写入数据默认**写入冲突策略**使用**更新（Replace）**，但是需要您在Hologres建表时设置主键。详情请参见Hologres结果表。

命令格式

`INSERT ON CONFLICT` 的语法格式如下。

```
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
    {  VALUES ( { expression } [, ...] ) [, ...] | query }
    [ ON CONFLICT [ conflict_target ] conflict_action ]
where conflict_target is pk
    ON CONSTRAINT constraint_name
and conflict_action is one of:
    DO NOTHING
    DO UPDATE SET { column_name = { expression } |
                    ( column_name [, ...] ) = ( { expression } [, ...] ) |
                  } [, ...]
              [ WHERE condition ]
```

参数说明如下表所示。

参数	描述
DO NOTHING	在指定列插入某行数据时，如果主键存在重复的行数据，则对该数据执行跳过操作。
DO UPDATE	在指定列插入某行数据时，如果主键存在重复的行数据，则对该数据执行更新操作。
expression	对应列执行的相关表达式，您可以参考Postgres来设置表达式，详情请参见INSERT ON CONFLICT。 常用表达式 `b=excluded.b` ，或者 `(a, b, c) = ROW (excluded.*)` 简化表达，等式左侧表示要被更新的字段，等式右侧表示插入的表达式，即values部分的值，或者select表达式，excluded是对插入表达式的别名，不是写入源头表的别名。例如， `column_name = excluded.column_name` ，column_name为插入数据至目标表指定列的列名称，假设column_name为目标表的第N列，则excluded.column_name为插入表达式的第N列，当使用excluded.*时，表示选择所有列，列的顺序为插入表达式中列的顺序，需要保证插入目标列的顺序与被写入表的DDL顺序一致。

使用示例

- `INSERT ON CONFLICT` 语句的示例用法：

```
BEGIN;
CREATE TABLE tbl_1 (
    a int NOT NULL PRIMARY KEY,
    b int,
    c int
);
CREATE TABLE tbl_2 (
    d int NOT NULL PRIMARY KEY,
    e int,
    f int
);
COMMIT;
INSERT INTO tbl_1
    VALUES (1, 1, 1), (2, 3, 4);
--此时tbl_1数据如下
a    b    c
2    3    4
1    1    1
INSERT INTO tbl_2
    VALUES (1, 5, 6);
--此时tbl_2数据如下
d    e    f
1    5    6
--主键相同时，将表tbl_2的某列数据更新到表tbl_1中。
INSERT INTO tbl_1 (a, b)
SELECT
    d,
    e
FROM
    tbl_2
ON CONFLICT (a)
    DO UPDATE SET
        b = excluded.b;
--此时tbl_1数据如下
a    b    c
2    3    4
1    5    1
--根据过滤条件筛选被更新的行
INSERT INTO tbl_1
    VALUES (2, 7, 8)
ON CONFLICT (a)
    DO UPDATE SET
        b = excluded.b, c = excluded.c
    WHERE
        tbl_1.c = 4;
--此时tbl_1数据如下
a    b    c
2    7    8
1    5    1
--主键相同时，向表tbl_1插入表tbl_2的数据，系统直接跳过表tbl_2的数据（即插入数据失败）。
INSERT INTO tbl_1
```

```
INSERT INTO tbl_1
SELECT
    *
FROM
    tbl_2
ON CONFLICT (a)
    DO NOTHING;
--此时tbl_1数据如下
a    b    c
2    7    8
1    5    1
--do nothing不指定冲突列时，默认冲突列为主键。
INSERT INTO tbl_1
SELECT
    *
FROM
    tbl_2
ON CONFLICT
    DO NOTHING;
--此时tbl_1数据如下
a    b    c
2    7    8
1    5    1
---更新整行数据。
INSERT INTO tbl_1
    VALUES (1, 2, 3)
ON CONFLICT (a)
    DO UPDATE SET
        (a, b, c) = ROW (excluded.*);
--此时tbl_1数据如下
a    b    c
2    7    8
1    2    3
--将tbl_2整表替换tbl1表相同主键的行
INSERT INTO tbl_1 (a, b, c)
SELECT
    d,e,f
FROM
    tbl_2
ON CONFLICT (a)
    DO UPDATE SET
        (a,b,c) = ROW (excluded.*)
--此时tbl_1数据如下
a    b    c
2    7    8
1    5    6
--将tbl_2整表替换tbl1表相同主键的行,但调整了更新映射关系，即tbl_2的e列更新到c，f列更新到b
INSERT INTO tbl_1 (a, b, c)
SELECT
    d,e,f
FROM
    tbl_2
ON CONFLICT (a)
    DO UPDATE SET
        (a,c,b) = ROW (excluded.*)
```

```
--此时tbl_1数据如下
a    b    c
2    7    8
1    6    5
```

- 行存表 `INSERT ON CONFLICT` 语句的优化：

  Hologres对行存表的更新场景实行了优化，建议您在使用时将UPDATE列的顺序与INSERT的顺序保持一致，并且更新为整行更新。

```
INSERT INTO tbl_1 (a, b, c)
SELECT
    d,e,f
FROM
    tbl_2
ON CONFLICT (a)
    DO UPDATE SET
        a = excluded.a,
        b = excluded.b,
        c = excluded.c;
INSERT INTO tbl_1 (a, b, c)
SELECT
    d,e,f
FROM
    tbl_2
ON CONFLICT (a)
    DO UPDATE SET
        (a,b,c) = ROW (excluded.*)
```

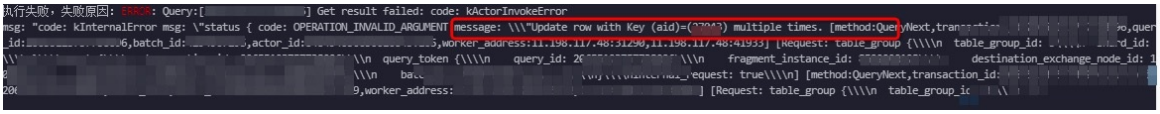
常见报错

- 问题现象

  对数据源执行 `INSERT ON CONFLICT` 语句时出现如下两种报错其中一个。

  - 报错一：

    `Update row with Key ()=() multiple times.`

  - 报错二：

    `ERROR: duplicate key value violates unique constraint`

- 问题原因

  Hologres兼容PostgreSQL，使用的也是标准PostgreSQL语法。在标准的PostgreSQL语义中，对数据源执行 `INSERT ON CONFLICT` 语句时，数据源不能包含重复数据，如果包含重复数据则会产生上述报错。

  > **说明** 数据源重复是指待插入的数据中包含重复数据，不是指待插入的数据与表里的数据重复。

使用 `INSERT ON CONFLICT` 语句插入数据时包含重复数据，示例语句如下。

```
INSERT INTO tbl_1
    VALUES (1, 2, 3), (1, 2, 3)
ON CONFLICT (a)
    DO UPDATE SET
        (a, b, c) = ROW (excluded.*);
```

- 解决方法

  如果数据源包含重复数据，可以配置如下参数，分别选择保留重复数据的第一条数据和最后一条数据。

  - 保留重复数据的第一条数据

    ```
    set hg_experimental_affect_row_multiple_times_keep_first = on;
    ```

  - 保留重复数据的最后一条数据

    ```
    set hg_experimental_affect_row_multiple_times_keep_last = on;
    ```

6.6.4. INSERT OVERWRITE

目前Hologres暂不支持INSERT OVERWRITE命令，本文为您介绍在Hologres中如何使用SQL实现INSERT OVERWRITE的功能。

命令格式

您可以使用如下SQL语句实现INSERT OVERWRITE的功能。

```
BEGIN ;
-- 清理潜在的临时表
DROP TABLE IF EXISTS <table_new>;
-- 创建临时表
SET hg_experimental_enable_create_table_like_properties=on;
CALL HG_CREATE_TABLE_LIKE ('<table_new>', 'select * from <table>');
COMMIT ;
-- 向临时表插入数据
INSERT INTO <table_new> [( <column> [, ...] )]
VALUES ( {<expression>}  [, ...] )
[, ...] | <query>}
ANALYZE <table_new>;
BEGIN ;
-- 删除旧表
DROP TABLE IF EXISTS  <table>;
-- 临时表改名
ALTER TABLE <table_new> RENAME TO <table>;
COMMIT ;
```

参数说明

参数	说明

参数	说明
table_new	新创建的临时表名称。 表名称也可以使用 `Schema.Table` 格式。
table	已存在的表名称。 表名称也可以使用 `Schema.Table` 格式。

临时表DDL	创建临时表有如下两种方式。
	• 通过复制已有表创建新表的结构
	`SET hg_experimental_enable_create_table_like_properties=on;`
	`CALL HG_CREATE_TABLE_LIKE ('<table_new>', 'select * from <table>');`
	• 新建表的结构
	`CREATE TABLE IF NOT EXISTS <table_new> ([ { column_name column_type [column_constraints, [...]] | table_constraints [, ...] } ]);`
	`CALL set_table_property('<table_new>', property, value);`

使用示例

- 场景一：MaxCompute向Hologres的非分区表导入数据

  在MaxCompute向Hologres导入数据的场景中，希望将数据全量覆盖，常见于离线加工后的结果表导出为线上服务表。此场景使用示例如下所示，将MaxCompute中的odps_region_10g表的数据写入Hologres的region表中，且将Hologres中region表的数据全量覆盖。

```
-- 刷新外表的Schema
IMPORT FOREIGN SCHEMA holo_demo LIMIT to
(
    odps_region_10g
)
FROM SERVER odps_server INTO public
OPTIONS(if_table_exist 'update',if_unsupported_type 'error');
BEGIN ;
-- 清理潜在的临时表
DROP TABLE IF EXISTS public.region_new;
-- 创建临时表
SET hg_experimental_enable_create_table_like_properties=on;
CALL HG_CREATE_TABLE_LIKE ('public.region_new', 'select * from public.region');
COMMIT ;
-- 向临时表插入数据
INSERT INTO public.region_new
SELECT *
FROM public.odps_region_10g;
ANALYZE public.region_new;
BEGIN ;
-- 删除旧表
DROP TABLE IF EXISTS public.region;
-- 临时表改名
ALTER TABLE IF EXISTS public.region_new RENAME TO region;
COMMIT ;
```

- 场景二：MaxCompute向Hologres的分区表导入数据

  在每天定期更新MaxCompute分区表的数据，且需要将MaxCompute分区表向Hologres的分区表导入数据的场景中，希望将数据全量覆盖，实现离线数据对实时数据的修正。此场景使用示例如下所示，将MaxCompute中的odps_lineitem_10g表的数据写入Hologres的lineitem表中，且全量覆盖Hologres中lineitem表的数据，两个表都是按照ds字段按天分区。

```
-- 刷新外表的Schema
IMPORT FOREIGN SCHEMA holo_demo LIMIT to
(
    odps_lineitem_10g
)
FROM SERVER odps_server INTO public
OPTIONS(if_table_exist 'update',if_unsupported_type 'error');
BEGIN ;
-- 清理潜在的临时表
DROP TABLE IF EXISTS public.lineitem_new_20210101;
-- 创建临时表
SET hg_experimental_enable_create_table_like_properties=on;
CALL HG_CREATE_TABLE_LIKE ('public.lineitem_new_20210101', 'select * from public.lineitem
');
COMMIT ;
-- 向临时表插入数据
INSERT INTO public.lineitem_new_20210101
SELECT *
FROM public.odps_lineitem_10g
WHERE DS = '20210101'
ANALYZE public.lineitem_new_20210101;
BEGIN ;
-- 删除旧分区
DROP TABLE IF EXISTS public.lineitem_20210101;
-- 临时表改名
ALTER TABLE public.lineitem_new_20210101 RENAME TO lineitem_20210101;
-- 将临时表绑定至指定分区表
ALTER TABLE public.lineitem ATTACH PARTITION lineitem_20210101 FOR VALUES IN ('20210101')
;
COMMIT ;
```

6.6.5. DELETE

DELETE语句用于删除目标表指定列的行数据。本文为您介绍在Hologres中DELETE语句的用法。

使用限制

- Hologres暂不支持直接删除分区表父表。您需要删除具体的分区表子表后，才可以删除分区表父表。
- 如果执行整表数据的清空删除，建议使用TRUNCATE语法，效率更高，参考 TRUNCATE。
- 推荐使用Fixed Plan优化Delete执行效率，参考 DELETE场景。

语法

DELETE命令的语法如下所示。

```
DELETE FROM table_name [ * ] [ [ AS ] alias ]
    [ WHERE condition  ]
```

参数说明如下表所示。

参数	描述
alias	别名。目标表的替代名称。
condition	删除目标表的条件。

示例

删除表的示例语句如下。

```
CREATE TABLE delete_test (
    id INT PRIMARY KEY,
    a INT,
    b text
);
INSERT INTO delete_test VALUES
(1, 10, 'a'),
(2, 30, 'b'),
(3, 50,  ''),
(4, 70, null);
DELETE FROM delete_test AS dt WHERE dt.a = 10;
DELETE FROM delete_test AS dt WHERE dt.b is null;
DELETE FROM delete_test AS dt WHERE dt.b='';
```

更多关于DELETE的详情，请参见PostgreSQL DELETE。

6.6.6. UPDATE

UPDATE语句用于更新目标表指定列的行数据。本文为您介绍在Hologres中UPDATE语句的用法。

使用限制

- Hologres不支持更新distribution key。
- Hologres不支持直接更新分区表父表，您只能更新具体的分区表子表。
- 推荐使用Insert on conflict语法实现Update语义，Update执行需要获取表级锁，因此执行代价较大，锁的时间长，而Insert on conflict采用行级锁，执行代价更低，效率更高，参考INSERT ON CONFLICT(UPSERT)。
- 推荐使用Fixed Plan优化Update执行效率，参考UPDATE场景。

语法

UPDATE命令的语法如下所示。

```
UPDATE table [ * ] [ [ AS ] alias ]
    SET { column = { expression } |
          ( column [, ...] ) = ( { expression } [, ...] ) } [, ...]
    [ FROM from_list ]
    [ WHERE condition ]
```

参数说明如下表所示。

参数	描述
alias	别名。目标表的替代名称。
expression	表达式
condition	更新目标表的条件。

示例

更新表的示例语句如下。

```
CREATE TABLE update_test (
  a text primary key,
  b int not null,
  c text not null,
  d text);
INSERT INTO update_test VALUES ('b1', 10, '', '');
UPDATE update_test SET b = b + 10 where a = 'b1';
UPDATE update_test SET c = 'new_' || a, d = null where b = 20;
UPDATE update_test SET (b,c,d) = (1, 'test_c', 'test_d');
CREATE TABLE tmp(a int);
INSERT INTO tmp VALUES (2);
UPDATE update_test SET b = tmp.a FROM tmp;
```

更多关于UPDATE命令的详情，请参见PostgreSQL官网文档。

6.6.7. TRUNCATE

TRUNCATE语句用于清空目标表。本文为您介绍Hologres中TRUNCATE的语法。

使用限制

`TRUNCATE` 语句的使用限制如下：

- Hologres支持Sequence，但目前仅支持CONTINUE IDENTITY，不支持RESTART IDENTITY。
- Hologres支持对普通表、分区父表及分区子表执行 `TRUNCATE` 语句。
- Hologres不支持对外部表执行 `TRUNCATE` 语句。

语法

`TRUNCATE` 的语法格式如下。

```
TRUNCATE [ TABLE ] name [, ... ]
    [CONTINUE IDENTITY ]
```

CONTINUE IDENTITY参数不修改当前Sequence的值。

使用示例

`TRUNCATE` 的使用示例如下。

- 示例一。

```
CREATE TABLE event (
    id int,
    name text,
    tag text
);
INSERT INTO event (id,name,tag) values (23,'buy', 'num');
SELECT * FROM event;
TRUNCATE TABLE event ;
```

- 示例二。

```
CREATE TABLE event_1 (
    id serial,
    name text,
    tag text
);
INSERT INTO event_1 (name,tag) values ('buy', 'num');
SELECT * FROM event_1;
#默认为CONTINUE IDENTITY
TRUNCATE TABLE event_1 CONTINUE IDENTITY;
```

6.6.8. ANALYZE和AUTO ANALYZE

本文将为您介绍如何使用Analyze命令，以及更加简单的Auto Analyze的相关机制。

Analyze

统计信息决定是否能够生成正确的执行计划。Hologres需要收集数据的采样统计信息，包括数据的分布和特征、表的统计信息、列的统计信息、行数、列数、字段宽度、基数、频度、最大值、最小值、高频值、分桶分布特征等信息。这些信息将为优化器更新算子执行预估COST、搜索空间裁剪、估算最优JOIN ORDER、估算内存开销、估算并行度，从而生成更优的执行计划。

Analyze命令用于收集数据库中表内容的统计信息，优化器会根据这些统计信息生成最佳的查询计划，从而提高查询效率。

- 使用语法

```
-- 更新某个表的统计信息，默认会收集表中所有列的统计信息
analyze <tablename>;
-- 更新某个列的统计信息，会比更新表时采样的数据更多，更精准，主要用于更新管理条件的列
analyze <tablename>(<colname>, <colname>);
```

- 参数说明

  tablename为更新统计信息的表名称，colname为更新统计信息的列名称。

- 语法说明

  两个Analyze命令的说明如下。

- 相同点
  - 对列统一收集包括行数、列宽、列的最常用值（Most Common Values）、列的直方图（Histogram）信息，列的非重复值的个数（Number of Distinct Value，NDV）在内的信息。
  - 两个命令都会相互覆盖指定列的统计信息，但不会覆盖其他列的信息。例如 `analyze <tablename>(<colname1>);` 命令会覆盖（更新）之前 `colname1` 列收集的统计信息，但并不会改变 `colname2` 列的统计信息。
- 不同点
  - `analyze <tablename>;` 基于采样数据，计算得出统计信息。
  - `analyze <tablename>(<colname>, <colname>);` 会对列的Number of Distinct Value（NDV）进行APPROX_COUNT_DISTINCT计算，在很多情况下，这样计算的值相比采样更准确，但开销比采样表更大，因此只适合对重点列进行指定ANALYZE。NDV以外的Histogram、Width等信息，仍然通过采样得到。

  因此对于具有两列的 `table (colname1, colname2)` ， `analyze table;` 不完全等价于 `analyze table(colname1, colname2);` 。

  对于常用的Join列、Group By列，推荐使用 `analyze <tablename>(<colname>, <colname>);` 命令进行额外的统计信息收集。

- 需要执行Analyze的情况

  推荐您在如下情况下运行 `analyze <tablename>;` 命令。
  - 在表执行大量的INSERT、UPDATE以及DELETE操作之后，包括导入数据。
  - 在性能下降的情况下，多表Join查询之前，对Join的列、Group by的列进行Analyze。
  - 执行 `CREATE FOREIGN TABLE` 命令后，通过Analyze收集当前外部表统计信息。
  - 执行 `IMPORT FOREIGN SCHEMA` 后，对后续需要查询的表进行Analyze。
- 注意事项
  - 在Hologres V0.10和V1.1版本中，如果有对父表的查询，需要Analyze分区父表；如果直接对子表查询，请对子表Analyze；如果两者都有，建议两者都进行Analyze，否则可能会有缺失统计信息的情况。
  - 如果遇到以下问题，您需要先执行Analyze，再运行导入任务，可以系统地提升效率。
    - 多表JOIN超出内存OOM：通常会产生 `Query executor exceeded total memory limitation xxxxx: yyyy bytes used` 报错。
    - 导入效率较低：在Hologres查询或导入数据时，效率较低，运行的任务长时间不结束。
  - 如果有超宽列（例如Bitmap等Bytea数据，超过1KB的Text数据等），这些超宽列的统计信息没有作用，还会使采样更消耗内存。因此对于具有上述超宽列的表，尽量避免执行 `analyze <tablename>;` 命令，而是采用 `analyze <tablename>(<colname>, <colname>);` 避开超宽列，转为Analyze必要的列（例如上面推荐的Join的列、Group by的列和Filter列等）。

    > **说明** 1KB是经验值，宽度标准可以根据业务情况自行决定。

Auto Analyze

为了减少重复、手动的Analyze，从Hologres V0.10版本开始，支持Auto Analyze机制。开启auto analyze后，系统会根据用户的建表、数据写入和修改情况等来判断是否需要对相关的表在后台自动Analyze，无需再手动对表进行Analyze，降低操作复杂度，同时减少遗漏Analyze而导致缺失统计信息的情况。

- 使用语法
  - 查看是否开启Auto Analyze

```
SHOW hg_enable_start_auto_analyze_worker;  -- V1.1语法，查看当前开启/关闭状态
SHOW hg_experimental_enable_start_auto_analyze_worker;  -- V0.10语法，查看当前开启/关闭状态
```

  - 开启/关闭语法如下，需要Superuser执行。

```
-- DB级别，执行后整个DB生效，V1.1开启/关闭语法
ALTER DATABASE dbname SET hg_enable_start_auto_analyze_worker = ON;  -- 开启（默认）
ALTER DATABASE dbname SET hg_enable_start_auto_analyze_worker = OFF; -- 关闭
-- DB级别，执行后整个DB生效，V0.10开启/关闭语法
ALTER DATABASE dbname SET hg_experimental_enable_start_auto_analyze_worker = ON;  -- 开启（默认）
ALTER DATABASE dbname SET hg_experimental_enable_start_auto_analyze_worker = OFF; -- 关闭
```

- 使用限制

  在Hologres中使用Auto Analyze，具体限制如下：

  - Auto Analyze功能仅Hologres V0.10及以上版本支持，请在Hologres管理控制台的实例详情页查看当前版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
  - 仅支持Superuser执行开启或关闭Auto Analyze操作。
  - Auto Analyze对分区表的限制如下。
    - 分区子表发生改变，需要Auto Analyze时，会统一Analyze其父表。
    - 分区表有扫描行数限制，采样数据时默认扫描的最大记录数是2^{24}条（16,777,216条），即若所有分区子表的记录数总和超过16,777,216条，会做一定的分区裁剪，只对其中若干分区（总和不超过16,777,216条）进行采样。

      > **说明** 分区列统计信息总是全的，不受裁剪影响，但是这可能会影响与分区列同分布的列（例如极端情况是，与分区列数据一样的列）的统计信息，即一部分值采样不到，行数估计可能不准确。如果有需求可以提交工单或者联系技术支持，技术侧根据实例情况评估调整扫描的最大记录数。

  - Auto Analyze默认最大收集256列的统计信息，如表超过256列，取前256列。可通过调整 `hg_experimental_auto_analyze_max_columns_count` 改变此值。
  - Auto Analyze默认单个Worker限制的内存是8 GB，如果存在超宽的列，采样可能超出内存而导致Analyze失败。可调整 `auto_analyze_work_memory_mb` 参数改变其大小，但是要注意对系统内存的影响。实例规格越大，Worker数越多，Auto Analyze可用内存限制越大。

- Auto Analyze工作原理

  当开始Auto Analyze后，系统后台会定期巡检是否有表需要执行Analyze。

- 普通表（内部表，包括单表和分区表）
  - 每隔1分钟巡检是否有表的最新动作（主要是INSERT、UPDATE、DELETE等DML操作，可能改变了数据量）。满足以下条件，系统后台触发表的Analyze，收集表的统计信息。
    - 表有DML执行完成且数据条数变化超过当前表的数据条数的10%。若表是分区子表，则是指变化条数超过此分区数据条数的10%。
    - TRUNCATE TABLE清空表。
    - 表的DDL发生变更。例如CREATE TABLE新建表，ALTER TABLE修改表结构等，不包括CALL SET_TABLE_PROPERTY修改表属性。
  - 每隔10分钟检测所有内部表的数据变化，如果满足数据条数变化超过上一次检测的10%，则后台触发该表的Analyze。

    > **说明** 这一步骤是为了检测到非显式DML（例如通过Flink、数据集成、HoloClient实时写入）更新的数据。
- 外部表
  - 当前仅支持Analyze MaxCompute外部表，其他引擎的外部表暂不支持Analyze和Auto Analyze。
  - 每隔4小时定期巡检外部表元数据或数据变化情况。满足以下条件，系统后台触发表的Analyze，收集统计信息。

    外部表对应的外部系统的表（例如MaxCompute表）在两次巡检间隔（例如4小时内）改变过，改变的标准是对应MaxCompute表的`last_modify_time`处于巡检间隔之间。

> **说明** 巡检和执行在同一个调度任务中，所以下一次巡检调度开始依赖Analyze执行结束，只要离上一次开始巡检的时间满足调度周期，就可以进入下一次巡检。

- 配置参数

  开启Auto Analyze后，系统默认会自动周期性巡检，决定需要执行Analyze的表，并进行采样计算，收集统计信息，对系统资源有一定的消耗。

  在某些业务场景下，默认的机制可能不适用于业务场景，例如数据写入更新不频繁场景，可以通过修改默认参数来减少自动Analyze的频率。诸如此类可以根据业务情况更改默认参数，以此达到部分性能调优的目的。

  > **说明** 只有Superuser能调整Auto Analyze的默认行为，且都需要数据库级别设置参数，且在下一分钟后生效。

  - 使用语法

    ```
    --Superuser修改Auto Analyze参数的默认值
    ALTER DATABASE <dbname> SET <GUC>=<values>;
    ```

    dbname为数据库名称；GUC为参数名称；values为参数值。
  - 参数列表

参数	参数描述	支持版本	默认值	使用示例

参数	参数描述	支持版本	默认值	使用示例
autovacuum_naptime	巡检是否有表的最新动作的周期，单位是秒（s）。	V1.1.0及以上版本 **说明** 需后台调整，如需调整请提交工单。	60s	■ `ALTER DATABASE <dbname> SET autovacuum_naptime = 60;` ■ `ALTER DATABASE <dbname> SET autovacuum_naptime = '60s';` ■ `ALTER DATABASE <dbname> SET autovacuum_naptime = '10min';`

hg_auto_check_table_changes_interval	检查所有内部表的数据变化的周期，单位是秒（s）。	V1.1.0及以上版本	600s（10min）	--V1.1及以上版本命令语法
				`ALTER DATABASE <dbname> SET hg_auto_check_table_changes_interval = '600s';`
				--V0.10版本命令语法
				`ALTER DATABASE <dbname> SET hg_experimental_auto_check_table_changes_interval = '600s';`

参数	参数描述	支持版本	默认值	使用示例

hg_auto_check_foreign_table_changes_interval	检查所有外部表的数据变化的周期，单位是秒（s）。	V1.1.0及以上版本	14400s（4小时）	--V1.1及以上版本命令语法
				`ALTER DATABASE <dbname> SET hg_auto_check_foreign_table_changes_interval = '14400s';`
				--V0.10版本命令语法
				`ALTER DATABASE <dbname> SET hg_experimental_auto_check_foreign_table_changes_interval = '14400s';`

hg_experimental_auto_analyze_max_columns_count	自动收集统计信息的列数，单位是个。	V1.1.0及以上版本	256个	`ALTER DATABASE <dbname> SET hg_experimental_auto_analyze_max_columns_count =300;`
auto_analyze_work_memory_mb	Auto Analyze单个表的内存限制，单位是MB。 说明 仅1.1及以上版本支持该参数。	V1.1.54及以上版本	默认单个Worker 8192 MB，即8GB，实例规格越大，Worker越多，真实内存限制越大。	Auto Analyze单个表的内存限制修改为9GB。 `ALTER DATABASE <dbname> SETauto_analyze_work_memory_mb =9216;`

参数	参数描述	支持版本	默认值	使用示例

hg_experimental_auto_analyze_start_time	Auto-Analyze在每天运行的开始时间
	说明 需要与end_time是同一时区，并且start time要小于等于end_time。
		V1.1.54及以上版本	00:00 +0800	修改为仅需要在0~6点执行Auto-Analyze，白天内外部表数据不变，无需Analyze的情况。
				`ALTER DATABASE <dbname> SET hg_experimental_auto_analyze_start_time = '00:00 +0800';`
				`ALTER DATABASE <dbname> SET hg_experimental_auto_analyze_end_time = '06:00 +0800';`

hg_experimental_auto_analyze_end_time	Auto Analyze在每天运行的结束时间。	V1.1.54及以上版本	23:59 +0800	
autovacuum_enabled	表Auto Analyze的开启状态。	V1.1.54及以上版本	true，即默认全部开启。	关闭某表的Auto Analyze，以后Analyze将跳过此表。 `ALTER TABLE <tablename> SET (autovacuum_enabled = false);`

查询统计信息

表的统计信息被记录在hologres_statistic.hg_table_statistic表中，可以通过检查该表信息了解Analyze的状态，命令如下。

说明 如果需要查最近一次Analyze的信息，根据 `analyze_timestamp` 排序即可。

```
SELECT schema_name,                -- 表的Schema
       table_name,                 -- 表名称
       schema_version,             -- 表的版本
       statistic_version,          -- 最近一次ANALYZE的统计信息版本
       total_rows,                 -- 最近一次ANALYZE的行数
       analyze_timestamp           -- 最近一次ANALYZE的结束时间
FROM   hologres_statistic.hg_table_statistic
WHERE  table_name = '<tablename>'
ORDER BY analyze_timestamp DESC;
```

- 每个表在hologres_statistic.hg_table_statistic表中有 `0~n` 条记录。0条表示从未进行过Analyze，1条及以上表示运行过Analyze。
- 若出现两条及以上的情况，两条记录的schema_version一定不一样，因为表的Schema变化了（例如执行 `ADD COLUMN` 等命令会产生新的版本），会增加一条统计信息记录，老的schema_version对应的记录不再被使用。
- 示例查询结果如下，同一个表有两条记录，而第二条记录的schema_version低于第一条，那么第二条将作废，不会被使用，也无需关注。

```
 schema_name |    table_name    | schema_version | statistic_version | total_rows |  anal
yze_timestamp
-------------+------------------+----------------+-------------------+------------+------
---------------
 public      | tbl_name_example |             13 |              8580 |      10002 | 2022-
04-29 16:03:18
 public      | tbl_name_example |             10 |              8576 |      10002 | 2022-
04-29 15:41:20
(2 rows)
```

- Hologres V0.10和V1.1版本暂不会清理hg_table_statistic表中的历史过期记录，同时不用关心老的数据。

常见问题

出现如下情况，代表Auto Analyze工作未正常，请参照解决方法进行处理。

- 表的统计信息是0条
  - 问题现象：通过hologres_statistic.hg_table_statistic表查看表的统计信息，没有数据。
  - 可能原因：
    - Auto Analyze没有工作，或者该表不符合Auto Analyze触发条件。
    - Auto Analyze本身的问题导致，需要提交工单具体排查原因。
  - 解决方法：手动触发一次Analyze。
- `analyze_timestamp` 过小
  - 问题现象：查询结果中 `analyze_timestamp` 过小（即比当前时间小很多），代表长时间没有进行过Analyze。
  - 可能原因：
    - 某种原因未能正常执行Auto Analyze。
    - 手动关闭过Auto Analyze。
  - 解决方法：先手动触发Analyze，再提交工单排查原因。

6.6.9. Fixed Plan加速SQL执行

Fixed Plan是Hologres独有的执行引擎优化方式，本文将为您介绍可以被Fixed Plan选中的SQL需要符合的条件和参数配置。

背景信息

Fixed Plan是Hologres独有的执行引擎优化方式，传统的SQL执行要经过优化器、协调器、查询引擎、存储引擎等多个组件，而Fixed Plan选择了短路径（Short-Cut）优化执行SQL，绕过了优化器、协调器、部分查询引擎的开销。通过Fixed Frontend直接对接Fixed Query Engine，实现SQL执行效率的成倍提升，是支持高吞吐实时写入，高并发查询的关键优化方法。关于Fixed Plan的介绍请参见产品架构。

相关GUC参数

- GUC列表

  以下为Fixed Plan需要用到的参数配置，所有参数取值为on或者off。所有参数已经在Holo Client中默认打开，session级别生效。

GUC名称	适用场景	默认值
hg_experimental_enable_fixed_dispatcher	查看实例的Fixed Plan是否打开。	on
hg_experimental_enable_fixed_dispatcher_for_multi_values	支持insert on conflict多行记录的Fixed Plan写入。建议客户端session级别打开。 **说明** 不保证原子性，即一次性写入多条的时候，如果没报错就是全部正常写入了，如果报错了有可能全部没写入也有可能部分写入了部分没写入，没有写入的部分会将错误反馈给上层应用端，由应用端进行重试。	off
hg_experimental_enable_fixed_dispatcher_autofill_series	支持含有Serial类型列的Fixed Plan写入。建议客户端session级别打开。	off
hg_experimental_enable_fixed_dispatcher_for_update	支持更新（UPDATE）场景的Fixed Plan更新。建议客户端session级别打开。	off
hg_experimental_enable_fixed_dispatcher_for_delete	支持删除（DELETE）场景的Fixed Plan删除。建议客户端session级别打开。	off
hg_experimental_enable_fixed_dispatcher_for_scan	支持PrefixScan场景的Fixed Plan查询。 **说明** PrefixScan是指多列主键，查询条件只给前面几列主键。	off

GUC名称	适用场景	默认值
hg_experimental_enable_bhclient_cache_on_session	支持更改缓存模式，存在以下两种情况。 ○ on：使用cached on session模式。 ○ off：使用cached on fe模式。 **说明** cached on session和cached on fe的区别如下。 ○ cached on session是每个连接拥有自己的Writer、Reader，单连接的吞吐更好，但启动会更慢（每个表第一次进行读或写时需要有启动时间）。 ○ cached on fe是FE（Frontend）节点上所有连接共享Writer、Reader，连接断开后Writer、Reader不会关闭，所以总体上没有启动时间。	off

- GUC使用
  - 查看GUC是否开启

    通过show命令查看GUC是否开启，命令语句如下。

    ```
    show <GUC_name>;
    ```

    使用示例如下。

    ```
    -- 查看实例级别是否开启Fixed Plan
    show hg_experimental_enable_fixed_dispatcher;
    ```

- 开启GUC
  - session级别开启GUC

    通过 `set` 命令可以在session级别设置GUC参数。session级别的参数只在当前session生效，当连接断开之后，将会失效，建议加在SQL前一起执行，语法示例如下。

    ```
    set <GUC_name> = <values>;
    ```

    GUC_name为GUC参数的名称；values为GUC参数的值。

    使用示例如下。

    ```
    -- insert on conflict多行记录支持Fixed Plan写入
    set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
    ```

  - 数据库级别开启GUC

    通过 `alter database xx set xxx` 命令来设置数据库级别的GUC参数，执行完成后在整个数据库生效，设置完成后当前连接需要重新断开连接才能生效。新建数据库不会生效，需要重新手动设置，语法示例如下。

    ```
    alter database <db_name> set <GUC_name> = <values>;
    ```

    db_name为数据库名称，GUC_name为GUC参数的名称，values为GUC参数的值。

    使用示例如下。

    ```
    --DB级别开启fixed plan
    alter database <db_name> set
     hg_experimental_enable_fixed_dispatcher =on;
    ```

数据类型要求

- 表的每一列都不能是MONEY或MONEY ARRAY类型。
- 进行DML(INSERT/UPDATE/DELETE)操作的列和进行SELECT（select的target列和where里的列都要满足）操作的列支持的类型如下。
  - BOOLEAN（别名BOOL）
  - SMALLINT
  - INTEGER（别名INT或INT4）
  - BIGINT（别名INT8）
  - FLOAT（别名FLOAT4）
  - DOUBLE PRECISION（别名FLOAT8）
  - CHAR(n)
  - VARCHAR(n)
  - BYTEA
  - JSON和JSONB
  - TEXT（别名VARCHAR）
  - TIMESTAMP WITH TIME ZONE（别名TIMESTAMPTZ）
  - DATE
  - TIMESTAMP
  - DECIMAL（别名NUMERIC）

- ROARINGBITMAP
- 数组类型
  - boolean[]
  - smallint[]
  - int4[]
  - int8[]
  - float4[]
  - float8[]
  - char(n)[]
  - varchar(n)[]
  - text[]

INSERT场景

- INSERT表达式

  Fixed Plan支持以下INSERT表达式。

```
-- 写入单行
insert into table(col1,col2,col3..) values(?,?,?..) on conflict xxx;
-- 写入多行
insert into table(col1,col2,col3..) values(?,?,?..),(?,?,?..) on conflict xxx;
```

  > 说明
  > - 支持使用INSERT命令写入内部表，不支持外部表。
  > - 支持使用INSERT命令写入分区表，Hologres V1.3及以上版本支持写入分区父表。

- Insert on conflict单行
  - 支持场景如下。
    - 支持没有 `on conflict` 的表达式。
    - 支持含有 `on conflict do nothing` 的表达式。
    - 支持 `on conflict do update` ，必须更新所有insert的非PK（Primary Key，主键，以下简称PK）列，PK是否更新都可以，并且只能是 `col = excluded.col` 方式更新。

- 使用示例如下。

```
begin;
create table test_insert_oneline(
  pk1 int,
  pk2 int,
  col1 int,
  col2 int,
  primary key(pk1, pk2)
);
commit;
--update所有非PK列，可以Fixed Plan
insert into test_insert_oneline values(1,2,3,4) on conflict(pk1,pk2) do update set col1
= excluded.col1, col2 = excluded.col2;
--update所有列(包含PK和非PK)，可以Fixed Plan
insert into test_insert_oneline values(1,2,3,4) on conflict(pk1,pk2) do update set col1
= excluded.col1, col2 = excluded.col2, pk1 = excluded.pk1, pk2 = excluded.pk2;
--必须update所有非pk的要insert的列，此例子不包含col2，因此不能Fixed Plan
insert into test_insert_oneline values(1,2,3,4) on conflict(pk1,pk2) do update set col1
= excluded.col1;
--必须是set col = excluded.col方式更新，因此不能Fixed Plan
insert into test_insert_oneline values(1,2,3,4) on conflict(pk1,pk2) do update set col1
= excluded.col1, col2 = 5;
```

- Insert on conflict多行
  - insert on conflict多行时，表达式如下。

```
set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
insert into table(col1,col2,col3..) values(?,?,?..),(?,?,?..) on conflict xxx;
```

    - 需要配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_multi_values=on;` 。
    - 不保证原子性，即一次性写入多条的时候，如果没报错就是全部正常写入了，如果报错了有可能全部没写入也有可能部分写入了部分没写入。

- 写入多行另一种写法表达式如下。

```
set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
insert into table select
unnest(ARRAY[true, false, true]::bool[]),
unnest(ARRAY[1,2,3]::int4[]),
unnest(ARRAY[1.11,2.222,3]::float4[]) on conflict xxx;
```

  - 需要配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_multi_values=on;`。
  - 写入的列不能是数组类型。
  - unnest里ARRAY必须显式转换为对应列类型的数组类型。

  使用示例如下。

```
begin;
create table test_insert_multiline(
  pk1 int8,
  col1 float4,
  primary key(pk1)
);
commit;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
insert into test_insert_multiline select unnest(ARRAY[1,2,3]::int8[]), unnest(ARRAY[1.1
1,2.222,3]::float4[]) on conflict do nothing;
--unnest里ARRAY没有显式cast，不支持Fixed Plan
insert into test_insert_multiline select unnest(ARRAY[1,2,3]), unnest(ARRAY[1.11,2.222,
3]) on conflict do nothing;
--第一列是int8,所以应该cast为int8[],这里例子是int4[]，因此不支持Fixed Plan
insert into test_insert_multiline select unnest(ARRAY[1,2,3]::int4[]), unnest(ARRAY[1.1
1,2.222,3]::float4[]) on conflict do nothing;
```

- 局部更新场景

  Hologres支持通过主键，对表的部分列更新，Fixed Plan同样支持局部更新场景，需要满足如下条件。

  - Insert的列需要与Update的列一一对应，包括数量和顺序。
  - 只能是`col = excluded.col`方式更新。

- Default列

  表中含有Default列时，可以进行Fixed Plan的条件如下。

  - 插入单条数据时支持。
  - 插入多条数据时，需要Hologres实例在V1.1.36及以上版本，若低于此版本请升级实例。
  - 配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_multi_values=on;`。
  - 有Default列的表，不支持`insert on conflict`表达式的Fixed Plan。

  使用示例如下。

```
begin;
create table test_insert_default(
  pk1 int,
  col1 int default 99,
  primary key(pk1)
);
commit;
--支持Fixed Plan
insert into test_insert_default(pk1) values(1);
--需要V1.1.36及以上版本支持
set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
insert into test_insert_default(pk1) values(1),(2),(3);
```

- Serial列

  表带有自增序列Serial时，支持单条或者多条写入进行Fixed Plan的条件如下。

  - 配置GUC参数：`set hg_experimental_enable_fixed_dispatcher_autofill_series=on;` 。
  - 插入多条数据时，还需配置GUC参数：`set hg_experimental_enable_fixed_dispatcher_for_multi_values=on;` 。
  - 有Serial列的表，不支持 `insert on conflict` 表达式的Fixed Plan。

  使用示例如下。

```
begin;
create table test_insert_serial(
  pk1 int,
  col1 serial,
  primary key(pk1)
);
commit;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_autofill_series =on;
insert into test_insert_serial (pk1) values(1);
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_autofill_series =on;
set hg_experimental_enable_fixed_dispatcher_for_multi_values =on;
insert into test_insert_serial (pk1) values(1),(2),(3);
```

UPDATE场景

- Update表达式

  Update时能进行Fixed Plan的表达式如下。

```
set hg_experimental_enable_fixed_dispatcher_for_update =on;
update table set col1 = ?, col2 = ? where pk1 = ? and pk2 = ?;
```

  - 需要配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_update=on;` 。
  - 支持Update内部表，不支持外部表。
  - 支持分区子表，不支持分区父表。
  - 表必须有主键（PK）。

- Update场景使用

Update场景支持进行Fixed Plan的条件如下。

- `set` 的列不能是主键（PK）。
- where条件里有且只能有全部的PK。
- 可以使用 `pk in (?,?,?)` 或 `pk = ANY()` 一次修改多条。示例：`pk1 in (1,2) and pk2 = any('{3,4}') and pk3 = 5`，改 `(1,3,5),(1,4,5),(2,3,5),(2,4,5)` 四条。
- where条件里同一列只能有一个条件（一模一样的视为一个条件）。

使用示例如下。

```
begin;
create table test_update(
  pk1 int,
  pk2 int,
  col1 int,
  col2 int,
  primary key(pk1, pk2)
);
commit;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_update =on;
update test_update set col1 = 1, col2 = 2 where pk1 = 3 and pk2 = 4;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_update =on;
update test_update set col1 = 1 where pk1 = 3 and pk2 = 4;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_update =on;
update test_update set col1 = 1, col2 = 2 where pk1 in (1,2) and pk2 = any('{3,4}');
--pk1多个过滤条件，不支持Fixed Plan
update test_update set col1 = 1, col2 = 2 where pk1 = 3 and pk1 = 4;
--pk1多个过滤条件，不支持Fixed Plan
update test_update set col1 = 1, col2 = 2 where pk1 in (1,2) and pk1 = 1;
--pk1多个过滤条件，但过滤条件相同，支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_update =on;
update test_update set col1 = 1, col2 = 2 where pk1 in (1,2) and pk1 in (1,2) and pk2 =4;
```

DELETE场景

- Delete表达式

  Delete时能进行Fixed Plan的表达式如下。

  ```
  set hg_experimental_enable_fixed_dispatcher_for_delete =on;
  delete from table where pk1 = ? and pk2 = ? and pk3 = ?;
  ```

  - 需要配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_delete=on;` 。
  - 支持Delete内部表，不支持外部表。
  - 支持分区子表，不支持分区父表。
  - 表必须有主键（PK）。

- Delete场景使用

  Delete场景支持进行Fixed Plan的条件如下。

- where条件里有且只能有全部的PK。
- 可以使用 `pk in (?,?,?)` 或 `pk = ANY()` 一次删除多条。示例：`pk1 in (1,2) and pk2 = any('{3,4}') and pk3 = 5` ，删除 `(1,3,5),(1,4,5),(2,3,5),(2,4,5)` 四条。
- 同一列只能有一个条件（一模一样的视为一个条件）。

使用示例如下。

```
begin;
create table test_delete(
  pk1 int,
  pk2 int,
  col1 int,
  col2 int,
  primary key(pk1, pk2)
);
commit;
--支持Fixed Plan，更多场景与Update样例一致
set hg_experimental_enable_fixed_dispatcher_for_delete =on;
delete from test_delete where pk1 = 1 and pk2 = 2;
```

SELECT场景

- SELECT表达式

  SELECT时能进行Fixed Plan的表达式如下。

  ```
  select col1,col2,col3,... from table where pk1 = ? and pk2 = ? and pk3 = ?;
  ```

  - 支持Select内部表，不支持外部表。
  - 支持分区子表，不支持分区父表。
  - 表必须有主键（PK）。

- 点查(key/value)场景

  点查场景支持的情况如下。

  - where条件里有且只能有全部的PK。
  - 可以使用 `pk in (?,?,?)` 或 `pk = ANY()` 一次查多条。示例：`pk1 in (1,2) and pk2 = any('{3,4}') and pk3 = 5` ，查 `(1,3,5),(1,4,5),(2,3,5),(2,4,5)` 四条。
  - 同一列只能有一个条件（一模一样的视为一个条件）。
  - 如果有limit，limit的值必须 `>0` 。

  使用示例如下。

```
begin;
create table test_select(
  pk1 int,
  pk2 int,
  col1 int,
  col2 int,
  primary key(pk1, pk2)
);
commit;
--支持Fixed Plan
select * from test_select where pk1 = 1 and pk2 = 2;
```

- PrefixScan场景
  - PrefixScan场景表达式

    PrefixScan场景是指表有多个主键，查询时按照左匹配原则只查几列主键，查询表达式如下。

    ```
    set hg_experimental_enable_fixed_dispatcher_for_scan = on;
    select col1,col2,col3,... from table where pk1 = ? and pk2 = ?;
    select col1,col2,col3,... from table where pk1 = ? and pk2 > ? and pk 2 < ?; --从1.1.48版本开始支持pk最后一列条件为范围
    select col1,col2,col3,... from table where pk1 = ? and pk2  between ? and ?; --从1.1.48版本开始支持pk最后一列条件为范围
    ```

    - 需要配置GUC参数：`hg_experimental_enable_fixed_dispatcher_for_scan=on;`，且实例在V1.1.24及以上版本 。

      > **说明** PrefixScan一次性返回所有结果行，如果结果的字节数大于`hg_experimental_fixed_scan_bytesize_limit`会报错：`scan result size larger than fixed scan size limit`，可以通过配置`hg_experimental_fixed_scan_bytesize_limit`参数设置更符合场景的值，默认值为1048576，即1MB。

    - PrefixScan从V1.1.48版本开始支持主键最后一列条件设置为范围。
    - 表必须有Distribution Key。
  - PrefixScan使用

    PrefixScan的使用条件如下。

    - where里有且只有PK的Prefix。

      > **说明** Prefix定义：若PK为(pk1,pk2,pk3)，则(pk1),(pk1,pk2)为Prefix。

    - where里必须包含所有的Distribution Key。若表PK为`(pk1,pk2,pk3,pk4)`，则Distribution Key为`pk1,pk3`。

    使用示例如下。

```
begin;
create table test_select_prefix(
  pk1 int,
  pk2 int,
  pk3 int,
  pk4 int,
  primary key(pk1, pk2,pk3,pk4)
);
call set_table_property('test_select_prefix', 'distribution_key', 'pk1,pk3');
commit;
--没有包含所有distribution key，不能走fixed plan
select * from test_select_prefix where pk1 = ? and pk2 = ?;
--不是pk的prefix，不能走fixed plan
select * from test_select_prefix where pk1 = ? and pk3 =?;
--可以走fixed plan
 set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_select_prefix where pk1 = ? and pk2 = ? and pk3 = ?;
```

可以使用 `pk in (?,?,?)` 或 `pk = ANY()` 一次性查多条，命令如下。

```
pk1 in (1,2) and pk2 = 3 <=> scan(1,3),(2,3)两组
pk2 =any('{3,4}') and pk1 in (1,2) <=> scan(1,3),(1,4),(2,3),(2,4)四组
```

- 同一列只能有一个条件（一模一样的视为一个条件）。
- 如果含有limit条件，limit的值必须>0。

使用示例如下。

```
begin;
create table test_scan(
  pk1 int,
  pk2 int,
  pk3 int,
  col1 int,
  primary key(pk1, pk2, pk3)
);
CALL SET_TABLE_PROPERTY ('test_scan', 'distribution_key', 'pk1,pk2');
commit;
INSERT INTO test_scan values (1,2,3,4);
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_scan where pk1 = 1 and pk2 = 2;
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_scan where pk1 = 1 and pk2 in (2,3);
--支持Fixed Plan
set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_scan where pk1 = ANY('{3,4}') and pk2 in (2,3);
--支持fixed plan,pk最后一列是range条件，需要1.1.48及以上版本支持
set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_scan where pk1 = 1 and pk2 = 1 and pk3 > 1 and pk3 < 3;
--支持fixed plan,pk最后一列是range条件，需要1.1.48及以上版本支持
set hg_experimental_enable_fixed_dispatcher_for_scan = on;
select * from test_scan where pk1 = 1 and pk2 = 1 and pk3 between 1 and 3;
--不包含所有的distribution key，不支持Fixed Plan
select * from test_scan where pk1 = 1;
--不符合主键前缀Prefix，不支持Fixed Plan
select * from test_scan where pk2 = 2;
```

验证Fixed Plan

- 通过FixedPlan执行的更新类SQL，在控制台的实时导入RPS面板中会显示为SDK类型，包括INSERT、UPDATE和DELETE类型的操作。建议实时写入类Insert、Update、Delete都尽量优化为Fixed Plan方案，改善数据更新的效率。
- 通过查看SQL执行计划（explain sql），如果返回的执行计划中含有 `FixedXXXNode` ，既表示触发了Fixed Plan，如下图所示。如未生成含有 `FixedXXXNode` 的执行计划，请对照上文场景支持条件， 查看是否满足条件。

```
test=# explain insert into t values(1,2) on conflict(id1) do update set id2 = excluded.id2;
                    QUERY PLAN
---------------------------------------------------
 FixedInsertNode  (cost=0.00..0.00 rows=0 width=0)
 Optimizer: HQO version 1.1.0
(2 rows)
```

性能调优

在某些场景上若是已经开启Fixed Plan但还需要做性能调优时，可选择如下方式。

- Hologres V1.1.49版本开始针对Fixed Plan点查场景进行了优化，在大规模点查的情况下提升了30%以上吞吐。若有需要请升级实例至V1.1.49及以上版本。
- 客户端合理的攒批（使用Holo Client会自动攒批），即一次执行SQL命令的数量，实践证明数量为512或者512的倍数性能会更好。

6.7. 高级功能

6.7.1. Hologres Binlog

6.7.1.1. 订阅Hologres Binlog

本文将会为您介绍如何订阅Hologres Binlog。

使用限制

在Hologres中订阅Hologres Binlog需要注意如下事项：

- 仅Hologres V0.9及以上版本支持订阅Hologres Binlog，如果您的实例是V0.9以下版本，请您提交工单或加入在线支持钉钉群申请升级实例，加入在线支持钉钉群请参见如何获取更多的在线支持？。
- Hologres V0.9以及V0.10版本，已存在的表无法修改表属性开启Binlog，需重新建表。从V1.1版本开始，可以按需开启Binlog。
- 仅支持Superuser消费Binlog，如果使用其他低权限账号消费Binlog会出现没有权限的报错：`permission denied for table hg_replication_slot_properties`。
- Hologres支持单表级别的Binlog功能，支持行存表和列存表。当前订阅Hologres Binlog支持的情况如下表。

Flink分类	Hologres行存表Binlog	Hologres列存表Binlog	Hologres行列共存表Binlog（从Hologres V1.1版本开始支持）
实时计算Blink	支持	支持	支持
全托管Flink	支持	支持	支持
开源Flink	不支持	不支持	不支持
JDBC	Hologres V1.1版本开始支持	Hologres V1.1版本开始支持	Hologres V1.1版本开始支持

- Blink消费Hologres Binlog暂不支持Hologres的TIMESTAMP类型，在Hologres建表时，请使用TIMESTAMPTZ类型。同时也不支持SMALLINT等特殊类型。
- 不支持消费分区表父表的Binlog，请使用普通表（非分区表）。
- 对于更新频繁的场景，理论上列存表开启Binlog的开销要大于行存表的开销，所以建议使用行存表开启Binlog。

开启Binlog

Hologres中，Binlog功能默认关闭，您可以通过设置表属性binlog.level和binlog.ttl开启该功能。如下示例开启Binlog，更多关于创建表的参数说明，请参见CREATE TABLE。

> **说明** 理论上列存表开启Binlog功能的成本要大于行存表。如果您对表格的更新比较频繁，建议您使用行存表开启Binlog功能。

```
begin;
create table test_message_src(
  id int primary key,
  title text not null,
  body text);
call set_table_property('test_message_src', 'orientation', 'row');--创建行存表test_message_src
call set_table_property('test_message_src', 'clustering_key', 'id');--在id列建立聚簇索引
call set_table_property('test_message_src', 'binlog.level', 'replica');--设置表属性开启Binlog功能
call set_table_property('test_message_src', 'binlog.ttl', '86400');--binlog.ttl，Binlog的TTL，单位为秒
commit;
```

参数说明如下。

参数	说明
binlog.level	是否开启Binlog，可选择： • replica：开启。 • none：关闭。
binlog.ttl	Binlog的TTL，单位秒。默认为30天，即默认值为2592000。

按需开启Binlog

Hologres V0.10版本开始支持Binlog，但是在使用上有一定的限制，已经创建的表需要重新建表才能开启Binlog，操作比较复杂。从Hologres V1.1版本开始，可以根据业务需要选择开启/关闭Binlog能力，同时支持配置TTL满足不同业务场景对Binlog保留时间的诉求，已有表无需重新建表就能开启Binlog，操作方便快捷。

> **说明** 以下功能仅针对Hologres V1.1及以上版本，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。

- 开启Binlog

  可以使用以下语句对已有表开启Binlog并设置Binlog TTL时间。

  ```
  -- 设置表属性开启Binlog功能
  begin;
  call set_table_property('<table_name>', 'binlog.level', 'replica');
  commit;
  -- 设置表属性，配置Binlog TTL时间，单位秒
  begin;
  call set_table_property('<table_name>', 'binlog.ttl', '2592000');
  commit;
  ```

  table_name为开启Binlog的表名称。

- 关闭Binlog

  可以使用以下语句对已开启Binlog的表关闭Binlog。

```
-- 设置表属性关闭Binlog功能
begin;
call set_table_property('<table_name>', 'binlog.level', 'none');
commit;
```

table_name为需要关闭Binlog的表名称。

- 修改Binlog的TTL

  通过以下语句可以对已开启Binlog的表修改TTL，满足业务对Binlog不同保留时间的诉求。

```
begin;
call set_table_property('<table_name>', 'binlog.ttl', '8640000'); --单位秒
commit;
```

table_name为修改Binlog TTL的表名称。

Binlog格式说明

Binlog字段由Binlog系统字段和用户Table字段组成，具体字段定义如下表。

字段名称	字段类型	说明
hg_binlog_lsn	BIGINT	Binlog的系统字段，表示Binlog序号。Shard内部单调递增不保证连续，不同Shard之间不保证唯一和有序。
hg_binlog_event_type	BIGINT	Binlog的系统字段，表示当前Record所表示的修改类型。
hg_binlog_timestamp_us	BIGINT	Binlog的系统字段，系统时间戳，单位为us。
user_table_column_1	用户自定义	用户Table字段。
...	...	...
user_table_column_n	用户自定义	用户Table字段。

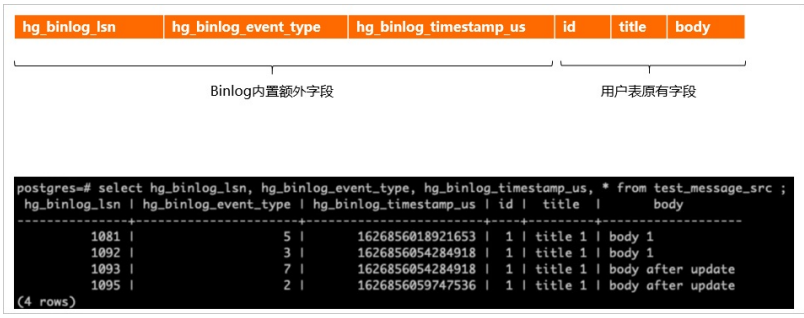
- hg_binlog_event_type有如下五种可能的取值：
  - INSERT=5，表示当前Binlog为插入一条新的记录。
  - DELETE=2，表示当前Binlog为删除一条已有的记录。
  - BEFORE_UPDATE=3，表示当前Binlog为一条已有记录其更新前的记录。
  - AFTER_UPDATE=7，表示当前Binlog为一条已有记录其更新后的记录。
  - HEARTBEAT_LOG_EVENT= 27，表示当前Shard截止到hg_binlog_timestamp_us的Binlog均已消费完毕（仅当JDBC和Holo-Client消费Binlog，且设置GUC：`hg_experimental_enable_binlog_heartbeat_record=on` 时有效），详情请参见通过JDBC消费Hologres Binlog（Beta）。
- UPDATE操作会产生两条Binlog记录，分别为更新前和更新后的记录。订阅Binlog功能会保证这两条记录是连续的且更新前的Binlog记录在前，更新后的Binlog记录在后。
- 用户字段的顺序与DDL定义的顺序一致。

查询Binlog

Hologres表的Binlog是一种强Schema格式的数据，如果要查询某个表的Binlog，可以使用Binlog内置额外字段与表原有字段组合的方式查询Binlog。查询test_message_src表Binlog的示例如下。

```
select hg_binlog_lsn,hg_binlog_event_type,hg_binlog_timestamp_us,* from test_message_src;
```

查询结果示例如下。



实时消费Hologres Binlog

当前支持通过Flink、Blink以及JDBC（包括Holo Client）消费Hologres Binlog，详情请参见文档：

- Flink和Blink实时消费Binlog，请参见Flink/Blink实时消费Hologres Binlog。
- 有关JDBC如何实时消费Binlog，请参见通过JDBC消费Hologres Binlog（Beta）。

6.7.1.2. Flink/Blink实时消费Hologres Binlog

本文将会为您介绍如何通过Flink和Blink实时消费Hologres Binlog。

注意事项

消费Hologres Binlog需要注意如下事项：

- 仅Hologres V0.9及以上版本支持消费Hologres Binlog，如果您的实例是V0.9以下版本，请您提交工单或加入在线支持钉钉群申请升级实例，加入在线支持钉钉群请参见如何获取更多的在线支持？。
- Hologres支持单表级别的Binlog功能，支持行存表和列存表，以及从Hologres V1.1版本开始支持行列混存表。
- Hologres Binlog的支持情况以及开启、配置Hologres Binlog，请参见订阅Hologres Binlog。
- 当前版本暂不支持配置引擎白名单，开启白名单后，会造成Binlog消费失败。
- Binlog消费目前不支持数组类型，只支持以下数据类型：INTEGER、BIGINT、TEXT、REAL、DOUBLE PRECISION、BOOLEAN、NUMERIC(38,8)、DATE、TIMESTAMPTZ。对不支持的数据类型（如SMALLINT）即使不消费此字段，仍然可能导致作业无法上线。

Flink实时消费Binlog

Flink支持Hologres Connector实时消费Binlog，使用方法如下。

1. 使用DDL语句创建源表
   - 使用语法

- 源表 DDL（非CDC模式）

  该模式下Source消费的Binlog数据是作为普通的Flink数据传递给下游节点的，即所有数据都是作为Insert类型的数据，可以根据业务情况选择如何处理特定 `hg_binlog_event_type` 类型的数据，具体示例如下。

```
create table test_message_src_binlog_table(
  hg_binlog_lsn BIGINT,
  hg_binlog_event_type BIGINT,
  hg_binlog_timestamp_us BIGINT,
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'binlog' = 'true',
  'binlogMaxRetryTimes' = '10',
  'binlogRetryIntervalMs' = '500',
  'binlogBatchReadSize' = '100'
);
```

- 源表 DDL（CDC模式）

  该模式下Source消费的Binlog数据，将根据 `hg_binlog_event_type` 自动为每行数据设置准确的Flink RowKind类型（INTERT、DELETE、UPDATE_BEFORE、UPDATE_AFTER），这样就能完成表的数据的镜像同步，类似MySQL和Postgres的CDC功能。

  > **说明** Hologres Binlog源表（CDC模式）暂不支持定义Watermark。如果您需要进行窗口聚合，您可以采用非窗口聚合的方式，详情请参见不支持定义Watermark，那如何进行窗口聚合？。

  Hologres表开启Binlog后，在Flink中源表（CDC模式）使用如下DDL可以实时消费Binlog。

```
create table test_message_src_binlog_table(
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',
  'tablename'='<yourTablename>',
  'username'='<yourAccessID>',
  'password'='<yourAccessSecret>',
  'endpoint'='<yourEndpoint>',
  'binlog' = 'true',
  'cdcMode' = 'true',
  'binlogMaxRetryTimes' = '10',
  'binlogRetryIntervalMs' = '500',
  'binlogBatchReadSize' = '100'
);
```

- 全增量一体源表

  VVR引擎1.13-vvr-4.0.13版本，Hologres实例0.10及以上版本开始，Hologres Binlog CDC源表支持全增量一体的消费，这种方式会先读取数据库的历史全量数据，并平滑切换到Binlog读取增量数据，详情请参见Hologres源表。

- 参数说明

  在语法示例中，hg_binlog_xxx开头的三个字段表示Binlog的系统字段，命名和类型不支持修改。其余字段需要和用户字段一一对应，且必须为小写。更多参数说明如下表所示：

参数名称	是否必填	说明
connector	是	源表类型，值填写为hologres。
dbname	是	读取的Hologres DB名称。
tablename	是	读取的表名称。
username	是	当前阿里云账号的AccessKey ID。您可以单击AccessKey管理，获取AccessKey ID。
password	是	当前阿里云账号的AccessKey Secret。您可以单击AccessKey 管理，获取AccessKey Secret。

参数名称	是否必填	说明
endpoint	是	Hologres对应VPC的区域。您可以进入Hologres管理控制台，获取区域和端口信息。
binlog	是	是否为Binlog source。如果需要消费，需要将binlog参数设置为true。
cdcmode	否	读取Binlog时是否采用CDC模式。如果是CDC模式，需要将cdcmode参数设置为true。
binlogMaxRetryTimes	否	读取Binlog出错重试次数，默认为60次。
binlogRetryIntervalMs	否	读取Binlog出错重试间隔，默认为2000ms。
binlogBatchReadSize	否	读取Binlog批量大小，默认为16个。
startTime	否	启动位点的时间。如果没有设置该参数，且作业没有从状态恢复，则从最早的Binlog开始消费Hologres数据。格式为yyyy-MM-dd hh:mm:ss。 **说明** 只要设置了此参数，binlogStartupMode都会按照 `timestamp` 生效。
binlogStartupMode	否	Binlog消费启动模式，推荐与CDC模式一同使用，取值如下。 ■ timestamp：表示从设置的startTime开始消费Binlog。 ■ initial：表示先全量消费数据，再读取Binlog开始增量消费。 ■ earliestOffset（默认值）：表示从最早的Binlog开始消费。

2. 配置Binlog并发

   Binlog订阅的并发等于Hologres中Table的Shard个数，请执行如下语句查看Shard数。其中 `<tablename>` 请替换为您实际的Table名称。Binlog并发建议执行计划配置，将其并发数与Binlog对应的Hologres中Table的Shard数保持一致。

   ```
   select tg.property_value from hologres.hg_table_properties tb join hologres.hg_table_group_properties tg on tb.property_value = tg.tablegroup_name where tb.property_key = 'table_group' and tg.property_key = 'shard_count' and table_name = '<tablename>';
   ```

3. 消费延迟监控

   支持指标项hologresBinlogEventTimeLag，该指标项可用于Flink消费的延迟状态监控，`0` 表示无延迟。

Blink实时消费Binlog

实时计算Blink 3.7及以上版本，支持Hologres Connector实时消费Binlog，开启方法如下。

1. 使用DDL语句创建源表
   - 使用语法

```
create table test_message_src_binlog_table(
  hg_binlog_lsn BIGINT,
  hg_binlog_event_type BIGINT,
  hg_binlog_timestamp_us BIGINT,
  id INTEGER,
  title VARCHAR,
  body VARCHAR
) with (
  type = 'hologres',
  'endpoint' = 'ip:port',--Hologres的vpc网络地址
  'username' = 'xxxx',--当前账号的AccessKey ID
  'password' = 'xxxx',--当前账号的AccessKey Secret
  'dbname' = 'xxxx',--Hologres的DB名
  'tablename' = 'xxxx',--Hologres的表名
  'binlog' = 'true',
  'binlogMaxRetryTimes' = '10',
  'binlogRetryIntervalMs' = '500',
  'binlogBatchReadSize' = '256'
);
```

   - 参数说明

     在语法示例中，hg_binlog_xxx开头的三个字段表示Binlog的系统字段，命名和类型不支持修改。其余字段需要和用户字段一一对应，且必须为小写。更多参数说明如下表所示：

参数名称	是否必填	说明
type	是	源表类型，值填写为hologres。
dbname	是	读取的Hologres DB名称。
tablename	是	读取的表名称。
username	是	当前阿里云账号的AccessKey ID。您可以单击AccessKey管理，获取AccessKey ID。
password	是	当前阿里云账号的AccessKey Secret。您可以单击AccessKey 管理，获取AccessKey Secret。
endpoint	是	Hologres对应VPC的区域。您可以进入Hologres管理控制台，获取区域和端口信息。
binlog	是	是否为Binlog source。如果需要消费，需要将binlog source设置为true。
binlogMaxRetryTimes	否	读取Binlog出错重试次数，默认为60次。
binlogRetryIntervalMs	否	读取Binlog出错重试间隔，默认为2000ms。
binlogBatchReadSize	否	读取Binlog批量大小，默认为16个。

参数名称	是否必填	说明
startTime	否	启动位点的时间。如果没有设置该参数，且作业没有从状态恢复，则从最早的Binlog开始消费Hologres数据。格式为yyyy-MM-dd hh:mm:ss。

2. 配置Binlog并发

   Binlog订阅的并发等于Hologres中Table的Shard个数，请执行如下语句查看Shard数。其中 `<tablename>` 请替换为您实际的Table名称。Binlog并发建议执行计划配置，将其并发数与Binlog对应的Hologres中Table的Shard数保持一致。

```
select tg.property_value from hologres.hg_table_properties tb join hologres.hg_table_group_properties tg on tb.property_value = tg.tablegroup_name where tb.property_key = 'table_group' and tg.property_key = 'shard_count' and table_name = '<tablename>';
```

6.7.1.3. 通过JDBC消费Hologres Binlog（Beta）

本文为您介绍如何通过JDBC和Holo-Client这两种方式消费Hologres Binlog。

使用限制

- 仅Hologres V1.1及以上版本支持通过JDBC消费Hologres Binlog，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 需提前开启和配置Binlog，详情请参见订阅Hologres Binlog。
- 仅以下数据类型支持消费Hologres Binlog：INTEGER、BIGINT、TEXT、REAL、DOUBLE PRECISION、BOOLEAN、NUMERIC(38,8)、DATE、TIME、TIMETZ、TIMESTAMP、TIMESTAMPTZ、BYTEA、JSON、int4[]、int8[]、float4[]、float8[]、boolean[]、text[]。如果表中有以上类型之外的数据类型，会造成消费失败。
- 消费Hologres Binlog之前，需要实例的Superuser执行以下语句创建extension，extension针对整个数据库生效，一个数据库只需执行一次，新建数据库需要再次执行。

```
--创建extension
create extension hg_binlog;
```

> **说明** 如需卸载extension请执行如下命令。
>
> ```
> DROP extension hg_binlog;
> ```

- 同普通连接类似，在使用JDBC进行Binlog的消费时，所消费的每张表的每个Shard都会使用1个Walsender连接，Walsender连接与普通连接独立，互不影响。
- Walsenders数也有使用上限，可以通过以下命令查看单个Frontend节点的最大Walsender数（V1.1.26版本之后默认为100），总的数目需要乘实例的Frontend节点数，不同规格实例的Frontend节点数请参见实例规格概述。

```
show max_wal_senders;
```

> **说明** Hologres实例支持的同时消费Binlog的表数量可以通过以下方式计算：`表数量<=(max_wal_senders(100) * FrontEnd节点数)/表的Shard Count`。
>
> 例如：
>
> - 表a和表b的Shard Count都为20，表c的Shard Count为30，则三张表同时进行消费Binlog占用的Walsenders数量为 `20 + 20 + 30 = 70`。
> - 表a和表b的Shard Count都为20，表a同时有两个作业在进行Binlog消费，同时进行消费占用的Walsenders数量为 `20 *2 + 20 = 60`。
> - 一个实例有两个Frontend节点，则其最大Walsenders数为 `100*2 = 200`，最多支持同时消费10张Shard Count为20的表同时进行消费Binlog。

Publication和Replication Slot

- Publication
  - 简介：

    本质上是一组表，这些表的数据更改旨在通过逻辑复制进行表中数据复制，详细内容请参见Publication。当前Hologres支持的Publication只支持绑定一张物理表，且该表需要开启Binlog功能。
  - 创建Publication
    - 语法示例

      ```
      CREATE PUBLICATION name FOR TABLE table_name;
      ```
    - 参数说明

参数	说明
name	自定义Publication名称。
table_name	数据库中表名称。
    - 使用示例

      ```
      --示例创建一个名为hg_publication_test_1的Publication，且将表test_message_src添加至该Publication下
      create publication hg_publication_test_1 for table test_message_src;
      ```

- 查询已经创建的Publication
  - 语法示例

```
select * from pg_publication;
```

  - 查询结果

```
        pubname         | pubowner | puballtables | pubinsert | pubupdate | pubdelete
| pubtruncate
------------------------+----------+--------------+-----------+-----------+-----------
+-------------
 hg_publication_test_1 |    16728 | f            | t         | t         | t
| t
(1 row)
```

参数	说明
pubname	Publication名称。
pubowner	Publication拥有者。
puballtables	绑定多个物理表，默认为False，目前暂不支持。
pubinsert	是否发布INSERT类型的Binlog，默认为True，Binlog类型请参考Binlog格式说明。
pubupdate	是否发布UPDATE类型的Binlog，默认为True。
pubdelete	是否发布DELETE类型的Binlog，默认为True。
pubtruncate	是否发布TRUNCATE类型的Binlog，默认为True。

- 查询Publication关联的表
  - 语法示例

```
select * from pg_publication_tables;
```

  - 查询结果

```
        pubname         | schemaname |    tablename
------------------------+------------+------------------
 hg_publication_test_1 | public     | test_message_src
(1 row)
```

参数	说明
pubname	Publication名称。
schemaname	表所属schema的名称。
tablename	表名称。

- Replication Slot

- 简介：

  在逻辑复制场景下，一个Replication Slot表示一个数据的更改流，该Replication Slot也与当前消费进度绑定，用于断点续传，详细内容可以参见Postgres文档Replication Slot。Replicaiton Slot用于维护Binlog消费的点位信息，使得消费端Failover之后可以从之前已经Commit的点位进行恢复。

- 创建Replication Slot
  - 语法示例

```
call hg_create_logical_replication_slot('replication_slot_name', 'hgoutput', 'publication_name');
```

  - 参数说明

参数	说明
replication_slot_name	自定义Replication Slot的名称。
hgoutput	Binlog输出格式的插件，当前仅支持hgoutput内置插件。
publication_name	Replication Slot所绑定的Publication名称。

  - 使用示例

```
--创建一个名称为hg_replication_slot_1的Replication Slot，并且绑定名称为hg_publication_test_1的Publication。
call hg_create_logical_replication_slot('hg_replication_slot_1', 'hgoutput', 'hg_publication_test_1');
```

- 查询已经创建的Replication Slot
  - 语法示例

```
select * from hologres.hg_replication_slot_properties;
```

  - 查询结果

```
       slot_name       | property_key |    property_value
-----------------------+--------------+----------------------
 hg_replication_slot_1 | plugin       | hgoutput
 hg_replication_slot_1 | publication  | hg_publication_test_1
 hg_replication_slot_1 | parallelism  | 1
(3 rows)
```

参数	说明
slot_name	Replication Slot名称。
property_key	包含如下三个参数。 ■ plugin：Replication Slot使用的插件，目前只支持hgoutput。 ■ publication：Replication Slot对应的Publication。 ■ parallelism：Replication Slot的并发数。
property_value	property_key包含参数对应的值。

- 查询Replication Slot的并发数

  Hologres是一个分布式数仓，所以一张表的数据会分布在多个Shard上，所以使用JDBC消费Binlog的时候，需要启动多个客户端连接，才能消费到完整的Binlog数据。通过以下命令可以查询消费hg_replication_slot_1所需要的并发数。

  - 语法示例

    ```
    select hg_get_logical_replication_slot_parallelism('hg_replication_slot_1');
    ```

  - 查询结果

    ```
    hg_get_logical_replication_slot_parallelism
    -----------------------------------------------
                                            20
    ```

- 查询Replication Slot的消费进度

  - 语法示例

    ```
    select * from hologres.hg_replication_progress;
    ```

  - 查询结果

    ```
           slot_name       | parallel_index | lsn
    -----------------------+----------------+-----
     hg_replication_slot_1 |              0 |  66
     hg_replication_slot_1 |              1 | 122
     hg_replication_slot_1 |              2 | 119
    (0 rows)
    ```

参数	说明
slot_name	Replication Slot名称。
parallel_index	并发序号。
lsn	当前消费到最后的Binlog序号。

使用JDBC消费Binlog

1. 添加POM依赖

   使用如下语句添加POM依赖。

   > **说明** 添加POM依赖，请使用42.2.18及以上版本的JDBC。

```
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.2.23</version>
        </dependency>
        <!-- 用于获取表shcema以及解析binlog -->
        <dependency>
            <groupId>com.alibaba.hologres</groupId>
            <artifactId>holo-client</artifactId>
            <version>2.1.1</version>
    </dependency>
```

2. Java代码示例

```
import com.alibaba.hologres.client.HoloClient;
import com.alibaba.hologres.client.HoloConfig;
import com.alibaba.hologres.client.impl.binlog.HoloBinlogDecoder;
import com.alibaba.hologres.client.model.Record;
import org.postgresql.PGConnection;
import org.postgresql.PGProperty;
import org.postgresql.replication.LogSequenceNumber;
import org.postgresql.replication.PGReplicationStream;
import org.postgresql.model.TableSchema;
import java.nio.ByteBuffer;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Arrays;
import java.util.List;
import java.util.Properties;
public class Test {
    public static void main (String[] args) throws Exception {
        String username = "";
        String password = "";
        String url = "jdbc:postgresql://Endpoint:Port/db_test";
        // 创建JDBC连接
        Properties properties = new Properties();
        PGProperty.USER.set(properties, username);
        PGProperty.PASSWORD.set(properties, password);
        PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
        // 消费Binlog，务必加上以下参数
        PGProperty.REPLICATION.set(properties, "database");
        try (Connection connection = DriverManager.getConnection(url, properties)) {
            // 创建PGReplicationStream并绑定Replicaiton slot
            PGConnection pgConnection = connection.unwrap(PGConnection.class);
            PGReplicationStream pgReplicationStream = pgConnection.getReplicationAPI().
replicationStream()
                    .logical()
                    .withSlotName("hg_replication_slot_1")
                    .withSlotOption("parallel_index", "0")
                    .withSlotOption("batch_size", "1024")
                    .withSlotOption("start_time", "2021-01-01 00:00:00")
                    .withSlotOption("start_lsn","0")
                    .start();
            // 创建holo-client
```

```
            HoloConfig holoConfig = new HoloConfig();
            holoConfig.setJdbcUrl(url);
            holoConfig.setUsername(username);
            holoConfig.setPassword(password);
            HoloClient client = new HoloClient(holoConfig);
            // 创建Binlog decoder用于Decode binary数据,schema需要通过HoloClient获取
            TableSchema schema = client.getTableSchema("test_message_src", true);
            HoloBinlogDecoder decoder = new HoloBinlogDecoder(schema);
            // 消费数据
            ByteBuffer byteBuffer = pgReplicationStream.readPending();
            while (true) {
                if (byteBuffer != null) {
                    List<BinlogRecord> records = decoder.decode(byteBuffer);
                    Long latestLsn = 0L;
                    for (BinlogRecord record : records) {
                        latestLsn = record.getBinlogLsn();
                        // Do Something
                        System.out.println( "lsn: " + latestLsn + ", record: " + Arrays
.toString(record.getValues()));
                    }
                    // Commit Binlog 点位信息
                    pgReplicationStream.setFlushedLSN(LogSequenceNumber.valueOf(latestL
sn));
                    pgReplicationStream.forceUpdateStatus();
                }
                byteBuffer = pgReplicationStream.readPending();
            }
        }
        // pgReplicationStream.close();
        // connection.close();
    }
```

创建PGReplicationStream时的withSlotOption可以指定如下参数。

参数	是否必须	说明
withSlotName	是	指定Replication Slot的名称
parallel_index	是	Replication Slot的并发数序号，表示PGReplicationStream消费的是第几个并发的数据。假设某个Replication Slot的并发数是3，则用户最多可以创建3个PGReplicationStream，每个PGReplicationStream的parallel_index参数分别是0、1、2。当前Hologres Binlog并不支持类似Kafka Consumer Group的实现，所以需要用户自己创建多个PGReplicationStream。
start_time	否	表示从某个时间点位开始消费Binlog，示例参数格式为：2021-01-01 12:00:00+08。 如果未指定此参数，分为如下两种情况： ○ 第一次开始消费Replication Slot的Binlog，则从头开始消费，类似Kafka的Oldest。 ○ 曾经消费过Replication Slot的Binlog，则尝试从之前Commit过的点位开始消费。

参数	是否必须	说明
start_lsn	否	表示从某个lsn之后开始消费binlog，同时设置优先级高于start_time
batch_size	否	单次获取的Binlog最大批大小，单位为行，默认值为1024。

> **说明**
> - BinlogRecord 是decoder返回的Record类型，可以通过以下接口获取这条数据对应的binlog系统字段，详情见订阅Hologres Binlog。
>   - getBinlogLsn() 获取binlog的序号。
>   - getBinlogTimestamp() 获取Binlog的系统时间戳。
>   - getBinlogEventType() 获取Binlog的事件类型。
> - 消费Binlog之后，用户需要手动Commit点位信息，确保下次Failover能够恢复。

使用Holo Client消费Binlog

消费Hologres Binlog功能已经集成至Holo Client中，您可以通过指定需要消费的物理表，方便的消费所有parallel_index的Binlog数据。使用Holo Client消费Binlog，需要占用与物理表shard数（slot并发数）相同的连接数，请保证连接数充足。Holo Client消费binlog支持断点续传，由于网络连接失败等原因消费终止，在重新消费时会使用保存在hologres.hg_replication_progress表的消费进度进行恢复。因此消费时每张表的每个slot，只有一个作业在消费。

1. 添加POM依赖

   使用如下语句添加POM依赖。

   > **说明** 添加POM依赖，请使用2.1.0及以上版本的Holo Client。

   ```
   <dependency>
     <groupId>com.alibaba.hologres</groupId>
     <artifactId>holo-client</artifactId>
     <version>2.1.1</version>
   </dependency>
   ```

2. Java代码示例

```
import com.alibaba.hologres.client.BinlogShardGroupReader;
import com.alibaba.hologres.client.HoloConfig;
import com.alibaba.hologres.client.HoloClient;
import com.alibaba.hologres.client.model.Record;
import org.postgresql.model.TableSchema;
import java.util.Arrays;
public class HoloBinlogExample {
    public static void main(String[] args) throws Exception {
        String username = "";
        String password = "";
        String url = "jdbc:postgresql://ip:port/database";
        String tableName = "test_message_src";
        String slotName = "hg_replication_slot_1";
        // 创建client的参数
        HoloConfig holoConfig = new HoloConfig();
        holoConfig.setJdbcUrl(url);
        holoConfig.setUsername(username);
        holoConfig.setPassword(password);
        holoConfig.setBinlogReadBatchSize(128);
        holoConfig.setBinlogIgnoreDelete(true);
        holoConfig.setBinlogIgnoreBeforeUpdate(true);
        HoloClient client = new HoloClient(holoConfig);
        // 消费binlog的请求，tableName和slotname为必要参数，Subscribe有StartTimeBuilder和Of
fsetBuilder两种，此处以前者为例
        Subscribe subscribe = Subscribe.newStartTimeBuilder(tableName, slotName)
                                       .setBinlogReadStartTime("2021-01-01 12:00:00+08"
)
                                       .build();
        // 创建binlog reader
        BinlogShardGroupReader reader = client.binlogSubscribe(subscribe);
        BinlogRecord record;
        while ((record = reader.getBinlogRecord()) != null) {
            //handle record
        }
    }
}
```

（可选）如需要，可以用OffsetBuilder创建Subscribe，从而为每个Shard指定起始消费点位。

```
// 此处shardCount为示例，请替换为所消费表对应的实际数量
int shardCount = 10;
Subscribe.OffsetBuilder subscribeBuilder = Subscribe.newOffsetBuilder(tableName, slotNa
me);
for (int i = 0; i < shardCount; i++) {
    // BinlogOffset通过setSequence指定lsn，通过setTimestamp指定时间，两者同时指定lsn优先级大
于时间戳
    subscribeBuilder.addShardStartOffset(i, new BinlogOffset().setSequence(0).setTimest
amp("2021-01-01 12:00:00+08"));
}
Subscribe subscribe = subscribeBuilder.build();
BinlogShardGroupReader reader = client.binlogSubscribe(subscribe);
```

使用Holo Client消费Binlog时可以指定如下参数。

参数	是否必须	默认值	说明
binlogReadBatchSize	否	1024	从每个Shard单次获取的Binlog最大批次大小，单位为行。
binlogHeartBeatIntervalMs	否	-1	binlogRead 发送BinlogHeartBeatRecord的间隔。-1表示不发送 当binlog没有新数据，每间隔binlogHeartBeatIntervalMs会下发一条BinlogHeartBeatRecord，此record的timestamp表示截止到这个时间的数据都已经消费完成。
binlogIgnoreDelete	否	false	是否忽略Delete类型的Binlog。
binlogIgnoreBeforeUpdate	否	false	是否忽略BeforeUpdate类型的Binlog。
retryCount	否	3	消费失败时的重试次数，成功消费时重试次数会被重置。

6.7.2. Table Group与Shard Count

6.7.2.1. 基本概念

Hologres是一款高性能的、计算存储分离的分布式一站式实时数仓引擎，数据存储在位于底层存储系统的数据分片（又称Shard）上。本文为您介绍Hologres中Table Group和Shard Count的概念。

在Hologres中，一个数据库包含一个或多个Table Group，每个Table Group包含多个表（table），一个表只能属于一个Table Group。一个Table Group唯一对应一组shard，由这组shard来负责其中表的数据存储和查询，其包含的shard个数称为Shard Count，Table Group一旦建立，shard数不可调整。

合理的Table Group选择与Shard Count制定，是写入和查询效率的基石，能够从根本上改善数据的存储与计算效率；反之，如果Table Group和Shard Count制定不当，很容易出现性能不如预期的情况，且无法根本上调优到最佳性能。

什么是Table Group?

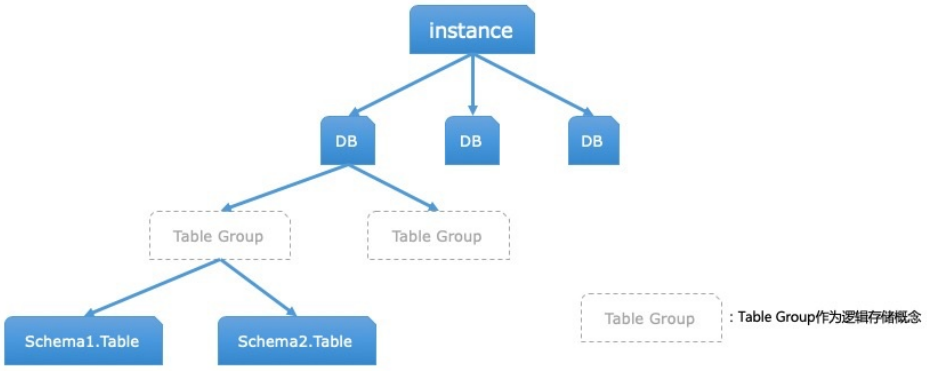
一个表的数据将会存储在固定的一组shard上，数据写入时会按照Distribution Key，将数据Sharding到具体的shard。从创建table开始，负责存储table数据的这一组shard就已经分配好了，这一组shard称为一个Table Group，直到Table Group被删除或者表被移动到另外的Table Group，负责table底层存储的shard始终是最初分配的那些shard。在使用时有以下几个基本原则：

- 多个table可以位于同一组shard，即多个table的数据可以存储在同一组shard上（Table Group）。
- 一个Table Group必须有1张或多张表，如果Table Group无表，则Table Group会自动删除。
- 一个表只能属于一个Table Group，除非重建表或显式调用迁移Table Group函数，否则无法变更表所属的Table Group。

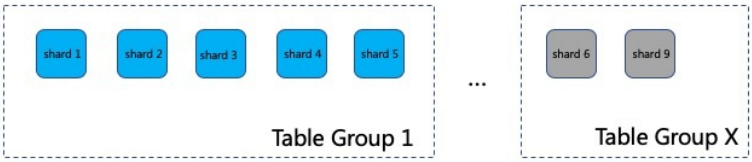
Table Group是Hologres特有的一个存储逻辑概念（PostgreSQL无此概念）。Table Group与PostgreSQL中的TABLESPACE是不一样的：TABLESPACE唯一标识了数据库对象的存储位置，类似一个目录的概念。而Table Group代表的是底层的逻辑shard组。

通过以下几个图直观了解Table Group。

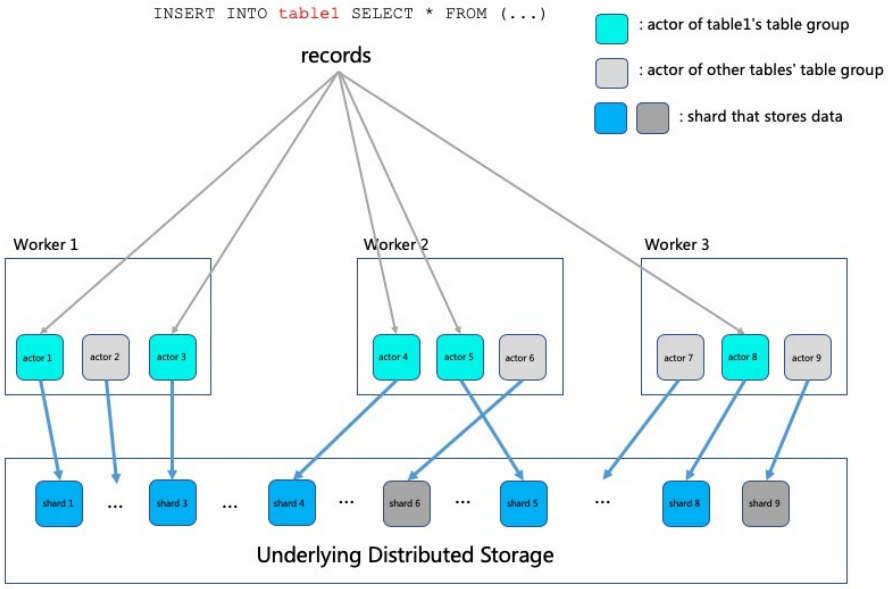
- Table Group与Schema的概念区别：Schema是一个标准的数据库概念，而Table Group是一个逻辑存储概念，并非数据库标准。不同Schema下的表可以位于同一个Table Group，即底层使用同一组shard存储。



- 不同Table Group之间的关系：Table Group 1有五个shard，Table Group X有两个shard count，两个Table Group的shard互不相交，每个shard在实例级别拥有独特的编号。



- Hologres底层的计算存储原理示意图：Hologres为计算存储分离架构，数据本身不存储在计算Worker上，而是存储在分布式存储上。因此每个Hologres计算Worker都有若干的actor，每个actor唯一对应一个真实存储shard，actor负责一对一地对应shard的数据读取、写入和管理。正因为是一对一的，所以Table Group可以认为是一组actor的集合，也可以认为是一组shard的集合，没有本质区别。



什么是Shard Count?

一个Table Group拥有的shard数量（即Shard Count）是它的一个重要属性，Shard数在创建Table Group时需要指定，后续无法更改，只能通过重新创建新的Table Group来更改Shard数。

Shard数多的Table Group，其数据写入和查询分析处理可以得到更大的并行度。一定范围内，增大Shard数可以加快数据写入和查询分析的速度。但Shard数也并非越多越好，更多的Shard数需要更多的节点间通信资源、计算资源以及内存资源，在资源不满足的时候，或者Query很小时可能会导致适得其反的效果。

在Hologres中，Shard数下限是1，在数据量只有几百几千条等很小的情况下，可以设置shard数为1。Shard数上限原则上是实例的总计算Core数（实例总计算Core数约等于实例总Core数的60%，有一部分资源用在前端请求处理、外表查询、集群管理和元数据管理等进程上），这样是为了保证每个Shard在计算时，至少可以占据1个core用于计算。如果Shard数超过计算Core数，那么运行查询时，将有部分Shard无法一直分到CPU资源，可能带来长尾和切换开销。

除Shard数量外，Table Group本身的数量，也不是越多越好。每个Shard无论是否正在使用，都会占据一定的内存空间，用于存放MemTable，Schema等信息，在表有写入时，则会占据更多。因此，如果Table Group越多，则实例内总Shard数越多，内存空间占用越大。另外，多个表之间有一些特殊的关系（例如需要local join）时，这些表必须要处在同一Table Group下才行。

从磁盘上来看，Shard数越多，对于同样的一张表，数据会分的越散，越容易出现小文件，从而文件个数更多，如果表多，并且shard也多，那文件数量就会非常庞大。在查询时、failover时都需要更多的开销，造成查询IO增多，恢复时间变长。

默认Shard Count数

Hologres实例的每个数据库（DB）都会创建一个默认Table Group：数据库创建后，当DB中第一个表被创建，并且没有显示指定表所在的Table Group信息时，Hologres会自动创建出一个默认Table Group，名为{DBname}_tg_default。此后，用户建表时如果不显示指定Table Group，表都将被放置在默认的Table Group中。

下表为默认Table Group对应的默认Shard数，这些默认Shard数，是经过大规模实验验证的最佳参数，能满足大部分场景下的数据量范围和查询种类。

实例规格（Core）	默认Shard数	建议单实例最大Shard数（通过多个TG实现）
32	20	20
64	40	40
96	60	80
128	80	100
160	80	120
192	80	160
256	120	220
400	160	360
512	160	460

但需要特别注意的是，当实例发生升降配时，已有DB的默认Table Group不会改变，其Shard数也不会改变。新建DB时，默认Table Group的Shard数将会变成新规格的默认shard数（如上表所示）。实践表明，无特殊情况，在扩容资源5倍以下的场景中，不需要更改原DB的默认shard数。

6.7.2.2. Table Group与Shard Count操作指南

在Hologres中合理的Table Group选择与Shard Count制定能够从根本上提升数据的存储与计算效率。本文将为您介绍在Hologres中设置Table Group与Shard Count。

获取Table Group元数据

1. 查看默认Table Group

```
SELECT * FROM hologres.hg_table_group_properties
WHERE tablegroup_name IN (
  SELECT tablegroup_name FROM hologres.hg_table_group_properties
  WHERE property_key = 'is_default_tg' AND property_value = '1'
);
```

结果：

> **说明** 结果中is_default_tg代表为默认Table Group，shard_count代表Table Group对应的Shard数。

```
tablegroup_name |   property_key    | property_value
-----------------+-------------------+----------------
 test_tg_default | tg_version        | 1
 test_tg_default | table_num         | 1
 test_tg_default | is_default_tg     | 1
 test_tg_default | shard_count       | 3
 test_tg_default | replica_count     | 1
 test_tg_default | created_manually | 0
(6 rows)
```

2. 查看当前数据库有哪些Table Group

```
SELECT tablegroup_name
FROM hologres.hg_table_group_properties GROUP BY tablegroup_name;
```

结果：

```
tablegroup_name
-----------------
 test_tg_default
(1 row)
```

3. 查看某Table Group设置的Shard数

```
SELECT property_value AS shard_count
FROM hologres.hg_table_group_properties
WHERE property_key = 'shard_count' AND tablegroup_name ='<tg_name>';
```

结果：

```
shard_count
-------------
 3
(1 row)
```

4. 查看某Table Group下有哪些表

```
SELECT table_namespace AS schema_name, table_name
FROM hologres.hg_table_properties
WHERE property_key = 'table_group' AND property_value = '<tg_name>';
```

结果：

```
schema_name | table_name
-------------+------------
 public      | a
(1 row)
```

5. 查看某张表所在的Table Group

```
SELECT property_value AS table_group_name
FROM hologres.hg_table_properties
WHERE property_key = 'table_group' AND table_name = '<table_name>';
```

结果：

```
table_group_name
------------------
 test_tg_default
(1 row)
```

新建Table Group

若是有新上的业务或者需要重新创建Table Group并指定新的Shard数，可以使用以下命令语句。

> **说明**
> - 新建Table Group后，原来的表和数据还会在原Table Group中，不会默认迁移到新的Table Group中。
> - 需要将原来表和数据都迁移至新的Table Group或者删除，原Table Group才会失效。

```
CALL HG_CREATE_TABLE_GROUP ('<new_tg_name>', <shard_count>);
```

参数说明如下：

参数	类型	说明
new_tg_name	Text	新建的Table Group名。
shard_count	INT4	Table Group对应的Shard数。

使用示例：

```
-- 新建一个Shard数为8的新Table Group，命名为tg_8
CALL HG_CREATE_TABLE_GROUP ('tg_8', 8);
```

修改默认Table Group

新建一个数据库后，实例会有一个默认Table Group，以及默认的Shard数。详情请参见实例规格概述。若是数据库中有多个Table Group，想要修改默认Table Group，使得后续新建的表存放于新的Table Group中，可以使用如下命令语句。

> **说明** Hologres实例 V0.9及以上版本执行以下命令语句修改默认Table Group，0.9以下版本请先升级到更高版本。

```
CALL HG_UPDATE_DATABASE_PROPERTY ('default_table_group', '<tg_name>');
```

参数说明：

参数	类型	说明
tg_name	TEXT	默认Table Group名称，设置后，其Shard count为设置的Table Group的Shard数。

使用示例：

```
-- 将新创建TG设为默认Table Group，后续新建的表，将默认使用新的Table Group (v0.9以上)
CALL HG_UPDATE_DATABASE_PROPERTY ('default_table_group', 'tg_8');
```

将新建表放入指定Table Group

可以通过以下命令语句将新建表，显示放入指定的Table Group中。

```
BEGIN;
CREATE TABLE table_name (
    col type,
  ....
);
CALL SET_TABLE_PROPERTY('table_name', 'table_group', '<tg_name>');
COMMIT;
```

参数说明：

参数	类型	说明
table_name	TEXT	新建的表名。
tg_name	TEXT	默认Table Group名称，设置后，其Shard Count为设置的Table Group的Shard数。

使用示例：

```
-- 新建表tbl1并直接关联名为tg_8的Table Group
BEGIN;
CREATE TABLE tbl1 (
    col1 text
);
CALL SET_TABLE_PROPERTY('tbl1', 'table_group', 'tg_8');
COMMIT;
```

（Resharding）迁移表至新建Table Group

在Hologres中，Shard用于提升数据处理的并发度，合理的设置Shard数，有利于提高查询或者写入的性能。一般情况下，Hologres实例默认的shard数已经能满足大部分场景，无需再手动修改。

当实例扩容后，例如32core扩容到128core，该数据库的Shard数不会随着扩容默认更改，因此建议您针对该数据库增加Shard数，以获取更好的性能。如果是该实例下新建的DB，其Shard数为当前规格的默认数量。实例规格与Shard的相关描述，请参见实例规格概述。

当实例扩容或者缩容后，其之前DB的Shard数不会自动随之增加或减少，需要通过命令语句设置Shard数，并重新进行数据导入。Resharding功能用于修改Shard数后，自动实现Rebalance的功能，无需再重新建表导数据，即可将原来的表和数据Resharding到各Shard上，简化操作步骤，实现最优性能。

- 使用限制
  - 仅Hologres V0.10 及以上版本支持Resharding，请前往Hologres管控台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
  - 目前Resharding是单表级别，当使用Resharding时，需要停止该表的写入（如Flink、数据集成等写入），对表的查询不受影响。

- Resharding会消耗CPU资源，且在Resharding过程中会导致存储增加，建议在业务低峰期处理。
- 如果表开启了Binlog，请在Resharding之前关掉Binlog，待Resharding完成后再开启Binlog，详细操作请参见订阅Hologres Binlog。
- 暂不支持表字段带有Serial自增序列以及Default值进行Resharding，带Serial字段的表在Resharding时会报错，带Default字段的表Resharding后会丢失Default属性。
- 表在Resharding时不能有其他的依赖比如view等，若存在请在Resharding前删除相关依赖。

- 语法示例

  可以通过以下命令将业务的部分表迁移至新建的Table Group。

  > **说明**
  > - 在迁移之前，需要有一个新建的Table Group，若是没有新建Table Group，请参见新建Table Group进行新建。
  > - 表迁移时需要停止对该表的写入，查询不受影响。
  > - 原Table Group的表全部迁移后，原Table Group将会被自动删除。若是因为业务需求需要建立多个Table Group，建议合理设置每个Table Group的Shard数。
  > - 表迁移时分区表只需要操作父表即可。

```
-- V1.1及以上版本命令语法
CALL HG_MOVE_TABLE_TO_TABLE_GROUP('<table_name>','<new_table_group_name>');
-- V.10版本命令语法
CALL HG_UPDATE_TABLE_SHARD_COUNT('<table_name>','<new_table_group_name>');
```

- 参数说明

参数	说明	示例
table_name	需要迁移的表名。一次命令仅支持单表迁移，若是有多个表，需要多次执行。分区表只需要操作父表即可。	new_table
table_group_name	新的Table Group名。	new_tg

删除Table Group

可以通过以下命令语句删除空的Table Group。如果Table Group有表存在，则无法删除。

```
CALL HG_DROP_TABLE_GROUP('<tg_name>');
```

使用示例：

```
--删除名为tg_8的Table Group
CALL HG_DROP_TABLE_GROUP('tg_8');
```

使用最佳实践

Table Group是比较高级的功能，一般情况下，不建议业务新建Table Group以及修改Shard数。若是因为业务有不同的需求，可以按照最佳实践合理规划，详情请参见Table Group设置最佳实践。

常见问题

Resharding是个比较复杂的过程，涉及到创建临时表、修改原始表只读状态、写入目标表、更换表名称和记录同步状态等多个环节，如果中间环节因为某些原因出现异常，可能造成系统状态不确定等问题，可通过以下方法进行排查。

当出现 `internal error: Get rundown is not allowed in recovering state.` 异常时，说明当前正在更新的表处于Read Only状态，不能执行Insert、Update、Delete等操作，此时需要退出Read Only状态。

1. 执行如下命令检索当前处于Read Only状态的表。

   ```
   select * from hologres.hg_table_properties where property_key ='readonly' and property_value='true';
   ```

2. 执行如下命令退出Read Only状态。

   ```
   call set_table_property('<table_name>','readonly','false');
   ```

   table_name为需要退出Read Only状态的表名称。

6.7.2.3. Table Group设置最佳实践

本文为您介绍Hologres中Table Group和Shard Count选定的基本原则、适用场景、设置策略以及常见问题，以助您获取更优的查询性能。

注意事项

Hologres拥有灵活指定Shard和Table Group的特性，相比一些同类产品，具备更加灵活、易用而方便的根据具体场景定制化Schema的能力，更加灵活应对业务需求，加上用户对自身业务的理解，能够高效地、充分地利用好Hologres的高性能。推荐设置Table Group基本原则如下：

- 如无必要，不要新建Table Group。
- 数据量过大时，可新建独立的较大Shard数的Table Group。
- 如果有大量数据量很小的表，可独立出一个小Shard数的Table Group，减小Query启动开销，减少小文件。
- 需要Join的表，尽量放在同一个Table Group，不在同一个Table Group的表，不具备Local Join优化。
- Shard数应与实例计算节点数成比例关系，让Shard均衡分布在各个计算节点中，不成比例关系，会造成部分计算节点承担更多的Shard读写职责，从而造成实例计算节点间负载倾斜，引起部分单点的不稳定。

设置Table Group强依赖于自身业务的数据规模、访问特性、资源规格和使用重心等。例如，如果您的Hologres实例只用于对新建外部表加速查询MaxCompute的场景，那么针对内部表的Table Group等概念无法适用。

适用场景

在Hologres实例中，一个数据库（DB）默认一个Table Group，不同的实例规格默认Table Group配置不同的Shard Count，详情请参见实例规格概述。这些默认Shard Count，是经过大规模实验验证的最佳参数，能满足大部分场景，如果没有特殊需求，不建议新建Table Group以及修改Shard Count。

本文最佳实践适用于对Hologres原理、架构有一定了解，且编程开发经验较为丰富的用户，主要适用修改Table Group和Shard Count的场景如下：

1. 在一些特殊需求下，例如写入数据量太大，或者数据量过小而分析发现Query启动开销太大（启动开销和shard数呈正相关关系）、或者Query shuffle开销太大等场景，建议您新建Table Group。
2. Table Group一个非常关键的作用就是做Local Join从而大大提升Join效率，因此，当您有表需要互相

Join，且Join Key为Distribution Key时，将Join的左右表放在同一Table Group即可获得Local Join的效果，产生比较大的性能提升。

实例规格推荐

在实践过程中存在数据量可预估，最适宜的Shard数区间应该设置为多少的问题，由于最适宜Shard数不仅和数据存储量有关，还和实际访问频率、实际数据访问量、计算负载的类型（点查、分析等）、写入吞吐、Table Group上表的个数等因素有关，该问题无法给出准确答案。您可参见下表中根据数据量估算的所需Shard数和实例规格的推荐数，选择适合您的参数配置。

数据总规模	推荐Shard数	推荐规格
4000万行以下	10 ~ 20	32Core以上
4000万行 ~ 4亿行	20 ~ 40	64Core以上
4亿行 ~ 40亿行	40 ~ 80	128Core以上
40亿行 ~ 400亿行	80 ~ 240	256Core以上，考虑创建新的Table Group
400亿行 ~ 4000亿行	160 ~ 400	512Core以上，通常配置多个Table Group

说明 上表根据数据量估算的所需Shard数和实例规格的推荐数不是唯一标准，小数据量的表也可以放在多的Shard Count之上，大数据量的表也可以放在单个Shard上。请您根据实际业务场景选择一个合适的Shard Count，既满足有较高的并发度，带来更高计算效率，又满足数据较集中，从而避免不必要的shuffle开销。

您可以根据以下Table Group规划的最佳实践，来决定是否需要新建以及如何放置Table Group。

- 规划一：使用默认Table Group

  如果您使用Hologres满足下列条件，建议您直接使用默认Table Group即可。Hologres实例升配或降配后，默认Table Group的Shard数不会变，因此可以通过如下命令语句，查看Table Group的Shard数。

```
SELECT * FROM hologres.hg_table_group_properties;
--结果示例
 tablegroup_name | property_key  | property_value
-----------------+---------------+----------------
 test_tg_default | is_default_tg | 1
 test_tg_default | shard_count   | 40
 test_tg_default | tg_version    | 1
 test_tg_default | table_num     | 1
(4 rows)
```

  - 数据量：

    当前默认Table Group的Shard数，符合目前数据量大小的需求。可以使用默认Table Group直接建表。

  - 总体规模：

    全部表的数据量规模总和可控，可预估，使用方式没有大的变化。

- Local Join：

  需要与已在默认Table Group的表，进行高效的Local Join。

- 规划二：新建Table Group

  如果默认Table Group不能满足您的需求，那么您可能需要多个Table Group。通常在如下条件下，在您的实例中可能需要多个Table Group：

  - 数据量

    已有Table Group的Shard数不适合当前表的预估数据量，例如，数据量小的表一般不适合放在大Table Group中，会造成小文件太多，IO开销变大，数据量大的表一般也不适合放在小Table Group中，造成查询并发度下降。这是最常见的需要划分多个Table Group的原因。

  - 负载分离

    已有的Table Group容纳的表数量很多，且大多数表需要同时写入，导致实例的负载很高，而即将创建的新表又需要较高的查询和写入吞吐，这时多个Table Group可以实现写入和查询一定程度上的独立，不受其他表写入查询影响（更准确的负载分离请参考计算资源隔离章节）。或者经过问题排查，确定是已有Table Group无法满足写入和查询需求时，也需要多个Table Group。

  - 表的相关性

    业务上有一系列具有独特写入或者查询模式的表，且这一系列表之间具有（或未来具有）Local Join的需求（Local Join需要左右表同在一个Table Group才能实现，并且Join Key是各自的分布列），同时这些表和其他Table Group的表具有很浅的联系或根本没有联系。这种情况下可以创建多个独立的Table Group。也就是说，如果您有一组表之间相关性很强的表，而这组表与其他表相关性很低、联合查询的概率很低，可以考虑创建多个Table Group。

  - 实例资源扩缩容

    如果实例进行过扩、缩容超过5倍以上，原来定的Shard数很可能不再满足需求，可以考虑更换默认Table Group。Shard数应大于计算节点数，小于实例总Core数的60%（即计算总Core数）。

- 规划三：多个Table Group放置

  如果需要规划多个Table Group，那么最好是能够在压测和生产之前，提前规划好多个Table Group的作用与意义，以及每个表所属的Table Group。规划时可以考虑如下因素：

  - 数据量

    首先应该考虑的是数据量，也就是大表放更多的Shard，中小表放更少的Shard。

  - 写入性能需求

    Shard数和数据写入性能呈一定的正相关性，单个Shard的写入能力是有上限的。Shard越多，写入的并发越多，写入的吞吐越高。因此，如果表有较高RPS的写入需求，那么可能需要增大shard数。Hologres单Shard将单core打满时，写入RPS为3000-5000条/秒（1KB/条），可以据此估算所需Shard数。考虑到每个shard还需要进行查询等读操作，一般不能使写入打满CPU，因此可以说，使用1/3core的shard，写入RPS为1000条/秒（每条1KB）。因此，举例而言，如果您要求写入RPS 6W/s，每条1KB，则应该选择的Shard数约在60Shard之上，视情况增减。

  - Table Group负载

    在建立一个新的Table Group时，需要考虑当前Table Group需要承载的表数量。例如，如果未来放在此Table Group的表很多，并且多数表都需要经常访问，那么Shard数很小将会存在并发不够的风险。

常见问题

- 我有512Core规格实例，主要针对一张实时事件表进行OLAP分析，表规模约200～400亿，该怎么设计Table Group和Shard Count？

计算负载比较单一，可以使用一个Table Group。512的默认Shard数为160，如果事件表列比较多，例如达到数百列，那么为了加强OLAP分析的并发度，可以适当增大Shard数。例如更改DB的默认Table Group的Shard数为200，或200以上，以放置事件表。

- 我有256Core规格实例，有很多张列存表，主要进行毫秒级快速OLAP分析，每张表数据量千万条级别，主要场景为group by多个字段，以及按条件查明细，该怎么设计Table Group和Shard Count？

  计算负载比较单一，可以一个Table Group解决。256Core规格实例，默认Table Group的Shard数为120，而对于千万级别的表，我们的建议是10-20shard，尤其对于group by等聚合操作，shard越多会有更多的shuffle开销，无法应对毫秒级分析。因此，默认Table Group可能无法应对我们的需求，可以视具体情况，更改DB的默认Table Group的Shard Count为16～40之间，并进行压测，效果会更好。

- 通过哪些手段排查慢Query是否为Shard Count不合适的原因？

  Shard数不合适分为过多和过少两种情况。

  - Shard数过多，一般会表现为：start query cost高，可以通过explain analyze后的start query cost行看出来；或者表现为shuffle开销大，这点可以通过explain analyze后，查看Redistribution Motion的Max_GetNext_Time的大小来判定。v0.10以上，可以通过慢查询日志查看历史Query的这些开销。
  - Shard数过少，一般会表现为：长时间计算时CPU也无法打满；或者scan数据的开销比较大（因为并发度不足），这点可以通过explain analyze后，查看Scan Node的Max_GetNext_Time大小来判定；或者数据写入的性能不足，上文提到Hologres单shard打满单core时，写入RPS为3000-5000，可据此估算Shard数是否过少。

- 我是点查的服务场景，我的QPS还不够高，是不是Shard不够的原因？

  首先要判断是不是别的原因引起的，例如并非点查而是分析查询、未走索引、未做Shard裁剪以及CPU已经打满等原因。当排查完后，不属于上述情况，且单SQL性能达到极致后，如果QPS仍不满足要求，则考虑增大shard数，以增大点查的后端并发。

- 如何排查Shard倾斜？

  Hologres提供了内部字段，hg_shard_id，即数据所在的Shard编号。可以通过SQL查看Shard倾斜情况。

  ```
  SELECT hg_shard_id, COUNT(1) FROM tbl1
  GROUP BY hg_shard_id ORDER BY COUNT(1) DESC;
  ```

  如果有Shard上的数据量显著高于其他Shard，则存在数据倾斜，可能需要调整分布列。

6.8. 系统参数

6.8.1. 时区

由于世界各国家与地区所处经度不同，地方时也不同，因此会划分为不同的时区。本文将会为你介绍在Hologres中时区的相关信息。

时区介绍

为了克服时间上的混乱，正式的时区划分包括二十四个时区（东、西各十二个时区），其中英国（格林尼治天文台旧址）为中时区（零时区）、东1~12区（正数，`+`），西1~12区（负数，`-`）。每个时区横跨经度十五度，时间正好是一小时。在国际上有两种时区的表达，分别为GMT和UTC。

- GMT 0:00

  是指格林尼治标准时间，这是以英国格林尼治天文台观测结果得出的时间，这是英国格林尼治当地时间，这个地方的当地时间过去被当成世界标准时间。

- UTC+00:00

  是指协调世界时。因为地球自转越来越慢，每年都会比前一年多出零点几秒，每隔几年协调世界时组织都会给世界时加一秒，让基于原子钟的世界时和基于天文学（人类感知）的格林尼治标准时间相差不至于太大。并将得到的时间称为UTC，这是现在使用的世界标准时间。

> **说明**
> - 协调世界时不与任何地区位置相关，也不代表此刻某地的时间，所以在说明某地时间时要加上时区也就是说GMT并不等于UTC，而是等于 `UTC+0` ，即 `GMT = UTC+0` ，只是格林尼治刚好在零时区上。
> - `+08` 是指时区为东八区，比零时区快八个小时。

在Hologres中所有日期和时间都用全球统一时间UTC格式存储，所有Region默认是 `UTC+08` （东八区时间），比世界协调时间（UTC）和格林尼治时间（GMT）快八小时的时区。同时Hologres提供两种存储时间戳的数据类型：不带时区的TIMESTAMP和带时区的TIMESTAMPTZ。

> **说明** 带有时区的数据建议都使用TIMESTAMPTZ类型存储。

名称	说明	精度	数据显示示例
TIMESTAMP	格式为 **日期+时间** ，不带时区信息。 存储的数据与写入的数据一样，当修改了客户端的时区时，里面存储的数据的值不会改变，即客户端展示的是写入的原始数据，没有时区偏移。	微秒	2022-01-01 01:01:01.123456
TIMESTAMP WITH TIME ZONE（TIMESTAMPTZ）	格式为 **日期+时间** ，带时区信息。 Hologres使用UTC时区的值来存储TIMESTAMPTZ 数据。在向TIMESTAMPTZ字段插入值的时候，Hologres会自动将客户端时区的值转换成UTC时区，在展示查询结果的时候，根据客户端TimeZone配置参数声明的时区转换为客户端时间。	毫秒	2022-02-01 10:33:20.125+08

查看默认的时区

通过以下命令语句查看当前客户端的时区。

> **说明**
> - 在Holoweb中执行的结果为实例的默认时区（PRC），对应 `UTC+08` （东八区）。
> - 如果使用其他开发工具，查询结果不是默认时区，则说明更改过客户端时区配置。

```
show timezone;
```

修改客户端时区

- 系统表

  在Postgres中通过系统表pg_timezone_names存储不同地区的默认时区信息，可以通过如下命令查询系统表查看每个地区对应的时区。

  ```
  select * from pg_timezone_names;
  ```

  系统表参数说明如下。

参数	类型	描述
name	text	时区名称。
abbrev	text	时区缩写。
utc_offset	interval	UTC偏移时区，正数（+）表示格林威治以东，负数（-）表示格林威治以西。
is_dst	boolean	如果当前正在观察夏令时，则为真（t）,否则为否（f）。

  通过如上的系统表，在Hologres中允许使用以下几种形式指定时区。

  - 完整的时区名称（name）：例如America/New_York。
  - UTC时区的偏移（utc_offset）：例如06:00:00，其中正数（+）表示格林威治以东，负数（-）表示格林威治以西。
  - 时区偏移的简写：例如+8，其中正数（+）表示格林威治以东，负数（-）表示格林威治以西。

  > **说明** 除以上几种形式外，不支持用 `UTC+8` 或者 `GMT+8` 这种格式指定时区。

- 修改客户端时区

  Hologres的时区默认按照UTC存储，所有Region默认为 `UTC+08` 。可以通过修改客户端时区参数达到客户端显示不同时区的目的。通过以下命令进行修改客户端显示时区。

  > **说明**
  > - 修改时区是修改客户端展示时区，并不是修改底层存储的真实时区信息。
  > - HoloWeb的时区显示默认都是 `UTC+08` ，不能修改。

- Session级别

  通过set命令可以在Session级别设置GUC参数。Session级别的参数只在当前Session生效，当连接断开之后将会失效，建议加在SQL前一起执行，使用方法如下。

```
--修改为加拿大/东部时区
set timezone ='Canada/Eastern';
--修改为东五区
set timezone ='05:00:00';
--修改为东六区
set timezone = '+06';
--修改为西三区
set timezone = '-03';
```

- 数据库级别

  可以通过 `alter database <db_name> set <value>;` 命令来DB级别设置GUC参数。

  > **说明** 此命令执行完成后：
  > - 在整个数据库级别生效，设置完成后当前连接需要重新断开连接才能生效。
  > - 新建数据库不会生效，需要重新手动设置。

  使用方法如下。

```
--DB级别修改时区为'UTC'零时区
alter database <db_name> set timezone = '0';
--DB级别修改时区为'UTC'西五时区
alter database <db_name> set timezone = '-05';
```

不同数据源与Hologres的时间类型映射

不同数据源与Hologres有关时间的数据类型映射如下表，建议按照推荐映射关系进行映射，否则会出现数据不一致/时区偏差的情况。

数据源	数据源数据类型	源数据示例	Hologres数据类型	映射Hologres数据示例	说明
	DATETIME	2001-07-14 02:14:19	TIMESTAMP	2001-07-14 02:14:19	MySQL的DATETIME存储的是没有时区的时间（可以理解为字符串的时间格式），取值范围为 `'1000-01-01 00:00:00' to '9999-12-31 23:59:59'` 。 数据存储时，会按照原始的时间格式存储，不进行任何转换，因此在Hologres中映射为TIMESTAMP类型。

数据源	数据源数据类型	源数据示例	Hologres数据类型	映射Hologres数据示例	说明
MySQL	TIMESTAMP	2019-02-23 05:21:16	TIMESTAMPTZ	2019-02-23 05:21:16+08	MySQL的TIMESTAMP默认是UTC时间，带时区，取值范围为 `'1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC` 。 数据存储前会根据数据库软件设置的时区（默认随系统），将写入的时间数据转化为UTC时间后，再进行存储。因此在Hologres映射为TIMESTAMPTZ类型。
MaxCompute	DATETIME	2021-11-29 00:01:00	TIMESTAMPTZ	2021-11-29 00:01:00.000+08	MaxCompute的DATETIME默认是UTC时间，取值范围为 `'0000-1-1' to '9999-12-31'` ，精确到毫秒。因此在Hologres中映射为TIMESTAMPTZ类型。

数据源	数据源数据类型	源数据示例	Hologres数据类型	映射Hologres数据示例	说明
	TIMESTAMP	2021-01-11 00:00:00.123456789	TIMESTAMPTZ	2021-01-11 00:00:00.123+08	MaxCompute的TIMESTAMP默认是UTC时间，取值范围为 '0000-01-01 00:00:00.000000000' to '9999-12-31 23:59:59.999999999' ，精确到纳秒。因此在Hologres中映射为TIMESTAMPTZ类型。 **说明** 说明：Hologres底层会将纳秒转换为毫秒，无需关心精度问题。
Flink	TIMESTAMP	2007-04-30 13:10:02.047	TIMESTAMPTZ	2007-04-30 13:10:02.047+08	Flink的TIMESTAMP类型默认是UTC时间，精确到纳秒，因此在Hologres中映射为TIMESTAMPTZ类型。
DataHub	TIMESTAMP	2020-03-05 10:10:00.123456+08	TIMESTAMPTZ	2020-03-05 10:10:00.123+08	DataHub的TIMESTAMP精确到微秒，使用UTC时间，Hologres Connecotr在写入过程中会自动将数据时区变为零时区，因此在Hologres中映射为TIMESTAMPTZ类型。

常见问题

- DataHub实时同步数据至Hologres，Hologres的字段类型为TIMESTAMP，DataHub字段类型为TIMESTAMP，写入后Hologres中时间比实际少八个小时。
  - 可能原因：Hologres Connecotr在写入过程中，会自动将数据时区变为零时区。如果Hologres字段设置为TIMESTAMP，不会带时区，写入后还是零时区，因此会少八个小时。
  - 解决方法：重新建表，将Hologres中的字段改成TIMESTAMPTZ。

- DataHub实时同步数据至Hologres，Hologres的字段类型为TIMESTAMPTZ，Tableau展示的时候与Hologres的时间差八个小时。
  - 可能原因：Tableau前端时区展示问题。
  - 解决方法：HoloWeb的时区默认为UTC+08（东八区），可以通过HoloWeb查看是否符合真实数据，然后在Tableau建立连接的时候，可以在**Initial SQL**（初始 SQL）里修改时区，命令示例如下。

    ```
    --修改为按照东八区显示
    set timezone to 'Asia/Shanghai';
    ```

- MySQL数据同步到Hologres后时间显示为什么会出现 `+08` ？
  - 可能原因： `+08` 代表的是当前客户端展示的是东八区时区，并不是数据是东八区。
  - 解决方法：通过修改客户端时区为指定的时区。
- JDBC里面如何设置时区？

  JDBC的时区显示默认同JVM时区，如果需要修改JDBC的时区显示，需要连接JDBC后，执行以下SQL进行修改。

  ```
  -- 修改为东七区
  set timezone = '+07';
  ```

6.8.2. GUC参数

为了更好的支持Hologres用户丰富的使用场景，Hologres提供一些GUC参数。本文将介绍Hologres中GUC参数的含义以及如何使用。

GUC参数一览表

GUC名称	适用场景	说明	使用示例
hg_enable_start_auto_analyze_worker	开启Auto Analyze，以及Auto Analyze相关配置，详情请参见ANALYZE和AUTO ANALYZE。	HologresV1.1及以上版本默认开启，值为 `on` 。	set hg_enable_start_auto_analyze_worker = on;
hg_auto_check_table_changes_interval		默认值为 `10min` 。	set hg_auto_check_table_changes_interval = '10min';
hg_auto_check_foreign_table_changes_interval		默认值为 `4h` 。	set hg_auto_check_foreign_table_changes_interval = '4h';
hg_auto_analyze_max_sample_row_count		默认值为 `16777216` 。	set hg_auto_analyze_max_sample_row_count = 16777216;
hg_fixed_api_modify_max_delay_interval		默认值为 `3d` 。	set hg_fixed_api_modify_max_delay_interval = '3day';

GUC名称	适用场景	说明	使用示例
hg_foreign_table_max_partition_limit	查MaxCompute外部表分区限制。	默认值为 `512` ，支持范围为 `0-1024` 。	set hg_foreign_table_max_partition_limit = 128;
hg_experimental_query_batch_size	MaxCompute性能调优参数，详情请参见优化MaxCompute外部表的查询性能。	默认值为 `8192` 。	set hg_experimental_query_batch_size = 4096;
hg_foreign_table_split_size		默认值为 `64` ，不建议设置过大。	set hg_foreign_table_split_size = 128;
hg_foreign_table_executor_max_dop		默认值调整为与实例Core数相同，最大为 `128` 。	set hg_foreign_table_executor_max_dop = 32;
hg_foreign_table_executor_dml_max_dop		默认值为 `32` 。	set hg_foreign_table_executor_dml_max_dop = 16;
hg_enable_access_odps_orc_via_holo		HologresV1.1及以上版本默认开启，值为 `on` 。	set hg_enable_access_odps_orc_via_holo = on;
hg_experimental_enable_result_cache	查询结果缓存。	默认值为 `on` ，不建议关闭。	set hg_experimental_enable_result_cache = on;
optimizer_join_order	内部性能调优参数，详情请参见优化内部表的性能。	默认值为 `exhaustive` ，后面可以接Query命令。	set optimizer_join_order = query;
optimizer_enable_motion_broadcast		默认值为 `on` ，按需关闭。	set optimizer_enable_motion_broadcast = off;
optimizer_enable_motion_redistribute		默认值为 `on` ，按需关闭。	set optimizer_enable_motion_redistribute =off;
optimizer_force_multistage_agg		默认值为 `off` ，按需开启。	set optimizer_force_multistage_agg = on;
hg_anon_enable	数据脱敏函数，详情请参见数据脱敏（Beta）。	默认值为 `off` ，建议数据库级别按需开启。	alter database <db_name> set hg_anon_enable = on;

GUC名称	适用场景	说明	使用示例
hg_experimental_encryption_options	数据加密规格设置，详情请参见数据存储加密。	默认值为 `off` ，建议数据库级别按需开启。	alter databse <db_name> set hg_experimental_encryption_options='AES256,623c26ee-xxxx-xxxx-xxxx-91d323cc4855,AliyunHologresEncryptionDefaultRole,187xxxxxxxxxxxx';
statement_timeout	活跃query超时时间，详情请参见Query管理。	默认值为 `8h` ，建议根据业务情况，session级别设置不同粒度的超时时间。	set statement_timeout = 5000 ;
idle_in_transaction_session_timeout	空闲事务的超时时间，详情请参见Query管理。	默认值为 `10min` ，建议数据库级别设置，否则当事务泄漏时容易造成死锁。	alter database db_name set idle_in_transaction_session_timeout=300000;
idle_session_timeout	自动释放空闲连接超时时间，详情请参见连接数管理。	默认值为 `0` ，即不会自动释放。建设设置，否则连接数太多导致超过实例默认上限，从而无法连接。	alter database <db_name> SET idle_session_timeout = 600000;
hg_experimental_functions_use_pg_implementation	时间范围扩展。 `to_char` 、 `to_date` ，和 `to_timestamp` 函数在处理时间类型时默认范围为 `1925-2282` ，通过设置GUC参数支持0000-9999年的时间，详情请参见类型转换函数。	HologresV1.1.31版本开始支持，设置后支持时间范围为 `0000-9999` 。	set hg_experimental_functions_use_pg_implementation = 'to_char';
hg_experimental_approx_count_distinct_precision	调整APPROX_COUNT_DISTINCT误差率，详情请参见APPROX_COUNT_DISTINCT。	默认值为 `17` ，取值范围为 `12-20` 。	set hg_experimental_approx_count_distinct_precision = 20;
timezone	时区设置。	默认值为 `PRC` （东八区）。	set timezone='GMT-8:00';
hg_experimental_enable_create_table_like_properties	复制表时同时复制表属性（主键、索引等），详情请参见CREATE TABLE LIKE。	默认值为 `off` 。	set hg_experimental_enable_create_table_like_properties=true;

GUC名称	适用场景	说明	使用示例
hg_experimental_affect_row_multiple_times_keep_first	使用 `insert on conflict` ，源数据重复时数据保留策略，详情请参见INSERT ON CONFLICT(UPSERT)。	默认值为 `off` 。	set hg_experimental_affect_row_multiple_times_keep_first = on;
hg_experimental_affect_row_multiple_times_keep_last			set hg_experimental_affect_row_multiple_times_keep_last = on;
hg_experimental_enable_read_replica	单实例多副本高可用以及相关配置，详情请参见单实例Shard多副本高吞吐（Beta）。	默认值为 `on` 。	set hg_experimental_enable_read_replica = on;

查看当前GUC参数的状态或默认值

通过 `show` 命令语句可以查看某个GUC参数的状态或者默认值，使用示例如下。

- 查看是否开启Auto Analyze。

```
show hg_enable_start_auto_analyze_worker;
```

- 查看读取MaxCompute分区限制大小。

```
show hg_foreign_table_max_partition_limit;
```

设置GUC参数

GUC在使用时，可以设置为session级别或者数据库级别生效。

> **说明** 具体是session级别还是数据库级别，需要根据业务场景以及参数的详情合理评估，不建议所有的参数都设置为数据库级别。

- session级别

  通过 `set` 命令可以在session级别设置GUC参数。session级别的参数只在当前session生效，当连接断开之后，将会失效，建议加在SQL前一起执行。

  - 语法示例如下。

    ```
    set <GUC_name> = <values>;
    ```

    GUC_name为GUC参数的名称，values为GUC参数的值。

  - 使用示例如下。

    ```
    -- 开启Auto Analyze
    set hg_enable_start_auto_analyze_worker = on;
    -- 读取MaxCompute的分区限制变为1024
    set hg_foreign_table_max_partition_limit =1024;
    ```

- 数据库级别

可以通过 `alter database xx set xxx` 命令来设置DB级别的GUC参数执，执行完成后在整个DB级别生效，设置完成后当前连接需要重新断开连接才能生效。新建DB不会生效，需要重新手动设置。

- 语法示例如下。

```
alter database <db_name> set <GUC_name> = <values>;
```

db_name为数据库名称，GUC_name为GUC参数的名称，values为GUC参数的值。

- 使用示例如下。

```
-- DB级别开启Auto Analyze
alter database testdb set hg_enable_start_auto_analyze_worker = on;
-- DB级别读取MaxCompute的分区限制变为1024
alter database testdb set hg_foreign_table_max_partition_limit =1024;
```

6.8.3. 系统表

本文将会为你介绍Hologres中的系统表以及每个表如何使用。

概述

Hologres系统表的组成如下表所示。

表名	使用场景
hologres.hg_table_properties	查看当前数据库下Hologres所有表以及表属性。
pg_catalog.pg_tables	查看表、视图等关系的信息。
pg_catalog.pg_locks	查看表的锁信息。
pg_catalog.pg_class	PostgreSQL原生元数据表，一般是结合其他PostgreSQL系统表一起使用，查看表关系等信息。
hologres_statistic.hg_table_statistic	Hologres的统计信息表，用于多节点共享统计信息存储。
pg_catalog.pg_stats	PostgreSQL原生统计信息表，在单节点本地直接供planner使用。
pg_catalog.pg_roles	查看实例内角色及其权限信息。
information_schema.role_table_grants	查看用户角色被授予对象（表、视图等）的权限信息。

hologres.hg_table_properties

hologres.hg_table_properties用于存放当前数据库下的所有表和表属性，包含如下字段。

字段	说明

字段	说明
table_namespace	Schema名称，Hologres会包含3个系统Schema： • hologres：用于Hologres的系统表。 • hologres_statistic：用于存放统计信息表。 • pg_catalog：用于PostgreSQL原生的元数据表。
table_name	表名称，Hologres包含多个系统表，如下。 • hologres.hg_insert_progress_stats：`insert`进度信息。 • hologres.hg_table_properties：表的索引、属性信息。 • hologres.hg_table_group_properties：Table Group元信息。 • hologres_statistic.hg_table_statistic：存放统计信息。 • pg_catalog.pg_stat_activity：Query运行数据。

property_key	表属性，包含如下属性。
	• table_id：表的ID信息，后端会给每个表分配一个ID，方便身份识别。
	• clustering_index_id：Clustering索引的ID信息。
	• clustering_index_name：Clustering索引的名称。
	• lifecycle_in_days：表的生存时间TTL，值为-1表示永久生效，目前Hologres不支持修改。
	• storage_format：表数据的存储格式，行存表为sst，列存表从HologresV0.10版本开始默认为orc。
	• table_group：表所在的Table Group名称。
	• schema_version：表的版本信息。
	• primary_key：表的主键信息。
	• orientation：表的存储模式，有以下三种模式。
	  ◦ row：行存。
	  ◦ column：列存。
	  ◦ row,column：行列混存（HologresV1.1版本开始支持）。
	• distribution_key：设置的分布列信息。
	• dictionary_encoding_columns：设置的字典编码列信息。
	• bitmap_columns：设置的Bitmap编码列。
	• clustering_key：设置的Clustering key（聚簇索引）信息。
	• create_time：表的创建时间。
	• last_ddl_time：最后执行DDL的时间。

property_value	表属性的值。

查看表对应的Hologres索引和属性信息的命令示例如下。

```
SELECT * FROM hologres.hg_table_properties where table_name = '<tablename>';
```

tablename为表名称。

pg_catalog.pg_tables

pg_tables保存表的元信息，包含如下字段。

字段	描述
schemaname	表所在的Schema名称，除了业务创建的Schema外，还包含系统Shcema名称如下。 • hologres：保存Hologres的系统表。 • pg_catalog：保存Potsgres的元数据信息。 • information_schema：保存当前数据库对象信息的视图。
tablename	表名称。
tableowner	表的Owner，如下两种类型。 • holo_admin：系统账号，是系统表的Owner，无法更改。 • developer用户组：开启了简单权限模型（SPM）或者基于Schema级别的简单权限模型（简称SLPM）。
tablespace	Hologres无此概念，无需关注
hasindexes	如果表有（或最近有过）任何一个索引，此列为true。
hasrules	如果表有（或曾经有过）规则，此列为true。
hastriggers	如果表有（或曾经有过）触发器，此列为true。
rowsecurity	如果表上启用了安全性规则，此列为true。在Hologres中无需关注。

查看当前实例所有表的命令示例如下。

```
--查看当前数据库下所有表（包含系统表）
SELECT * FROM pg_tables;
--查看当前schema下所有表以及表owner（不包含系统表）
SELECT  n.nspname as "Schema"
        ,c.relname as "Name"
        ,CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materiali
zed view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' T
HEN 'foreign table' WHEN 'p' THEN 'partitioned table' WHEN 'I' THEN 'partitioned index' END
as "Type"
        ,pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM    pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n
ON      n.oid = c.relnamespace
WHERE   c.relkind IN ('r','p','v','m','S','f','')
and     n.nspname <> 'pg_catalog'
and     n.nspname <> 'information_schema'
and     n.nspname !~ '^pg_toast'
and     pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1, 2;
```

pg_catalog.pg_locks

pg_locks记录运行时的锁信息，常用于当发现DDL卡住或者Query卡住时，定位是否有锁，其字段如下。

字段	说明
locktype	锁的类型，包含如下类型。 • relation：表锁。 • extend、page, tuple、transactionid、virtualxid、object、userlock：Postgres原生锁，在Hologres内无需关注。 • advisory：DDL锁。
database	目标所在的数据库的对象标识符（OID）。
relation	表的对象标识符（OID），如果目标不是表，也不是表的一部分，则为null。
page	在Hologres内无需关注。
tuple	在Hologres内无需关注。
virtualxid	事务的虚拟ID，如果目标不是虚拟事务ID，就为null。
transactionid	事务的ID，如果目标不是事务ID，就为null。
classid	包含该目标的系统表的对象标识符（OID），如果目标不是普通数据库对象，则为null。在Hologres内无需关注。
objid	在Hologres内无需关注。
objsubid	在Hologres内无需关注。
virtualtransaction	持有或者等待此锁的事务ID。在Hologres内无需关注。
pid	持有或者等待这个锁的服务器进程的进程 ID，可以通过pg_stat_activity这个表查看进程信息。
mode	这个进程的锁模式，分为共享锁和排他锁等模式。
granted	• 如果持有锁，为true。 • 如果等待锁，为false。
fastpath	• 如果锁通过快速路径获得为true。 • 如果通过主锁表获得为false。 在Hologres内无需关注。

查看表锁信息以及排查锁的示例如下。

```
-- 查看所有在等的锁，持有的锁，及持有/等待的进程等信息
select * from pg_locks where granted = 'f';
-- 查看拿到了表锁的进程信息
select pid from pg_locks where relation = <OID> and granted = 't';
-- 可根据上述结果，查找pid具体的行为信息
select * from pg_stat_activity where pid = <PID>;
```

pg_catalog.pg_class

pg_class用于保存原生Postgres的所有系统信息，包含如下字段。

名字	说明
oid	表、索引、视图等关系的唯一标志符。
relname	表、索引、视图等关系的名称。
relnamespace	包含这个关系的Schema的OID。
reltype	在Hologres内无需关注。
reloftype	在Hologres内无需关注。
relowner	关系的Owner。
relam	在Hologres内无需关注。
relfilenode	在Hologres内无需关注。
reltablespace	在Hologres内无需关注。
relpages	在Hologres内无需关注。
reltuples	表中行的数目。只是查询规划器使用的一个估计值，由VACUUM、ANALYZE和几个DDL命令更新。在Hologres中用作统计信息的行数。
relallvisible	在表的可见映射中标记所有可见的页的数目。只是查询规划器使用的一个估计值， 由VACUUM、ANALYZE 和几个 DDL 命令更新。在Hologres中用作统计信息的版本
reltoastrelid	在Hologres内无需关注。
relhasindex	如果它是一个表而且至少有（或者最近有过）一个索引，则为true。
relisshared	如果该表在整个集群中由所有数据库共享则为true。只有某些系统表（比如pg_database）是共享的，在Hologres内无需关注。
relpersistence	有如下值。 • p：permanent table（永久表）。 • u：unlogged table（未加载的表）。 • t： temporary table （临时表）。

名字	说明
relkind	有如下值。 • r：ordinary table（普通表）。 • i： index（索引）。 • S：sequence（序列）。 • v：view（视图）。 • m： materialized view（物化视图）。 • c： composite type（复合类型）。 • t： TOAST table（TOAST 表）。 • f：foreign table（外部表）。
relnatts	表中列的数目（不包含系统字段）。
relchecks	表里的CHECK约束的数目，详情请参见pg_constraint，在Hologres内无需关注。
relhasoids	如果为关系中每行都生成一个OID则为true。在Hologres内无需关注。
relhaspkey	如果这个表有一个（或者曾经有一个）主键，则为true。
relhasrules	如果这个表有（或曾经有）规则就为true，详情请参见pg_rewrite。在Hologres内无需关注。
relhastriggers	如果表有（或者曾经有）触发器，则为true，详情见pg_trigger。在Hologres内无需关注。
relhassubclass	如果表有（或者曾经有）任何继承的子表，则为true。
relispopulated	在Hologres内无需关注。
relreplident	在Hologres内无需关注。
relfrozenxid	在Hologres内无需关注。
relminmxid	在Hologres内无需关注。
relacl	访问权限。详情请参见GRANT REVOKE。
reloptions	表的属性，例如autovacuum_enabled=false代表关闭此表的autovacuum/autoanalyze功能。

命令语句示例。

- 查看父表对应的所有子表示例。

```
-- 含分区键值
SELECT  c.oid::pg_catalog.regclass
        ,c.relkind
        ,pg_catalog.pg_get_expr(c.relpartbound, c.oid)
FROM    pg_catalog.pg_class c
        ,pg_catalog.pg_inherits i
WHERE   c.oid = i.inhrelid
AND     i.inhparent::pg_catalog.regclass = 'parent_table_name'::pg_catalog.regclass
ORDER BY pg_catalog.pg_get_expr(c.relpartbound, c.oid) = 'DEFAULT'
;
-- 不含分区键值
SELECT
    nmsp_parent.nspname AS parent_schema,
    parent.relname      AS parent,
    nmsp_child.nspname  AS child_schema,
    child.relname       AS child
FROM pg_inherits
    JOIN pg_class parent            ON pg_inherits.inhparent = parent.oid
    JOIN pg_class child             ON pg_inherits.inhrelid   = child.oid
    JOIN pg_namespace nmsp_parent   ON nmsp_parent.oid  = parent.relnamespace
    JOIN pg_namespace nmsp_child    ON nmsp_child.oid   = child.relnamespace
WHERE parent.relname='parent_table_name';
```

- 查看所有外表以及外表对应的MaxCompute表。

```
SELECT  n.nspname
        ,c.relname
        ,s.srvname
        ,pg_catalog.array_to_string(
            ARRAY(
                SELECT pg_catalog.quote_ident(option_name) || ' ' || pg_catalog.quote_lit
eral(option_value) FROM pg_catalog.pg_options_to_table(ftoptions)
            )
            ,', '
        )
FROM    pg_catalog.pg_foreign_table f
        ,pg_catalog.pg_foreign_server s
        ,pg_catalog.pg_class c
        ,pg_catalog.pg_namespace n
WHERE   s.oid = f.ftserver
and     c.oid = f.ftrelid
and     c.relnamespace = n.oid
and     n.nspname not in ('hologres', 'hologres_statistic', 'pg_catalog', 'pg_toast')
;
```

- 查看当前Schema所有View。

```
SELECT n.nspname as "Schema",
  c.relname as "Name",
  CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized v
iew' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN
'foreign table' WHEN 'p' THEN 'table' WHEN 'I' THEN 'index' END as "Type",
  pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('v')
      AND n.nspname <> 'pg_catalog'
      AND n.nspname <> 'information_schema'
      AND n.nspname !~ '^pg_toast'
  AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
```

- 查看表的注释。

```
select  c.relname 表名
        ,cast(obj_description(relfilenode,'pg_class') as varchar) 名称
        ,a.attname 字段
        ,d.description 字段备注
        ,concat_ws(
            ''
            ,t.typname
            ,SUBSTRING(format_type(a.atttypid,a.atttypmod) from '\(.*\)')
        ) as 列类型
from    pg_class c
        ,pg_attribute a
        ,pg_type t
        ,pg_description d
where   a.attnum > 0
and     a.attrelid = c.oid
and     a.atttypid = t.oid
and     d.objoid = a.attrelid
and     d.objsubid = a.attnum
and     c.relname in (select tablename from pg_tables where schemaname = 'public' and pos
ition('_2' in tablename) = 0)
order by c.relname
         ,a.attnum;
```

hologres_statistic.hg_table_statistic

Hologres的统计信息表，其字段如下。

字段	说明
unique_name	表的唯一标志。
schema_version	表的版本号。
statistic_version	统计信息版本。
statistics	统计信息内容，Base64编码。

字段	说明
schema_name	表所在Schema名称。
table_name	表名称。
total_rows	总行数。
sample_rows	本统计信息的采样行数。
nattr	表的字段个数。
used_attrs	Analyze用到的字段。
histogram_attrs	具备直方图统计信息的字段。
ndv_attrs	具备distinct value统计信息的字段。
user_name	Analyze或者Auto Analyze的执行者。
analyze_timestamp	Analyze或者Auto Analyze的执行开始时间。
analyze_cost	Analyze或者Auto Analyze的耗时。
analyze_count	Analyze或者Auto Analyze的次数。

pg_catalog.pg_stats

pg_stats用于保存Postgres原生的统计信息，字段如下。

字段	说明
schemaname	Schema名称。
tablename	表名称。
attname	列名（字段名）。
inherited	如果为true，表示此行包括继承子列。
null_frac	记录中字段为空的百分比。
avg_width	列的平均字节宽度。
n_distinct	• 大于零，表示列中distinct值的估计个数。 • 小于零，是distinct值个数除以行数的负值（当ANALYZE认为distinct值的数量会随着表增长而增加时采用负值的形式，而如果认为列具有固定数量的distinct值时采用正值的形式）。 例如，-1表示一个唯一列，即其中distinct值的个数等于行数。
most_common_vals	列中Most Common Values的一个列表（如果没有任何一个值看起来比其他值更常用，此列为空）。

字段	说明
most_common_freqs	Most Common Values值的频率列表，即每一个常用值的出现次数除以总行数（如果most_common_vals为空，则此列为空）。
histogram_bounds	将列值划分成大小接近的组的值列表，即直方图列表。如果存在most_common_vals，其中的值会被直方图计算所忽略。
correlation	在Hologres内无需关注。
most_common_elems	在列值中，Most Common Values出现的非空元素列表。
most_common_elem_freqs	Most Common Values值的频度列表，即含有至少一个给定值实例的行的分数。（如果most_common_elems为空，则此列为空）。
elem_count_histogram	在Hologres中无需关注。

pg_catalog.pg_roles

pg_roles用于存放实例内角色及其权限信息，字段如下。

字段	说明
rolname	角色名称。
rolsuper	角色是否具有超级用户权限，取值如下。 • f：无超级用户权限。 • t：有超级用户权限。
rolinherit	如果此角色是另一个角色的成员，角色是否能自动继承，取值如下。 • f：不可以继承另一个角色的权限。 • t：可以继承另一个角色的权限。
rolcreaterole	能否创建更多角色，取值如下。 • f：不可以创建更多角色。 • t：可以创建更多角色。
rolcreatedb	能否创建数据库，取值如下。 • f：不可以创建数据库。 • t：可以创建数据库。
rolcanlogin	角色是否能登录实例，取值如下。 • f：不可以登录实例。 • t：可以登录实例。
rolreplication	在Hologres内无需关注。
rolconnlimit	用户的连接数限制，-1表示无限制。
rolpassword	在Hologres内无需关注。

字段	说明
rolvaliduntil	在Hologres内无需关注。
rolbypassrls	在Hologres内无需关注。
rolconfig	在Hologres内无需关注。
oid	角色的ID，唯一标志符。

查看某个账号的对应权限示例如下。

```
SELECT * FROM pg_roles where rolname='<uid>';
```

查看当前实例下所有的用户以及权限示例如下。

```
SELECT  r.rolname
        ,r.rolsuper
        ,r.rolinherit
        ,r.rolcreaterole
        ,r.rolcreatedb
        ,r.rolcanlogin
        ,r.rolconnlimit
        ,r.rolvaliduntil
        ,ARRAY(
            SELECT b.rolname FROM pg_catalog.pg_auth_members m JOIN pg_catalog.pg_roles b O
N (m.roleid = b.oid) WHERE m.member = r.oid
        ) as memberof
        ,r.rolreplication
        ,r.rolbypassrls
FROM    pg_catalog.pg_roles r
WHERE   r.rolname !~ '^pg_'
ORDER BY 1;
```

information_schema.role_table_grants

Hologres实例中用户角色被授予对象（表、视图等）的权限信息表，其字段如下。

字段	描述
grantor	授权方角色。
grantee	被授权方角色。
table_catalog	数据库名称。
table_schema	Schema名称。
table_name	表名称。

字段	描述
privilege_type	被授予权限的类型： • SELECT。 • INSERT。 • UPDATE。 • DELETE。 • TRUNCATE。 • REFERENCES。 • TRIGGER。
is_grantable	如果权限是可授予的，则此列值为 `YES` ，否则为 `NO` 。
with_hierarchy	权限类型为SELECT时，此列值为 `YES` ，其他权限类型时值为 `NO` 。

查看某个账号在整个数据库中权限关联关系示例如下。

```
SELECT * FROM information_schema.role_table_grants where grantor='<uid>' or grantee='<uid>'
;
```

其它信息

除上述外，还可以查表的DDL以及实例endpoint等信息，如下示例。

- 查看表/视图DDL

```
select hg_dump_script('tablename'); -- 表DDL
select hg_dump_script('viewname');  -- 视图DDL
```

> **说明** 如果执行失败，需要在DB中创建extension。执行 create extension hg_toolkit;

- 查看实例的endpoint

  除了在Hologres管理控制台查看实例的endpoint外，还可以通过以下命令查看实例的endpoint。

```
show hg_frontend_endpoints;
```

同时，一些常见命令可以通过psql简写来实现，请参见psql。

6.8.4. Extension扩展

Hologres中支持加载扩展（extension）以实现更丰富的功能，本文为您介绍Hologres中支持的extension扩展以及如何加载、查看、卸载extension。

使用限制

- 一个数据库只能在一个Schema下加载一次extension。例如在某数据库的默认Schema下加载了extension，则此数据库下其他的Schema不能再加载extension。
- 可以将extension加载在全局空间pg_catalog系统Schema下，默认该数据库的所有Schema都能访问该

extension的功能；如果不指定Schema，则会默认加载在publicSchema下。

- 账号具有Superuser权限才能加载、卸载extension。
- 目前仅支持加载系统内置extension，不支持加载自定义及外部extension。

Extension扩展

Extension名称	适用场景	相关文档
spm、slpm	开启权限模型调用函数的开关。	简单权限模型的使用、基于Schema级别的简单权限模型的使用
hive_compatible	使用`get_json_object()`函数。	GET_JSON_OBJECT
hologres_fdw	Hologres跨库查询。	跨库查询
oss_fdw	通过外部表访问OSS。	通过外部表直接访问OSS
dlf_fdw	通过DLF读取OSS数据。	通过DLF读取OSS数据
proxima	使用Proxima进行向量计算。	Proxima向量计算
flow_analysis	使用明细圈人、漏斗分析函数。	漏斗分析函数、明细圈人函数
aggregate_view	使用聚合视图。（Beta）	聚合视图（Beta）
roaringbitmap	使用Roaring Bitmap函数。	Roaring Bitmap函数
hg_binlog	消费Hologres Binlog。	通过JDBC消费Hologres Binlog（Beta）
foreign_table_exposer	针对特定BI工具优化访问外表。	Power BI
postgis	使用空间函数。	PostGIS (Beta)
clickhouse	兼容ClickHouse迁移函数。	迁移ClickHouse至Hologres

加载Extension

- 语法示例

  加载Extension的SQL命令如下。

```
-- 需Superuser权限执行
CREATE extension IF NOT EXISTS <extension_name> SCHEMA <schema_name>;
```

参数	说明
extension_name	需要加载的extension名称，Hologres支持的部分extension参见本文上述列表。

参数	说明
schema_name	加载extension所在的schema名称。如果不指定schema，则会默认在public下加载extension，建议加载在pg_catalog下，数据库级别可用。

- 使用示例

  在pg_catalog下加载postgis扩展示例如下。

```
CREATE extension if not exists postgis schema pg_catalog;
```

查看当前数据库已加载Extension

通过如下SQL命令可以查看当前数据库下已加载的extension，包括系统默认加载的extension。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
    ORDER BY        1;
```

卸载Extension

卸载extension的SQL命令如下。

```
-- 需Superuser权限执行
DROP extension <extension_name>;
```

参数	说明
extension_name	需要卸载的extension名称，Hologres支持的部分extension参见本文上述列表。

6.9. 其他语句

6.9.1. 其他SQL语句

交互式分析Hologres兼容PostgreSQL，除了支持创建、查询及更新数据库的表和Schema等对象的SQL语句外，同时还支持其他SQL语句。本文为您介绍Hologres支持的其他SQL语句。

Hologres已支持的其他PostgreSQL功能语句如下表所示，您可以参考PostgreSQL官网文档的用法示例进行使用。

命令	说明
ALTER TABLE	修改表，仅支持PostgreSQL的部分功能，详情请参见如下： • ALTER TABLE • ALTER PARTITION TABLE • ALTER FOREIGN TABLE
ALTER ROLE	修改角色。
ANALYZE	更新统计信息。
BEGIN	事务开始，Hologres的 `BEGIN` 仅对DDL语句生效。
COMMIT	事务确认，Hologres的 `COMMIT` 仅对DDL语句生效。
CREATE DATABASE	创建数据库。
CREATE EXTENSION	创建扩展。
CREATE FOREIGN DATA WRAPPER	创建外部访问接口。
CREATE FOREIGN TABLE	创建外部表，Hologres仅支持创建MaxCompute外部表。
CREATE GROUP	创建组。
CREATE SERVER	创建外部服务器。
CREATE TABLE	创建表，Hologres仅支持PostgreSQL中 `CREATE TABLE` 语句的部分功能。其中，不支持的功能如下： • UNLOGGED • TEMP • IF NOT EXISTS • LIKE • CHECK • DEFAULT • GENERATED • UNIQUE • EXCLUDE • FOREIGN KEY • DEFERRABLE • WITH OIDS • GLOBAL • LOCAL Hologres的PARTITION仅支持LIST类型，并且PARTITION LIST只能取类型为STRING的唯一值。
CREATE VIEW	创建视图。

命令	说明
CREATE USER	创建用户。
CREATE USER MAPPING	创建用户映射。
DROP DATABASE	删除数据库。
DROP FOREIGN DATA WRAPPER	删除外部访问接口。
DROP FOREIGN TABLE	删除外部表。
DROP GROUP	删除组。
DROP OWNED	删除所有权。
DROP ROLE	删除角色。
DROP SERVER	删除外部服务器。
DROP TABLE	删除表。
DROP USER	删除用户。
DROP USER MAPPING	删除用户映射。
END	事务结束，Hologres的 `END` 仅支持与DDL语句配合使用。
EXPLAIN	查看执行计划。
INSERT	导入。
ROLLBACK	事务回滚。
SELECT	Hologres仅支持PostgreSQL中 `SELECT` 语句的部分语句功能。其中，不支持的功能如下： • 递归查询 • TABLESAMPLE • LOCKING • ONLY
SET	执行GUC参数，详情请参见GUC参数。

命令	说明

CALL set_table_property('<tablename>', 'time_to_live_in_seconds', '<values>');	设置表数据生命周期，表数据的生存时间TTL的单位为秒，必须是非负数字类型，整数或浮点数均可。使用限制如下：
	• 若是没有显式指定TTL，则默认数据保留永久（100年）。
	• TTL过期时间以数据第一次写入的时间为准，而不是数据最后一次修改的时间；当到达TTL后，表数据会在某一段时间内被清除（没有固定时间段），只是数据被清除，表不会被删除。
	• 可以单独执行该语句，表示修改表数据生命周期。
	• TTL相比于 `DELETE FROM` 命令几乎不占资源，也不会锁表。但是对于过期的数据不能保证数据一致性。这意味着：
	  ○ 读取过期数据，可能能读到，可能读不到，也可能读到某个历史版本。
	  ○ 修改或删除过期数据，可能能正常工作，也可能出现PK重复之类的意外结果。
	  ○ 所以使用TTL功能，最好要保证不会读取或修改过期的数据，或者对过期数据的一致性不关心。

7.BI分析及可视化

7.1. 概述

本文为您介绍交互式分析Hologres支持连接的BI分析工具。

Hologres兼容PostgreSQL生态，提供JDBC/ODBC接口，支持对接第三方ETL和BI工具，包括Tableau及Quick BI等。您可以将Hologres查询的数据直接对接BI工具，进行多维分析和探索业务。

Hologres支持连接的BI分析工具如下表所示。

BI工具	是否支持分析外部表
Apache Superset	是
Apache Zeppelin	是
Dataiku	是（有限支持）
DataV	是
DataWorks数据服务	是
Davinci	是（有限支持）
FineBI	是
FineReport	是
Grafana	是
IBM Cognos Analytics	是
Metabase	是
Power BI	是（有限支持）
Qlik	是（有限支持）
Quick BI	是
Redash	是
SAP BusinessObjects	是
SQL Workbench/J	是
Tableau	是
Yonghong BI	是

7.2. Apache Superset

本文为您介绍Apache Superset如何连接Hologres并可视化分析数据。

前提条件

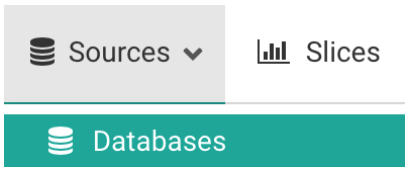
- 开通Hologres，详情请参见购买Hologres。
- 安装Apache Superset，详情请参见Apache Superset官网。

背景信息

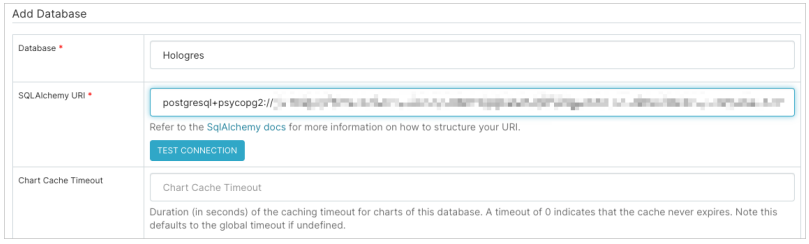
Apache Superset（incubating）是一款数据探索和可视化分析的开源BI工具。Apache Superset基于Python开发，使用了Flask、Pandas、SQLAlchemy等组件。Hologres兼容PostgreSQL，如下示例使用Apache Superset V0.38.0版本连接Apache Superset进行数据分析。

操作步骤

1. 登录Superset，在顶部菜单栏选择**Sources > Databases**，进入**Databases**页面。



2. 在页面右上角单击按钮，进入**Add Database**页面配置如下参数，添加数据源。



参数	说明
Database	您定义的数据库名称。
SQLAlchemy URI	连接Hologres目标数据库的连接字符串。SQLAlchemy URI具体格式如下： `postgresql+psycopg2://{AccessKey ID}:{AccessKey Secret}@{host}:{port}/{database}` 其中，各参数说明如下所示： ○ AccessKey ID和AccessKey Secret：当前阿里云账号的AccessKey ID和AccessKey Secret。获取方式请参见创建访问密钥。 ○ host：Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取公共网络地址。 ○ port：Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。 ○ database：Hologres创建的数据库名称。 更多有关连接信息的说明，请参见SQLAlchemy。

3. 单击**Test Connection**按钮以确认可以成功连接和验证。

4. 如果连接正常，请单击页面底部的Save按钮保存配置。

   保存完成后，您就可以连接Hologres以可视化的方式分析展示数据，具体操作请参见Apache Superset。

7.3. Apache Zeppelin

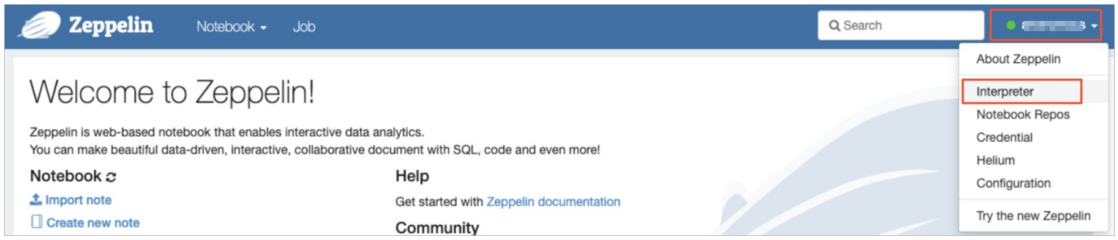
Apache Zeppelin是一款基于Web的Notebook产品，能够交互式数据分析。使用Zeppelin，您可以使用丰富的预构建语言后端（或解释器）制作交互式的协作文档，例如Scala、Python、SparkSQL、Hive等。Hologres兼容PostgreSQL，支持直接连接Apache Zeppelin进行数据分析。本文为您介绍Apache Zeppelin如何连接Hologres并可视化分析。

使用限制

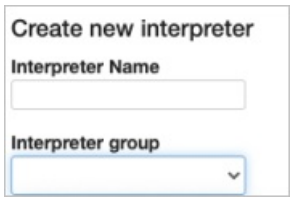
最新版的Apache Zeppelin中，PostgreSQL解释器已被弃用，并合并到JDBC解释器中。您可以使用具有相同功能的JDBC Interpreter对接Hologres。

Apache Zeppelin连接Hologres

1. 安装Apache Zeppelin
   安装Apache Zeppelin，详情请参见Apache Zeppelin官方文档。
2. 配置解释器
   i. 登录Apache Zeppelin，单击右上方的用户名，从下拉菜单中选择Interpreter。

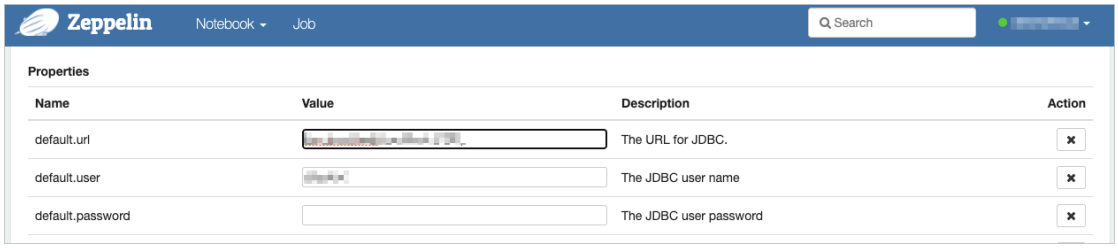


   ii. 在页面右上角单击+Create，配置如下参数信息，创建一个新的Interpreter。



   - Interpreter Name填写您自定义的名称。
   - 在创建时，您需要将Interpreter Group设置为**JDBC**。

iii. 在将Interpreter Group设置为**JDBC**后，您需要配置如下参数信息。



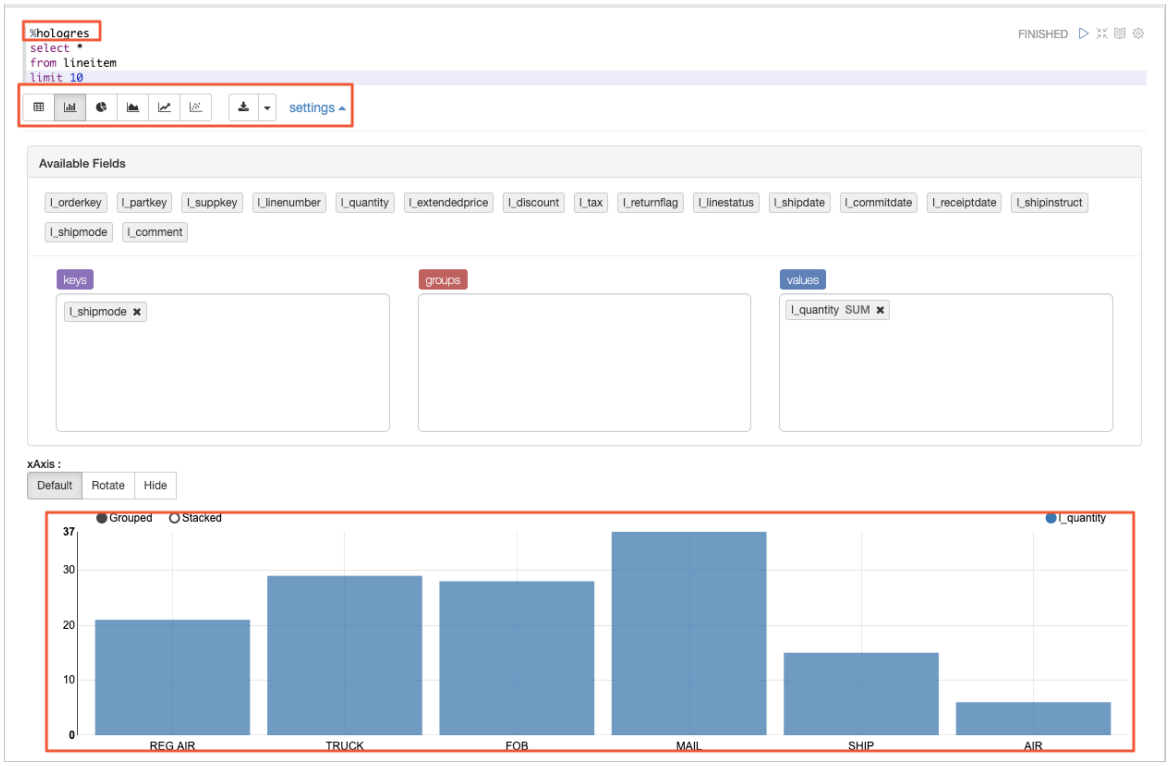
参数	说明
default.url	JDBC的URL，填写格式为 `jdbc:postgresql://{host}:{port}/{database}` ，其中各参数解释如下所示： ■ host：Hologres实例的网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。 ■ port：Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。 ■ database：Hologres创建的数据库名称。
default.user	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
default.password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

您也可以根据业务需求设置页面中的其它参数，更多关于参数的解释说明，请参见Apache Zeppelin官方文档。

iv. 单击页面下方的**Save**保存配置。

3. 新建Notebook进行操作分析。

在Notebook编写SQL时，必须要在SQL前加入解释器的名称，例如 `%hologres` ，以便Zepplin使用指定的解释器生成并发送查询。您可以在页面指定数据展示类型，查询之后的结果将展示在页面下方。



7.4. Dataiku

本文为您介绍Dataiku如何连接Hologres并可视化分析数据。

使用限制

由于Dataiku的限制，支持直接使用SQL Notebook查询外部表，但不支持使用Import tables to datasets查看外部表。

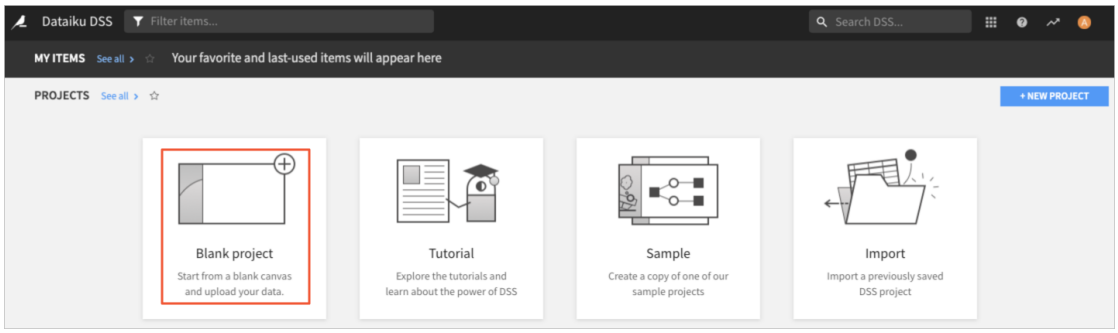
操作步骤

1. 安装Dataiku

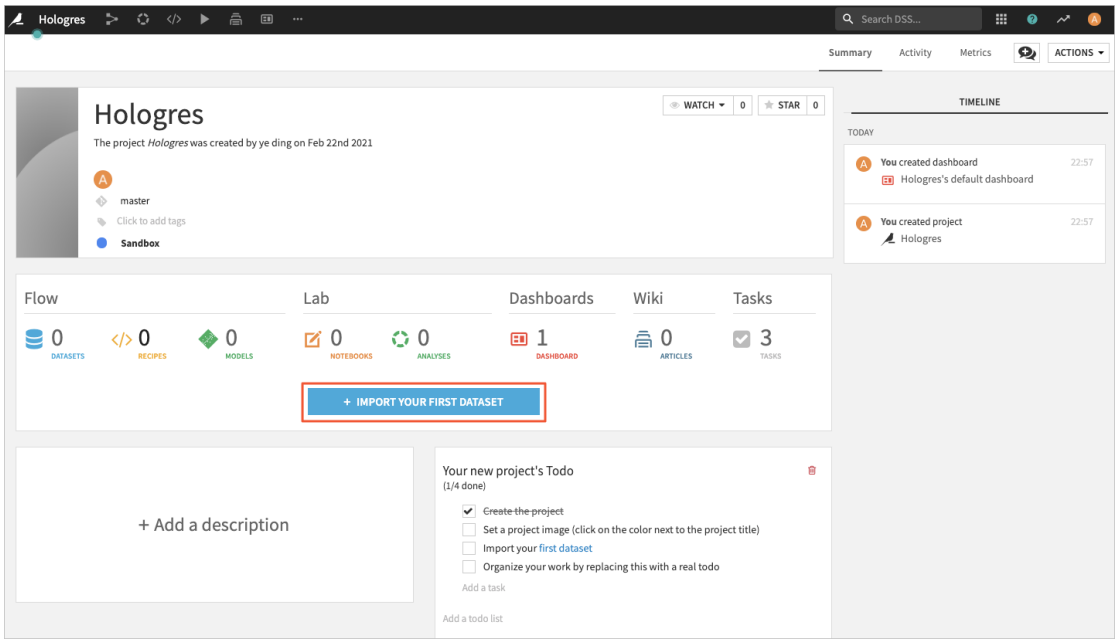
   安装Dataiku，详情请参见Dataiku官方文档。

2. 连接Hologres

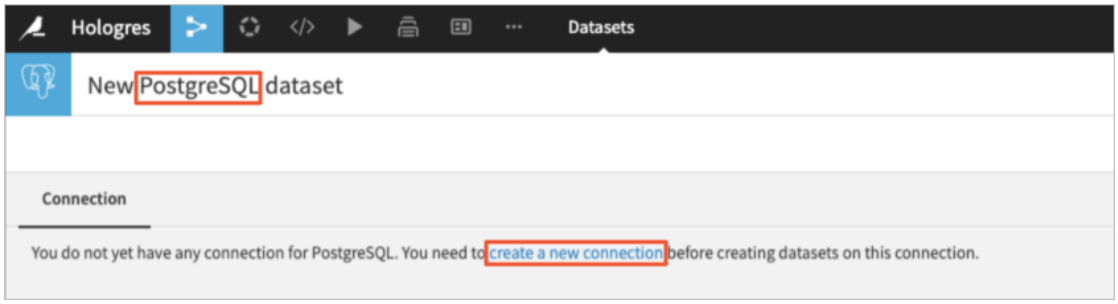
   i. 登录Dataiku，单击**Blank project**创建一个项目，本示例的项目命名为Hologres。



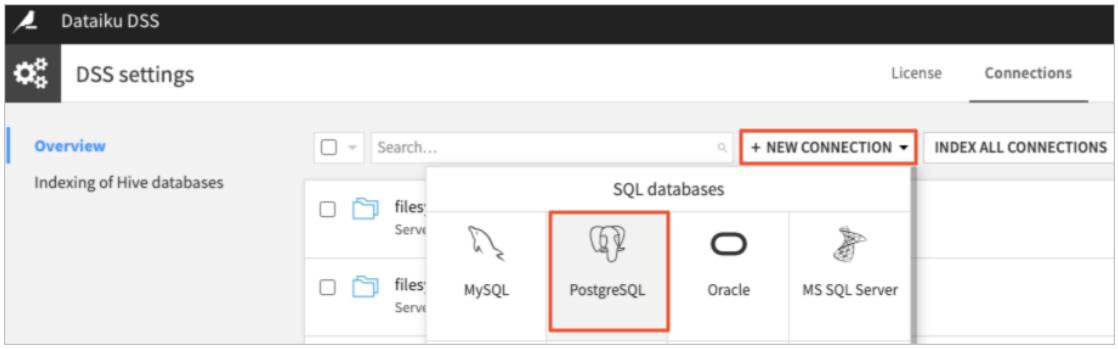
ii. 在项目详情页面，单击+IMPORT YOUR FIRST DATASET进入数据源页面。



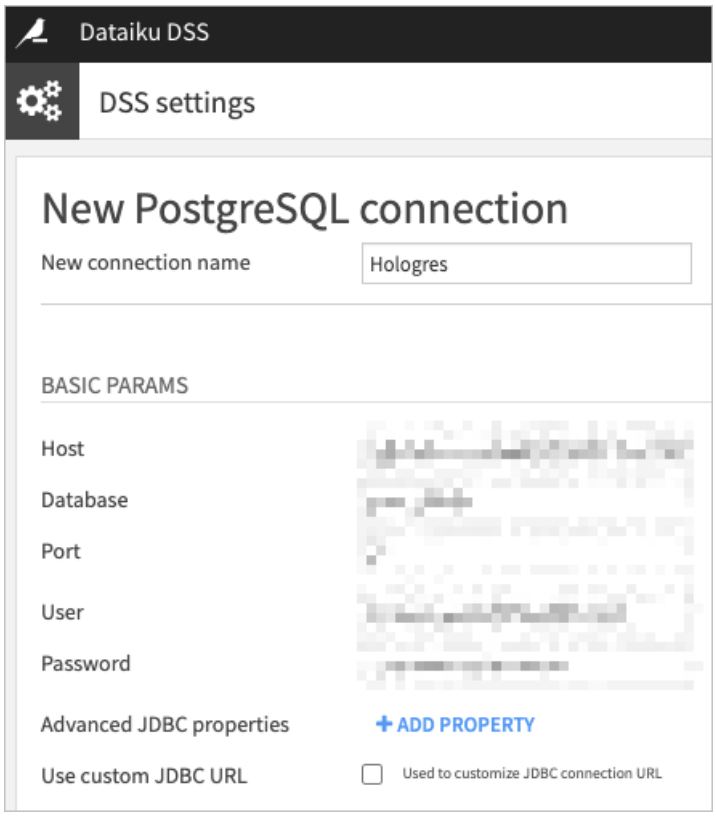
iii. 在数据源页面，选择SQL数据源类型下的PostgreSQL。在页面单击create a new connection。



iv. 在页面上方单击+NEW CONNECTION，从下拉框中选择PostgreSQL，创建新的连接。



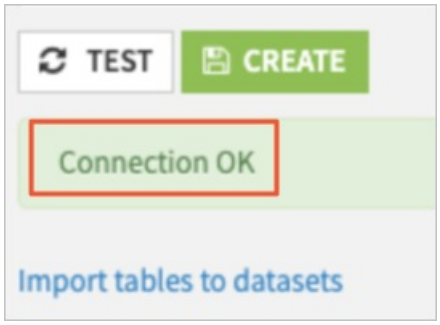
v. 根据需求填写BASIC PARAMS，如有需要您也可以填写schema参数。



具体参数配置如下所示：

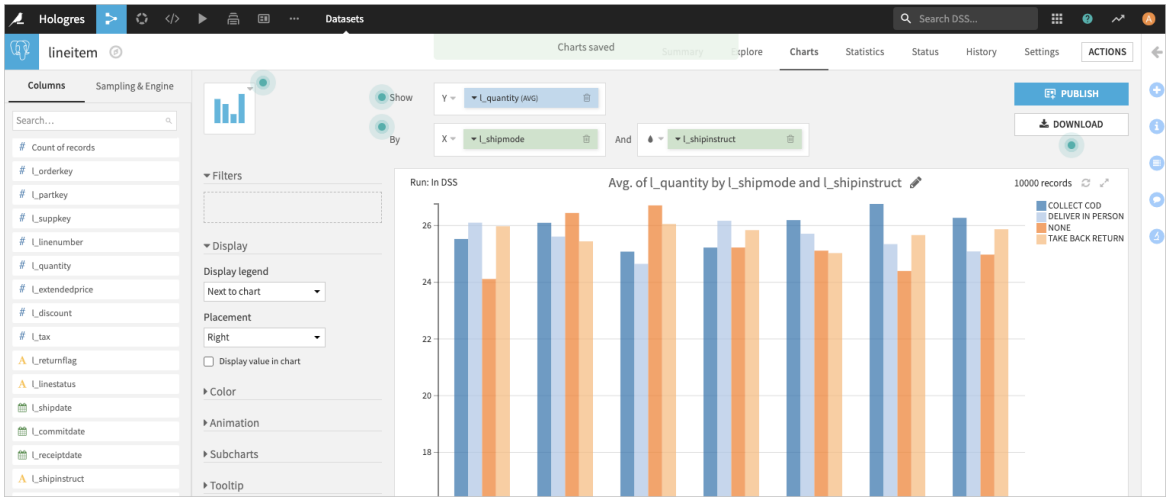
参数	描述
New connection name	输入您的连接名称。
Host	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
Port	Hologres的实例端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。
Database	Hologres的数据库名称。
User	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
更多参数	其余更多参数说明，请参见Dataiku官方文档。

vi. 参数填写完成后，单击页面下方的TEST，如果提示可以成功连接，您就可以单击CREATE创建连接。



3. 数据分析

完成配置之后，您可以单击页面底部的Import tables to datasets批量导入数据库中的表结构到数据集。导入之后您可以进行数据分析，更多关于数据分析的操作请参见Dataiku官方文档。



7.5. DataV

本文为您介绍DataV如何连接交互式分析Hologres。

背景信息

DataV是数据可视化分析BI产品，支持使用拖拽的方式在图形化编辑界面配置样式和数据，搭建数据可视化应用，满足多种业务的展示需求。

交互式分析独立数据源与DataV深度合作，您可以将Hologres查询的数据快速对接DataV，实现数据的可视化展示。

操作步骤

1. 连接Hologres。

i. 进入DataV管理控制台，在**我的数据**页面中，单击**添加数据**。

ii. 配置**添加数据**对话框的参数，单击**确定**。

参数名称	描述	说明
类型	选择**交互式分析 Hologres**。	无
名称	数据源的显示名称。	无
域名	Hologres实例的公网或经典网络（内网）地址。	进入Hologres管理控制台的实例详情页，从**实例配置**获取域名。
用户名	当前账号的AccessKey ID。	您可以单击AccessKey 管理，获取用户名。
密码	当前账号的AccessKey Secret。	您可以单击AccessKey 管理，获取密码。
端口	Hologres实例的公网端口。	进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
数据库	当前所选数据库的名称。	无

2. 可视化分析展示数据。

   DataV成功连接Hologres后，您可以选择已创建的模板或使用空白模版，以可视化的方式分析展示数据，详情请参见创建可视化应用（已创建的模板）或创建可视化应用（空白模板）。

7.6. DataWorks数据服务

本文为您介绍交互式分析Hologres如何连接DataWorks数据服务并生成API。

前提条件

- 开通DataWorks，详情请参见入门概述。
- 开通Hologres实例，并绑定至DataWorks工作空间。

背景信息

DataWorks数据服务旨在为您搭建统一的数据服务总线，支持快速将数据表生成数据API，并注册现有API至数据服务平台，帮助您统一管理和发布API服务。

Hologres与DataWorks深度集成，支持直接对接DataWorks数据服务，帮助您快速将查询的数据生成API，并提供对外服务。

操作步骤

1. 配置Hologres数据源。
    i. 登录DataWorks管理控制台。
    ii. 在左侧导航栏，单击**工作空间列表**。
    iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据集成**。
    iv. 在左侧导航栏，单击**数据源**。
    v. 在**数据源管理**页面，单击顶部菜单栏的**新增数据源**。
    vi. 在**大数据存储**区域，选择**Hologres**。
    vii. 配置**新增Hologres数据源**对话框的各项参数。

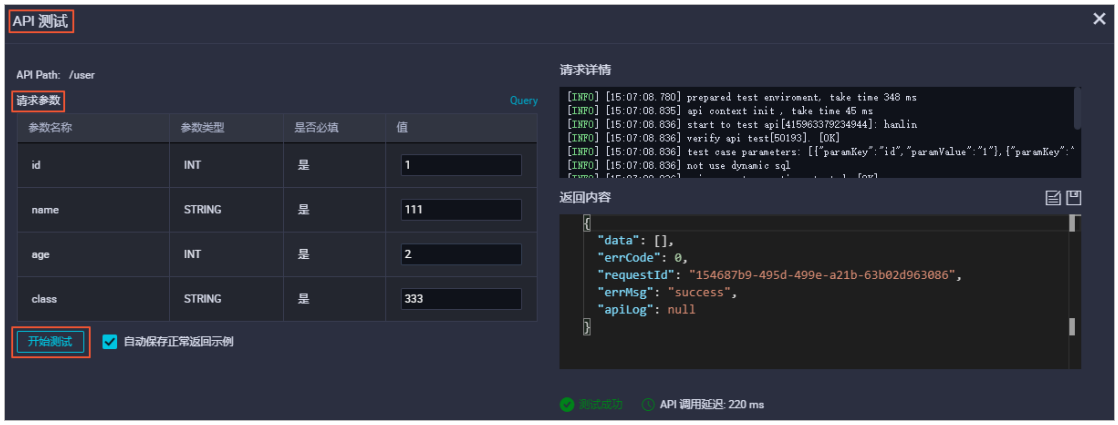
参数	描述
数据源类型	目前仅支持**阿里云实例模式**。
数据源名称	数据源名称必须是字母、数字和下划线的组合，并且以字母开头。
数据源描述	数据源的信息描述，不得超过80个字符。
适用环境	■ **开发** ■ **生产** **说明** 如果使用的是标准模式的DataWorks工作空间，则需要配置该参数。
实例ID	需要同步的Hologres实例ID。 您可以进入Hologres管理控制台，获取实例ID。
数据库名	Hologres的数据库名称。

参数	描述
AccessKey ID	您可以单击AccessKey 管理，获取AccessKey ID。
AccessKey Secret	您可以单击AccessKey 管理，获取AccessKey Secret。
资源组连通性	选择**数据服务**。 您需要保证公共资源组和数据源是可以联通的。

2. 生成API。
   i. 在DataWorks开发界面，单击顶部菜单栏左侧的图标。
   ii. 鼠标悬停至**全部产品**，在**数据开发**区域，单击**数据服务**。
   iii. 在**服务开发**页面，鼠标悬停至图标。
   iv. 鼠标悬停至**生成API**，您可以选择**向导模式**或**脚本模式**生成API。

使用**向导模式**生成API，详情请参见通过向导模式生成API。
使用**脚本模式**生成API，详情请参见通过脚本模式生成API。

3. 测试API。
   i. 在API编辑页面，单击顶部菜单栏右侧的**测试**。
   ii. 在**API测试**对话框，输入**请求参数**，单击**开始测试**。



如果**API测试**对话框页面底部显示**测试成功**，则表示API测试通过。
您也可以使用数据服务的测试API模块来完成测试，详情请参见测试API。

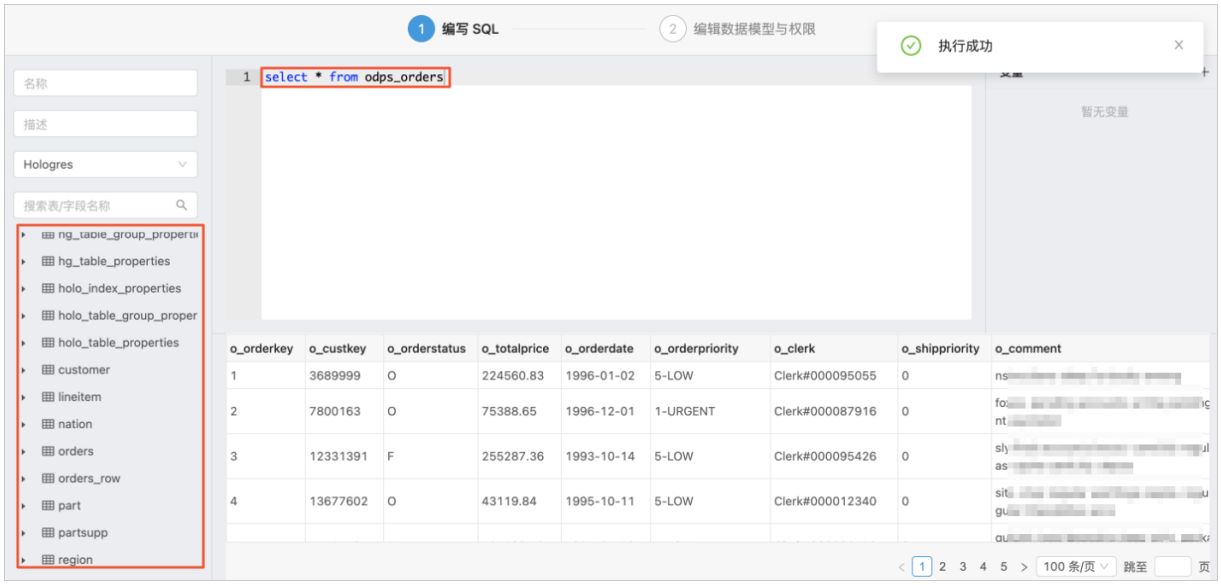
4. 发布并查看API。
   i. 在API编辑页面，单击顶部菜单栏右侧的**发布**。
      发布API至API网关，并上架至API市场，详情请参见发布API。
   ii. 在**数据服务**页面，单击顶部菜单栏右侧的**服务管理**。
   iii. 单击已发布的API名称，查看API详情。
5. 调用API。
   如果您需要调用已成功发布的API，请参见客户端调用API示例。

7.7. Davinci

Davinci是宜信数据团队开源的一款商业智能（Business Intelligence）产品。本文将为您介绍Davinci如何连接Hologres并进行可视化分析。

注意事项

Davinci默认不读取PostgreSQL中外部表的表结构，所以暂时无法在页面中查询到外部表元数据，但是您可以直接使用SQL语句进行查询。



1. 安装Davinci
   安装Davinci，详情请参见Davinci官方文档。
2. 安装JDBC驱动
   i. 请前往PostgreSQL官网下载42.2.18以上版本的JDBC驱动。
   ii. 将下载的PostgreSQL的JDBC驱动包放置于*<Davinci安装目录>\lib*目录下。
   iii. Davinci默认支持的不是PostgreSQL数据源，您需要配置自定义数据源。进入*<Davinci安装目录>\lib*目录，运行如下命令，创建一个自定义数据源配置文件。

```
mv datasource_driver.yml.example datasource_driver.yml
```

iv. 编辑该文件，添加如下内容，配置PostgreSQL数据源。

```
postgresql:
    name: postgresql
    desc: postgresql
    version:
    driver: org.postgresql.Driver
    keyword_prefix: \"
    keyword_suffix: \"
    alias_prefix: \"
    alias_suffix: \"
```

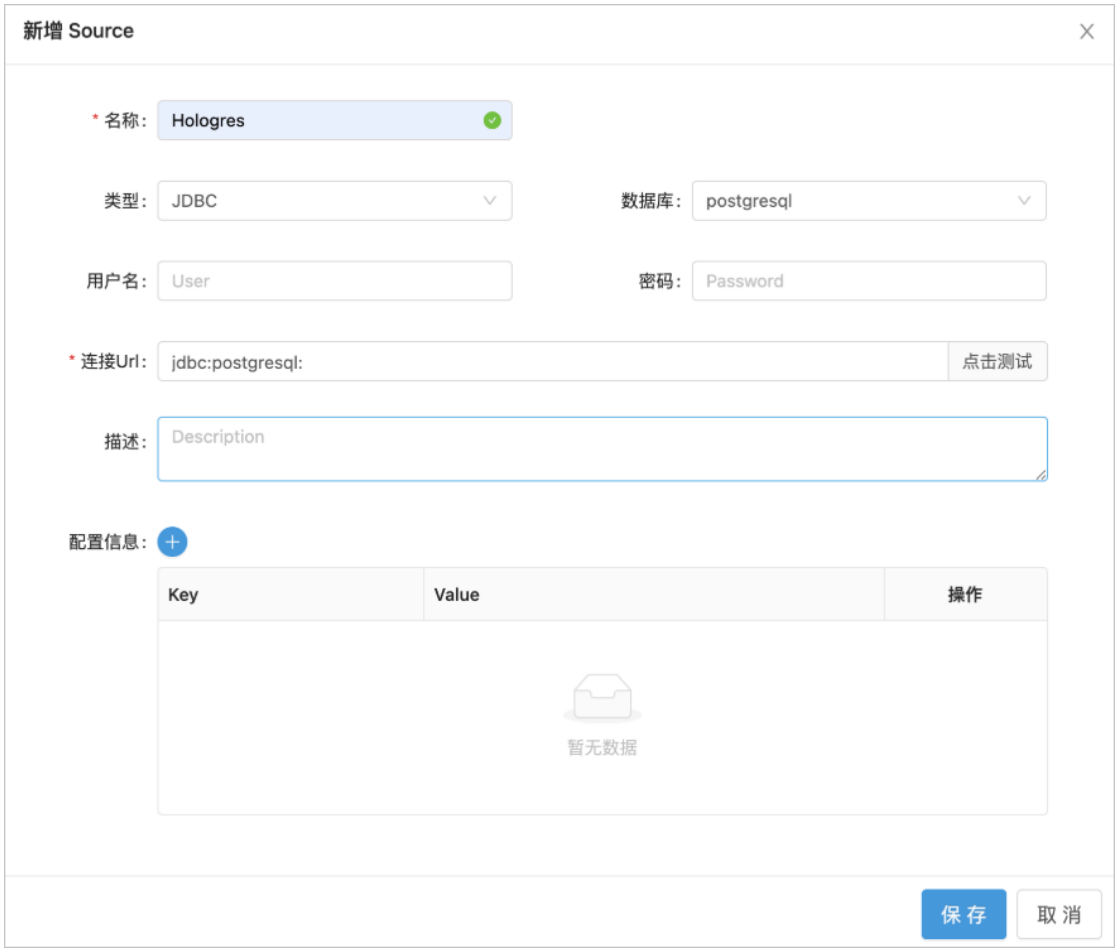
v. 重启Davinci，完成JDBC驱动安装和PostgreSQL数据源配置。

3. 连接Hologres

i. 登录Davinci，单击页面右上角的**+创建**，创建一个项目。

ii. 创建项目完成后，单击进入项目。在左侧导航栏单击图标，进入数据源页面。单击右上角的图标，创建数据源。

iii. 在**新增Source**页面，选择此前配置的PostgreSQL数据源，配置数据源信息。



参数	说明
名称	您可以输入数据源的名称。如Hologres。
类型	选择 `JDBC` 。
数据库	选择此前配置的数据源 `postgresql` 。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
数据连接URL	数据连接URL，填写格式为 `jdbc:postgresql://<host>:<port>/<databasename>` ，其中各参数解释如下所示： ■ host：Hologres实例的公共或VPC网络地址，具体按部署环境决定。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取。 ■ port：端口，Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。 ■ databasename：数据库名称，Hologres创建的数据库名称。

iv. 单击**点击测试**按钮，如果页面提示**测试成功**，表示能够正常连接数据库。

v. 单击**保存**，即完成Hologres数据连接的配置。

4. 数据分析
   您可以利用Hologres进行数据可视化操作，更多数据操作指导请参见Davinci官方文档

7.8. FineBI

FineBI是帆软软件有限公司推出的一款商业智能（Business Intelligence）产品。FineBI自助分析以业务需求为方向，通过便捷的数据处理和管控，提供自由的探索分析。本文为您介绍FineBI如何连接Hologres进行可视化分析。

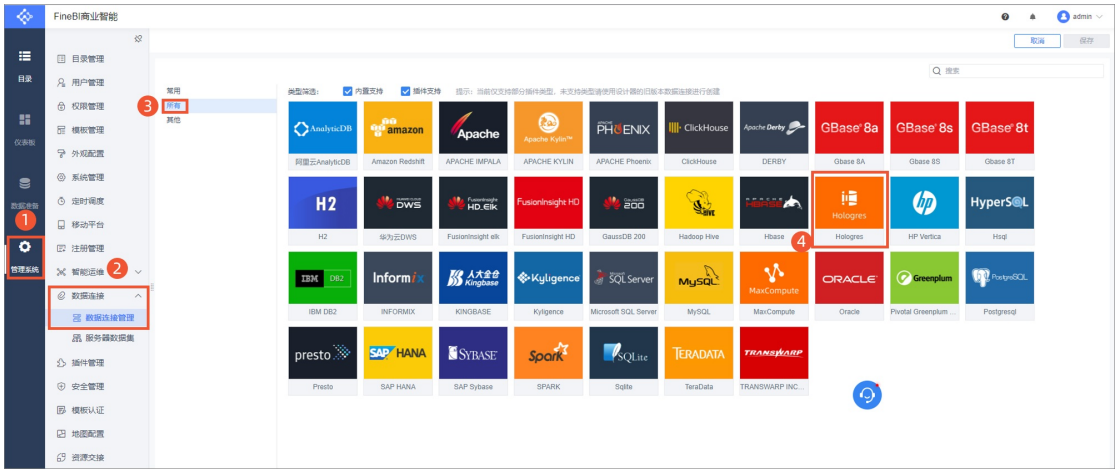
连接Hologres

1. 安装FineBI。
   安装FineBI，详细步骤请参见FineBI官方文档。
2. 安装JDBC驱动。
   请前往PostgreSQL官网下载JDBC驱动。

   > **说明**
   > - 请您下载并使用42.2.18以上版本的JDBC驱动。
   > - 您需要将下载的PostgreSQL的JDBC驱动包放置于*<FineBI安装目录>\webapps\webroot\WEB-INF\lib*目录下，并重启FineBI。

3. 连接Hologres。
   i. 使用管理员账号登录FineBI，选择**管理系统 > 数据连接 > 数据连接管理**。
   ii. 单击**新建数据连接**，在**所有**页签下选择**Hologres**。

   > **说明** 如您的FineBI版本低于5.1.14，也可以使用PostgreSQL数据源连接Hologres。

   iii. 在**Hologres**页面配置连接信息。

具体参数说明如下：

参数	说明
数据连接名称	您可以输入数据连接的名称。如Hologres。
驱动	驱动选择 `org.postgresql.Driver` 。
数据库名称	Hologres的数据库名称。
主机	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
端口	Hologres的实例端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
数据连接URL	数据连接URL，填写格式为 `jdbc:postgresql://<host>:<port>/<databasename>` ，其中各参数解释如下所示： ■ host：主机，Hologres实例的公共网络地址。 ■ port：端口，Hologres实例的端口。 ■ databasename：数据库名称，Hologres创建的数据库名称。

参数	说明
高级设置	该示例不涉及高级设置。如业务涉及配置，更多内容请参见FineBI官方文档。

iv. 配置完以上参数，在页面中单击**点击连接数据库**，读取模式列表并从下拉列表中选择您需要使用的Schema。

v. 单击页面右上角的**测试连接**，如果提示**连接成功**表示FineBI能够正常连接Hologres数据库。

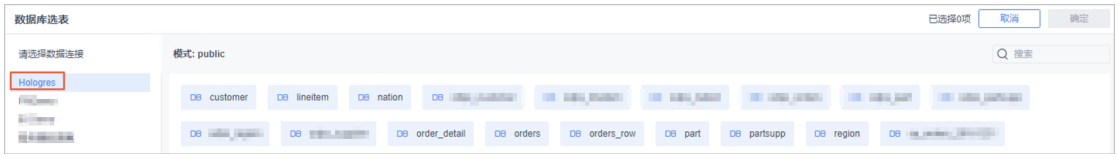
vi. 单击页面右上角的**保存**，完成数据连接的配置。

4. 添加数据表。

i. 在左侧导航栏单击数据准备，进入数据列表页。

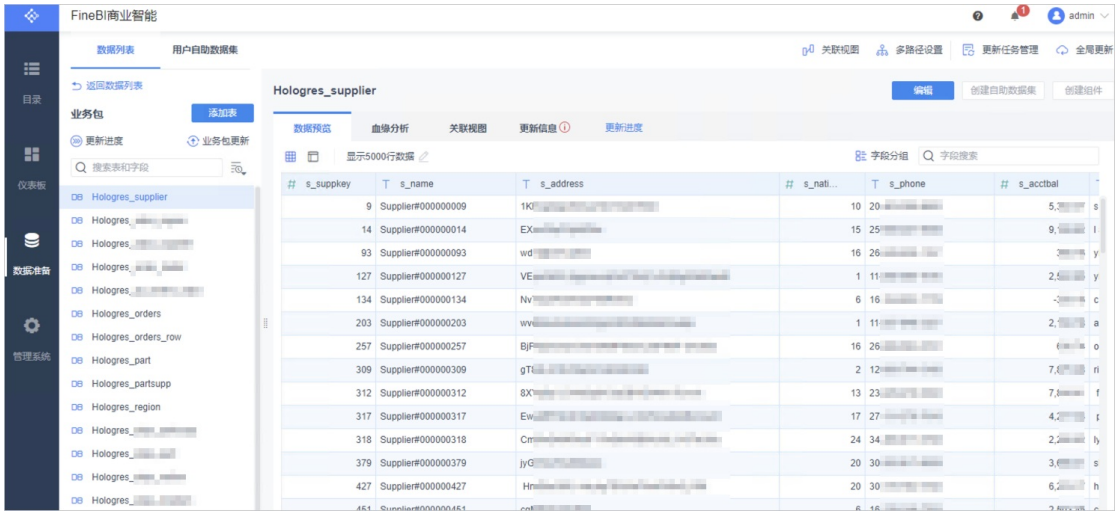
ii. 单击进入目标业务包详情页，单击**添加表**。

iii. 您可以使用数据库表、SQL数据集等方式从Hologres中加载数据，此处以数据库表为例获取数据表信息。单击**数据库表**进入数据库选表页面，可以选择已创建的数据连接，右侧会列出数据连接的数据库对应Schema中的所有数据表。



iv. 选择需要加载的数据表，单击页面右上角的**确定**，FineBI即会加载选中的数据表，已添加的表灰化不可选择。

v. 加载完成后，您可以在目标业务包的数据列表中选择具体的数据表，进行数据准备的相关操作。



5. 数据分析。
   当您完成数据表的加载操作后，您可以利用这些数据表进行数据可视化相关操作，更多操作指导请参见FineBI官方文档。

时间筛选控件最佳实践

在Hologres创建表时，我们可以设置Segment_key（别名event_time_column）属性，对时间类型过滤条件进行索引优化，防止全表扫描，能够加速查询。Hologres默认把表中第一个时间戳类型作为Segment_key。

您可以通过创建带参数的SQL数据集，当您的SQL的执行计划中有**Segment Filter**关键字出现时，即表示Segment_key已经生效，您可以使用带参数的SQL数据集创建的时间控件。

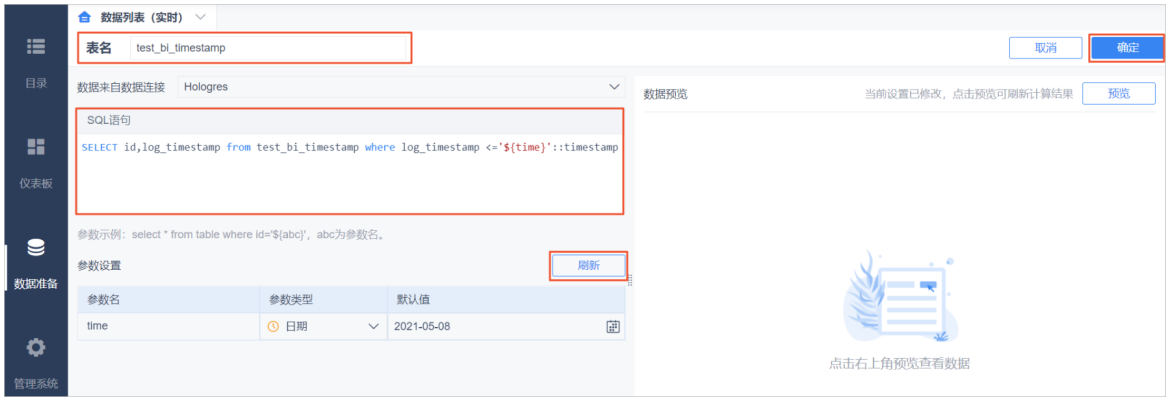
```
[1] on lineitem  (cost=0.00..1.00 rows=1000 width=24)
                                                    Segment Filter:
((l_shipdate >= '1993-01-01 00:00:00+08'::timestamp with time zone) AND (l_shipdate <= '1998-
12-31 23:59:59+08'::timestamp with time zone))
                                                    Cluster Filter:
((l_shipdate >= '1993-01-01 00:00:00+08'::timestamp with time zone) AND (l_shipdate <= '1998-
12-31 23:59:59+08'::timestamp with time zone))
Optimizer: HQO version 0.10.0
```

创建带参数的SQL数据集具体步骤如下：

1. 登录FineBI，在左侧导航栏单击数据准备，进入数据列表页。
2. 单击进入目标业务包详情页，单击**添加表**，选择**SQL数据集**。

> **说明**
> - FineBI支持带参数的数据集，详细的功能介绍，请参见FineBI官方文档。
> - 您可以根据文档描述的带参数的SQL数据集功能，使用参数来设置日期控件。

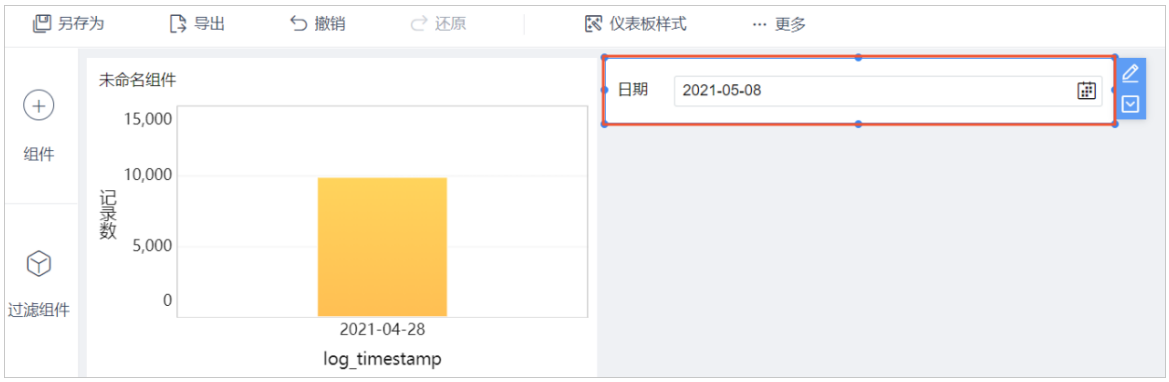
3. 在表名处填写表名，并在SQL语句模块填写如下SQL。单击**刷新**，系统可自动识别参数，您可以手动选择参数类型，单击页面右上角的**确定**保存数据集。




```
SELECT
  id,
  log_timestamp
from test_bi_timestamp
where log_timestamp <='${time}'::timestamp;
```

4. 在仪表盘中选择日期控件，选中绑定参数，从列表中选择*time*后单击**确定**。

5. 在页面设置日期并进行过滤。



6. 设置完成后，您可以通过日志查看执行的SQL。查看日志具体操作，请参见FineBI官方文档。

   生成的SQL如下所示：

```
select
  "T_13E5C8"."id" as "id",
  "T_13E5C8"."log_timestamp" as "log_timestamp"
from (
  select
      id,
      log_timestamp
  from test_bi_timestamp
  where log_timestamp <='2021-05-08'::timestamp
) as "T_13E5C8"
limit 5000;
```

生成的执行计划具体如下：

```
 Limit  (cost=0.00..0.15 rows=1000 width=16)
   ->  Gather Motion  (cost=0.00..0.14 rows=1000 width=16)
         ->  Limit  (cost=0.00..0.10 rows=1000 width=16)
               ->  Parallelism (Gather Exchange)  (cost=0.00..0.10 rows=1000 width=16)
                     ->  DecodeNode  (cost=0.00..0.10 rows=1000 width=16)
                           ->  Limit  (cost=0.00..0.10 rows=1000 width=16)
                                 ->  Index Scan using holo_index:[1] on test_bi_timesta
mp  (cost=0.00..1.00 rows=1000 width=16)
                                       Segment Filter: (log_timestamp <= '2021-05-08 00
:00:00+08'::timestamp with time zone)
 Optimizer: HQO version 0.8.0
(9 行记录)
```

当您SQL的执行计划中有**Segment Filter**关键字出现时，即表示Segment_key已经生效，您可以使用带参数的SQL数据集创建的时间控件。

7.9. FineReport

本文为您介绍交互式分析Hologres如何连接FineReport并可视化分析数据。

前提条件

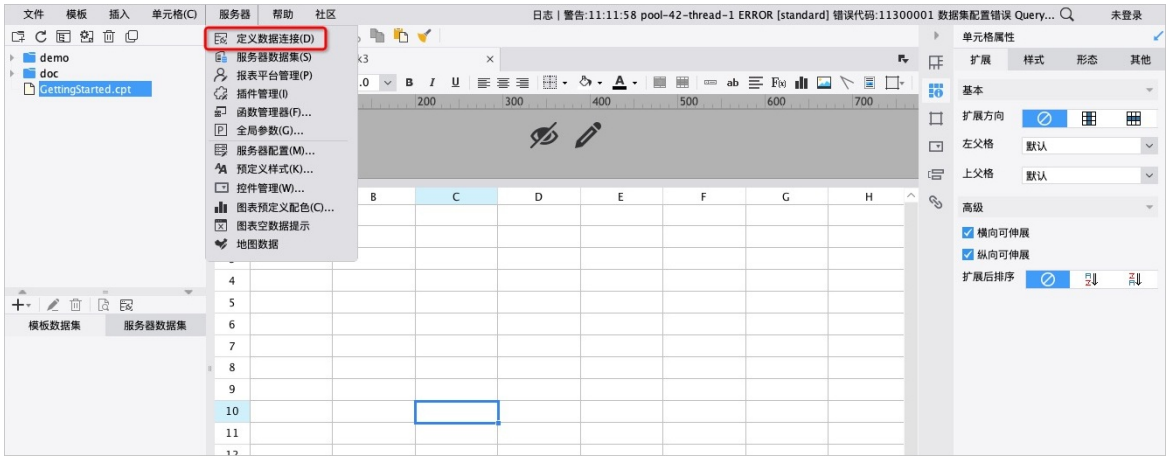
- 开通Hologres实例，详情请参见购买Hologres。
- 进入帆软官网，根据项目需求选择**解决方案**并下载客户端工具。本次示例采用FineReport报表。

背景信息

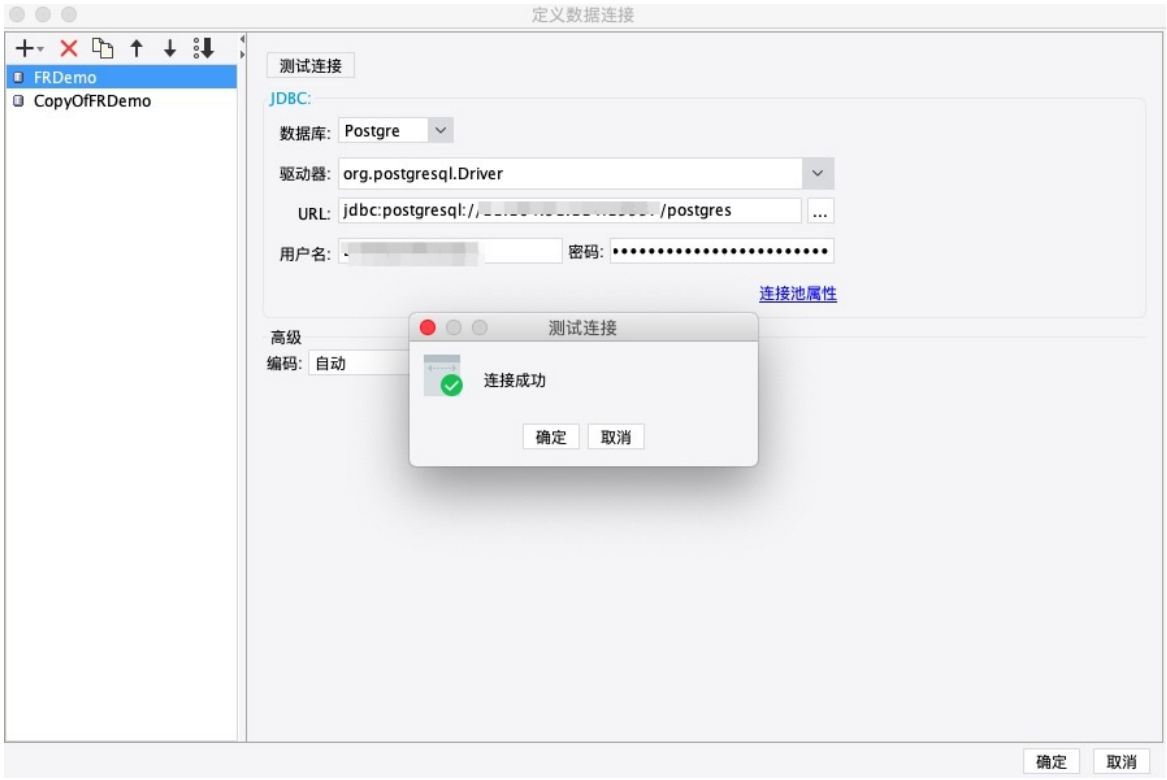
FineReport是一款专业、灵活及使用简捷的企业级Web报表软件产品，支持使用简单的拖拽方式设计出复杂的中国式报表、参数查询报表、填报表及驾驶舱，轻松搭建数据决策分析系统。

操作步骤

1. 打开FineReport客户端，在界面顶部菜单栏选择**服务器 > 定义数据连接** 。



2. 配置**定义数据连接**对话框的各项参数。



参数说明如下表所示。

参数	描述	示例
数据库	Hologres实例的数据库名称。	*testdb*
驱动器	使用FineReport自带的 `org.postgresql.Driver` 。 FineReport添加数据库后会自动生成驱动器。	无

参数	描述	示例
URL	Hologres的公共网络地址 `jdbc:postgresql://localhost:port/dbname` 。	`jdbc:postgresql://demo-cn-hangzhou.hologres.aliyuncs.com:80/postgres`
用户名	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。	无
密码	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。	无

3. 单击**测试连接**，如果弹出的对话框提示**连接成功**，则表明Hologres与FineReport已成功连接。

   > **说明** ：如果提示连接失败，请先确认是否已安装PostgreSQL驱动，详细操作请参见官网驱动安装。同时，请您安装42.2.18及以上的JDBC版本。

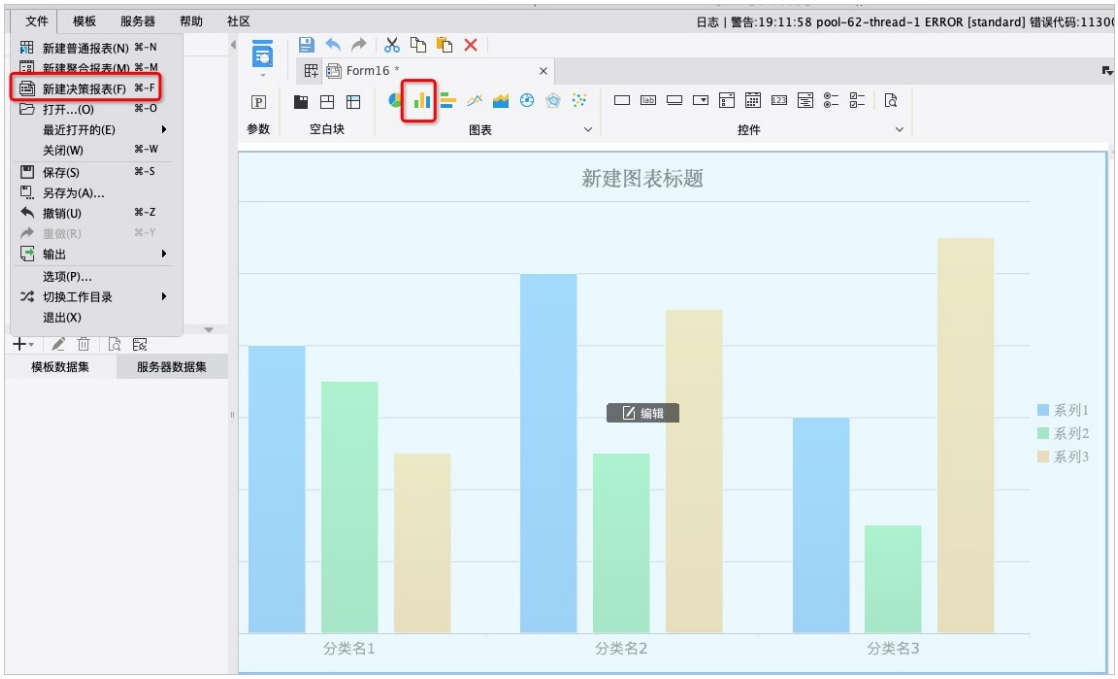
4. 单击**确定**，完成配置。
5. 可视化分析数据。

   Hologres成功连接FineReport客户端后，即可进行可视化数据分析。具体操作步骤请参见FineReport官网文档。

   > **说明** FineReport支持查询Hologres的外部表，并进行可视化分析，但是在新增数据集时不会显示外部表，您需要使用SQL语句进行查询。

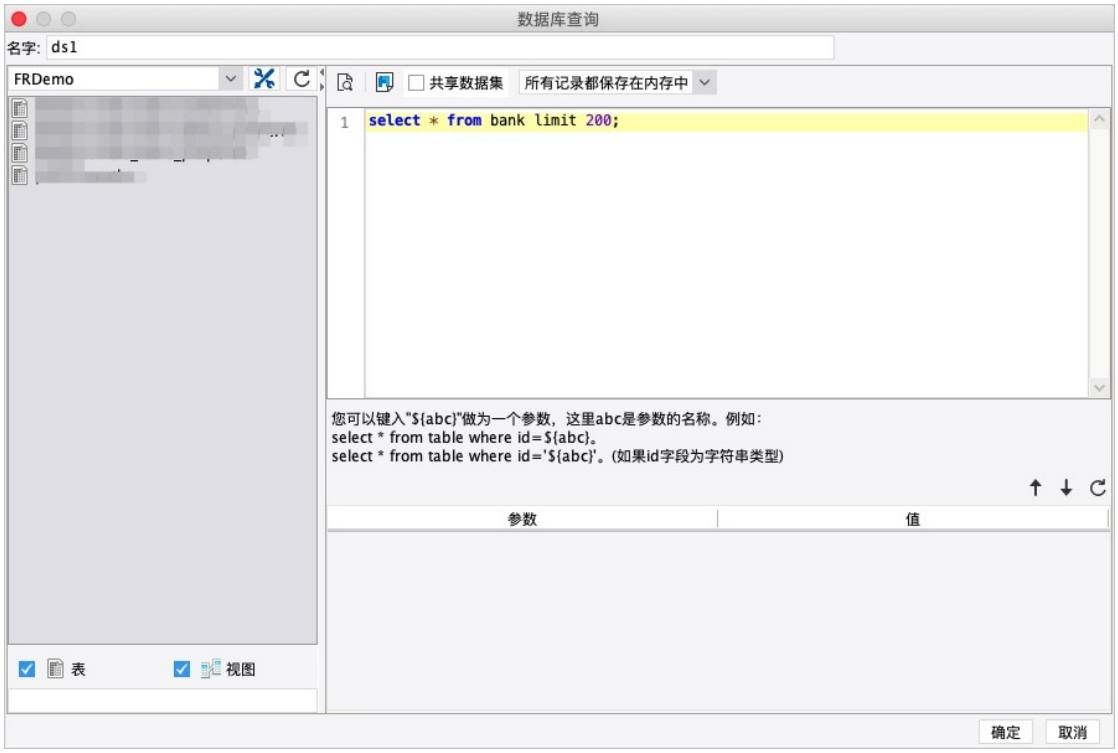
   示例使用Hologres的一张外部表数据制作柱状图，步骤如下：

i. 打开FineReport客户端，在界面顶部菜单栏单击**文件 > 新建决策报表（F）**，选择一张柱状图，如下图所示。



ii. 在左侧导航栏**模版数据集**新建**数据库查询**。

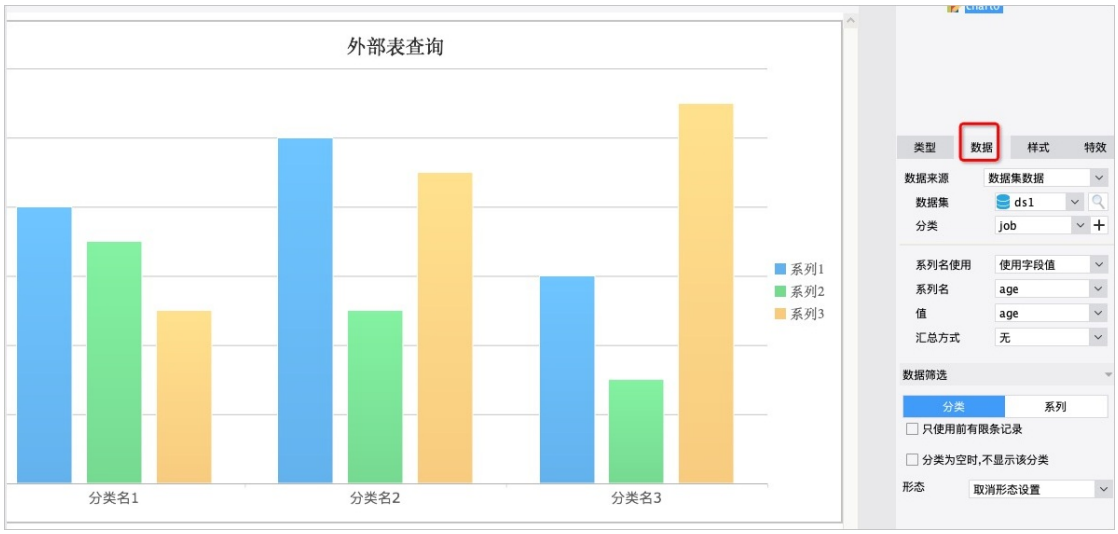
iii. 在**数据库查询**对话框，配置查询数据的SQL语句，并单击**确定**。示例查询Hologres外部表的其中200条数据，如下图所示。



> **说明** 数据库查询界面的左侧菜单栏不会显示当前Hologres数据源的外部表，您可以选择在右侧编辑框中直接查询外部表。

iv. 配置完**数据库查询**后，返回图表界面。您可以根据展示要求配置图表数据。

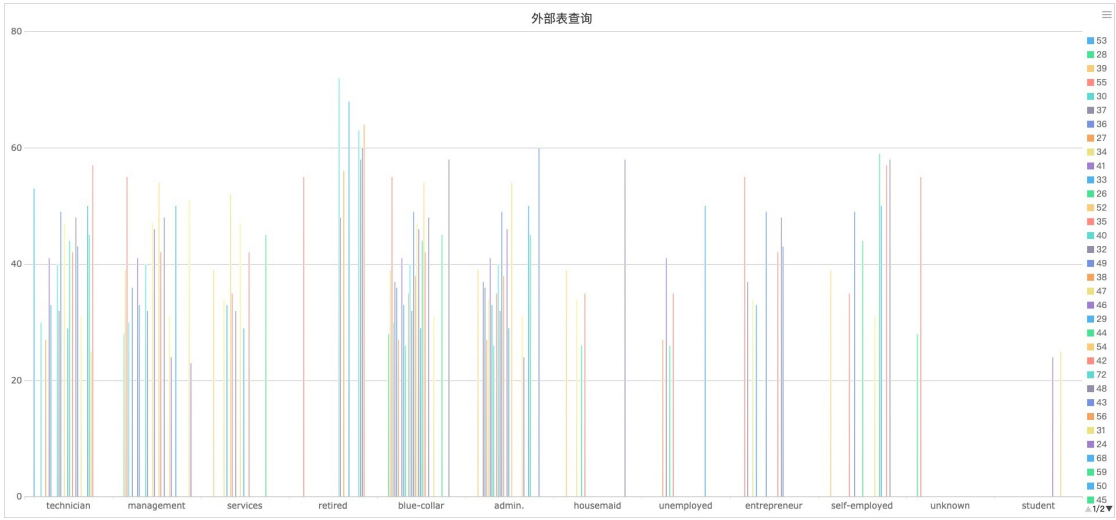
示例在图表界面右侧的**数据**栏中，为图表配置横纵坐标的数据信息（本示例横坐标为*job*数据，纵坐标为*age*数据），您可以根据需求设置图表的**样式**及**特效**。



v. 配置完图表数据，单击顶部菜单栏左侧按钮，保存图表数据。

vi. 单击按钮，查看当前数据配置的图表。

使用Hologres外部表制作的图表展示如下。



关于FineReport的更多操作，请参见帆软官网。

7.10. Grafana

本文为您介绍Grafana如何连接Hologres并可视化分析数据。

操作步骤

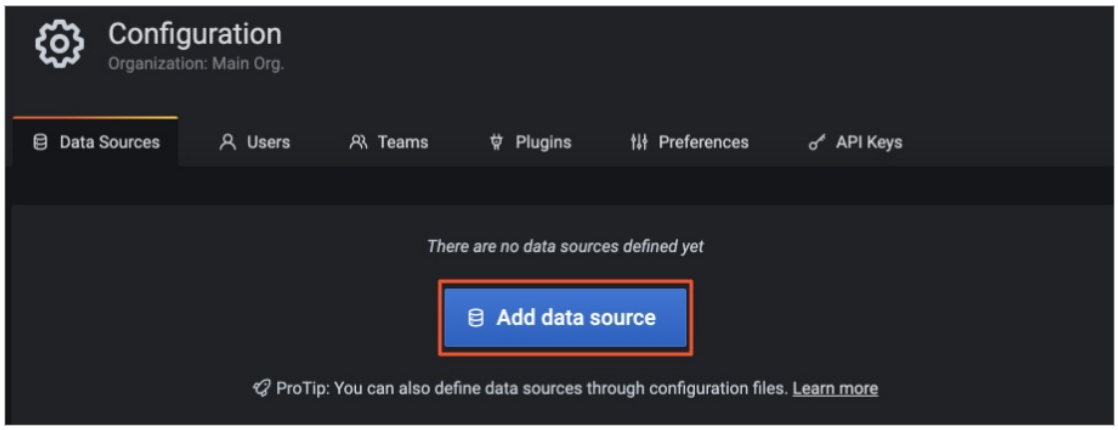
1. 安装Grafana

   安装Grafana，详情请参见Grafana官方文档。

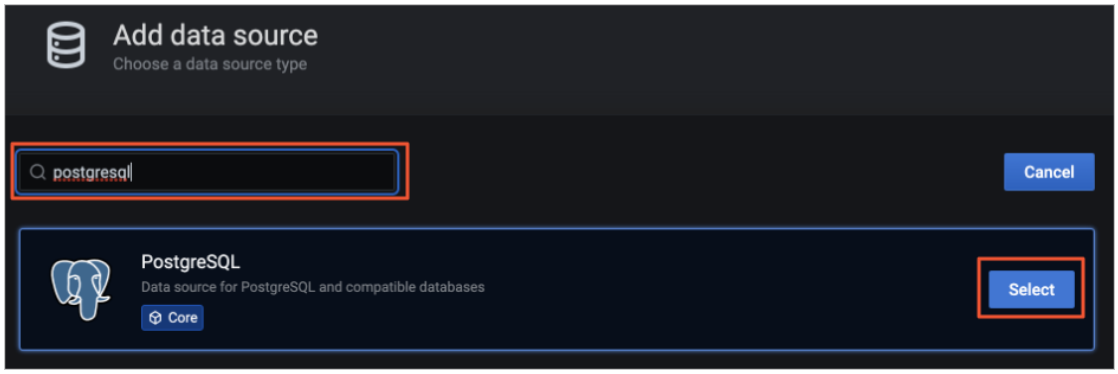
2. 连接Hologres

i. 登录Grafana，在左侧导航栏单击图标，选择Data Source创建一个数据源。

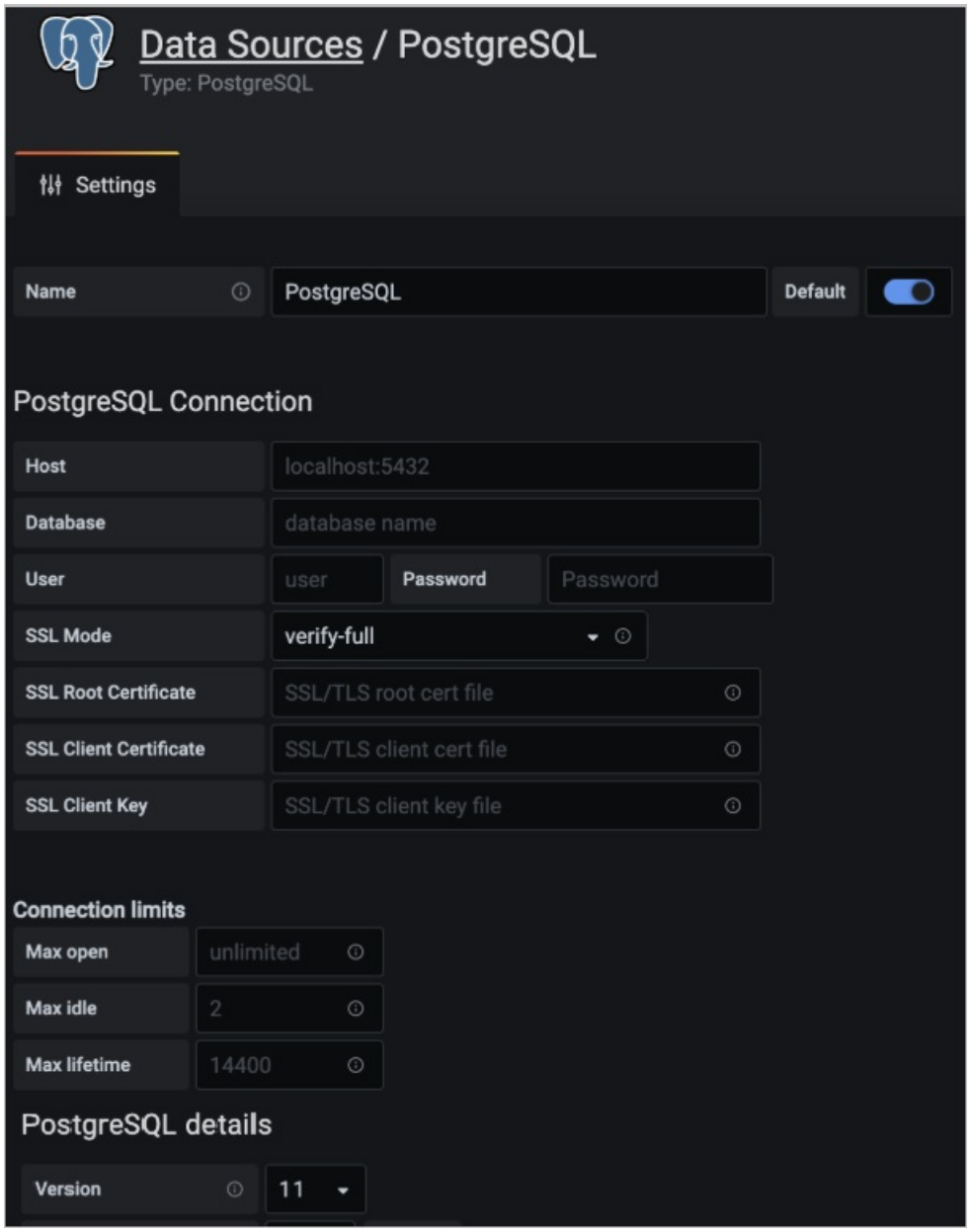
ii. 在页面中单击Add data source进入数据源页面。



iii. 在Add data source页面上方搜索框中输入PostgreSQL，找到PostgreSQL数据源后，单击Select。



iv. 根据需求填写PostgreSQL Connection的连接信息。



具体参数配置如下所示：

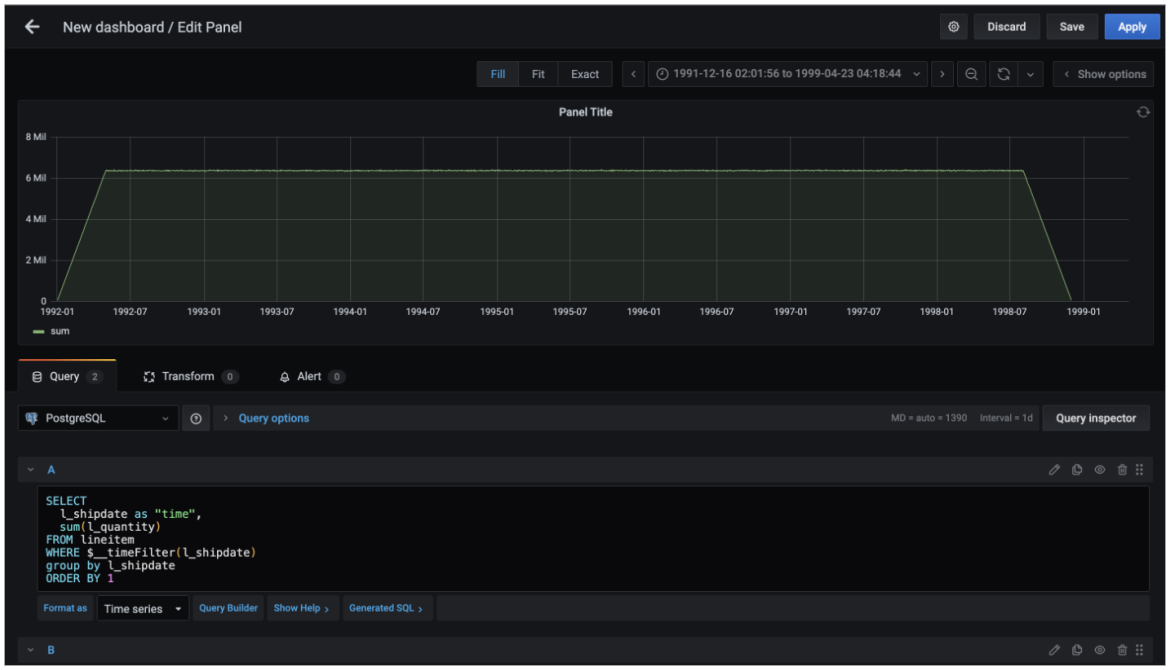
参数	描述
Host	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
Database	Hologres的数据库名称。
User	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
SSL Mode	按需配置，根据您的Endpoint是否开启SSL决定。

参数	描述
Version	PostgreSQL的版本选择11。
更多参数	其余更多参数说明，请参见Grafana官方文档。

v. 参数填写完成后，单击页面下方的**Save & Test**，如果页面提示 `Database Connection OK` 则表示可以成功连接。

3. 数据分析

完成配置之后，您可以根据该数据源进行数据分析，更多关于数据分析的操作请参见Grafana官方文档。



7.11. IBM Cognos Analytics

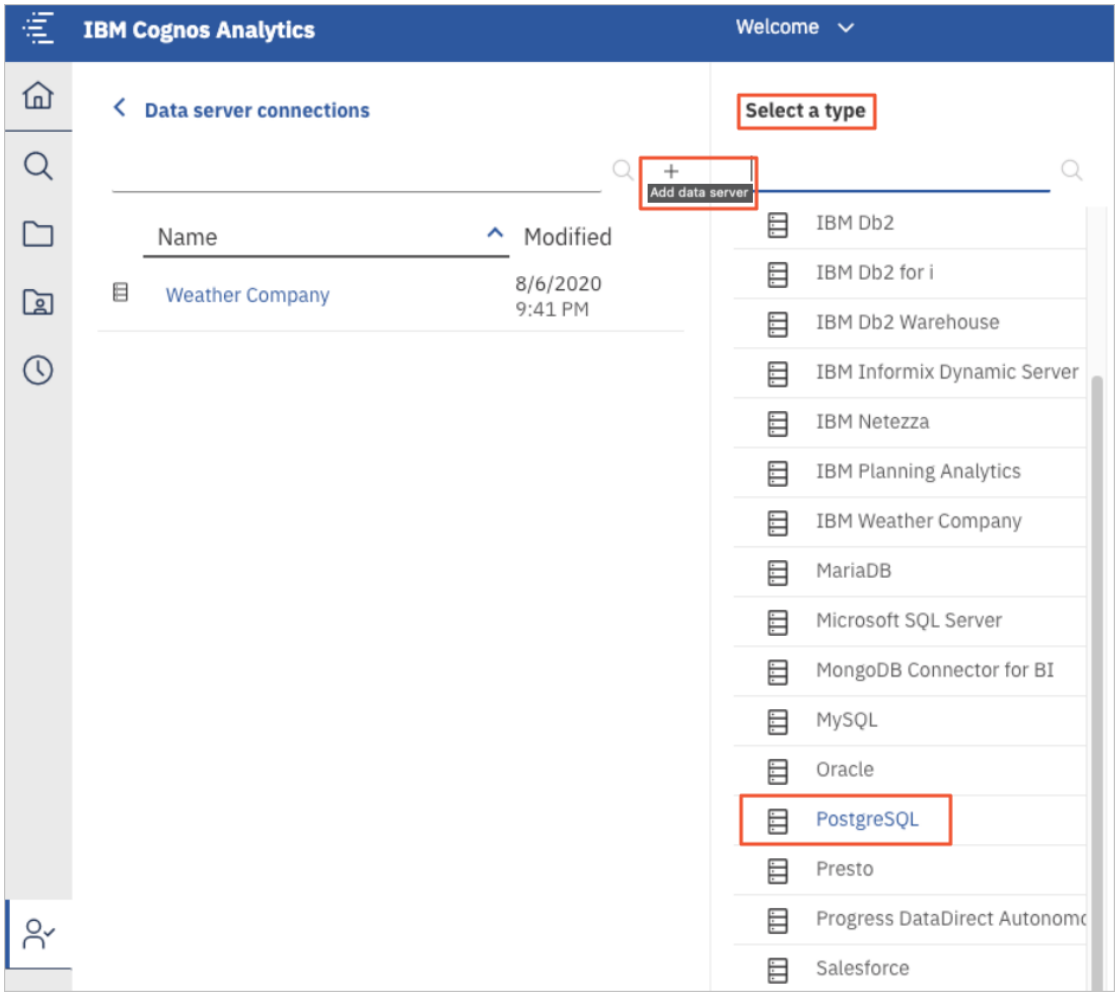
IBM Cognos Analytics可以通过轻松地连接数据、处理数据并创建可视化效果，帮助您了助制定更有信心的决策。本文为您介绍IBM Cognos Analytics如何连接Hologres并可视化分析数据。

操作步骤

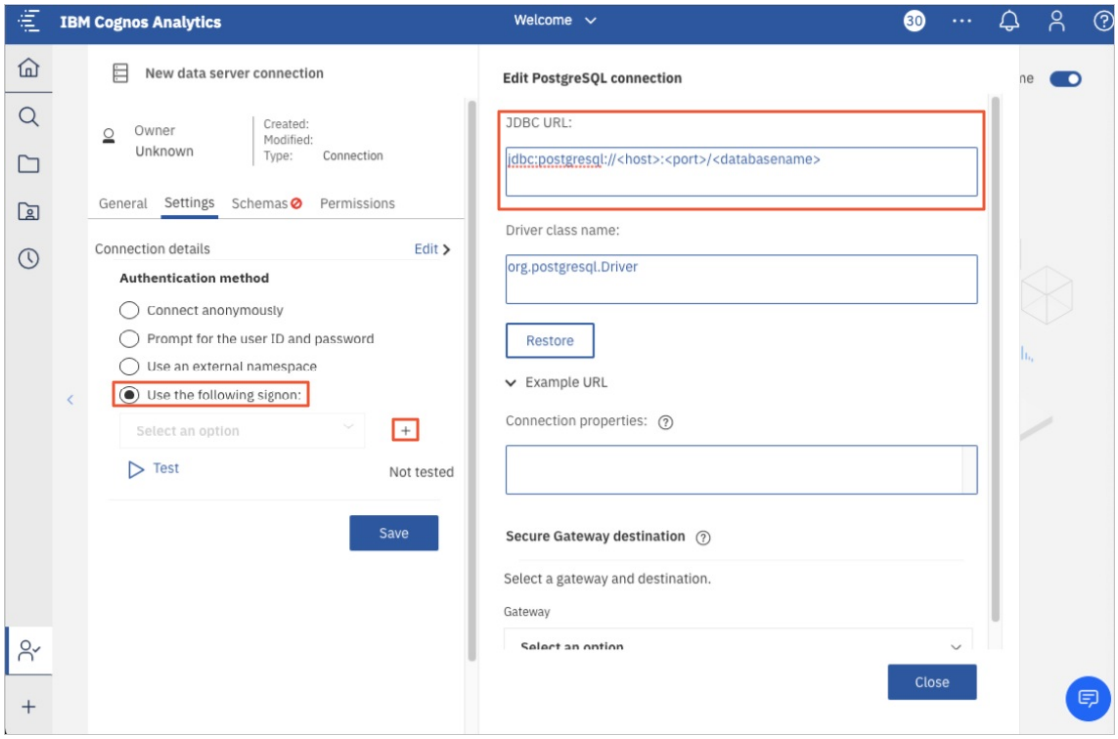
1. 安装IBM Cognos Analytics
   安装IBM Cognos Analytics，详情请参见IBM Cognos Analytics官方文档。
2. 连接Hologres

   i. 登录IBM Cognos Analytics，单击左下方的（Manage）图标，创建一个连接。

ii. 在页面上方单击+图标创建数据源连接，数据源类型选择PostgreSQL。

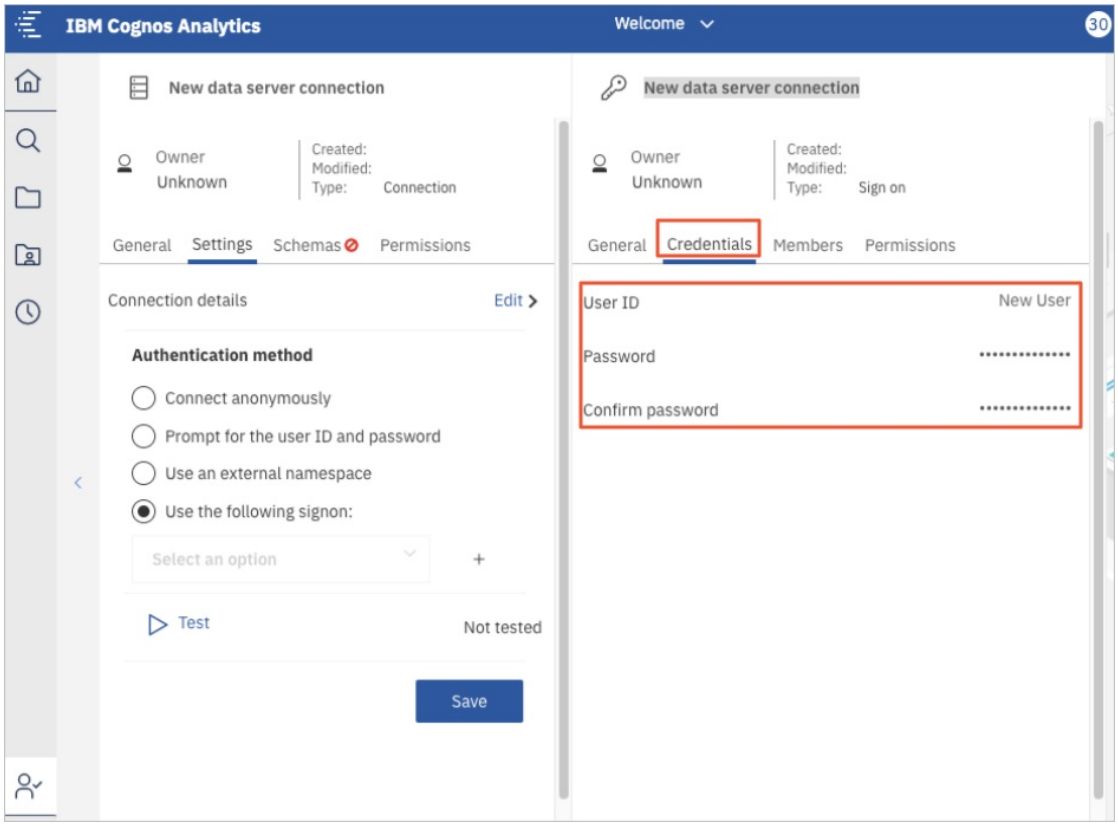


iii. 在数据源类型选择PostgreSQL后，您需要在页面配置JDBC URL和Authentication method参数信息。



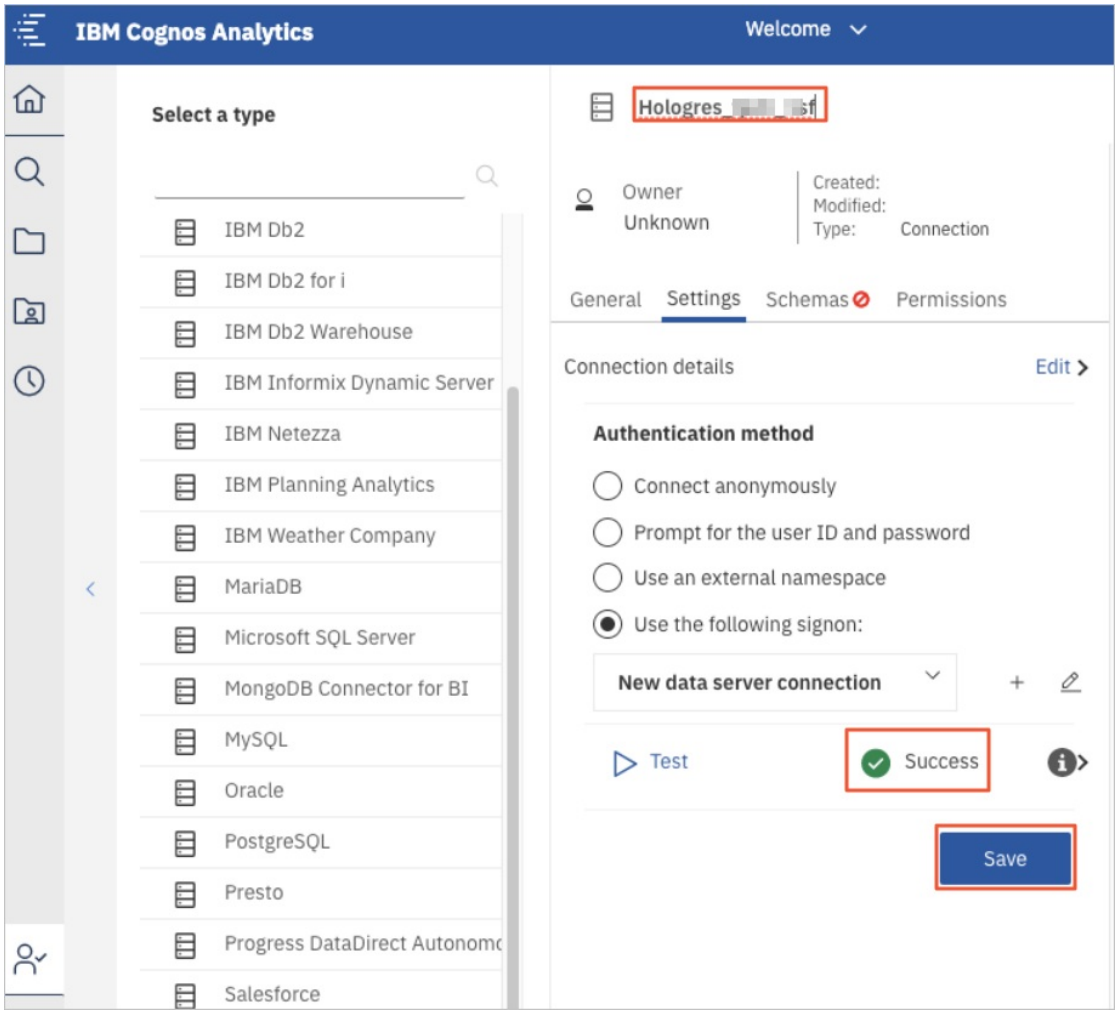
参数	说明
JDBC URL	JDBC的URL，填写格式为 `jdbc:postgresql://<host>:<port>/<databasename>` ，其中各参数解释如下所示： ■ host：Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。 ■ port：Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。 ■ databasename：Hologres创建的数据库名称。
Driver class name	该参数值保持默认，不需要修改。
Authentication method	选择Authentication method为**Use the following signon**。

iv. 单击Authentication method下方的+图标增加认证信息，在页面中输入User ID和Password。

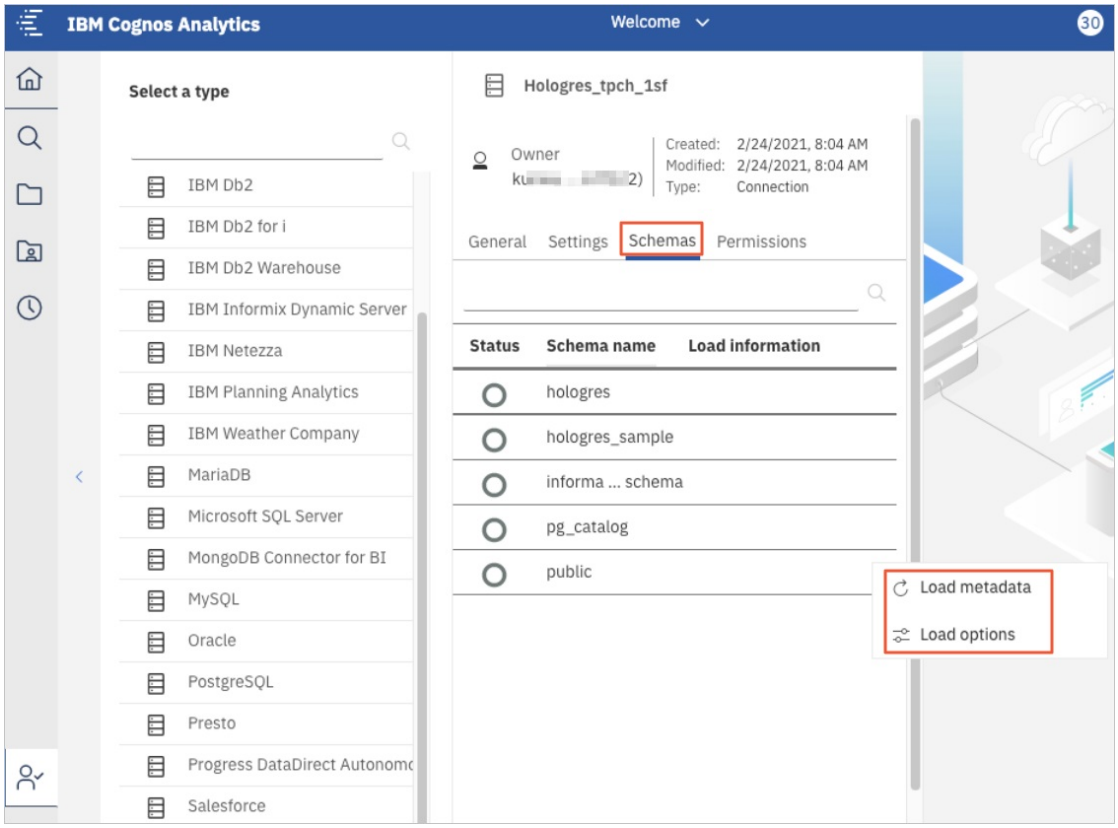


参数	说明
User ID	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

v. 配置完成后，在页面左下方单击**Test**，如果页面下方提示Success，则表示测试成功。您可以在页面上方输入连接名称后，单击**Save**。

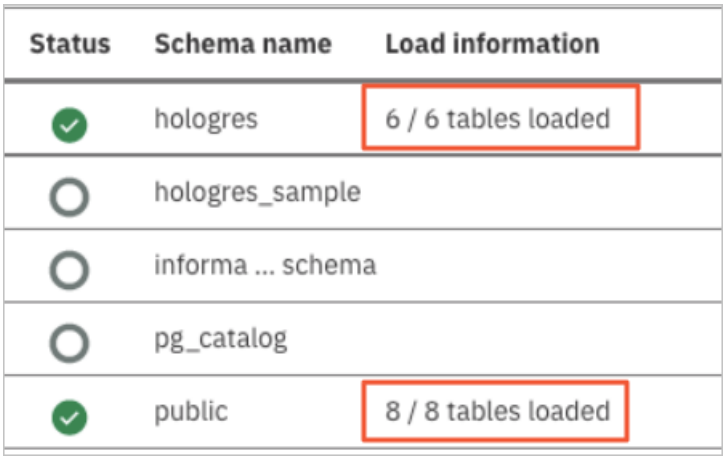


vi. 连接配置保存完毕后，您可以单击进入**Schemas**页签，选择需要加载的Schema。单击目标Schema名称最右侧的显示更多图标，在弹框中选择目标参数**Load metadata**或**Load options**。



其中，**Load metadata**表示系统会加载该Schema下所有表的元数据。**Load options**表示您可以详细的设置数据信息。本文我们以**Load metadata**为例进行说明。

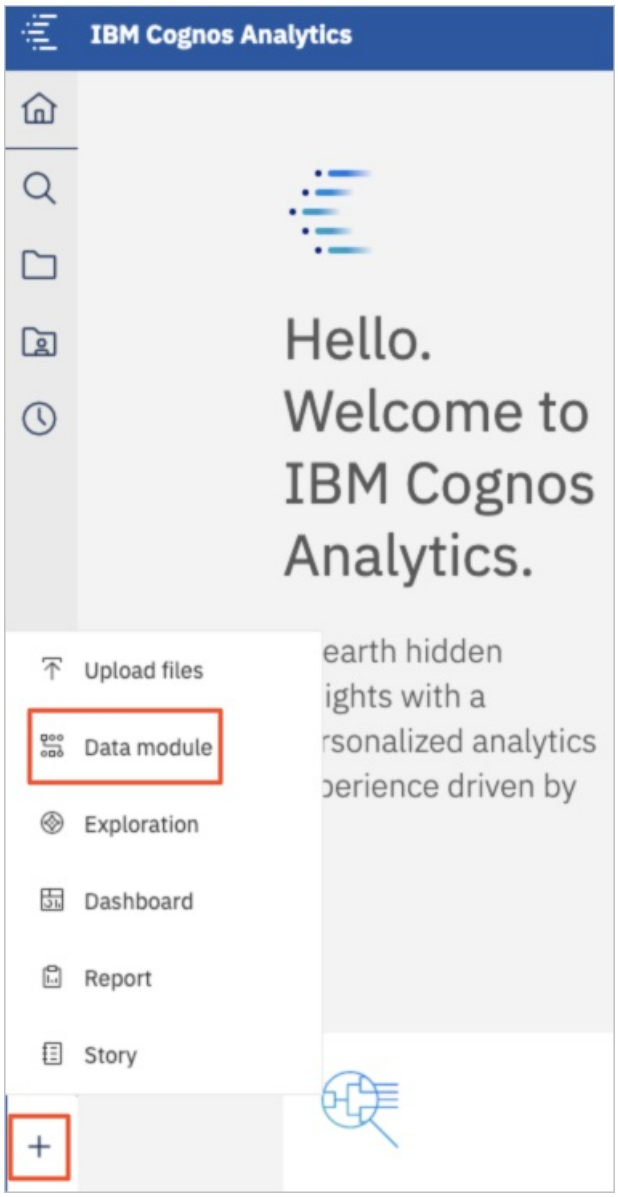
vii. Schema信息加载完毕后，系统会提示相关内容已加载，如下图所示。数据加载完毕，表示您已经成功连接Hologres并进行数据可视化作业。



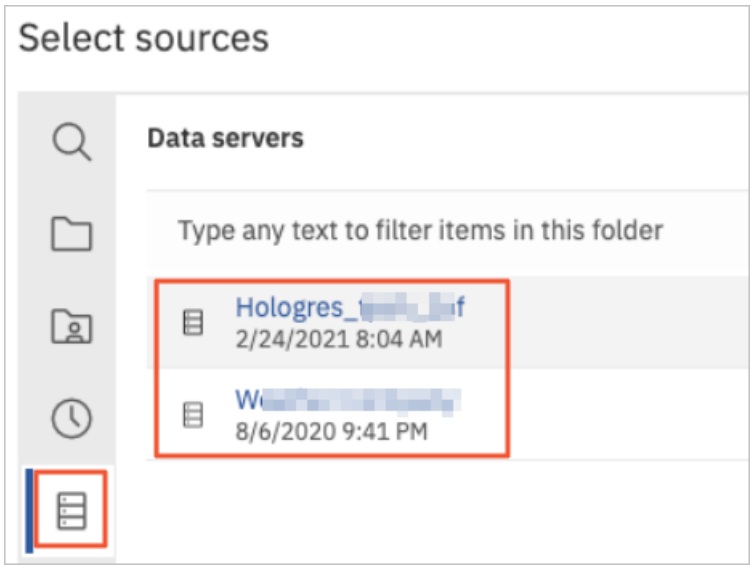
3. 创建Data module

数据连接创建完成后，您可以创建Data module进行数据分析。

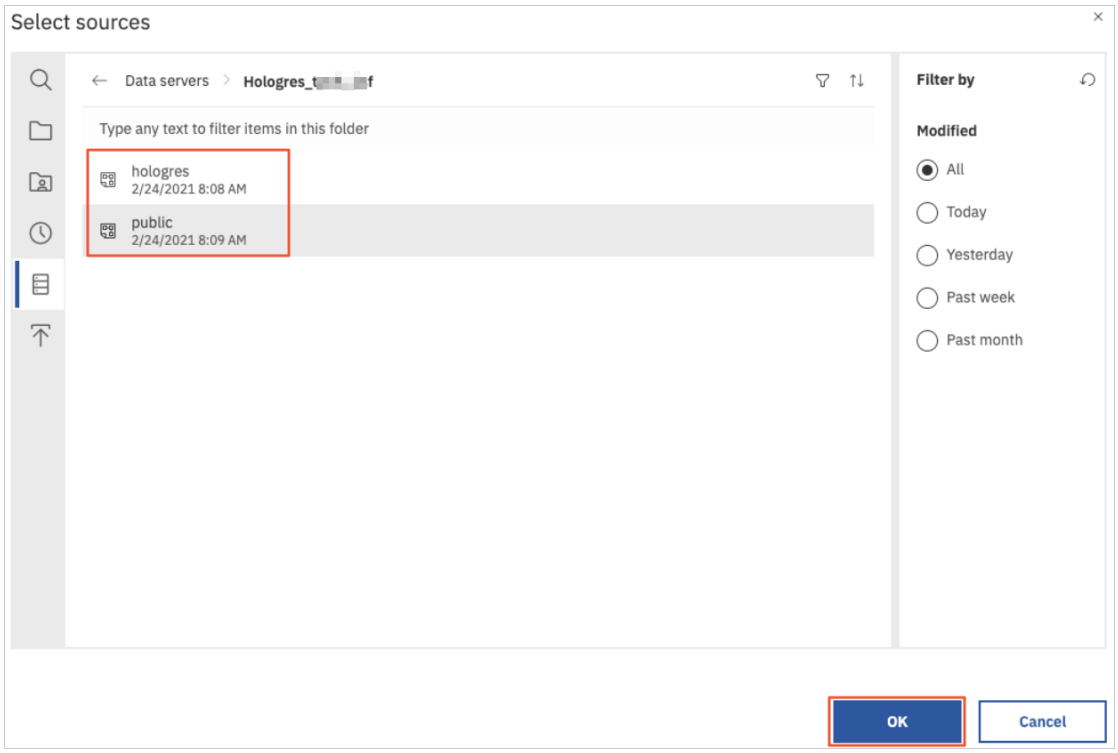
i. 登录IBM Cognos Analytics，单击左下方的+图标，在列表中选择Data module。



ii. 在Select sources页面左侧单击图标。进入Data servers页签，单击之前创建的目标数据连接。

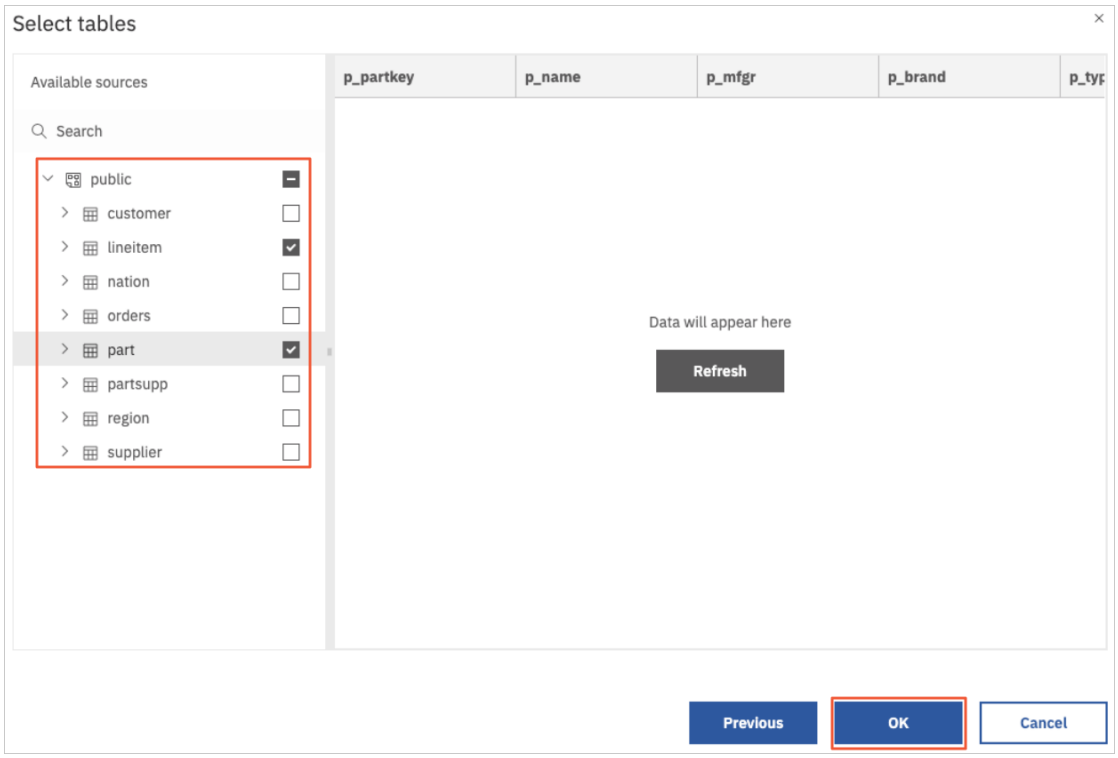


iii. 在目标数据连接的详情页，系统会显示已经加载元数据的Schema，您可以选择目标Schema并在页面下方单击OK。

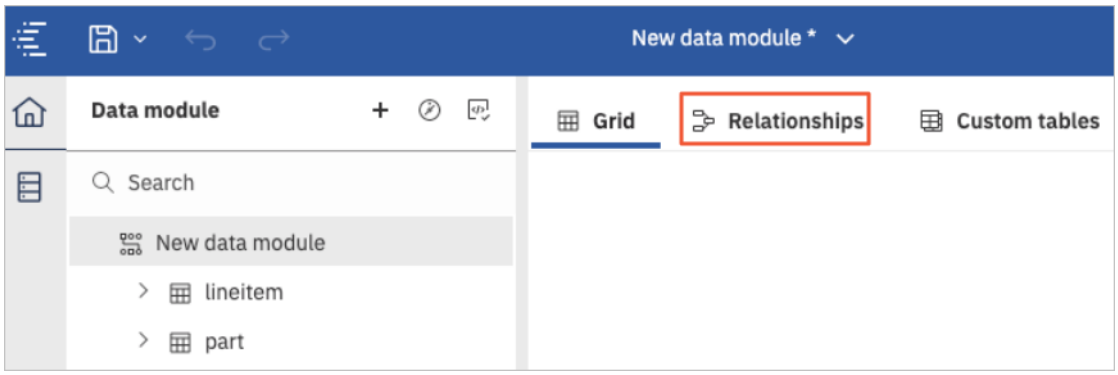


iv. 在Add tables页面，选择Select tables并在页面下方单击Next。

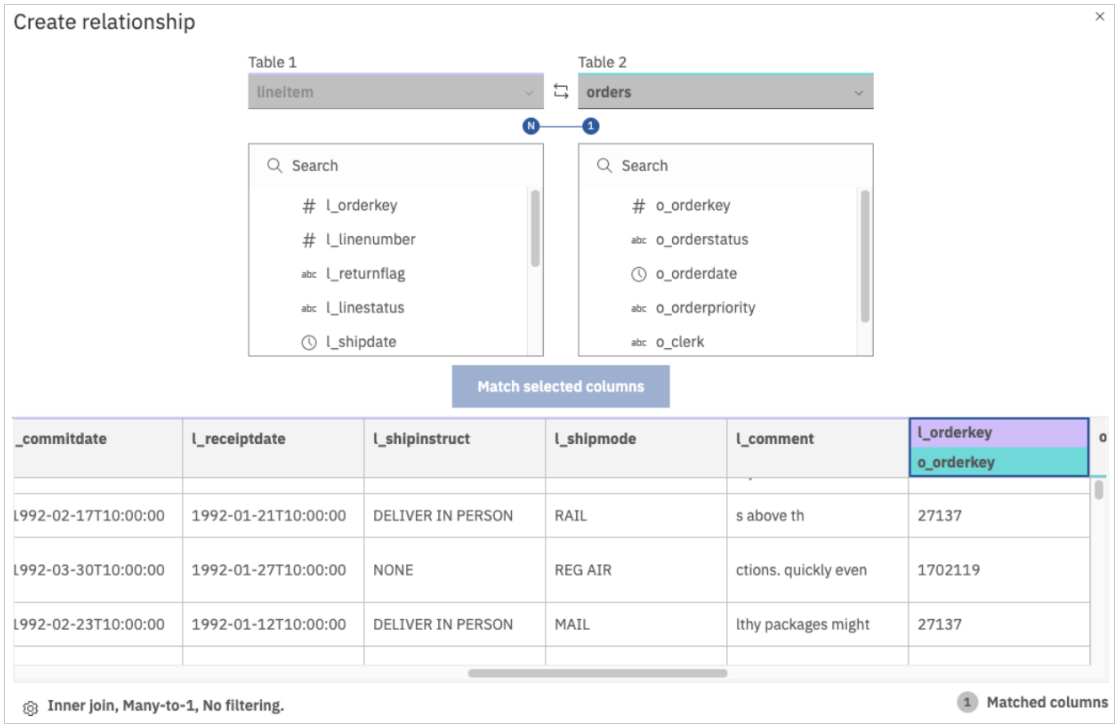
v. 在Select tables页面，勾选目标表的名称并单击OK。



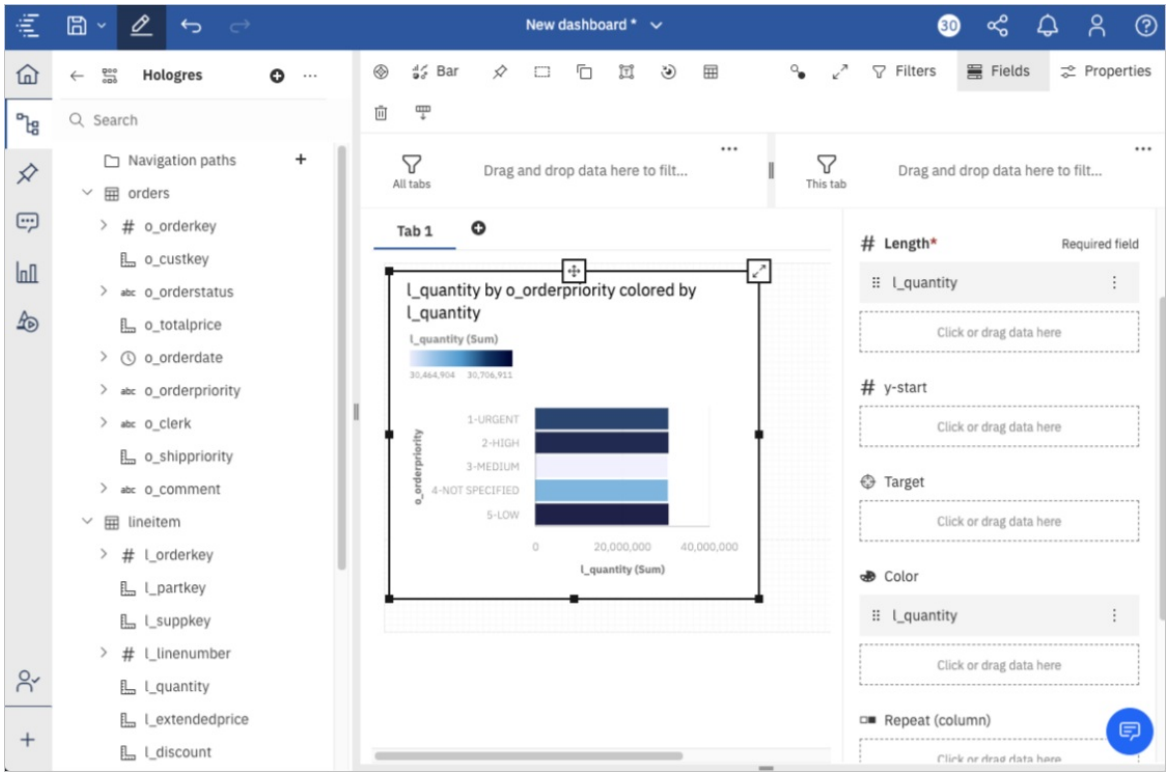
vi. 在建模页面，单击Relationships进行表关系设置。



vii. 在Create relationship页面，完成表关系设置后，单击Save保存该模型。



4. 模型创建完成后，您可以使用该模型进行可视化分析。更多关于IBM Cognos Analytics的操作指导，请参见IBM Cognos Analytics官方文档。



7.12. Metabase

Metabase是一个开源的商业智能工具，您可以通过它理解数据、分析数据，进行数据查询并获取格式化结果（图形化视图），以数据驱动决策。Hologres兼容PostgreSQL，支持直接连接Metabase进行数据分析。本文为您介绍Metabase如何连接Hologres并可视化分析数据。

注意事项

当Metabase成功连接Hologres读取数据库元数据时，会自动过滤没有任何表的Schema，该Schema不会展示在数据列表中。

操作步骤

1. 安装Metabase

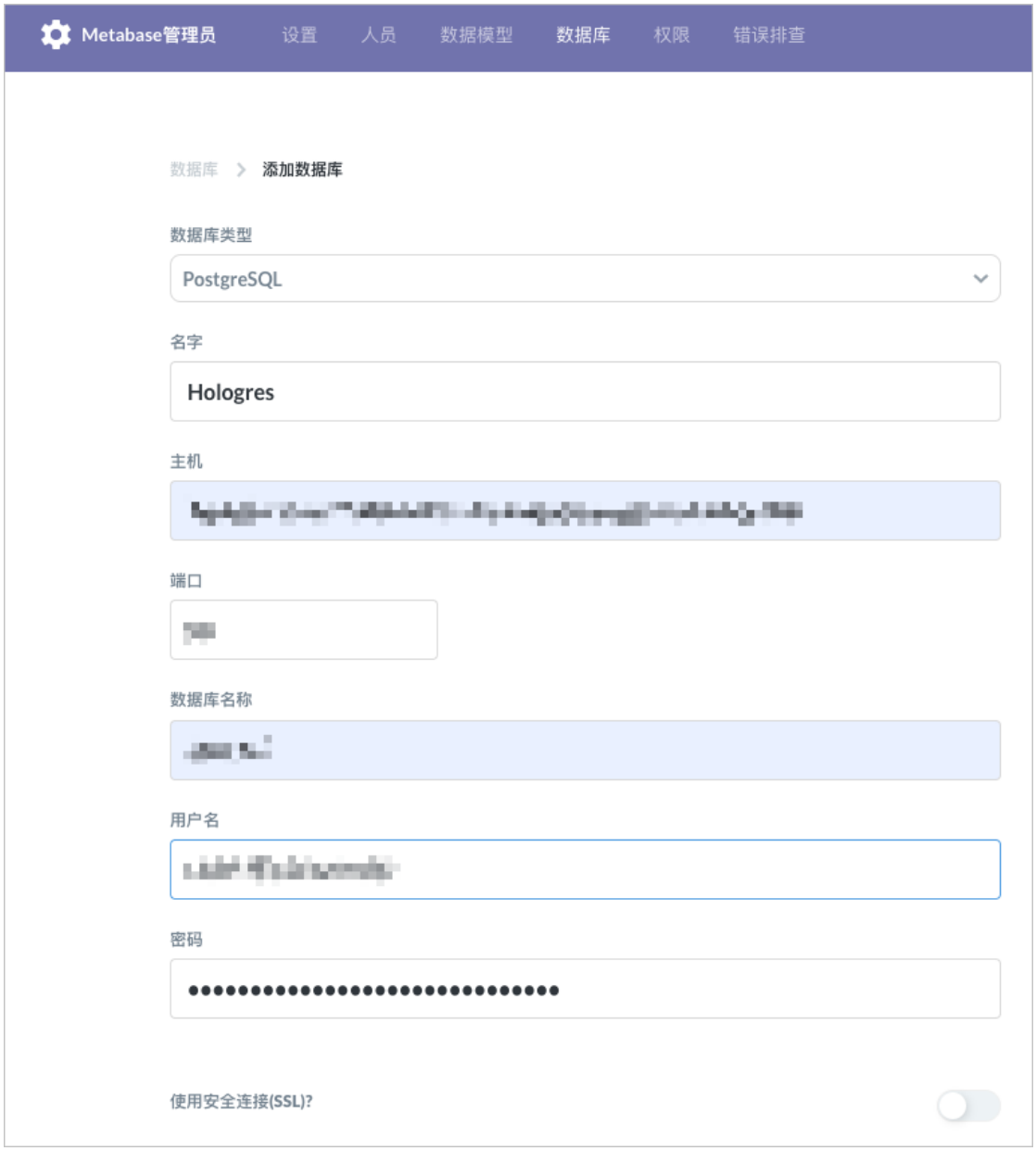
   安装Metabase，详情请参见Metabase官方文档。

2. 连接Hologres

   i. 登录Metabase，在顶部导航栏的右上角单击图标，在下拉菜单中单击**管理员**。

   ii. 单击**Metabase管理员**页面顶部的**数据库**。在数据库页面右上角，单击**添加数据库**，增加Hologres的连接。

   iii. 选择数据库类型并配置连接数据库参数后，单击**保存**。



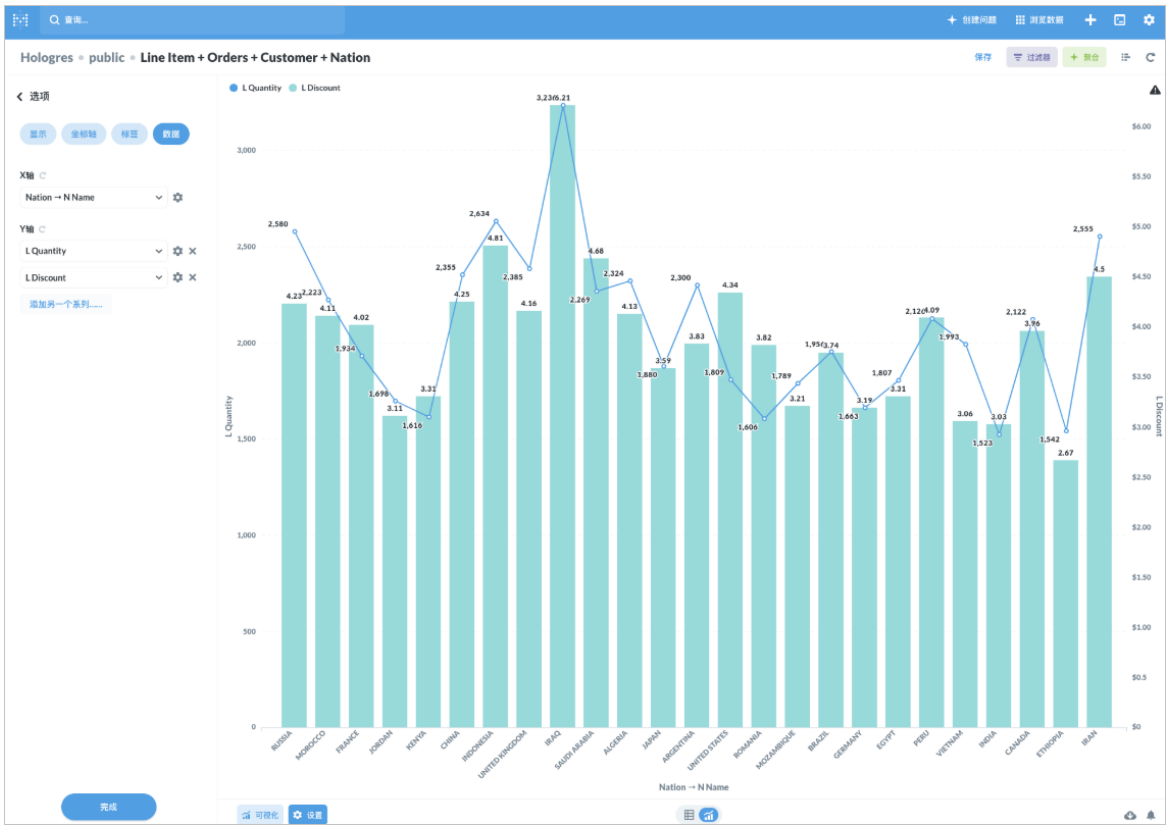
具体参数配置如下所示：

参数	描述
数据库类型	在下拉列表中选择**PostgreSQL**。
主机	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
端口	Hologres的实例端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。
数据库名称	Hologres的数据库名称。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。

参数	描述
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
更多参数	其余更多参数说明请参见Metabase官方文档。

3. 数据分析

   完成配置之后，您即可开始连接Hologres进行数据分析，更多关于数据分析的操作请参见Metabase官方文档。



说明

7.13. Power BI

本文为您介绍Power BI如何连接Hologres并可视化分析数据。

背景信息

Power BI是微软知名的BI软件。Hologres兼容PostgreSQL，支持直接连接Power BI Desktop进行数据分析。若是业务有需要您也可以将Power BI Desktop的报表发布至Power BI Service或者Power BI Report Server查看分析结果。

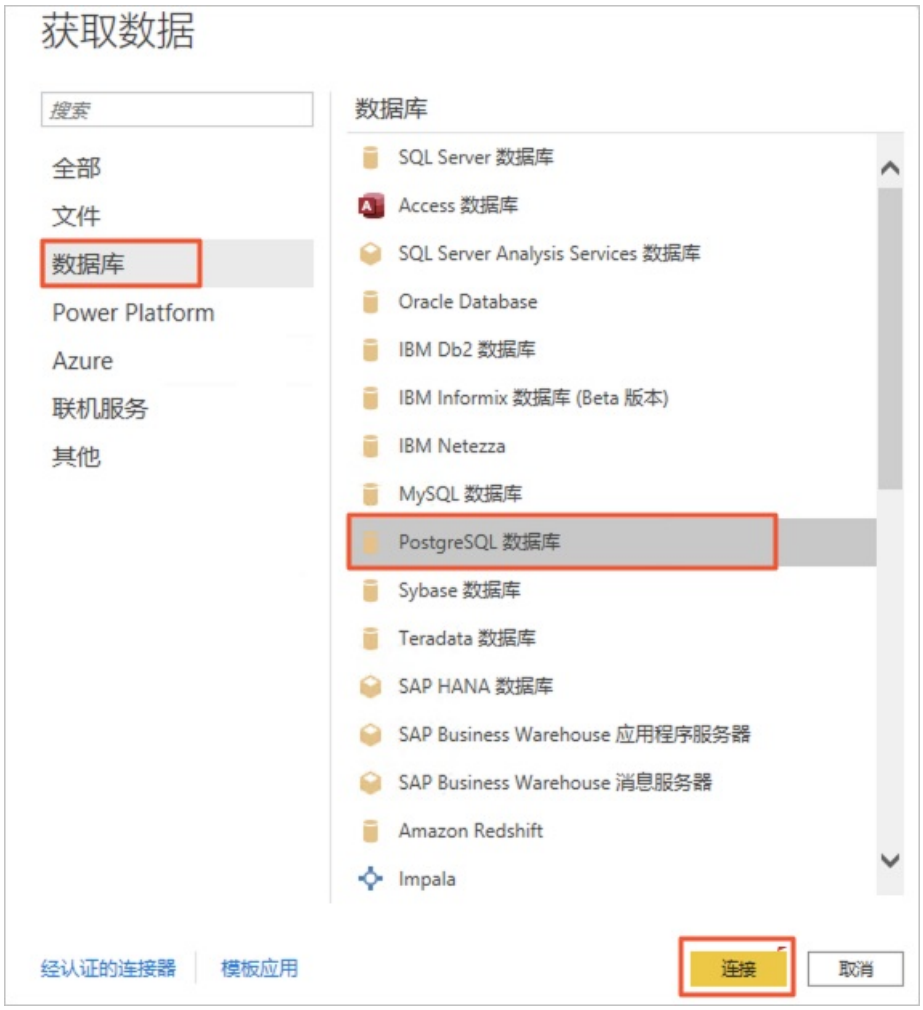
注意事项

Power BI默认不同步Hologres中的外部表，您需要在连接Hologres后，在数据库中执行如下SQL命令，才可以在Power BI中同步Hologres的外部表信息。

```
CREATE EXTENSION foreign_table_exposer;
```

Power BI Desktop连接Hologres

1. 安装Power BI Desktop
   安装Power BI Desktop，详情请参见Power BI Desktop官方文档。
2. 连接 Hologres
   i. 打开Power BI Desktop，在页面上方单击**获取数据 > 更多**。
   ii. 在**获取数据**页面的**数据库**类别中，选择**PostgreSQL 数据库**。



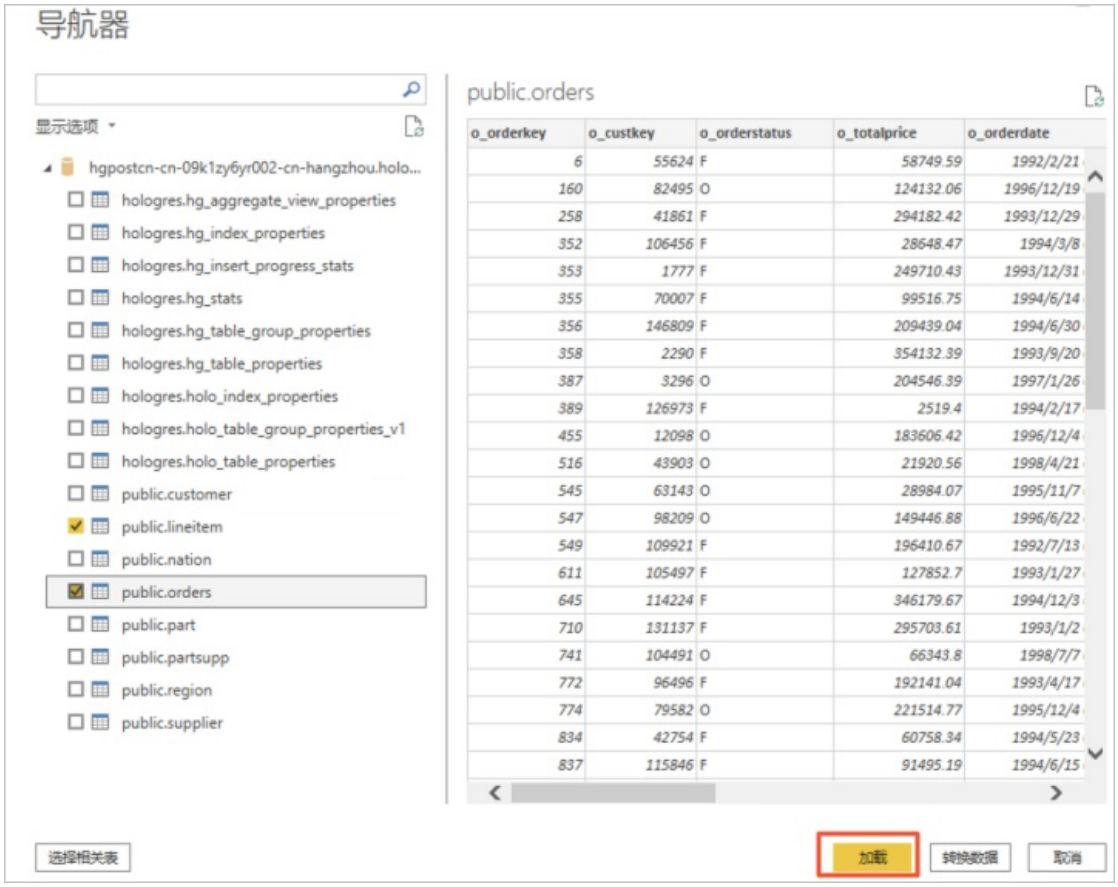
iii. 单击**连接**，在弹框中配置实例连接信息。

参数	描述
服务器	Hologres实例的网络地址。进入Hologres管理控制台的实例详情页，从**实例配置**获取网络地址。
数据库	Hologres创建的数据库名称。
数据连接模式	数据连接模式选择**DirectQuery**.
高级选项	此处示例无需配置，保持默认值即可。您也可以根据实际业务需求进行精细化配置。

iv. 单击**确定**，配置用户名和密码信息。

参数	描述
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

v. 单击**连接**，会在**导航器**页面展示数据库中的表信息，您可以根据需要选择要加载到Power BI Desktop的表数据进行数据可视化分析。



vi. 单击下方的**加载**，将数据导入Power BI Desktop当中。
将数据导入Power BI Desktop之后您可以对数据进行查询操作并创建报表或仪表盘进行分析。更多关于Power BI Desktop对数据操作和分析的介绍，请参见Power BI官方文档。

发布至Power BI Service

您可以将数据发布至Power BI Service查看分析结果。Power BI Service通过Power BI Gateway和本地服务器进行通信。在将Power BI Desktop的报表发布至Power BI Service之后您需要安装并配置Power BI Gateway。

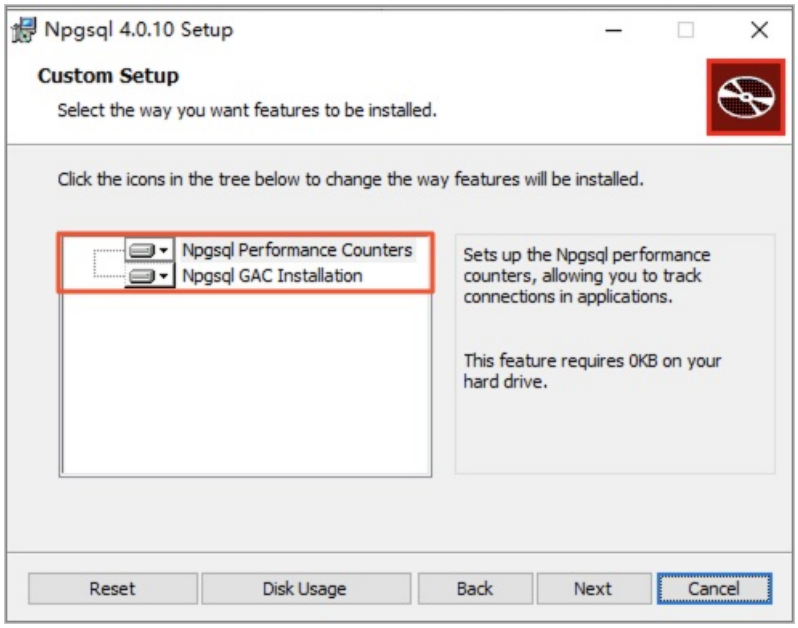
1. Power BI Desktop数据报表发布
   i. 打开Power BI Desktop的数据报表，在页面上方单击**发布**。
   ii. 在**发布到Power BI**弹框页面，选择需要发布的工作区。

iii. 单击**选择**，Power BI Desktop会将报表发布到Power BI Service。发布成功后，您可以单击页面中的链接访问Power BI Service。

2. 安装Power BI Gateway

   Power BI Service通过Power BI Gateway和本地服务器进行通信。安装Power BI Gateway，详情请参见Power BI官方文档。在安装时您需要注意如下事项：

   - Power BI Gateway可以不用和Power BI Desktop 安装在同一台机器上。
   - 需要保证运行Power BI Gateway的服务器网络通畅且能访问 Hologres 数据源。
   - 必须安装Npgsql 4.0.10，请单击下载Npgsql 4.0.10安装包。安装时请务必安装Npgsql GAC Installation，并且在安装完毕后务必重启 Power BI Gateway。



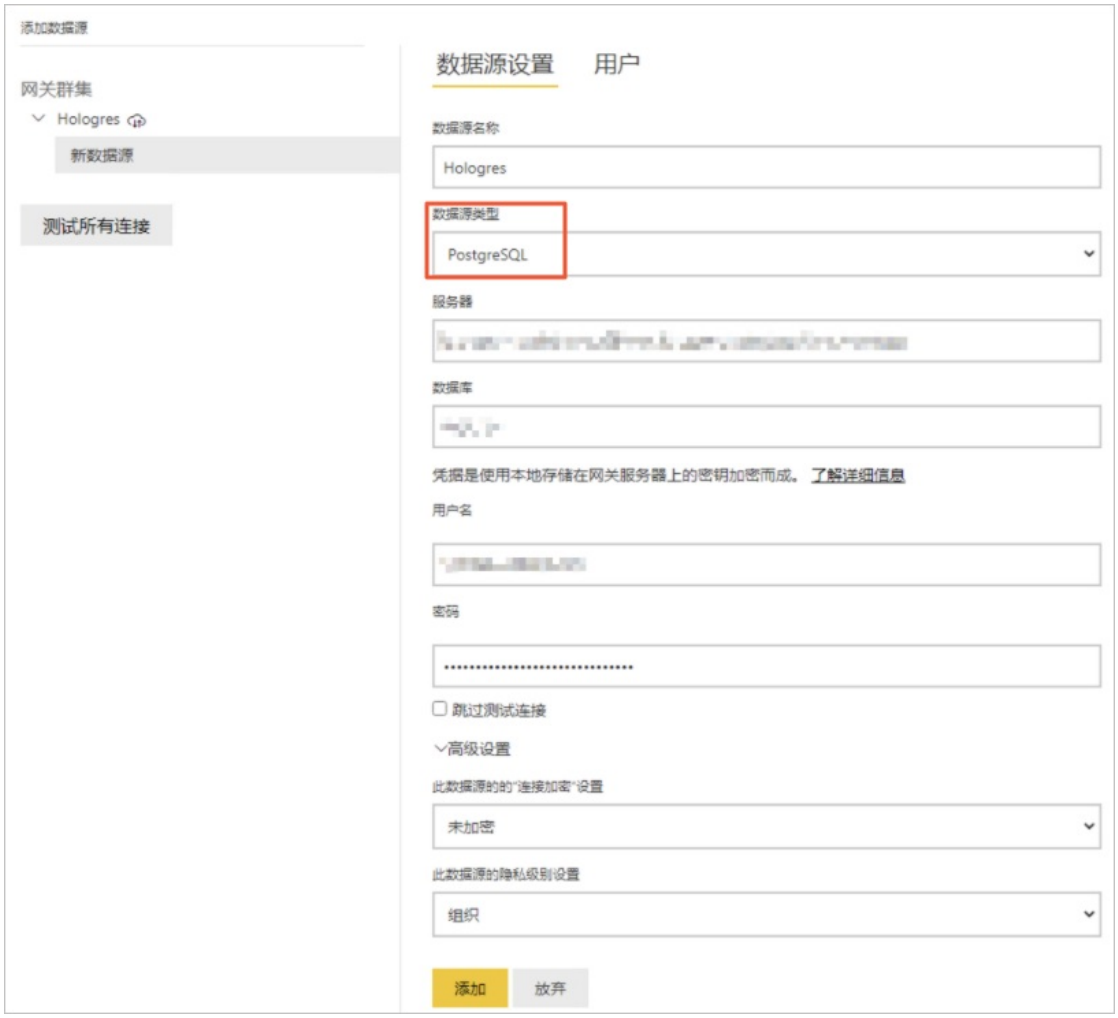
3. 配置Power BI Gateway

   Power BI Gateway安装完成后，在运行网关的服务器上，启动Power BI Gateway后您可以通过双击Power BI Gateway 图标进行配置。具体操作步骤，请参见微软Gateway官方文档。

4. 配置Power BI Service数据源

   i. 登录Power BI Service，在页面右上角，单击图标，在列表中选择**管理网关**。

   ii. 选择目标网关，单击**添加数据源**，进行参数配置。



数据源配置信息需与Power BI Desktop的连接信息保持一致。具体参数说明如下：

参数	说明
数据源名称	为新数据源命名。
数据源类型	在下拉框中选择**PostgreSQL数据库**。
服务器	Hologres实例的网络地址。进入Hologres管理控制台的实例详情页，从**实例配置**获取网络地址。
数据库	Hologres创建的数据库名称。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
高级设置	在高级设置中，您需要配置如下内容： ■ 此数据源的“连接加密”设置选择为**未加密**。 ■ 此数据源的隐私级别设置可以为默认值。

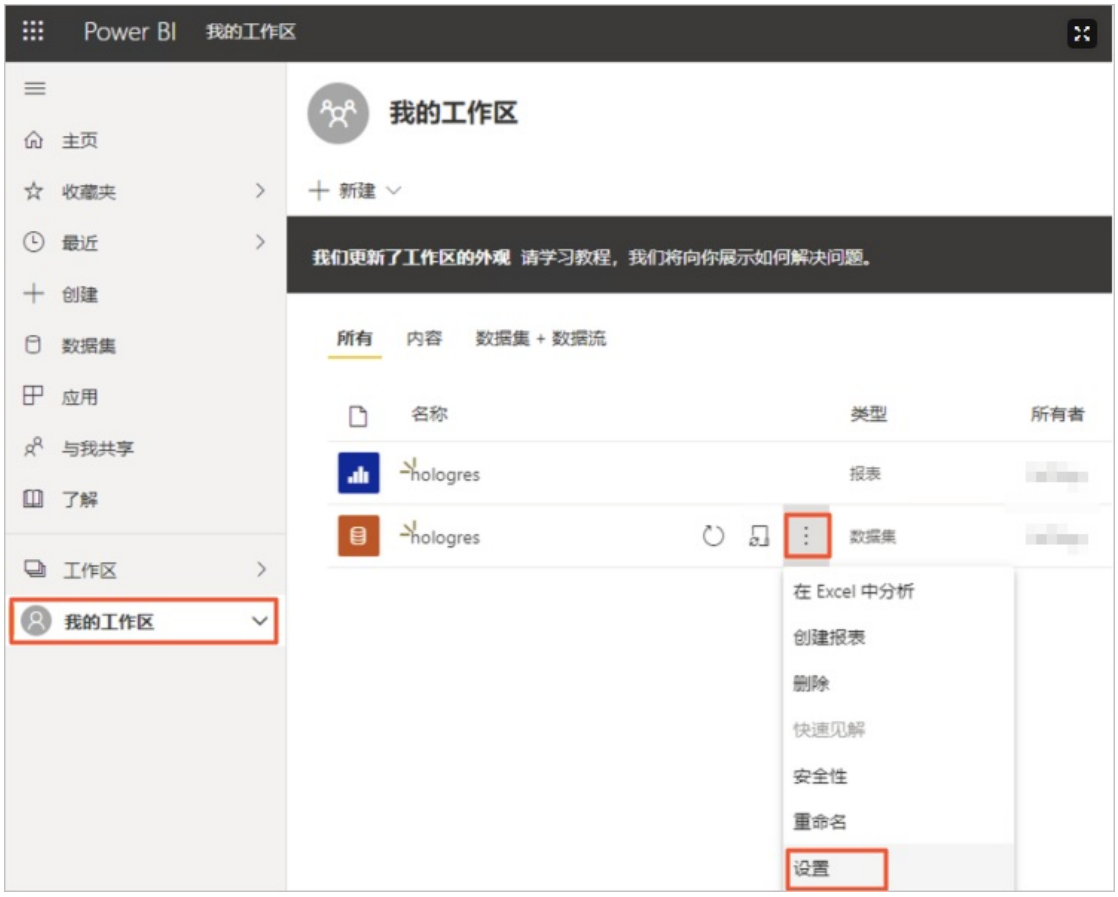
iii. 单击**添加**，系统会开始测试联通性，完成测试之后会提示您数据源连接成功。

5. 配置报表网关

将数据报表发布到Power BI Service后需要配置报表网关才能查看您的报表并进行交互式分析。

i. 登录Power BI Service，在左侧导航栏单击进入**我的工作区**页面。

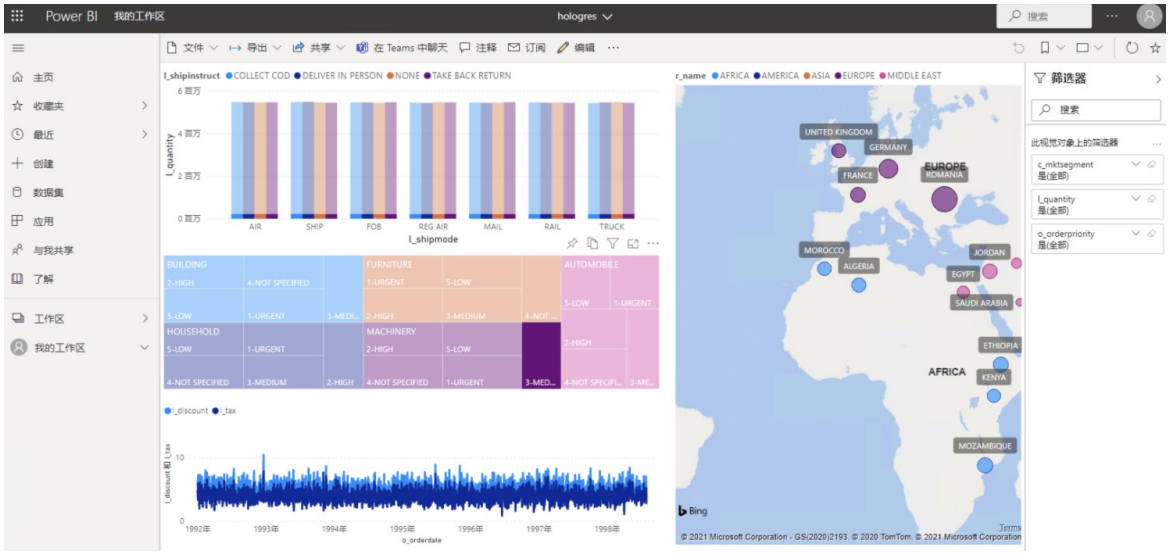
ii. 单击目标数据集后的⋮图标，选择**设置**。



iii. 在设置页面，单击**网关连接**并打开**使用网关连接**开关。选择对应的网关和映射的数据源。

iv. 单击**应用**，系统提示网关已经更新，则表示网关设置成功。

完成上述设置，您可以在Power BI Service上查看您的报表并进行交互式分析。



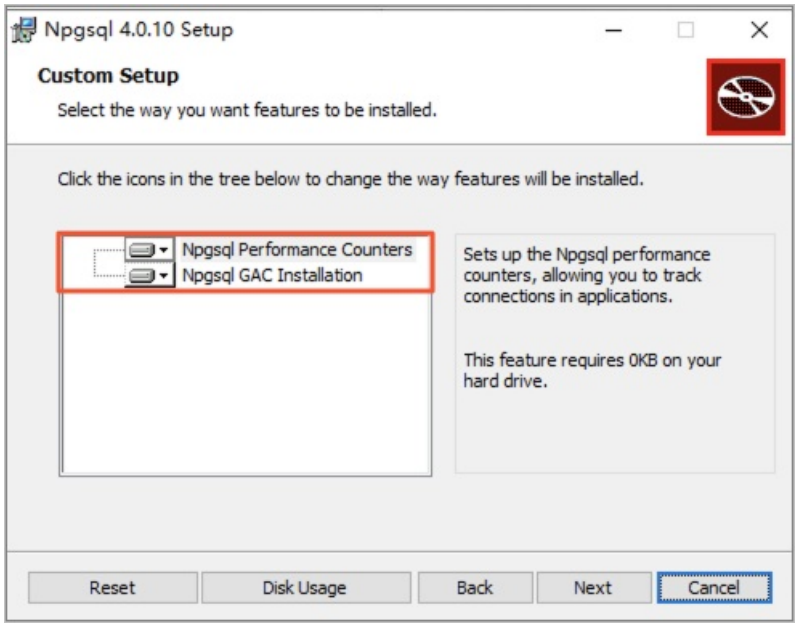
发布至Power BI Report Server

您可以将数据发布至Power BI Report Server查看分析结果。

1. 安装Power BI Report Server

   安装Power BI Report Server，详情请参见Power BI Report Server官方文档。在安装时您需要注意如下事项：

   - 需要保证运行Power BI Report Server的服务器网络通畅且能访问 Hologres 数据源。
   - 必须安装Npgsql 4.0.10，请单击下载Npgsql 4.0.10安装包。安装时请务必安装Npgsql GAC Installation，并且在安装完毕后务必重启 Power BI 报表服务器。



2. Power BI Desktop数据发布

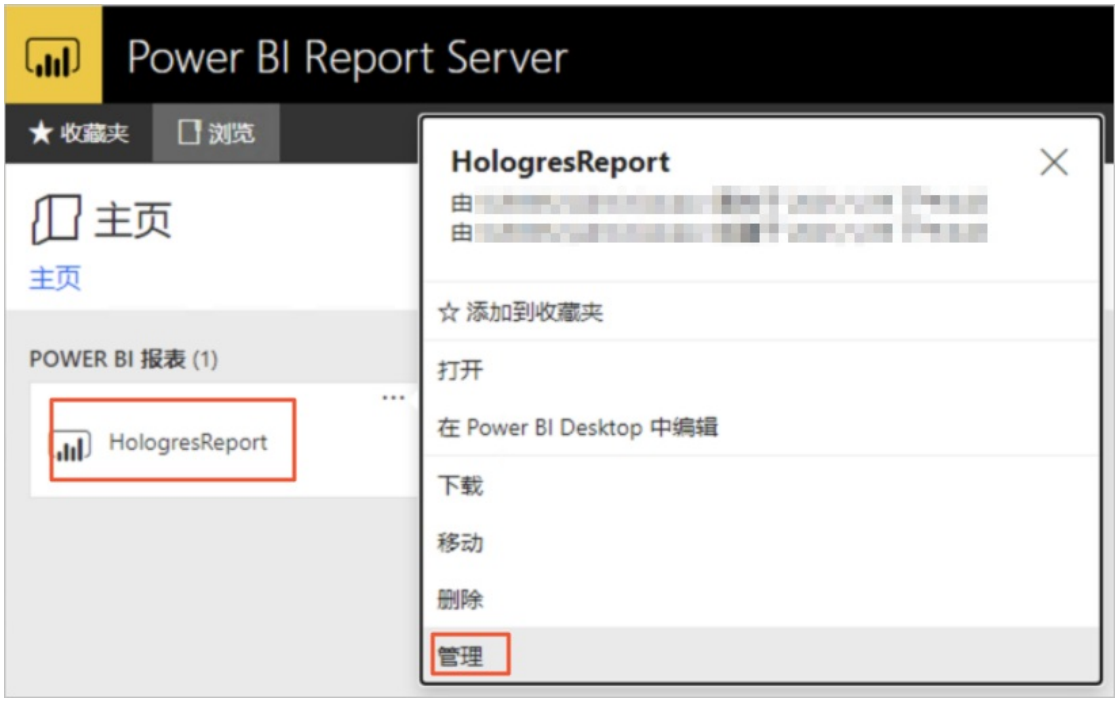
i. 打开Power BI Desktop的数据报表，在页面上方选择**文件 > 另存为 > Power BI报表服务器**。

ii. 在弹框页面，输入服务器地址后单击**确定**。

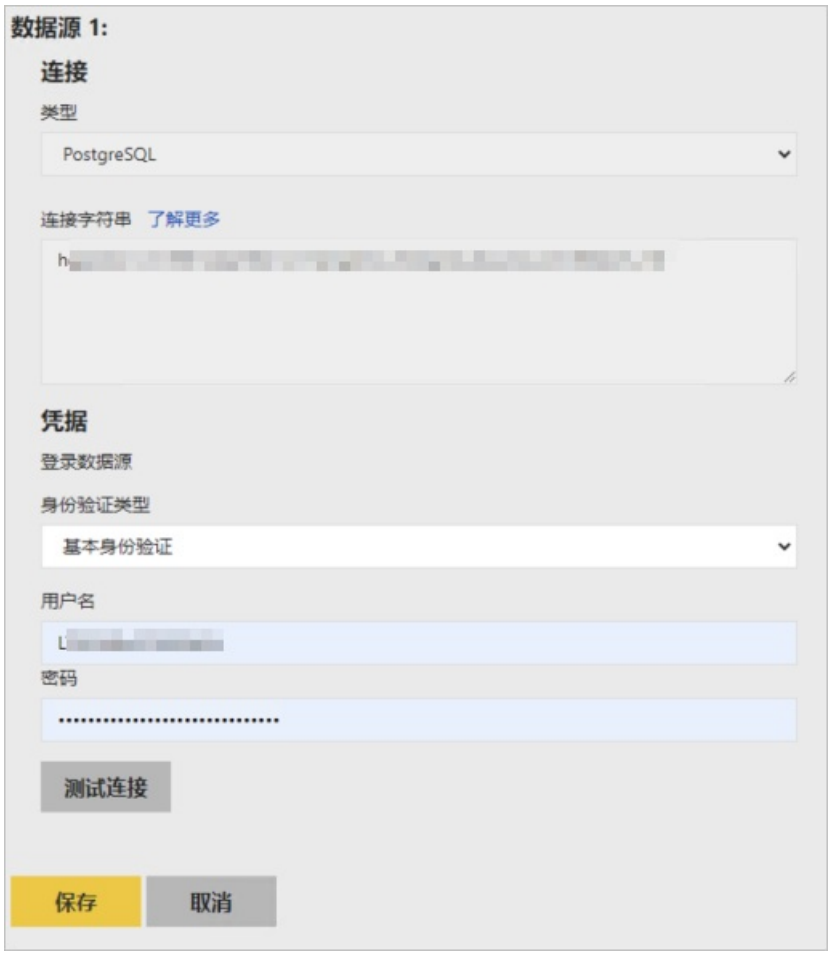
iii. 选择报表的存储位置，单击**确定**，Power BI Desktop会将报表发布到Power BI Report Server。发布成功后，您可以单击页面中的链接访问Power BI Report Server。

3. 配置Power Report Server数据源

i. 登录Power Report Server，单击目标数据源报表右侧的 ··· 图标，在列表中选择**管理**。



ii. 在数据源页面进行参数配置。

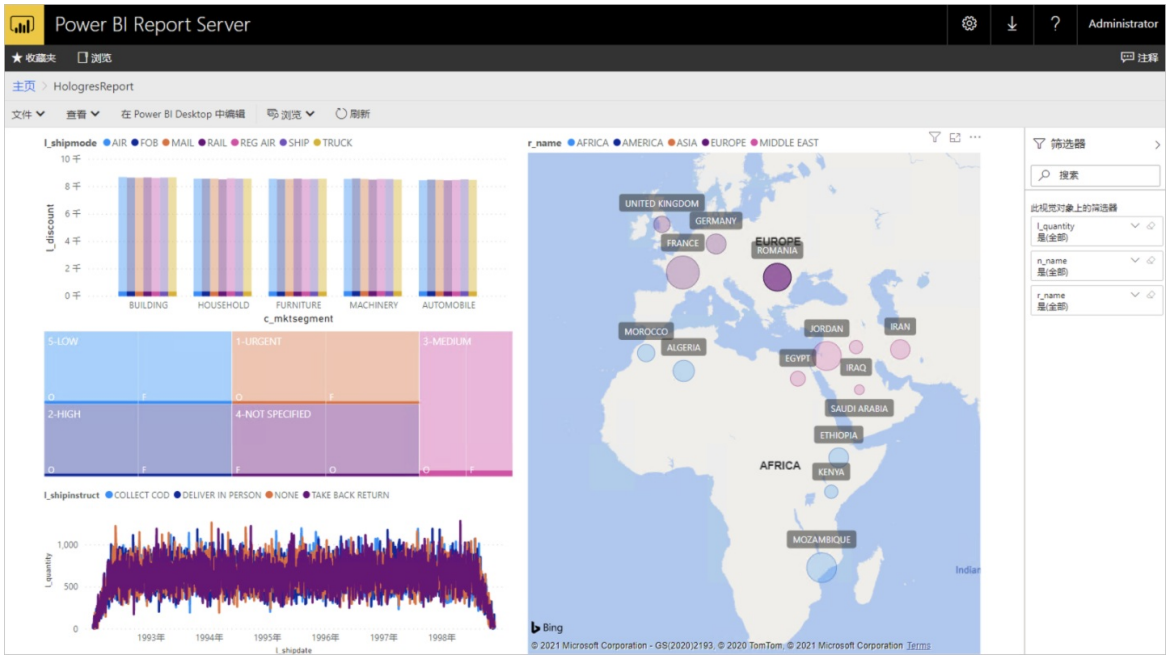


数据源配置信息需与Power BI Desktop的连接信息保持一致。具体参数说明如下：

参数	说明
类型	在下拉框中选择**PostgreSQL**。
连接字符串	自动生成，无需填写。
身份验证类型	在下拉框中选择**基本身份验证**。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

iii. 单击**测试连接**，系统会开始测试联通性，完成测试之后会提示您数据源连接成功。

完成上述设置，您可以在Power Report Server上查看您的报表并进行交互式分析。



7.14. Qlik

Qlik是经典的商业智能分析软件，其家族拥有Qlik Sense等多款BI软件。它使您能够快速开发和交付交互式指导分析应用程序和仪表板。本文为您介绍Qlik Sense Desktop如何连接 Hologres并可视化分析数据。

使用限制

Qilk暂不支持可视化显示Hologres的外部表，但是您可以在**数据加载编辑器**中通过SQL语句查询外部表并可视化分析。

Qlik Sense Desktop连接Hologres

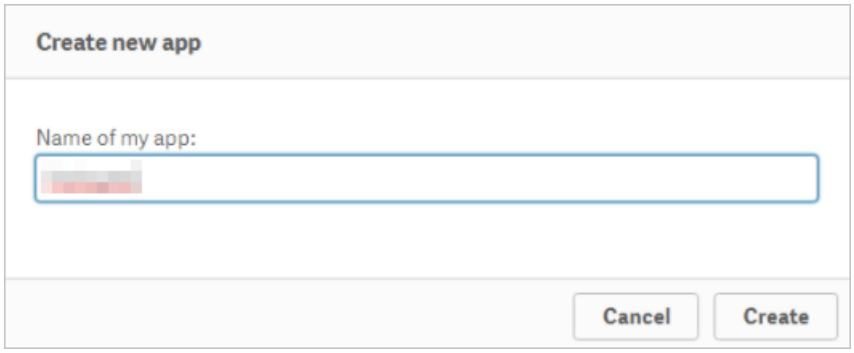
1. 安装Qlik Sense Desktop

   安装Qlik Sense Desktop，详情请参见Qlik官方文档。

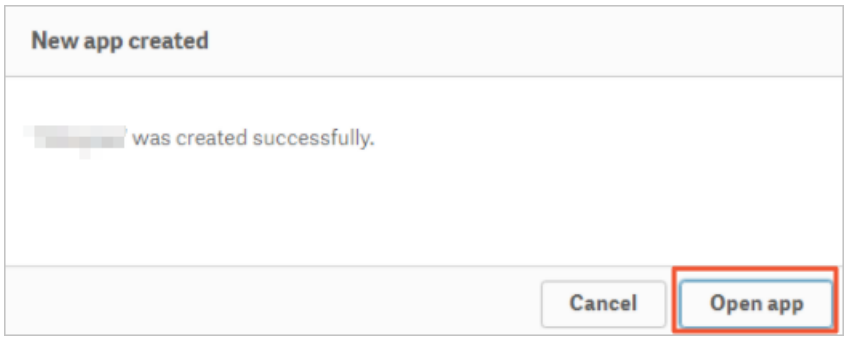
2. 连接Hologres

   i. 打开Qlik Sense Desktop，在页面右上方单击**创建新应用程序**。

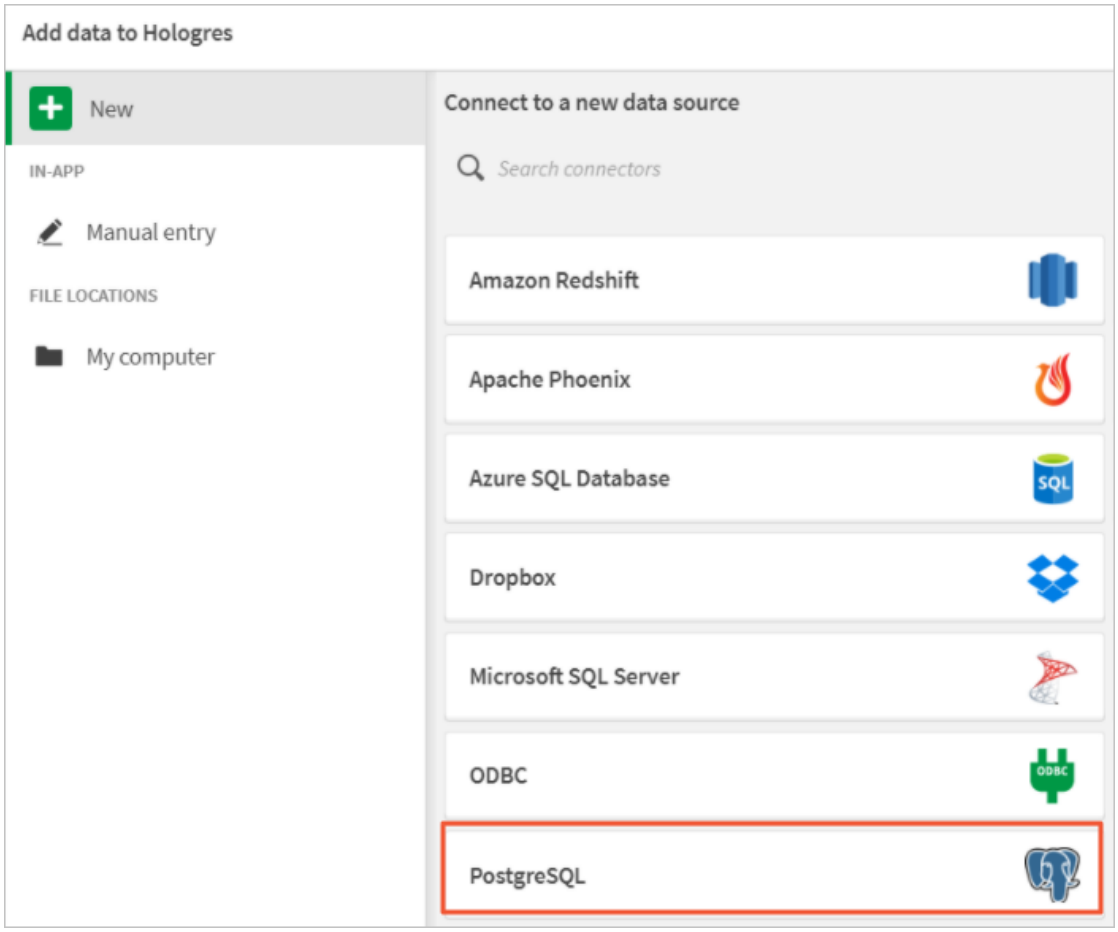
ii. 在**创建新应用程序**弹框，为应用程序命名后，单击**创建**。



iii. 在弹框中单击**打开应用程序**，在应用程序对应页面添加数据，单击**从文件和其他源添加数据**。

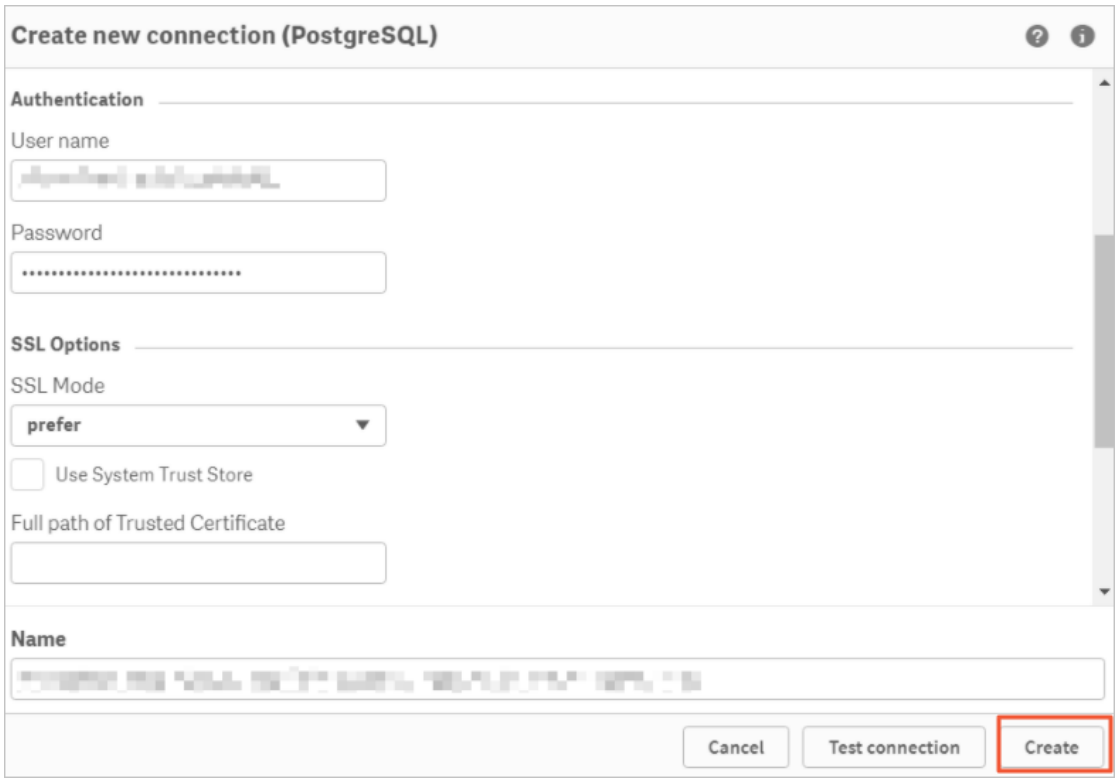


iv. 在弹框中选择PostgreSQL创建新的连接，配置参数信息。

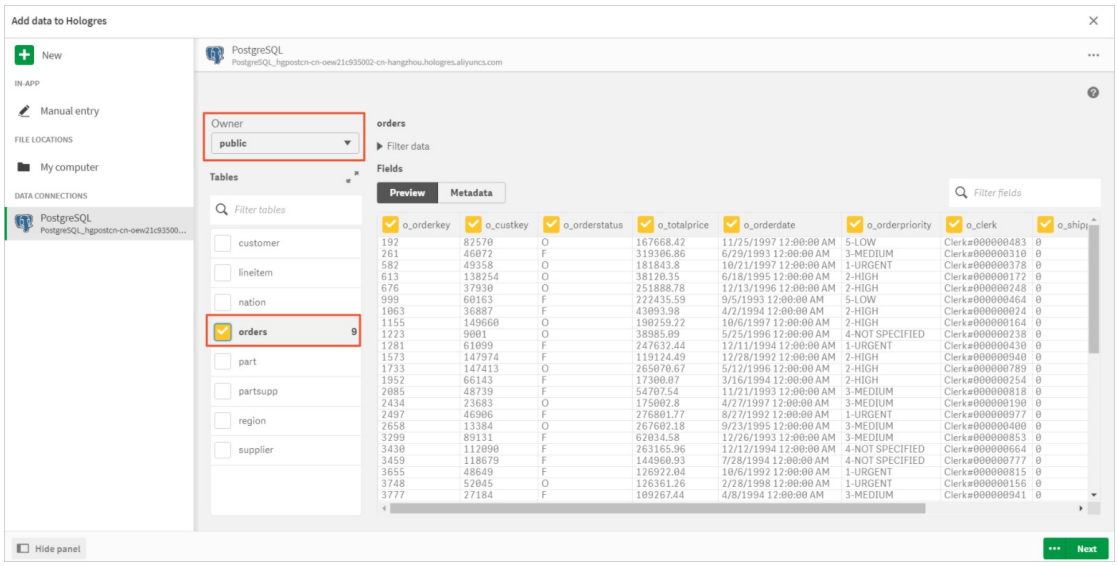


参数	描述
Host name	Hologres实例的公共网络地址。进入Hologres管理控制台的实例详情页，从**实例配置**获取网络地址。
Port	Hologres的实例端口。进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
Database	Hologres创建的数据库名称。
User Name	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

v. 单击**测试连接**，如果提示 `Connection succeeded` ，则表示连接成功。您可以单击弹框右下角的**创建**，保存新的连接信息。



vi. 配置PostgreSQL数据连接。



- 选择目标Owner（即Hologres中的 schema），此处您可以选择public。
- 在Tables区域选择需要分析的表。

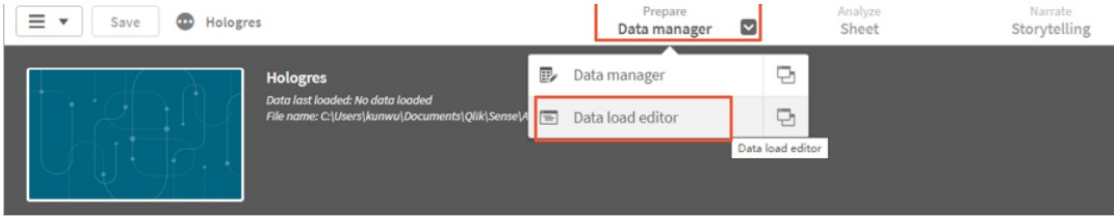
vii. 单击下方的**添加数据**，添加数据执行完毕后，Qlik Sense会将数据从Hologres导入Qlik Sense，您可以在Qilk查看Hologres中的数据。

> **说明** 使用该模式Qlik Sense会将数据全部加载到Qlik Sense的引擎中，并非根据页面操作实时发送查询到数据库。

3. 配置Direct Query模式

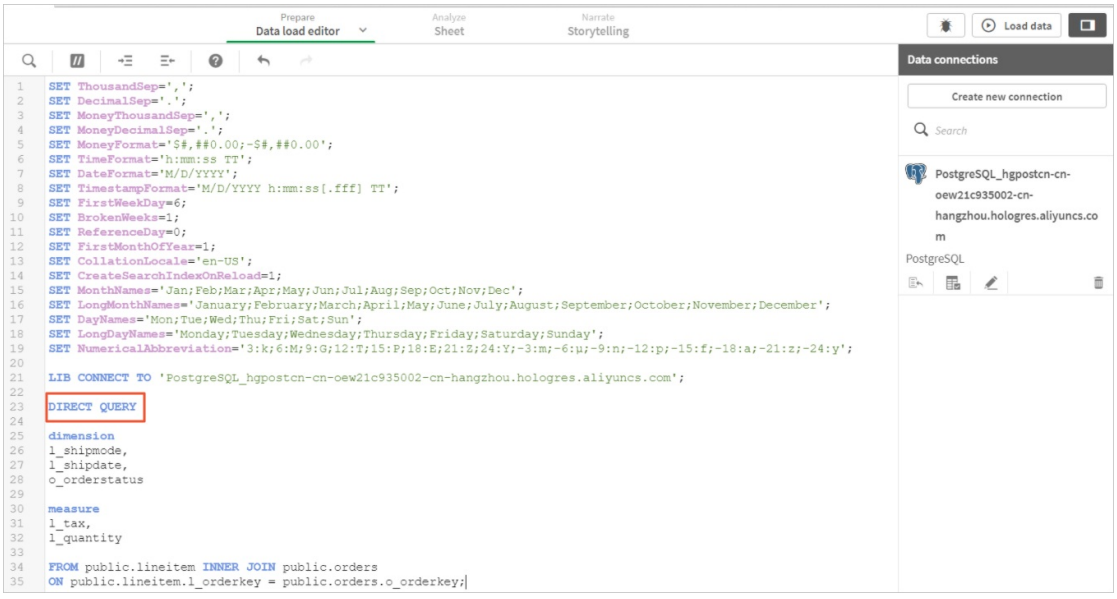
在日常生产场景中，数据库会包含PB级数据，建议您在Qlik Sense中使用Direct Query模式，而不用将数据导入Qlik Sense，关于Direct Query模式的详细解释，请参见Qlik官方文档。

i. 打开之前创建的应用，在页面上方选择**数据管理器 > 数据加载编辑器**。



ii. 在页面右侧显示数据源连接信息，单击右下方的图标，将连接信息插入到编辑器中。

iii. 在脚本编辑器中查询脚本的最前方，输入 `Direct Query` ，即可启用 Direct Query 模式。样例数据图示和脚本如下所示：



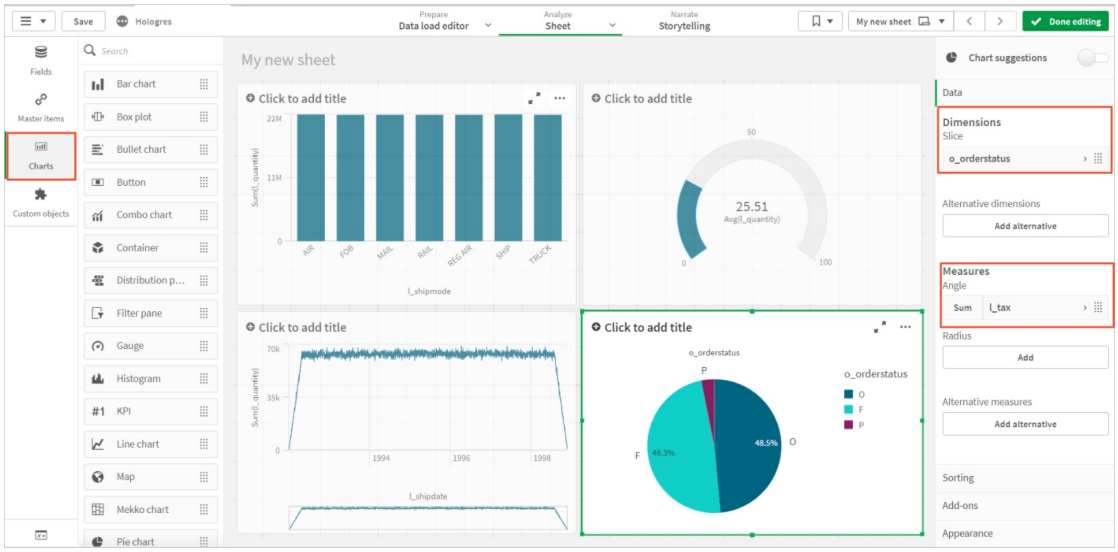
```
SET ThousandSep=',';
SET DecimalSep='.';
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='¥#,##0.00;-¥#,##0.00';
SET TimeFormat='TTh:mm:ss';
SET DateFormat='YYYY/M/D';
SET TimestampFormat='YYYY/M/D TTh:mm:ss[.fff]';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET FirstMonthOfYear=1;
SET CollationLocale='zh-CN';
SET CreateSearchIndexOnReload=1;
SET MonthNames='1月;2月;3月;4月;5月;6月;7月;8月;9月;10月;11月;12月';
SET LongMonthNames='一月;二月;三月;四月;五月;六月;七月;八月;九月;十月;十一月;十二月';
SET DayNames='周一;周二;周三;周四;周五;周六;周日';
SET LongDayNames='星期一;星期二;星期三;星期四;星期五;星期六;星期日';
SET NumericalAbbreviation='3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:
p;-15:f;-18:a;-21:z;-24:y';
LIB CONNECT TO 'PostgreSQL_hgpostcn-cn-st21y8nlm007-cn-hangzhou.hologres.aliyuncs.c
om';
--输入 Direct Query，启用 Direct Query 模式
DIRECT QUERY
dimension
l_shipmode,
l_shipdate,
o_orderstatus
measure
l_tax,
l_quantity
FROM public.lineitem INNER JOIN public.orders
ON public.lineitem.l_orderkey = public.orders.o_orderkey;
```

iv. 单击窗口右上方的**加载数据**，Qlik Sense将发送即时查询。

v. 在页面上方选择**工作表 > 编辑工作表**，开始创建可视化。

vi. 在编辑工作表页面，您可以在页面左侧单击图表，选择需要的图表类型创建可视化。

> **说明**
>
> - 直接拖动字段到画布，Qlik Sense无法生成Direct Query的查询，因此建议您单击图表创建可视化。
> - 鉴于Qlik Sense的处理逻辑，建议您先添加度量项，后添加维度项，以提高响应速度。



更多关于Qlik Sense对数据操作和分析的介绍，请参见Qlik官方文档。

使用自定义ODBC连接Hologres

1. 安装PostgreSQL ODBC驱动

   您可以从PostgreSQL官方下载PostgreSQL ODBC驱动，请务必安装**psqlodbc_11_01_0000**以上的版本。

2. 配置DSN

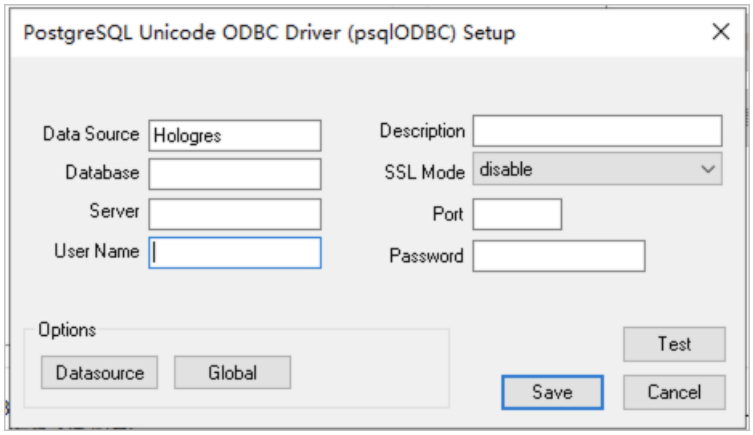
   i. 打开计算机的**控制面板**，在页面右上方将查看方式切换为**大图标**，选择**管理工具**。

   ii. 双击打开**ODBC 数据源（64位）**管理工具，单击进入**系统DSN**页签。

   > **说明** **用户DSN**只有特定的用户可以调用，而**系统DSN**对该系统的所有登录用户可用。如果用户需要在Web BI Server通过ODBC访问Hologres，应使用系统DSN。

   iii. 单击**添加**，在**创建新数据源**弹框选择 PostgreSQL Unicode(x64)。

iv. 单击**完成**，在弹框内完成如下参数配置。

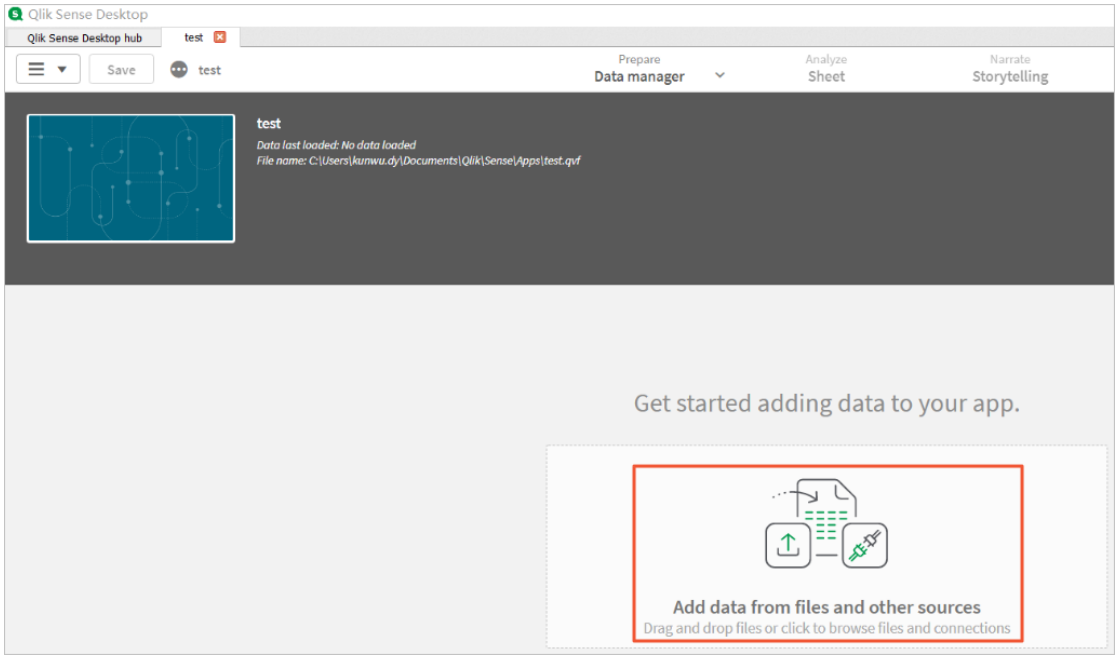


参数	描述
Database	Hologres创建的数据库名称。
Server	Hologres实例的公共网络地址。进入Hologres管理控制台的实例详情页，从**实例配置**获取公共网络地址。
Port	Hologres的实例端口。进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
User Name	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。

v. 单击**Test**，如果页面提示 `Connection successful` ，则表示连接成功。您可以单击弹框右下角的**Save**，保存该DSN。

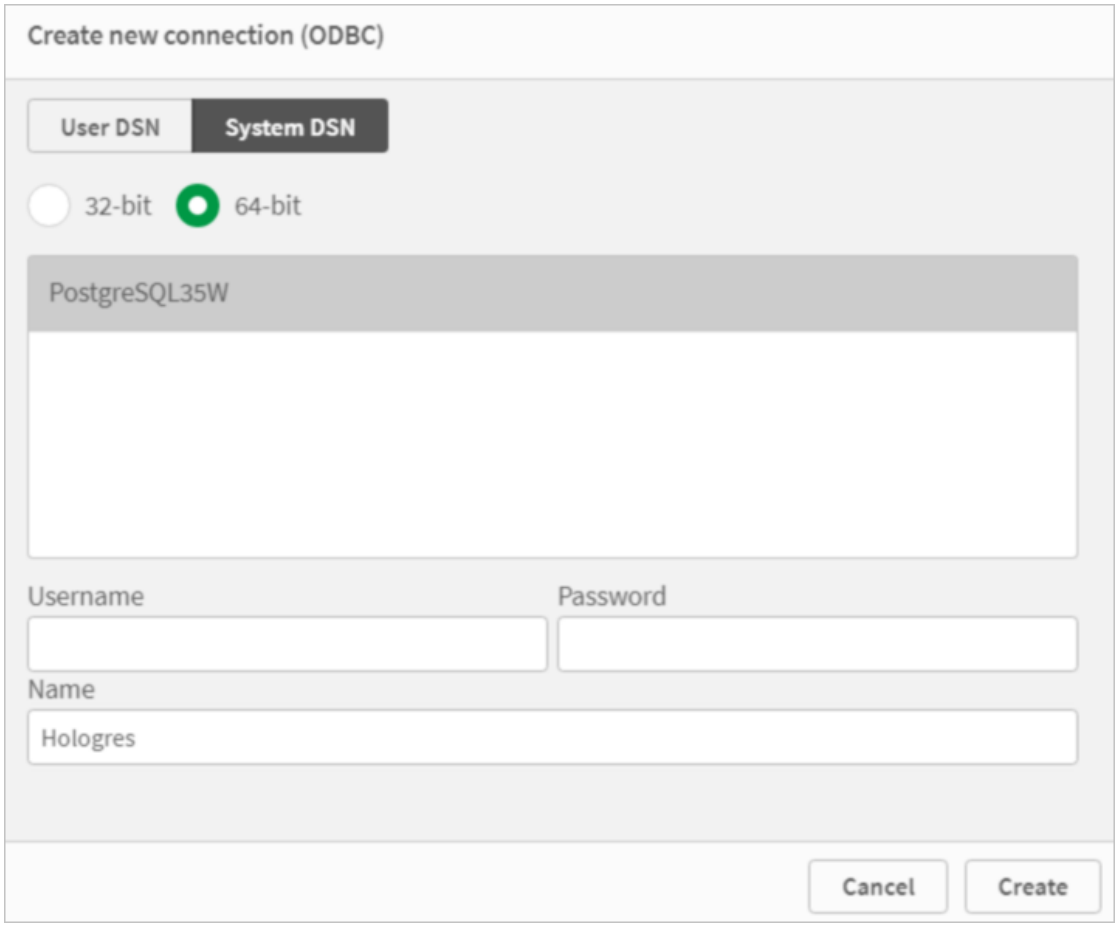
3. 连接Hologres

i. 打开Qlik Sense Desktop，在页面选择一个应用程序并在对应页面添加数据，单击**从文件和其他源添加数据**。

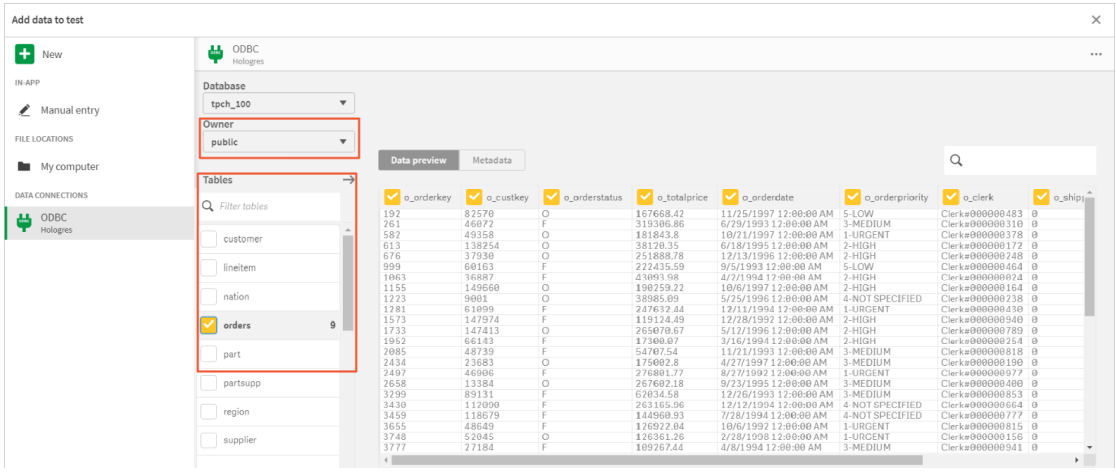


ii. 在弹框中选择ODBC创建新的连接。

iii. 从列表中选择此前创建的DSN，并填写连接的名称，如Hologres。单击**创建**，以保存新的连接信息。



iv. 配置ODBC数据连接。



- 选择目标Owner（即Hologres中的Schema），此处您可以选择public。使用该连接方式即可读取外部表的Schema。
- 在Tables区域选择需要分析的表。

v. 数据加载完毕之后，数据已导入至Qlik Sense，您可以开始创建报告。

使用该模式Qlik Sense会将数据全部加载Qlik Sense 的引擎中，并非根据页面操作实时发送查询到数据库。

7.15. Quick BI

Quick BI为您提供海量数据的实时在线分析服务以及丰富的可视化效果，您可以通过拖拽式操作或SQL语句方式，轻松地完成数据分析、业务数据探查及报表制作。本文为您介绍如何使用交互式分析Hologres连接Quick BI，并进行可视化分析。

前提条件

- 开通Hologres，详情请参见购买Hologres。
- 开通Quick BI，详情请参见Quick BI购买、升级、降级、续费、欠费。

背景信息

Hologres与Quick BI高效连通，支持通过Hologres数据源直接连接Quick BI，将Hologres高效查询的数据直接进行可视化分析。

> **说明** 当前Quick BI高级版和专业版支持Hologres数据源，其余版本请使用PostgreSQL数据源，更多关于PostgreSQL数据源的操作请参见云数据源PostgreSQL。

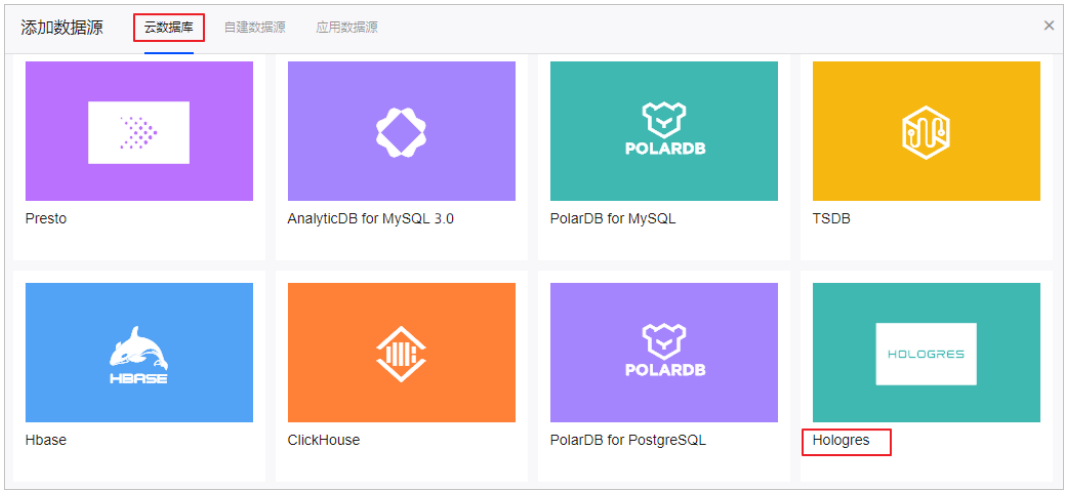
本文以高级版Qucik BI为例，为您演示连接Hologres并进行可视化分析的相关操作。

使用限制

- Quick BI当前仅支持使用公共网络和经典网络连接Hologres，不支持使用VPC网络连接Hologres。
- 您无需在Hologres中配置白名单即可连接至Quick BI。
- Hologres连接Quick BI时，您需要根据Quick BI的版本选择相应的数据源，推荐使用Hologres数据源。

使用Quick BI进行可视化分析

1. 登录Quick BI管理控制台。
2. 添加数据源。
   i. 在Quick BI管理控制台页面，单击顶部菜单栏的**工作空间**。
   ii. 在**工作空间**页面左侧导航栏，单击**数据源**。
   iii. 在**数据源**页面，单击右上角的**+新建数据源**。
   iv. 选择目标数据源并配置相应参数。
      a. 选择**云数据库 > Hologres**。



      b. 配置**添加Hologres数据源**的各项参数。

参数说明如下表所示。

参数	描述
显示名称	自定义的显示名称。

参数	描述
数据库地址	连接的Hologres实例的服务器地址。 您可以登录管理控制台，进入实例详情页，从**实例配置**页面获取。 **说明** Quick BI当前仅支持使用公共网络和经典网络连接Hologres，不支持使用VPC网络连接Hologres。
端口	连接的Hologres实例的端口。 您可以登录管理控制台，进入实例详情页，从**实例配置**页面获取。 **说明** Quick BI当前仅支持使用公共网络和经典网络连接Hologres，不支持使用VPC网络连接Hologres。
数据库	连接的Hologres实例的数据库名称。 您可以登录管理控制台，进入实例详情页，从**DB管理**页面获取。
Schema	默认为**public** Schema。您也可以使用新创建的Schema，在数据源配置了Schema后，列表中会展示当前Schema下所有的表。但是在使用即席SQL时，还需要在表名称前手动添加对应的Schema名（即shcema.table），才能正确索引到Schema下面的表。
用户名	具有当前数据库登录权限账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
密码	具有当前数据库登录权限账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。

参数配置完成后，您可以单击**连接测试**，测试Quick BI与Hologres的连接状态：

- 如果显示**数据源连通性正常**，则表示Quick BI与Hologres连接成功。
- 如果显示**数据源连通性异常，请检查参数是否正确**，则表示Quick BI与Hologres连接失败。请根据报错信息处理异常。

v. 单击**确定**，完成配置。

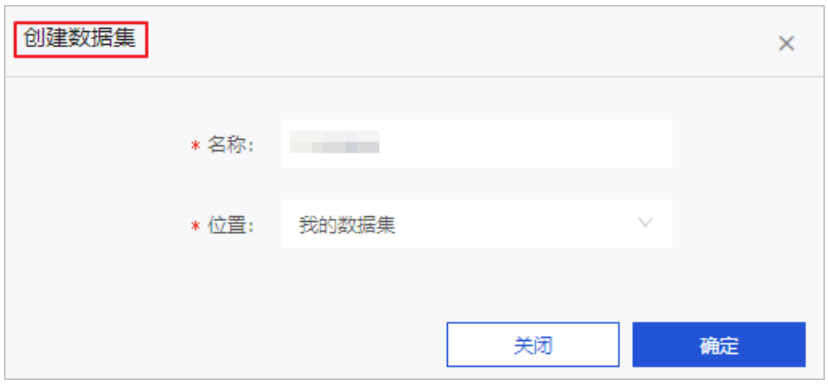
3. 可视化分析数据。

成功连接数据源后，在**数据源 > 我的数据源**页面，单击已创建的数据源，显示当前数据库中的所有表。

- 如果您希望使用界面化操作完成数据的可视化分析，则可以使用拖拽方式。
- 如果您希望使用SQL语句完成数据的可视化分析，则可以使用即席分析SQL方式。

两种方式的具体操作如下：

- 拖拽方式。
  a. 单击目标表**操作**列的图标，创建数据集。
  b. 配置**创建数据集**对话框的**名称**和**位置**参数。

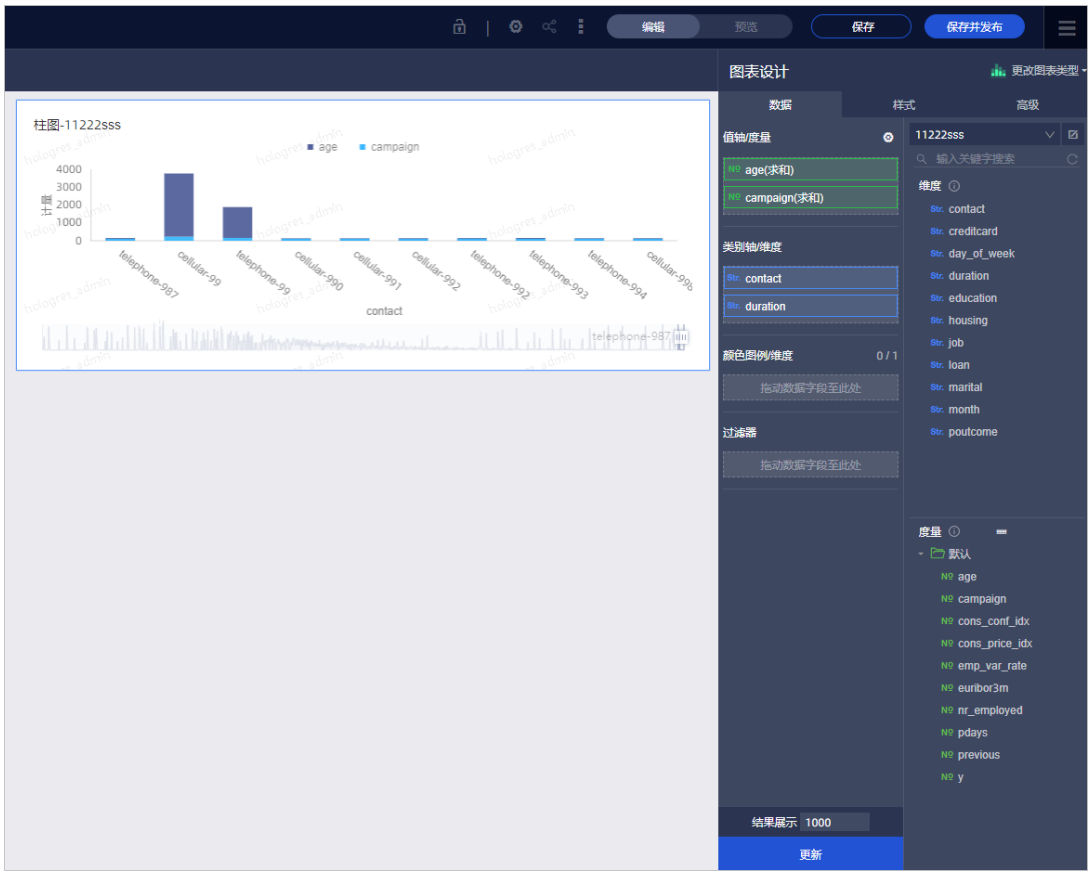


  c. 单击**确定**。
  d. 在**我的数据集**页面，您可以单击目标数据集**操作**列下的图标，进行相应操作：

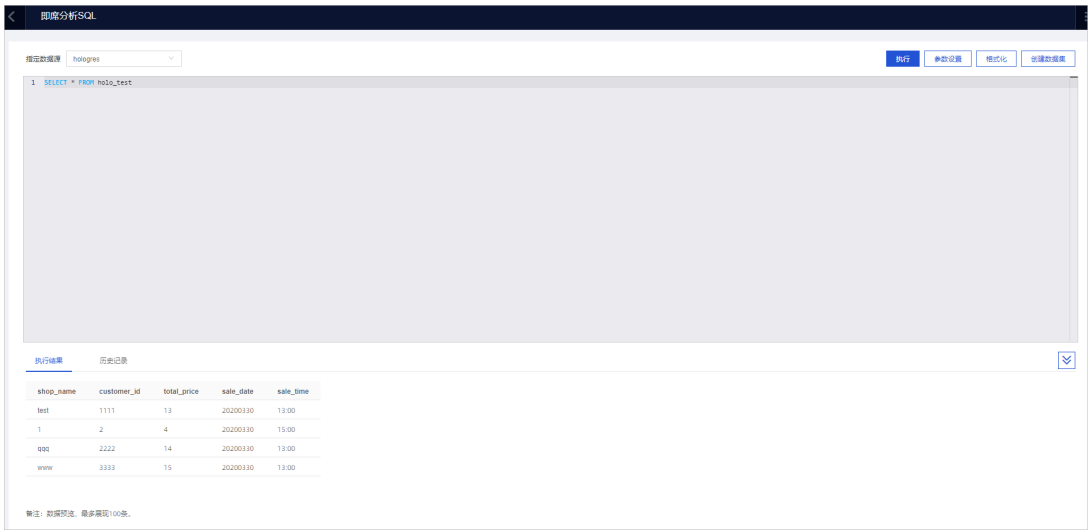
    - 单击图标，查看当前数据集的数据。
    - 单击图标，新建仪表盘，将表数据导入并生成图表，进行可视化展示。
    - 单击图标，新建电子表格，将目标表数据导入并生成电子表格，进行可视化展示。
    - 单击图标，执行更多操作。例如，数据脱敏、缓存配置及清除缓存等。

    具体功能的操作，详情请参见概述。

示例新建仪表板，在仪表板编辑页面创建堆积柱状图，并设计图表样式展示数据。



- 即席分析SQL方式。
  a. 在**数据源**页面，单击右上角的**即席分析SQL**，进入**即席分析SQL**页面。
  b. 您可以根据业务需求输入查询SQL语句，并单击**执行**。



  c. 获取需要查询的数据后，单击**创建数据集**。

d. 配置**保存自定义SQL**对话框的**名称**、**位置**及SQL参数。

e. 单击**确定**。

成功创建数据集后，您可以在**数据集**中可视化分析表数据，详情请参见即席分析SQL建模。

时间筛选控件最佳实践

在Hologres创建表时，您可以设置Segment_key（别名event_time_column）属性，对时间类型过滤条件进行索引优化，防止全表扫描并加速查询。Hologres默认把表中第一个时间戳类型作为Segment_key。

Quick BI支持高级的SQL占位符功能，您可以使用SQL占位符来设置时间控件。更多SQL占位符信息，请参见即席分析SQL建模。

1. 设置时间筛选字段占位符。

   执行以下步骤前，需要在Hologres创建名称为LINEITEM的内部表，并且将MaxCompute公共空间MAXCOMPUTE_PUBLIC_DATA中public.odps_lineitem_10g表的数据导入到LINEITEM表中，详细步骤请参见Hologres查询体验快速入门。

   i. 在Quick BI管理控制台的**数据源**页面，单击右上角的**SQL创建数据集**，进入**新建代码片段**对话框。

   ii. 输入以下SQL语句，从LINEITEM表中取出时间戳数据。

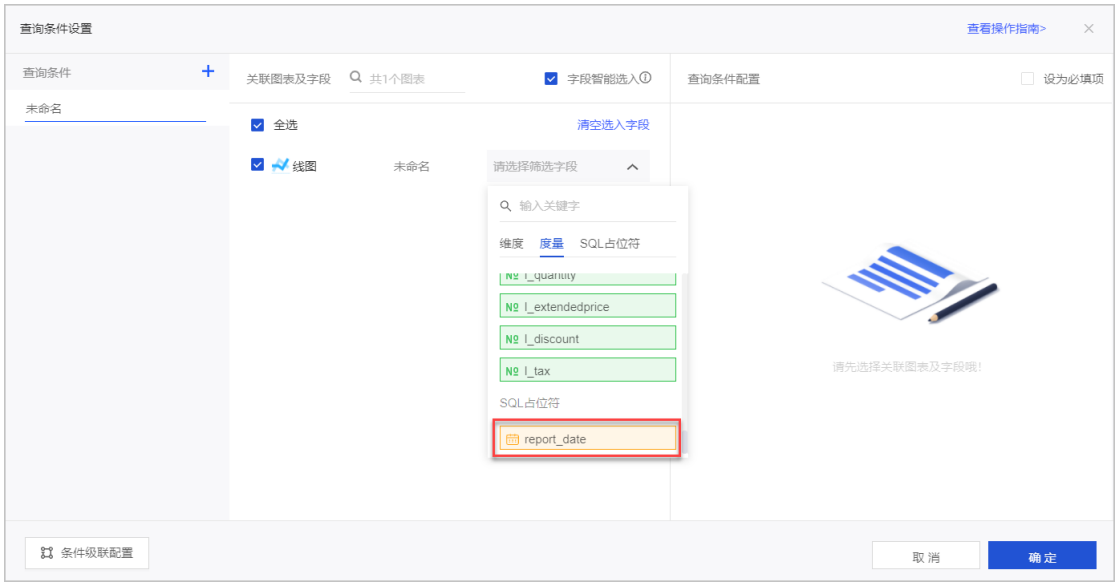
```
SELECT
    *
FROM
  "public"."lineitem" AS AME_T_1_
WHERE
  AME_T_1_."l_shipdate" >= TO_TIMESTAMP('${report_date.get(0)}', 'yyyy-MM-dd hh24:mi:ss')
  AND AME_T_1_."l_shipdate" <= TO_TIMESTAMP('${report_date.get(1)}', 'yyyy-MM-dd hh24:mi:ss')
```

   iii. 单击**参数设置**，设置SQL占位符。

   iv. 在**变量类型**下拉列表选择“日期-年月日时分秒 > YYYY-MM-DD HH-MI-SS”。

   v. 单击**确定**后，单击**确认编辑**。

vi. 在**仪表板**编辑页面，单击**新建筛选项**，在**查询条件设置**对话框，字段选择所设置的SQL占位符。



vii. 在**仪表板**编辑页面，单击**查看SQL**，获取到如下SQL。

```
SELECT
  AME_T_1_."l_shipmode" AS T_A0_2_,
  AME_T_1_."l_shipinstruct" AS T_A1_3_,
  SUM(AME_T_1_."l_extendedprice") AS T_A2_4_
FROM
  "public"."lineitem" AS AME_T_1_
WHERE
  AME_T_1_."l_shipdate" >= TO_TIMESTAMP('1993-01-01 00:00:00', 'yyyy-MM-dd hh24:mi:
ss')
  AND AME_T_1_."l_shipdate" <= TO_TIMESTAMP('1998-12-31 23:59:59', 'yyyy-MM-dd hh24
:mi:ss')
GROUP BY
  AME_T_1_."l_shipmode",
  AME_T_1_."l_shipinstruct"
LIMIT
  1000 OFFSET 0
```

2. 验证Segment_key是否生效。

   您可查看SQL的执行计划中是否有Segment Filter的关键字出现以验证Segment_key是否生效。

   i. 登录Hologres管理控制台，单击左侧**实例列表**。

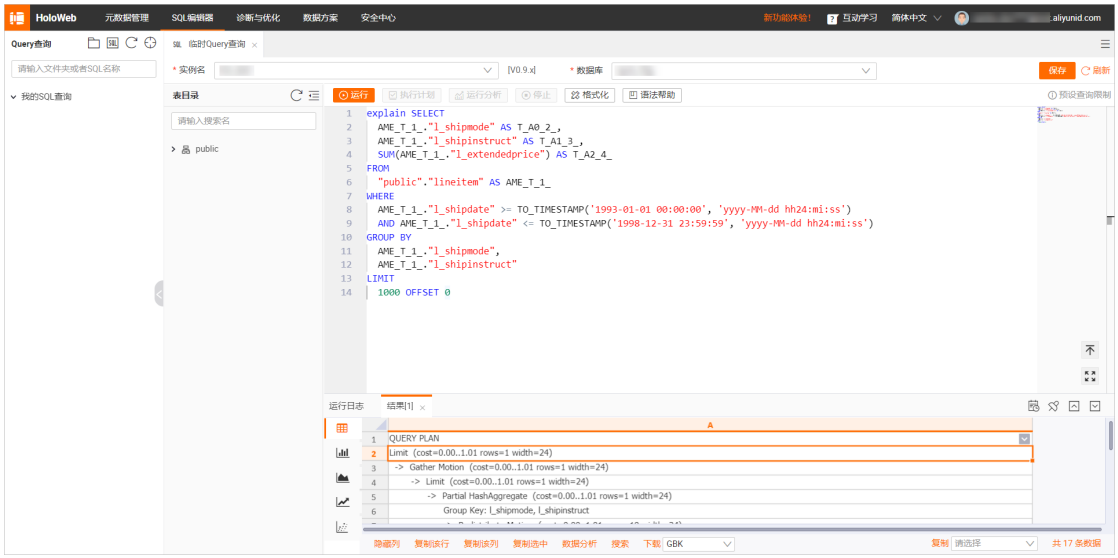
   ii. 在**实例详情**页左侧导航栏，单击**Database管理**。

   iii. 在**DB授权**页面，单击**SQL编辑器**。

   iv. 在**SQL编辑器**页面，单击左上角的**新建SQL窗口**。

v. 在新增的**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

以下SQL语句用来查看获取到SQL的执行计划，用于验证Segment_key是否生效。




```
explain SELECT
  AME_T_1_."l_shipmode" AS T_A0_2_,
  AME_T_1_."l_shipinstruct" AS T_A1_3_,
  SUM(AME_T_1_."l_extendedprice") AS T_A2_4_
FROM
  "public"."lineitem" AS AME_T_1_
WHERE
  AME_T_1_."l_shipdate" >= TO_TIMESTAMP('1993-01-01 00:00:00', 'yyyy-MM-dd hh24:mi:ss')
  AND AME_T_1_."l_shipdate" <= TO_TIMESTAMP('1998-12-31 23:59:59', 'yyyy-MM-dd hh24:mi:ss')
GROUP BY
  AME_T_1_."l_shipmode",
  AME_T_1_."l_shipinstruct"
LIMIT
  1000 OFFSET 0
```

vi. 在**结果**页签中获取执行计划，如下。

```
->  Limit  (cost=0.00..1.01 rows=1 width=24)
      ->  Partial HashAggregate  (cost=0.00..1.01 rows=1 width=24)
            Group Key: l_shipmode, l_shipinstruct
            ->  Redistribute Motion  (cost=0.00..1.01 rows=10 width=24)
                  ->  Result  (cost=0.00..1.01 rows=10 width=24)
                        ->  Partial HashAggregate  (cost=0.00..1.01 rows=10 width=2
4)
                              Group Key: l_shipmode, l_shipinstruct
                              ->  Parallelism (Gather Exchange)  (cost=0.00..1.01 r
ows=32 width=24)
                                    ->  Result  (cost=0.00..1.01 rows=32 width=24)
                                          ->  DecodeNode  (cost=0.00..1.01 rows=32
width=24)
                                                ->  Partial HashAggregate  (cost=0.
00..1.01 rows=32 width=24)
                                                      Group Key: l_shipmode, l_ship
instruct
                                                      ->  Index Scan using holo_ind
ex:[1] on lineitem  (cost=0.00..1.00 rows=1000 width=24)
                                                            Segment Filter: ((l_shi
pdate >= '1993-01-01 00:00:00+08'::timestamp with time zone)
```

执行计划中，出现了Segment Filter，说明Segment_key已生效，您可以使用SQL占位符创建的时间控件。

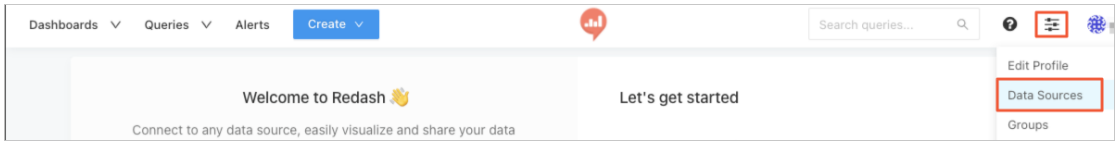
7.16. Redash

Redash是一款开源BI工具，可以使用其进行数据可视化。同时也具备了报警、订阅等功能。Hologres兼容PostgreSQL，支持直接连接Redash进行数据分析。本文为您介绍Redash如何连接Hologres并可视化分析数据。

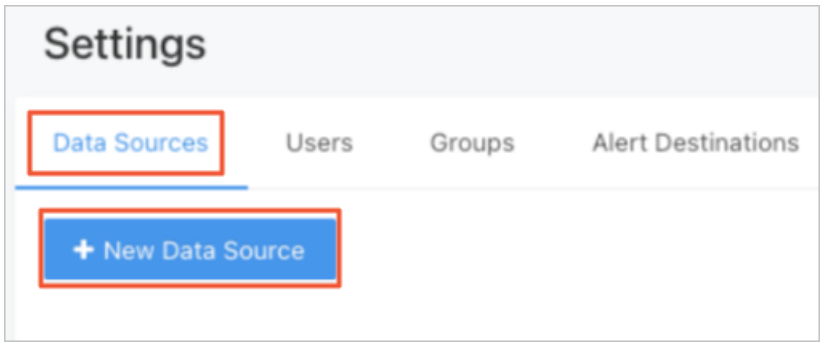
操作步骤

1. 安装Redash
   安装Redash，详情请参见Redash官方文档。
2. 连接Hologres

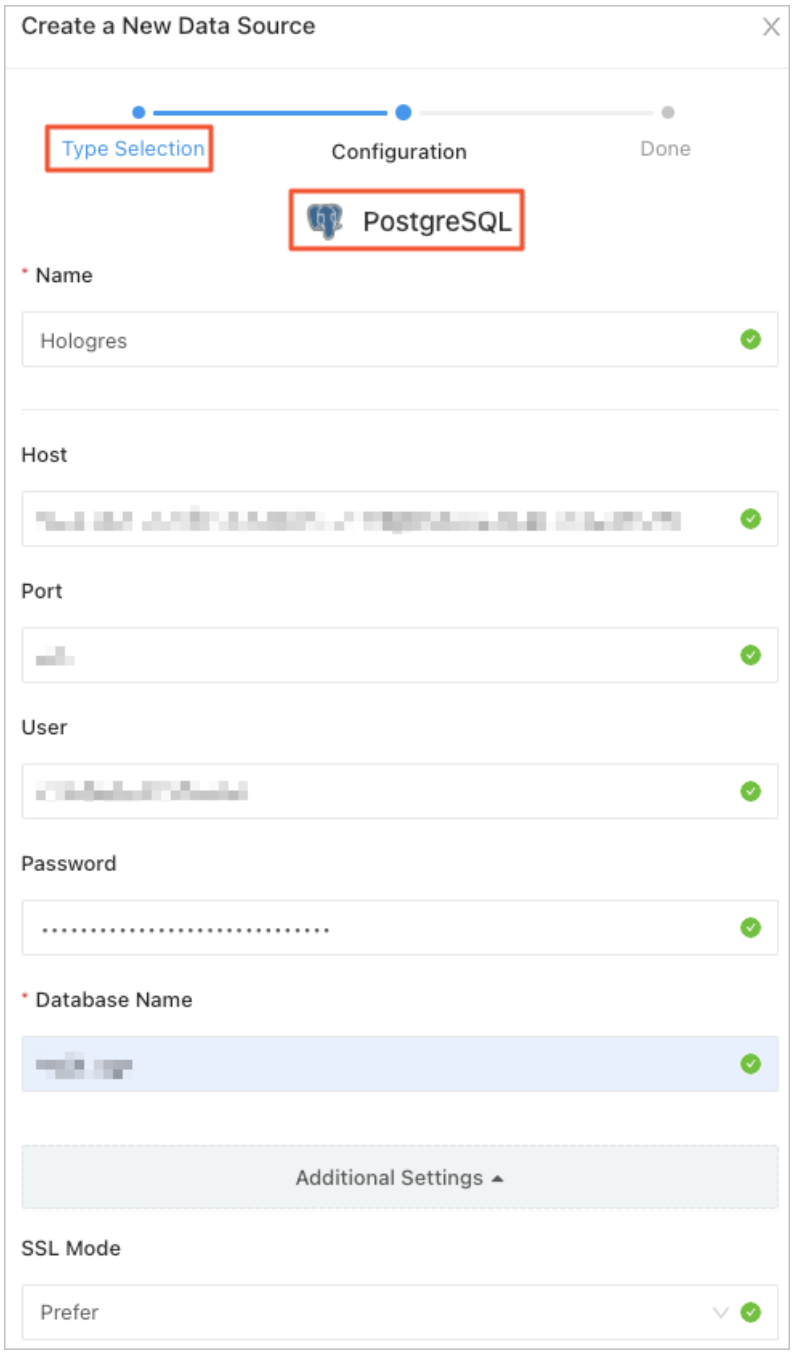
   i. 登录Redash，单击页面右上方⚙的图标，从下拉选项中选择Data Sources。



ii. 进入**New Data Source**页签，单击**+New Data Source**添加数据源。

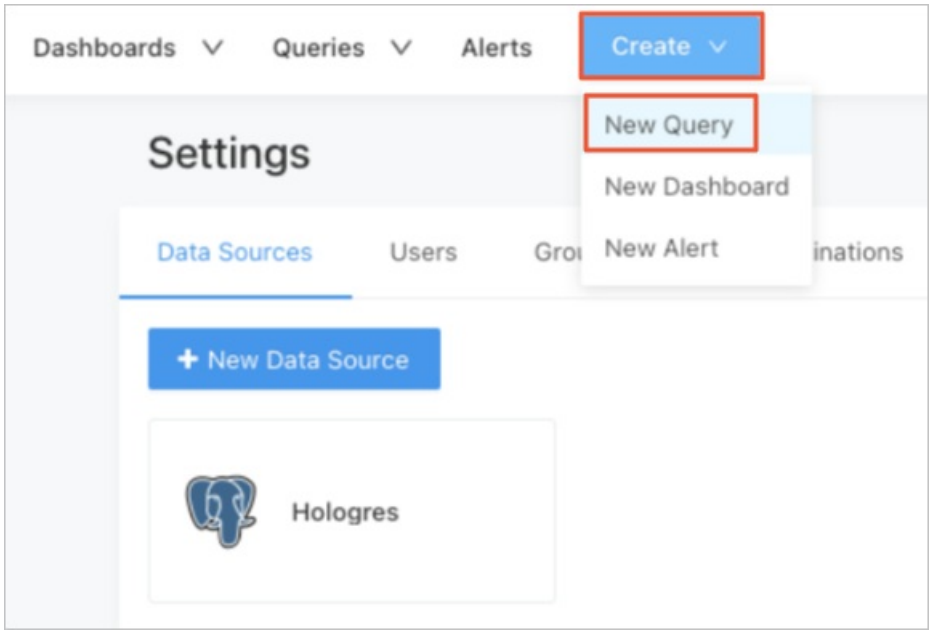


iii. 在弹框页面配置参数后，单击**Create**创建数据连接，您可以在Redash查看Hologres的数据。

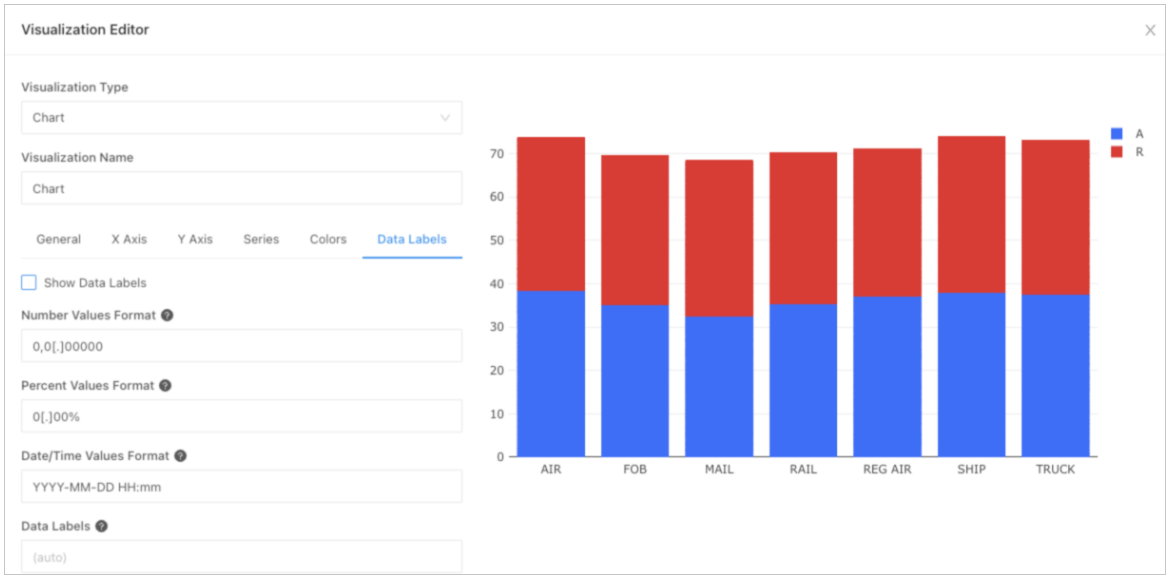


参数	说明	
Type Selection	PostgreSQL	添加数据源时，在搜索框中查找PostgreSQL，并单击进行添加。
Configuration	Host	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
	Port	Hologres实例的端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。
	User	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
	Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
	Database Name	Hologres创建的数据库名称。

3. 在页面上方选择**Create > New Query**，您可以在页面的编辑框中编写SQL语句。



4. SQL语句编写完成后，可以根据SQL创建可视化图表。



更多关于Redash的Query编写事项及创建可视化内容的操作，请参见Redash官方文档。

7.17. SAP BusinessObjects

SAP BusinessObjects（简称SAP BO）可提供报表、仪表盘、即席分析和数据管理功能，是SAP的知名的BI软件。本文将为您介绍如何SAP BusinessObjects连接Hologres并可视化分析数据。

操作步骤

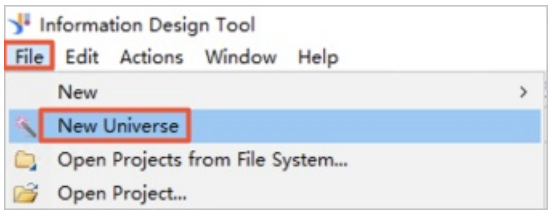
1. 安装PostgreSQL ODBC驱动

   您可以从PostgreSQL官网下载PostgreSQL ODBC驱动。

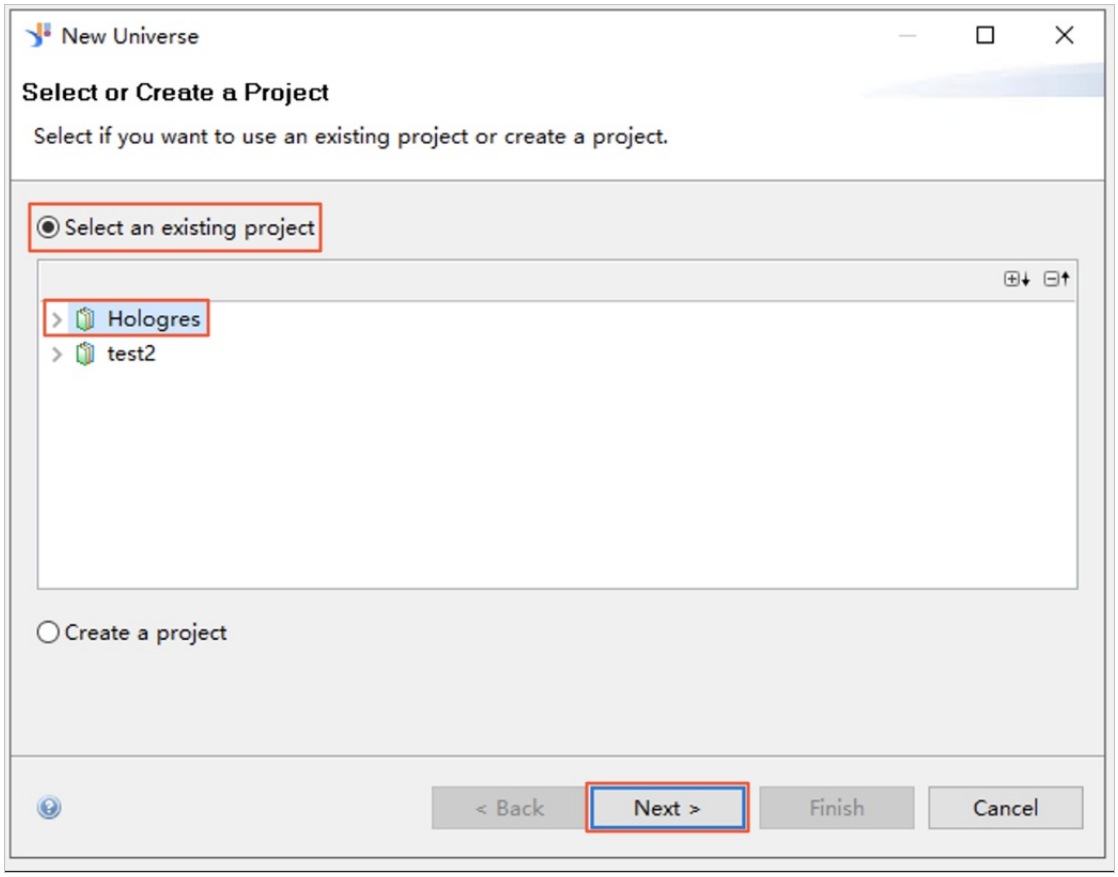
   > **说明** 由于Hologres兼容PostgreSQL 11的生态，请您务必在SAP BusinessObjects服务器端和客户端安装psqlodbc_11_01_0000以上的版本。

2. 创建Relational Connection

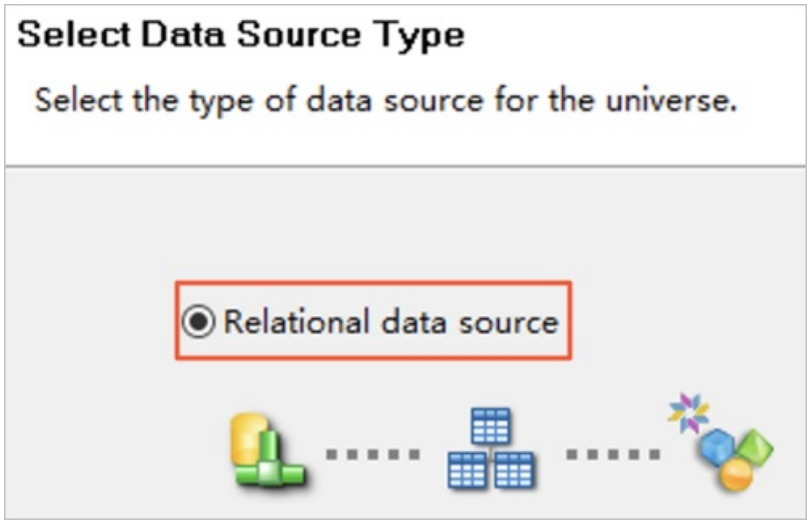
   i. 打开SAP Information Tool，选择**File > New Universe**



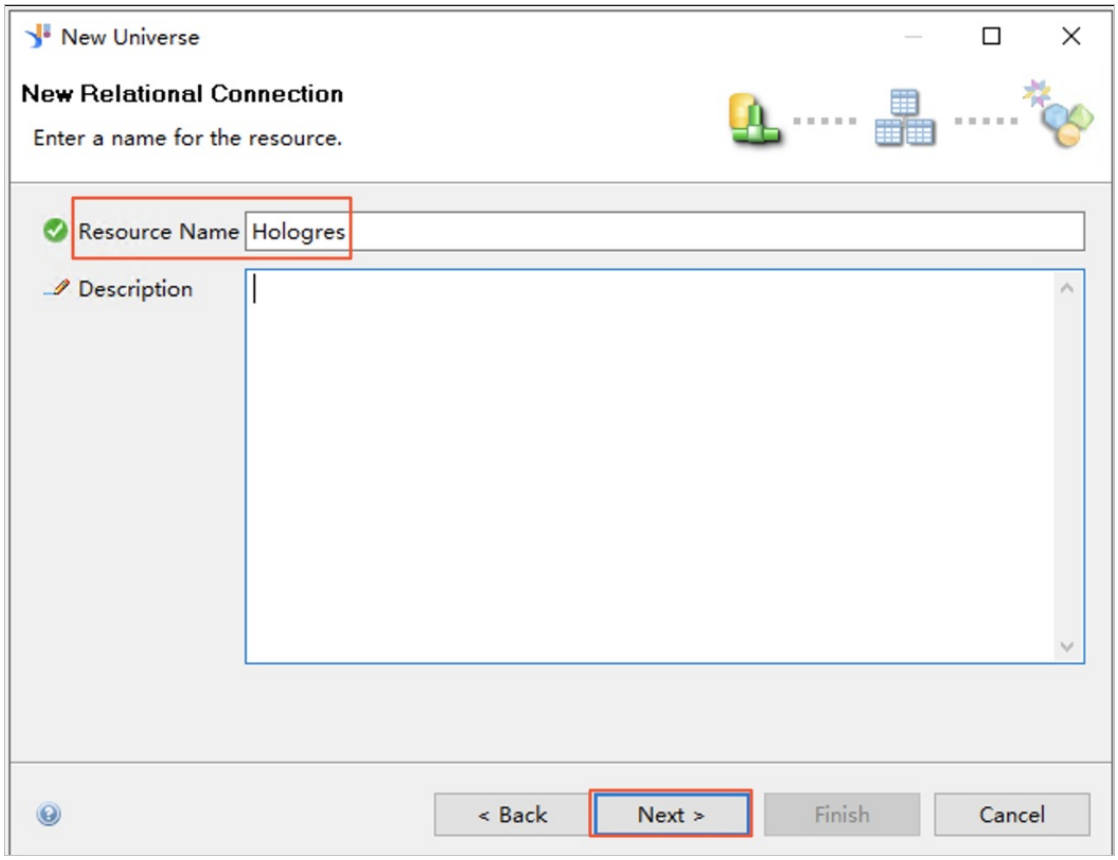
ii. 您可以从已有的Project中选择需要创建连接的Project，单击**Next**。如果当前不存在Project，您也可以选择**Create a project**进行创建。



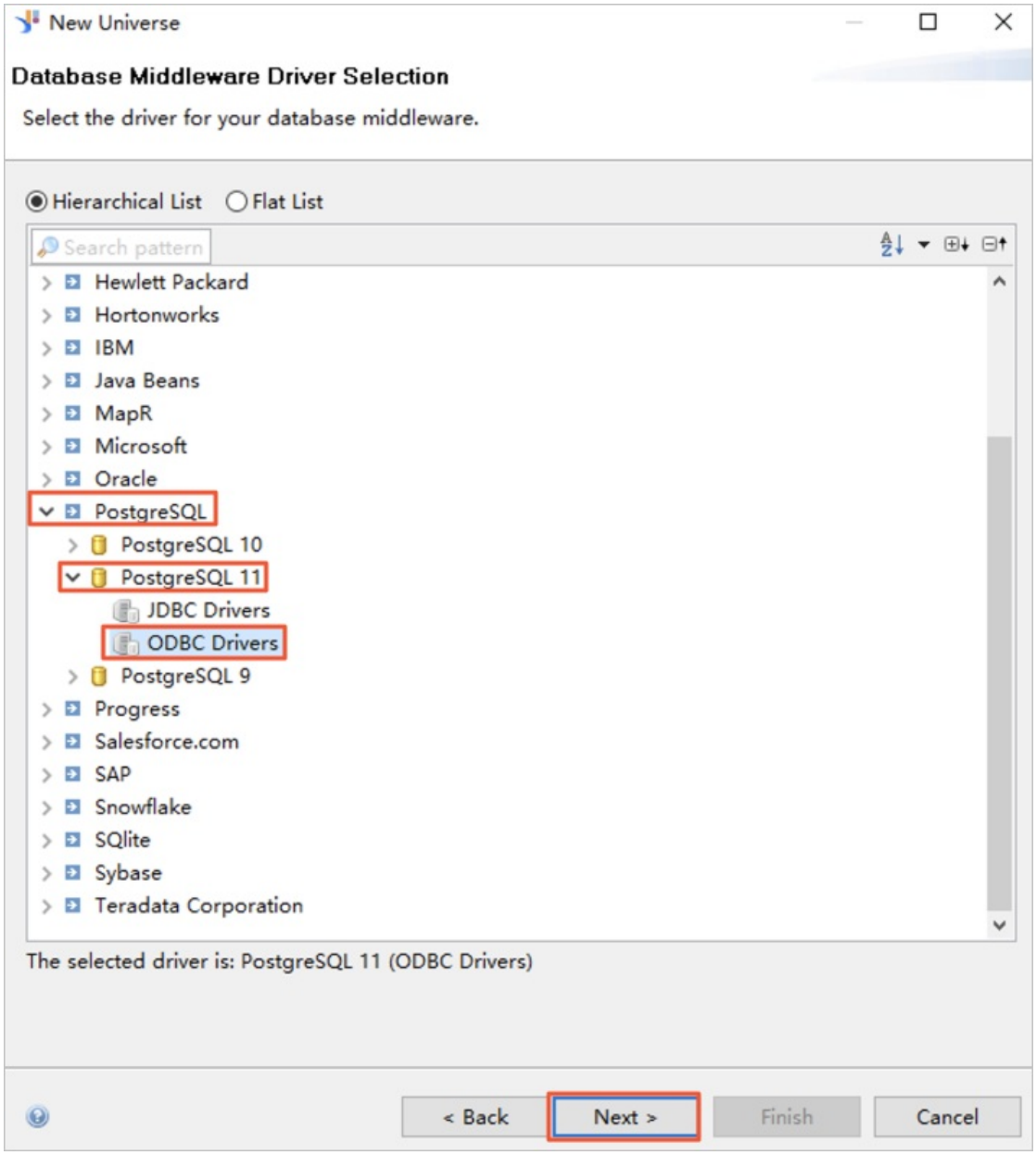
iii. 选择Relational data source，单击**Next**。



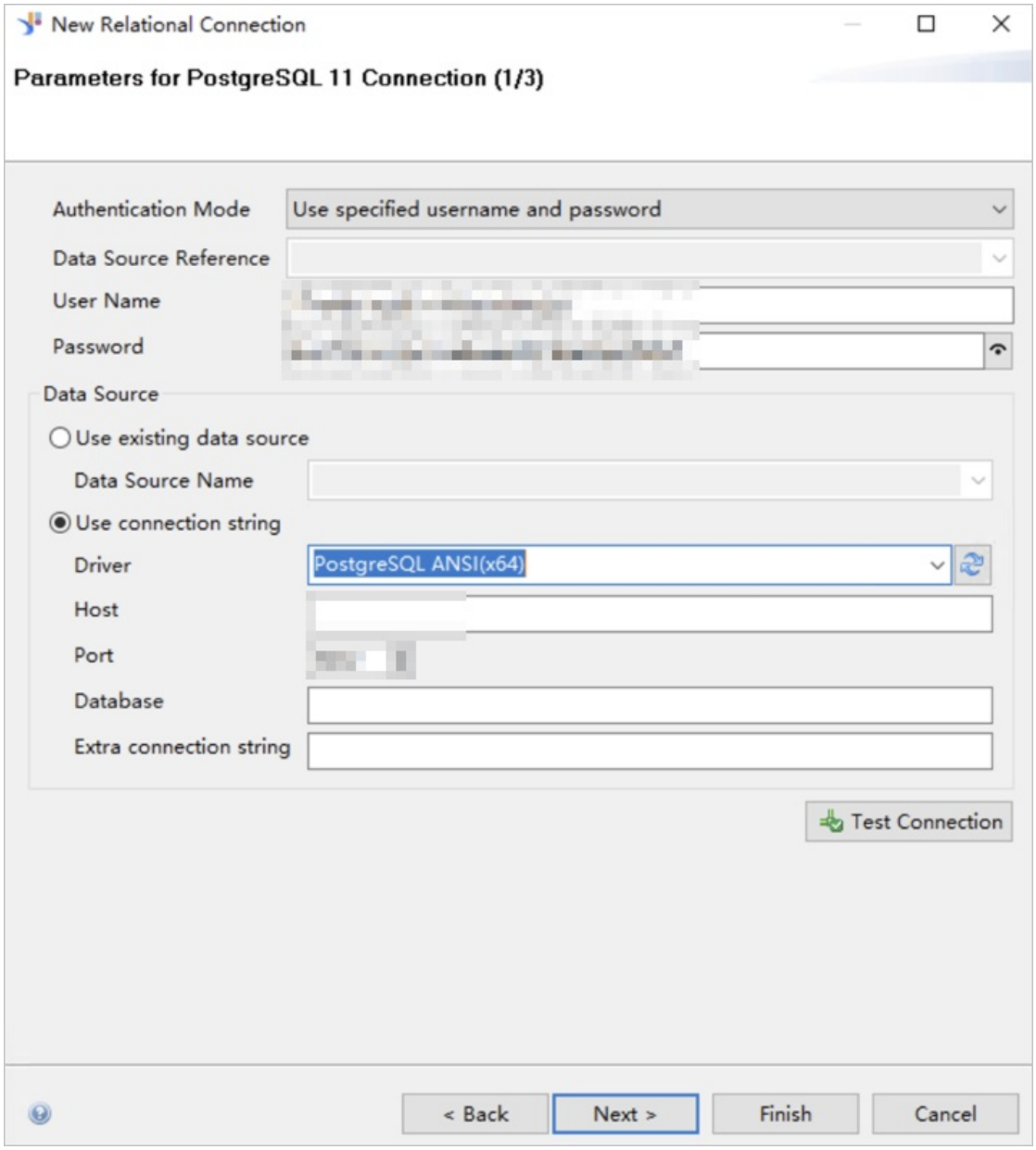
iv. 填写Resource Name，单击**Next**。



v. 在列表中选择PostgreSQL > PostgreSQL 11 > ODBC Drivers，单击Next。



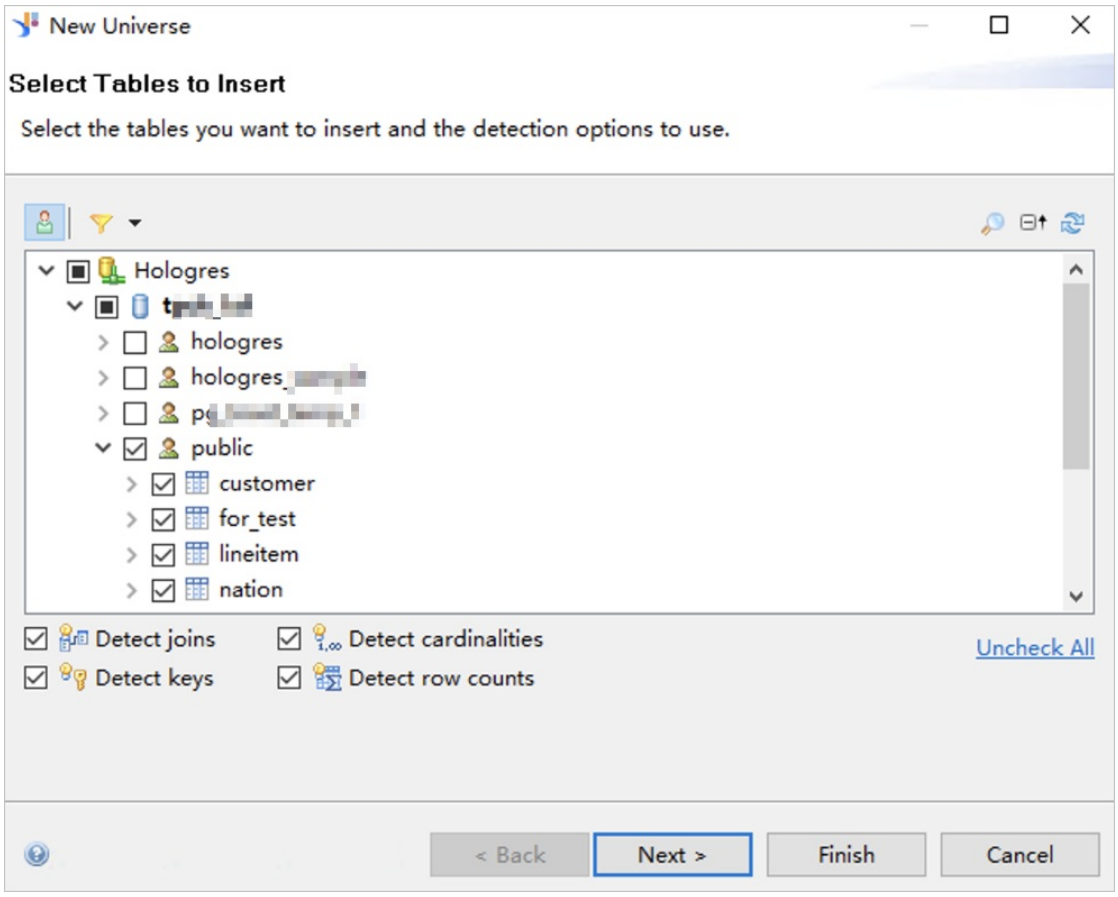
vi. 输入User Name、Password，选择Use connection string，单击Driver后的图标，从下拉框中选择目标Driver并配置剩余参数。具体参数说明如下表所示。



参数	说明
User Name	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
Password	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
Host	Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
Port	Hologres的实例端口。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端口。
Database	Hologres的数据库名称。

vii. 单击Test Connection，页面提示Test Successful表示连接成功。单击Next，进行连接配置和参数配置，如果没有特殊的配置需求，可以直接单击Next。

3. 创建Data Foundation
   i. 在**New Data Foundation**页面，填写Resource Name，单击**Next**。
   ii. 在列表中勾选需要加入的表，单击**Next**。



4. 创建Relational Business Layer
   在**New Relational Business Layer**页面，填写Resource Name，单击**Finish**。执行完成后，即创建了SAP BusinessObjects到Hologres的连接，您就可以开始进行模型设计，创建可视化分析等操作。

7.18. Tableau

本文为您介绍Tableau如何连接Hologres并可视化分析数据。

背景信息

Tableau是安全并且灵活的端到端数据分析平台，提供从连接到协作的一整套功能。Hologres兼容PostgreSQL，支持直接连接Tableau并可视化分析数据。

Tableau Desktop 连接 Hologres

1. 下载并安装Tableau。

   进入Tableau官网，根据业务需求下载相应的Tableau客户端，并根据提示安装。本次试验使用Tableau Desktop。

2. 连接Hologres。
   i. 成功安装客户端后，打开客户端。

ii. 在左侧导航栏的**连接 > 到服务器**区域，选择**PostgreSQL**，配置连接Hologres的信息。

参数说明如下表所示。

参数	描述
服务器	Hologres实例的公共网络地址。 进入Hologres管理控制台的实例详情页，从**实例配置**获取公共网络地址。
端口	Hologres的实例端口。 进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
数据库	Hologres创建的数据库名称。 进入Hologres管理控制台的实例详情页，从**DB管理**获取数据库名称。
身份验证	选择**用户名和密码**。
用户名	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
密码	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
需要SSL	不勾选。

iii. 单击**登录**。

3. Tableau可视化分析数据。

   使用Tableau成功连接Hologres后，您可以可视化分析已有的表数据，详情请参见Tableau官网教程。

发布至Tableau Server

如果您需要将Tableau Desktop的报表发布至Tableau Server进行分析和管理，您可以按照如下步骤操作：

1. 下载并安装Tableau Server。

   进入Tableau官网，根据业务需求下载相应的Tableau Server客户端，并根据提示安装。

2. 通过Tableau Desktop访问Tableau Server。

   i. 打开Tableau Desktop客户端，在顶部菜单选择**服务器 > 登录**。

   ii. 在弹窗中输入您对应Tableau Server的地址之后，单击**连接**。

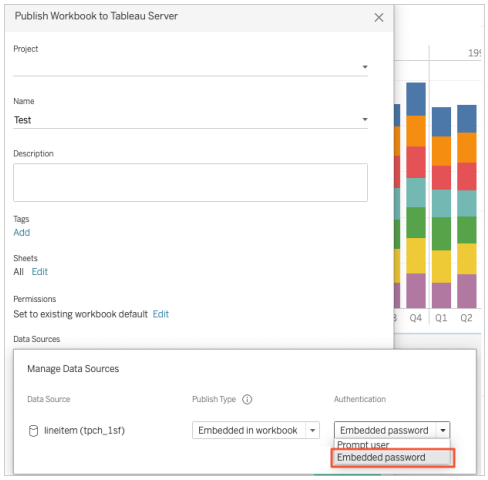
   

   iii. 在登录页面输入Tableau Server的用户名和密码，单击**登录**。

3. 将工作簿发布至 Tableau Server。

   i. 登录Tableau Server客户端后，在顶部菜单选择**服务器 > 发布工作簿**。

ii. 在**发布工作簿**对话框中，配置如下参数信息。



参数	说明
项目	选择目标项目名称。
名称	为工作簿输入一个名称
标记	在用户浏览服务器时，标记可帮助用户查找相关工作簿。使用逗号或空格分隔标记。
权限	接受默认项目设置。
数据源	Tableau Server支持两种数据源身份验证类型： ■ 提示用户 ■ 嵌入密码 选择嵌入式密码时，Tableau会将该报表与数据源的连接信息内嵌在报表中，并允许任何可以查看工作簿的用户查看工作簿。由于连接 Hologres时需要使用Access ID和Access Key，较为繁琐，所以更推荐使用嵌入式密码模式。

iii. 参数配置完成后，单击**发布**。您就可以在Tableau Server中查看您发布的报表信息。

常见问题

- 查询结果中TIMESTAMPTZ类型数据时区异常
  - 可能原因：对于TIMESTAMPTZ的数据类型，Hologres的默认时区为PRC，Tableau默认的时区是UTC，所以对于TIMESTAMPTZ类型的数据会因为时区不一致导致显示的时间不一致。

- 解决方法：通过在Tableau中设置初始化SQL的方式解决时区不一致问题。

  > **说明** 该配置仅对设置的数据源生效。

  设置初始化SQL分为如下两种场景。

  - 首次连接Hologres。

    如果是首次连接Hologres，请参见如下步骤。

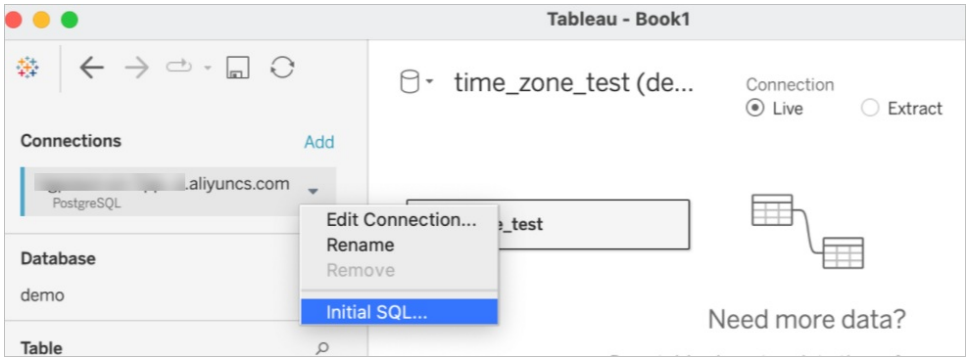
    建立Tableau连接Hologres的数据连接时，在**Initial SQL**中输入如下SQL语句。

    ```
    SET TIME ZONE "PRC";
    ```

  - 已经建立了Tableau和Hologres之间的连接，请参见如下步骤。

    a. 单击**Initial SQL**。

    b. 在**Initial SQL**弹窗，输入如下SQL命令。

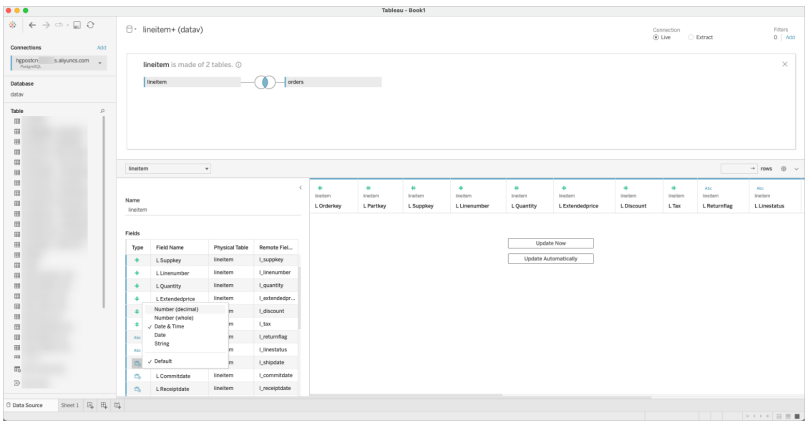
    ```
    SET TIME ZONE "PRC";
    ```

    c. 单击**OK**，完成设置初始化SQL。

    d. 在页面左上角，单击图标，设置生效。

- 时间类型维度设置

建议在Tableau使用时间类型的维度时，尽量不要在Tableau中做二次转换。例如如下例子中，希望将l_shipdate作为日期&时间格式使用，那么在Hologres建表时应该直接将该列设置为TIMESTAMP或者TIMESTAMPTZ格式，以避免多余的字段类型转换。

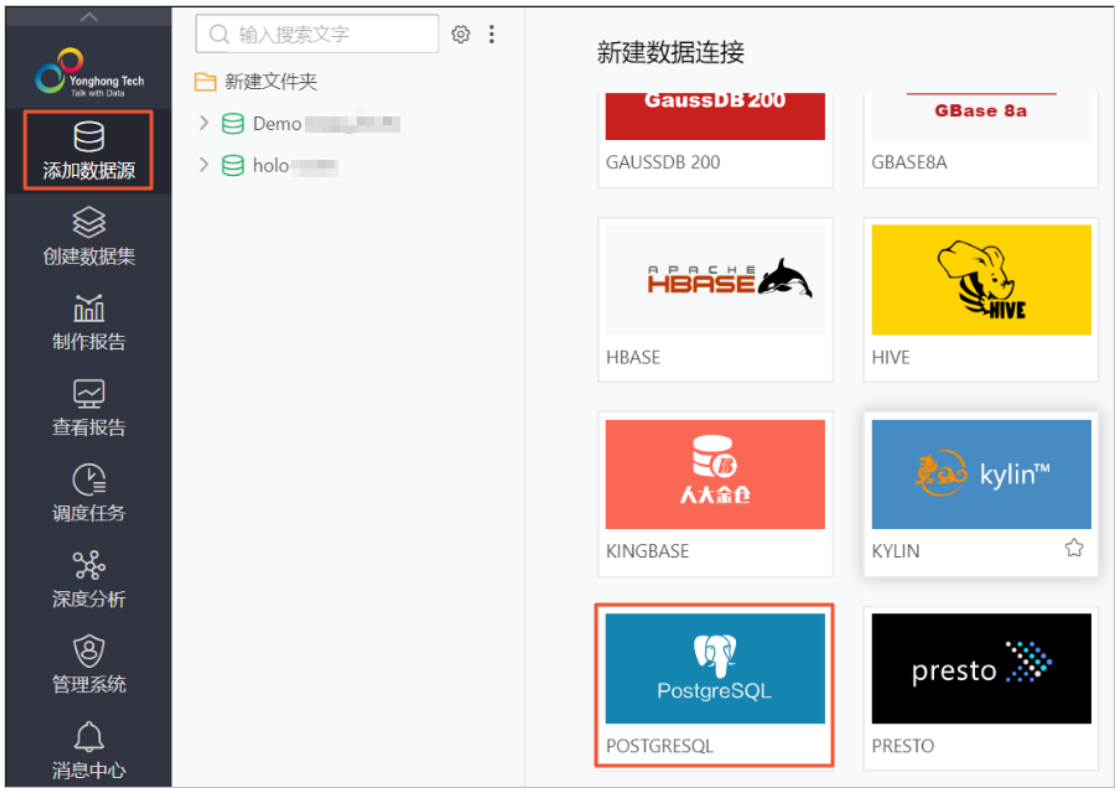


7.19. Yonghong BI

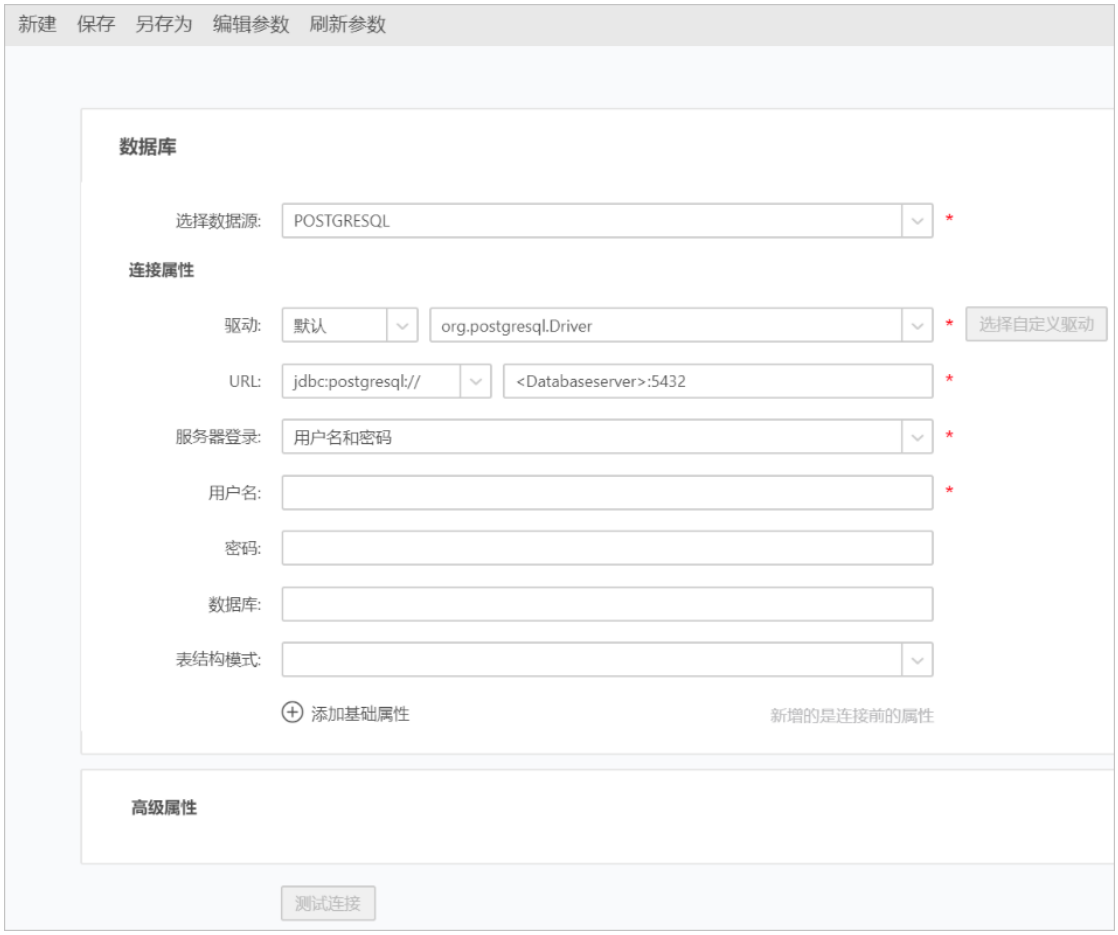
本文将为您介绍如何通过Yonghong BI工具连接Hologres并进行数据分析。

操作步骤

1. 下载并安装客户端。
   下载并安装永洪Yonghong BI客户端，请参见Yonghong官方文档的安装产品章节。
2. 连接Hologres。
   i. 登录客户端，在左侧导航栏单击**添加数据源**，选择**POSTGRESQL**进入新建页面。



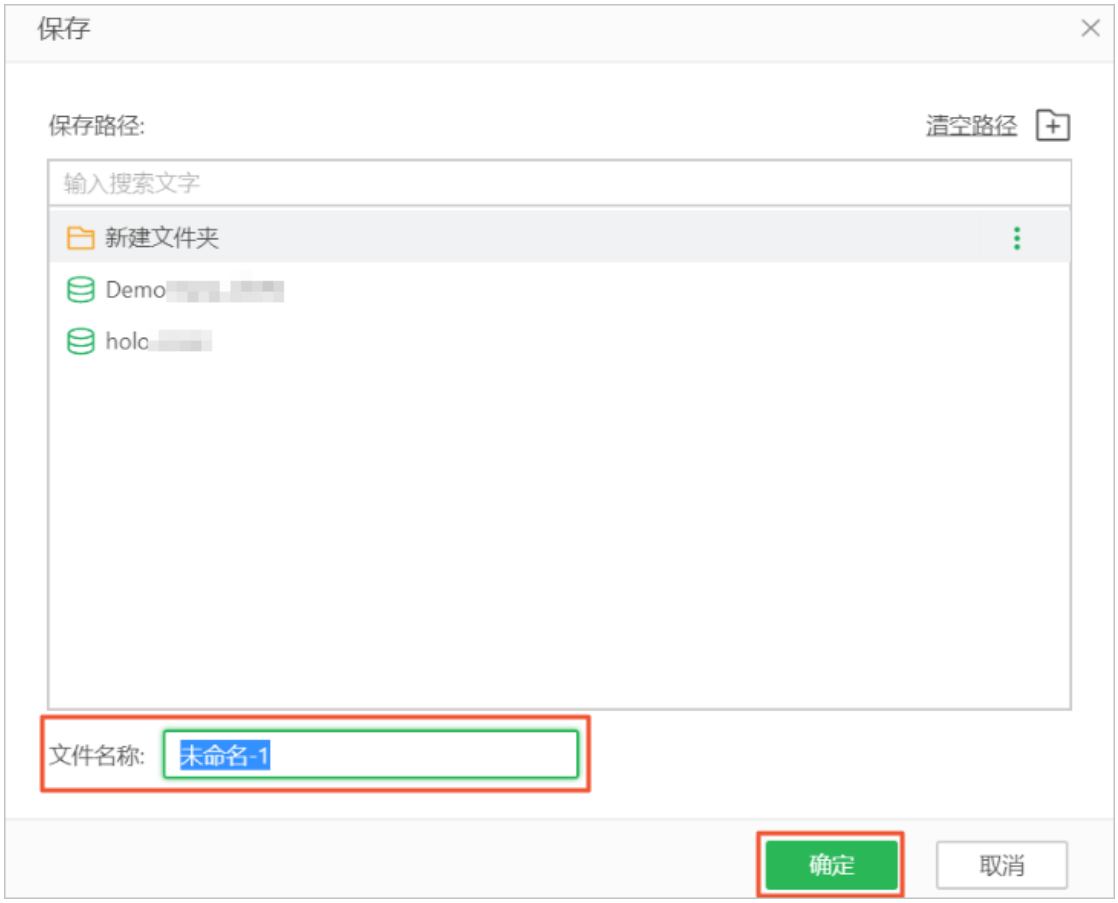
   ii. 在弹框中配置连接信息，并单击**测试连接**。如果提示测试成功，即该数据源成功连接到Hologres。



具体参数说明如下：

参数	说明
选择数据源	您可以选择**POSTGRESQL**。
驱动	驱动选择 `org.postgresql.Driver` 。
URL	数据连接URL，填写格式为 `jdbc:postgresql://<Databaseserver>:<port>` ，其中各参数解释如下所示： ■ Databaseserver：主机，Hologres实例的公共网络地址。您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。 ■ port：端口，Hologres实例的端口。
服务器登录	您可以选择**用户名和密码**方式登录。
用户名	当前阿里云账号的AccessKey ID。获取方式请参见创建访问密钥。
密码	当前阿里云账号的AccessKey Secret。获取方式请参见创建访问密钥。
数据库	Hologres的数据库名称。
表结构模式	输入数据库下的Schema名称。

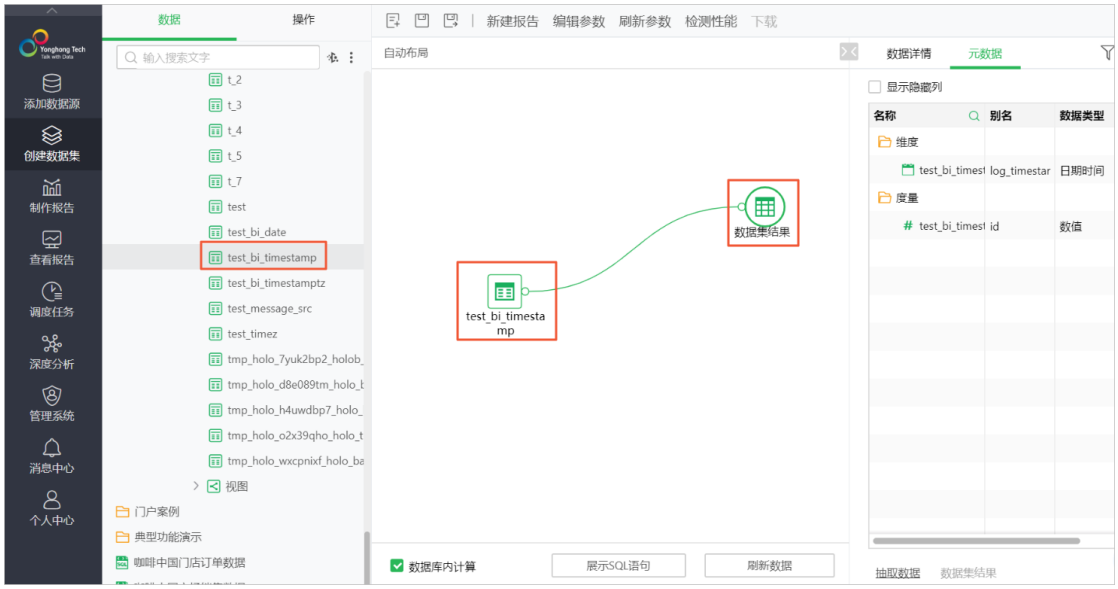
iii. 单击页面上方的**保存**，并输入文件名称。单击**确定**，保存该数据源。



3. 创建数据集。

i. 登录客户端，在左侧导航栏单击**创建数据集**，在页面中选择**自服务数据集**进入新建页面。

ii. 单击页面上方的**数据集**，在列表中单击目标数据源。将目标表拖拽到右侧空白画布区域，并将表和**数据集结果**相连。

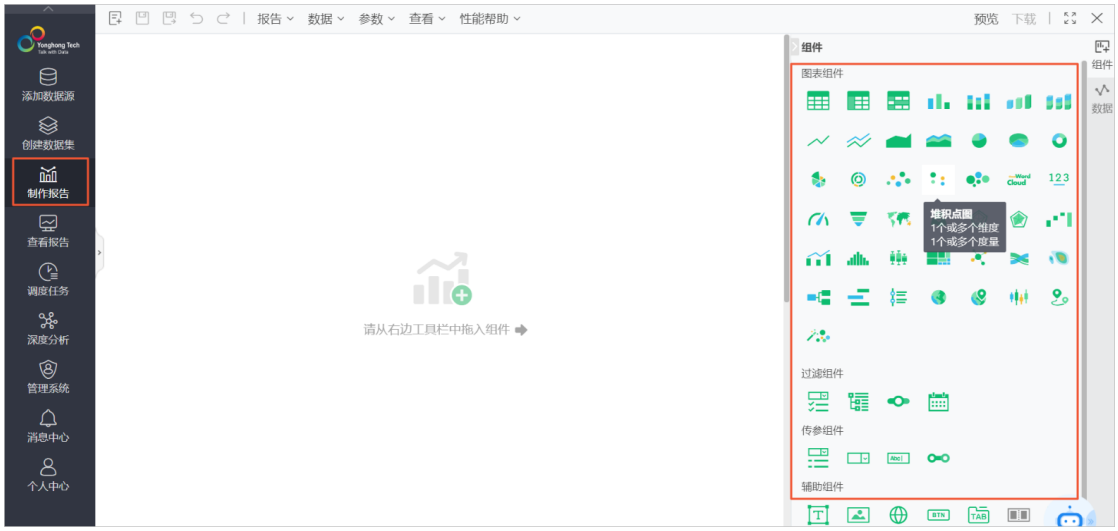


iii. 单击页面上方的图标，并输入文件名称。单击**确定**，保存该数据集。

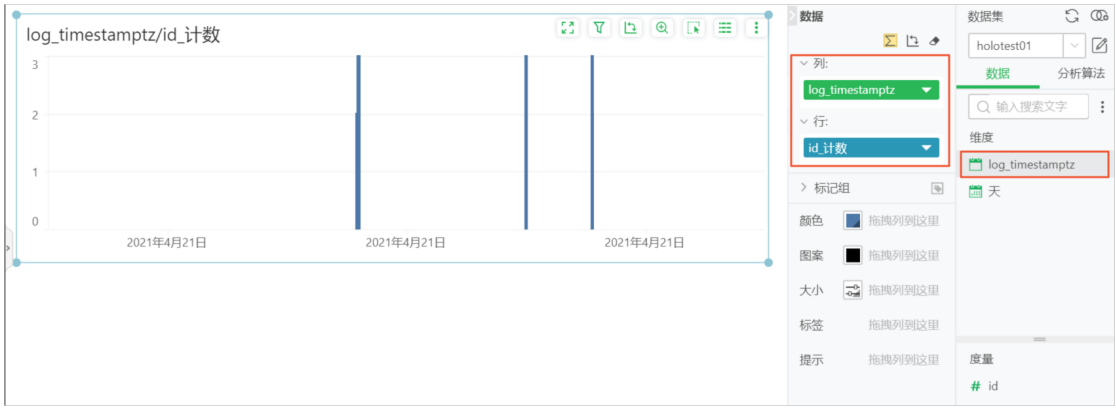
4. 制作报告。

i. 登录客户端，在左侧导航栏单击**制作报告**，在页面中选择**新建报告**。在主题页面选择目标主题来创建报告。

ii. 选择主题后，打开编辑页面。在右侧组建中单击目标组件，将其拖拽到左侧报告编辑区域。



iii. 您可以对选定的组件选择一个数据集进行报告的制作。更多关于报告的操作指导，请参见Yonghong官方文档的制作可视化报告章节。



8.监控与运维

8.1. 实例升级

Hologres实例当前支持两种升级方式：标准升级和热升级。本文为您介绍Hologres实例的升级方式和升级所需要提供的信息。

背景信息

Hologres具备向下兼容版本的能力，大版本的发布会进行新功能迭代、优化等；小版本发布会进行缺陷修复等。详细版本差异请参见产品版本发布动态和关键缺陷通知，推荐所有的用户升级到Hologres当前发布的最新正式版本。

升级方式

升级方式	升级时间	实例状态	作业情况	升级场景	说明
标准升级（停服升级）	5~10分钟	升级期间服务不可用。	• 全托管Flink和DataHub涉及写入Hologres表的需要停止任务，待Hologres实例升级完成后再启动任务。否则可能会出现数据丢失。 • DataWorks数据集成任务和Blink任务无需暂停任务，升级期间会触发Failover，根据Failover策略自动恢复，建议Failover重试次数配置为10次以上。	• 支持小版本升级： 例如从0.10.25升级至0.10.36。 • 支持大版本升级： 例如从0.10升级至1.1。	需要提交工单由Hologres运维人员操作，升级不会改变实例的Endpoint，但是无法保证Endpoint的IP地址不变更。
热升级	10~30分钟	升级期间，系统处于只读状态（查询服务不受影响，写入服务不可用）。			Hologres在V1.1版本开始支持热升级模式。需要提交工单进行升级，升级不改变系统Endpoint，但无法保证IP地址不变更。

> **说明** 升级期间，建议暂停实时同步作业，升级完成之后再启动。

升级信息

Hologres实例升级需要提供升级信息，分为如下两种升级场景。

- 小版本升级

  小版本升级不需要做实例检查准备，可以直接升级。请直接提交工单升级，并提供实例ID和实例升级时间窗口。

- 大版本升级

  大版本升级需要对实例进行检查，请先提交工单，并在工单中备注如下信息。

信息项	说明
客户名称	请填写您的用户名称。
实例ID	请填写需要升级Hologres实例的ID，在实例配置页面获取。
实例所在地域	请填写需要升级Hologres实例的地域可用区，在实例配置页面获取。
当前实例版本	请填写需要升级Hologres实例的当前版本号，在实例配置页面获取。
是否有实时写入	○ 如果存在Blink、Flink、开源Flink，请填写所使用实时写入的版本号。 说明 如果实时写入版本过低，升级可能会操成数据丢失，需要升级Blink版本在3.7.9及以上，Flink（VVR）版本在3.0及以上才可对Hologres实例进行升级。 ○ 如果存在DataWorks实时写入，请提供运行中的实时同步日志和所属Region，如果有多个独享数据集成资源组，那每个资源组都需要提供一个日志。
期望升级时间	请填写您希望Hologres实例升级的时间。 说明 ○ 为了保证实例版本间兼容度，参数个性化设置等，在升级前需要对实例先进行一个检查，等检查完成才能进行升级，一般检查时间窗口是1~3天，请耐心等待运维人员通知升级时间。 ○ 因为升级会停止服务，升级因版本和实例规格不同而略有差异，几分钟到十几分钟不等，请耐心等待。

8.2. 资源隔离与高可用

8.2.1. 单实例Shard多副本高吞吐（Beta）

本文将为您介绍在Hologres中Shard级别的Replication使用。

功能概述

从Hologres V1.1版本开始，支持通过设置Table Group副本数的方式来提高某个Table Group查询并发能力和可用性。您可以在创建Table Group的时候通过显示指定的replica count或者修改已有的replica count来打开Replication的功能。新增的副本属于内存级别的运行时副本，不会增加额外存储成本。

关于replica count的说明具体如下：

- 数据是按Shard分布的，不同Shard管理不同的数据，不同Shard之间的数据没有重复，所有的Shard在一起是一份完整的数据。
- 默认情况下每一个Shard只有一个副本，即 `replica count = 1` ，其对应的属性为leader。您可以通过调整replica count的值，使相同的数据有多个副本，其他副本的属性为follower。
- 写请求由leader shard负责，读请求会均衡由多个follower shard（也包含leader shard）共同服务。当使用follower shard查询时，数据可能会出现10~20ms级别延迟。
- replica count默认值是1，表示不启用Replication。大于1表示开启Replication，建议设置为2，replica

count数字越大，对资源的消耗也越大，最多可以有3个Replication，即replica count数为4。由于Replica布局具有反亲和特性，即多个replica不可以布局在同一个计算节点上，因此replica count参数应小于等于计算节点数。有关不同规格拥有的计算节点数，参考实例规格概述。

- 考虑到计算节点计算力的均衡性，在增加Replication时，应该同步缩小shard_count，保持 `shard_count * replica count = 默认实例推荐shard数` 时，具备最好的性能。

使用限制

在Hologres中使用Shard级别的Replication时，仅Hologres V1.1及以上版本支持，请在Hologres管控台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。

语法说明

- **查询当前DB的Table Group**

  您可以使用如下语法查看当前DB有哪些Table Group。

  ```
  select * from hologres.hg_table_group_properties ;
  ```

- **查询已有Table Group的replica count**
  - 语法示例

    ```
    select property_value from hologres.hg_table_group_properties where tablegroup_name = '
    table_group_name' and property_key = 'replica_count';
    ```

  - 参数说明

参数	说明
table_group_name	请输入您需要查询的Table Group名称。
replica_count	此处为固定参数名称，无需修改。

- **开启Replication**

  通过以下命令开启Replication功能，开启是数据库或者session级别的操作，开启完成后，还需要通过指定replica count为Table Group开启Replication功能。请务必先完成replica_count的调整，再开启读Replication功能，否则在副本复制的过程中，系统开销高，可能引起读操作的超时。建议新建空Table Group的方式，开启Replication，减少对已有业务的调整。用户查询端无需修改。

  ```
  -- 为Table Group 开启replication
  call hg_set_table_group_property ('table_group_name', 'replica_count', '2');
  -- 为database开启read replica功能
  alter database <database_name> set hg_experimental_enable_read_replica = on;
  ```

  参数说明如下。

参数	说明
database_name	数据库名称。
hg_experimental_enable_read_replica	是否开启read replica功能，每个DB执行一次即可。 ○ on：开启read replica功能。 ○ off：关闭read replica功能。

参数	说明
hg_set_table_group_property	修改Table Group的replica_count。 ○ table_group_name：请输入您需要修改的Table Group名称。 ○ replica_count：设置目标Table Group的副本数量，replica_count应小于计算节点数，一般是2。 ○ 设置是否开启Replication：1为默认值，表示不启用Replication功能。大于1的数值表示启用Replication功能。

- **关闭Replication**
  - 语法示例

```
-- 修改replica_count，关闭replication
call hg_set_table_group_property ('table_group_name', 'replica_count', '1');
```

  - 参数说明

参数	说明
hg_set_table_group_property	修改Table Group的replica_count。 ■ table_group_name：请输入您需要修改的Table Group名称。 ■ replica_count：设置目标Table Group的副本数量。 ■ 设置是否开启Replication：1为默认值，表示不启用read replica功能。大于1的数值表示启用read replica功能。

8.2.2. 单实例计算资源隔离（Beta）

Hologres支持细粒度资源管理能力，通过为不同的用户账号分配不同的计算资源（即CU，包括CPU和内存），限制用户使用计算资源的上限，实现单实例多负载的隔离，保证了用户之间、应用之间作业的互不影响。本文为您介绍如何使用资源组管理Hologres实例内的计算资源，实现资源隔离。

背景信息

Hologres V1.0及以下版本支持在实例间进行资源隔离，不支持对于实例内部进行更细粒度的用户级别的资源隔离。但是在实际生产环境中，往往需要根据用户在实例内部进行资源隔离，限制每个用户使用的资源上限，以保证用户之间的作业互不影响。为满足上述细粒度的资源隔离诉求，Hologres新增支持使用资源组来助力您管理Hologres实例内的计算资源，实现资源隔离。

使用限制

- 仅Hologres V1.1及以上版本支持使用资源组管理Hologres实例内的计算资源，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅限具备Superuser权限的用户使用资源组管理Hologres实例内计算资源，否则系统会提示权限不足。
- 计算资源属于实例级别，如果用户有多个数据库，所有数据库共享同一个实例的计算资源，所有数据库共享同一份资源分配方案。

HoloWeb可视化配置资源组

通过HoloWeb可视化配置资源组，相关操作如下。

- **新建资源组**
  i. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
  ii. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
  iii. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
  iv. 在**资源组管理**页面，选择目标实例名称，单击**新增资源组**。

  v. 在**新增资源组**对话框，输入**资源组名称**并设置**资源组配额**，单击**确认**，即可完成资源组的新建。

> **说明** 一个Hologres实例内所有资源组配额总和不能超过1，否则系统会报错。

- **删除资源组**
  i. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
  ii. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
  iii. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
  iv. 在**资源组管理**页面，单击对应资源组操作列的**删除**，进行资源组的删除。

> **说明** 如果有用户绑定到资源组，则该资源组不能被删除。

- **调整资源组配额**
  i. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
  ii. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。
  iii. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。
  iv. 在**资源组管理**页面目标资源组的资源组配额列，单击**调整配额**。
  v. 在**调整配额**对话框，调整资源组配额，单击**确认**。

- **绑定用户**

  创建资源组后，您可以使用HoloWeb可视化绑定用户至资源组。

  i. 进入HoloWeb开发页面，详情请参见连接HoloWeb。
  ii. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。

iii. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。

iv. 在**资源组管理**页面，单击对应资源组操作列的**绑定用户**。

v. 在**绑定资源组**页面，单击**新增绑定用户**。

vi. 在**绑定用户**对话框，选择用户，单击**确认**。

> **说明**
>
> - 如果在用户的下拉列表找不到对应的账号，则说明该账号并未添加至当前实例，您需要前往用户管理页面添加用户。
> - 一个用户仅能被绑定一个资源组，若重复绑定用户，以最新绑定的资源组为准。

- **解绑用户**

i. 进入HoloWeb开发页面，详情请参见连接HoloWeb。

ii. 在HoloWeb开发页面的顶部菜单栏，单击**安全中心**。

iii. 在**安全中心**页面，单击左侧导航栏的**资源组管理**。

iv. 在**资源组管理**页面，单击对应资源组操作列的**绑定用户**。

v. 在**绑定资源组**页面，单击对应用户操作列的**解绑用户**。

vi. 在**解绑用户**对话框，单击**确认**。

SQL方式配置资源组

- 查看资源组配置

查看所有资源组，每个资源组配置的配额，以及每个资源组绑定的用户的SQL语句如下。

```
SELECT  * FROM pg_holo_resource_groups;
```

样例结果如下。

```
 res_group_name | property_key |         property_value
----------------+--------------+-----------------------------------------
 resource_1     | worker_limit | 0.3
 default        | worker_limit | 0.7
 resource_1     | bind_users   | [ "13xxxxxxxxx13", "p4_29xxxxxxxxxx19" ]
```

- 新增资源组

> **说明**
>
> - 默认情况下系统会创建一个名为default的资源组，并将所有计算资源都分配到该资源组，且将所有未明确绑定资源的用户都绑定至该资源组。
> - 创建其他资源组后，default资源组的规格等于所有资源组分配之后的剩余部分。
> - 建议default资源组分配资源的百分比至少为0.3。

创建资源组的SQL语句如下。

```
CALL hg_create_resource_group ('resource_group_name', quota);
```

参数	说明
resource_group_name	资源组名称。您可以使用英文字符，数字和下划线自定义其命称，最大长度50个字符。
quota	资源组分配资源的百分比。取值范围为0.1~0.9，只支持一位小数点。

- 修改资源组配额

> **说明** 一个Hologres实例内所有资源组配额总和不能超过1，否则系统会报错。

修改资源组配额的SQL语句如下。

```
CALL hg_alter_resource_group ('resource_group_name', quota);
```

参数	说明
resource_group_name	资源组名称。资源组必须是已经创建的资源组，否则系统会报错。
quota	资源组分配资源的百分比。取值范围为0.1~0.9，只支持一位小数点。

- 删除资源组

> **说明** 如果有用户绑定到资源组，则该资源组不能被删除。

删除资源组的SQL语句如下。

```
CALL hg_drop_resource_group ('resource_group_name');
```

参数	说明
resource_group_name	资源组名称。资源组必须是已经创建的资源组，否则系统会报错。

- 将用户绑定至资源组

> **说明**
> - 一个用户仅能被绑定一个资源组，若重复绑定用户，以最新绑定的资源组为准。
> - 您可以使用如下SQL查看当前的用户。
>
> ```
> SELECT current_user;
> ```

在创建资源组后，您可以将用户绑定至资源组，以限制其使用的计算资源。将用户绑定至资源组的SQL语句如下。

- 语法示例

```
CALL hg_bind_resource_group('resource_group_name', 'user_name');
```

- 参数说明

参数	说明
resource_group_name	资源组名称。资源组必须是已经创建的资源组，否则系统会报错。
user_name	用户的名称。必须是存在的用户，且该用户必须拥有访问资源组对应实例的权限，否则系统会报错。

- 使用示例

```
CALL hg_bind_resource_group ('resource_1', 'p4_29xxxxxxxxxx');
--注意，阿里云账户需要添加双引号
CALL hg_bind_resource_group ('resource_1', '"ALIYUN$xxxx@aliyun.com"');
CALL hg_bind_resource_group ('resource_1', '"RAM$xxx@xxx:xxxx"');
CALL hg_bind_resource_group ('resource_1', '"13xxxxxxxxxx13"');
```

- 将用户从资源组解绑

> **说明** 将用户从资源组解绑后，该用户隶属于default资源组。

将用户从资源组解绑的SQL语句如下。

- 语法示例

```
CALL hg_drop_resource_group('resource_group_name', 'user_name');
```

- 参数说明

参数	说明
resource_group_name	资源组名称。资源组必须是已经创建的资源组，否则系统会报错。
user_name	用户的名称。必须是存在的用户，且该用户必须拥有访问资源组对应实例的权限，否则系统会报错。

- 使用示例

```
CALL hg_unbind_resource_group ('resource_1', 'p4_29xxxxxxxxxxxx9');
-- 注意，阿里云账户需要添加双引号
CALL hg_unbind_resource_group ('resource_1', '"RAM$xxxx@xxx:xxx"');
```

常见问题

Q：为什么资源组配额设置为0.5，CPU和内存指标仍然会大于50%？

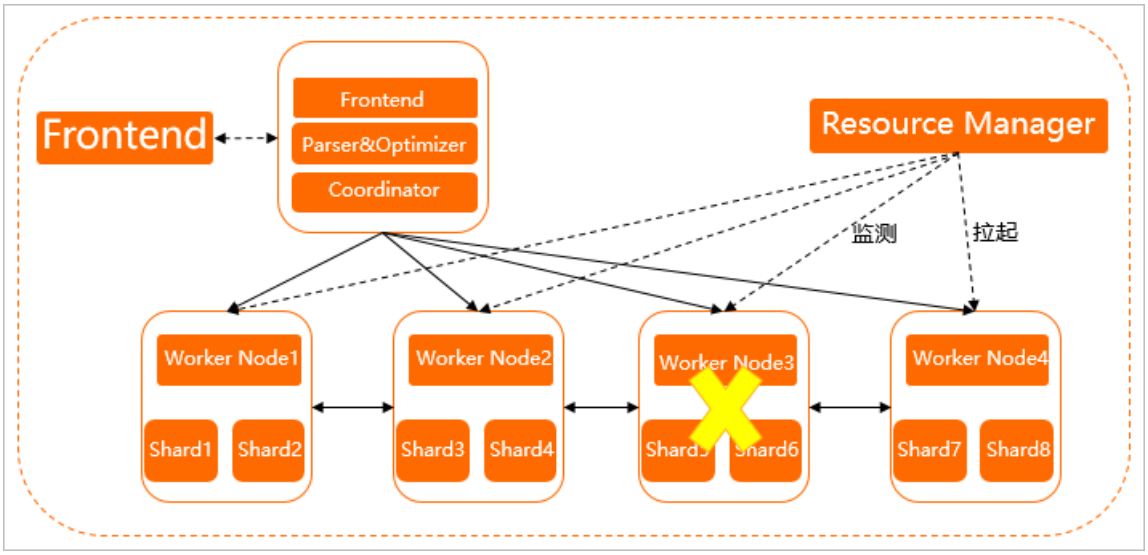
目前资源组配额限制了大部分查询引擎的CPU和内存，但是有一小部分CPU和内存在资源组之外不受限制（包括但不限于SQL解析、优化、元数据处理、调度、PQE执行、Compaction等）。同时实时写入情况下，CPU受限制，内存不受限制。所以会出现资源组总CPU和内存指标高于资源组所设置配额的情况。

8.2.3. 多实例读写分离高可用部署（共享存储）

Hologres 从V1.1版本开始，针对线上生产环境高可用的场景，提供了共享存储的多实例部署方式，在该模式下支持故障隔离，负载隔离，有效支撑了高可用场景。本文介绍高可用方案的一些基本原理以及如何配置共享存储多实例。

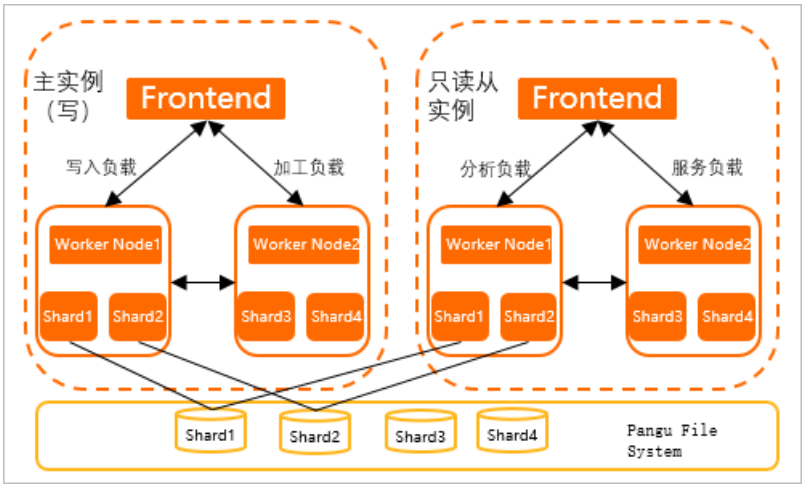
单实例自动恢复的高可用方案

Hologres计算节点均为容器调度（即下图中的Worker Node），资源管理器（Resource Manager）负责周期性健康检查。当出现1分钟容器响应超时（可能是内存溢出、硬件故障、软件Bug等原因导致），Resource Manager会自动拉起新的计算节点，并迁移Shard职责到新的节点上（例如Worker Node3响应超时，Resource Manager拉起Worker Node4取代Worker Node3），实现系统状态的快速恢复。数据状态保存在盘古分布式存储系统中，无需从计算节点迁移，计算节点轻量无状态，系统可以快速从故障中恢复。该方案为当前每个实例内部默认启用，当系统发生故障时，无需手工运维介入，系统可以自动恢复。在恢复期间，如果查询算子需要访问恢复中的节点，则查询会立即失败。Hologres从V1.1版本开始，采用全新恢复机制，节点恢复速度在一分钟左右，比早期版本提速5~10倍。



共享存储的多实例高可用方案

在单实例方案中，采用的是故障实时监测、节点替换的方案，在节点恢复时存在一定的服务不可用周期，对于关键业务场景，需要更高级别的高可用方案，支持故障隔离、负载隔离。Hologres 在V1.1版本，支持采用共享存储的多实例部署方案。在该方案中，主实例具备完整能力，数据可读可写，权限、系统参数可配置，而只读从实例处于只读状态，所有的变更都通过主实例完成，如下图所示。



- 当前最多可以配置四个只读从实例，各实例之间的资源配置可以不同，但不应该有明显差异，所有实例之

间Shard数相同。

- 不同只读从实例具备独立的访问Endpoint，在使用上，通过Endpoint隔离不同的业务场景。
- 所有实例共享同一份数据和访问控制，仅有一份存储费用。
- 实例之间内存状态自动实时同步，同一个Region毫秒级同步，同步以实例为单位，不支持细粒度同步单位配置。
- 实例之间计算资源不共享，负载隔离，故障隔离。
- 建议将主实例作为数据写入和加工节点，只读从实例作为OLAP和在线服务的节点。
- 由于实例的隔离性，减少了读写负载的冲突，减少了锁和事务的开销，可以显著提升节点的健壮性，也可以通过部署多个只读从实例，实现节点的高可用。

配置共享存储多实例

共享存储多实例使用限制如下。

- 仅Hologres V1.1及以上版本支持作为主实例，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 只读从实例未绑定主实例时，无法连接只读从实例。
- 主实例的版本和只读从实例版本必须一致。
- 主实例和只读从实例必须处于同一个Region。

绑定和解绑操作的权限说明

您需要为RAM用户授予AliyunHologresFullAccess权限策略，方可进行绑定和解绑只读从实例操作。更多关于RAM角色权限的介绍请参见授予RAM用户权限。

配置共享存储多实例的高可用操作步骤如下。

1. 购买新的Hologres实例

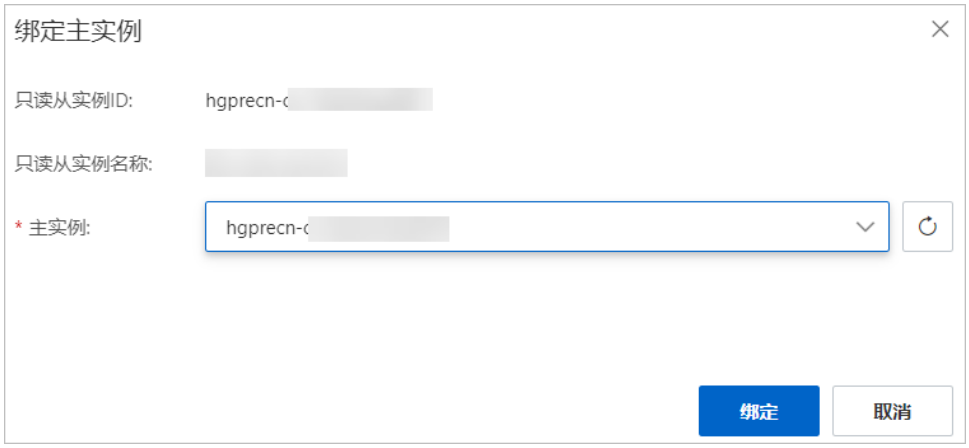
   > **注意** 只读从实例需要与主实例处在同一个Region中。

   购买新实例时，实例类型选择为只读从实例，详情请参见购买Hologres。

2. 绑定只读从实例

   i. 在Hologres管理控制台的**实例列表**页面，单击需要绑定只读从实例**实例类型**列的**绑定**。

ii. 在**绑定主实例**弹窗选择要绑定至的主实例，单击**绑定**完成绑定只读从实例。



3. 使用实例

使用共享存储多实例的过程中，需要注意如下事项。

- 配置完成后，可以用只读从实例的Endpoint提供线上服务。
- 所有的操作，包括建表、用户授权等都在主实例内完成，只读从实例只能读取数据。
- 只读从实例会自动拥有主实例内的所有对象，包括用户、表等，在权限控制层，不能为只读从实例单独创建用户。

8.3. 监控与告警

8.3.1. 云监控

云监控为您提供企业级开放型一站式监控解决方案。Hologres已经接入云监控的云服务监控，方便您通过云监控全面了解Hologres实例的资源使用、业务运行及健康状况，及时收到异常报警并做出响应，保证应用程序运行顺畅。本文为您介绍如何通过云监控监测Hologres实例的相关指标并上报告警。

前提条件

- 开通Hologres，详情请参见psql快速入门。
- 开通云监控，详情请参见概览。

使用限制

Hologres仅支持使用自定义大盘的方式查看实例的相关指标。

授予RAM用户云监控查看权限

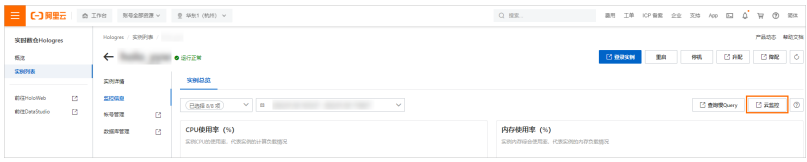
您可为RAM用户授权如下权限，授权方法请参见为RAM用户授权。

权限名称	权限功能描述
AliyunCloudMonitorFullAccess	管理云监控（CloudMonitor）的权限。
AliyunCloudMonitorReadOnlyAccess	只读访问云监控（CloudMonitor）的权限。
AliyunCloudMonitorMetricDataReadOnlyAccess	访问云监控（CloudMonitor）时序指标数据的权限。

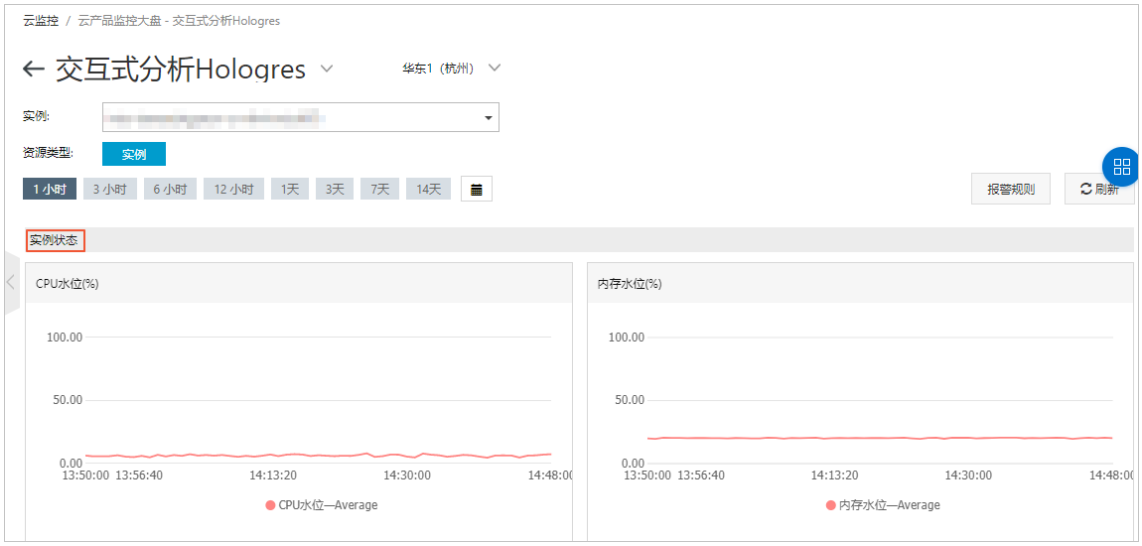
查看监控指标

进入云监控的方式有两种，分别为从Hologres管理控制台跳转和直接登录云监控控制台。

- 从Hologres管理控制台跳转
  i. 登录Hologres管理控制台，单击左侧**实例列表**。
  ii. 在**实例列表**页面，单击对应实例名称。
  iii. 在**实例详情**页左侧导航栏，单击**监控信息**。
  iv. 在**监控信息**页面，单击右上角**云监控**，即可进入云监控Hologres实例监控页面查看实例相关指标的状态。



- 直接登录云监控控制台
  i. 登录云监控控制台。
  ii. 在左侧导航栏，单击**云产品监控**。
  iii. 在**大数据（数加）**区域，单击**交互式分析Hologres**进入Hologres监控大盘。
  iv. 单击地域后的图标，选择目标地域。
  v. 单击目标实例ID或操作列的**监控图表**，查看实例相关指标的状态。



> **说明** 您可以自定义查看实例指标的时间段，监控数据最多保留30天。

创建并配置报警规则

您可以根据业务的实际情况，创建并配置监控指标的阈值报警，当监控指标超过设定阈值后，系统自动发送报警通知，帮助您及时发现监控数据的异常并快速处理。创建并配置报警规则的步骤如下：

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 报警规则**。
3. 在**阈值报警**页面，单击**创建报警规则**。

4. 在**创建报警规则**页面，配置报警规则的各项参数。

参数	描述
产品	选择**交互式分析Hologres**。
资源范围	报警规则的作用范围。取值如下： ○ **全部资源**：表示该规则对用户名下对应产品的所有实例生效。例如：您设置了全部资源粒度的Hologres CPU使用率大于80%报警，则只要用户名下有Hologres的CPU使用率大于80%，就会发送报警通知。 当**资源范围**选择**全部资源**时，报警的资源最多为1000个，超过1000个可能会导致达到阈值不报警的问题，建议您使用应用分组按业务划分资源后再设置报警。 ○ **应用分组**：报警规则作用于Hologres的指定应用分组内的全部资源上。 ○ **实例**：表示该规则只对某个具体的实例生效。例如：您如果设置了实例粒度的主机CPU使用率大于80%报警，则当该实例CPU使用率大于80%时，会发送报警通知。
规则描述	报警规则的主体。当监控数据满足报警条件时，触发报警规则。规则描述的设置方法如下： i. 单击**添加规则**。 ii. 在**添加规则描述**面板，设置规则名称、监控指标类型、监控指标、阈值、报警级别和报警方式等。 iii. 单击**确定**。
通道沉默周期	指报警发生后如果未恢复正常，间隔多久重复发送一次报警通知。
生效时间	报警规则的生效时间，报警规则只在生效时间内才会检查监控数据是否需要报警。 **说明** 单击**高级设置**，可设置该参数。
报警联系人组	发送报警的联系人组。 应用分组的报警通知会发送给该报警联系人组中的报警联系人。报警联系人组是一组报警联系人，可以包含一个或多个报警联系人。 关于如何创建报警联系人和报警联系人组，请参见创建报警联系人或报警联系组。
报警回调	公网可访问的URL，用于接收云监控通过POST请求推送的报警信息。目前仅支持HTTP协议。关于如何设置报警回调，请参见使用阈值报警回调。
弹性伸缩	当**产品**选择**弹性伸缩**时，您需要设置弹性伸缩的**地域**、**弹性伸缩组**和**弹性伸缩规则**。在发生报警时触发相应的伸缩规则。
日志服务	当**产品**选择**日志服务**时，您需要设置日志服务的**地域**、Project和Logstore。在发生报警时，将报警信息写入日志。
消息服务MNS-Topic	如果您打开**消息服务MNS-Topic**开关，当报警发生时，会将报警信息写入消息服务的主题。您需要设置消息服务的地域和主题。 关于如何创建主题，请参见创建主题。

参数	描述
无数据报警处理方法	无监控数据时报警的处理方式。取值： ○ **不做任何处理**（默认值） ○ **发送无数据报警** ○ **视为正常**

5. 单击**确认**。

   您也可以单击目标实例ID**操作**列的**报警规则**，查看或修改已有的报警规则。

配置一键报警

您可以在云监控为Hologres开启一键报警功能，为所有实例设置默认的告警规则，方便对多个常见的重要指标进行异常告警，快速发现问题。Hologres默认的告警规则如下：

- 如果连接数使用率（Info）连续3次平均值>=95就报警，通知对象为**云账号报警联系人**。
- 如果存储水位（Warn）连续3次平均值>90%就报警，通知对象为**云账号报警联系人**。
- 如果内存水位（Warn）连续3次平均值>=90%就报警，通知对象为**云账号报警联系人**。
- 如果CPU水位（Info）连续3次平均值>=99%就报警，通知对象为**云账号报警联系人**。

> **说明** 每次告警的周期为5分钟。

配置一键报警的注意事项如下：

- 一键报警功能开启后，默认开启云监控预置的报警规则，快速建立云监控报警体系监控重要指标，并未全面覆盖所有监控指标。
- 一键报警功能开启后，对应的报警规则作用于您选中产品的当前实例及后续新生成的实例。
- 一键报警功能支持您对预置的报警规则进行修改、禁用和删除操作。

配置一键报警的步骤如下：

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 一键报警**，进入**一键报警**页面。
3. 在**一键报警列表**，单击**交互式分析Hologres**的**一键报警**开关，开启一键报警。

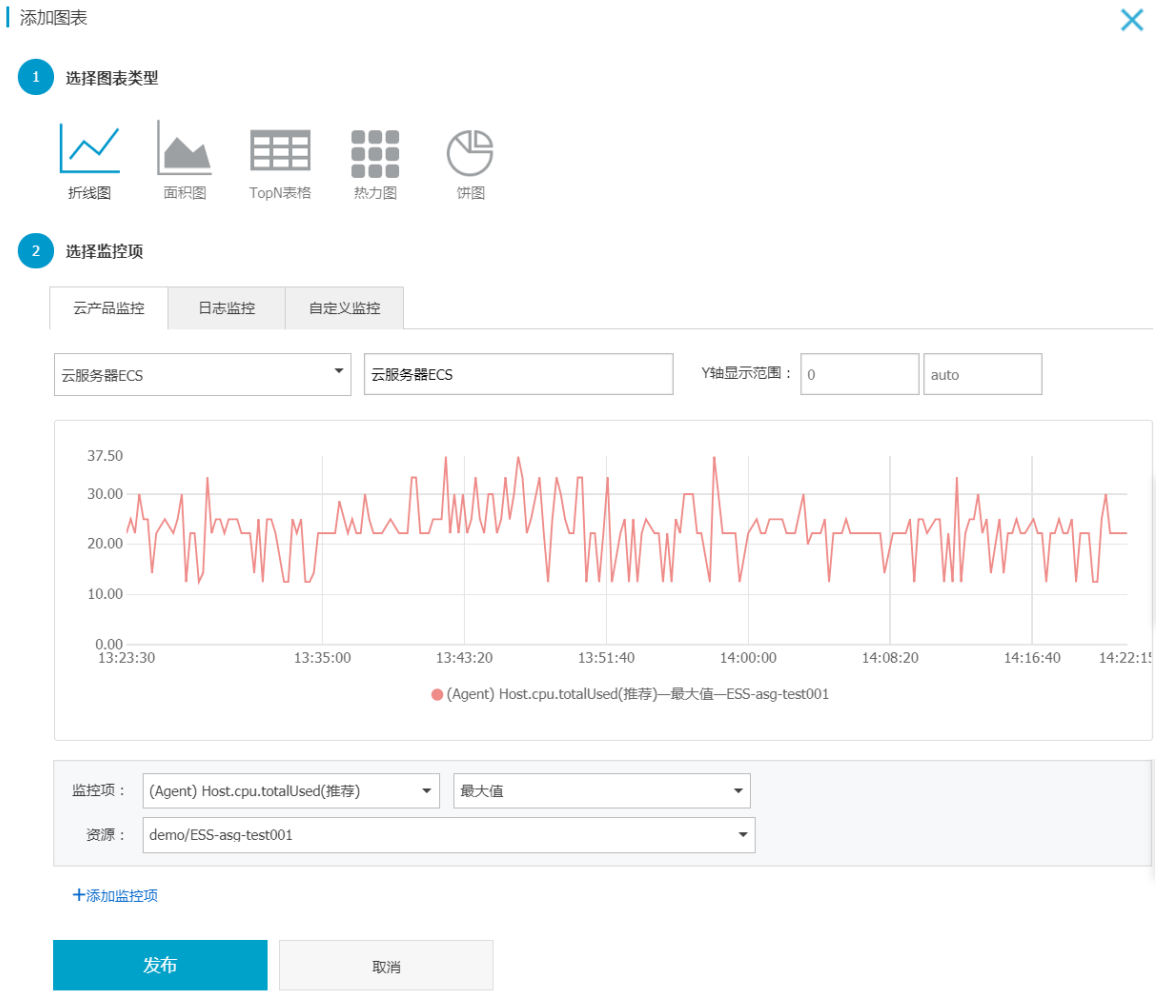
4. 单击**交互式分析Hologres**左侧的图标，查看云监控为您自动生成的报警规则。

5. （可选）您可以单击操作栏中的**禁用**、**修改**、**删除**，禁用、修改、删除对应的报警规则。

使用自定义大盘定制监控指标

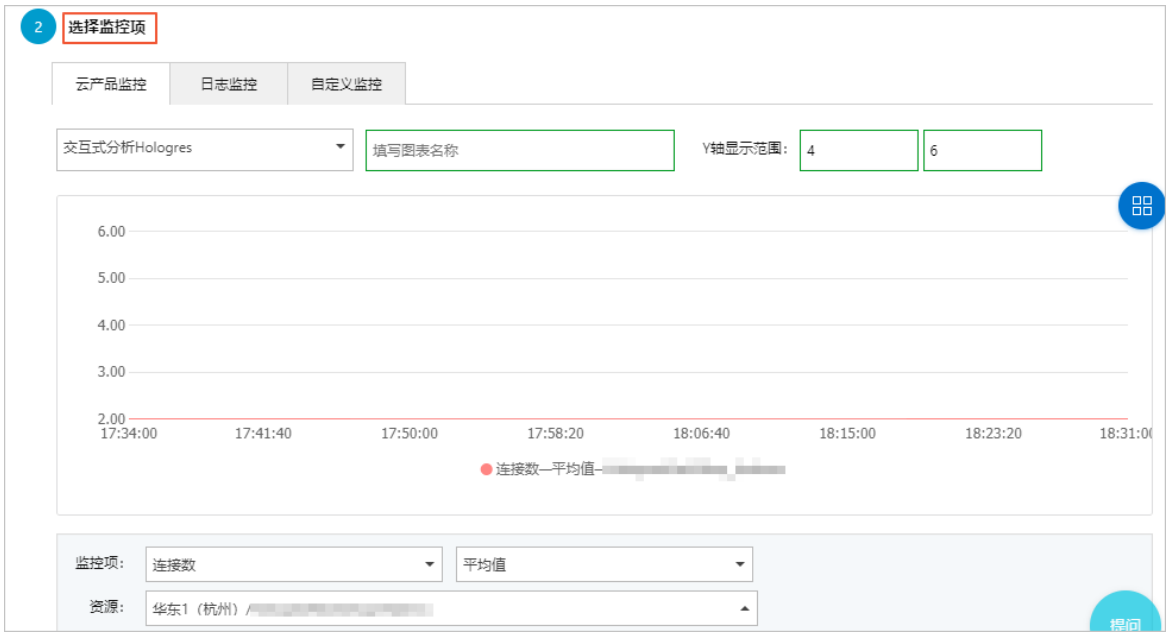
云监控已为您初始化了用户维度的Hologres监控大盘，如果您需要查看其他数据，可以使用自定义大盘定制监控指标。自定义监控大盘的步骤如下：

1. 登录云监控控制台。
2. 在左侧导航栏，单击**Dashboard > 自定义大盘**，进入**当前监控大盘**页面。
3. 在顶部菜单栏右侧，单击**创建监控大盘**。
4. 在**创建视图组**对话框输入新建的监控大盘名称。
5. 单击**创建**，系统自动跳转至新建的监控大盘页面。
6. 单击右上角的**添加图表**，进入**添加图表**页面。



7. 在**选择图表类型**区域，选择折线图、面积图、TopN表格、热力图或饼图中任意一种图表类型。
8. 在**选择监控项**区域，配置**云产品监控**、**日志监控**及**自定义监控**。

   以**云产品监控**为例，选择云产品为**交互式分析Hologres**并为图表命名，配置**监控项**和**资源**。



Hologres支持的实例监控项如下表所示。

监控项	Code	描述
CPU水位	cpu_usage	实例CPU的使用百分比。
内存水位	memory_usage	实例内存的使用百分比。
存储已用容量	storage_usage	实例存储的使用量。
存储水位	storage_usage_percent	实例存储已用容量与购买存储规格的比例。未购买存储规格无数据。
Query QPS明细	query_qps	每秒SQL语句执行个数，按cmdType（Query类型）展示。 cmdType包含如下。 ○ SELECT语句。 ○ INSERT语句。 ○ UPDATE语句。 ○ DELETE语句。

监控项	Code	描述
Query响应时间明细	query_latency	Query响应时间（Latency）明细，按cmdType（Query类型）展示。 cmdType包含如下。 ○ SELECT语句。 ○ INSERT语句。 ○ UPDATE语句。 ○ DELETE语句。
Query P99 延迟	query_latency_p99	Query响应时间P99明细，按cmdType（Query类型）展示。
连接数	connection_count	实例的SQL连接数明细，按数据库名称展开。连接数指标未展示在主图，需在自定义监控配置。
SQL总连接数	connection_num	实例的SQL总连接数。
连接数使用率	connection_usage	单个节点占用连接数与总连接数的比值。
每秒失败查询数	failed_query_qps	实例每秒失败SQL查询数量。
实时写入RPS	dml_rps	实例每秒实时导入数据条数。
Select语句响应时间	无	Select Query查询的延迟时间。
Insert语句响应时间	无	Insert Query查询的延迟时间。
Update语句响应时间	无	Update Query查询的延迟时间。
Delete语句响应时间	无	Delete Query查询的延迟时间。

告警最佳实践

如何为业务设置合适的告警，达到不频繁地对不重要事件报警，又能及时感知到重要异常事件发生的目的，实际上取决于您的具体业务情况，以及Hologres在您的整体架构中所处的位置，请根据具体情况进行设置。不过有一些通用性的指标，建议您对其设置告警，并且设置合适的告警，如下所示。

- CPU水位

  CPU水位反映了Hologres的资源是否存在瓶颈，也反映了您的资源使用是否充分。CPU水位告警的核心逻辑在于，设置规则检测出CPU水位持续100%，即CPU持续打满的情况。因为这种情况，意味着当前实例规模的资源已充分利用，难以支撑业务数据量、查询量、计算量等的增长，需要考虑扩容等手段。

  告警规则建议

  - 不建议出现一次CPU使用率达到100%就告警。

    如果CPU水位持续到达100%一段时间，随后降低维持在中等或较低水平，那么一般是那段时间在做大的计算，例如大规模数据的写入，或者大规模数据的查询，因此不建议出现一次CPU使用率达到100%就告警，这样可能会产生较多误报。

- 建议设置为“CPU水位连续3次>=99% Info（每次间隔5分钟）”类似的告警规则。

  设置为“CPU水位连续3次>=99% Info（每次间隔5分钟）”的告警规则，即15分钟内检测3次出现CPU使用率100%，产生Info级别的告警。此告警规则可以有效避免误报，根据具体情况的不同，您可以降低间隔时间，或者提高检测次数。

- 内存水位

  内存水位的逻辑与CPU水位大体相同。不同于CPU水位具有不产生计算则几乎为0的特点，内存水位在没有计算时，并不会为0，有时甚至会比较高。Hologres把计算内存分为了三块，一是预留给计算的内存，约占30%，没有计算时，这块几乎为0；二是数据缓存，在诸多情况下，数据缓存可以极大地减少I/O读取，提高SQL的计算效率，约占30%；三是实例中所有表、索引等元数据缓存，以及表在内存中的句柄和缓冲区等，约占30%。

  告警规则建议

  - 不建议内存水位设置过低的阈值。

    没有计算时只能保证前30%是空的，还有60%甚至以上的内存，可能是被占据的，因此体现出内存水位不为0甚至较高。有可能超过60%被占据，是因为元数据优先放入内存，如果元数据过多，加上数据缓存的30%，无计算时内存使用会超过60%。

  - 建议设置为“内存水位连续3次>=90% Warning（每次间隔5分钟）”类似的告警规则。

    设置为内存水位连续3次>=90% Warning（每次间隔5分钟）”的告警规则，即15分钟内检测到3次出现内存使用率超过90%，产生Warning级别的告警。

- 连接数使用率

  对连接使用率进行告警，是为了防止连接数打满，而造成无法连接的异常影响业务。一般连接数在总连接数的95%以下都算安全。因此可以设置告警规则为：连接数使用率（Info）连续3次>=95就报警。

- P99延迟

  SQL语句从提交到返回的过程中，Hologres引擎处理99%查询语句的时间。响应时间异常性变大反映出系统可能有慢Query的趋势，可能会影响下游业务处理。平均响应时间很难有一个标准，需要根据您的实际查询使用模式而定。P99延迟代表了99% Query跑完的时间，相对平均而言有特定的意义。

  例如您是持续服务型应用，平常的正常延迟都在1s以内，那么响应时间突然到5s或10s，就不太正常，需要告警出来，偶尔到1.2s，1.5s，则正常。

  例如您是分析性应用，根据不同业务人员的需求，查询的大小、响应时间都不尽相同，那么就需要根据自己的场景进行设置。

  另外要注意的是，当实例负载很高的时候，SQL运行时间波动可能越大，这种情况下就容易产生告警。解决方法是适当调整阈值，或者扩增实例规格以维持负载平稳。

- Query QPS

  在您的业务中，有持续的SQL语句执行时，往往有持续稳定的QPS（Query Per Second）指标。当QPS突然降到很低，或者为0时，如果不是业务有意停止，那么可能意味着系统有异常。如果业务QPS持续稳定在某个值A，那么建议告警阈值设置为A的80%、50%或10%等，即告警规则设置为：QPS连续3次<=（A*0.8） Warning（每次间隔5分钟）。

  SQL语句的QPS包含如下。

类型	说明
（推荐）Query QPS明细	实例每秒SQL语句执行数。
SELECT语句QPS	实例每秒SELECT语句数。

类型	说明
INSERT语句QPS	实例每秒INSERT语句数。
UPDATE语句QPS	实例每秒UPDATE语句数。
DELETE语句QPS	实例每秒DELETE语句数。

- 实时写入RPS

  在您的业务中，有持续的外部数据导入（通过Flink/数据集成等等）时，往往有持续的实时导入RPS（Record Per Second）指标。当RPS突然降到很低，或者为0时，如果不是作业有意停止，那么可能意味着系统有异常。例如，设置告警为：RPS(cmdType=sdk)连续1次<=10 Warning。

告警后处理最佳实践

在告警之后，请根据以下类别对可能存在的异常情况进行处理，Hologres针对具体的情况推荐的处理方法如下。

- 出现连接数使用率告警

  处理方法： 对连接数进行诊断和管理，详情请参见连接数管理进行适当处理。

- 出现SQL总连接数高的告警

  处理方法：对实例的连接进行诊断和管理，详情请参见连接数管理。

- 出现内存水位高的告警

  具体分为如下两种情况。

  - 情景一：无计算但内存水位偏高。

    处理方法：如果您的实例出现没有计算但内存水位高，可能说明您的实例元数据过多，例如表的数量很多；或者是索引设置不合理，例如对高基数的列设置了Dictionary索引或Bitmap索引。

  - 情景二：有计算且内存水位偏高。

    处理方法：如果有计算，且内存水位高，或者SQL发生内存超出报错，这时可以通过检查SQL的执行计划，观察是否合理等手段，改善SQL的内存消耗。

- 出现存储水位高的告警

  处理方法：当出现存储水位告警时，代表您实例存储告急，请适当扩容存储。

- 出现QPS低的告警

  处理方法：

  i. 排查是否存在前台任务手工重启、发布、实例是否正在升级等人为或正常原因造成。

  ii. 排查完毕后，请检查当前前台任务的报错状况，SQL日志报错情况，是否有报错信息，根据具体信息排查。

- 出现平均响应时间长的告警

  处理方法：出现平均响应时间的尖峰，一般情况下，需要具体SQL执行人员排查具体的SQL的执行计划是否符合预期。

- 出现RPS低的告警

  处理方法：

i. 排查是否存在前台任务手工重启、发布、实例是否正在升级等人为或正常原因造成。

ii. 排查完毕后，请检查当前导入任务的健康状况，是否有报错信息，再根据具体信息排查。

8.3.2. Hologres管控台的监控指标

Hologres管理控制台的监控告警页面，为您提供了实例的资源管理服务。您可以在该页面查看当前实例的资源使用及SQL语句的执行情况，及时识别系统报错并处理实例异常。本文为您介绍相关指标的含义。

Hologres管理控制台提供的监控指标如下：

- CPU使用率（%）
- 内存使用率（%）
- 实例存储用量（字节）
- 连接数（个）
- QPS（个/秒）
- Query延迟（毫秒）
- 实时导入RPS（记录/秒）
- IO吞吐（字节）

需要您注意的是：

- 目前Hologres V0.8版本，无连接数（个）指标。
- 目前Hologres V0.9.27以下版本，如果当前实例中表数量超过1000时，由于指标量太大，实时导入RPS（记录/秒）指标可能会存在丢失指标或者指标值过低的情况，影响准确率。

> **说明** 您可以调用云监控的API接口获取Hologres监控指标的监控信息，详情请参见调用方式。

CPU使用率（%）

CPU使用率指实例的CPU综合使用率。

Hologres的计算资源采用预留模式，即使没有执行查询操作，后台也会存在运行的进程，此时CPU使用率不为零属于正常现象。

Hologres可以充分发挥CPU多核并行计算的能力。通常，单个查询可以迅速将CPU使用率提高至100%，这表明计算资源得到了充分利用。

如果CPU使用率长期接近100%，表明实例的负载非常高，此时，CPU资源可能会影响系统的运行。您可以通过分析具体的业务场景，判断资源占用情况：

- 是否存在较大的离线数据导入情况（INSERT），且数据规模还在逐渐增长。
- 是否存在高QPS的查询或写入。
- 是否存在上述场景或场景以外的混合负载。

如果确定是资源不足影响了系统的运行，则可以对实例资源进行扩容，以便承载更大数据量的复杂查询。

内存使用率（%）

内存使用率指实例的内存综合使用率。

Hologres的内存资源采用预留模式，即使没有执行查询操作，也会有部分Meta或Index元数据加载到内存中，该类元数据用于提升计算速度。此时，在不存在查询的情况下，内存使用率可能会达到30%~40%左右，属于正常现象。

如果内存使用率持续升高，甚至接近100%，通常会影响系统的运行，可能会影响实例的稳定性或性能。关于该问题产生的原因、主要影响和解决方法具体如下：

- **产生原因**
  - 表数量增加，数据总量也随之增加，以至于数据规模远大于当前计算规格。由于内存使用率和元数据、索引量为正相关关系，因此，表数量越多、数据量越大、索引越多，都会导致内存使用率升高。
  - 索引设置不合理。例如，设置了过多的Bitmap或Dictionary索引。
- **主要影响**
  - 影响稳定性。当元数据等过大时，会超额占据正常Query可用的内存空间，导致在查询过程中，可能会偶发 `SERVER_INTERNAL_ERROR` 、 `ERPC_ERROR_CONNECTION_CLOSED` 、 `Total memory used by all existing queries exceeded memory limitation` 等报错。
  - 影响性能。当元数据等过大时，会超额占据正常Query可用的缓存空间，从而导致缓存命中减少，Query延迟增加。
- **解决方法**
  - 删除不再查询的数据，以释放占用的内存。
  - 设置合理的索引，可以根据业务场景具体分析，删除不涉及的Bitmap和Dictionary。
  - 扩容，对实例的计算和存储资源进行升配。升配的具体建议如下：
    - 普通场景：允许读磁盘数据的延迟，响应时间RT（return time）要求不严格，1 CU即1core 4G内存，可以支持50 GB~100 GB的数据存储。这种场景下，每32 CU对应的数据存储最高不应超过3 TB。
    - RT要求低的Serving场景：建议查询热点数据全部在内存的缓存中。内存中缓存的比例默认占总内存的30%，即1 CU=1core+4 GB，那么在业务场景中，例如，1.3 GB用于数据缓存，表的元数据也会使用部分数据缓存。例如，低RT要求的场景，热点数据为100 GB，那么缓存至少需要100 GB可用（实际上读取数据并解压后，占用内存不止100 GB），因此需要约320 GB内存以上，从而推算计算资源至少需要96 CU左右。

实例存储用量（字节）

实例存储用量指实例存储的数据占用逻辑磁盘的大小，是所有DB存储用量的总和。

如果您是使用**包年包月**付费模式购买实例，则存储资源的额度使用完后，超出部分会自动转为**按量计费**。请您在超出存储资源后及时对存储进行升级配置，或者合理设置生命周期并及时清理无用数据，避免产生不必要的存储费用。设置表的生命周期详情请参见CREATE TABLE。您也可以通过SQL语句分析各个DB和DB表占用存储的大小，更多内容请参见查看表和DB的存储大小。

连接数（个）

连接数指实例中总的SQL连接数，包括active及idle状态的JDBC或PSQL连接。

实例的连接数通常与实例的规格有关，具体的规格说明，请参见实例规格概述。

如果您遇到如下情况，则说明系统连接数已经达到上限，需要检查您的应用是否存在连接数泄漏的情况并进行连接的释放，具体操作请参见连接数管理：

- 连接数达到甚至超出 `max_connections` 的取值，您可以执行 `show max_connections;` 语句查看当前实例默认的最大连接数。
- 产生 `FATAL: sorry, too many clients already connection limit exceeded for superusers` 报错。
- 产生 `FATAL: remaining connection slots are reserved for non-replication superuser connections` 报错。

QPS（个/秒）

QPS指平均每秒执行SELECT、INSERT、UPDATE或DELETE 4种SQL语句的次数。该指标每20秒上报一次，如果20秒内仅执行了一条SQL语句，则SQL语句的QPS将在该20秒的数据点显示1/20=0.05。

Query延迟（毫秒）

Query延迟指执行SELECT、INSERT、UPDATE或DELETE 4种SQL语句的平均延迟（即响应时间）。您可以通过查看慢Query日志分析Query延迟，更多内容请参见慢Query日志查看与分析。

实时导入RPS（记录/秒）

实时导入RPS指每秒通过SQL语句或SDK方式导入或更新的数据记录条数。

Insert RPS表示使用外部表批量导入、使用COPY语句批量导入或Hologres表间插入数据的导入速率。

Update RPS表示通过执行更新或删除SQL语句，每秒更新或删除的记录条数。

SDK RPS表示通过SDK方式写入Hologres的每秒数据写入或更新的条数。SDK方式包括：

- 可以通过FixedPlan优化的Insert、Update、Delete操作。
- 通过Flink写入。
- 通过Holo Client读写数据。
- Spark的数据写入至Hologres。
- 实时同步DataHub的数据至Hologres。
- 通过数据集成DataX离线写入。
- SQL或JDBC 通过 `insert into values ()` 语句写入。

IO吞吐（字节）

IO吞吐指Hologres实例的IO吞吐量。该指标描述实例的读写数据量，反映实例的读写繁忙程度。您可以从IO层面了解实例的压力情况和负载变化，及时诊断问题。

8.3.3. 自助健康检查常用命令

本文为您介绍Hologres使用过程中自助健康检查常用命令。

表规划检查

- 避免Table Group & Shard过多

  实例在256Core以下规格，建议只使用默认Table Group；256Core及以上规格实例，可以定义2~3个Table Group。数据库总的Table Group数不建议超过3个。Shard数应大于Worker Node数，小于Core数的60%。如果配置了Replication，Shard数应等比例减小，或等比例增加Worker Node计算资源。检查命令如下。

```
-- 查询当前Table Group个数
select  count(distinct tablegroup_name)
from    hologres.hg_table_group_properties;
-- 检查每个TG配置，一个TG不建议超过3000张表(table_num)
select  tablegroup_name as table_group
        ,max(property_value) filter( where property_key='shard_count') as shard_count
        ,max(property_value) filter( where property_key='replica_count') as replica_count
        ,max(property_value) filter( where property_key='is_default_tg') as is_default
        ,max(property_value) filter( where property_key='table_num') as table_num
from    hologres.hg_table_group_properties
group by tablegroup_name;
```

如果Table Group数量超过3个，建议规划为一个核心Table Group，合并多余Table Group；或者一个主Table Group，以及一个面向维表的小Table Group，通过（Resharding）迁移表至新建Table Group操作。

- 检查表数量是否合理

  过多的表，引起小文件多，导致元数据占用内存过多，检查表数量命令如下。

```
-- 检查不同Schema下的内部表数量
select  table_namespace as schema
        ,count(distinct table_name) as total_tables
        ,count(distinct table_name) filter( where property_key='storage_format') as inner
_tables
from    hologres.hg_table_properties
where   table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
group by table_namespace
order by table_namespace;
```

  如果分区表小且多，建议重新建表，将分区表合并为普通表。

- 检查表统计信息是否更新及时

  检查表统计信息命令如下。

```
-- 检索超过一天没有更新统计信息的表
select  schema_name
        ,table_name
        ,total_rows
        ,analyze_timestamp
from    hologres_statistic.hg_table_statistic
where   analyze_timestamp < now() - interval '1 days'
order by schema_name
         ,table_name
limit   200;
```

  如果有统计信息更新不及时的表，且该表有数据更新，请执行 `analyze tablename` 命令，刷新表的统计信息。

- 避免过多资源组

  资源组会限制CPU和内存资源的使用，通常资源组最多设置3个，且保证default资源组分配资源在0.3以上。

  检查资源组的命令如下。

```
SELECT * FROM pg_holo_resource_groups
where property_key='worker_limit';
```

表设计检查

- 有限使用行存表

  行存表使用场景相对有限，主要用在Flink关联维表场景，因此需要避免误用。列出所有行存表命令如下。

```
select  table_namespace as schema
        ,table_name as tables
from    hologres.hg_table_properties
where
        property_key = 'orientation'
and     property_value = 'row'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog');
```

如果误用了行存表，请重新建表，选择列存或者行列共存表。

- Distribution Key应明确设置，且不建议超过2列

  建议每个表都至少设置一列做Distribution Key，并且Distribution Key不建议超过2个，检查命令如下。

```
-- 列出所有Distribution Key超过2列的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as distribution_key
from    hologres.hg_table_properties
where   property_key = 'distribution_key'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     array_length(string_to_array(property_value,','),1) > 2;
-- 列出所有没有设置Distribution Key的表
select  distinct table_namespace as schema
        ,table_name as tables
from    hologres.hg_table_properties a
where property_key='storage_format' and not exists (
                    select  1
                    from    hologres.hg_table_properties b
                    where   b.property_key = 'distribution_key'
                    and     a.table_namespace = b.table_namespace
                    and     a.table_name = b.table_name)
                    and     table_namespace NOT IN ('hologres','hologres_statistic','pg_
catalog'
                )
order by schema
         ,tables;
```

如果Distribution Key列超过两个，请重新建表，选择一到两个列作为Distribution Key。

- Dictionary Encoding不建议超过20列

  仅对低基数列设置Dictionary Encoding，一般建议不超过20列。如果不确定，请选择 `auto encoding` ，避免全部列设置Dictionary Encoding。检查命令如下。

```
--列出所有dictionary_encoding_columns超过20列，并且没有配置为auto encoding的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as dictionary_encoding_columns
from    hologres.hg_table_properties
where   property_key = 'dictionary_encoding_columns'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     array_length(string_to_array(property_value,','),1) > 20
and     property_value not like '%:auto';
```

如果设置Dictionary Encoding列超过20列，请通过 `SET_TABLE_PROPERTY` 或者 `UPDATE_TABLE_PROPERTY`

命令更改dictionary_encoding_columns的配置，详情请参见SET_TABLE_PROPERTY或者ALTER TABLE。

- Bitmap Columns不建议超过30列

  仅对需要等值比较的列设置Bitmap，一个表建议不超过30列设置Bitmap，过多字符串列设置Bitmap会占用额外存储和内存开销。

```
-- 列出所有bitmap_columns超过30个列的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as bitmap_columns
from    hologres.hg_table_properties
where   property_key = 'bitmap_columns'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     array_length(string_to_array(property_value,','),1) > 30;
```

  如果设置Bitmap列超过30列，请通过 SET_TABLE_PROPERTY 或者 UPDATE_TABLE_PROPERTY 命令更改bitmap_columns的配置，详情请参见SET_TABLE_PROPERTY或者ALTER TABLE。

- Clustering Key不建议超过2列

  Clustering Key具备左匹配原则，因此一般不建议超过两个列，否则适用场景减少。检查命令如下。

```
-- 列出所有clustering_key超过2个列的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as clustering_key
from    hologres.hg_table_properties
where   property_key = 'clustering_key'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     array_length(string_to_array(property_value,','),1) > 2;
```

  如果Clustering Key设置超过两列，请重新建表，选择一到两个排序列作为Clustering Key。

- Segment Key最多仅设置一个实时写入时间戳相关列

  Segment Key用于文件分块，建议最多只设置一列，且类型为时间戳或者整型。检查命令如下。

```
-- 列出所有Segment Key大于一列的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as segment_key
from    hologres.hg_table_properties
where   property_key = 'segment_key'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     array_length(string_to_array(property_value,','),1) > 1;
```

  如果Segment Key设置超过一列，请重新建表，选择一个时间戳列作为Segment Key。

- 数据TTL不建议小于7天

  TTL表示一个表数据的回收时间，由于回收是异步进行，不建议TTL小于七天，否则可能会由于回收不及时，造成重复数据。检查命令如下。

```
-- 列出所有time_to_live_in_seconds小于7天的表
select  table_namespace as schema
        ,table_name as tables
        ,property_value as time_to_live_in_seconds
from    hologres.hg_table_properties
where   property_key = 'time_to_live_in_seconds'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
and     property_value::bigint < 604800;
```

如果表的TTL小于七天，请通过 `SET_TABLE_PROPERTY` 命令更改TTL大于七天，详情请参见SET_TABLE_PROPERTY。

- 按需使用Binlog

  Binlog能力强大，但消耗资源也更多，使用Binlog的表写入性能会受到较大影响，行存表Binlog的开销会远小于列存表，因此对于列存表，谨慎开通Binlog能力。检查命令如下。

```
-- 列出所有配置了Binlog的表
select  table_namespace as schema
        ,table_name as tables
from    hologres.hg_table_properties
where   property_key = 'binlog.level'
and     property_value = 'replica'
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
;
-- 列出所有Binlog TTL大于7天的表，建议缩短TTL
select  table_namespace as schema
        ,table_name as tables
        ,property_value as "binlog.ttl"
from    hologres.hg_table_properties
where   property_key = 'binlog.ttl'
and     property_value::bigint > 604800
and     table_namespace NOT IN ('hologres','hologres_statistic','pg_catalog')
;
```

如果Binlog设置不合适，请通过 `SET_TABLE_PROPERTY` 命令调整Binlog级别及TTL时间，详情请参见SET_TABLE_PROPERTY。

- 避免数据倾斜性

  数据分布在不同的Shard中，如果部分Shard的数据明显多余其他Shard，说明数据具有显著的倾斜性，此时应该调整Distribution Key的设计，实现更为均衡的数据分布。检查命令如下。

```
select  hg_shard_id
        ,count(*)
from    table_name
group by hg_shard_id;
```

如果数据明显倾斜，需要通过调整Distribution_key重新导入数据。

运行态检查

- 资源使用检查

  CPU、内存、连接数使用情况，通过云监控分析，详情请参见Hologres管控台的监控指标。

- 查询成功率检查

不同类型Query的占比、成功率、延时、并发同比环比分析命令如下。

```
-- 过去7天各个数据库的DML次数(Select\Insert\Update\Delete)
select datname, query_date, count(*) from hologres.query_log
where query_date > to_char(current_date - interval'7 days','YYYYMMDD')
and command_tag in ('SELECT','INSERT','UPDATE','DELETE')
group by datname, query_date
order by datname, query_date desc;
-- 过去1天各类DML的执行成功情况
select datname, query_date, command_tag,
       count(*) filter( where status='SUCCESS') as SUCCESS,
       count(*) filter( where status='FAILED') as FAILED
from hologres.query_log
where query_date > to_char(current_date - interval'1 days','YYYYMMDD')
and command_tag in ('SELECT','INSERT','UPDATE','DELETE')
group by datname, query_date, command_tag
order by datname, query_date desc;
-- 最近2天成功查询的响应延时分析
select datname, query_date, command_tag, count(*), AVG(duration) as duration
from hologres.query_log
where query_date > to_char(current_date - interval'1 days','YYYYMMDD')
and command_tag in ('SELECT','INSERT','UPDATE','DELETE')
and status = 'SUCCESS'
group by datname, query_date, command_tag
order by datname, query_date desc;
```

- 慢查询检查

  过去一天耗时最长的重点慢查询检查命令如下。

```
-- 查询过去1天耗时最长的重点慢查询
select status as "状态",
       duration as "耗时(ms)",
       optimization_cost as "优化耗时(ms)",
       start_query_cost as "启动耗时(ms)",
       get_next_cost as "执行耗时(ms)",
       duration-optimization_cost-start_query_cost-get_next_cost as "其他耗时(ms)",
       query_id as "QueryID",
       query
 from hologres.hg_query_log
 where query_start > current_date - interval '1 days'
  and command_tag in ('SELECT')
  and duration > 1000
 order by duration desc,
          start_query_cost desc,
          optimization_cost,
          get_next_cost desc,
          duration-optimization_cost-start_query_cost-get_next_cost desc
 limit 200;
```

消耗资源最多查询的检查命令如下。

```
-- 查询最近一天消耗比较高的Query
select status as "状态",
       duration as "耗时(ms)",
       query_start as "开始时间",
       (read_bytes/1048576)::text || ' MB' as "读取量",
       (memory_bytes/1048576)::text || ' MB' as "内存",
       (shuffle_bytes/1048576)::text || ' MB' as "Shuffle",
       (cpu_time_ms/1000)::text || ' s' as "CPU时间",
       physical_reads as "读盘",
       query_id as "QueryID",
       query
 from hologres.hg_query_log
 where query_start > current_date - interval'1 days'
 and command_tag in ('SELECT','INSERT','UPDATE','DELETE')
 and duration > 1000
 order by duration desc,
          read_bytes desc,
          shuffle_bytes desc,
          memory_bytes desc,
          cpu_time_ms desc,
          physical_reads desc
 limit 500;
```

8.4. 连接与SQL

8.4.1. 慢Query日志查看与分析

慢Query的查询与分析可以帮助您对系统中发生的慢Query或失败Query进行诊断、分析和采取优化措施。本文将介绍在Hologres中如何查看慢Query日志并分析。

使用限制

在Hologres中查询慢Query日志，具体限制如下：

- 该功能仅Hologres V0.10及以上版本支持，请在Hologres管理控制台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- 由于慢Query日志存储在远端服务器，非本地存储，为了保证系统稳定性，避免流量超载，单次查询最多从服务器端传输100万条Query日志，超出数量提示number of read rows (XXX) exceeds limit (1000000)。
- 目前慢Query日志支持记录和查询执行慢的DML（包括INSERT、SELECT、UPDATE、DELETE等SQL语句），以及所有的DDL语句（包括CREATE、ALTER、DROP、GRANT等SQL语句）。
- 查询返回的memory_bytes、shuffle_bytes和cpu_time_ms三个字段不是精准值，只有部分相对意义，没有绝对意义。
- Hologres V0.10版本的FAILED Query，不显示内存、读盘、读取量、CPU时间、query_stats等运行时的统计信息。
- 慢Query日志默认保留一个月的数据，但是HoloWeb仅支持查最近7天的数据，如果需要查询一个月内的数据请使用SQL进行查询。

查看query_log表

存放慢Query查询日志的系统表为hologres.hg_query_log，将实例升级到V0.10版本后，将会默认采集大于100ms的慢DML Query，以及所有的DDL（查询时默认只展示大于 `1s` 的Query，若需要展示 `100ms~1s` 内的Query，请按照下面的配置项修改展示时间）。其主要包含的字段信息如下：

Column	Type	Description
usename	text	查询的用户名
status	text	查询的最终状态，成功或失败
query_id	text	查询ID
datname	text	查询的数据库名
command_tag	text	查询的类型(INSERT/SELECT/UPDATE/DELETE，以及EXPLAIN/ANALYZE/BEGIN/COMMIT/ROLLBACK/CREATE TABLE/DROP TABLE/ALTER TABLE/COMMENT/CALL/GRANT等)
duration	integer	查询的耗时(ms)
message	text	报错的信息
query_start	timestamp with time zone	查询开始时间
query_date	text	查询开始日期
query	text	查询的文本内容
result_rows	bigint	查询返回的行数
result_bytes	bigint	查询返回的字节数
read_rows	bigint	查询读取的行数
read_bytes	bigint	查询读取的字节数
affected_rows	bigint	DML影响的行数
affected_bytes	bigint	DML影响的字节数
memory_bytes	bigint	单节点峰值内存消耗（非精确）
shuffle_bytes	bigint	数据Shuffle估计字节数（非精确）
cpu_time_ms	bigint	总的CPU时间(毫秒，非精确)
physical_reads	bigint	物理读的次数
pid	integer	查询服务进程ID
application_name	text	查询应用类型
engine_type	text[]	查询用到的引擎
client_addr	text	查询的来源地址
table_write	text	SQL改写的表
table_read	text[]	SQL读取的表
session_id	text	查询Session ID
session_start	timestamp with time zone	查询Session开启时间
command_id	text	命令/语句编号ID
optimization_cost	integer	查询执行计划的耗时
start_query_cost	integer	查询启动耗时
get_next_cost	integer	查询执行耗时
extended_cost	text	查询的其他详细耗时
plan	text	查询对应的plan
statistics	text	查询对应的执行时统计信息
visualization_info	text	查询Plan可视化信息
query_detail	text	查询的其他扩展信息(JSON格式存储)
query_extinfo	text[]	查询的其他扩展信息(ARRAY格式存储)

授予查看权限

慢Query日志需要有一定的权限才能查看，其权限规则和授权方式说明如下：

- **查看实例所有DB的慢Query日志。**

- 授予用户Superuser权限。

  Superuser账号可以查看实例所有DB的全部慢Query日志，给用户授予Superuser用户的权限，使用户有权限查看实例所有DB的慢Query日志。

  ```
  ALTER USER "云账号ID" SUPERUSER;--将"云账号ID"替换为实际用户名。如果是RAM用户，账号ID前需要添加"p4_"。
  ```

- 将用户添加到pg_read_all_stats用户组。

  除Superuser外，Hologres还支持pg_read_all_stats用户组查看所有DB慢Query日志，普通用户如果需要查看所有日志，可以联系Superuser授权加入该用户组。授权方式如下：

  ```
  GRANT pg_read_all_stats TO "云账号ID";--专家权限模型授权
  CALL spm_grant('pg_read_all_stats', '云账号ID');  -- SPM权限模型
  CALL slpm_grant('pg_read_all_stats', '云账号ID'); -- SLPM权限模型
  ```

- **查看本DB的慢Query日志。**

  开启简单权限模型（SPM）或基于Schema级别的简单权限模型（SLPM），将用户加入db_admin用户组，db_admin角色可以查看本DB的query日志。

  ```
  CALL spm_grant('<db_name>_admin', '云账号ID');  -- SPM权限模型
  CALL slpm_grant('<db_name>.admin', '云账号ID'); -- SLPM权限模型
  ```

- **普通用户只能查询当前账号对应DB下自己执行的慢Query日志。**

HoloWeb可视化查看慢Query

您可以通过HoloWeb可视化查看慢Query日志。

> **说明**
> - 当前HoloWeb支持查看最多七天的历史慢Query日志。
> - 仅支持Superuser账号查看，普通有权限的账号请使用SQL命令查看。

1. 登录HoloWeb控制台，详情请参见连接HoloWeb。
2. 单击顶部导航栏的**诊断与优化**。
3. 在左侧导航栏单击**历史慢Query**。
4. 在**历史慢Query**页面顶部，编辑查询条件。

   参数说明如下表。

参数	是否必选	说明
实例名	是	需要查询慢Query的实例名称，默认是当前登录的实例。
数据库	否	需要查询慢query的数据库名称。需要在权限范围内才能查看慢Query日志，否则只能查看自己账号的相关日志。

参数	是否必选	说明
表名	否	根据表名查看当前表相关的慢Query。需要在权限范围内才能查看慢Query日志，否则只能查看自己账号的相关日志。
用户	否	根据用户云账号搜索相关的慢Query。
限制行数	是	慢Query日志返回的行数，最多可展示2000条慢Query。
运行时长	否	SQL的运行时长，默认采集大于1秒的Query。
图维度	否	可选择慢Query和失败Query，用于限制Query趋势分析图的展示纬度。
Query	否	搜索SQL，支持%模糊匹配表名。
Type	否	执行的Query类型，包括DDL以及DML等。
时间范围	是	慢Query的时间范围，默认选择近十分钟，最多只能选择七天的数据。
Query ID	否	Query所对应的ID。

5. 单击**查询**，在**Query趋势分析**和**Query列表**区域展示查询结果。
   - **Query趋势分析**

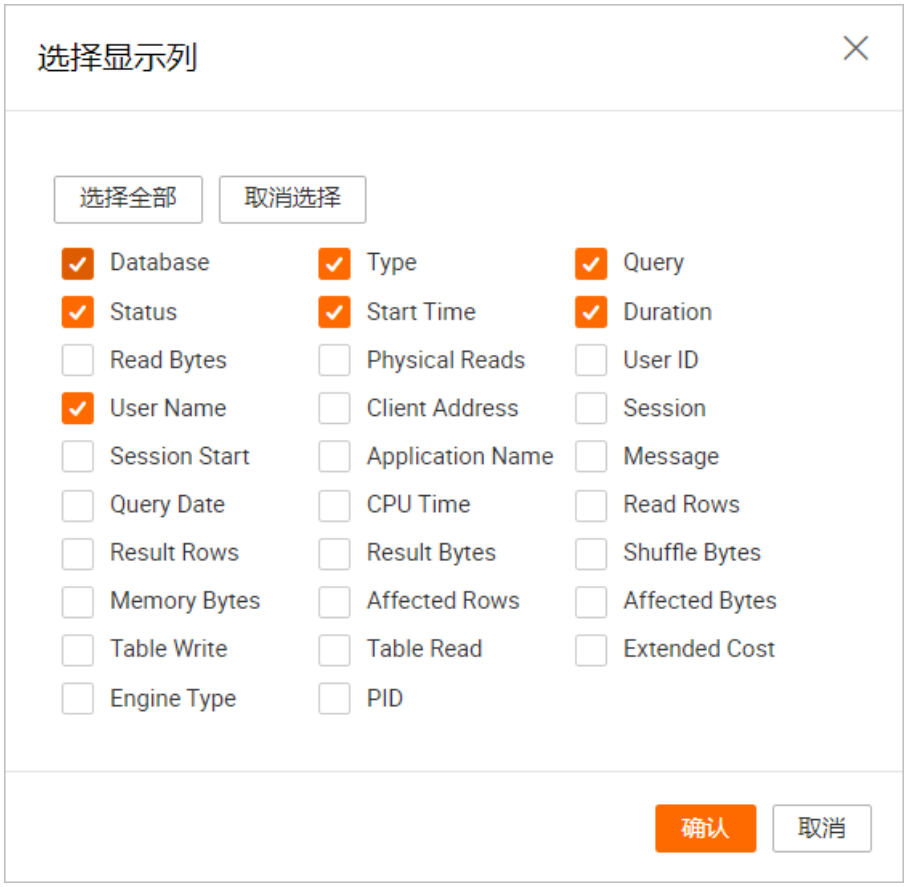
     Query趋势分析通过可视化的方式展示近期慢Query、失败Query发生的趋势，可以重点监控慢Query发生频率较高的阶段，更好的定位问题高发期并集中解决问题。
   - **Query列表**

     Query列表可视化展示慢Query、失败Query的详细信息。

参数	说明
ID	执行SQL的用户云账号ID。
Database	Query所属数据库名称。
User Name	执行SQL的用户名。
Type	SQL的操作类型。
Query	单击可查看Query详情。

参数	说明
Status	Query的状态。
Start Time	Query开始执行时间。
Duration	SQL运行总耗时，包括优化器执行时间、开始执行Query的时间和返回Query结果的时间。
操作	■ **详情**：展示该Query的详细信息，可为优化Query提供一定的帮助。 ■ ：可为当前Query直接打开一个新的SQL编辑器。

也可以单击**自定义列**选择Query列表需要展示的列信息。



诊断Query

您可以通过查询hologres.hg_query_log表，获取慢Query日志。如下将为您介绍进行Query日志诊断常用的固定SQL语句。主要包括：

- 查询query_log总查询数
- 查询每个用户的慢Query情况
- 查询某个慢Query的具体信息

- 查询最近10分钟消耗比较高的Query
- 查询最近10分钟Query各阶段耗时比较高的Query
- 查询最先失败的Query
- 查询query_log总查询数（默认为近1个月内的数据）。

```
select count(*) from hologres.hg_query_log;
```

执行结果具体如下，表示近1个月内总共执行了44个耗时大于指定阈值的Query。

```
count
-------
    44
(1 row)
```

- 查询每个用户的慢Query个数。

```
select usename as "用户",
       count(1) as "Query个数"
from hologres.hg_query_log
group by usename
order by count(1) desc;
```

执行结果具体如下，其中 `count(1)` 的单位是个数。

```
用户                    | Query个数
------------------------+-----
 1111111111111111       |  27
 2222222222222222       |  11
 3333333333333333       |   4
 4444444444444444       |   2
(4 rows)
```

- 查询某个慢Query的具体信息。

```
select * from hologres.hg_query_log where query_id = '13001450118416xxxx';
```

执行结果具体如下，更多关于各参数的说明，请参见查看query_log表。

```
       usename      | status  |      query_id      | datname| command_tag | duration | me
ssage |      query_start       | query_date |
query                                                | result_rows | result_bytes | rea
d_rows |read_bytes | affected_rows | affected_bytes | memory_bytes | shuffle_bytes | cpu_
time_ms | physical_reads |   pid   | application_name | engine_type |  client_addr  | tab
le_write | table_read |   session_id   |     session_start      | command_id | optimizati
on_cost | start_query_cost | get_next_cost | extended_cost | plan | statistics | visualiz
ation_info | query_detail | query_extinfo
-----------------------+---------+--------------------+---------+-------------+----------
+---------+-----------------------+------------+-----------------------------------------
-----------------------------------------------------------------+-------------+---------
-----+-----------+------------+---------------+---------------+--------------+----------
-----+-------------+---------------+---------+-----------------+-------------+---------
------+-------------+------------+-----------------+-----------------------+------------
+-------------------+-----------------+---------------+---------------+------+----------
--+--------------------+-------------+---------------
 p4_11111111111xxxx | SUCCESS | 13001450118416xxxx | dbname | SELECT      |      149 |
| 2021-03-30 23:45:01+08 | 20210330   | explain analyze select  * from tablename where us
er_id = '20210330010xxxx' limit 1000;             |        1000 |       417172 |
0 |         0 |            -1 |             -1 |     26731376 |     476603616 |      3216
26 |              0 | 1984913 | psql             | {HQE}      | 33.41.xxx.xxx |
|            | 6063475a.1e4991 | 2021-03-30 23:44:26+08 | 0          |                58
|              22 |            67 |               |      |            |
|             |
(1 row)
```

- 查询最近某个时间段（如10分钟）消耗比较高的Query。您也可以根据业务需求修改具体时间，查询目标时间段消耗比较高的Query。

```
select status as "状态",
       duration as "耗时(ms)",
       query_start as "开始时间",
       (read_bytes/1048576)::text || ' MB' as "读取量",
       (memory_bytes/1048576)::text || ' MB' as "内存",
       (shuffle_bytes/1048576)::text || ' MB' as "Shuffle",
       (cpu_time_ms/1000)::text || ' s' as "CPU时间",
       physical_reads as "读盘",
       query_id as "QueryID",
       query::char(30)
 from hologres.hg_query_log
 where query_start >= now() - interval '10 min'
 order by duration desc,
          read_bytes desc,
          shuffle_bytes desc,
          memory_bytes desc,
          cpu_time_ms desc,
          physical_reads desc
 limit 100;
```

执行结果具体如下：

```
状态     | 耗时(ms) |     开始时间              | 读取量 | 内存   | Shuffle | CPU时间 | 读盘 |
QueryID       |             query
---------+---------+-------------------------+--------+-------+---------+---------+------+
--------------------+--------------------------------
 SUCCESS |  149    | 2021-03-30 23:45:01+08 | 0 MB   | 25 MB | 454 MB  | 321 s   |    0 |
13001450118416xxxx | explain analyze select  * from
 SUCCESS |  137    | 2021-03-30 23:49:18+08 | 247 MB | 21 MB | 213 MB  | 803 s   | 7771 |
13001491818416xxxx | explain analyze select  * from
 FAILED  |   53    | 2021-03-30 23:48:43+08 | 0 MB   | 0 MB  | 0 MB    | 0 s     |    0 |
13001484318416xxxx | select ds::bigint / 0 from pub
(3 rows)
```

- 查询最近时间段（如最近10分钟）Query各阶段耗时比较高的Query。您也可以根据业务需求修改具体时间，查询目标时间段Query各阶段耗时比较高的Query。

```
select status as "状态",
       duration as "耗时(ms)",
       optimization_cost as "优化耗时(ms)",
       start_query_cost as "启动耗时(ms)",
       get_next_cost as "执行耗时(ms)",
       duration-optimization_cost-start_query_cost-get_next_cost as "其他耗时(ms)",
       query_id as "QueryID",
       query::char(30)
 from hologres.hg_query_log
 where query_start >= now() - interval '10 min'
 order by duration desc,
          start_query_cost desc,
          optimization_cost,
          get_next_cost desc,
          duration-optimization_cost-start_query_cost-get_next_cost desc
 limit 100;
```

执行结果具体如下：

```
状态     | 耗时(ms) | 优化耗时(ms) | 启动耗时(ms) | 执行耗时(ms) | 其他耗时(ms) |      QueryID
|             query
---------+----------+--------------+--------------+--------------+--------------+--------
------------+--------------------------------
 SUCCESS |     4572 |          521 |          320 |         3726 |            5 | 6000260
625679xxxx  | -- /* user: wang ip: xxx.xx.x
 SUCCESS |     1490 |          538 |           98 |          846 |            8 | 1200025
0867886xxxx | -- /* user: lisa ip: xxx.xx.x
 SUCCESS |     1230 |          502 |           95 |          625 |            8 | 2600051
2070295xxxx | -- /* user: zhang ip: xxx.xx.
(3 rows)
```

- 查询最先失败的Query。

```
select status as "状态",
       regexp_replace(message, '\n', ' ')::char(150) as "报错信息",
       duration as "耗时(ms)",
       query_start as "开始时间",
       query_id as "QueryID",
       query::char(100) as "Query"
 from hologres.hg_query_log
 where query_start between '2021-03-25 17:00:00'::timestamptz
                   and '2021-03-25 17:42:00'::timestamptz + interval '2 min'
   and status = 'FAILED'
 order by query_start asc
 limit 100;
```

执行结果具体如下：

```
状态     |                                                                  报错信息
 | 耗时(ms)  |          开始时间          |       QueryID       | Query
--------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------+--------+--------
----------------+-------------------+-------
 FAILED | Query:[1070285448673xxxx] code: kActorInvokeError msg: "[holo_query_executor.cc
:330 operator()] HGERR_code XX000 HGERR_msge internal error: status { c |  1460 | 2021-03
-25 17:28:54+08 | 1070285448673xxxx | S...
 FAILED | Query:[1016285560553xxxx] code: kActorInvokeError msg: "[holo_query_executor.cc
:330 operator()] HGERR_code XX000 HGERR_msge internal error: status { c |   131 | 2021-03
-25 17:28:55+08 | 1016285560553xxxx | S...
(2 rows)
```

配置项

为了实现针对性地记录目标Query，Hologres有多个配置项可供选择。主要包括：

- log_min_duration_statement
- log_min_duration_query_stats
- log_min_duration_query_plan

- **log_min_duration_statement**
  - 函数说明

    该配置项记录Query耗时，系统默认记录大于100ms的Query，查询结果仅展示1s及以上的Query，也可以通过修改该配置项，修改默认展示时间（展示时间最小可设置为100ms）。打开该配置项可以方便地跟踪需要优化的查询。如果某个语句的执行时间大于或者等于该毫秒数，慢Query日志会记录该语句、执行时间以及其它信息。该参数取值设置为-1则表示不记录任何查询。

    说明 只支持超级用户（Superuser）修改该配置项。

- 使用示例
  - 设置当前DB记录大于10s的Query，需要Superuser设置。

    ```
    ALTER DATABASE dbname SET log_min_duration_statement = '10s';
    ```

  - 设置当前Session记录大于10s的Query，普通用户可以执行。

    ```
    SET log_min_duration_statement = '10s';
    ```

- log_min_duration_query_stats
  - 函数说明

    该配置项记录Query执行时统计信息，系统默认将记录大于10s的Query执行时的统计信息，打开这个选项主要用来跟踪需要优化的查询。如果某个语句的执行时间大于或者等于该数值，慢Query日志会记录该语句执行时的统计信息，包括Query在各个执行环节的执行信息。该参数取值设置为-1则表示关闭对query stats信息的记录。该信息存储量较高，记录较多时会降低查询分析慢Query日志的速度。因此，可以在排查具体问题时设置小于10s，否则不建议调整为更小的值。

    > **说明** 支持普通用户根据业务情况修改该配置项。

  - 使用示例
    - 设置当前DB记录大于10s的Query执行时的统计信息，需要Superuser设置。

      ```
      ALTER DATABASE dbname SET log_min_duration_query_stats = '10s';
      ```

    - 设置当前Session记录大于10s的Query执行时的统计信息，普通用户可以执行。

      ```
      SET log_min_duration_query_stats = '10s';
      ```

- log_min_duration_query_plan
  - 函数说明

    该配置项记录Query的执行计划信息，系统将会默认展示大于等于10s的慢Query日志的执行计划。如果某个语句的执行时间大于或者等于该数值，慢Query日志会记录该语句的执行计划。执行计划一般情况下可以即时地通过explain得到，无需记录。该参数取值设置为-1则表示关闭对Query执行计划的记录。

    > **说明** 支持普通用户根据业务情况修改该配置项。

  - 使用示例
    - 设置当前DB记录大于10s的Query日志的执行计划，需要Superuser设置。

      ```
      ALTER DATABASE dbname SET log_min_duration_query_plan = '10s';
      ```

    - 设置当前Session记录大于10s的Query日志的执行计划，普通用户可以执行。

      ```
      SET log_min_duration_query_plan = '10s';
      ```

常见问题

- 问题现象：

  在Hologres V1.1版本中，查看慢Query日志，无法显示查询行数、返回行数等信息。

- 可能原因：

慢Query日志采集不完整。

- 解决方法：

  在Hologres V1.1.36至V1.1.49版本可以通过如下GUC参数使其显示完整。在HologresV1.1.49及以上版本可直接显示。

  > **说明** 如果您的Hologres实例版本低于 V1.1.36，请您提交工单升级实例。

```
--（建议）数据库级别，一个db设置一次即可。
alter databse <db_name> set hg_experimental_force_sync_collect_execution_statistics = on;
--session级别
set hg_experimental_force_sync_collect_execution_statistics = on;
```

  db_name为数据库名称。

8.4.2. Query管理

本文将为您介绍如何对实例中的Query进行诊断和管理。

Query管理概述

Hologres兼容PostgreSQL，可以通过查询pg_stat_activity视图信息来查看实例Query的运行信息，以达到分析和诊断运行SQL的目的。具体涉及的操作内容如下所示：

- HoloWeb可视化活跃Query管理：通过HoloWeb可视化查看和管理活跃Query。
- 查看SQL运行信息：查看SQL运行信息，更好的管理SQL语句。
- 修改活跃Query超时时间：修改活跃Query运行超时时间，防止引发死锁。
- 修改空闲Query超时时间：修改空闲Query运行超时时间，防止引发死锁。
- 查询慢Query日志：通过慢Query的查询可以对慢Query或失败Query进行诊断、分析和采取优化措施。

查询pg_stat_activity视图信息

pg_stat_activity是一个非常有用的视图，可以分析排查当前运行的SQL任务以及一些异常问题。您可以执行如下命令查看Query的运行信息。pg_stat_activity不表示所有的网络连接，部分后台管理进程不占用连接，也会出现在pg_stat_activity表中。

```
select * from pg_stat_activity ;
```

pg_stat_activity视图的参数说明如下所示：

字段	描述
datid	Hologres后端连接到的数据库的OID。
datname	Hologres后端连接到的数据库的名称。
pid	Hologres后端的进程ID。
usesysid	登录到Hologres后端的用户OID。

字段	描述
usename	当前连接的用户名。 `holo_admin`是Hologres内置的服务账号，运行结果显示为PSQL，该连接是必要的。如果您的连接数未超出当前实例的最大限度，不需要针对此连接进行优化。连接数查询方式，请参见通过SQL查询连接信息。
application_name	客户端的应用类型。
client_addr	客户端的IP地址。 显示的IP地址可能是被解析过的，不保证一定是源端IP。
client_hostname	客户端的主机名。
client_port	客户端的端口。
backend_start	后台进程开始的时间。
xact_start	该进程的当前事务被启动的时间。 • 如果没有活动事务，则为空。 • 如果当前查询是该进程的第一个事务，这一列等于query_start。
query_start	当前活动查询开始的时间，如果当前连接状态不是active，取值为上一个查询开始的时间。
state_change	连接的状态（state）上一次被改变的时间。
wait_event_type	后端正在等待的事件类型，如果不存在则为NULL。可能的取值有： • LWLock：后端正在等待一个轻量级锁。 • Lock：后端正在等待一个重量级锁。`wait_event`将标识等待的锁的类型。 • BufferPin：服务器进程正在等待访问一个数据缓冲区，而此时没有其他进程正在检查该缓冲区。 • Activity：服务器进程处于闲置状态。被用于在其主处理循环中等待活动的系统进程。 • Extension：服务器进程正在一个扩展模块中等待活动。 • Client：服务器进程正在等待来自用户应用的某种查询，并且该服务器预期某种与其内部处理无关的事情发生。 • PC：服务器进程正在等待来自服务器中另一个进程的某种活动。 • Timeout：服务器进程正在等待一次超时发生。 • IO：服务器进程正在等待一次IO完成。
wait_event	如果后端当前正在等待，则是等待事件的名称，否则为 NULL。

字段	描述
state	表示连接的状态。常见的状态如下： • active：活跃。 • idle：空闲。 • idle in transaction：长事务中的空闲状态。 • idle in transaction（Aborted）：已失败事务中的空闲状态。 • \N: 状态为空，表示非用户连接的进程，一般属于系统后台的维护进程，可以忽略。
backend_xid	Hologres后端的顶层事务标识符。
backend_xmin	当前后端的xmin范围。
query	后端最近执行的查询。如果state为 `active` ，将会显示当前正在执行的查询。在所有其他状态下，显示上一个被执行的查询。
backend_type	当前后端的类型。可能的类型为autovacuum launcher、autovacuum worker、logical replication launcher、logical replication worker、parallel worker、background writer、client backend、checkpointer、startup、walreceiver、walsender以及 walwriter。除此之外还包括后端的执行组件，例如PQE等。

HoloWeb可视化活跃Query管理

您可以通过HoloWeb可视化查看活跃Query，并进行管理。

1. 登录HoloWeb控制台，详情请参见连接HoloWeb。
2. 单击顶部导航栏的**诊断与优化**。
3. 在左侧导航栏单击**活跃Query**。
4. 进入**活跃Query**页面，查看当前实例的活跃Query及对活跃Query进行管理，例如执行**取消**操作。
5. （可选）单击目标查询操作列的**详情**，查看当前查询的详细信息。您还可以单击**详情**页面的**复制**，复制当前查询执行的SQL语句。

查看SQL运行信息

Superuser可以查看所有用户的SQL运行信息，RAM用户只能查看自己的SQL运行信息。更多关于参数的说明，请参见参数说明。

1. 您可以通过如下语句查看当前实例内用户的SQL运行信息。

   ```
   SELECT datname::text,usename,query,pid::text,state FROM pg_stat_activity;
   ```

2. 您可以执行如下语句查看当前正在运行的SQL信息。

   ```
   SELECT datname::text,usename,query,pid::text,state
      FROM pg_stat_activity
      WHERE state != 'idle' ;
   ```

3. 您可以执行以下语句查看当前实例正在运行且耗时较长的SQL信息。

```
SELECT current_timestamp - query_start as runtime, datname::text, usename, query, pid::
text
    FROM pg_stat_activity
    WHERE state != 'idle'
    order by 1 desc;
```

示例执行结果如下所示，该示例中可以看到耗时较久的语句为 `UPDATE` 。

```
runtime          | datname   | usename  |       current_query            | pid
-----------------+-----------+----------+--------------------------------+-------------
 00:00:24.258388 | holotest  | 123xxx   |  UPDATE holo_order             |1267xx
                                           : set gmt = now(),
                                           : trade_id = $1,
                                           : trade_create_time = $2;
 00:00:1.186394  | testdb    | 156xx    |  SELECT * FROM oder;           |1783xx
(2 rows)
```

4. 终止Query

   如果当前存在不符合预期的Query，您可以根据实际情况通过如下命令进行终止。

   - 取消当前连接上的Query。

     ```
     select pg_cancel_backend(<pid>);
     ```

   - 批量取消Query。

     ```
     SELECT pg_cancel_backend(pid)
            ,query
            ,datname::text
            ,usename
            ,application_name
            ,client_addr
            ,client_port
            ,backend_start
            ,state
     FROM    pg_stat_activity
     WHERE   length(query) > 0
     AND     pid != pg_backend_pid()
     AND     backend_type = 'client backend'
     AND     application_name != 'hologres'
     AND     usename != 'holo_admin'
     AND     query not like '%pg_cancel_backend%';
     ```

修改活跃Query超时时间

Hologres支持您通过如下方式修改活跃Query运行超时时间。

- 语法示例

  ```
  set statement_timeout = <time>;
  ```

- 参数说明

  time：超时时间取值范围为0~2147483647ms，单位默认为ms（当time后加单位时需要使用单引号，否则会报错）。当前默认超时时间为8小时，该设置针对session级别生效。

> **说明** set statement_timeout = <time> 和要修改超时时间的SQL语句一起执行方可生效。

- 使用示例
  - 设置超时时间为5000min，其中具体时间带单位，5000min需要整体添加单引号。

```
set statement_timeout = '5000min' ;
select * from tablename;
```

  - 设置超时时间为5000ms。

```
set statement_timeout = 5000 ;
select * from tablename;
```

修改空闲Query超时时间

参数idle_in_transaction_session_timeout描述了事务进入idle状态后的超时行为，如果不设置参数值，默认不会做事务超时的释放，容易发生事务不释放，导致查询被锁死的情况。Hologres支持您通过如下方式修改空闲Query运行超时时间。

- 应用场景

  当Query执行产生死锁时，需要设置超时时间。例如如下代码，未执行 `commit` ，开启了一个事务，但是没有提交，会造成事务泄漏，进而引发数据库级别的死锁，影响服务正常使用。

```
begin;
select * from t;
```

  当出现这种死锁场景时，可以通过设置idle_in_transaction_session_timeout超时时间来解决。当一个带事务的空闲连接超过idle_in_transaction_session_timeout设置的时间还未提交或者回滚事务，系统将自动根据超时时间回滚事务，并关闭连接。

- 语法示例

```
--session修改空闲事务超时时间
set idle_in_transaction_session_timeout=<time>;
--DB级别修改空闲事务超时时间
alter database db_name set idle_in_transaction_session_timeout=<time>;
```

- 参数说明

  time：超时时间取值范围为0~2147483647ms，单位默认为ms（当time后加单位时需要使用单引号，否则会报错）。在Hologres V0.10及以下版本，默认值为0，即不会自动清理；在Hologres V1.1版本，默认值为10分钟，超过10分钟后将会回滚事务。

> **说明** 不建议超时时间设置过短，如果过短容易错误回滚正在使用中的事务。

- 使用示例

  设置超时时间为300000ms。

```
--session修改空闲事务超时时间
set idle_in_transaction_session_timeout=300000;
--DB级别修改空闲事务超时时间
alter database db_name set idle_in_transaction_session_timeout=300000;
```

查询慢Query日志

从Hologres V0.10版本开始，支持进行慢query日志查询，详情请参见慢Query日志查看与分析。

8.4.3. 连接数管理

本文将为您介绍如何对实例中的连接进行诊断和管理。

连接和Query管理概述

Hologres兼容PostgreSQL，可以通过查询pg_stat_activity视图信息来查看实例连接信息，以达到分析实例连接状态和诊断运行SQL的目的。具体涉及的操作内容如下所示：

- 查询实例的默认最大连接数：不同的实例规格对应不同的默认连接数，通过命令查询当前实例规格的最大连接数。
- HoloWeb可视化管理连接：通过HoloWeb可视化查看活跃连接，并进行管理如Kill等操作。
- 通过SQL查询连接信息：通过查询实例、DB的连接数、每个连接状态以及终止空闲连接，更好的管理实例。
- 释放连接：通过SQL函数，释放指定连接资源。
- 自动释放空闲连接（Beta）：开启自动释放空闲连接功能自动释放长期不使用连接。
- 管理员预留连接：用于在连接数达到最大时对连接进行管理操作。
- 单个用户连接数限制：为单个用户设置连接数上限，以防止某个用户占用过多连接造成资源浪费。
- 连接数使用最佳实践：使用Hologres连接数的最佳实践建议。

查询pg_stat_activity视图信息

pg_stat_activity是一个非常有用的视图，可以分析排查当前运行的SQL任务以及一些异常问题。您可以执行如下命令查看实例连接和Query的运行信息。

```
select * from pg_stat_activity ;
```

pg_stat_activity视图的参数说明如下所示。

字段	描述
datid	Hologres后端连接到的数据库的OID。
datname	Hologres后端连接到的数据库的名称。
pid	Hologres后端的进程ID。
usesysid	登录到Hologres后端的用户OID。
usename	当前连接的用户名。 `holo_admin` 是Hologres内置的服务账号，运行结果显示为PSQL，该连接是必要的。如果您的连接数未超出当前实例的最大限度，不需要针对此连接进行优化。连接数查询方式，请参见通过SQL查询连接信息。
application_name	客户端的应用类型。

字段	描述
client_addr	客户端的IP地址。 显示的IP地址可能是被解析过的，不保证一定是源端IP。
client_hostname	客户端的主机名。
client_port	客户端的端口。
backend_start	后台进程开始的时间。
xact_start	该进程的当前事务被启动的时间。 • 如果没有活动事务，则为空。 • 如果当前查询是该进程的第一个事务，这一列等于query_start。
query_start	当前活动查询开始的时间，如果当前连接状态不是active，取值为上一个查询开始的时间。
state_change	连接的状态（state）上一次被改变的时间。
wait_event_type	后端正在等待的事件类型，如果不存在则为NULL。可能的取值有： • LWLock：后端正在等待一个轻量级锁。 • Lock：后端正在等待一个重量级锁。`wait_event`将标识等待的锁的类型。 • BufferPin：服务器进程正在等待访问一个数据缓冲区，而此时没有其他进程正在检查该缓冲区。 • Activity：服务器进程处于闲置状态。被用于在其主处理循环中等待活动的系统进程。 • Extension：服务器进程正在一个扩展模块中等待活动。 • Client：服务器进程正在等待来自用户应用的某种查询，并且该服务器预期某种与其内部处理无关的事情发生。 • PC：服务器进程正在等待来自服务器中另一个进程的某种活动。 • Timeout：服务器进程正在等待一次超时发生。 • IO：服务器进程正在等待一次IO完成。
wait_event	如果后端当前正在等待，则是等待事件的名称，否则为 NULL。
state	表示连接的状态。常见的状态如下： • active：活跃。 • idle：空闲。 • idle in transaction：长事务中的空闲状态。 • idle in transaction（Aborted）：已失败事务中的空闲状态。 • \N: 状态为空，表示非用户连接的进程，一般属于系统后台的维护进程，可以忽略。
backend_xid	Hologres后端的顶层事务标识符。
backend_xmin	当前后端的xmin范围。

字段	描述
query	后端最近执行的查询。如果state为 `active` ，将会显示当前正在执行的查询。在所有其他状态下，显示上一个被执行的查询。
backend_type	当前后端的类型。可能的类型为autovacuum launcher、autovacuum worker、logical replication launcher、logical replication worker、parallel worker、background writer、client backend、checkpointer、startup、walreceiver、walsender以及 walwriter。除此之外还包括后端的执行组件，例如PQE等。

查询实例的默认最大连接数

不同的实例规格对应不同的默认连接数，您可以通过以下命令查询当前实例规格的最大连接数。命令执行完成后显示结果为单个FrontEnd节点的连接数，总连接数需要乘实例的FrontEnd节点数，不同实例的Frontend节点数请参见实例规格概述。

```
--查看单接入节点的最大连接数。
show max_connections;
```

HoloWeb可视化管理连接

您可以通过HoloWeb可视化查看活跃连接并进行管理。

1. 登录HoloWeb控制台，详情请参见连接HoloWeb。
2. 单击顶部导航栏的**诊断与优化**。
3. 在左侧导航栏单击**活跃连接管理**。
4. 进入**活跃连接管理**页面，查看当前实例的活跃连接及对连接进行管理，例如执行**Kill**操作。

通过SQL查询连接信息

若您更倾向于使用SQL的方式查询连接信息，可以通过以下方式操作。

1. 查询当前数据库的连接数。

   您可以通过如下命令查看当前数据库的连接数，更多关于参数的说明，请参见参数说明。

   - Hologres V1.1及以上版本

```
SELECT  datname::TEXT
        ,COUNT(1) AS COUNT
FROM    pg_stat_activity
WHERE   backend_type = 'client backend'
AND     application_name != 'hologres'
AND     usename != 'holo_admin'
GROUP BY datname::TEXT;
```

- Hologres V0.10及以下版本

```
SELECT  datname
        ,COUNT(1) AS COUNT
FROM    pg_stat_activity
WHERE   backend_type = 'client backend'
AND     application_name != 'hologres'
AND     usename != 'holo_admin'
GROUP BY datname;
```

2. 查看每个连接的状态。

   您可以在Hologres管控台通过HoloWeb查看每个实例的连接状态，也可以执行如下语句，通过查询pg_stat_activity视图来获取所有JDBC或PSQL连接的状态。

```
select * from pg_stat_activity where backend_type = 'client backend' and state = '<stat
ename>';
```

   其中statename是需要填写的状态参数名，包括以下几种：

   - idle：空闲连接，表示进程在等待新的客户端命令。
   - active：活跃连接，表示进程正在执行查询操作。
   - idle in transaction：表示进程处于一个事务中，但是当前没有执行查询操作。
   - idle in transaction (aborted)：表示进程处于一个事务中，该事务存在语句错误，并且进程当前没有执行查询操作。

   例如，您可以执行如下命令查询当前实例的空闲连接。

```
select * from pg_stat_activity where backend_type = 'client backend' and state = 'idle'
;
```

   Holoweb等Hologres周边组件会通过JDBC的方式占用一定的连接数，如果您的连接数满足需求，无需关心此类连接数的占用。当SQL连接数长期接近或达到 `max_connections` 时，意味着您需要检查您的应用是否存在连接数泄漏情况，需要在应用端合理设置连接池大小，或者您也可以根据下面章节释放空闲连接。

3. 查看每个接入节点的连接数

   Hologres实例是由多个接入节点（FrontEnd）组成的集群模式，每个节点独立维护一组连接，节点之间采用负载均衡调度方式，一个连接属于一个接入节点。当连接在不同节点之间发生不均衡，造成某个接入节点连接数超过单节点上限，会出现连接数过多的异常提示。可以通过如下命令查询每个接入节点的实时连接数。

```
-- pid表示全局视角的进程号，进程号的后两位是固定编码的FrontEnd ID。
select mod(pid,100) as fe_id, count(*)
from pg_stat_activity
where backend_type='client backend'
group by 1 order by 1;
```

释放连接

如果您遇到如下情况，则说明系统连接数（或者某个接入节点）已经达到上限：

- 连接数达到甚至超出 `max_connections` 的取值，您可以在Hologres管控台的监控告警页查看连接数。
- 产生 `FATAL: sorry, too many clients already connection limit exceeded for superusers` 报错。

- 产生 `FATAL: remaining connection slots are reserved for non-replication superuser connections` 报错。

当您有上述情况产生，可以通过Superuser账号连接实例，执行如下语句查看空闲连接是否过多。

```
select * from pg_stat_activity where backend_type = 'client backend' and state = 'idle';
```

如果查询结果显示空闲进程过多，并且确定是无用的空闲连接时，可以找到上述语句结果中的pid字段，并执行如下语句释放空闲连接。更多关于参数的说明，请参见参数说明。

```
--结束对应的后台连接进程
select pg_terminate_backend(<pid>);
--批量终止后台idle连接进程，释放连接
SELECT pg_terminate_backend(pid)
FROM    pg_stat_activity
WHERE   pid != pg_backend_pid()
AND     backend_type = 'client backend'
AND     state = 'idle'
AND     application_name != 'hologres'
AND     usename != 'holo_admin';
```

自动释放空闲连接（Beta）

当实例存在连接数长期接近或达到上限值时，可能存在连接泄漏情况，可以开启自动释放空闲连接功能来释放长期不使用连接。当连接空闲时间（无SQL执行）超过设置时间时，连接会自动被断开。

- 使用限制

  仅Hologres V0.10.25及以上版本支持自动释放空闲连接功能，如果您的实例是V0.10.25以下版本，请您提交工单或加入在线支持钉钉群申请升级实例，加入在线支持钉钉群请参见如何获取更多的在线支持？。

- 语法示例
  - session级别。

    ```
    -- 连接在10分钟没有执行sql的情况，会自动断开，单位为毫秒。
    SET idle_session_timeout = 600000;
    ```

  - 数据库级别。

    ```
    -- 连接在10分钟没有执行sql的情况，会自动断开，单位为毫秒。
    ALTER DATABASE  <db_name> SET idle_session_timeout = 600000;
    ```

    db_name为要开启自动释放空闲连接功能的数据库名称。

管理员预留连接

Hologres会为Superuser预留连接，不同的实例规格预留的连接数不同，详情请参见实例规格概述。Superuser预留连接用于在连接数达到最大时对连接进行管理操作（如终止idle连接），普通用户的连接数最大为max_connections减去预留连接。在实践中，不建议普通用户使用Superuser账号操作数据库，否则会导致连接全部占满，且无法通过管理渠道释放连接。

单个用户连接数限制

由于Hologres兼容PostgreSQL，因此支持为单个用户设置连接数上限，以防止某个用户占用过多连接造成资源浪费。

> **说明** 单个用户连接数限制只能限制普通用户，不能限制Superuser，请在实际业务中尽量避免使用Superuser账号直接连接应用。

1. 限制单个用户在单个接入节点上的最大连接数。
   - 语法示例

     ```
     ALTER ROLE "云账号ID" CONNECTION LIMIT <number>;
     ```

   - 参数说明

参数	说明
云账号ID	需要限制的账号ID，如果为RAM用户，需要在账号UID前加p4_。更多关于账号的说明，请参见账号概述。
number	限制的连接数个数。

   - 使用示例

     如下示例限制RAM用户283813xxxx一个节点最多只有一个连接。

     ```
     ALTER ROLE "p4_283813xxxx" CONNECTION LIMIT 1;
     ```

2. 您可以执行如下语句查看当前已经为实例用户设置的单节点限制连接数。

   ```
   SELECT rolname, rolconnlimit
   FROM pg_roles
   WHERE rolconnlimit <> -1;
   ```

   查询示例结果如下：

   ```
          rolname | rolconnlimit
   ---------------+--------------
    p4_283813xxxx |            1
   (1 row)
   ```

连接数使用最佳实践

在使用Hologres连接数时的最佳实践建议如下所示。

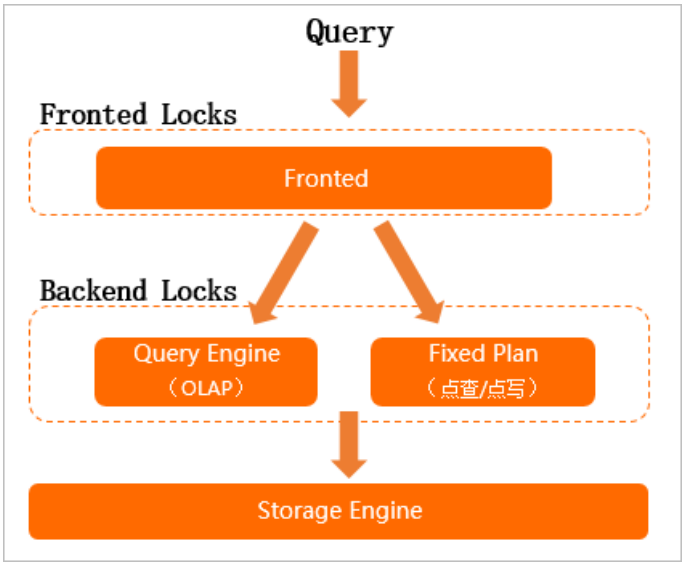
- 善用Superuser账号
  - 不建议使用Suerpser账号直接操作实例或者连接应用，否则当连接数超过实例默认连接数之后，账号将无法连接实例。
  - 建议专门设置一个Superuser账号为运维账号，当出现连接数超过实例默认连接数、Query卡死等紧急情况时，使用该Superuser账号登录HoloWeb管理连接和Query。
- 设置合理的连接池
  - 为了业务安全考虑，Hologres不会在后台自动释放连接，建议业务上给应用设置合理的连接池机制，让空闲连接及时释放。
  - 定期清理空闲连接，防止空闲连接占用过多连接影响线上业务。
  - Hologres连接中，名称为holo_admin的连接为后台运维连接，会定期清理，可以忽略。

8.4.4. 锁以及排查锁

锁是一种数据库的信号量管理机制，作用是保证SQL执行上不同事务的隔离性。本文为您介绍Hologres中的锁及如何排查锁。

背景信息

当一个Query发起时，在Hologres中其经过的链路如下图所示。



Frontend解析Query，Query Engine生成执行计划，然后Storage Engine读取数据，整个链路的锁有如下两种。

- Frontend Locks（前端锁）

  Frontend为接入层，兼容PostgreSQL协议，因此锁也会兼容PostgreSQL的部分锁，主要用来管理Frontend（FE）元数据信息。

- Backend Locks（后端锁）

  Backend是指Query Engine和Fixed Plan，将会享有Hologres自带的锁，主要用于管理Storage Engine的Schema和数据。

锁介绍

- Frontend Locks（FE Locks）

  Hologres接入层Frontend兼容PostgreSQL，因此在接入层的锁与PostgreSQL兼容。PostgreSQL提供了Table-level Lock、Row-level Lock和Advisory Lock三种锁模式来控制并发的数据访问。 目前Hologres兼容的是Table-level Lock和Advisory Lock。

  > **说明** Hologres目前不支持显式的设置锁命令以及Advisory Lock相关的UDF。

  - Table-level Lock
    - 分类

      Table-level Lock是指表锁，表锁包含如下种类。

锁名称	说明	备注
ACCESS SHARE	一般情况下只有 `SELECT` 命令会获取相关表的这个锁。	不涉及。
ROW SHARE	只有 `SELECT FOR UPDATE` 和 `SELECT FOR SHARE` 这两个命令需要获取目标表的这个锁，非目标表（比如JOIN关联的其他表）仅获取ACCESS SHARE锁。	Hologres不支持 `SELECT FOR UPDATE` 和 `SELECT FOR SHARE` 命令，因此无需关注。
ROW EXCLUSIVE	`UPDATE` 、 `DELETE` 和 `INSERT` 修改数据的命令需要获取此锁。	需要结合Backend Locks一起关注。

锁名称	说明	备注
SHARE UPDATE EXCLUSIVE	为了防止Vacuum和并发的Schema变更发生冲突的锁，如下命令需要获取此锁。 ■ `lazy VACUUM` 即非 `Vacuum Full` 。 ■ `ANALYZE` 。 ■ `CREATE INDEX CONCURRENTLY` 。 **说明** Hologres使用此命令时不会获取 SHARE UPDATE EXCLUSIVE锁，而是与非CONCURRENTLY形式的 `CREATE INDEX` 一样获取 `SHARE` 锁。 ■ `CREATE STATISTICS` ，此命令Hologres不支持。 ■ `COMMENT ON` 。 ■ `ALTER TABLE VALIDATE CONSTRAINT` ，此命令Hologres不支持。 ■ `ALTER TABLE SET/RESET (storage_parameter)` ，Hologres仅支持使用此命令设置自己扩展的属性以及PostgreSQL原生的autovacuum_enabled属性，不支持设置其他属性；且设置上述属性时不会获取表的任何锁。修改PostgreSQL内置的其他某些storage parameter参数需要获取此锁，详情请参见ALTER TABLE。 ■ `ALTER TABLE ALTER COLUMN SET/RESET options` 。 ■ `ALTER TABLE SET STATISTICS` ，此命令Hologres不支持。 ■ `ALTER TABLE CLUSTER ON` ，此命令Hologres不支持。 ■ `ALTER TABLE SET WITHOUT CLUSTER` ，此命令Hologres不支持。	重点关注 `ANALYZE` 命令。

锁名称	说明	备注
SHARE	只有非concurrently的 `CREATE INDEX` 需要获取此锁。 **说明** Hologres中创建JSON索引时会需要拿此锁。	关注 `CREATE INDEX` 命令（创建JSON相关索引）。
SHARE ROW EXCLUSIVE	为了防止并发的数据修改的锁，需要获取此锁的命令如下。 ■ `CREATE COLLATION` ，此命令Hologres不支持。 ■ `CREATE TRIGGER:` ，此命令Hologres不支持。 ■ 部分 `ALTER TABLE` 命令。 ■ `DISABLE/ENABLE [ REPLICA | ALWAYS ] TRIGGER` ，此命令Hologres不支持。 ■ `ADD table_constraint` ，此命令Hologres不支持。	Hologres不支持获取此锁的命令，所以无需关注。
EXCLUSIVE	仅有 `REFRESH MATERIALIZED VIEW CONCURRENTLY` 命令需要获取此锁。	Hologres不支持 `REFRESH MATERIALIZED VIEW CONCURRENTLY` 命令，无需关注。
ACCESS EXCLUSIVE	完全独占访问需要的锁，与其他所有的锁都冲突，需要获取此锁的命令如下。 ■ `DROP TABLE` ■ `TRUNCATE TABLE` ■ `REINDEX` ，此命令Hologres不支持。 ■ `CLUSTER` ，此命令Hologres不支持。 ■ `VACUUM FULL` ■ `REFRESH MATERIALIZED VIEW (without CONCURRENTLY)` ，此命令Hologres不支持。 ■ `LOCK` ，显式的LOCK命令，如果不指明具体的锁类型，默认需要这个锁；此命令Hologres不支持。 ■ `ALTER TABLE` ，除去上面明确提到获取特定锁的 `ALTER TABLE` 形式以外，其他的 `ALTER TABLE` 命令形式都默认获取这个锁。	需要重点关注的锁，在Hologres中执行的DDL操作都会获取该锁，与其他锁冲突。

- 超时时间

  FE Locks没有默认超时时间，需要业务设置超时时间，防止锁等待时间过长，详情请参见Query管理。

- 冲突关系

  锁之间的冲突关系如下表所示，冲突意味着一个操作获取锁，其他操作获取的锁都需要等待。

  > **说明** ✓表示不冲突，✗表示冲突。

Requested Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	✓	✓	✓	✓	✓	✓	✓	✗
ROW SHARE	✓	✓	✓	✓	✓	✓	✗	✗
ROW EXCLUSIVE	✓	✓	✓	✓	✗	✗	✗	✗
SHARE UPDATE EXCLUSIVE	✓	✓	✓	✗	✗	✗	✗	✗
SHARE	✓	✓	✗	✗	✓	✗	✗	✗
SHARE ROW EXCLUSIVE	✓	✓	✗	✗	✗	✗	✗	✗
EXCLUSIVE	✓	✗	✗	✗	✗	✗	✗	✗
ACCESS EXCLUSIVE	✗	✗	✗	✗	✗	✗	✗	✗

  - Advisory Lock

    Advisory Lock指咨询锁，PostgreSQL提供了一种方法创建由应用定义其含义的锁。Hologres大部分业务情况无需额外关注。

- Backend Locks（BE Locks）

- 分类

  在Hologres中BE Locks锁分类如下。

锁分类	锁介绍
Exclusive(X)	排他锁（也称互斥锁），当事务需要修改一批或一条数据时申请排他锁，例如DML语句（DELETE、INSERT、UPDATE）。排他锁申请成功的前提是同一资源上没有其他的共享锁或排他锁，当排他锁申请成功后，锁资源上将不能同时有其他锁。
Shared(S)	共享锁，当事务需要读取一批或一条数据时申请共享锁，以免其他事务对将要读取的数据做修改。同一个资源上允许存在多个共享锁，即允许DQL之间并发执行，因为DQL不会改变资源本身。
Intent(I)	意向锁，这种类型的锁通常用于表达锁的层次结构，同一个资源上允许存在多个意向锁，当申请成功后，该资源上就不能有排他锁。例如，当事务申请一行的排他锁时，它会同时申请表的意向锁（表是比行更高层次的资源），以防止其他事务申请表的排他锁。

- 超时时间

  BE Locks超时时间默认为5min，超过该时间则报错。

- 冲突关系

  BE Locks的冲突关系如下，冲突表示一个操作获取锁，其余操作都需要等待锁。

  > **说明** ✅表示不冲突，❌表示冲突。

操作	DROP	ALTER	SELECT	UPDATE	DETELE	INSERT（包含INSERT ON CONFLICT）
DROP	❌	❌	❌	❌	❌	❌
ALTER	❌	❌	✅	❌	❌	❌
SELECT	❌	✅	✅	✅	✅	✅
UPDATE	❌	❌	✅	❌	❌	❌
DETELE	❌	❌	✅	❌	❌	❌
INSERT（包含INSERT ON CONFLICT）	❌	❌	✅	❌	❌	❌

锁的作用范围

根据锁的分类，不同的锁作用范围不同。

- FE Locks

  FE Locks只会作用在表对象上，与表数据无关，且锁只有成功或者卡住两种状态，卡住就代表有锁冲突在等待锁。

- BE Locks

  BE Locks会作用在数据或者表结构上，因此在作用范围上会分为表数据锁、行数据锁、表结构（Schema）锁。

  - 表数据锁：代表整个表的数据都需要获取锁，如果多个任务同时都需要获取表数据锁，会造成任务等待锁现象，从而任务延迟。
  - 行数据锁：行锁是指整行数据获取锁，在执行效率上会更高，其中走Fixed Plan加速SQL执行的Query都是行锁或者表Schema锁。
  - 表Schema锁：即表结构锁，当事务需要读取或修改表结构时申请Schema锁，大部分的事务都会申请Schema锁。目前主要有如下类型的Schema锁。
    - SchX：Schema排他锁，用于DDL语句，目前只支持 `DROP TABLE` 命令。
    - SchU：Schema更新锁，用于修改表结构的DDL语句包含 `ALTER TABLE` 和 `set_table_property` 命令。
    - SchE：Schema存在锁，用户DML和DQL语句，确保读写数据期间表不会被删除。

    > **说明**
    > - SchU是DDL锁更细粒度的管控，允许ALTER TABLE期间DQL正常运行，无需等待；SchX是最粗粒度的DDL排他锁，所有的DDL、DML、DQL都会等待。
    > - 如果Start Query耗时较长，则可能在等待BE Locks。

在Hologres中，其中常见命令作用的锁范围如下。✅表示操作命令会获取该锁。

> **说明**
> - 非Fixed Plan的写入、更新、删除都为Bulkload。
> - `CREATE INDEX` 命令目前是指创建JSON相关索引。
> - DDL命令包括 `CREATE` 、 `DROP` 、 `ALTER` 等。

操作/锁范围	表锁	表数据锁	行数据锁	表Schema锁
CREATE	✅	不涉及		

操作/锁范围	表锁	表数据锁	行数据锁	表Schema锁
DROP	✅ **说明** 一旦DROP命令获取锁，无法执行其他命令，其他命令会等待锁，直到发现表被删除了，则其他命令失败。 说明：	不涉及	不涉及	✅ **说明** 与其他操作都冲突。
ALTER	✅ **说明** 与DROP命令一致。	不涉及	不涉及	✅ **说明** 在获取表Schema锁的同时可以对表执行 `SELECT` 命令。
SELECT	✅ **说明** 在获取表锁的同时可以对表执行 `INSERT` 、 `UPDATE` 、 `DELETE` 命令，只与DDL锁冲突。	不涉及	不涉及	✅ **说明** 执行 `SELECT` 命令时除了不能执行 `DROP` 命令，其他操作都可以做。

操作/锁范围	表锁	表数据锁	行数据锁	表Schema锁
INSERT（包含INSERT ON CONFLICT）	✓ 说明 INSERT 命令与 CREATE INDEX 、DDL冲突。	如果是通过Bulkload方式则获取表数据锁。 说明 Bulkload与Fixed Plan相互冲突。	如果是通过Fixed Plan方式则获取行数据锁。 说明 如果有离线写入，不能同时刻同一个表既有离线又有实时Fixed Plan写入。	✓ 说明 与DDL、DML冲突。
UPDATE	✓ 说明 与 CREATE INDEX 、DDL锁冲突。			✓ 说明 与DDL、DML冲突。
DELETE	✓ 说明 与 CREATE INDEX 、DDL锁冲突。			✓ 说明 与DDL、DML冲突。

事务相关的锁说明

Hologres目前仅支持DDL的显式事务，并不支持单纯的DML事务，也不支持DDL和DML混合的事务。

- Hologres不支持嵌套子事务。
- 单纯的DML事务虽然语法上可以通过，但是实际不支持原子提交和回滚。

  如下DML即使 `insert` 成功，如果之后的 `update` 失败了， `insert` 的数据也不会被回滚。

```
begin;
insert into t1(id, salary) values (1, 0);
update t1 set salary = 0.1 where id = 1;
commit;
```

- 纯DDL事务可以按预期的方式工作

其中任何一行DDL命令失败则会整个事务被回滚。例如下面 `alter` 命令失败时，前面 `create` 和 `drop` 操作会被回滚。

```
begin;
create table t1(i int);
drop table if exists t2;
alter table t1 add column n text;
commit;
```

- DDL和DML命令混合的事务会被禁止

  如下示例，当事务中包含DDL和DML命令时，DML命令就会报错。

```
begin;
create table t1(i int);
update t1 set i = 1 where i = 1;
-- DML命令报错
ERROR:  UPDATE in ddl transaction is not supported now.
```

- 显式事务中任何一个命令获得的锁都只会在整个事务结束（提交或回滚）时才统一释放。

  如下示例，当对父表做 `alter` 操作时，会同时获取父表（login_history）和子表（login_history_202001）的ACCESS EXCLUSIVE锁， 但是这个命令执行完不会立即释放对应的锁，而是要等待最后的 `commit` 执行完（不管成功还是失败）才会释放锁。若是一直不执行 `commit` ，则会一直保持锁，这时若有其他对这个表的DDL操作，则会锁表并报错。

```
 -- suppose we have three tables
create table temp1(i int, t text);
create table login_history(ds text, user_id bigint, ts timestamptz) partition by list (ds
);
create table login_history_202001 partition of login_history for values in ('202001');
begin;
alter table login_history_s1 add column user_id bigint;
drop table temp1;
create table tx2(i int);
commit;
```

锁排查

FE Locks在实际业务中是最常见也最容易引起问题的锁。可以通过系统表查询锁住的表以及SQL，步骤如下所示。

在具体的锁排查介绍之前，需要用到如下PostgreSQL系统视图或者表，通过这些信息的结合来查出锁信息。

- pg_stat_activity：查看当前实例正在运行的SQL，详情请参见查询pg_stat_activity视图信息。
- pg_locks：查看当前实例正在运行中的进程或者正在等待的锁，详情请参见pg_catalog.pg_locks。

1. （可选）查询运行时间较长的Query

   通过pg_stat_activity查看运行时间较长的Query以及对应的状态，若是有Query长时间不结束，可能是这个表可能已经有其他的操作获取了锁，导致这个Query等待锁。

   > **说明** 客户端可以通过参数设置application_name，更容易排查问题。 例如在通过命令行psql连接时，可以通过环境变量PGAPPNAME来指定，命令为： `PGAPPNAME=mac-sql psql -h holo-url -p 80 -d PostgreSQL` 。

```
select datname, pid, application_name, wait_event_type, state, backend_xid, backend_sta
rt, xact_start, query_start, query from pg_stat_activity where backend_type in ('client
backend');
```

2. 查询是否有锁

   通过以下命令查看是否有锁，排查长时间运行的Query是否被锁住，`f` 即 `false`，表示当前PID进程正在等待其他进程释放锁。

   ```
   -- 查看所有在等的锁，持有的锁，及持有/等待的进程等信息
   select * from pg_locks where granted = 'f';
   ```

   若是知道哪个Query运行时间比较长，可以通过PID反查是否有锁，若是出现 `f` 则说明在等待锁。

   ```
   select * from pg_locks where pid = <PID>;
   ```

3. 排查哪个进程保持着锁状态

   通过步骤2的结果得出哪个PID进程在等待锁，根据其OID（表关系）通过以下命令查看是否拿锁，`t` 即 `true`，代表正在拿着锁。

   ```
   -- 查看获取了表锁的进程信息
   select pid from pg_locks where relation = <OID> and granted = 't';
   ```

4. 排查保持锁的Query

   通过步骤3查询到的PID，查询拿锁的Query。

   ```
   select * from pg_stat_activity where pid = <PID>;
   ```

5. 释放锁

   找到拿锁的Query之后，通过如下命令可以直接将Query进行删除（Kill）以释放锁。

   ```
   select pg_cancel_backend(<PID>);
   ```

常见问题

- 报错：`internal error: Cannot acquire lock in time, current owners: [(Transaction =302xxxx, Lock Mode = SchS|SchE|X)].`。
  - 可能原因：执行Query的表有被其他Query获取了BE Locks（如报错 `Lock Mode = SchS|SchE|X`，则为Schema稳定锁、存在锁、排他锁），导致Query等待BE Locks超时（5min）从而报错。
  - 解决方法：报错中的 `transaction` 即 `Transaction =302xxxx` 对应Query ID，可以通过Query ID在慢Query日志或者活跃Query中查看对应获取锁的Query。
- 报错：`ERROR: Operation timed out: Wait schema version timeout.: Server current target schema version:xxx is late from request schema version: yyy`。
  - 可能原因：执行DDL后，会先在Frontend（FE）节点执行，再异步在Storage Engine（SE）执行，当FE执行DDL结束后会对节点版本（Version）进行更新，若SE的DDL还没执行完成，导致SE的版本比FE的版本低，此时Query会等待SE执行DDL，如果超过5min后SE仍然没有执行完成，就会报错等待Schema版本超时。
  - 解决方法：
    - 先Kill掉等待锁的DML，然后重试Query。
    - 重启实例（极端手段）。

- 报错：`The requested table name: xxx (id: 10, version: 26) mismatches the version of the table (id: 10, version: 28) from server` 。
  - 可能原因：执行DDL后，会先在Frontend（FE）单个节点执行，再异步在Storage Engine（SE）执行。SE已经完成DDL并更新版本，但是因为FE节点较多，节点间Replay未完成，导致部分FE节点的版本低于SE，而Query正好在版本较低的FE中执行，从而出现报错。
  - 解决方法：
    - 多重试几次Query。
    - 若是超过几分钟还报错建议重启实例。
- 报错：`internal error: Cannot find index full ID: 86xxx (table id: 20yy, index id: 1) in storages or it is deleting!` 。
  - 可能原因：执行`DROP Table`或者`Truncate Table`时，对这个表进行DML（SELECT、DROP、TRUNCATE）需要获取DDL锁，导致Query需要等锁，直到发现表被删除了从而报错。
  - 解决方法：执行`DROP`和`TRUNCATE`命令时，不对表执行其他Query。

8.5. 存储优化

8.5.1. 更改列存表的数据存储格式

从Hologres V0.10版本开始，Hologres创建的列存表数据存储格式全面升级为AliORC，该存储格式能够进一步压缩数据存储大小，降低存储成本。本文将会为您介绍在Hologres中如何更改列存表的数据存储格式。

使用限制

在Hologres中更改列存表的数据存储格式，具体限制如下：

- 该功能仅Hologres V0.10及以上版本支持，请在Hologres管控台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
- 升级前的数据列存表存储格式为segment，升级至Hologres V0.10版本后，新建的列存表将会默认为AliORC（参数值为orc）存储格式，行存表为sst存储格式。当前版本支持对升级前的历史数据更换，但有如下限制：
  - 当前版本仅支持单表转换列存表存储格式，不支持批量转换。
  - 针对分区表，仅需对父表进行列存数据存储格式转换操作即可。
  - 转换数据存储格式后，通过后台线程异步将旧的format数据切换到新的format上。**后台异步切换过程会消耗系统资源**，包括CPI、I/O、网络等，**切换期间可能会对线上任务有一定的性能影响**，整个切换时长跟存量数据大小以及实时写入量相关，**请在业务低峰期进行切换**。
- 不支持通过修改存储格式完成列存表转换为行存表，如需变换，请重新创建新的表结构。

查看数据存储格式

您可以使用如下SQL语句查看数据库中各张表的存储格式：

- 语法示例

```
SELECT  *
FROM    hologres.hg_table_properties
WHERE   property_key = 'storage_format'
;
```

- 查询结果

```
 table_namespace |     table_name     |  property_key  | property_value
-----------------+--------------------+----------------+----------------
 public          | part               | storage_format | segment
 public          | partsupp           | storage_format | segment
 public          | customer           | storage_format | segment
 public          | orders_row         | storage_format | sst
 public          | sp_orders          | storage_format | segment
 public          | sp_orders_20161231 | storage_format | segment
 public          | sp_orders_20171231 | storage_format | segment
 public          | sp_orders_20181231 | storage_format | segment
 public          | lineitem           | storage_format | segment
 public          | nation             | storage_format | orc
 public          | region             | storage_format | orc
 public          | supplier           | storage_format | orc
 public          | orders             | storage_format | orc
(13 rows)
```

- 查询结果参数说明

参数	说明
segment	Hologres早期版本列存表的默认格式。
sst	行存表的默认格式。
orc	Hologres V0.10及以上版本列存表的默认格式。其中property_value为orc的即为使用AliORC格式存储数据的表。

更新表的数据存储格式

在您升级Hologres实例至V0.10版本后，默认创建的表均会以AliORC格式存储数据。但是已经存在的数据表需要您手工修改其存储类型，命令语法如下：

```
call set_table_property ('table_name', 'storage_format', 'orc');
```

其中，您需要将table_name替换为实际的表名。如下示例将表part的存储类型更换为AliORC存储格式。

```
call set_table_property ('public.part', 'storage_format', 'orc');
```

9.数据迁移

9.1. 迁移Lightning至Hologres

本文为您介绍如何迁移Lightning服务至交互式分析Hologres进行数据开发。

背景信息

Lightning是MaxCompute产品的交互式查询服务，支持使用PostgreSQL语句查询分析MaxCompute项目中的数据，快速获取查询结果。

Lightning采用公共集群，多用户之间共用资源。查询大量数据时，会出现性能波动和资源排队问题，并且耗时较长。Lightning不支持通过创建索引查询数据。您可以迁移Lightning服务至Hologres，避免上述问题。

Hologres是兼容PostgreSQL协议的实时交互式分析产品，具有如下优势：

- 用户之间资源独立，查询操作互不干扰。
- 与大数据生态无缝打通，支持高并发和低延时地分析处理PB级数据。
- 在底层与MaxCompute资源无缝打通，支持使用新建外部表的方式加速查询MaxCompute数据。
- 支持通过创建索引查询导入至Hologres的MaxCompute数据，性能相比开源系统大幅提升。
- 提供JDBC/ODBC Driver，支持对接各种BI工具，多维分析数据。

原Lightning用户默认已具备Hologres共享集群的开通资格，若购买时出现如下提示信息。请先申请开通资格，审核通过后，便可开通Hologres共享集群。

对比Hologres与Lightning的产品特性

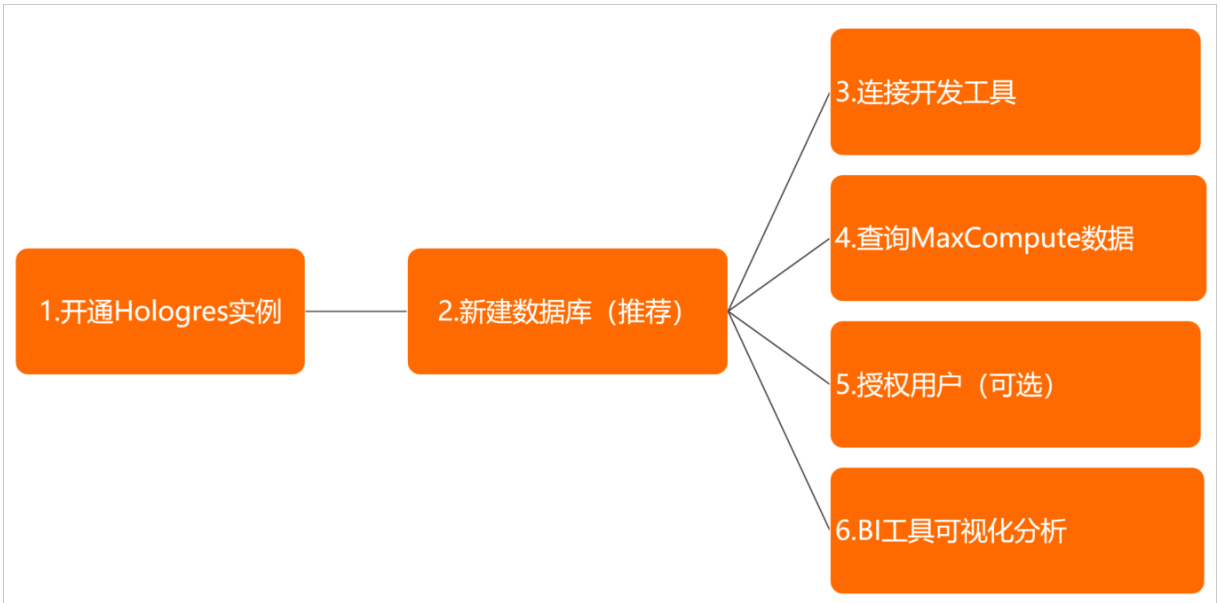
Hologres与Lightning的产品特性对比如下。

类别	对比项	Hologres	Lightning
功能	支持的语法	PostgreSQL11	PostgreSQL8.2
	是否支持使用内部表查询数据	支持	不支持
	是否支持写入实时数据	支持	不支持
	是否支持使用外部表查询数据	支持	支持

类别	对比项	Hologres	Lightning
优化参数	是否支持定向调整参数	支持，例如您可以根据实际业务调整数据库或用户的相关参数。	不支持
	实例连接开发工具的数量（QPS）。	随资源规模线性增加	固定为20个
内部表	是否支持使用DDL语句增加、修改或删除表	支持	不支持
	是否支持更新数据	支持	不支持
	是否支持实时写入实时数据	支持	不支持
外部表	加载Meta模块的速度	加载Meta模块的速度快，实现查询数据的响应速度达到毫秒级。	加载Meta模块的速度随MaxCompute表的数量线性增加，导致查询数据速度较慢。
	数据类型	支持的数据类型如下： • 基本类型 • Decimal2.0 • Datetime • Array类型	仅支持基本类型
性能	引擎	• 支持使用外部表加速查询数据。 • 使用内部表查询数据的性能大幅优于开源系统。	不涉及
	资源	用户之间资源独立，查询操作性能稳定。	用户之间共享资源，查询数据时容易出现延时和性能波动。

操作步骤

迁移Lightning至Hologres进行数据开发的操作流程如下图所示。



1. 开通Hologres实例。

   迁移Lightning前，您需要开通Hologres实例，详情请参见购买Hologres。

   > **说明** Hologres实例与MaxCompute实例的地域必须保持一致。

   成功开通实例后，您可以前往Hologres管理控制台查看实例信息。

2. 新建数据库。

   开通Hologres实例后，系统自动创建**postgres**数据库。该数据库分配到的资源较少，仅用于运维管理。您需要新建数据库来处理实际业务。

   i. 进入Hologres管理控制台，单击**实例列表**。

   ii. 在**实例列表**列表，单击已购买的实例名称，进入实例详情页。

   iii. 单击左侧菜单栏的**数据库管理**，进入**DB授权**页面。

   iv. 单击**新增数据库**，输入**数据库名称**，**简单权限策略**选择**SPM**，单击**确认**。

3. 连接开发工具。

   成功新建数据库后，您需要连接Hologres实例至原有的开发工具。修改原Lightning服务的端口为对应Hologres实例的端口，网络地址使用公网地址。

   进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。

   连接开发工具操作如下：

   - 使用DataWorks开发数据，详情请参见DataWorks快速入门。
   - 使用Psql客户端开发数据，详情请参见PSQL客户端。
   - 使用JDBC开发数据，详情请参见JDBC。
   - 使用HoloWeb开发数据，详情请参见连接HoloWeb。

4. 查询MaxCompute数据。

   您可以通过新建外部表或导入MaxCompute数据至Hologres，加速查询MaxCompute的数据。操作如下：

   - 新建外部表。

示例为使用HoloStudio新建外部表。

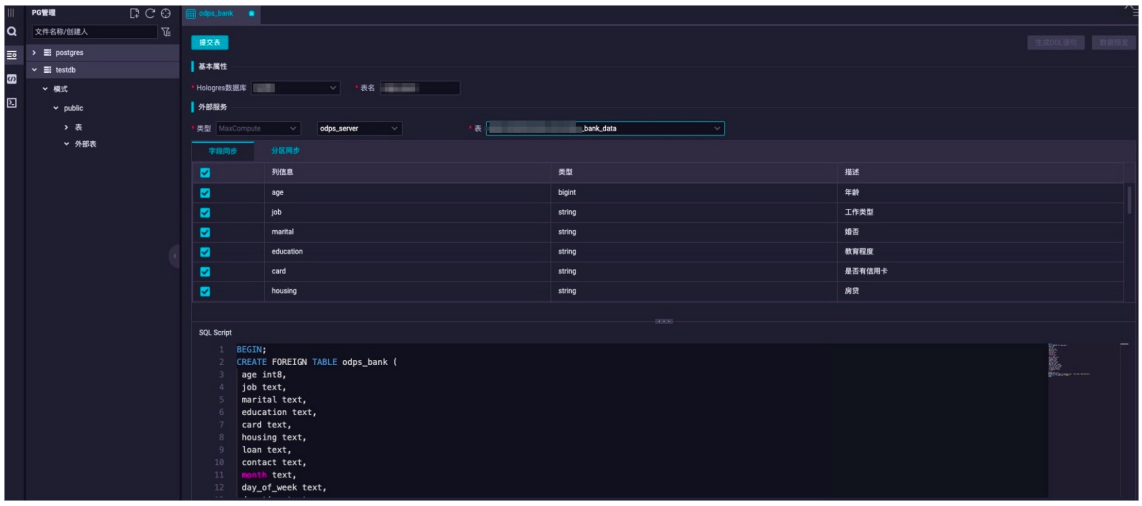
a. 进入Hologres管理控制台，选择开通实例的地域，单击**前往DataStudio**。

b. 进入HoloStudio开发界面，单击左侧菜单栏的**PG管理**，鼠标悬停至图标，单击**外部表**。

c. 配置外部表参数，单击**提交表**。

参数名称	描述
Hologres数据库	当前连接的Hologres数据库名称。
表名	当前Hologres数据库的表名称。
类型	外部表的类型。
表	外部表的名称。

输入MaxCompute表名称后即可索引出当前MaxCompute表的字段，您可以根据需要选择查询部分或全部字段。详情请参见通过创建外部表加速查询MaxCompute数据。



- 导入MaxCompute数据至Hologres。

  如果您单次需要查询的MaxCompute表数据超过100 GB，建议您导入MaxCompute表数据至Hologres后再进行查询。详情请参见使用SQL导入MaxCompute的数据至Hologres。

5. （可选）授权子账号。

   授权子账号使用Hologres，详情请参见RAM用户权限授权快速入门。

6. 可视化分析数据。

   您需要修改原Lightning服务的端口为Hologres实例的端口，使用如下BI工具通过可视化方式分析数据：

   - 使用DataWorks数据服务分析数据，详情请参见DataWorks数据服务。
   - 使用Tableau分析数据，详情请参见Tableau。
   - 使用Quick BI分析数据，详情请参见Quick BI。

   注意：

   对接BI等工具时，在链接信息处往往需要填写Schema信息。使用Lightning时，Schema处应填写MaxCompute的Project Name，但是迁移至Hologres后，需要填写创建外部表所对应的Schema名称。

例如下图为Quick BI的数据源连接配置，其对应关系如下表所示。

参数	Lightning	Hologres
显示名称	自定义的显示名称。	自定义的显示名称。
数据库地址	Lightning服务的Endpoint。	连接的Hologres实例的服务器地址。您可以登录管理控制台，进入实例详情页，从实例配置页面获取。
端口	Lightning服务的端口。	连接的Hologres实例的端口。您可以登录管理控制台，进入实例详情页，从实例配置页面获取。
数据库	MaxCompute的Project名称。	连接的Hologres实例的数据库名称。您可以登录管理控制台，进入实例详情页，从Database管理页面获取。
Schema	MaxCompute的Project名称。	默认为public Schema。您也可以使用新创建的Schema，在数据源配置了Schema后，列表中会展示当前Schema下所有的表。但是在使用即席SQL时，还需要在表名称前手动添加对应的Schema名（即shcema.table），才能正确索引到Schema下面的表。
用户名	具有当前数据库登录权限账号的AccessKey ID。您可以单击AccessKey 管理，获取AccessKey ID。	具有当前数据库登录权限账号的AccessKey ID。您可以单击AccessKey 管理，获取AccessKey ID。
密码	具有当前数据库登录权限账号的AccessKey Secret。您可以单击AccessKey 管理，获取AccessKey Secret。	具有当前数据库登录权限账号的AccessKey Secret。您可以单击AccessKey 管理，获取AccessKey Secret。

9.2. 迁移MySQL至Hologres

本文将为您介绍，MySQL数据平滑迁移至Hologres的操作方法，以及迁移完成后MySQL与Hologres查询语句与函数的使用区别，方便您更加快速的完成数据迁移。

数据迁移方法

下表将根据您的迁移类别，为您介绍该类别适用的场景以及迁移方法。

> **说明** 如存在ETL处理场景，您可通过Flink读取MySQL数据再写入Hologres，详情请参见Hologres结果表。

迁移类别	适用场景	使用文档
单表离线同步	适用于MySQL单表数据离线同步至Hologres的场景。	MySQL单表离线同步至Hologres
单表实时同步	通过开启MySQL Binlog，将单表数据实时同步至Hologres。	MySQL单表实时同步至Hologres
整库实时同步	将MySQL数据库整库实时同步至Hologres。	MySQL整库实时同步至Hologres
同步解决方案	数据集成支持同步解决方案功能，您可以通过配置同步规则，一次性实时同步数据至对应的数据源中。 同步解决方案支持整库内批量同步多张表，也支持全量、增量数据一体化同步（先同步全量数据，再实时同步增量数据）。	同步解决方案至Hologres

数据类型映射关系

您可参见下表，查看MySQL中的数据迁移至Hologres后对应的数据类型映射关系，更多数据类型请参见数据类型汇总。

MySQL迁移至Hologres时，数据类型映射需注意如下事项：

- Hologres中有3种整型（SMALLINT（2 Bytes）、INTEGER（4 Bytes）、BIGINT（8 Bytes）），而MySQL中有5种整型（TINYINT（1 Byte）、SMALLINT（2 Bytes）、MEDIUMINT（3 Bytes）、INT（4 Bytes）、BIGINT（8 Bytes）），此时您需选择Bytes数更高的类型进行映射。
- Hologres不支持无符号整型，在进行数据类型映射时需考虑无符号字段造成的数据溢出，如超出对应字段范围则需考虑映射更大范围的整型。
- 您可使用Hologres的TEXT类型，替换MySQL中的TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT类型。
- 浮点类型（DECIMAL、NUMERIC、DOUBLE、FLOAT）可直接映射。
- MySQL中的DATETIME类型（不含时区信息，格式为YYYY-MM-DD HH:MM:SS）对应Hologres中的TIMESTAMP类型（TIMESTAMP WITHOUT TIME ZONE）。

MySQL中的数据类型	迁移至Hologres后对应的数据类型
BIGINT	BIGINT

MySQL中的数据类型	迁移至Hologres后对应的数据类型
BIGINT(20) UNSIGNED	TEXT
`BINARY(n)`	BYTEA
BIT	BOOLEAN
`CHAR(n)` 、 `CHARACTER(n)`	`CHAR(n)` 、 `CHARACTER(n)`
DATE	DATE
DATETIME	`TIMESTAMP [WITHOUT TIME ZONE]`
`DECIMAL(p,s)` 、 `DEC(p,s)`	`DECIMAL(p,s)` 、 `DEC(p,s)`
DOUBLE	DOUBLE PRECISION
FLOAT	REAL
INT、INTEGER	INT、INTEGER
MEDIUMINT	INTEGER
`NUMERIC(p,s)`	`NUMERIC(p,s)`
SMALLINT	SMALLINT
TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB	BYTEA
TINYINT	SMALLINT
TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT	TEXT
TIME	`TIME [WITHOUT TIME ZONE]`
TIMESTAMP	`TIMESTAMP [WITH TIME ZONE]`
`VARBINARY(n)` 、 `VARBINARY(max)`	BYTEA
`VARCHAR(n)`	`VARCHAR(n)`
`VARCHAR(max)`	TEXT

查询语法

MySQL和Hologres的查询语法在使用中有部分差异，具体内容如下。

- **引号**

  Hologres对大小写不敏感，如需区分大小写请添加英文双引号（""）。

  例如，将 `select `A` from b` 替换为 `select "A" from b` 。

- **条件筛选**

条件筛选时存在类型不匹配的情况，Hologres要求条件筛选的类型必须完全匹配且默认不做隐式类型转换。具体示例如下：

- 示例代码：

```
select * from business_module where ds = 20210329;
```

- 问题描述：

  如果*ds*在Hologres表里是TEXT类型，而*20210329*是INTEGER类型，则这个语句会直接提示类型不匹配的错误。错误提示如下。

```
operator does not exist: text = integer;
```

- 解决方案：

  Hologres支持创建自定义类型转换，您可参见如下代码创建转换规格。

```
CREATE CAST (TEXT  AS INTEGER) WITH INOUT AS IMPLICIT;
CREATE CAST (TEXT  AS BIGINT) WITH INOUT AS IMPLICIT;
CREATE CAST (TEXT  AS DECIMAL) WITH INOUT AS IMPLICIT;
CREATE CAST (TEXT  AS TIMESTAMP ) WITH INOUT AS IMPLICIT;
CREATE CAST (NUMERIC  AS TEXT ) WITH INOUT AS IMPLICIT;
```

- **分页**

  MySQL中的分页语法为 `limit 0,10` ，迁移至Hologres后的标准语法为 `offset 0 limit 10` 。

- **排序**

  MySQL的排序行为是 `desc nulls first asc nulls first` ，而Hologres排序的默认行为是 `desc nulls first asc nulls last` 。

  为保证使用体验一致，请将Hologres查询语句手动调整为 `order by XXX desc nulls last` 。

- **分组**

  Hologres默认不支持FLOAT、DOUBLE等非精确类型的GROUP BY，您可将类型更改为DECIMAL类型，或通过如下参数进行配置。

  > **说明** 下述内容需要您的Hologres版本为0.10及以上版本，如您的版本低于该要求，请提交工单由专业人员为您升级实例。

```
set hg_experimental_enable_double_equivalent=on;--session级别
alter database XXX set hg_experimental_enable_double_equivalent=on;--整个库生效
```

- **Union**

  Union要求列的字段类型必须完全匹配。示例如下。

  - 示例代码：

```
SELECT project_id FROM tableA union ALL  select project_id from tableB;
```

- 问题描述：

  如*project_id*在*tableA*中是BIGINT类型，*project_id*在*tableB*中是TEXT类型。这类SQL在MySQL里会做隐式转换正常返回结果，在Hologres里执行则会提示异常。异常语句如下。

  ```
  UNION types bigint and text cannot be matched;
  ```

- 解决方案：

  Union操作需要显式的做类型转换。

  ```
  SELECT project_id FROM tableA union ALL  select cast(project_id as bigint) from tableB;
  ```

函数使用

Hologres已兼容PostgreSQL的大部分函数，详情请参见PostgreSQL兼容函数。MySQL和Hologres的函数在使用中有部分差异，具体内容如下。

- **除数为0**
  - 问题描述：

    MySQL里除数为*0*时会返回*NULL*值，而在Hologres中会提示如下错误。

    ```
    ERROR: division by zero;
    ```

  - 解决方案：

    ```
    select a/ b from table;
    转换为 select a/ NULLIF(b,0) from table;
    ```

- **整数相除**
  - 问题描述：

    两数相除有余数时，MySQL会返回小数点，而Hologres会返回整数舍弃余数。

    例如，5除以2，MySQL会返回2.5，而Hologres会返回2。

  - 解决方案：

    如果需要兼容MySQL的除法，需要显式做类型转换。

    ```
    select 1/2::FLOAT;
    ```

- **IF函数**

  Hologres不支持IF函数，需转换为CASE WHEN函数。

- **IFNULL函数**

  MySQL的IFNULL函数，对应Hologres中的`COALESCE(x,y)`函数。

- **LENGTH函数**

  MySQL中的LENGTH函数，对应Hologres中的`CHAR_LENGTH(string)`函数。

9.3. 迁移ClickHouse至Hologres

本文为您介绍如何迁移自建ClickHouse的数据库表和数据至交互式分析Hologres上进行数据开发。

前提条件

- 开通Hologres，详情请参见购买Hologres。
- 已有ClickHouse实例，并且安装ClickHouse-Client工具，如需安装请单击ClickHouse-Client，安装和使用详情请参见Getting Started。
- 使用PSQL客户端连接交互式分析Hologres实例，详情请参见PSQL客户端。

背景信息

ClickHouse是一个用于联机分析（OLAP）的列式数据库管理系统。Hologres是阿里巴巴自主研发的一款交互式分析产品，支持亚秒级响应与高QPS，您可以从ClickHouse迁移表和数据至Hologres获取更好的数据开发体验。

Hologres与ClickHouse产品特性对比如下。

分类	对比项	Clickhouse	Hologres
产品	定位	流量分析	通用实时数仓：数据分析和在线服务。
写入	存储	列存	列存和行存。
	写入可见性	秒级（需要客户端攒数据进行批处理，分布式表写入依赖Shard数据复制完成）	毫秒级（写入自适应批处理，写入即可查）
	写入性能	高	非常高
	明细存储	支持	支持
	主键（Primary Key）	非数据库主键（不支持唯一性约束，仅用于索引+聚合）	标准数据库主键，支持唯一性约束。
	可更新	不完备，能力弱（不支持基于主键的高QPS更新）。	完整支持（支持基于主键的高QPS更新）。
	实时写入	Append	• Append • insert or ignore • insert or replace • update
	索引	• primary key • minmax • ngram • token • bloom filter	• bitmap • dictionary • segment • primary • clustering **说明** 自动建有minmax、bloom filter、ngram等索引，对用户透明。
	优化器	RBO（Rule-Based Optimizer）	CBO（Cost-Based Optimizer）

分类	对比项	Clickhouse	Hologres
查询	联邦查询	支持（Engine支持HDFS、Kafka）	支持（FDW直读MaxCompute、Hive）
	预聚合	支持（通过MergeTree）	支持（存储过程+定期调度）
	高QPS点查	不支持	支持，QPS可达千万以上。
	单表复杂查询	性能好	性能好
	多表JOIN	性能差	性能好
	SQL语法	自定义语法	兼容PostgreSQL，功能更丰富。
	WINDOW FUNCTION	不支持	支持
事务	事务ACID	无（不保证写入即可查，最终一致性）	有限支持（支持DDL事务、单行事务、基于snapshot的可见性）
复制	容灾和备份	通过Replication实现（远程ZK+CK）	通过Binlog复制实现逻辑复制，通过底层机制实现物理复制。
高级功能	Binlog	无	提供Binlog
	向量检索	不支持	支持
	空间数据	不支持	支持
	安全管理	自定义权限	兼容PG权限模型、丰富的权限控制、IP白名单、数据脱敏。
	存储计算分离	不分离，单机容量限制	分离，存储容量近乎无限。
	可用性	用户手工处理Failover	Failover自动恢复
	运维	复杂（手工维护Shard分布）	免运维
生态	数据接入	Kafka、Flink、Spark、...	Flink、Spark、JDBC、DataX、...
	BI工具	支持对接少量BI工具（Tableau、Superset、...）	兼容PostgreSQL生态，支持对接100+主流BI工具。

数据类型映射

ClickHouse与Hologres的数据类型映射如下表所示。

类别	ClickHouse	Hologres
日期	Date	Date
	DateTime	TIMESTAMPTZ
	DateTime(timezone)	TIMESTAMPTZ

类别	ClickHouse	Hologres
	DateTime64	TIMESTAMPTZ
数值	Int8	不支持单字节INT，可选SMALLINT。
	Int16	SMALLINT
	Int32	INT
	Int64	BIGINT
	UInt8	INT
	UInt16	INT
	UInt32	BIGINT
	UInt64	BIGINT
	Float32	FLOAT
	Float64	DOUBLE PRECISION
	Decimal(P, S)	DECIMAL
	Decimal32(S)	DECIMAL
	Decimal64(S)	DECIMAL
	Decimal128(S)	DECIMAL
布尔	无，使用UInt8代替。	BOOLEAN
字符	String	TEXT
	FixString(N)	无，使用TEXT代替。
	LowCardinality	无，自动智能设定或使用 `call set_table_properties('x', 'dictionary_encoding_columns', 'col');` 命令进行设置。
二进制	无，使用String或FixString(N)。	BIT(n)、VARBIT(n)、BYTEA、CHAR(n) 等数据类型。
其他	UUID	UUID
	Enum	不支持，使用TEXT代替。
	Nested、 Tuple、Array	数组

元数据迁移

元数据的迁移，主要指进行建表DDL的迁移。

1. 在ClickHouse-Client使用如下命令语句查看源ClickHouse实例的数据库列表。

   > **说明** 查询到的表中system是系统数据库，不需要迁移，可以直接过滤掉。

   ```
   clickhouse-client --host="<host>" --port="<port>" --user="<username>" --password="<pass
   word>" --query="SHOW databases"  > database.list;
   ```

   参数说明如下。

参数	说明
host	ClickHouse源实例的地址。
port	ClickHouse源实例的端口。
username	登录ClickHouse源实例的账号，拥有DML读写和设置权限，允许DDL权限。
password	登录ClickHouse源实例账号的密码。

2. 在ClickHouse-Client使用如下命令语句查看源ClikHouse实例的数据表列表。

   > **说明** 查询到的表中，如果有以.inner.开头的表，此类表是物化视图的内部表，不需要迁移，可以直接过滤掉。

   ```
   clickhouse-client --host="<host>" --port="<port>" --user="<username>" --password="<pass
   word>" --query="SHOW tables from <database_name>" > table.list;
   ```

   参数说明如下。

参数	说明
host	源ClickHouse实例的地址。
port	源ClickHouse实例的端口。
username	登录源ClickHouse实例的账号，拥有DML读写和设置权限，允许DDL权限。
password	登录源ClickHouse实例账号的密码。
database_name	源ClickHouse实例迁移表所在的数据库名称。

   您也可以通过以下命令语句查询源ClickHouse实例所有的数据库和表名称。

   ```
   select distinct database, name from system.tables where database != 'system';
   ```

3. 在ClickHouse-Client使用如下命令语句导出ClickHouse源实例的建表DDL。

   ```
   clickhouse-client --host="<host>" --port="<port>" --user="<username>" --password="<pass
   word>" --query="SHOW CREATE TABLE <database_name>.<table_name>"  > table.sql;
   ```

   您也可以使用如下命令直接查看system.tables元数据表。

```
SELECT * FROM system.tables
where database = '<database_name>' and engine != 'Distributed';
```

system.tables中字段的转换说明如下。

字段	说明
database	ClickHouse的database映射到Hologres（PostgreSQL语法）的Schema概念，即ClickHouse的 `create database "<database_name>";` 命令映射为Hologres的 `create schema "<schema_name>";` 命令。
name	表名称，无需改动。
engine	Hologres没有Distributed表概念，没有Local和Distributed之分，就是一个单表，分布式存储和查询，所以需要过滤掉 `engine='Distributed'` 的表。
is_temporary	Temporary表逻辑上无须迁移，同时Hologres暂不支持Temporary表。 | ○ data_paths ○ metadata_path ○ metadata_modification_time	可忽略。
○ dependencies_database ○ dependencies_table	常见于View、Materialized View。具有dependencies的View，在Hologres中，需要先于base表创建。Hologres的Materialized View还未支持。
create_table_query	ClickHouse源实例表的DDL，需转换成Hologres DDL（PostgreSQL语法）。
engine_full	Engine详细信息，可忽略。
partition_key	对应Hologres的分区列，ClickHouse的源实例partition_key如果为col1，则Hologres建表语句后添加 `partition by list (col1);` 语句。
sorting_key	对应Hologres的Segment Key和Clustering Key索引。
primary_key	主键，对应Hologres DDL语法的Primary Key。
sampling_key	Hologres DDL不支持采样。
storage_policy	存储策略，可忽略。

4. 将源ClickHouse实例的建表DDL转换为Hologres的语法（兼容PostgreSQL SQL标准）。

   根据system.tables中字段的转换说明和数据类型映射进行DDL转换，示例如下。

   ○ 将名称为lineitem表在ClickHouse实例中的DDL转换为Hologres的建表DDL。

- ClickHouse实例的建表DDL如下所示。

```
-- lineitem on ClickHouse
CREATE TABLE lineitem_local ON CLUSTER default(
  l_orderkey            UInt64,
  l_partkey             UInt32,
  l_suppkey             UInt32,
  l_linenumber          UInt32,
  l_quantity            decimal(15,2),
  l_extendedprice       decimal(15,2),
  l_discount            decimal(15,2),
  l_tax                 decimal(15,2),
  l_returnflag          LowCardinality(String),
  l_linestatus          LowCardinality(String),
  l_shipdate            Date,
  l_commitdate          Date,
  l_receiptdate         Date,
  l_shipinstruct        LowCardinality(String),
  l_shipmode            LowCardinality(String),
  l_comment             LowCardinality(String)
) ENGINE = MergeTree
PARTITION BY toYear(l_shipdate)
ORDER BY (l_orderkey, l_linenumber);
CREATE TABLE lineitem on cluster default as lineitem_local ENGINE = Distributed(def
ault, default, lineitem_local, l_orderkey);
```

- 转换后Hologres实例的建表DDL如下所示。

```
-- lineitem on Hologres
-- create a table group with 32 shards
CALL hg_create_table_group ('lineitem_tg', 32);
BEGIN;
CREATE TABLE LINEITEM
(
    L_ORDERKEY      BIGINT         NOT NULL,
    L_PARTKEY       INT         NOT NULL,
    L_SUPPKEY       INT         NOT NULL,
    L_LINENUMBER    INT         NOT NULL,
    L_QUANTITY      DECIMAL(15,2) NOT NULL,
    L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
    L_DISCOUNT      DECIMAL(15,2) NOT NULL,
    L_TAX           DECIMAL(15,2) NOT NULL,
    L_RETURNFLAG    TEXT        NOT NULL,
    L_LINESTATUS    TEXT        NOT NULL,
    L_SHIPDATE      TIMESTAMPTZ NOT NULL,
    L_COMMITDATE    TIMESTAMPTZ NOT NULL,
    L_RECEIPTDATE   TIMESTAMPTZ NOT NULL,
    L_SHIPINSTRUCT  TEXT        NOT NULL,
    L_SHIPMODE      TEXT        NOT NULL,
    L_COMMENT       TEXT        NOT NULL,
    PRIMARY KEY (L_ORDERKEY,L_LINENUMBER)
);
CALL set_table_property('LINEITEM', 'clustering_key', 'L_SHIPDATE,L_ORDERKEY');
CALL set_table_property('LINEITEM', 'segment_key', 'L_SHIPDATE');
CALL set_table_property('LINEITEM', 'table_group', 'lineitem_tg');
CALL set_table_property('LINEITEM', 'distribution_key', 'L_ORDERKEY');
-- columns with LowCardinality
CALL set_table_property('LINEITEM', 'bitmap_columns', 'L_RETURNFLAG,L_LINESTATUS,L_
SHIPINSTRUCT,L_SHIPMODE,L_COMMENT');
-- columns with LowCardinality
CALL set_table_property('LINEITEM', 'dictionary_encoding_columns', 'L_RETURNFLAG,L_
LINESTATUS,L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT');
CALL set_table_property('LINEITEM', 'time_to_live_in_seconds', '31536000');
COMMIT;
```

- 将名称为customer表在ClickHouse实例中的DDL转换为Hologres的建表DDL。

- ClickHouse实例的建表DDL如下所示。

```
-- customer on ClickHouse
CREATE TABLE customer_local ON CLUSTER default(
  c_custkey             UInt32,
  c_name                String,
  c_address             String,
  c_nationkey           UInt32,
  c_phone               LowCardinality(String),
  c_acctbal             decimal(15,2),
  c_mktsegment          LowCardinality(String),
  c_comment             LowCardinality(String)
) ENGINE = MergeTree
ORDER BY (c_custkey);
CREATE TABLE customer on cluster default as customer_local
ENGINE = Distributed(default, default, customer_local, c_custkey);
```

- 转换后Hologres实例的建表DDL如下所示。

```
-- customer on Hologres
BEGIN;
CREATE TABLE CUSTOMER (
    C_CUSTKEY    INT    NOT NULL PRIMARY KEY,
    C_NAME       TEXT   NOT NULL,
    C_ADDRESS    TEXT   NOT NULL,
    C_NATIONKEY  INT    NOT NULL,
    C_PHONE      TEXT   NOT NULL,
    C_ACCTBAL    DECIMAL(15,2) NOT NULL,
    C_MKTSEGMENT TEXT   NOT NULL,
    C_COMMENT    TEXT   NOT NULL
);
CALL set_table_property('CUSTOMER', 'distribution_key', 'C_CUSTKEY');
CALL set_table_property('CUSTOMER', 'table_group', 'lineitem_tg');
CALL set_table_property('CUSTOMER', 'bitmap_columns', 'C_CUSTKEY,C_NATIONKEY,C_NAME
,C_ADDRESS,C_PHONE,C_MKTSEGMENT,C_COMMENT');
CALL set_table_property('CUSTOMER', 'dictionary_encoding_columns', 'C_NAME,C_ADDRES
S,C_PHONE,C_MKTSEGMENT,C_COMMENT');
CALL set_table_property('CUSTOMER', 'time_to_live_in_seconds', '31536000');
COMMIT;
```

5. 在PSQL客户端使用如下命令语句将转换后的建表DDL导入到目标Hologres实例中。

```
PGUSER="<username>" PGPASSWORD="<password>" psql -h "<host>" -p "<port>" -d "<database_
name>" -f table.sql;
```

数据迁移

从源ClickHouse迁移数据至Hologres有如下三种方法。

- （推荐）在源实例将数据导出为文件，然后通过 `COPY语句` 命令语句（JDBC/PSQL）将文件导入到Hologres目标实例。
- 通过编写Flink、Spark job将源实例数据读出，然后写入目标Hologres实例，请参见Spark的数据写入至Hologres。
- 通过DataWorks数据集成或DataX，读取源实例数据，后写入目标Hologres实例，请参见数据集成概述。

在源实例将数据导出为文件，再导入目标Hologres实例，操作步骤如下。

1. 在ClickHouse-Client使用如下命令语句导出源实例数据至本地CSV文件。

```
clickhouse-client --host="<host>" --port="<port>" --user="<username>" --password="<pass
word>"  --query="select * from <database_name>.<table_name> FORMAT CSV"  > table.csv;
```

参数说明如下。

参数	说明
host	ClickHouse源实例的地址。
port	ClickHouse源实例的端口。
username	登录ClickHouse源实例的账号，拥有DML读写和设置权限，允许DDL权限。
password	登录ClickHouse源实例账号的密码。
database_name	ClickHouse源实例迁移表所在的数据库名称。
table_name	ClickHouse源实例迁移的表名称。

2. 在PSQL客户端使用如下命令语句将本地CSV文件导入到Hologres实例。

```
PGUSER="<username>" PGPASSWORD="<password>" psql -h "<host>" -p "<port>" -d "<database_
name>" -c "COPY <schema_name>.<table_name> FROM STDIN (FORMAT 'csv')" < table.csv;
```

参数说明如下。

参数	说明
username	登录Hologres目标实例的账号，拥有DML读写和设置权限，允许DDL权限。通常是阿里云账号的AccessKey ID，您可以单击AccessKey 管理，获取AccessKey ID。
password	登录Hologres目标实例账号的密码，通常是阿里云账号的AccessKey Secret，您可以单击AccessKey 管理，获取AccessKey Secret。
host	Hologres实例的服务器地址。 您可以登录管理控制台，进入实例详情页，从**实例配置**页面获取。
port	Hologres实例的端口。
database_name	迁移到Hologres实例的数据库名称。
schema_name	迁移到Hologres实例的Schema名称，不填默认为public。
table_name	迁移到Hologres实例的表名称。

3. 在Hologres中查询导入数据，验证数据是否导入成功。

数据查询语句迁移

Hologres的数据查询语句采用PostgreSQL语法，ClickHouse为自创语法，部分兼容SQL ANSI，两者语法基本类似，但有细节上的差异。所以需要对数据查询语句进行迁移，常见的是SQL中使用函数名的迁移，例如Scalar函数、Window函数等。

ClickHouse和Hologres的SQL有如下差别。

- ClickHouse中用 `''` 包围的列名，在Hologres中需要替换成 `""` 包围。
- ClickHouse中使用 `SELECT X FROM <database_name>.<table_name>` 命令，在Hologres中使用 `SELECT X FROM <schema_name>.<table_name>` 命令。
- 表达式差异，主要表现在函数上，函数映射表（仅列出有差异的函数，未列出则无差异）如下所示。

ClickHouse	Hologres
toYear(expr)	to_char(expr, 'YYYY')
toInt32(expr)	CAST(expr as INTEGER)
○ uniq() ○ uniqCombined() ○ uniqCombined64() ○ uniqHLL12()	approx_count_distinct()
uniqExact()	count(distinct x)
quantile(level) (expr)	approx_percentile(level) WITH GROUP(ORDER BY expr)
quantileExact(level) (expr)	percentile_cont (level) WITH GROUP(ORDER BY expr)

数据查询语句迁移有如下方法。

- 正则替换

  采用正则替换的方式，将ClickHouse的一些固定模式的语法（函数名、标志符、表达式等）转换成Hologres的语法，例如将 `''` 转换为 `""` 。

- ClickHouse Extension

  Hologres中具备ClickHouse Extension，兼容部分ClickHouse函数，无需转换，例如 `toUInt32()` 函数。

下面以部分TPC-H Query为例，将ClickHouse源实例的数据查询语句迁移至Hologres的示例如下所示。

- 示例一。

- 在ClickHouse实例上的数据查询语句。

```
-- Q1 on ClickHouse
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= date '1998-12-01' - interval '90' day
group by
  l_returnflag,
  l_linestatus
order by
  l_returnflag,
  l_linestatus;
```

- 转换后Hologres实例的数据查询语句。

```
-- Q1 on Hologres
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= date '1998-12-01' - interval '90' day
group by
  l_returnflag,
  l_linestatus
order by
  l_returnflag,
  l_linestatus;
```

- 示例二。

- 在ClickHouse实例上的数据查询语句。

```
-- Q4 on ClickHouse
select
  o_orderpriority,
  count(*) as order_count
from
  orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and o_orderdate in (
    select
      o_orderdate
    from
      lineitem,
      orders
    where
      l_orderkey = o_orderkey
      and l_commitdate < l_receiptdate
  )
group by
  o_orderpriority
order by
  o_orderpriority;
```

- 转换后Hologres实例的数据查询语句。

```
-- Q4 on Hologres
select
  o_orderpriority,
  count(*) as order_count
from
  orders
where
  o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and exists (
    select
      *
    from
      lineitem
    where
      l_orderkey = o_orderkey
      and l_commitdate < l_receiptdate
  )
group by
  o_orderpriority
order by
  o_orderpriority;
```

- 示例三。

- 在ClickHouse实例上的数据查询语句。

```
-- Q11 on ClickHouse
select
  ps_partkey,
  sum(ps_supplycost * ps_availqty) as value
from
  partsupp,
  supplier,
  nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'GERMANY'
group by
  ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select
        sum(ps_supplycost * ps_availqty) * toDecimal32(0.0000010000,9)
      from
        partsupp,
        supplier,
        nation
      where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY'
    )
order by
  value desc
  limit 100;
```

- 转换后Hologres实例的数据查询语句。

```
-- Q11 on Hologres
select
  ps_partkey,
  sum(ps_supplycost * ps_availqty) as value
from
  partsupp,
  supplier,
  nation
where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'GERMANY'
group by
  ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
      select
        sum(ps_supplycost * ps_availqty) * 0.0000010000
      from
        partsupp,
        supplier,
        nation
      where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY'
    )
order by
  value desc
  limit 100;
```

ClickHouse扩展

Hologres中具备ClickHouse extension，兼容部分ClickHouse函数，具体如下。

函数	描述	返回类型
toString(anyelement)	将类型转换为TEXT类型。	TEXT
toInt64(anyelement)	将类型转换为BIGINT类型。	BIGINT
toInt32(anyelement)	将类型转换为INT类型。	INT
toDate(text)	将TEXT类型转换为DATE类型。	DATE
toFloat64(anyelement)	将类型转换为DOUBLE类型。	INT

- 使用方法

  扩展函数存放在ClickHouse extension中，使用之前需要先创建extension，使用命令如下。

  > 说明 extension需要Superuser角色创建。

```
--创建扩展
CREATE  extension clickhouse;
--使用函数
SELECT <function> from tablename;
```

- 使用示例

```
CREATE EXTENSION clickhouse;--创建扩展，若有则无需创建
CREATE TABLE public.tb1 (
    id          bigint NOT NULL,
    data_date   text
    ,PRIMARY KEY (id)
);
INSERT INTO public.tb1 VALUES (1234,'20190102');
SELECT toString(id) from public.tb1;
 tostring
----------
 1234
(1 row)
SELECT toDate(data_date) from public.tb1;
   todate
------------
 2019-01-02
(1 row)
```

9.4. 迁移HBase至Hologres

本文为您介绍如何迁移HBase的语法和数据至Hologres上进行数据开发。

背景信息

Hologres支持行存表模式，在该模式下，提供高性能基于主键的点查能力，广泛应用在Flink维度表、ID-Mapping、标签获取、订单明细查询等场景中。相比传统的HBase等技术，Hologres保留了横向扩展能力和高性能的主键点查能力，同时解决了Schema Free带来的数据质量管理难题，也通过更少的外部依赖提升了系统的稳定性。支持宽表设计、多流合并、前缀扫描（Prefix Scan）等多种HBase常用场景。支持针对HBase接口平滑迁移的SDK。您可以从HBase迁移表和数据至Hologres获取更好的数据开发体验。

Hologres与HBase对比

Hologres与HBase产品特性对比如下。

能力	HBase	Hologres
产品定位	分布式面向列簇的开源数据库。	云原生分布式实时数仓。
系统架构	存储计算耦合，存储依赖底层Hadoop分布式文件系统（HDFS），HDFS集群需要手动扩容。HBase表根据Region大小进行分区，分别存储在集群不同的节点上。	计算节点采用海量并行处理（MPP，Massively Parallel Processing）架构，基于存储计算分离（Storage Disaggregation），计算和存储资源独立扩展，支持根据不同业务场景对计算能力和存储空间进行配比，更加灵活、经济。

能力		HBase	Hologres
多态存储		仅支持行存，以 `<rowkey, column, value, timestamp>` 4元组形式存储。	支持行存储，列存储，行列共存多种模式。多态存储可以根据数据访问模式的不同而使用灵活的存储方式。
Schema表达能力		弱Schema，弱数据类型。	强Schema，丰富的数据类型。强Schema可以保证开发的效率，在数据质量不可靠，数据接口不明确的情况下，更易于通过Schema排查开发问题。
全局排序		全局排序	局部排序，聚簇索引。
分片策略		支持预先分片和自动分片的模式。	支持哈希(Hash)和随机分片的模式。
批量导入		支持，BulkLoad接口。	支持，BulkLoad接口(COPY)。
实时写入		支持，写入即可查。写入系统吞吐量（TPS）受限于Compaction性能。	支持，写入即可查。高TPS写入。
实时更新		支持。	支持。
SQL支持		通过Phoenix扩展支持，功能弱，不支持Join。受限于键值对（KV）存储模式，SQL性能弱。	高度兼容PostgreSQL：协议兼容、语法兼容、生态兼容。
存储能力		基于HDFS，用户自行维护集群，集群会自动同步数据至多副本，存储能力与集群规模有关，支持线性扩展，LSM-Tree数据结构，多种压缩算法。	基于分布式文件系统Pangu/HDFS，存储能力与集群规模有关，支持线性扩展，单表最大容量3PB+；多种存储模式和多种压缩算法赋能存储。
查询及分析能力		原生仅支持点查(GET)和扫描(SCAN)。点查每秒查询率（QPS，Queries per second）高，SCAN性能弱。Phoenix SQL通过Coprocessor支持，性能弱，不支持复杂计算，不支持联机分析处理（OLAP，Online Analytical Processing）场景。	亿级以上数据，实时查询及分析亚秒级响应；Join能力强大。点查QPS高。支持OLAP场景。
联邦计算		不支持。	支持离线实时联邦计算，支持MaxCompute、OSS等外部表查询加速。
备份及容灾		支持，在数据方面，具备大数据标准三副本保障。	支持，在数据方面，具备大数据标准三副本保障。
查询语言	查询语言	Java API（需要与其他框架共同使用，例如Apache Phoenix）。	PostgreSQL，无需额外结合其他组件和框架即可使用支持完备的SQL语法，DDL对象也更丰富，支持全Join关联查询。
	DDL	支持（关键字：create，alter，drop，describe，list；对象：namespace，table，column family）。	支持（关键字：create，alter，drop；对象：database，table，view，schema，cast，extension，role，user，user mapping，group）。

能力		HBase	Hologres
	DML	支持（关键字：put，get，scan，delete，truncate）。	支持（关键字：select，insert，update，delete）。
	DCL	支持（关键字：grant，revoke，rollback）。	支持（关键字：grant，revoke，rollback）。
运维		需自行运维。	全托管，系统自动化感知集群的拓扑信息变化，用户侧无感知。
生态		HBase兼容Hadoop生态。	Hologres高度兼容PostgreSQL生态。
适用场景		海量存储，非结构化存储，单点查询性能优异，写密集型数据库。	实时数仓，在线数据服务。联通数据孤岛，海量数据实时查询及分析，弹性扩展集群，完整SQL支持。
开发方式		应用开发复杂，需要将业务分析的指标、维度、表、聚合等概念，转化为存储的KeyValue概念，将应用层查询过滤场景翻译为对Key的字节过滤操作，系统效率严重依赖Key设计的质量。整个系统从数据录入到数据分析查询等复杂多样的场景，依赖应用层对KeyValue基础接口的使用。	应用开发简单，面向Table开发，使用SQL标准语句，适用于复杂多维分析，嵌套查询，关联查询等场景。提供JDBC、ODBC接口，面向数据主题建模开发。从面向指标，面向宽表开发，转化为Hologres面向主题域建模，减少了数据模型在采集端、处理端、分析端的异构信息衰减，减少了数据加工的层次，提高了数据使用的灵活性。

语法迁移

- SQL语法和常见命令映射

  Hologres支持HBase常见的基本语法，并提供SQL接口，以及Hologres Client客户端SDK。

HBase语法	Hologres语法	客户端命令
PUT	`insert into values on conflict`	`HoloClient.put(Put(表名,主键值,数据列的值))`
DELETE	`delete from`	`HoloClient.put(Delete(表名,主键值))`
SCAN	`select from where filter=XX`	`HoloClient.scan(Scan(表名,主键值+主键范围,返回列))`
GET	`select from where PK=XX`	`HoloClient.get(Get(表名,主键值,返回列))`

- 宽表设计

  宽表是HBase最基本的数据模型，在Hologres中使用行存表，有关行存表的设计方案，请参见Key/Value查询场景最佳实践。

  行存表可以支持3000列，列存表的列不建议超过1000列。

由于TEXT类型会默认创建Bitmap索引和字典编码，因此对于超多列场景，不建议使用默认设置，建议手工配置需要的Bitmap索引和字典编码。

- 多流合并

  多流合并是利用Hologres的局部更新能力，多个数据流同时写入单一表，通过表的主键实现Merge即Join的效果。有关多流合并的实践请参见Hologres结果表。

- 前缀扫描

  前缀扫描是利用HBase的前缀顺序特性，高效率过滤数据，在Hologres通过Distribution Key和Clustering Key的设计实现相同的过滤效果。

  假如HBase的rowkey为 `[hash(pk0, pk1), pk0, pk1,..., pkN]` ，Hologres应设置Primary Key为 `(pk0, pk1,..., pkN)` 、Distribution Key为 `(pk0, pk1)` 、Clustering Key与Primary Key保持一致。

  - 那么支持前缀equals扫描的SQL如下：

    ```
    where pk0=? and pk1=? and ... and pkX=? ;
    ```

  - 支持前缀range扫描的SQL如下：

    ```
    where pk0=? and ... and pkX=? and pkX+1 between ? and ?;
    ```

  - 如果表的主键为(rowkey, ts)，查询命令如下：

    ```
    select col1, col2, ts from tbl where rowkey = '1234567890' and ts between 1637637479606
    and 1640229479607 limit 100;
    ```

    这种条件下，适合Distribution key设计为rowkey，Clustering key设计为(rowkey, ts)的组合键。

  - 如果表的主键为 (rowkey, ts)，查询命令如下：

    ```
    select col1, col2, ts from tbl where rowkey = '1234567890' and ts = 1637637479606;
    ```

    这种条件下，适合Distribution key设计为(rowkey, ts), Clustering key设计为(rowkey, ts)的组合键。

- 批量更新

  Hologres支持Insert、Update、Insert on Conflict等多种更新方式，支持行更新、列更新和Upsert等场景，在实现批量更新场景下，写入更新的吞吐量比较大，有可能影响线上对延迟敏感的在线服务，这种情况建议通过写入临时表再原子替换的方式实现批量更新，类似BulkLoad模式，示例如下。

```
--假设线上服务的表为 t_online;
--创建临时表
begin;
create table t_tmp
(
  ...
)
commit;
--向临时表写入数据
insert into t_tmp select * from t_offline;
--更新统计信息
vacumm t_tmp;
analyze t_tmp;
--在事务中，实现原子替换
begin;
alter table t_online rename to t_online_outdate;
alter table t_tmp rename to t_online;
commit;
--延迟5s，让针对老表上的查询结束再删除无用数据
pg_sleep(5);
drop table t_online_outdate;
```

- OLAP

  Hologres不仅支持行存主键点查，也支持OLAP场景，在存储上，可以设计为行列共存模式，在查询时，查询引擎根据查询特征，选择最佳的存储结构，详情请参见CREATE TABLE。

9.5. 迁移工具Holo Shipper

本文为您介绍Holo Shipper工具，此工具支持将Hologres实例的部分表导入导出的Hologres生态离线备份工具。实现在Hologres之间搬迁表和将表转储到中间存储然后再恢复的功能。

使用限制

- 仅Hologres V1.1及以上版本支持Holo Shipper工具，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅限具备Superuser权限的用户使用Holo Shipper工具，否则系统会提示权限不足。
- 使用Holo Shipper工具时，需确保能连接上源表和目标表以及具有Java环境。
- 使用Holo Shipper工具时，源表需要停止写入。

> **说明** 当前Holo Shipper已开源，更多信息请参见Holo Shipper。

命令行参数

- 导出表的源头

  导出表源可以来自于Hologres实例、 OSS或本地存储，命令格式分别如下。

- Hologres实例

```
-s holo -h endpoint -p port_number -u username -w password
```

参数	说明
endpoint	Hologres实例的网络地址。您可以登录Hologres管理控制台，在实例详情页获取。
port_number	Hologres实例的网络端口。您可以登录Hologres管理控制台，在实例详情页获取。
username	登录Hologres实例的账号。
password	登录Hologres实例的账号密码。

- OSS

```
-s oss -h endpoint -u accessKeyId -w accessKeySecret -b bucketName -p folder_path
```

参数	说明
endpoint	OSS存储空间的地域节点。获取OSS的endpoint请参见获取OSS内网地址。
accessKeyId	前访问账号的AccessKey ID。您可以单击用户中心，获取AccessKey ID。
accessKeySecret	前访问账号的AccessKey Secret。您可以单击用户中心，获取AccessKey Secret。
bucketName	OSS存储空间的名称。
folder_path	OSS存储空间下的目录路径。

- 本地存储

```
-s local_path
```

参数	说明
local_path	本地存储的目录路径。

- 导出表的目的

  迁移表的目的可以为Hologres实例、 OSS或本地存储，命令格式分别如下。

- Hologres实例

```
-d holo -h endpoint -p port_number -u username -w password
```

参数	说明
endpoint	Hologres实例的网络地址。您可以登录Hologres管理控制台，在实例详情页获取。
port_number	Hologres实例的网络端口。您可以登录Hologres管理控制台，在实例详情页获取。
username	登录Hologres实例的账号。
password	登录Hologres实例的账号密码。

- OSS

```
-d oss -h endpoint -u accessKeyId -w accessKeySecret -b bucketName -p folder_path
```

参数	说明
endpoint	OSS存储空间的地域节点。获取OSS的endpoint请参见获取OSS内网地址。
accessKeyId	前访问账号的AccessKey ID。您可以单击用户中心，获取AccessKey ID。
accessKeySecret	前访问账号的AccessKey Secret。您可以单击用户中心，获取AccessKey Secret。
bucketName	OSS存储空间的名称。
folder_path	OSS存储空间下的目录路径。

- 本地存储

```
-d local_path
```

参数	说明
local_path	本地存储的目录路径。

- 导出表的数据库和表信息的JSON文件路径。

- 命令格式如下

```
-l ship_list_json_path --no-owner --no-all-roles --no-guc --no-ext --no-priv --no-data
--incl-foreign
```

参数	说明
ship_list_json_path	包含将要迁移的数据库和表信息的JSON文件的路径。
--no-owner	可选参数。不将表的所有权设置为对应源数据库，否则Holo Shiper默认保留表的owner，配置了此参数的情况下如果不是SPM/SPLM模式，表owner则是迁移时提供的用户。
--no-all-roles	可选参数。不迁移源实例的所有用户，只迁移需要的用户（需要迁移表的owner和对表有权限的相关用户）。
--no-guc	可选参数。不同步源数据库的GUC参数，否则默认同步GUC参数。
--no-ext	可选参数。不同步源数据库安装的extension，否则默认同步extension。
--no-priv	可选参数。不同步源表的相关权限，否则默认会同步表的权限。
--no-data	可选参数。不同步表的数据，只创建数据库、表结构等。否则默认同步所有表数据。
--incl-foreign	可选参数。迁移满足shipList条件的外部表的DDL，否则默认忽略外部表。

- JSON文件格式如下。

  JSON文件为一个JSONArray，array中每一个JSONObject代表一个数据库，每个数据库的JSONObject包含如下内容。

参数	说明
dbName	数据库名称。
shipList	JSONObject，其中每个key为schema名称，value为要迁移表的名称。
blackList	可选参数。JSONObject，其中每个key为schema名称，value为不要迁移表的名称。
sinkDB	可选参数。目的地数据库名称， 不提供的话默认和dbName相同。 说明 当源和目的实例为同一个holo实例时可以视为将表从dbName数据库移到sinkDB数据库。
schemaMapping	可选参数。JSONObject，其中每个key为源schema名称， value为在目标的schema名称。如果需要改变schema就在这里指定，如果不指定默认schema不变。

- JSON文件示例

  如下JSON文件，分别迁移DB1、DB2、DB3数据库中的表。迁移DB1数据库schema1中除table3的所有表；迁移schema2中的table1和table2。迁移DB2数据库schema3中的所有表和schema4中的table4和table5，且将schema4更改为schema5。迁移DB3数据库public schema下的所有表到DB3_backup public schema下。

```
[
 {
     "dbName": "DB1",
     "shipList": {
         "schema1": ["*"],
         "schema2": ["table1", "table2"]
     },
     "blackList": {
         "schema1": ["table3"]
     }
 },
 {
     "dbName": "DB2",
     "shipList": {
         "schema3": ["*"],
         "schema4": ["table4", "table5"]
     },
     "schemaMapping": {
         "schema4": "schema5"
     }
 },
 {
     "dbName": "DB3",
     "shipList": {
         "public": ["*"]
     },
     "sinkDB" : "DB3_backup"
 }
]
```

使用示例

为您介绍使用Holo Shipper进行表迁移适用不同场景的命令如下。

- 将一个Hologres实例中的表迁移到另一个Hologres实例中并且不保留表的Owner信息。

```
$ java -jar holo-shipper.jar -s holo -h xx.xx.xx.xx -p xxxx -u username -w password -d ho
lo -h xx.xx.xx.xx -p xxxx -u username2 -w password2  -l ship_list_json_path --no-owner
```

- 将一个Hologres实例中的某些表导出到另一个Hologres实例中，保留源表Owner和权限，不同步与这些表无关的用户，不同步源数据库的GUC参数和extension。

```
$ java -jar holo-shipper.jar -s holo -h xx.xx.xx.xx -p xxxx -u username -w password -d ho
lo -h xx.xx.xx.xx -p xxxx -u username2 -w password2  -l ship_list_json_path --no-all-role
s --no-guc --no-ext
```

- 将一个Hologres实例中的表转储到本地存储，并且不保留GUC参数信息。

```
$ java -jar holo-shipper.jar -s holo -h xx.xx.xx.xx -p xxxx -u username -w password -d lo
cal_storage_path -l ship_list_json_path --no-guc
```

- 将一个之前转储到本地的备份r恢复到另一个Hologres实例。

```
$ java -jar holo-shipper.jar -s local_storage_path -d holo -h xx.xx.xx.xx -p xxxx -u user
name -w password -l ship_list_json_path
```

- 将一个Hologres实例中的表备份到OSS存储（bucket为testBucket，存储的根目录为testDump/），并保留所有信息。

```
$ java -jar holo-shipper.jar -s holo -h xx.xx.xx.xx -p xxxx -u username -w password -d os
s -h endpoint -u accessKeyId -w accessKeySecret -b testBucket -p testDump/ -l ship_list_j
son_path
```

- 将一个之前在OSS备份的表恢复到另一个Hologres实例。

```
$ java -jar holo-shipper.jar -s oss -h endpoint -u accessKeyId -w accessKeySecret -b test
Bucket -p testDump/ -d holo -h xx.xx.xx.xx -p xxxx -u username -w password -l ship_list_j
son_path
```

JAR

获取JAR文件，请单击Holo Shipper。

10.安全管理

10.1. 安全白皮书

访问控制

- 网络隔离

  实时数仓Hologres作为阿里巴巴自主研发的一站式实时数仓引擎，在安全性方面需要满足安全隔离规范的要求。目前Hologres支持网络的具体情况如下所示：

  - 每个实例的经典网络、VPC网络、公网网络三网隔离，只能访问各自对应的Endpoint及虚拟内网IP（VIP）。
  - Hologres实例支持配置特定的VPC ID，配置了特定的VPC ID后，实例只能被对应的VPC访问。
  - 配置了IP白名单的实例只能被对应IP网段的服务器访问 。

- IP白名单

  Hologres安全上的访问控制有多个层次，如安全认证机制，只有获取了正确且经过授权的AccessKey ID及AccessKey Secret才能通过鉴权，在已经赋予的权限范围内进行数据访问和计算。下面主要介绍在以上访问认证基础上增强的一种以IP白名单的方式，进行访问控制的配置方法和策略，详细配置方式请参见IP白名单。

  获取需要配置的IP地址的方式如下。

  - 使用PSQL进行项目空间数据访问，您可以直接获取机器的IP地址。
  - 使用了代理服务器或者经过了多跳代理服务器访问Hologres实例，需要添加的IP地址为最后一跳代理服务器的IP地址。
  - ECS机器中访问Hologres服务，需要添加的IP地址为NAT IP。
  - 配置IP地址时，多个IP由逗号（ , ）分隔，支持如下IP格式：
    - 单独IP地址。
    - 带有子网掩码的IP。

    示例如下。

    ```
    --单独IP地址
    10.32.180.8,10.32.180.9,10.32.180.10
    --带有子网掩码的IP地址
    10.32.180.0/23
    ```

鉴权认证

- 身份验证
  - 您可以在阿里云控制台中自行创建AccessKey，AccessKey由AccessKey ID和AccessKey Secret组成，其中AccessKey ID是公开的，用于标识用户身份，AccessKey Secret是秘密的，用于鉴别用户身份。
  - 当您向Hologres发送请求时，首先需要将发送的请求按照Hologres指定的格式生成签名字符串，然后使用AccessKey Secret对签名字符串进行加密以生成请求签名。Hologres收到用户请求后，会根据AccessKey ID使用正确的AccessKey Secret对签名字符串生成签名，如果和请求签名一致即认为该请求是有效的。否则，Hologres将拒绝处理这次请求，并返回HTTP 403错误。

- 权限控制

对Hologres实例访问分为两种，即用阿里云账号访问和RAM用户访问。阿里云账号下可以包含不同的RAM用户以便您可以灵活使用。Hologres支持阿里云账号和RAM用户的权限访问策略：

- 当使用阿里云账号访问时，Hologres会校验该阿里云账号是否为对应实例的所有者，只有对应实例的所有者才具备访问该Hologres实例的权限。
- 当使用RAM用户访问时，此时会触发RAM用户授权策略。Hologres会校验该RAM用户是否被对应阿里云账号授予了访问该实例的权限。

- 权限模型

  Hologres目前主要支持如下三种授权机制来完成对RAM用户的访问权限控制，详细配置方式请参见Hologres权限模型概述。

权限类型	适用场景	说明
专家权限模型（PostgreSQL）	适用于需要非常严格权限管控的场景。例如，精确到某个人用某个表。例如允许用户zinan.tang读取表table1中的数据。	专家权限模型的权限授予粒度小且灵活，可以为用户授予具体某个表的权限，使用 `GRANT/REVOKE` 命令进行授权，通过对应的授权命令来完成对已存在的实例中的数据库、Schema、Table、View进行授权或撤销授权。
简单权限模型（Simple Permission Model，SPM）	数据库级别的权限管控，适用于粗粒度的权限管理场景。	简单权限模型是封装好的权限模型，以数据库为维度，每个用户组都有对应的权限，不可修改，能满足大部分授权场景，且授权操作比较简单。
Schema级别的简单权限模型（Schema-level Permission Model，SLPM）	精确到Schema级别的权限管控，使用于对权限粒度较为细致且又希望简化授权流程的场景。	Schema级别的简单权限模型是已经封装好的权限模型，以Schema为维度，每个用户组都有对应的权限，不可修改，满足对于权限较为细粒度的管控，且授权操作比较简单。

- RAM鉴权
  - Hologres支持RAM鉴权。RAM（Resource Access Management）是阿里云提供的资源访问控制服务。通过RAM阿里云账号可以创建出RAM用户，RAM用户从属于阿里云账号，所有实例都属于阿里云账号，阿里云账号可以将所属Hologres的访问权限授予给RAM用户。
  - 同时您也可以通过角色SSO的方式登录阿里云，并使用Hologres。此时，阿里云访问控制角色（RAM Role）将成为Hologres某个实例的成员，扮演该RAM Role的使用者将拥有和云账号类成员同样的产品使用权限。详细配置方式请参见RAM角色授权模式。

数据安全

- 数据可靠性

  实时数仓Hologres通过了PCI DSS的认证。Hologres使用了分布式文件系统存储数据，数据会以三副本形式存储，并将这些副本按照一定的策略存放在集群中的不同节点上，保证用户数据的可靠。

- 数据脱敏

  Hologres支持数据脱敏功能，可以按照列级别设置脱敏，并且支持对于指定用户设置脱敏策略。启用该功能后，若查询的数据涉及敏感信息，则在展示结果中，该部分数据会被脱敏展示，提高了对敏感及私密数据的保护。现在支持多种脱敏规则，例如IP地址脱敏、邮箱地址脱敏、Hash脱敏等，详细配置方式请参见数据脱敏（Beta）。

- 存储加密

Hologres支持通过密钥管理服务KMS（Key Management Service）对数据进行加密存储，提供数据静态保护能力。现在支持的加密算法有AES256、AESCTR、RC4和SM4，详细配置方式请参见数据存储加密。

日志审计

- Hologres支持通过阿里云操作审计ActionTrail的控制台、OpenAPI、开发者工具等，查询90天内的实例操作事件日志，以实现对事件的监控告警、及时审计、问题回溯分析等需求，详细配置方式请参见查询事件日志。
- Hologres提供Query日志，系统会记录30天内的所有DDL、超过100ms的DML和DQL，详细配置方式请参见慢Query日志查看与分析。

10.2. 数据脱敏（Beta）

Hologres为您提供数据脱敏功能，支持按照列级别设置脱敏。启用该功能后，如果您查询的数据涉及敏感信息，则在展示结果中，该部分数据会被脱敏展示，提高了对敏感及私密数据的保护。本文为您介绍Hologres如何开启、查询及删除数据脱敏功能。

背景信息

随着大数据时代的来临，大数据、云计算和人工智能等新技术应用不断深化，为数据的深度挖掘及分析提供了强有力的支撑，大数据中蕴含的巨大价值被逐步挖掘出来。与此同时也带来了敏感及私密信息保护方面的棘手难题。

Hologres提供的数据脱敏功能，实现了数据在高效共享、挖掘及分析的同时，使得敏感及隐私信息不被泄露，提高了对敏感及私密数据的保护。

使用限制

- 出于安全考虑，Hologres暂不支持从设置了脱敏规则的表和列向未设置脱敏规则的表和列导入数据，相关报错信息如下。

  错误原因：`ERROR: The insert table has not set SECURITY LABEL`

- 数据脱敏会一定程度的影响查询性能，影响程度和脱敏方式、数据量都有关，性能有10%~20%的下降，极端场景可能存在数倍性能下降。

使用数据脱敏

Hologres支持对目标列或目标用户设置数据脱敏。使用数据脱敏前，您需登录目标数据库开启脱敏功能。完成后可配置对应的脱敏规则，针对某一列或某个用户设置脱敏。具体操作步骤如下。

1. 为目标数据库开启数据脱敏。

   数据脱敏功能默认不开启，需要Superuser登录至目标数据库开启该功能，开启命令及相关参数说明如下。

   - 命令：

     ```
     CREATE EXTENSION IF NOT EXISTS hg_anon; --创建hg_anon扩展函数。
     ALTER DATABASE <db_name> SET hg_anon_enable = on;  --为指定数据库开启数据脱敏功能，默认为off。
     ```

   - 示例：

     ```
     CREATE EXTENSION IF NOT EXISTS hg_anon;--创建hg_anon扩展函数。
     ALTER DATABASE test SET hg_anon_enable = on;--为数据库test开启数据脱敏功能。
     ```

- 参数说明：

参数	参数的相关描述
hg_anon	hg_anon是Hologres内部封装的扩展函数，您需要调用该扩展函数才能开启数据脱敏功能。
<db_name>	需要开启数据脱敏功能的数据库名称。使用时需要替换<db_name>为实际数据库名称。
hg_anon_enable	选择开启或关闭数据脱敏功能，取值如下： ■ *on*，开启数据脱敏。 ■ *off*，关闭数据脱敏。 默认该参数设置为*off*。

2. 设置脱敏列

   可由Superuser或表Owner对目标列进行脱敏，具体命令、示例、参数说明如下。

   - 命令：

     > **说明** 对多列数据进行脱敏时，需要多次执行该语句。

     ```
     SECURITY LABEL FOR hg_anon ON COLUMN <tablename>.<col_name> IS <label_name>;
     ```

   - 示例：对*holotest*表的ID列按照名称脱敏。

     ```
     SECURITY LABEL FOR hg_anon ON COLUMN holotest.id IS 'name';
     ```

   - 参数说明：

参数	描述
hg_anon	hg_anon是Hologres内部封装的扩展函数，您需要调用该扩展函数才能开启数据脱敏功能。
<tablename>	需要脱敏的列所在的表名称。使用时需要替换<tablename>为实际表名称。
<col_name>	需要脱敏的列名称。使用时需要替换<col_name>为实际列名称。
label_name	系统预设的脱敏函数，您可以使用 `show hg_anon_labels;` SQL查看当前数据库设置的label_name。

   从HologresV1.1版本开始，默认的label_name如下。

lable_name	说明	示例
name	姓名脱敏。	■ 脱敏前：李华；脱敏后：*华。 ■ 脱敏前：王小强；脱敏后：**强。
email	邮箱地址脱敏。	脱敏前：lihuang@alibaba.com；脱敏后：lih***@alibaba.com

lable_name	说明	示例
ip	IP地址脱敏。	脱敏前：1.2.3.4；脱敏后：1.*.*.*。
id	身份证号码脱敏。	脱敏前：110345188812011234；脱敏后：1****************4
phone	电话号码脱敏。	脱敏前：13900001234；脱敏后：*********34。
bank_id	银行卡、信用卡账号脱敏。	脱敏前：2349867902834701928；脱敏后：***************1928。
hash	使用MD5算法进行脱敏。	脱敏前：浙江省杭州市文一西路；脱敏后：dbf894b409d4a2ef17dfd9c7fdcafcd8。
first_mask	定义了一个first_mask的规则，只显示第一个字符。	脱敏前：123456789；脱敏后：1********。

- 脱敏结果示例：

```
db_1625191993_1346616=# select * from test_tb;
  name  |          email           |    ip     |         id         |       detail
--------+--------------------------+-----------+--------------------+--------------------
 李华   | lihua@alibaba.com        | 125.9.8,3 | 110123199011211234 | 北京市长安街200号
 张三丰 | zhangsanfeng@alibaba.com | 125.9.8,3 | 510123199811201234 | 重庆市红岩路100号
 王刚   | wanggang@alibaba.com     | 128.9.9,3 | 310512199912201234 | 上海市浦东大道700号
(3 rows)

db_1625191993_1346616=# SECURITY LABEL FOR hg_anon ON COLUMN test_tb.name IS 'name';
SECURITY LABEL
db_1625191993_1346616=# SECURITY LABEL FOR hg_anon ON COLUMN test_tb.email IS 'email';
SECURITY LABEL
db_1625191993_1346616=# SECURITY LABEL FOR hg_anon ON COLUMN test_tb.ip IS 'ip';
SECURITY LABEL
db_1625191993_1346616=# SECURITY LABEL FOR hg_anon ON COLUMN test_tb.id IS 'first_mask';
SECURITY LABEL
db_1625191993_1346616=# SECURITY LABEL FOR hg_anon ON COLUMN test_tb.detail IS 'hash';
SECURITY LABEL
db_1625191993_1346616=# select * from test_tb;
 name |       email        |    ip     |         id         |              detail
------+--------------------+-----------+--------------------+----------------------------------
 **丰 | zha***@alibaba.com | 125.*.*** | 5***************** | efe068bdec29c96bff4ba541ec1a301e
 *华  | lih***@alibaba.com | 125.*.*** | 1***************** | 620fffefd75522818a8553e1333ac910
 *刚  | wan***@alibaba.com | 128.*.*** | 3***************** | ad6f465b0cac046ac837291c87379f0c
(3 rows)
```

- 其他相关命令
  - 删除已经设置的脱敏

```
SECURITY LABEL FOR hg_anon ON COLUMN test_hg_anon_demo.name is NULL;
```

  - 查看设置的脱敏列

```
SELECT c.relname, a.attname, provider, label FROM pg_seclabel s INNER JOIN pg_catal
og.pg_class c on s.objoid = c.relfilenode INNER JOIN pg_catalog.pg_attribute a on s
.objoid = a.attrelid where a.attnum = objsubid;
```

高级技巧

- 对不同用户设置脱敏

  默认情况下所有用户都会脱敏，可以单独设置某些用户不脱敏。

  - 命令格式如下

    ```
    SECURITY LABEL FOR hg_anon ON ROLE user_name IS 'all:unmasked';
    ```

参数	说明
user_name	账号ID。请通过用户信息页面获取。

  - 使用示例

    对账号ID为1365xxxxxxxxxxx的用户不脱敏的语句如下。

    ```
    SECURITY LABEL FOR hg_anon ON ROLE "1365xxxxxxxxxxx" IS 'all:unmasked';
    ```

  - 相关命令

    - 查看对用户设置的脱敏情况

      ```
      select usename, label from pg_shseclabel s inner join pg_catalog.pg_user u on s.objoi
      d = u.usesysid;
      ```

    - 删除对用户设置的脱敏

      ```
      SECURITY LABEL FOR hg_anon ON ROLE "1365xxxxxxxxxxx" IS NULL;
      ```

- 自定义脱敏规则

  如果默认的label_name无法满足需求，您可以通过修改GUC的hg_anon_lables参数自定义脱敏规则。

  - 命令

    ```
    ALTER DATABASE <db_name> SET hg_anon_labels = '[
      {"label": <label_name1>, "method", <method1>},
      {"label": <label_name2>, "method", <method2>},
      ...
    ]'; --label_name是您自定义的名称，method是Hologres内置的一些方法。
    ```

    > **说明** 执行ALTER DATABASE命令后，当前链接会不生效，需要重开一个连接。您可使用如下命令查看设置是否生效。
    >
    > ```
    > show hg_anon_enable;
    > ```

  - 示例

    ```
    ALTER DATABASE test_db SET hg_anon_labels = '[
    {"label":"ip", "method":{"desensType":"mask", "type":"ip"}},
    {"label":"email", "method":{"desensType":"mask", "type":"email"}},
    {"label":"name", "method":{"desensType":"mask", "type":"name"}},
    {"label":"first_mask", "method":{"desensType":"mask", "type":"user_define", "before":1,
    "after":0}},
    {"label":"hash", "method":{"desensType":"hash", "type":"md5", "salt":""}}]';
    ```

- 参数说明：

脱敏项	脱敏内容描述	脱敏结果示例
{"desensType":"mask", "type":"ip"}	IP地址脱敏。	*192.*.*.**
{"desensType":"mask", "type":"email"}	邮箱地址脱敏。	*abc***@example.net*
{"desensType":"mask", "type":"name"}	名称脱敏。	**五*
{"desensType":"hash", "type":"md5", "salt":""}	Hash脱敏。	*e086aa137fa19f67d27b39d0eca186103228f322c9c98a125554a24f875f0f7e*
{"label":"first_mask", "method":{"desensType":"mask", "type":"user_define", "before":1, "after":0}}{"label":"last_mask", "method":{"desensType":"mask", "type":"user_define", "before":0, "after":1}}	自定义内容脱敏。	*无*

使用示例

使用Hologres数据脱敏功能完整的使用示例如下。

1. 创建数据库

   使用如下命令语句创建数据库。

   ```
   CREATE DATABASE hg_anon_demo;
   ```

2. 创建样例数据表

   切换到创建的数据库，使用如下命令语句创建样例数据表。

   ```
   DROP TABLE IF EXISTS personal_basic_information;
   CREATE TABLE personal_basic_information
   (
       name TEXT
       ,email TEXT
       ,ip TEXT
       ,id TEXT
       ,phone TEXT
       ,bank_id TEXT
   )
   ;
   ```

3. 插入样例数据

   使用如下命令语句，在personal_basic_information表中插入样例数据。

```
INSERT INTO personal_basic_information(name,email,ip,id,phone,bank_id) VALUES
('张三','zhangsan@alibaba.com','127.0.0.1','142732199102290022','13900001234','451461080
3067088'),
('李四','lisi@alibaba.com','127.0.0.1','510622198402308000','13900001111','6252470010027
800'),
('李逍遥','lixiaoyao@alibaba.com','172.21.4.234','511025188812271696','13900002222','625
2470010027800');
```

4. 查看数据

   查看数据脱敏前的数据。

```
SELECT * FROM personal_basic_information;
```

5. 设置脱敏规则

   使用如下命令语句为数据设置脱敏规则。

```
--创建hg_anon扩展函数。
CREATE EXTENSION IF NOT EXISTS hg_anon;
--为数据库hg_anon_demo开启数据脱敏功能。
ALTER DATABASE hg_anon_demo SET hg_anon_enable = on;
--设置每一列的脱敏规则
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.name IS 'name';
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.id IS 'id';
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.phone IS 'phone';
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.email IS 'email';
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.bank_id IS 'bank_id';
SECURITY LABEL FOR hg_anon ON COLUMN personal_basic_information.ip IS 'ip';
```

6. 查看数据

   请断开数据库，重新登录数据库使用如下命令查看数据脱敏后的数据。

```
SELECT * FROM personal_basic_information;
```

   脱敏后的数据如下所示。

name	email	ip	id	phone	bank_id
*四	lis***@alibaba.com	127.*.*.*	5**************0	*********11	***********7800
**遥	lix***@alibaba.com	172.**.*.***	5**************6	*********22	***********7800
*三	zha***@alibaba.com	127.*.*.*	1**************2	*********34	***********7088

使用数据保护伞进行数据脱敏

您不仅可以手工设置脱敏规则，也可以使用数据保护伞进行数据脱敏。

- 使用限制。
  - 仅Hologres V1.1及以上版本支持使用数据保护伞进行数据脱敏，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
  - 为了能够检测出敏感数据，保护伞对于主账号设置不脱敏的规则。
  - 目前每天9:00:00，保护伞会抽样数据，识别敏感数据，并对设置的敏感数据列设置脱敏规则。
  - 目前Hologres支持使用数据保护伞进行数据脱敏的Region有：中国（香港）。
- 使用方法。

- 开启脱敏功能。

  数据脱敏功能默认不开启，需要拥有Superuser权限的用户在对应的数据库中执行如下命令开启该功能。

  ```
  --安装数据脱敏EXTENSION
  CREATE EXTENSION IF NOT EXISTS hg_anon;
  --对指定的数据库开启数据脱敏功能，默认为关闭状态
  ALTER DATABASE <db_name> SET hg_anon_enable = on;
  ```

  db_name为需要开启数据脱敏功能的数据库。

  > **注意**
  > - `hg_anon_enable` 是一个GUC，运行ALTER DATABASE命令后当前连接不生效。
  > - 您可以使用如下SQL查看设置是否生效。
  >
  > ```
  > show hg_anon_enable;
  > ```

- 设置脱敏数据库。

  a. 登录数据保护伞控制台，详情请参见进入数据保护伞。

  b. 在左侧导航栏，单击**规则配置 > 数据识别规则**，进入**数据识别规则**页面。

  c. 在**敏感数据识别**页面创建一条数据识别规则，详情请参见敏感数据识别。

  d. 在左侧导航栏，单击**数据脱敏管理**，进入**数据脱敏管理**页面。

  e. 在**脱敏场景**下拉列表中，选择Hologres展示脱敏（hologres_display_desense_code），并单击右侧**选择脱敏database**。

  f. 在**授权账号脱敏**对话框，从**未脱敏database**列表选择需要脱敏的数据库显示在**脱敏database**列表中，单击**我同意授权数据保护伞对该database脱敏**，单击**确定**。

  g. 在**数据脱敏管理**页面，单击右上方的**新建脱敏规则**，详情请参见数据脱敏管理，之后系统会对您设置的数据库进行脱敏。

常见问题

按照使用示例发现数据未脱敏。

- 问题现象：按照使用示例执行后发现查询出的数据未进行脱敏。
- 可能原因：
  - 对部分用户设置了不脱敏的规则。

- 未设置脱敏标签。

- 解决方法：
  - 执行如下SQL命令检查是否对于部分用户设置了不脱敏的规则。

    ```
    select
      usename
      , label
    from pg_shseclabel s
    inner join pg_catalog.pg_user u on s.objoid = u.usesysid;
    ```

    默认情况下该SQL查询结果为空，表示对所有用户均需要进行数据脱敏；若不为空，请对用户设置为脱敏，详情请参见高级技巧。
  - 使用如下SQL，检查设置的脱敏标签。

    ```
    show hg_anon_labels;
    ```

    如果结果中并未包含ip等标签，请执行如下SQL命令设置脱敏标签。

    ```
    ALTER DATABASE compress_test SET hg_anon_labels = '[
    {"label":"ip", "method":{"desensType":"mask", "type":"ip"}},
    {"label":"email", "method":{"desensType":"mask", "type":"email"}},
    {"label":"name", "method":{"desensType":"mask", "type":"name"}},
    {"label":"id", "method":{"desensType":"mask", "type":"id"}},
    {"label":"phone", "method":{"desensType":"mask", "type":"phone"}},
    {"label":"bank_id", "method":{"desensType":"mask", "type":"bank_id"}},
    {"label":"hash", "method":{"desensType":"hash", "type":"md5", "salt":""}},
    {"label":"first_mask", "method":{"desensType":"mask", "type":"user_define", "before":1,
    "after":0}}
    ]';
    ```

    以上标签对应的规则请参见默认的label_name。

10.3. IP白名单

为保障Hologres的安全稳定，Hologres支持在HoloWeb中设置IP白名单来进行访问管理，本文将指导您在Hologres中设置IP白名单。

注意事项

在HoloWeb中设置IP白名单之前，您需要注意如下事项：

- 仅Hologres V0.10.14及以上版本支持设置IP白名单，请在Hologres管理控制台的实例详情页或者执行命令 `select hg_version()` 查看当前实例版本。如果您的实例是V0.10.14以下版本，可以提交工单或加入在线支持钉钉群申请升级实例。
- 购买Hologres实例成功后，若是没有设置IP白名单， 则默认对所有网络开放。
- 仅支持实例管理员（Superuser）设置IP白名单。
- 在HoloWeb配置数据连接时，需要将连接的登录方式设置为当前用户免密登录，才可以为当前连接配置IP白名单。连接Hologres实例配置操作指导，请参见连接Hologres实例。

- 设置白名单之后，HoloStudio将不能访问。因此，您需要按照新增IP白名单的操作指导，将其对应的分组加入到IP白名单，保证正常访问。

分组	说明
HoloStudioGroup	加入该分组后可以访问HoloStduio。

- 如果DataWorks数据集成资源组和Hologres实例网络已连通，但资源组仍然无法访问您的Hologres，则需要获取资源组的IP地址与网段，添加至数据库的白名单中，获取资源组的IP地址与网段，请参见添加白名单。

新增IP白名单

1. 登录Hologres管理控制台，在顶部菜单栏左侧，选择相应的地域。
2. 单击左侧导航栏的**前往HoloWeb**，进入**HoloWeb**开发界面。
3. 在**HoloWeb**页面，在**安全中心**页面的左侧导航栏，选择**IP白名单**。
4. 在页面右上角单击**新增IP白名单**，配置如下参数信息。

参数	说明
分组	自定义的分组名称。 当您将连接的登录方式设置为当前用户免密登录后，DataWorks数据集成资源组和HoloStudio也必须加入到IP白名单内，否则其功能将不可用。在分组下拉框中选择各自对应的分组名称即可。
数据库限制	从下拉框中选择需要设置白名单的数据库。如果需要设置当前实例所有的DB，可以选择ALL。
用户限制	从下拉框中选择需要设置白名单的用户。如果需要设置当前实例所有的用户，可以选择ALL。

参数	说明
IP地址	设置白名单的IP地址。配置信息如下： ○ 所有IP地址：填写为ALL。 ○ 指定IP地址：如192.168.0.1，允许192.168.0.1的IP地址访问。 ○ 指定IP段：如192.168.0.0/24，允许从192.168.0.1到192.168.0.255的IP地址访问。 ○ 多个IP设置：换行展示。

5. 单击**确认**，完成配置。白名单设置成功后，可以允许您在IP白名单的范围内进行操作。

编辑IP白名单

如果您需要修改IP白名单地址信息，可以对IP白名单进行修改操作。当前仅允许更改IP地址，若是要更改数据库、用户限制等信息，请您新建IP白名单。

> **说明** 仅实例管理员（Superuser）可以编辑IP白名单。

1. 在**HoloWeb**页面，在**安全中心**页面的左侧导航栏，选择**IP白名单**。
2. 在**IP白名单**管理页面，单击目标IP白名单右侧的**编辑**。
3. 在**编辑IP白名单**页面，修改IP地址信息。IP信息的配置内容，请参见新增IP白名单。
4. 单击**确认**，完成配置。

删除IP白名单

如果您不再需要设置的IP白名单信息，可以对IP白名单执行删除操作。如果您删除所有的IP白名单，则默认不设置白名单。

> **说明** 仅实例管理员（Superuser）可以编辑IP白名单。

1. 在**HoloWeb**页面，在**安全中心**页面的左侧导航栏，选择**IP白名单**。
2. 在**IP白名单**管理页面，单击目标IP白名单右侧的**删除**。
3. 单击**确认**，删除完毕。

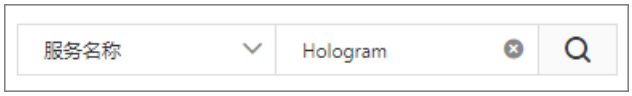
10.4. 查询事件日志

Hologres支持通过阿里云操作审计ActionTrail的控制台、OpenAPI、开发者工具等，查询90天内的实例操作事件日志。您可将查询结果投递至日志服务LogStore或OSS Bucket中，以实现对事件的监控告警、及时审计、问题回溯分析等需求。本文将为您介绍在操作审计ActionTrail控制台中查询事件日志的方法。

注意事项

- 对Hologres实例进行的相关操作，需要5~10分钟左右的时间生成行为事件日志。
- Hologres通过操作审计ActionTrail仅能查询90天以内的事件日志。
- 如您需要实时处理异常行为，可为重要事件配置跟踪警告，配置方法请参见使用操作审计监控阿里云账号的使用。
- 如您无法通过操作审计ActionTrail的控制台查询事件日志，也可使用OpenAPI、开发者工具等进行查询，详情请参见API概览。

1. 登录操作审计控制台。
2. 在左侧导航栏，单击**事件查询**。
3. 在顶部导航栏选择您想查询事件的地域。
4. 在**事件查询**页的下拉列表中选择**服务名称**，在右侧文本框中输入产品名称**Hologram**。



5. 单击图标，在下方查看具体的事件内容。其中**事件名称**的具体含义如下表所示。

事件名称	事件名称对应的实例操作行为
create	新建实例。
Modify	实例配置变更（升配或降配）。
StopInstance	暂停实例。
ResumeInstance	恢复实例。
DeleteInstance	删除实例。
ModifyInstance	修改实例名称。
modifyInstanceNetworkType	网络配置变更（公网、VPC网络域名开关操作）。
ListInstances	查看实例列表。
DescribeInstance	查看实例详情。
GetInstanceMetrics	查看监控指标。

6. （可选）如果需要查询事件代码记录，您可以单击事件前面的加号，然后单击**事件详情**。

> **说明** 关于事件中字段的更多信息，请参见管控事件结构定义。

10.5. 数据加密

10.5.1. 数据存储加密

本文为您介绍如何在Hologres中对数据进行加密处理，本文包括数据加密机制、使用限制及操作步骤。

背景信息

Hologres支持通过密钥管理服务KMS（Key Management Service）对数据进行加密存储，提供数据静态保护能力，满足企业监管和安全合规需求。开启加密存储后，由于涉及加密和解密操作，所以会影响查询和写入性能，大约有20%-40%的性能损耗，具体情况根据查询特征有不同程度的影响。

使用限制

- 仅Hologres V1.1及以上版本支持数据加密存储，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 您在KMS上对自带密钥（BYOK）的操作（例如禁用或删除），会影响Hologres对数据的加密或解密操作。由于Hologres服务涉及缓存，您在KMS的相关操作会在24小时内生效。
- 存储加密仅会对开启存储加密后的数据表生效，对于开启存储加密前创建的数据表，系统不会对其进行存储加密。

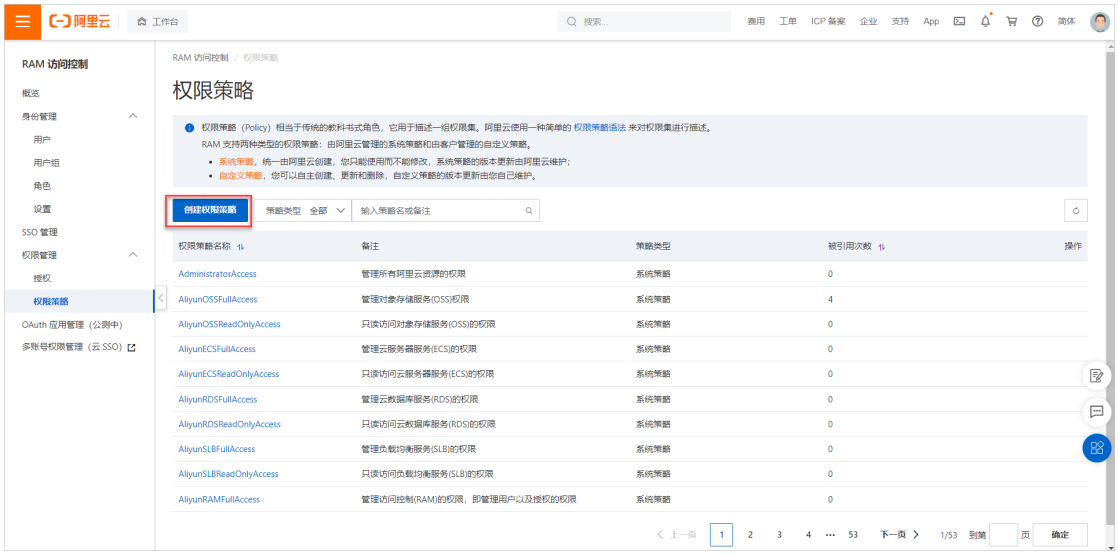
数据加密机制

Hologres通过KMS托管密钥，实现数据加密或解密功能，数据加密机制如下。

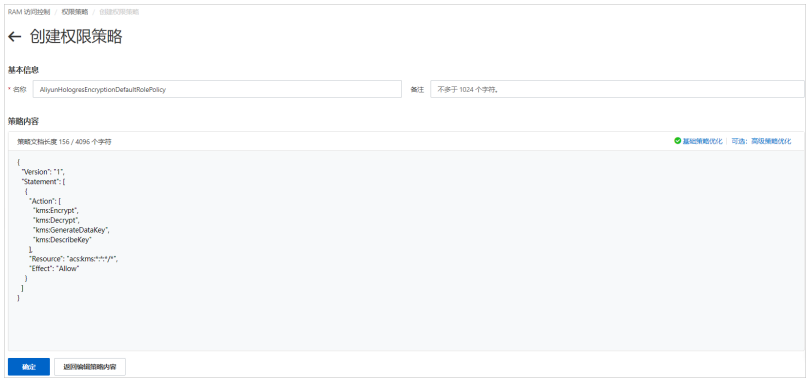
- Hologres以数据库为单位，通过KMS加密或解密存储在Hologres的数据。在使用数据加密功能前，请确保您所在区域已开通KMS服务。
- KMS生成和管理您的密钥并保障密钥的安全性。
- Hologres支持的加密算法有AES256、AESCTR、RC4和SM4。
- Hologres仅支持自带密钥（BYOK）加密或解密数据。
  - 您可以通过KMS创建自带密钥（BYOK），并在Hologres中针对数据库，选择该密钥作为密钥进行数据加密。在KMS上创建BYOK的详情请参见创建密钥。
  - 如果项目使用自带密钥（BYOK），需完成RAM授权，以便Hologres可以正常创建使用了自带密钥（BYOK）的项目空间。
- 在数据读写时，Hologres会调用KMS的API获取相关的密钥信息，获取相关的密钥信息系统默认会缓存24小时。因此在使用数据加密功能时会产生相关的KMS费用，KMS相关计费说明请参见KMS计费说明。

操作步骤

1. 创建自定义权限策略
   i. 登录RAM控制台，单击左侧导航栏的**权限策略**，进入**权限策略**页面，单击**创建权限策略**。



ii. 进入**新建自定义权限策略**页面，策略名称命名为AliyunHologresEncryptionDefaultRolePolicy，配置模式选择*脚本配置*，配置脚本如下所示。




```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "acs:kms:*:*:*/*",
            "Effect": "Allow"
        }
    ]
}
```

iii. 单击**确定**，完成自定义权限策略的创建。

2. 创建Hologres代理角色并授权

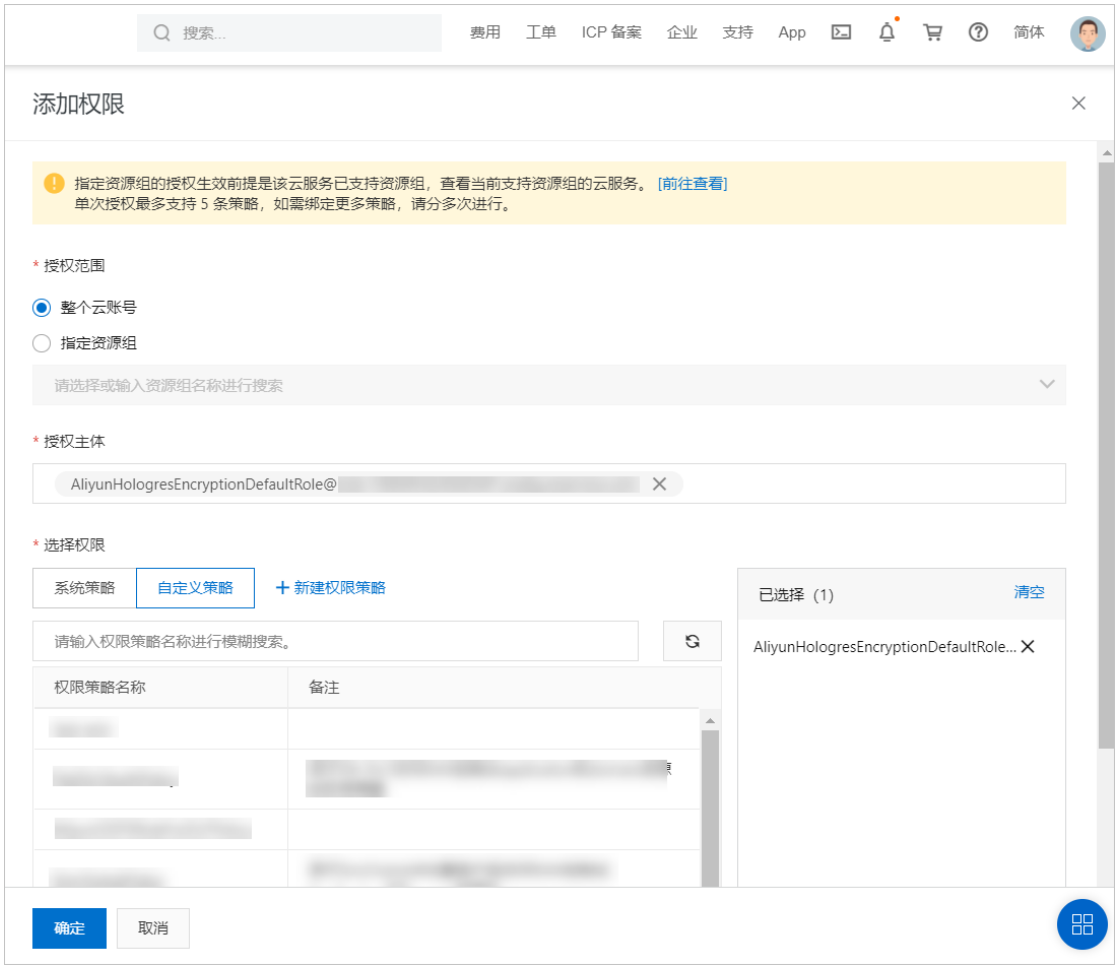
i. 登录RAM控制台，单击左侧导航栏的**角色**。

ii. 进入**角色**页面，单击**创建角色**，进入**创建角色**对话框，选择可信实体类型为*阿里云服务*。

iii. 单击**下一步**，角色类型选择*普通服务角色*，角色名称命名为AliyunHologresEncryptionDefaultRole，设置选择受信服务为*交互式分析*，单击**完成**。

iv. 进入**创建完成**页面，单击**为角色授权**。

v. 进入**添加权限**页面，授权范围选择*整个云账号*，选择权限为*自定义策略*中第一步创建的自定义角色策略（AliyunHologresEncryptionDefaultRolePolicy）。



vi. 单击**确定**，完成角色创建和授权操作。

在创建完角色后，单击创建的角色，在**信任策略管理**页签，查看信任策略设置是否和下图一致，若不一致，请确认创建角色的步骤是否正确。

3. 创建密钥

i. 登录密钥管理服务控制台。

ii. 在页面左上角的地域下拉列表，选择密钥所在的地域。

iii. 在左侧导航栏，单击**用户主密钥**。

iv. 单击**创建密钥**，在弹出的**创建密钥**对话框，根据控制台提示进行配置。

配置项	说明
KMS实例	选择要创建密钥的KMS实例。
密钥类型	取值： ■ 对称密钥的类型： ■ Aliyun_AES_256 ■ Aliyun_SM4 ■ 非对称密钥的类型： ■ RSA_2048 ■ RSA_3072 ■ EC_P256 ■ EC_P256K ■ EC_SM2 说明 ■ Aliyun_SM4或EC_SM2的密钥类型，仅在中国内地使用托管密码机的地域支持。 ■ RSA_3072的密钥类型，仅支持在专属KMS实例中创建。
密钥用途	取值： ■ Encrypt/Decrypt：数据加密和解密。 ■ Sign/Verify：产生和验证数字签名。
别名	用户主密钥的可选标识。 更多信息，请参见别名概述。
保护级别	取值： ■ Software：通过软件模块对密钥进行保护。 ■ Hsm：将密钥托管在密码机中，使密钥获得高安全等级的专用硬件的保护。
描述	密钥的说明信息。

配置项	说明
轮转周期	自动轮转的时间周期。取值： ■ 30天。 ■ 90天。 ■ 180天。 ■ 365天。 ■ 不开启：不开启轮转。 ■ 自定义：7~730天。 说明 仅密钥类型为Aliyun_AES_256和Aliyun_SM4的对称密钥支持设置轮转周期。

4. 设置加密规则

   i. 登录需要进行存储数据加密的数据库实例，登录数据库请参见登录数据库。

   ii. 在**临时Query查询**页面，选择已创建的**实例名**和**数据库**后，请您在SQL查询的编辑框输入如下语句，单击**运行**。

   以下SQL语句用于设置数据存储加密规则。

```
ALTER DATABASE <db_name> SET hg_experimental_encryption_options='<encryption_type>,
<cmk_id>,<ram_role>,<uid>';
```

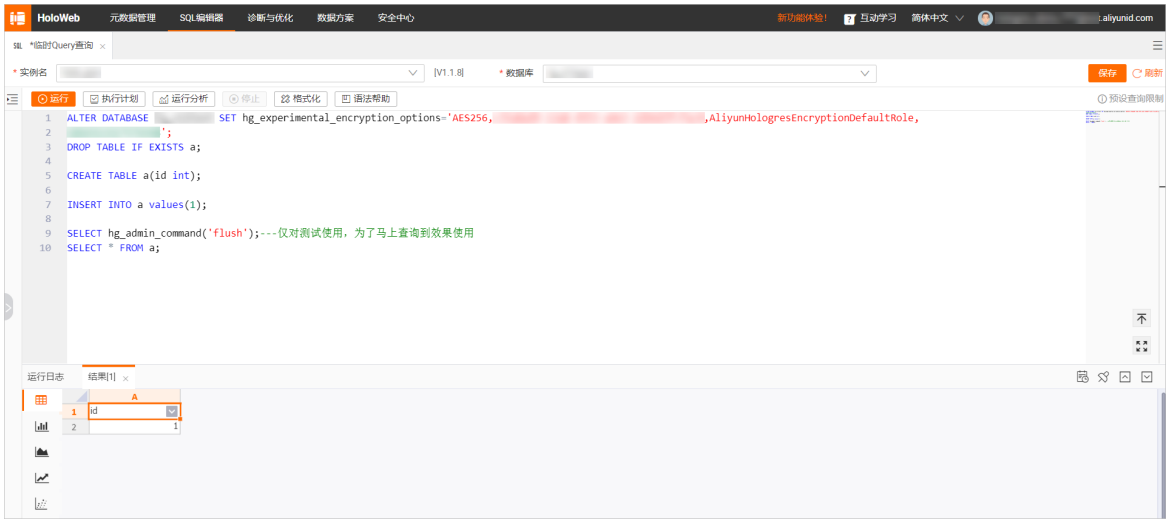
   参数说明

参数	说明
db_name	需要进行加密存储的数据库名称。
encryption_type	加密类型。仅支持AES256、AESCTR、RC4、SM4四种类型。
cmk_id	密钥ID。登录密钥管理服务控制台，密钥详情页获取。
ram_role	创建的Hologres代理角色名称。
uid	阿里云账号ID。获取详情请参见账号ID。

一个设置加密规则并查询数据的简单使用示例如下。

```
ALTER DATABASE hoxxxx set hg_experimental_encryption_options='AES256,623c26ee-xxxx-xxxx
-xxxx-91d323cc4855,AliyunHologresEncryptionDefaultRole,187xxxxxxxxxxxx';
DROP TABLE IF EXISTS a;
CREATE TABLE a(id int);
INSERT INTO a values(1);
SELECT hg_admin_command('flush');---仅对测试使用，为了马上查询到效果使用
SELECT * FROM a;
```

运行结果如下。



若此时禁用了KMS，在实例重启或者24小时后，查询a表数据会报错。

10.5.2. 查询MaxCompute加密数据（BYOK模式）

本文为您介绍Hologres如何使用外部表查询MaxCompute BYOK加密数据。

背景信息

MaxCompute支持通过密钥管理服务KMS（Key Management Service）对数据进行加密存储，提供数据静态保护能力，满足企业监管和安全合规需求。本文为您介绍在Hologres中使用外部表查询MaxCompute的加密数据的限制条件和完整步骤。

使用限制

- 仅Hologres V1.1及以上版本支持使用外部表查询MaxCompute的加密数据，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 仅外部表的执行引擎为SQE时，支持访问MaxCompute存储的加密数据，如果您的执行引擎是HQE，请您通过以下两种SQL语句将执行引擎更改为SQE。

```
-- session级别修改
set hg_experimental_enable_access_odps_orc_via_holo = false;
```

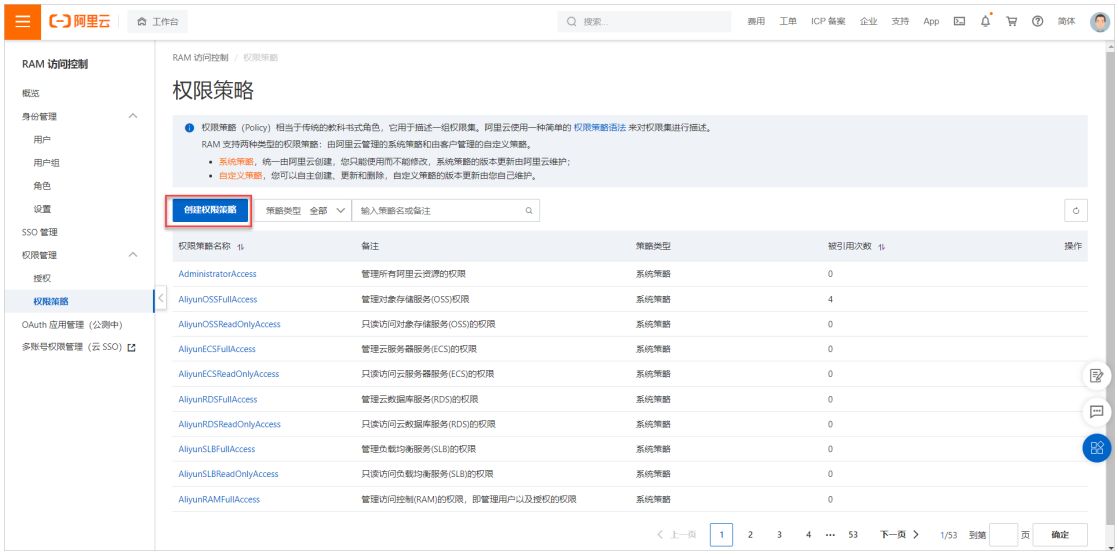
```
-- DB级别修改
ALTER DATABASE <DB_Name> SET hg_experimental_enable_access_odps_orc_via_holo = false;
```

- 仅支持查询BYOK方式加密的MaxCompute数据，对于使用DataWorks Default Key方式加密的MaxCompute的加密数据无法查询。
- 查询数据时，Hologres会调用KMS的API获取相关的密钥信息，获取相关的密钥信息系统默认会缓存24小时。

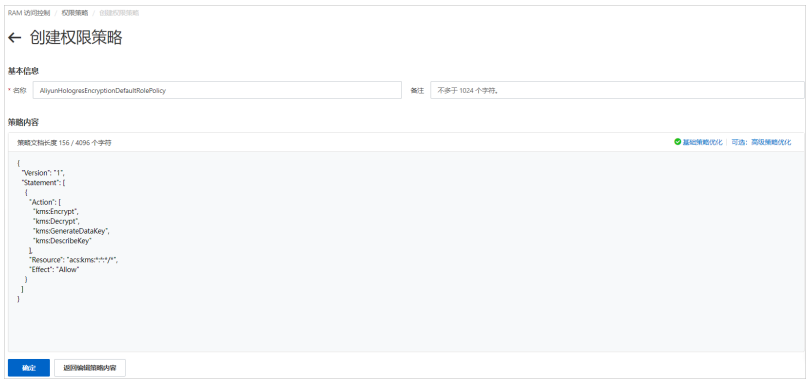
操作步骤

1. 创建自定义权限策略

i. 登录RAM控制台，单击左侧导航栏的**权限策略**，进入**权限策略**页面，单击**创建权限策略**。



ii. 进入**新建自定义权限策略**页面，策略名称命名为AliyunHologresEncryptionDefaultRolePolicy，配置模式选择*脚本配置*，配置脚本如下所示。




```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "acs:kms:*:*:*/*",
            "Effect": "Allow"
        }
    ]
}
```

iii. 单击**确定**，完成自定义权限策略的创建。

2. 创建Hologres代理角色并授权

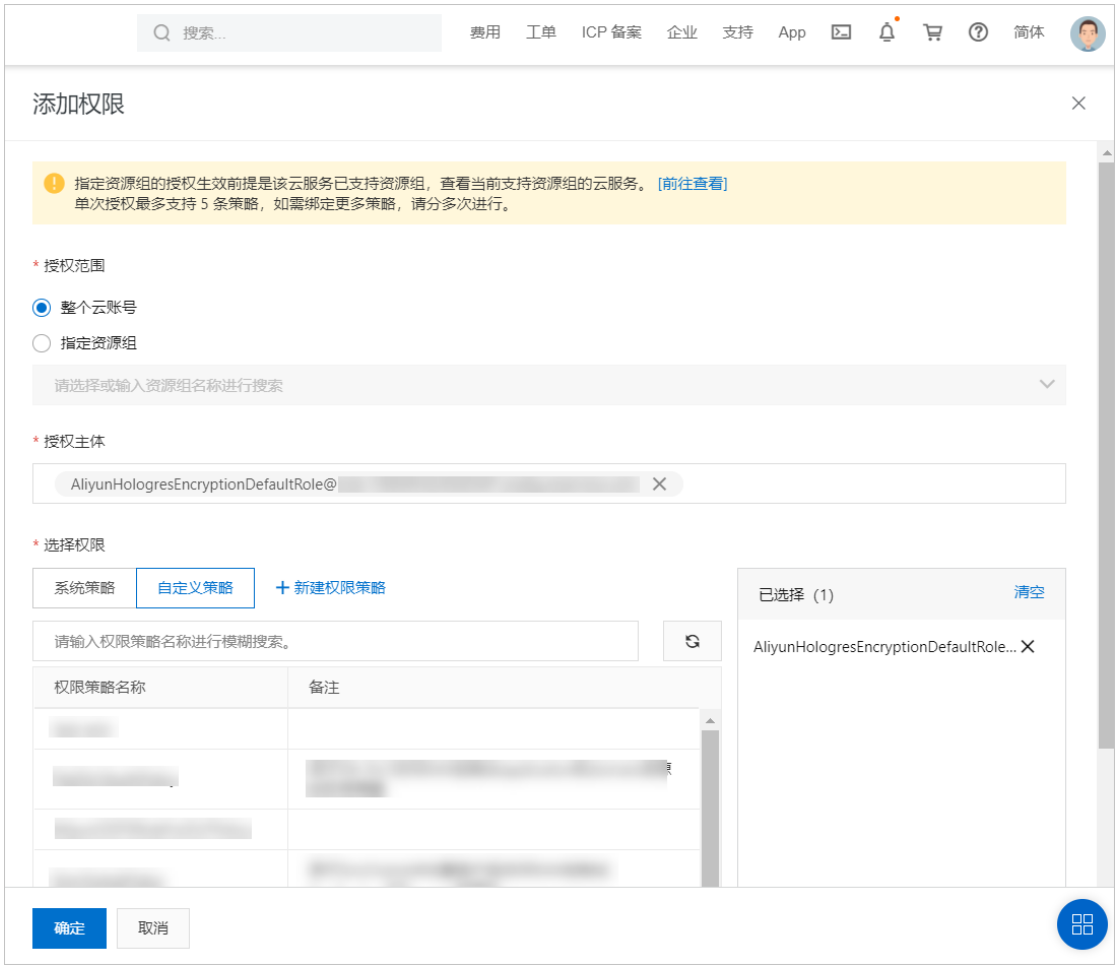
i. 登录RAM控制台，单击左侧导航栏的**角色**。

ii. 进入**角色**页面，单击**创建角色**，进入**创建角色**对话框，选择可信实体类型为*阿里云服务*。

iii. 单击**下一步**，角色类型选择*普通服务角色*，角色名称命名为AliyunHologresEncryptionDefaultRole，设置选择受信服务为*交互式分析*，单击**完成**。

iv. 进入**创建完成**页面，单击**为角色授权**。

v. 进入**添加权限**页面，授权范围选择*整个云账号*，选择权限为*自定义策略*中第一步创建的自定义角色策略（AliyunHologresEncryptionDefaultRolePolicy）。



vi. 单击**确定**，完成角色创建和授权操作。

在创建完角色后，单击创建的角色，在**信任策略管理**页签，查看信任策略设置是否和下图一致，若不一致，请确认创建角色的步骤是否正确。

3. 查询数据

角色创建和授权完成以后，您便可以同使用Hologres的外部表查询MaxCompute的普通数据一样查询加密数据，详情请参见通过创建外部表加速查询MaxCompute数据。

10.6. 数据地图（Beta）

DataWorks数据地图是在元数据基础上提供的企业数据目录管理模块，元数据详情查看、数据血缘和数据类目管理等功能。数据地图可以帮助您更好地查找、理解和使用Hologres数据。本文为您介绍如何在数据地图中配置Hologres元数据采集器及相关操作。

使用限制

- 仅Hologres V1.1及以上版本支持使用数据地图功能，如果您的实例是V1.1以下版本，请您提交工单或加入在线支持钉钉群申请升级实例。
- 在数据地图中配置Hologres元数据采集器一个小时后才能看到血缘信息。
- 仅在如下地域提供数据血缘服务。

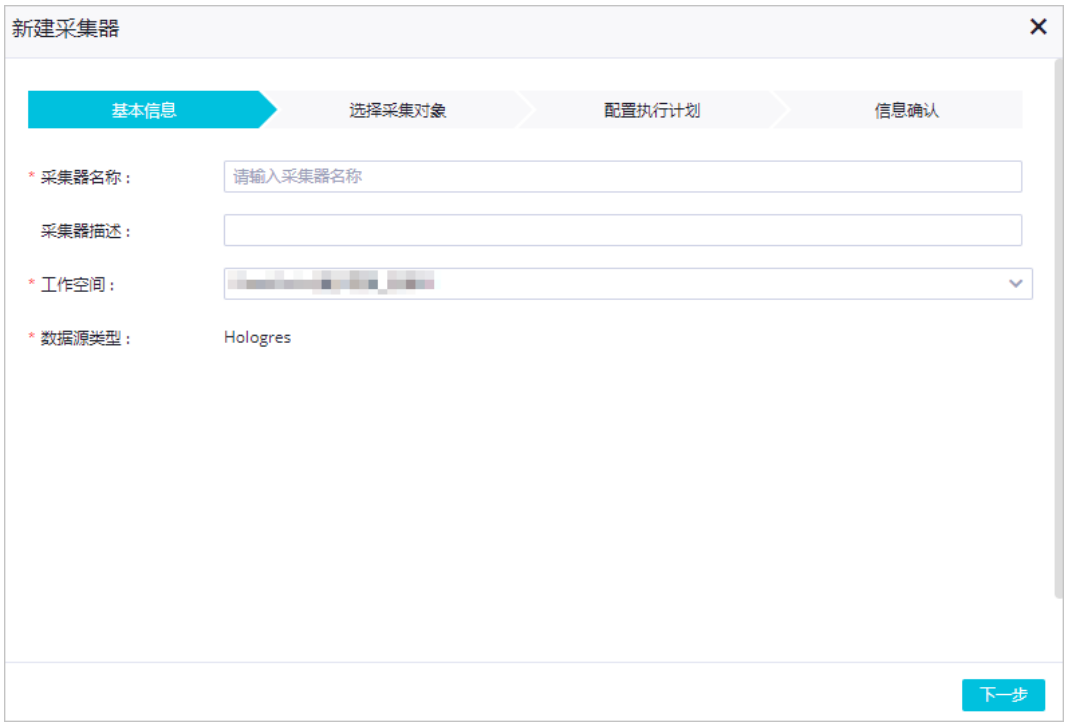
  华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国（香港）、新加坡。

元数据采集与接入

您需要通过元数据采集功能将Hologres数据源中的元数据导入数据地图进行统一管理，步骤如下。采集完成后，您可以在数据地图搜索并查看Hologres数据源的元数据信息。

1. 登录**DataWorks控制台**后，进入**数据地图**页面，操作详情请参见进入首页。
2. 在顶部菜单栏，单击**数据发现**。
3. 在左侧导航栏，单击**元数据采集 > Hologres**。
4. 在**Hologres元数据采集**页面，单击**新建采集器**。
5. 在**新建采集器**配置向导页面，完成以下操作。
   i. 配置基本信息。

a. 在**基本信息**页签下，配置各项参数。



参数	描述
采集器名称	采集器的名称，必填且唯一。
采集器描述	对采集器进行简单描述。
工作空间	采集对象所属的DataWorks工作空间。
数据源类型	采集对象的类型，默认为**Hologres**。

b. 单击**下一步**。

ii. 选择采集对象。

a. 在**选择采集对象**页签，选择相应的**数据源**。

目前仅支持采集已绑定的Hologres实例的元数据。如果没有您需要的数据源，请单击**去新建**，创建新的数据源。详情请参见配置Hologres数据源。

b. 单击**测试采集连通性**后的**开始测试**，待显示**测试成功**，说明已连通DataWorks元数据服务网络。

> **说明** 如果显示**测试连通性未通过**，则您需要查看具体原因解决相关问题。

c. 单击**下一步**。

iii. 配置执行计划。

在**配置执行计划**页签，配置**执行计划**。

执行计划包括**按需执行**、**每月**、**每周**、**每天**及**每小时**。根据不同的执行周期，生成不同的执行计划，在相应执行计划的时间内，对目标数据源进行元数据采集。具体如下：

- 按需采集：根据实际业务需求，在业务需要时才会采集Hologres元数据。
- 月采集：即在每月的特定几天，在特定时间点自动采集一次Hologres元数据。

> 注意 部分月份不包含29、30、31日，请您谨慎选择月末日期。

如下图所示，在每月的1、11及21日的09:00，系统会自动采集一次Hologres元数据。**CRON 表达式**会根据您的配置自动生成。

- 周采集：即在每周的特定几天，在特定时间点自动采集一次Hologres元数据。

  如下图所示，在每周的星期一（MON）及星期天（SUN）的03:00，系统会自动采集一次Hologres元数据。**CRON 表达式**会根据您的配置自动生成。

  不输入**时间**时，则默认在每周指定几天的00:00:00采集。

- 天采集：即在每天特定的时间点自动采集一次Hologres元数据。

  如下图所示，在每天的01:00，系统会自动采集一次Hologres元数据。**CRON 表达式**会根据您的配置自动生成。

- 小时采集：即在每小时的第 `N*5分钟` 自动采集一次Hologres元数据。

  > **说明** 目前小时周期的采集任务，仅支持选择的周期时间为第5分钟的倍数。

  如下图所示，在每小时的第5分钟和第10分钟，系统会自动采集一次Hologres元数据。**CRON 表达式**会根据您的配置自动生成。

- 单击**下一步**。

iv. 确认信息。

在**信息确认**页签，确认新建采集器的内容。

6. 确认配置信息无误后，单击**确认**，成功创建采集器。
7. 在**Hologres元数据采集**页面，您可以查看并管理目标采集器的相关信息。

主要操作说明如下：

- 您可以查看相应采集器的运行**状态**、**运行计划**、**上次运行时间**、**上次消耗时间**、**平均运行耗时**及上次运行时更新及添加的表数量。
- 单击目标采集器**操作**列的**详情**、**编辑**、**删除**、**运行**、**停止**，执行相应操作：
  - **详情**：查看该采集器的**采集器名称**、**数据源类型**及**执行计划**。
  - **编辑**：修改该采集器的信息。
  - **删除**：删除该采集器。
  - **运行**：单击**运行**，即可根据该采集所配置的任务采集数据。仅当**执行计划**配置为**按需执行**时，才会生成**运行**操作，其他周期计划的任务不涉及该操作。
  - **停止**：停止运行中的采集器。仅运行中状态的采集器会显示该操作按钮。

其他操作

- 数据总览

  您可以在数据总览页面查看当前地域（Region）下配置了数据采集器的Hologres数据库统计信息和表信息，进入数据总览详细步骤请参见数据总览。

- 查找表

  数据地图支持通过表名，表描述、字段名及字段描述等搜索表，同时还可以通过表所在类目，项目或数据库进行表过滤。查找表详细步骤请参见查找表。

- 查看表详情

  您可以单击目标表名称跳转至表详情页面，查看表的基础信息、产出信息和血缘信息等表的详情信息。查看表详情步骤请参见查看表详情。

11.共享集群（MaxCompute BI加速版）

11.1. 概述

交互式分析Hologres共享集群（MaxCompute BI加速版）是针对MaxCompute交互式分析场景设计的在线查询加速服务，基于Hologres存储计算分离的云原生架构，使用共享集群资源的形式，加快了对存储在MaxCompute中数据的访问。本文为您介绍MaxCompute BI加速版的相关内容。

应用场景

MaxCompute BI加速版适用的场景如下：

- 需要加速查询MaxCompute数据的场景。
- 查询MaxCompute数据的频率低，并且要求查询时延较低的场景。
- 使用复杂查询来查询大量MaxCompute数据，小规格的独享实例难以满足要求的场景。
- 加速查询MaxCompute数据并对接BI分析的场景。例如，对接Tableau、Quick BI及帆软等。

产品优势

MaxCompute BI加速版的优势如下：

- 提供兼容PostgreSQL的查询接口，与MaxCompute底层数据存储无缝连通，无需移动数据，即可为数据分析场景提供稳定、快速以及低成本的查询服务，可达到秒级响应。
- 使用Servless架构，消除了资源扩展和弹性方面的限制，无需提前规划容量即可应对规模的快速变化，节约运维成本。
- 与MaxCompute相比，访问MaxCompute数据的速度提升了5~10倍。

功能介绍

MaxCompute BI加速版的实例在资源上是共享的，但实例作为单独的个体也拥有丰富的功能，具体如下：

- 内置查询引擎。
  - 支持通过创建外部表加速查询MaxCompute数据。该查询方式，数据仍然存储在MaxCompute中。
  - 不支持创建内部表并将数据导入Hologres进行查询。如果您需要导入数据至Hologres内部表查询，可以购买独享资源实例。
- 支持标准SQL语句。

  Hologres兼容PostgreSQL 11，您可以使用标准的Postgres语言进行查询。
- 权限管理。

  共享集群中的每个实例都具有独立的权限管控，其权限管控同独享资源的实例。您可以在当前实例新增及删除用户，也可以根据业务需求为不同用户授予不同的访问权限。

> **说明** 共享集群的实例，除了不支持内部表相关功能以及不能导入数据外，其他功能同独享资源实例。

产品形态对比

Hologres独享实例、Hologres共享集群（MaxCompute BI加速版）以及Lightning的功能对比如下表所示。

对比项		Hologres独享实例	Hologres共享集群（MaxCompute BI加速版）	Lightning
功能	类别	支持Postgres11	支持Postgres11	支持Postgres8.2
参数优化	参数优化	支持根据数据库或用户调整参数。	支持根据数据库或用户调整参数。	不支持定制化优化，例如不能按照时区调整参数。
	SQL连接数/QPS	与实例规格有关，您可以执行 `show max_connections` 语句查看SQL连接数。	默认SQL连接数为*100*个。	固定SQL连接数为*20*个。
外部表查询	meta加载	加速查询数据可达到毫秒级响应。	加速查询数据可达到毫秒级响应。	查询响应较慢，响应速度随Maxcompute Project的表数量线性增加。
	数据类型	支持基本类型、DECIMAL2.0、DATETIME及ARRAY类型。	支持基本类型、DECIMAL2.0、DATETIME及ARRAY类型。	仅支持基本类型。
	引擎优化	不涉及	较Lightning的查询性能大幅提升。	不涉及
内部表查询	索引支持	支持Bitmap及Cluster索引。	无	无
	分布列指定	支持指定数据分布列，实现Local Join及Local Group By。	无	无
资源		独立资源，即独立享有所购买的计算和存储资源。	计算资源共享，存储资源独立。	计算资源共享，存储资源独立。

11.2. 产品定价

Hologres共享集群（MaxCompute BI加速版）经过一年多的公测，服务稳定、生产可用，服务自2022年3月1日起正式开始按量计费。本文为您介绍Hologres共享集群（MaxCompute BI加速版）实例的计费情况。

生命周期管理

Hologres共享集群（MaxCompute BI加速版）根据SQL扫描的数据量进行收费，采用根据使用量的后付费模式。Hologres共享集群（MaxCompute BI加速版）实例周期管理流程如下：

- 在阿里云官网购买一个Hologres共享集群（MaxCompute BI加速版）实例，开通成功后即可运行该实例，系统将会汇总前一天内所有SQL对应的数据，推送账单并从您的账户中自动扣除费用。

- 阿里云账号欠费后，该账号下所有Hologres共享集群（MaxCompute BI加速版）实例将会变成欠费状态。如果未启用延期免停，欠费后实例将进入停机状态。如已开启延期免停，实例会延期停机，具体延期情况请参见用户中心。
- 若实例停机后的15天内未充值，在此期间将不会推送账单，该实例将会被释放，系统将会从管控台将实例删除，实例相关的数据也会被清除不能再恢复。
- 阿里云提供延停权益，即当按量付费的资源发生欠费后，提供一定额度或时长继续使用云服务的权益。延停期间正常计费。延停的权益额度不是欠费总额的上限。您延停的额度或时长根据您在阿里云的历史消费情况等因素，每个月自动计算并更新。更多信息请参见延期免停权益。

计费模式

Hologres共享集群（MaxCompute BI加速版）服务自2022年3月1日起正式开始按量计费。

每执行一次Hologres共享集群（MaxCompute BI加速版）查询作业，实时数仓Hologres共享集群（MaxCompute BI加速版）将根据作业的输入数据量进行计费。

Hologres共享集群（MaxCompute BI加速版）查询作业的计费公式为：

一次实时数仓Hologres共享集群（MaxCompute BI加速版）查询作业费用=查询输入数据量×单价（0.4469元/GB）

> **说明**
>
> - Hologres共享集群（MaxCompute BI加速版）服务使用单独的计算资源。
> - Hologres共享集群（MaxCompute BI加速版）按照每条查询作业扫描的数据量（每条查询至少 10 MB，即若由于存在查询启动开销，即使原始扫描数据量未满 10 MB，也需要按照 10 MB 计算）计费。
> - 账单每小时推送一次，每次推送两个小时前的账单数据。
> - 不查询不产生任何费用。
> - 查询分区表时，您可以应用分区过滤条件，减少数据扫描量并提升查询性能。

备注

针对MaxCompute Lightning的迁移用户和参与Hologres共享集群（MaxCompute BI加速版）公测的用户，自2022年3月1日至2023年3月1日可使用同一个账号享受5折优惠，优惠无需申请，自动生效。

常见问题

- 如何查询一段时间的扫描数据量？

  您可以使用如下SQL命令核对明细数据，执行此命令的账号需要具有实例的Superuser角色。

- 语法示例

```
select  instance as instance_id
        ,round(
            sum(
                case when read_bytes is null or read_bytes < 10*1024*1024
                  then 10*1024*1024
                else read_bytes
                    end
            )/1024/1024
        ) as "scan_size_mb" -- 每条查询至少10MB
        ,count(1) as "sql_count"
from    hologres.query_log
where   status = 'SUCCESS'
and     command_tag in ('SELECT')
and     query_end >= '开始时间'::TIMESTAMPTZ
and     query_end < '结束时间'::TIMESTAMPTZ
group by 1
order by 3 desc
;
```

- 使用示例

  例如查询2022年1月1日10:00至2022年1月1日11:00的扫描数据量的示例如下。

```
select  instance as instance_id
        ,round(
            sum(
                case when read_bytes is null or read_bytes < 10*1024*1024
                  then 10*1024*1024
                else read_bytes
                    end
            )/1024/1024
        ) as "scan_size_mb" -- 每条查询至少10MB
        ,count(1) as "sql_count"
from    hologres.query_log
where   status = 'SUCCESS'
and     command_tag in ('SELECT')
and     query_end >= '2022-01-01 10:00:00+08'::TIMESTAMPTZ
and     query_end < '2022-01-01 11:00:00+08'::TIMESTAMPTZ
group by 1
order by 3 desc
;
```

- 如何查询一段时间的每条SQL扫描数据量？

  您可以使用如下SQL命令核对查询一段时间内每条SQL扫描数据量，执行此命令的账号需要具有实例的Superuser角色。

- 语法示例

```
select  usename
        ,status
        ,query_id
        ,datname
        ,command_tag
        ,duration
        ,message
        ,query_start
        ,query_end
        ,query_date
        ,query
        ,case when read_bytes is null or read_bytes < 10*1024*1024
              then 10*1024*1024
         else read_bytes
         end as billing_read_bytes -- 每条查询至少10MB
        ,application_name
from    hologres.query_log
where   status = 'SUCCESS'
and     command_tag in ('SELECT')
and     query_end >= '开始时间'::TIMESTAMPTZ
and     query_end < '结束时间'::TIMESTAMPTZ
;
```

billing_read_bytes字段即为用于计费的扫描数据量。

- 使用示例

例如查询2022年3月1日10:00至2022年3月1日11:00每条SQL扫描数据量的示例如下。

```
select  usename
        ,status
        ,query_id
        ,datname
        ,command_tag
        ,duration
        ,message
        ,query_start
        ,query_end
        ,query_date
        ,query
        ,case when read_bytes is null or read_bytes < 10*1024*1024
              then 10*1024*1024
         else read_bytes
         end as billing_read_bytes     -- 每条查询至少10MB
        ,application_name
from    hologres.query_log
where   status = 'SUCCESS'
and     command_tag in ('SELECT')
and     query_end >= '2022-03-01 10:00:00+08'::TIMESTAMPTZ
and     query_end < '2022-03-01 11:00:00+08'::TIMESTAMPTZ
;
```

11.3. 使用说明

本文以一个简单的流程为您介绍如何使用交互式分析Hologres共享集群（MaxCompute BI加速版）。

前提条件

- 阿里云账号注册，详情请参见阿里云账号注册流程。
- 实名认证，详情请参见个人实名认证或企业实名认证和个体工商户认证。
- 开通Quick BI，详情请参见Quick BI购买、升级、降级、续费、欠费。

使用限制

共享集群（MaxCompute BI加速版）的使用限制如下：

- 不支持使用Hologres的内部表功能，仅支持新建外部表加速查询MaxCompute数据。
- 实例为共享资源，不支持升配及降配。
- 查询分区表时，每次最多扫描1024个分区。
- 目前不支持MAP、LIST及STRUCT数据类型。MaxCompute与Hologres的数据类型映射请参见数据类型汇总。
- 每个查询中对单张表的最大数据扫描量为200GB。
- 提交的查询语句大小不超过100KB。
- 默认查询超时时间为30分钟，您可以通过statement_timeout参数修改查询超时时间。
- 单个查询语句的Join及Group By的总数量最大值为20。
- 不支持使用COPY命令进行导入导出数据。
- 仅支持创建hive_compatible extension，不支持创建其他extension，详情请参见GET_JSON_OBJECT。
- 不支持对表进行Analyze操作。
- 共享集群部支持DML操作，仅支持如下DDL。
  - DATABASE
    - CREATE DATABASE。
    - ALTER DATABASE。
    - DROP DATABASE。
  - SCHEMA
    - CREATE SCHEMA。
    - ALTER SCHEMA。
    - DROP SCHEMA。
  - FOREIGN TABLE
    - CREATE FOREIGN TABLE。
    - DROP FOREIGN TABLE。
    - IMPORT FOREIGN SCHEMA。

共享集群（MaxCompute BI加速版）的使用流程

1. 购买实例。
   i. 使用阿里云主账号登录阿里云官网。

ii. 进入Hologres产品详情页。

iii. 单击**立即购买**，进入购买页面。

iv. **商品类型**选择**共享集群（MaxCompute BI加速版）**，输入**实例名称**，选择目标**地域**，单击**立即购买**。

2. 新建数据库。

成功购买实例后，您可以进入Hologres的管理控制台，查看实例状态。

成功创建实例后，系统默认生成一个名为**postgres**的数据库，用于监控管理，实际业务需要您新建数据库，操作如下：

i. 在**Hologres引擎管理**页面，单击实例名称。

您也可以单击目标实例**操作**列的**管理**，进入实例详情页。

ii. 在实例详情页左侧导航栏，单击**DB管理**。

iii. 在**DB管理**页面，单击**新增Database**。

iv. 在**新增Database**对话框，输入**Database名称**，并根据实际业务选择是否**开启简单权限模型**。

Hologres为您提供了**专家模式授权**和**简单权限模型**两套授权体系。

专家模式授权与PostgreSQL的权限模型完全一致，简称专家模式，详情请参见专家权限模型。

简单权限模型是Hologres基于实际业务，为了简化授权操作而抽象的一套简单权限模型（SPM），详情请参见简单权限模型概述。

创建数据库时，为了方便权限管理，建议您**开启简单权限模型**。

v. 单击**确定**。

您可以在**DB管理**页面，查看已创建的数据库。

3. 连接开发工具。

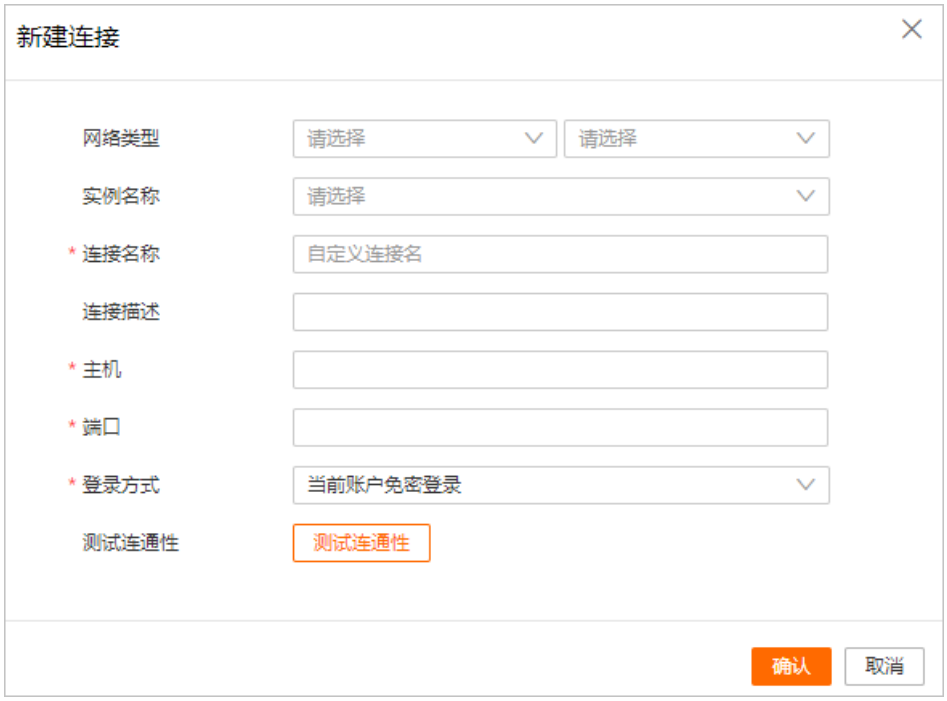
Hologres兼容Postgres，提供JDBC/ODBC Driver。新建数据库后，您可以使用实例连接开发工具进行数据开发。您可以根据业务需求选择合适的开发工具，详情请参见概述。

本次试验以Hologres的自研开发工具HoloWeb为您演示，如何使用实例连接开发工具，步骤如下：

i. 在Hologres管理控制台的**Hologres引擎管理**页面，单击**登录Hologres数据库**，进入**HoloWeb**开发界面。

ii. 单击**连接管理 > 数据连接**。

iii. 配置**新建连接**对话框的参数。



参数	描述
连接名称	自定义的连接名称。
连接描述	连接的描述信息。

参数	描述
网络类型	选择需要连接的网络类型及地域。网络类型如下： ■ 公网 ■ VPC
实例名称	根据实际业务选择所选地域已创建的实例。 选择实例后，会自动显示该实例的主机和端口。
主机	Hologres实例的网络域名。 进入Hologres管理控制台的实例详情页，从**实例配置**获取主机。
端口	Hologres实例的网络端口。 进入Hologres管理控制台的实例详情页，从**实例配置**获取端口。
AccessKey ID	系统会自动显示当前账号的AccessKey ID。 您也可以单击AccessKey 管理获取AccessKey ID。
AccessKey Secret	系统会自动显示当前账号的AccessKey Secret。 您也可以单击AccessKey 管理获取。
测试连通性	检测数据连接是否成功： ■ 成功：显示**测试通过**。 ■ 不成功：显示**测试不通过**。

iv. 单击**确认**。

4. MaxCompute加速查询。

实例成功连接HoloWeb后，您可以创建外部表，加速查询MaxCompute的数据。

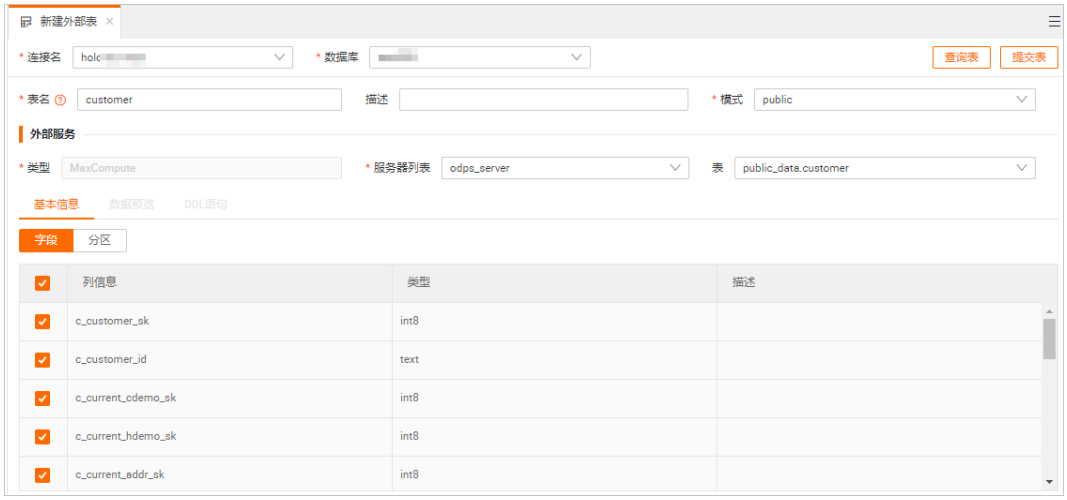
本次实验以在HoloWeb中新建外部表查询MaxCompute公共数据集中public_data项目的表数据为例，步骤如下：

> **说明** 获取表的方法请参见公开数据集。

i. 新建外部表。

a. 单击**连接管理 > MaxCompute加速 > 创建外部表**，使用可视化的方式创建外部表。

b. 在**新建外部表**的编辑页面，配置各项参数。



参数描述如下表所示。

参数	描述
连接名	已配置的连接名称。
数据库	Hologres的数据库名称。
表名	新建的Hologres外部表名称。
描述	新建的Hologres外部表信息描述。
模式	模式名称。 您可以选择默认创建的模式**public**，也可以选择新建的模式名称。
类型	外部表类型。 目前仅支持MaxCompute。
服务器列表	创建外部查询MaxCompute数据是通过外部服务器来实现的。您可以直接调用Hologres底层已创建的名为**odps_server**的外部表服务器。详细原理请参见Postgres FDW。
表	MaxCompute的项目名和表名。 格式为**project.table_name**。 **说明** ■ 目前暂不支持跨地域查询外部表数据。 ■ 输入表名称后，会显示外部源表的所有字段。您可以选择同步外部源表的部分或所有字段至Hologres。

c. 输入MaxCompute表的名称，就可以索引出表的字段，您可以根据实际业务，选择需要同步的表字段，单击**提交表**。

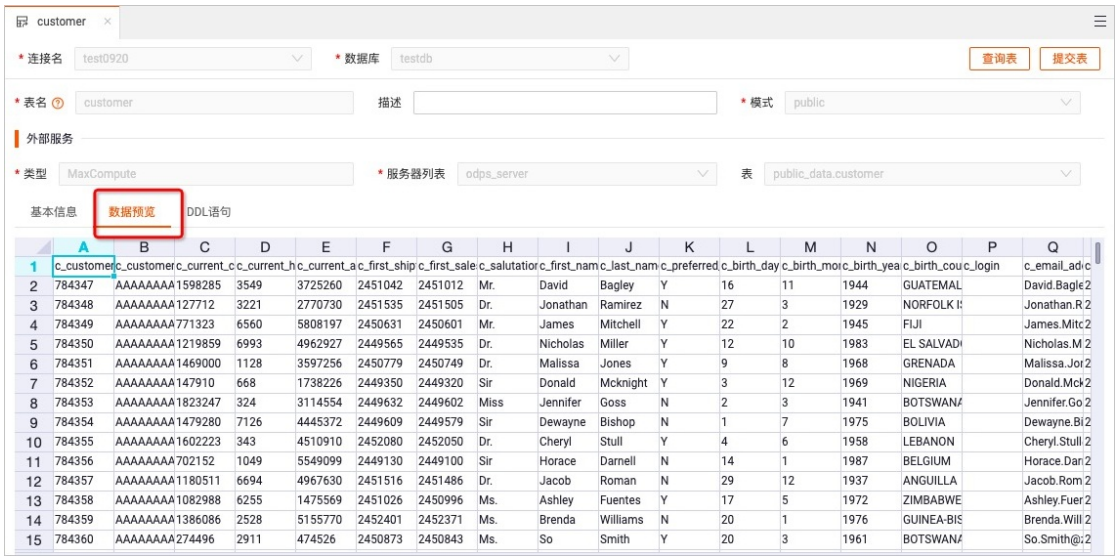
您也可以新建一个Query查询窗口，使用SQL语句批量创建外部表。示例语句如下。创建Query查询窗口请参见SQL窗口 。

```
IMPORT FOREIGN SCHEMA public_data LIMIT to(
  customer,
  customer_address,
  customer_demographics,
  inventory,item,
  date_dim,
  warehouse)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'update');
```

ii. 预览外部表数据。

成功新建外部表后，选择**连接管理 > 我的连接**。

鼠标双击新建的外部表，在表编辑页面单击**数据预览**，查看MaxCompute表的数据。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	c_customer	c_customer	c_current_c	c_current_h	c_current_a	c_first_ship	c_first_sale	c_salutatior	c_first_nam	c_last_nam	c_preferred	c_birth_day	c_birth_mor	c_birth_yea	c_birth_cou	c_login	c_email_adc
2	784347	AAAAAAAA	1598285	3549	3725260	2451042	2451012	Mr.	David	Bagley	Y	16	11	1944	GUATEMAL		David.Bagle 2
3	784348	AAAAAAAA	127712	3221	2770730	2451535	2451505	Dr.	Jonathan	Ramirez	N	27	3	1929	NORFOLK I:		Jonathan.R 2
4	784349	AAAAAAAA	771323	6560	5808197	2450631	2450601	Mr.	James	Mitchell	Y	22	2	1945	FIJI		James.Mitc 2
5	784350	AAAAAAAA	1219859	6993	4962927	2449565	2449535	Dr.	Nicholas	Miller	Y	12	10	1983	EL SALVAD		Nicholas.M 2
6	784351	AAAAAAAA	1469000	1128	3597256	2450779	2450749	Dr.	Malissa	Jones	Y	9	8	1968	GRENADA		Malissa.Jor 2
7	784352	AAAAAAAA	147910	668	1738226	2449350	2449320	Sir	Donald	Mcknight	Y	3	12	1969	NIGERIA		Donald.Mck 2
8	784353	AAAAAAAA	1823247	324	3114554	2449632	2449602	Miss	Jennifer	Goss	N	2	3	1941	BOTSWANA		Jennifer.Go 2
9	784354	AAAAAAAA	1479280	7126	4445372	2449609	2449579	Sir	Dewayne	Bishop	N	1	7	1975	BOLIVIA		Dewayne.Bi 2
10	784355	AAAAAAAA	1602223	343	4510910	2452080	2452050	Dr.	Cheryl	Stull	Y	4	6	1958	LEBANON		Cheryl.Stull 2
11	784356	AAAAAAAA	702152	1049	5549099	2449130	2449100	Sir	Horace	Darnell	N	14	1	1987	BELGIUM		Horace.Dar 2
12	784357	AAAAAAAA	1180511	6694	4967630	2451516	2451486	Dr.	Jacob	Roman	N	29	12	1937	ANGUILLA		Jacob.Rom 2
13	784358	AAAAAAAA	1082988	6255	1475569	2451026	2450996	Ms.	Ashley	Fuentes	Y	17	5	1972	ZIMBABWE		Ashley.Fuer 2
14	784359	AAAAAAAA	1386086	2528	5155770	2452401	2452371	Ms.	Brenda	Williams	N	20	1	1976	GUINEA-BIS		Brenda.Will 2
15	784360	AAAAAAAA	274496	2911	474526	2450873	2450843	Ms.	So	Smith	Y	20	3	1961	BOTSWANA		So.Smith@: 2

您也可以在**Query查询**模块中新建SQL窗口，使用SQL命令批量创建外部表，创建SQL窗口详情请参见SQL窗口。批量创建外部表的示例语句如下。

```
IMPORT FOREIGN SCHEMA public_data LIMIT to(
  customer,
  customer_address,
  customer_demographics,
  inventory,item,
  date_dim,
  warehouse)
  FROM server odps_server INTO PUBLIC options(if_table_exist 'update');
```

iii. 加速查询外部表数据。

加速查询外部表数据的示例SQL语句如下。

```
# SQL1：查询首选客户分布情况，按人数降序排列。
SELECT c_preferred_cust_flag,
       count(*) AS cnt
FROM customer
WHERE c_preferred_cust_flag IS NOT NULL
GROUP BY c_preferred_cust_flag
ORDER BY cnt DESC LIMIT 10;
# SQL2：查询客户年龄人数大于1000的分布情况，按人数降序排列。
SELECT c_birth_year,
       count(*) AS cnt
FROM customer
WHERE c_birth_year IS NOT NULL
GROUP BY c_birth_year HAVING count(*) > 1000
ORDER BY cnt DESC LIMIT 10;
# SQL3：查询客户所在城市的人数大于10的分布情况，按人数降序排序。
SELECT ca_city,
       count(*) AS cnt
FROM customer ,
     customer_address
WHERE c_current_addr_sk = ca_address_sk
  AND ca_city IS NOT NULL
GROUP BY ca_city HAVING count(*) > 10
ORDER BY cnt DESC LIMIT 10;
# SQL4：查询首选客户出生于1980~1990年且所在城市的人数大于10的分布情况，按人数降序排列。
SELECT ca_city,
       count(*) AS cnt
FROM customer ,
     customer_address
WHERE c_current_addr_sk = ca_address_sk
  AND c_birth_year >= 1980
  AND c_birth_year < 1990
  AND c_preferred_cust_flag = 'Y'
  AND ca_city IS NOT NULL
GROUP BY ca_city HAVING count(*) > 10
ORDER BY cnt DESC LIMIT 10;
```

5. 连接BI工具进行可视化分析。

Hologres兼容Postgres，支持直接对接BI工具。新建外部表加速查询MaxCompute后，您可以根据业务情况选择连接合适的BI工具，进行可视化分析。Hologres支持的BI工具请参见概述。

本次实验以Quick BI为例，为您介绍如何连接BI工具。步骤如下：

i. 登录Quick BI管理控制台。

ii. 添加数据源。

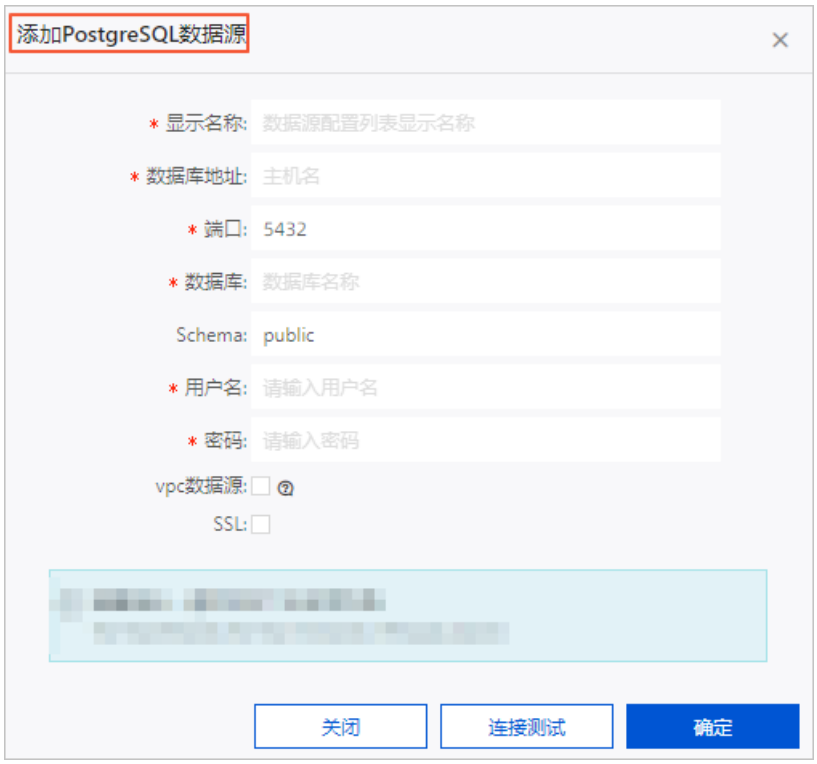
a. 在Quick BI管理控制台页面，单击顶部菜单栏的**工作空间**。

b. 在**工作空间**页面，单击左侧导航栏的**数据源**。

c. 在**数据源**页面，单击右上角的**+新建数据源**。

d. 选择**云数据库 > PostgreSQL**。

e. 配置**添加PostgreSQL数据源**对话框的各项参数。



参数说明如下表所示。

参数	描述
显示名称	自定义的显示名称。
数据库地址	连接的Hologres实例的公共网络地址。
端口	连接的Hologres实例的公共网络端口。
数据库	连接的Hologres实例的数据库名称。
Schema	默认为**public**。
用户名	当前账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
密码	当前账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
vpc数据源	不勾选
SSL	不勾选

参数配置完成后，您可以单击**连接测试**查看连接情况，如果显示**数据源连通性正常**，则表明Hologres与Quick BI成功连接。

f. 单击**确定**。

iii. 可视化分析数据。

成功连接数据源后，在**数据源 > 我的数据源**页面，单击已创建的数据源，显示当前数据库中的所有表。您可以使用即席分析SQL的方式可视化分析数据，步骤如下：

a. 在**数据源**页面，单击右上角的**即席分析SQL**，进入**即席分析SQL**页面。

b. 您可以根据业务需求输入查询SQL语句，并单击**执行**。

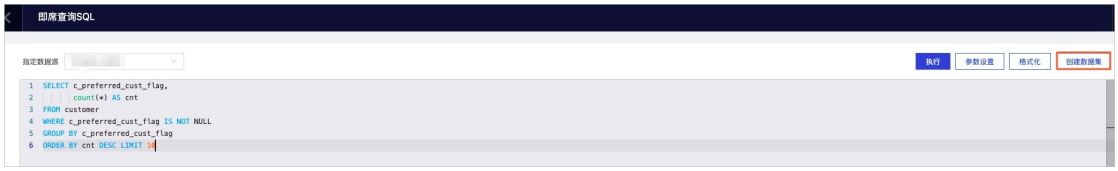
c. 获取需要查询的数据后，单击**创建数据集**。

d. 配置**保存自定义SQL**对话框的**名称**、**位置**及SQL参数。

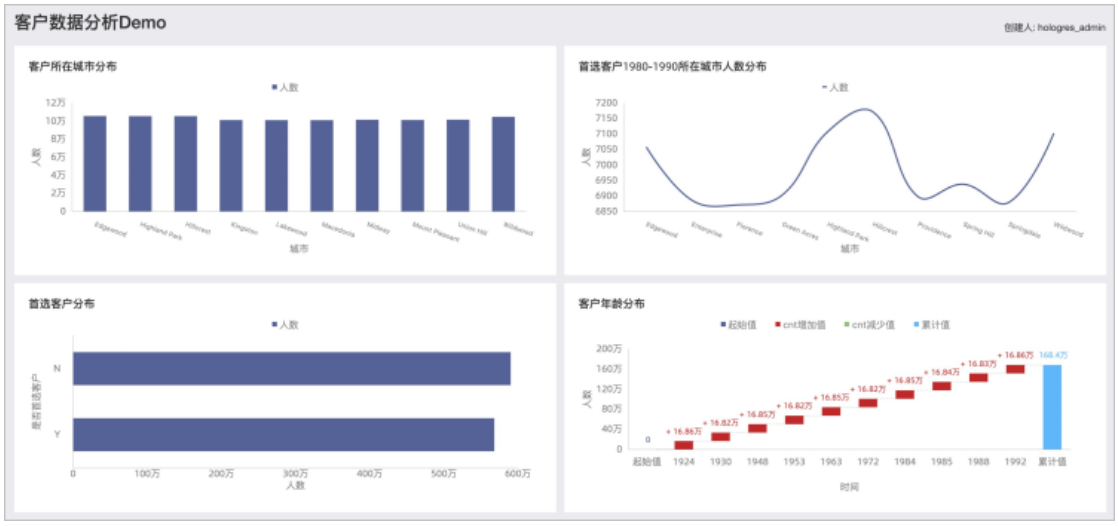
e. 单击**确定**。

成功创建数据集后，您可以在**数据集**中将需要展示的数据制作成报表，可视化分析表数据，详情请参见即席分析SQL建模。

本次试验使用即席查询SQL的方式创建数据集，如下图所示。



根据业务需求展示的报表示例如下。



数据类型映射

当前Hologres支持的数据类型与MaxCompute数据类型映射关系，请参见数据类型汇总。

11.4. 管理控制台实例管理

Hologres管理控制台为您提供了实例的操作及管理信息。您可以在Hologres管理控制台对交互式分析Hologres共享集群（MaxCompute BI加速版）实例进行停机、恢复及删除操作，并查看实例的相关信息。

- 停机实例。

共享集群（MaxCompute BI加速版）实例使用的是**按量付费**模式，您可以根据业务情况选择是否停机，释放资源。停机之后实例将不能运行，您无法在实例内运行任务。

- 恢复实例。

  如果您需要恢复已经被手动停机的实例，可以单击**恢复**。恢复成功之后，您可以继续使用该实例进行开发。

  > **说明** 实例恢复需要几分钟，请您耐心等待。

- 删除实例。

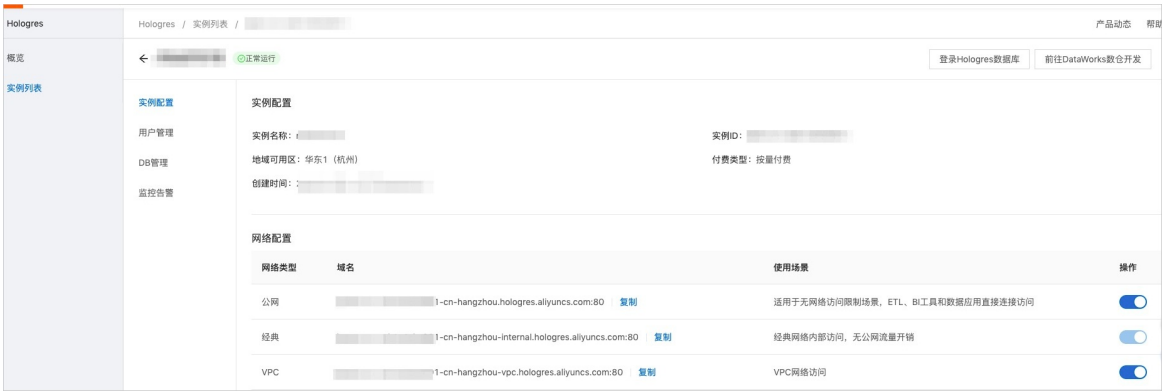
  如果您不需要再使用当前实例，可以单击**删除**，删除该实例。实例删除后，实例内的表和对象也会一并删除，并且无法恢复，请您谨慎操作。

- 管理实例。

  在管理控制台单机目标实例后的**管理**，进入实例的详情页，您可以在该页面查看实例信息、进行用户管理和DB管理，及查看监控告警等，具体如下：

  - **实例配置**

    实例配置为您提供当前实例的基本信息，包括实例的名称、ID、地域、付费类型、以及开通时间和网络类型等。详情请参见实例配置。



- 用户管理

  您可以在用户管理页面管理Hologres实例内的用户，包括新增、删除及授权用户等，以便于您对实例内的用户进行更细致化的管理，减免了繁琐的SQL授权操作。详情请参见用户管理。

- **DB管理**

  DB管理页面为您提供当前实例的所有DB管理服务，您可以在当前页面查看DB、创建DB并为DB选择相应的权限策略等。详情请参见DB管理。

- 监控告警

  监控告警页面为您提供了实例的资源管理服务。您可以在该页面查看当前实例的连接数、QPS以及Query延迟。关于指标的具体含义请参见Hologres管控台的监控指标。

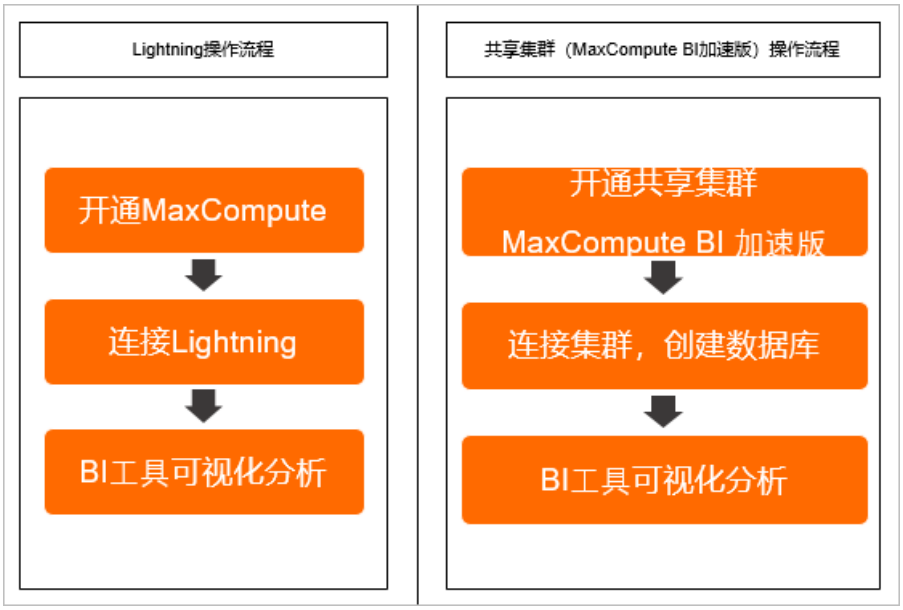
11.5. 迁移Lightning至共享集群（MaxCompute BI加速版）

11.5.1. 迁移Lightning至共享集群概述

由于Lightning后期会逐步下线，下线后不再继续维护。因此，建议您选择共享集群（MaxCompute BI加速版）处理业务。本文将为您介绍将原MaxCompute Lightning服务迁移至Hologres共享集群（MaxCompute BI加速版）的相关流程，方便您进行业务迁移。

Lightning与共享集群（MaxCompute BI加速版）操作流程

Lightning与共享集群（MaxCompute BI加速版）的操作流程分别如下图所示。



操作流程说明

如下步骤为您介绍将原MaxCompute Lightning服务迁移至Hologres共享集群（MaxCompute BI加速版）并对接BI工具的简单操作流程。

1. 开通共享集群（MaxCompute BI加速版）并在Hologres管理控制台查看实例的详细信息。
2. 新建数据库并进行授权。
3. 连接BI工具进行可视化分析。

   共享集群（MaxCompute BI加速版）兼容PostgreSQL生态，支持直接对接各种BI分析工具。连接BI工具时需要您修改原Lightning服务的Endpoint为共享集群（MaxCompute BI加速版）实例的Endpoint，然后再进行可视化分析。常见的BI工具请参见BI工具概述。

更多详细的操作指导请参见：

- 迁移Lightning至共享集群（数据服务场景）
- 迁移Lightning至共享集群（Quick BI场景）

迁移用户权限

共享集群（MaxCompute BI加速版）的权限采用与Lightning服务不同的权限控制方式，需要您重新给用户授予相关权限，具体操作请参见授予RAM用户实例的开发权限。

11.5.2. 迁移Lightning至共享集群（数据服务场景）

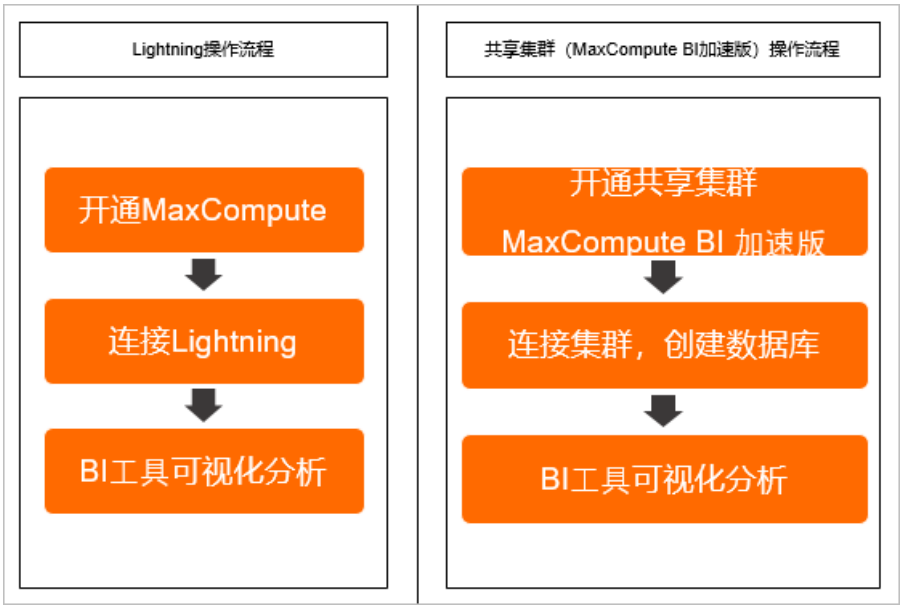
共享集群（MaxCompute BI加速版）是针对MaxCompute交互式分析场景设计的在线查询加速服务，基于Hologres存储计算分离的云原生架构，以共享集群资源的形式，加快存储在MaxCompute中的数据访问。本文内容将指导您在使用Lightning集群的前提下，如何使用Hologres共享集群（MaxCompute BI加速版）对接DataWorks数据服务。

背景信息

由于Lightning后期会逐步下线，下线后不再继续维护。您可以将共享集群（MaxCompute BI加速版）认为是Lightning的升级版，性能和服务更优于Lightning。逐步使用共享集群（MaxCompute BI加速版）替换Lightning。

操作流程说明

针对使用数据服务场景的操作流程，Lightning和共享集群（MaxCompute BI加速版）的操作流程对比如下图所示。



因此，在数据服务场景下，使用Hologres共享集群（MaxCompute BI加速版）对接DataWorks数据服务的操作流程具体如下：

- 开通共享集群（MaxCompute BI加速版）并确认开通实例的详细信息。
- 新建数据库并进行授权。
- 在数据服务新建Hologres数据源，重新创建数据服务任务。

步骤一：开通共享集群（MaxCompute BI加速版）

在迁移Lightning服务之前，需要您先开通共享集群（MaxCompute BI加速版），然后在管理控制台查看自己的共享集群实例是否开通成功。

1. 单击购买，进入交互式分析Hologres共享集群（MaxCompute BI加速版）购买页面进行如下配置。
   - **商品类型**：选择**共享集群（MaxCompute BI加速版）**。
   - **地域**：请根据业务实际情况进行选择。
   - **实例名称**：支持长度为2~64个字符的名称。
2. 配置完成后，单击**立即购买**进行订单确认。
3. 勾选*我已阅读并同意交互式分析Hologres共享集群（MaxCompute BI加速版）服务协议*，单击**立即开通**。

订购开通交互式分析Hologres服务实例，一般需要5-10分钟，请您耐心等待。

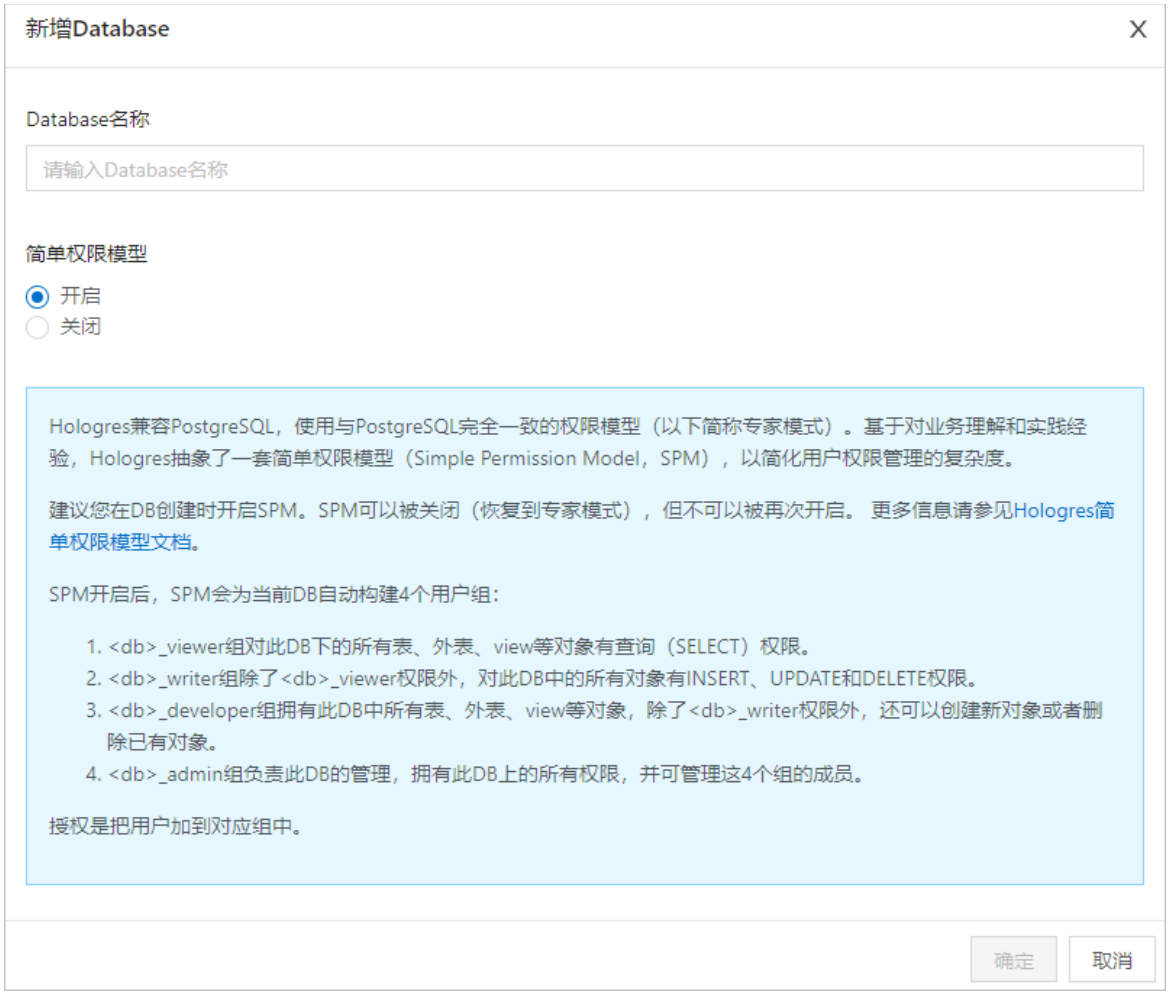
4. 购买成功后，进入Hologres的管理控制台。
5. 在**Hologres引擎管理**页面，单击目标实例名称。

   您也可以单击目标实例**操作**列的**管理**，进入实例详情页，查看实例的详细信息。

步骤二：新建数据库

成功创建实例后，系统默认生成一个名为postgres的数据库，用于监控管理。实际业务需要您按照如下操作指导新建数据库。

1. 进入实例详情页左侧导航栏的**DB管理**页面，单击**新增Database**。
2. 在**新增Database**对话框，输入**Database名称**，并根据实际业务选择是否开启简单权限模型。创建数据库时，为了方便权限管理，建议您选择**开启**简单权限模型。



Hologres为您提供了专家模式授权和简单权限模型两套授权体系。

- 专家模式授权：与PostgreSQL的权限模型完全一致，简称专家模式，详情请参见专家权限模型。
- 简单权限模型：Hologres基于实际业务，为了简化授权操作而抽象的一套简单权限模型（SPM），详情请参见简单权限模型概述。

3. 单击**确定**完成数据库创建，您可以在**DB管理**页面，查看已创建的数据库。

步骤三：对接DataWorks的数据服务

共享集群（MaxCompute BI加速版）兼容PostgreSQL生态，支持直接对接各种BI分析工具。若您是使用其他BI工具，需要将原Lightning的域名修改为共享集群的域名之后，再进行可视化分析。

如下内容以DataWorks的数据服务为例，指导您进行BI工具的连接。

1. 配置Hologres数据源。
   i. 登录DataWorks管理控制台，在页面上方选择目标区域后，在左侧导航栏单击进入**工作空间列表**页面。
   ii. 单击目标工作空间操作的**进入数据集成**按钮。
   iii. 在**数据集成**首页的**数据源**页签，单击**进入数据源管理**页面。
   iv. 单击顶部菜单栏的**新增数据源**，在**大数据存储**区域，选择**Hologres**。
   v. 配置**新增Hologres数据源**对话框的参数。

具体参数说明如下表所示：

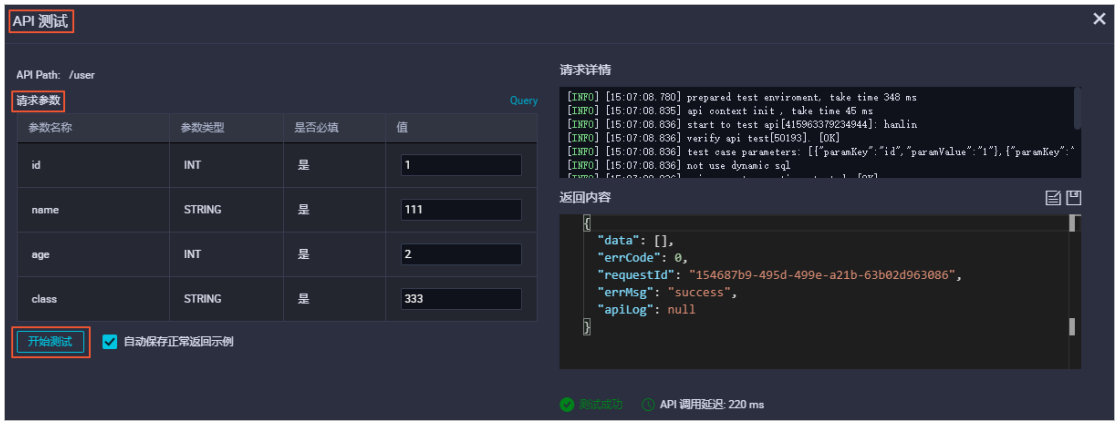
参数	描述
数据源类型	目前仅支持**阿里云实例模式**。
数据源名称	数据源名称必须是字母、数字和下划线的组合，并且以字母开头。
数据源描述	数据源的信息描述，不得超过80个字符。
适用环境	■ **开发** ■ **生产** **说明** 如果使用的是标准模式的DataWorks工作空间，则需要配置该参数。
实例ID	需要同步的Hologres实例ID。 您可以进入Hologres管理控制台，获取实例ID。

参数	描述
数据库名	Hologres的数据库名称。
AccessKey ID	您可以单击AccessKey 管理，获取AccessKey ID。
AccessKey Secret	您可以单击AccessKey 管理，获取AccessKey Secret。
资源组连通性	选择**数据服务**。 您需要保证公共资源组和数据源是可以联通的。

2. 生成并配置API。
   i. 登录DataWorks管理控制台，在页面上方选择目标区域后，在左侧导航栏单击进入**工作空间列表**页面。
   ii. 单击目标工作空间操作的**进入数据开发**按钮。
   iii. 在**数据开发**页面，单击顶部菜单栏左侧的图标，选择**全部产品 > 数据开发 > 数据服务**。
   iv. 在**服务开发**页面，单击顶部菜单栏的图标，选择**API > 生成API**。

   您可以选择**向导模式**或**脚本模式**生成API。
   - 使用**向导模式**生成并配置API，详情请参见通过向导模式生成API。
   - 使用**脚本模式**生成并配置API，详情请参见通过脚本模式生成API。
3. 测试API。
   i. 生成API后，在右侧菜单栏依次进行属性、请求参数和返回参数的配置并保存。
   ii. 在API编辑页面，单击顶部菜单栏右侧的**测试**。

iii. 在**API测试**对话框，检查输入的**请求参数**，单击**开始测试**。



如果**API测试**对话框页面底部显示**测试成功**，则表示API测试通过。您也可以使用数据服务的测试API模块来完成测试，详情请参见测试API。

4. 发布并查看API。

   API测试成功之后，在API编辑页面，单击顶部菜单栏右侧的**发布**。将API发布至API网关，并上架至API市场，详情请参见发布API。您也可以在**数据服务**页面，单击顶部菜单栏右侧的**服务管理**，查看已发布的API名称，查看API详情。

5. 调用API。

   如果您需要调用已成功发布的API，请参见客户端调用API示例。

迁移用户权限

共享集群（MaxCompute BI加速版）的权限控制方式与Lightning不同，需要您重新给共享集群（MaxCompute BI加速版）用户授予相关权限。

为共享集群（MaxCompute BI加速版）用户授权的具体操作请参见授予RAM用户实例的开发权限。

11.5.3. 迁移Lightning至共享集群（Quick BI场景）

共享集群（MaxCompute BI加速版）是针对MaxCompute交互式分析场景设计的在线查询加速服务，基于Hologres存储计算分离的云原生架构，以共享集群资源的形式，加快存储在MaxCompute中的数据访问。本文内容将指导您在Quick BI场景下如何将原MaxCompute Lightning服务迁移至Hologres共享集群（MaxCompute BI加速版）。

前提条件

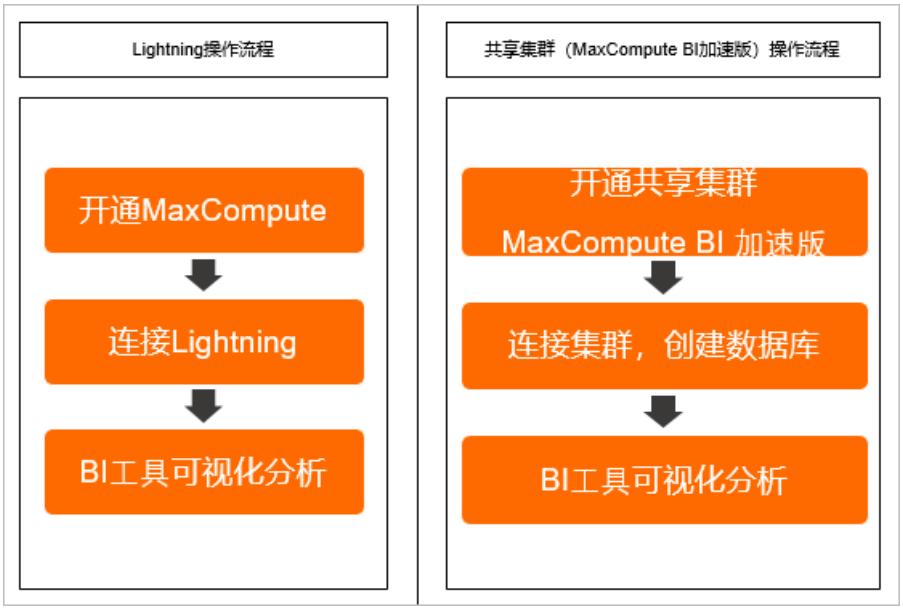
在开始本教程之前，请确保您已经开通并使用MaxCompute Lightning服务。

背景信息

由于Lightning后期会逐步下线，下线后不再继续维护。您可以将共享集群（MaxCompute BI加速版）认为是Lightning的升级版，性能和服务更优于Lightning。更多BI工具也可以按照本文描述步骤进行数据源替换，Hologres支持的BI工具请参见BI分析及可视化。

操作流程说明

针对使用Quick BI可视化分析的场景，Lightning和共享集群（MaxCompute BI加速版）的操作流程对比如下图所示。



因此，将Lightning业务迁移至Hologres共享集群（MaxCompute BI加速版）的操作流程具体如下：

- 步骤一：开通共享集群（MaxCompute BI加速版）
- 步骤二：新建数据库
- 步骤三：配置Quick BI一键替换数据源

步骤一：开通共享集群（MaxCompute BI加速版）

在迁移Lightning服务之前，需要您先开通共享集群（MaxCompute BI加速版），然后在管理控制台查看自己的共享集群实例是否开通成功。

1. 单击购买，进入交互式分析Hologres共享集群（MaxCompute BI加速版）购买页面进行如下配置。
   - **商品类型**：选择**共享集群（MaxCompute BI加速版）**。
   - **地域**：请根据业务实际情况进行选择。
   - **实例名称**：支持长度为2~64个字符的名称。
2. 配置完成后，单击**立即购买**进行订单确认。
3. 勾选*我已阅读并同意交互式分析Hologres共享集群（MaxCompute BI加速版）服务协议*，单击**立即开通**。

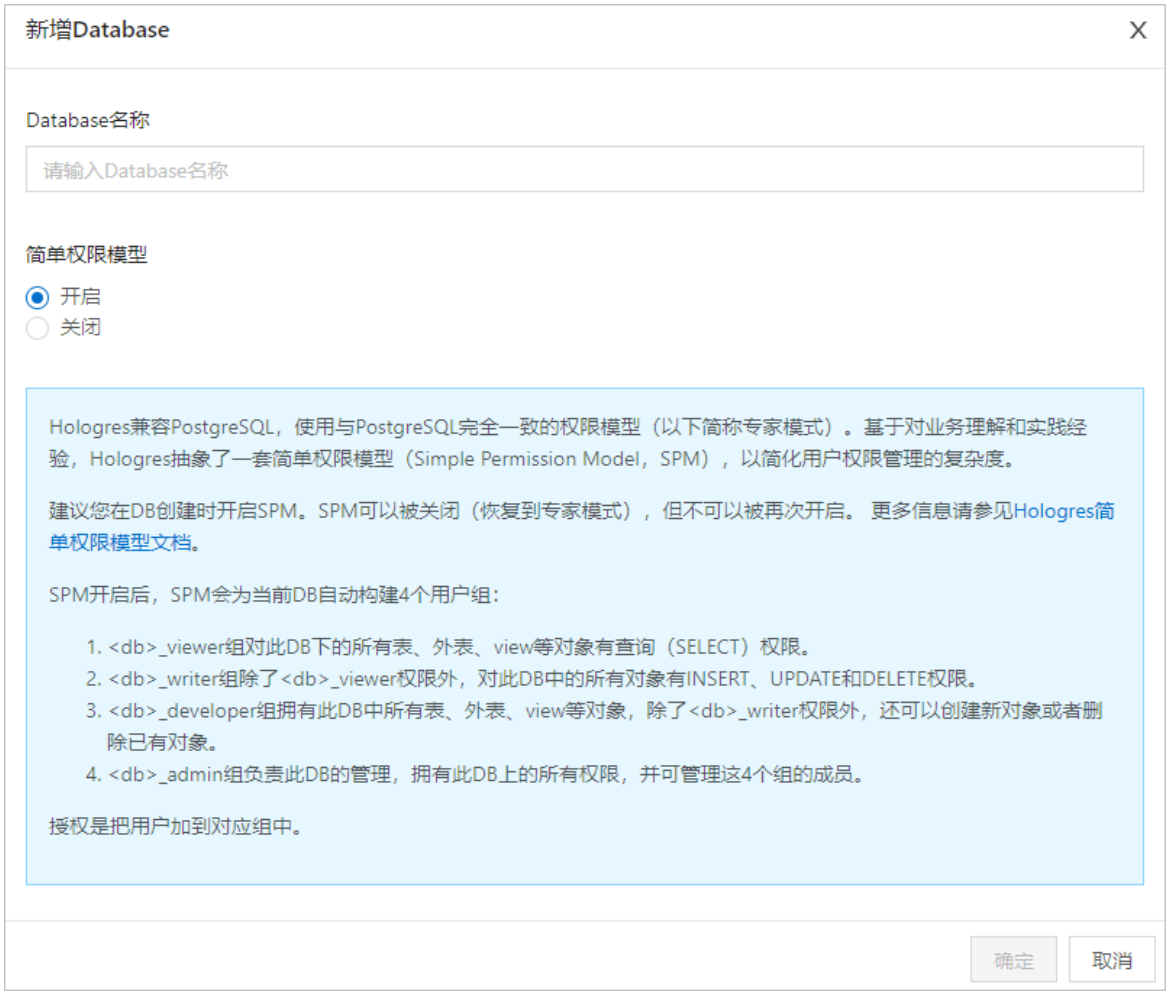
   订购开通交互式分析Hologres服务实例，一般需要5-10分钟，请您耐心等待。
4. 购买成功后，进入Hologres的管理控制台。
5. 在**Hologres引擎管理**页面，单击目标实例名称。

   您也可以单击目标实例**操作**列的**管理**，进入实例详情页，查看实例的详细信息。

步骤二：新建数据库

成功创建实例后，系统默认生成一个名为postgres的数据库，用于监控管理。实际业务需要您按照如下操作指导新建数据库。

1. 进入实例详情页左侧导航栏的**DB管理**页面，单击**新增Database**。
2. 在**新增Database**对话框，输入**Database名称**，并根据实际业务选择是否开启简单权限模型。创建数据库时，为了方便权限管理，建议您选择**开启**简单权限模型。



Hologres为您提供了专家模式授权和简单权限模型两套授权体系。

- 专家模式授权：与PostgreSQL的权限模型完全一致，简称专家模式，详情请参见专家权限模型。
- 简单权限模型：Hologres基于实际业务，为了简化授权操作而抽象的一套简单权限模型（SPM），详情请参见简单权限模型概述。

3. 单击**确定**完成数据库创建，您可以在**DB管理**页面，查看已创建的数据库。

步骤三：配置Quick BI一键替换数据源

共享集群（MaxCompute BI加速版）兼容PostgreSQL生态，支持直接对接各种BI分析工具。若您是使用其他BI工具，需要将原Lightning的域名修改为共享集群的域名之后，再进行可视化分析。

如下内容以Quick BI为例，指导您进行数据源替换操作。Quick BI提供一键替换的功能，您可以通过Quick BI将原Lightning的数据源替换为Hologres数据源。替换完成后，原Lightning的仪表盘、报表等数据将会默认修改为Hologres数据源。

> **说明** 在执行此操作之前，请确保您创建的外部表名称同Lightning表名称一致。

1. 登录Quick BI管理控制台，在页面上方选择工作空间。
2. 在**数据源**页面，单击我的数据列表中目标数据源的**替换**按钮。

3. 在**替换数据源**页面的**云数据库**页签，单击选择**Hologres**。
4. 配置**替换为Hologres数据源**对话框的参数。

具体参数说明如下表所示：

参数	说明
显示名称	显示名称可以根据需求自定义。
数据库地址	连接的Hologres的服务器地址。目前仅支持公网和经典网络（内网），进入Hologres管理控制台的实例详情页，从**实例配置**获取。获取方式请参见。

参数	说明
端口	连接的Hologres的端口地址。目前仅支持公网和经典网络（内网），进入Hologres管理控制台的实例详情页，从**实例配置**获取。
数据库	填写创建完成的目标数据库名称。
Schema	默认为**public**，您也可以使用其他创建的Schema。
用户名	当前账号的Access ID，您也可以单击AccessKey 管理获取AccessKey ID。
密码	当前账号的Access Key，您也可以单击AccessKey 管理获取AccessKey ID。

5. 单击**确定**完成数据源切换。

迁移用户权限

共享集群（MaxCompute BI加速版）的权限控制方式与Lightning不同，因此在完成数据源替换之后，需要您重新给共享集群（MaxCompute BI加速版）用户授予相关权限。

为共享集群（MaxCompute BI加速版）用户授权的具体操作请参见授予RAM用户实例的开发权限。

12.性能调优

12.1. 优化内部表的性能

本文为您介绍在Hologres中对内部表性能进行调优的最佳实践。

更新统计信息

统计信息决定是否能够生成正确的执行计划。例如，Hologres需要收集数据的采样统计信息，包括数据的分布和特征、表的统计信息、列的统计信息、行数、列数、字段宽度、基数、频度、最大值、最小值、长键值、分桶分布特征等信息。这些信息将为优化器更新算子执行预估COST、搜索空间裁剪、估算最优JOIN ORDER、估算内存开销、估算并行度，从而生成更优的执行计划。关于统计信息更多的介绍，请参见Using Explain。

统计信息的收集也存在一定局限，主要是**针对非实时、手动触发或者周期性触发，不一定反映最准确的数据特征**。您需要先检查explain的信息，查看explain中包含的统计信息是否正确。统计信息中每个算子的rows和width表示该算子的行数和宽度。

- 查看统计信息是否正确
  - 通过查看执行计划

    未及时同步统计信息导致生成较差的执行计划，示例如下：

    tmp1表的数据量为1000万行，tmp表的数据量为1000行。 Hologres默认统计信息中的行数为1000行，通过执行explainSQL语句，如下展示结果所示，tmp1表的行数与实际的行数不符，该展示结果表明未及时更新统计信息。

```
Seq Scan on tmp1 (cost=0.00..5.01 rows=1000 width=1)
```

```
tt=# explain  select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Partial Aggregate  (cost=0.00..10.11 rows=1 width=8)
   ->  Gather Motion  (cost=0.00..10.11 rows=10 width=8)
         ->  Partial Aggregate  (cost=0.00..10.11 rows=10 width=8)
               ->  Hash Join  (cost=0.00..10.11 rows=1 width=1)
                     Hash Cond: (tmp.a = tmp1.b)
                     ->  Parallelism (Gather Exchange)  (cost=0.00..5.04 rows=1000 width=1)
                           ->  DecodeNode  (cost=0.00..5.04 rows=1000 width=1)
                                 ->  Seq Scan on tmp  (cost=0.00..5.01 rows=1000 width=1)
                     ->  Hash  (cost=5.04..5.04 rows=1000 width=1)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..5.04 rows=1000 width=1)
                                 ->  DecodeNode  (cost=0.00..5.04 rows=1000 width=1)
                                       ->  Seq Scan on tmp1  (cost=0.00..5.01 rows=1000 width=1)
 Optimizer: HQO version 0.8.0
(13 rows)
```

- 通过查看系统表

  您可以通过查系统表查看行数、宽度等是否正确。

  - 查询系统表hologres.hg_table_properties中的analyze_tuple列，确认数据行数是否正确。或者直接查看Scan节点中rows的值。
  - 查询系统表hologres.hg_stats可看到每一列的直方图、平均宽度、不同值的数量等信息，如下图所示。

```
tt=# \d hologres.hg_stats;
                    Table "hologres.hg_stats"
        Column         |  Type   | Collation | Nullable | Default
-----------------------+---------+-----------+----------+---------
 schemaname            | text    |           | not null |
 tablename             | text    |           | not null |
 attname               | text    |           | not null |
 inherited             | text    |           | not null |
 null_frac             | real    |           |          |
 avg_width             | integer |           |          |
 n_distinct            | real    |           |          |
 most_common_vals      | text[]  |           |          |
 most_common_freqs     | real[]  |           |          |
 histogram_bounds      | text[]  |           |          |
 correlation           | real    |           |          |
 most_common_elems     | text[]  |           |          |
 most_common_elem_freqs | real[]  |           |          |
 elem_count_histogram  | real[]  |           |          |
Indexes:
    "hg_stats_name_idx" btree (schemaname, tablename, attname)
```

- 更新统计信息

  tmp1和tmp表Join时，正确的**explain**信息展示为*数据量大的表tmp1在数据量小的表tmp上方，Hash Join应该采用数据量小的tmp表*。因为tmp1表未及时更新统计信息，导致Hologres选择tmp1表创建Hash表进行Hash Join，效率较低，并且可能造成OOM（Out Of Memory，内存溢出）。因此，需要参与Join的两张表均执行 `analyze` 收集统计信息，语句如下。

```
analyze tmp;
analyze tmp1;
```

  执行 `analyze` 命令后，Join的顺序正确。数据量大的表tmp1在数据量小的表tmp上方，使用数据量小的表tmp做Hash表，如下图所示。并且tmp1表展示的行数为1000万行，表明统计信息已经更新。

```
tt=# explain  select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Partial Aggregate  (cost=0.00..428.66 rows=1 width=8)
   ->  Gather Motion  (cost=0.00..428.66 rows=10 width=8)
         ->  Partial Aggregate  (cost=0.00..428.65 rows=10 width=8)
               ->  Hash Join  (cost=0.00..428.65 rows=1 width=1)
                     Hash Cond: (tmp1.b = tmp.a)
                     ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                           ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                 ->  Seq Scan on tmp1  (cost=0.00..146.17 rows=10000001 width=4)
                     ->  Hash  (cost=5.04..5.04 rows=1000 width=1)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..5.04 rows=1000 width=1)
                                 ->  DecodeNode  (cost=0.00..5.04 rows=1000 width=1)
                                       ->  Seq Scan on tmp  (cost=0.00..5.01 rows=1000 width=1)
 Optimizer: HQO version 0.8.0
(13 rows)
```

当发现explain返回结果中 `rows=1000` ，说明缺少统计信息。一般性能不好时，其原因通常是优化器缺少统计信息，需要通过及时更新统计信息，执行 `analyze <tablename>` ，可以简单快捷优化查询性能。

- 推荐更新统计信息的场景

  推荐在以下情况下运行 `analyze <tablename>` 命令。

  - 导入数据之后。
  - 大量的INSERT、UPDATE以及DELETE操作之后。
  - 内部表、外部表均需要ANALYZE。
  - 分区表针对父表做ANALYZE。
  - 如果遇到以下问题，您需要先执行 `analyze <tablename>` ，再运行导入任务，可以系统地优化效率。
    - 多表JOIN超出内存OOM。

      通常会产生 `Query executor exceeded total memory limitation xxxxx: yyyy bytes used` 报错。
    - 导入效率较低。

      在Hologres查询或导入数据时，效率较低，运行的任务长时间不结束。

- 优化器Join Order算法
  - 当SQL Join关系比较复杂时，Join的表多时，优化器消耗在连接关系最优选择上的时间会更多，调整Join Order策略，在一定场景下会降低Query Optimization的时间，设置优化器Join Order算法语法如下。

    ```
    set optimizer_join_order = '<value>';
    ```

  - 参数说明

参数	说明
value	优化器Join Order算法，有如下三种。 ■ *query*：不进行Join Order转换，按照SQL书写的连接顺序执行，优化器开销最低。 ■ *greedy*：通过贪心算法进行Join Order的探索，优化器开销适中。 ■ *exhaustive*：通过动态规划算法进行Join Order转换，会生成最优的执行计划，但优化器开销最高。 默认值为exhaustive。

  - 补充说明
    - 使用默认的***exhaustive***算法可以全局探索最优的执行计划，但对于很多表的Join（例如表数量大于10），优化耗时可能较高。使用query或者greedy算法可以减少优化器耗时，但无法生成最优的执行计划。
    - 可以为Session和Query级别分别设置优化器Join Order策略。

设置适合的Shard数

Shard数代表查询执行的并行度。Shard个数对查询性能影响至关重要，Shard数设置少，会导致并行度不足。Shard数设置过多，也会引起查询启动开销大，降低查询效率，同时引起小文件过多，占用内存更多的元数据管理空间。设置与实例规格匹配的Shard数，可以改善查询效率，降低内存开销。

Hologres为每个实例设置了默认的Shard数，Shard数约等于实例中用于核心查询的Core数。这里的core数，略小于实际购买的Core数（实际购买的Core会被分配给不同的节点，包括查询节点、接入节点、控制节点和调度节点等）。不同规格实例默认的Shard数，请参见实例规格概述。当实例扩容后，扩容之前旧的DB对应的默认Shard数不会自动修改，需要根据实际情况修改Shard数，扩容后新建DB的Shard数为当前规格的默认数量。默认的Shard数是已经考虑扩容的场景，在资源扩容5倍以上的场景中，建议考虑重新设置Shard数，小于5倍的场景，无需修改也能带来执行效率的提升。具体操作请参见Table Group设置最佳实践。

如下场景需要修改Shard数：

- 扩容后，因业务需要，原有业务有规模增长，需要提高原有业务的查询效率。此时，您需要创建新的Table Group，并为其设置更大的Shard数。原有的表和数据仍然在旧的Table Group中，您需要将数据重新导入新的Table Group中，完成Resharding的过程。
- 扩容后，需要上线新业务，但已有业务并不变化。此时，建议您创建新的Table Group，并为其设置适合的Shard数，并不调整原有表的结构。

> **说明** 一个DB内可以创建多个Table Group，但所有Table Group的Shard总数之和不应超过Hologres推荐的默认Shard数，这是对CPU资源的最有效利用。

选择合适的分布列（Distribution Key）

分布列（Distribution Key）用于将数据划分到多个Shard，划分均衡可以避免数据倾斜。多个相关的表设计为相同的Distribution Key，可以起到Local Join的加速效果。创建表时，您可以通过如下原则选择合适的分布列：

- Distribution Key设置建议
  - 选择JOIN查询时的连接条件列作为分布列。
  - 选择Group By频繁的列作为分布列。
  - 选择数据分布均匀离散的列作为分布列。
- 设置Distribution Key场景示例

  例如设置Distribution Key，表tmp和tmp1做Join，通过执行explain SQL语句看到执行计划中有Redistribution Motion，说明数据有重分布，没有Local Join，导致查询效率比较低。您需要重新建表并同时设置Join Key为Distribution Key，避免多表连接时数据重分布带来的额外开销。

```
tt=# explain select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Partial Aggregate  (cost=0.00..1554.71 rows=1 width=8)
   ->  Gather Motion  (cost=0.00..1554.71 rows=10 width=8)
         ->  Partial Aggregate  (cost=0.00..1554.70 rows=10 width=8)
               ->  Hash Join  (cost=0.00..1550.97 rows=10000001 width=1)
                     Hash Cond: (tmp.a = tmp1.b)
                     ->  Redistribute Motion  (cost=0.00..605.79 rows=10000001 width=4)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                                 ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                       ->  Seq Scan on tmp  (cost=0.00..146.17 rows=10000001 width=4)
                     ->  Hash  (cost=605.79..605.79 rows=10000001 width=4)
                           ->  Redistribute Motion  (cost=0.00..605.79 rows=10000001 width=4)
                                 ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                                       ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                             ->  Seq Scan on tmp1  (cost=0.00..146.17 rows=10000001 width=4)
 Optimizer: HQO version 0.8.0
(15 rows)
```

重新建表后两个表的DDL示例语句如下。

```
begin;
create table tmp(a int, b int, c int);
call set_table_property('tmp', 'distribution_key', 'a');
commit;
begin;
create table tmp1(a int, b int, c int);
call set_table_property('tmp1', 'distribution_key', 'b');
commit;
-- 设置分布列为Join Key。
select count(1) from tmp join tmp1 on tmp.a = tmp1.b ;
```

通过重新设置表的Distribution Key，再次执行explain SQL语句，可以看到执行计划中，红框内的算子被优化掉了，数据按照相同的Hash Key分布于Shard中。因为数据分布相同，Motion算子被优化（上图中红框内的算子），表明数据不会重新分布，从而避免了冗余的网络开销。

```
tt=# explain  select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Partial Aggregate  (cost=0.00..1178.71 rows=1 width=8)
   ->  Gather Motion  (cost=0.00..1178.71 rows=10 width=8)
         ->  Partial Aggregate  (cost=0.00..1178.70 rows=10 width=8)
               ->  Hash Join  (cost=0.00..1174.97 rows=10000001 width=1)
                     Hash Cond: (tmp.a = tmp1.b)
                     ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                           ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                 ->  Seq Scan on tmp  (cost=0.00..146.17 rows=10000001 width=4)
                     ->  Hash  (cost=417.79..417.79 rows=10000001 width=4)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                                 ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                       ->  Seq Scan on tmp1  (cost=0.00..146.17 rows=10000001 width=4)
 Optimizer: HQO version 0.8.0
(13 rows)
```

Hologres包含四种Motion Node，分别对应四种数据重分布场景，如下表所示。

类型	描述
Redistribute Motion	数据通过哈希分布或随机分布，Shuffle到一个或多个Shard。
Broadcast Motion	复制数据至所有Shard。 仅在Shard数量与广播的表的数量均较少时，Broadcast Motion的优势较大。
Gather Motion	汇总数据至一个Shard。
Forward Motion	用于联邦查询场景。外部数据源或执行引擎与Hologres执行引擎进行数据传输。

结合explain SQL语句执行结果您可以注意如下事项：

- 如果Motion算子耗时较高，则您可以重新设计分布列。
- Broadcast Motion只有在Shard数较少，且广播表的数量较少的场景下有优势。
- 如果统计信息错误，导致生成Gather Motion或Broadcast Motion，则您可以通过 `analyze tablename` 命令将其修改为更高效的Redistribute Motion分布方式。

- 您可以设置如下参数，禁止生成Motion算子，再对比查询耗时，示例语句如下。

```
-- 禁止生成Broadcast Motion。
set optimizer_enable_motion_broadcast = off;
-- 禁止生成Redistribute Motion。
set optimizer_enable_motion_redistribute = off;
```

- Group By优化

  Group By Key会导致数据按照分组列的Key重新分布数据，如果Group By耗时较高，您可以将Group By的列重新设置为分布列。

```
-- 数据按照a列的值进行Hash重分布。
select a from t1 group by a;
```

- 数据倾斜处理

  数据在多个Shard上分布不均匀会导致查询速度较慢，您可以通过如下语句判断数据分布是否存在倾斜。

```
select count(1) from t1 group by hg_shard_id;
```

  如果数据存在倾斜，则需要更改distribution_key，选择数据分布均匀离散的列作为分布列。

> **说明** 更改distribution_key需要重新创建表并导入数据。

关闭Dictionary Encoding

对于字符类型（包括Text/Char/Varchar）的相关查询，Dictionary Encoding或Decoding会减少比较字符串的耗时，但是会带来大量的Decode或Encode开销。

Hologres默认对所有的字符类型列建立Dictionary Encoding，您可以设置dictionary_encoding_columns为空，或关闭部分列的自动Dictionary Encoding功能。注意，修改Dictionary Encoding设置，会引起数据文件重新编码存储，会在一段时间内消耗一部分CPU和内存资源，建议在业务低峰期执行变更。

您可以通过上文*获取执行后的统计信息*获取Decode算子的耗时，如果耗时较高，则请关闭Decode。当表的字符类型字段较多时，建议关闭Dictionary Encoding功能可以改善性能。

当表的字符类型字段较多时，按需选择，可以不用将所有的字符类型都加入dictionary_encoding_columns。示例语句如下：

```
begin;
create table tbl (a int not null, b text not null, c int not null, d int);
call set_table_property('tbl', 'dictionary_encoding_columns', '');
commit;
```

SQL优化手段

您可以通过优化相应的SQL语句来提高查询效率，具体如下：

- 采用Fixed Plan优化SQL写入和更新。

  Fixed Plan是Hologres针对高吞吐数据写入、更新、删除场景的特定优化，通过简化的执行路径，实现数倍性能和吞吐的提升，配置方式和使用方法参考文档Fixed Plan加速SQL执行。

- PQE算子改写。

Hologres底层有Hologers（HQE，自研）和Postgres（PQE）两个执行引擎，如果SQL语句中包含Hologres执行引擎不支持的算子，则系统会将该算子发送至Postgres的执行引擎执行。此时查询的性能较差，需要修改相关查询语句。

通过执行计划（`explain` SQL）查询，若执行计划中出现`External SQL（Postgres）`则说明这部分的SQL是在Postgres中执行的。

具体示例如下：Hologres不支持*not in*，则会将*not in*操作转到外部查询引擎PQE(Postgres Query Engine)执行。建议将*not in*重写为*not exists*。优化前的SQL语句如下。

```
explain select * from tmp where a not in (select a from tmp1);
```

*Externa*算子代表该部分SQL语句是在外部引擎Postgres中执行的。

```
 Gather Motion  (cost=0.00..112640000002698064.00 rows=4000000 width=12)
   ->  Froward Motion  (cost=0.00..112640000002696272.00 rows=4000000 width=12)
         ->  ExecuteExternalSQL  (cost=0.00..112640000002696192.00 rows=4000000 width=12)
               External SQL(Postgres): SELECT c_8e2b1b58_0, c_8e2af178_1, c_8e11b4f8_2 FROM (SELECT * FROM hologres.hg_internal_holo_link('LINK_PARAMS', 1) AS T
0 INT4, c_8e2af178_1 INT4, c_8e11b4f8_2 INT4)) as T_RESULT WHERE (SELECT TRUE::bool AS c_8e2e3cd8_31 FROM (SELECT c_8e2e3198_30 FROM (SELECT CASE  WHEN c_8e2e0
UE::bool ELSE CASE  WHEN (c_8e2b2a58_29 > 0::Int8) THEN NULL::bool ELSE CASE  WHEN c_8e2b1b58_0 IS NULL THEN NULL::bool ELSE CASE  WHEN (c_8e2e0718_28 = 0::Int
E FALSE::bool END END END END AS c_8e2e3198_30 FROM (SELECT sum(c_b0578918_26 ) AS c_8e2e0718_28, sum(c_8e2b15b8_27 ) AS c_8e2b2a58_29 FROM (SELECT CASE  WHEN
2378_13) THEN 1::int4 ELSE 0::int4 END AS c_b0578918_26, CASE  WHEN c_8e2b2378_13 IS NULL THEN 1::int4 ELSE 0::int4 END AS c_8e2b15b8_27 FROM (SELECT * FROM ho
o_link('LINK_PARAMS', 0, TRUE) AS T_8e1e78b8(c_8e2b2378_13 INT4)) as T_RESULT ) AS T_AGG) as T_RESULT ) as T_RESULT WHERE (c_8e2e3198_30 = TRUE::bool)) as T_RES
               ->  Froward Motion  (cost=0.00..100000000000627.80 rows=30000003 width=4)
                     ->  Broadcast Motion  (cost=0.00..100000000000417.80 rows=30000003 width=4)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                                 ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                       ->  Seq Scan on tmp1  (cost=0.00..146.17 rows=10000001 width=4)
               ->  Froward Motion  (cost=0.00..680.37 rows=10000001 width=12)
                     ->  Parallelism (Gather Exchange)  (cost=0.00..470.37 rows=10000001 width=12)
                           ->  DecodeNode  (cost=0.00..465.90 rows=10000001 width=12)
                                 ->  Seq Scan on tmp  (cost=0.00..146.17 rows=10000001 width=12)
 Optimizer: HQO version 0.8.0
(14 rows)
```

优化后的SQL语句如下，不再使用外部查询引擎。

```
explain select * from tmp where not exists (select a from tmp1 where a = tmp.a);
```

```
tt=# explain select * from tmp where  not exists  (select a from tmp1 where a = tmp.a);
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Gather Motion  (cost=0.00..3260.35 rows=4000000 width=12)
   ->  Hash Left Anti Join  (cost=0.00..1471.55 rows=4000000 width=12)
         Hash Cond: (tmp.a = tmp1.a)
         ->  Parallelism (Gather Exchange)  (cost=0.00..470.37 rows=10000001 width=12)
               ->  DecodeNode  (cost=0.00..465.90 rows=10000001 width=12)
                     ->  Seq Scan on tmp  (cost=0.00..146.17 rows=10000001 width=12)
         ->  Hash  (cost=605.79..605.79 rows=10000001 width=4)
               ->  Redistribute Motion  (cost=0.00..605.79 rows=10000001 width=4)
                     ->  Parallelism (Gather Exchange)  (cost=0.00..417.79 rows=10000001 width=4)
                           ->  DecodeNode  (cost=0.00..416.30 rows=10000001 width=4)
                                 ->  Seq Scan on tmp1  (cost=0.00..146.17 rows=10000001 width=4)
 Optimizer: HQO version 0.8.0
(12 rows)
```

通过改写函数，将算子运行在Hologres原生执行引擎中，以下为函数改写建议。

Hologres原生引擎不支持的函数	建议改写的函数	样例	备注

Hologres原生引擎不支持的函数	建议改写的函数	样例	备注
not in	not exists	select * from tmp where not exist (select a from tmp1 where a = tmp.a);	不涉及。
regexp_split_to_table(string text, pattern text)	unnest(string_to_array)	`select name,unnest(string_to_array(age,',')) from demo;`	regexp_split_to_table支持正则表达式。

substring	extract(hour from to_timestamp(c1, 'YYYYMMDD HH24:MI:SS'))	`select cast(substring(c1, 13, 2) as int) AS hour from t2;`
		改写为：
		`select extract(hour from to_timestamp(c1, 'YYYYMMDD HH24:MI:SS')) from t2;`
			Hologres部分V0.10版本及更早版本不支持substring。

regexp_replace	replace	`select regexp_replace(c1::text,'-','') from t2;`
		改写为：
		`select replace(c1::text,'-','') from t2;`
			replace不支持正则表达式。

Hologres原生引擎不支持的函数	建议改写的函数	样例	备注

at time zone 'utc'	删除at time zone 'utc'	`select date_trunc('day',to_timestamp(c1, 'YYYYMMDD HH24:MI:SS') at time zone 'utc') from t2`
		改写为：
		`select date_trunc('day',to_timestamp(c1, 'YYYYMMDD HH24:MI:SS') ) from t2;`
			不涉及。

cast(text to timestamp)	to_timestamp	`select cast(c1 as timestamp) from t2;`
		改写为：
		`select to_timestamp(c1, 'yyyyMMdd hh24:mi:ss') from t2;`
			不涉及。

timestamp::text	to_char	`select c1::text from t2;`
		改写为：
		`select to_char(c1, 'yyyyMMdd hh24:mi:ss') from t2;`
			不涉及。

- 采用Insert on Conflict语法替代Update。

  Update命令采用表级锁，锁的粒度比较大，并发的update会造成更多的锁等待开销。Insert on Conflict语法采用行级锁，执行效率更高。

- 避免模糊查询。

  模糊查询（例如Like操作）不会创建索引，会退化为全表扫描。

- 结果缓存对查询的影响

  Hologres会默认对相同的查询、子查询结果进行缓存，重复执行会命中缓存结果。若要关闭缓存对性能测试的影响，请使用如下命令语句。

  ```
  set hg_experimental_enable_result_cache = off;
  ```

- OOM优化

命令	含义	参数取值范围	参数建议取值
set hg_experimental_scan_node_dop=<number>;	设置单个Shard内scan算子扫描的并发度，最大为单个Shard内扫描表的文件数。默认为性能最高的设置。适用于写入时出现OOM的场景。	0-512 **说明** 并发度太大可能造成OOM，导入或查询失败，甚至实例重启，以至于服务不可用。并发度太小会造成查询性能很差。	i. 使用 `show hg_experimental_scan_node_dop` 命令查看当前的并发度。 ii. 得到的结果除以2作为hg_experimental_scan_node_dop的参数值。 **说明** 也可自行决定并发度值减小的比例。 也可自行决定并发度值减小的比例。
set hg_experimental_dml_bulkload_dop=<number>;	设置BulkLoad写入或更新的并发度，最大为单个Shard内写入表的文件数，最小为1。默认为性能最高的设置。适用于写入时出现OOM的场景。	0-512 **说明** 并发度太大可能造成OOM，导入或查询失败，甚至实例重启，以至于服务不可用。并发度太小会造成查询性能很差。	i. 使用 `show hg_experimental_dml_bulkload_dop` 命令查看当前的并发度。 ii. 得到的结果除以2作为hg_experimental_dml_bulkload_dop的参数值。 **说明** 也可自行决定并发度值减小的比例。

- Order By Limit场景优化

  在Hologres V1.3之前版本，对Order By Limit场景不支持Merge Sort算子，生成执行计划时，在最后输出时还会做一次排序，导致性能相对较差。从1.3版本开始，引擎通过对order by limit场景优化，支持Merge Sort算子，实现多路归并排序，无需再进行额外的排序，提升了查询性能。若您的Hologres版本低于V1.3，请提交工单升级。

  优化示例如下。

- 建表DDL

```
begin;
create table test_use_sort_1
(
          uuid           text not null,
          gpackagename   text not null,
          recv_timestamp text not null
);
call set_table_property('test_use_sort_1', 'orientation', 'column');
call set_table_property('test_use_sort_1', 'distribution_key', 'uuid');
call set_table_property('test_use_sort_1', 'clustering_key', 'uuid:asc,gpackagename:asc
,recv_timestamp:desc');
commit;
--插入数据
insert into test_use_sort_1 select i::text, i::text, '20210814' from generate_series(1,
10000) as s(i);
--更新统计信息
analyze test_use_sort_1;
```

- 查询命令

```
select uuid from test_use_sort_1 order by uuid limit 5;
```

- 执行计划对比
  - Hologres V1.3之前版本（V1.1）的执行计划如下。

```
QUERY PLAN
Limit  (cost=0.00..0.61 rows=5 width=4)
 ->  Sort  (cost=0.00..0.00 rows=5 width=4)
       ->  Gather  (cost=0.00..0.61 rows=5 width=4)
             Sort Key: uuid
             ->  Limit  (cost=0.00..0.61 rows=5 width=4)
                   ->  Sort  (cost=0.00..0.61 rows=5 width=4)
                         Sort Key: uuid
                         ->  Exchange (Gather Exchange)  (cost=0.00..0.61 rows=5 width=4)
                               ->  Limit  (cost=0.00..0.61 rows=5 width=4)
                                     ->  Sort  (cost=0.00..6.12 rows=10000 width=4)
                                           Sort Key: uuid
                                           ->  Decode  (cost=0.00..5.11 rows=10000 width=4)
                                                 ->  Seq Scan on test_use_sort_1  (cost=0.00..5.00 rows=10000 width=4)
Optimizer: HQO version 1.1.0
```

  - Hologres V1.3版本的执行计划如下。

```
QUERY PLAN
Limit  (cost=0.00..0.54 rows=5 width=4)
 ->  Gather  (cost=0.00..0.54 rows=5 width=4)
       Merge Key: uuid
       ->  Limit  (cost=0.00..0.54 rows=5 width=4)
             ->  Sort  (cost=0.00..0.54 rows=5 width=4)
                   Sort Key: uuid
                   ->  Exchange (Gather Exchange)  (cost=0.00..0.54 rows=5 width=4)
                         ->  Limit  (cost=0.00..0.54 rows=5 width=4)
                               ->  Sort  (cost=0.00..5.38 rows=10000 width=4)
                                     Sort Key: uuid
                                     ->  Decode  (cost=0.00..5.10 rows=10000 width=4)
                                           ->  Seq Scan on test_use_sort_1  (cost=0.00..5.00 rows=10000 width=4)
Optimizer: HQO version 1.3.0
```

  从执行计划对比中可以看出，Hologres V1.3版本在最后输出会少一个排序，直接多路归并，提升了查询性能。

- Count Distinct优化
  - 改写为APPROX_COUNT_DISTINCT

    Count Distinct是精确去重，需要把相同key的记录shuffle到同一个节点去去重，比较耗费资源。Hologres实现了扩展函数APPROX_COUNT_DISTINCT，采用HyperLogLog基数估计的方式进行非精确的COUNT DISTINCT计算，提升查询性能。误差率平均可以控制在0.1%-1%以内，可以根据业务情况适当改写，详情请参见APPROX_COUNT_DISTINCT。
  - 使用UNIQ函数

    Hologres从 V1.3版本开始，支持UNIQ精确去重函数，在GROUP BY KEY的KEY基数较高时，比Count Distinct性能更好，更节省内存。当使用Count Distinct出现OOM时，可以使用UNIQ做替换，详情请参见UNIQ。
  - 设置合适的Distribution Key

    当有多个Count Distinct且是key是同一个并且数据离散均匀分布，建议将Count Distinct的key设置成Distribution Key，这样相同的数据可以分布相同的Shard，避免数据Shuffle。
  - 多个Count Distinct优化

当SQL中含有多个Count Distinct时，会先将每个Count Distinct单独求值，再基于join进行关联，而当from子句是subquery时，会导致subquery的重复计算，消耗资源较大、性能较差。从1.3版本开始，针对多个Count Distinct场景进行了优化，当Count Distinct数量大于1个时，会自动改写为UNIQ实现（结果和语义和当前实现是一样的，对上层透明），以减少基于当前模式下不必要的subquery重复计算，提升查询性能。若您的Hologres实例版本低于V1.3版本，请先提交工单升级实例。示例如下。

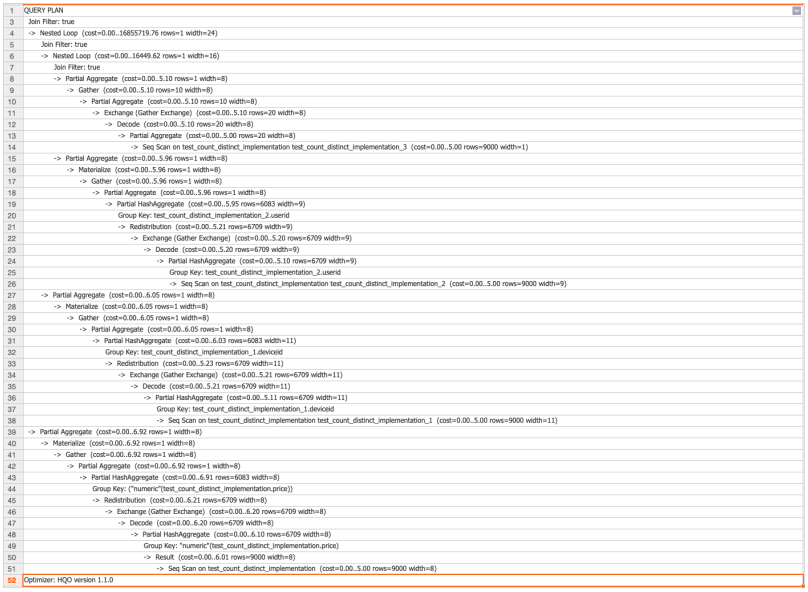
- 建表DDL

```
create table test_count_distinct_implementation
(
    id int
    ,dim1 int
    ,dim2 text
    ,dim3 int8
    ,userid text
    ,deviceid text
    ,price float8
    ,ds text
);
insert into test_count_distinct_implementation select i, i % 17, (i % 13)::text, i %
37, 'user_' || round(i % 97723)::text, 'device_' || floor(i % 179357)::text, (sqrt(i
% 24658)), '1' from generate_series(1, 5000)i;
insert into test_count_distinct_implementation select i, i % 19, (i % 13)::text, i %
37, 'user_' || round(i % 87723)::text, 'device_' || floor(i % 139557)::text, (sqrt(i
% 38658)), '2' from generate_series(1, 4000)i;
analyze test_count_distinct_implementation;
```

- 查询命令

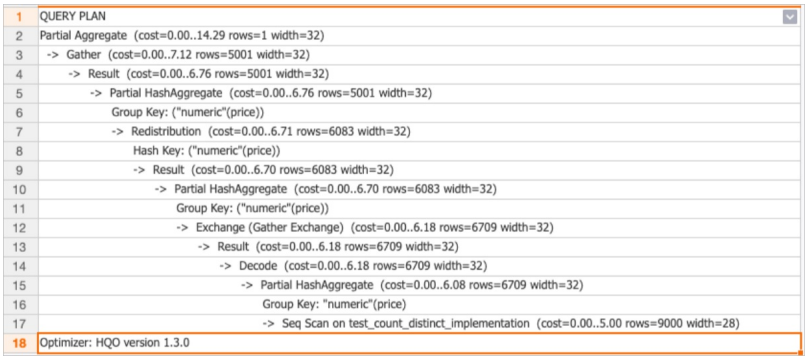
```
select count(1), count(distinct userid), count(distinct deviceid), sum(distinct price
::numeric) from test_count_distinct_implementation;
```

- 执行计划对比
  - Hologres V1.3之前版本（V1.1）的执行计划如下。



```
QUERY PLAN
 Join Filter: true
 ->  Nested Loop  (cost=0.00..16855719.76 rows=1 width=24)
       Join Filter: true
       ->  Nested Loop  (cost=0.00..16449.62 rows=1 width=16)
             Join Filter: true
             ->  Partial Aggregate  (cost=0.00..5.10 rows=1 width=8)
                   ->  Gather  (cost=0.00..5.10 rows=10 width=8)
                         ->  Partial Aggregate  (cost=0.00..5.10 rows=10 width=8)
                               ->  Exchange (Gather Exchange)  (cost=0.00..5.10 rows=20 width=8)
                                     ->  Decode  (cost=0.00..5.10 rows=20 width=8)
                                           ->  Partial Aggregate  (cost=0.00..5.00 rows=20 width=8)
                                                 ->  Seq Scan on test_count_distinct_implementation test_count_distinct_implementation_3  (cost=0.00..5.00 rows=9000 width=1)
             ->  Partial Aggregate  (cost=0.00..5.96 rows=1 width=8)
                   ->  Materialize  (cost=0.00..5.96 rows=1 width=8)
                         ->  Gather  (cost=0.00..5.96 rows=1 width=8)
                               ->  Partial Aggregate  (cost=0.00..5.96 rows=1 width=8)
                                     ->  Partial HashAggregate  (cost=0.00..5.95 rows=6083 width=9)
                                           Group Key: test_count_distinct_implementation_2.userid
                                           ->  Redistribution  (cost=0.00..5.21 rows=6709 width=9)
                                                 ->  Exchange (Gather Exchange)  (cost=0.00..5.20 rows=6709 width=9)
                                                       ->  Decode  (cost=0.00..5.20 rows=6709 width=9)
                                                             ->  Partial HashAggregate  (cost=0.00..5.10 rows=6709 width=9)
                                                                   Group Key: test_count_distinct_implementation_2.userid
                                                                   ->  Seq Scan on test_count_distinct_implementation test_count_distinct_implementation_2  (cost=0.00..5.00 rows=9000 width=9)
       ->  Partial Aggregate  (cost=0.00..6.05 rows=1 width=8)
             ->  Materialize  (cost=0.00..6.05 rows=1 width=8)
                   ->  Gather  (cost=0.00..6.05 rows=1 width=8)
                         ->  Partial Aggregate  (cost=0.00..6.05 rows=1 width=8)
                               ->  Partial HashAggregate  (cost=0.00..6.03 rows=6083 width=11)
                                     Group Key: test_count_distinct_implementation_1.deviceid
                                     ->  Redistribution  (cost=0.00..5.23 rows=6709 width=11)
                                           ->  Exchange (Gather Exchange)  (cost=0.00..5.21 rows=6709 width=11)
                                                 ->  Decode  (cost=0.00..5.21 rows=6709 width=11)
                                                       ->  Partial HashAggregate  (cost=0.00..5.11 rows=6709 width=11)
                                                             Group Key: test_count_distinct_implementation_1.deviceid
                                                             ->  Seq Scan on test_count_distinct_implementation test_count_distinct_implementation_1  (cost=0.00..5.00 rows=9000 width=11)
 ->  Partial Aggregate  (cost=0.00..6.92 rows=1 width=8)
       ->  Materialize  (cost=0.00..6.92 rows=1 width=8)
             ->  Gather  (cost=0.00..6.92 rows=1 width=8)
                   ->  Partial Aggregate  (cost=0.00..6.92 rows=1 width=8)
                         ->  Partial HashAggregate  (cost=0.00..6.91 rows=6083 width=8)
                               Group Key: ("numeric"(test_count_distinct_implementation.price))
                               ->  Redistribution  (cost=0.00..6.21 rows=6709 width=8)
                                     ->  Exchange (Gather Exchange)  (cost=0.00..6.20 rows=6709 width=8)
                                           ->  Decode  (cost=0.00..6.20 rows=6709 width=8)
                                                 ->  Partial HashAggregate  (cost=0.00..6.10 rows=6709 width=8)
                                                       Group Key: "numeric"(test_count_distinct_implementation.price)
                                                       ->  Result  (cost=0.00..6.01 rows=9000 width=8)
                                                             ->  Seq Scan on test_count_distinct_implementation  (cost=0.00..5.00 rows=9000 width=8)
 Optimizer: HQO version 1.1.0
```

  - Hologres V1.3版本的执行计划如下。



```
QUERY PLAN
Partial Aggregate  (cost=0.00..14.29 rows=1 width=32)
 ->  Gather  (cost=0.00..7.12 rows=5001 width=32)
       ->  Result  (cost=0.00..6.76 rows=5001 width=32)
             ->  Partial HashAggregate  (cost=0.00..6.76 rows=5001 width=32)
                   Group Key: ("numeric"(price))
                   ->  Redistribution  (cost=0.00..6.71 rows=6083 width=32)
                         Hash Key: ("numeric"(price))
                         ->  Result  (cost=0.00..6.70 rows=6083 width=32)
                               ->  Partial HashAggregate  (cost=0.00..6.70 rows=6083 width=32)
                                     Group Key: ("numeric"(price))
                                     ->  Exchange (Gather Exchange)  (cost=0.00..6.18 rows=6709 width=32)
                                           ->  Result  (cost=0.00..6.18 rows=6709 width=32)
                                                 ->  Decode  (cost=0.00..6.18 rows=6709 width=32)
                                                       ->  Partial HashAggregate  (cost=0.00..6.08 rows=6709 width=32)
                                                             Group Key: "numeric"(price)
                                                             ->  Seq Scan on test_count_distinct_implementation  (cost=0.00..5.00 rows=9000 width=28)
Optimizer: HQO version 1.3.0
```

  从执行计划的对比中可以看出，Hologres V1.3版本减少了对Count Distinct的重复计算，可以很高效的算出结果。

- With表达式优化（Beta）

  Hologres兼容PostgreSQL，支持CTE（Common Table Expression），常用在with递归查询，其实现原理同PostgreSQL，都是基于Inlining展开的，所以当有多次使用CTE时会造成重复计算。在HologresV1.3版本中，可以通过如下GUC参数支持CTE Reuse（复用），这样CTE只需计算一次而被多次引用，用以节省计算资源，提升查询性能。若您的Hologres实例版本低于 V1.3请先提交工单升级实例。

```
set optimizer_cte_inlining=off;
```

  > **说明**
  > - 该功能当前还处于Beta阶段，默认没有开启（默认会将CTE全部Inline展开，重复计算），可手动设置GUC后开启使用。
  > - CTE Reuse开启后，依赖Shuffle阶段的Spill功能，因为下游用户消费CTE的进度是不同步的，所以数据量大时候会影响性能。

- 示例

```
create table cte_reuse_test_t
(
    a integer not null,
    b text,
    primary key (a)
);
insert into cte_reuse_test_t values(1, 'a'),(2, 'b'), (3, 'c'), (4, 'b'), (5, 'c'), (6,
''), (7, null);
set optimizer_cte_inlining=off;
explain with c as (select b, max(a) as a from cte_reuse_test_t group by b)
select a1.a,a2.a,a1.b, a2.b
from c a1, c a2
where a1.b = a2.b
order by a1.b
limit 100;
```

- 执行计划对比
  - Hologres V1.3之前版本（V1.1）的执行计划如下。

```
QUERY PLAN
Limit  (cost=0.00..10.20 rows=1 width=24)
  ->  Sort  (cost=0.00..0.00 rows=1 width=24)
        ->  Gather  (cost=0.00..10.20 rows=1 width=24)
              Sort Key: cte_reuse_test_t.b
              ->  Limit  (cost=0.00..10.20 rows=1 width=24)
                    ->  Sort  (cost=0.00..10.20 rows=1 width=24)
                          Sort Key: cte_reuse_test_t.b
                          ->  Hash Join  (cost=0.00..10.20 rows=1 width=24)
                                Hash Cond: (cte_reuse_test_t.b = cte_reuse_test_t_1.b)
                                ->  Partial HashAggregate  (cost=0.00..5.10 rows=1 width=12)
                                      Group Key: cte_reuse_test_t.b
                                      ->  Redistribution  (cost=0.00..5.10 rows=10 width=12)
                                            ->  Result  (cost=0.00..5.10 rows=10 width=12)
                                                  ->  Partial HashAggregate  (cost=0.00..5.10 rows=10 width=12)
                                                        Group Key: cte_reuse_test_t.b
                                                        ->  Exchange (Gather Exchange)  (cost=0.00..5.10 rows=32 width=12)
                                                              ->  Result  (cost=0.00..5.10 rows=32 width=12)
                                                                    ->  Decode  (cost=0.00..5.10 rows=32 width=12)
                                                                          ->  Partial HashAggregate  (cost=0.00..5.00 rows=32 width=12)
                                                                                Group Key: cte_reuse_test_t.b
                                                                                ->  Seq Scan on cte_reuse_test_t  (cost=0.00..5.00 rows=1000 width=12)
                                ->  Hash  (cost=5.10..5.10 rows=1 width=12)
                                      ->  Partial HashAggregate  (cost=0.00..5.10 rows=1 width=12)
                                            Group Key: cte_reuse_test_t_1.b
                                            ->  Redistribution  (cost=0.00..5.10 rows=10 width=12)
                                                  ->  Result  (cost=0.00..5.10 rows=10 width=12)
                                                        ->  Partial HashAggregate  (cost=0.00..5.10 rows=10 width=12)
                                                              Group Key: cte_reuse_test_t_1.b
                                                              ->  Exchange (Gather Exchange)  (cost=0.00..5.10 rows=32 width=12)
                                                                    ->  Result  (cost=0.00..5.10 rows=32 width=12)
                                                                          ->  Decode  (cost=0.00..5.10 rows=32 width=12)
                                                                                ->  Partial HashAggregate  (cost=0.00..5.00 rows=32 width=12)
                                                                                      Group Key: cte_reuse_test_t_1.b
                                                                                      ->  Seq Scan on cte_reuse_test_t cte_reuse_test_t_1  (cost=0.00..5.00 rows=1000 width=12)
Optimizer: HQO version 1.1.0
```

- Hologres V1.3版本的执行计划如下。

```
QUERY PLAN
Gather  (cost=0.00..15.10 rows=5 width=10)
  Sort Key: share0_ref3.b
  ->  Sort  (cost=0.00..15.10 rows=5 width=10)
        Sort Key: share0_ref3.b
        ->  Sequence  (cost=0.00..15.10 rows=5 width=10)
              ->  Shared Scan (share slice:id -1:0)  (cost=0.00..5.10 rows=5 width=1)
                    ->  Forward  (cost=0.00..5.10 rows=5 width=5)
                          ->  Partial HashAggregate  (cost=0.00..5.10 rows=5 width=5)
                                Group Key: cte_reuse_test_t.b
                                ->  Redistribution  (cost=0.00..5.10 rows=6 width=5)
                                      ->  Exchange (Gather Exchange)  (cost=0.00..5.10 rows=6 width=5)
                                            ->  Result  (cost=0.00..5.10 rows=6 width=5)
                                                  ->  Decode  (cost=0.00..5.10 rows=6 width=5)
                                                        ->  Partial HashAggregate  (cost=0.00..5.00 rows=6 width=5)
                                                              Group Key: cte_reuse_test_t.b
                                                              ->  Seq Scan on cte_reuse_test_t  (cost=0.00..5.00 rows=7 width=5)
              ->  Redistribution  (cost=0.00..10.00 rows=5 width=10)
                    ->  Limit  (cost=0.00..10.00 rows=5 width=10)
                          ->  Gather  (cost=0.00..10.00 rows=5 width=10)
                                Sort Key: share0_ref3.b
                                ->  Limit  (cost=0.00..10.00 rows=5 width=10)
                                      ->  Sort  (cost=0.00..10.00 rows=5 width=10)
                                            Sort Key: share0_ref3.b
                                            ->  Hash Join  (cost=0.00..10.00 rows=5 width=10)
                                                  Hash Cond: (share0_ref3.b = share0_ref2.b)
                                                  ->  Shared Scan (share slice:id -1:0)  (cost=0.00..5.00 rows=5 width=5)
                                                  ->  Hash  (cost=5.00..5.00 rows=5 width=5)
                                                        ->  Shared Scan (share slice:id -1:0)  (cost=0.00..5.00 rows=5 width=5)
Optimizer: HQO version 1.3.0
```

从执行计划的对比中可以看出Hologres V1.3之前版本会有多个AGG计算（HashAggregate），Hologres V1.3版本只需计算一次就被结果复用，提升了性能。

- 单阶段Agg优化为多阶段Agg

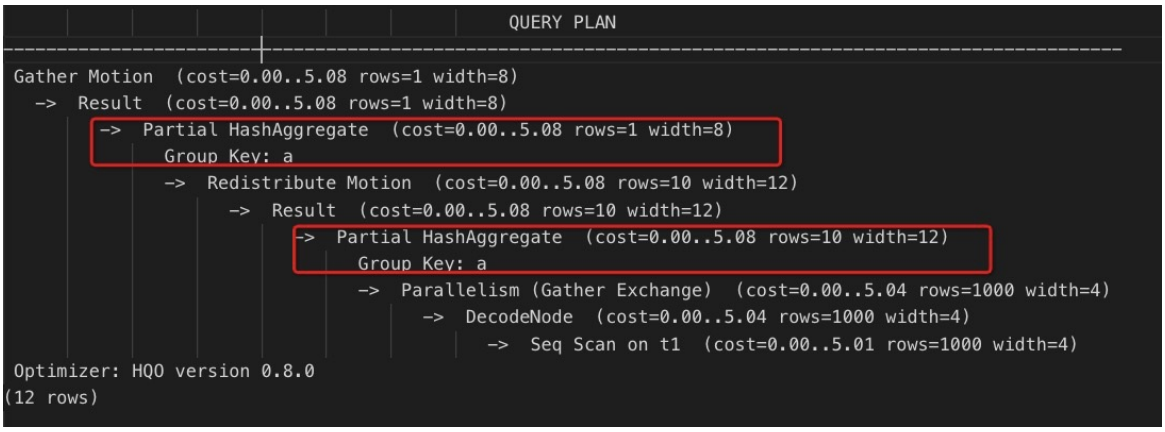
  如果Agg算子耗时过高，您可以检查是否没有做Local Shard级别的预聚合。

  通过在单个Shard内先进行本地的Agg操作，可以减少最终聚合操作的数据量，提升性能。具体如下：

  - 三阶段聚合：数据先进行文件级别的聚合计算，再聚合单个Shard内的数据，最后汇总所有Shard的结果。

```
                                                QUERY PLAN
---------------------------------------------------------------------------------------------------------
Gather Motion  (cost=0.00..5.03 rows=1 width=8)
  ->  Result  (cost=0.00..5.03 rows=1 width=8)
        ->  Partial HashAggregate  (cost=0.00..5.03 rows=1 width=8)
              Group Key: a
              ->  Redistribute Motion  (cost=0.00..5.03 rows=3 width=12)
                    ->  Result  (cost=0.00..5.03 rows=3 width=12)
                          ->  Partial HashAggregate  (cost=0.00..5.03 rows=3 width=12)
                                Group Key: a
                                ->  Parallelism (Gather Exchange)  (cost=0.00..5.03 rows=32 width=4)
                                      ->  Result  (cost=0.00..5.03 rows=32 width=4)
                                            ->  DecodeNode  (cost=0.00..5.03 rows=32 width=4)
                                                  ->  Partial HashAggregate  (cost=0.00..5.03 rows=32 width=4)
                                                        Group Key: a
                                                        ->  Seq Scan on t1  (cost=0.00..5.01 rows=1000 width=4)
Optimizer: HQO version 0.8.0
15 rows)
```

- 两阶段聚合：数据先在单个Shard内进行聚合计算，再汇总所有Shard的结果。



您可以强制Hologres进行多阶段聚合操作，语句如下。

```
set optimizer_force_multistage_agg = on;
```

建表属性优化

- 选择合适的存储类型。

  Hologres支持行存储、列存储和行列共存多种存储模式，您可以根据业务场景选择合适的存储类型，如下表所示。

类型	适用场景	缺点
行存储	○ 按主键进行高QPS的点查询场景。 ○ 一次能读取所有列，并且对UPDATE、DELETE及INSERT操作的性能较好。	大范围的查询、全表扫描及聚合等操作性能较差。
列存储	适用于多列按范围查询、单表聚合及多表连接等数据分析场景。	UPDATE和DELETE操作及无索引场景下的点查询性能慢于行存储。
行列共存	同时具备以上行列两种使用场景。	存储开销更高。

- 选择合适的数据类型。

  Hologres支持多种数据类型，您可以根据业务场景以及需求选择合适的数据类型，原则如下：

  - 尽量选用存储空间小的类型。
    - 优先使用INT类型，而不是BIGINT类型。
    - 优先使用精确确定的DECIMAL/NUMERIC类型，明确数值精度（PRECISION，SCALE），且精度尽量小，减少使用FLOAT/DOUBLE PRECISION等非精确类型，避免统计汇总中的误差。
    - GROUP BY的列不建议使用FLOAT/DOUBLE等非精确类型。
    - 优先使用TEXT，适用范围更广，当使用 `VARCHAR(N)` 和 `CHAR(N)` ，N的取值尽量小。
    - 日期类型使用TIMESTAMPTZ、DATE，避免使用TEXT。
  - 关联条件使用一致的数据类型。

    进行多表关联时，不同列尽量使用相同的数据类型。避免Hologres将不同类型的列进行隐式类型转换，造成额外的开销。

- UNION或GROUP BY等操作避免使用FLOAT/DOUBLE等非精确类型。

  UNION或GROUP BY等操作暂不支持DOUBLE PRECISION和FLOAT数据类型，需要使用DECIMAL类型。

- 选择合适的主键。

  主键（Primary Key）主要用于保证数据的唯一性，适用于主键重复的导入数据场景。您可以在导入数据时设置*option*选择去重方式，如下所示：

  - *ignore*：忽略新数据。
  - *update*：新数据覆盖旧数据。

  合理的设置主键能帮助优化器在某些场景下生成更好的执行计划。例如，查询为 `group by pk,a,b,c` 的场景。

  但是在列存储场景，主键的设置对于写入数据的性能会有较大的影响。通常，不设置主键的写入性能是设置主键的3倍。

- 选择合适的分区表。

  Hologres当前仅支持创建一级分区表。合理的设置分区会加速查询性能，不合理的设置（比如分区过多）会造成小文件过多，查询性能显著下降。

  > **说明** 对于按天增量导入的数据，建议按天建成分区表，数据单独存储，只访问当天数据。

  设置分区适用的场景如下：

  - 删除整个子表的分区，不影响其他分区数据。DROP/TRUNCATE语句的性能高于DELETE语句。
  - 对于分区列在谓词条件中的查询，可以直接通过分区列索引到对应分区，并且可以直接查询子分区，操作更为灵活。
  - 对于周期性实时导入的数据，适用于创建分区表。例如，每天都会导入新的数据，可以将日期作为分区列，每天导入数据至一个子分区。示例语句如下。

    ```
    begin;
    create table insert_partition(c1 bigint not null, c2 boolean, c3 float not null, c4 tex
    t, c5 timestamptz not null) partition by list(c4);
    call set_table_property('insert_partition', 'orientation', 'column');
    commit;
    create table insert_partition_child1 partition of insert_partition for values in('20190
    707');
    create table insert_partition_child2 partition of insert_partition for values in('20190
    708');
    create table insert_partition_child3 partition of insert_partition for values in('20190
    709');
    select * from insert_partition where c4 >= '20190708';
    select * from insert_partition_child3;
    ```

- 选择合适的索引。

  Hologres支持设置多种索引，不同索引的作用不同。您可以根据业务场景选择合适的索引，提升查询性能。索引类型如下表所示。

类型	名称	描述	使用建议	示例查询语句

类型	名称	描述	使用建议	示例查询语句
clustering_key	聚簇列	文件内聚簇索引，数据在文件内按该索引排序。 对于部分范围查询，Hologres可以直接通过聚簇索引的数据有序属性进行过滤。	将范围查询或Filter查询列作为聚簇索引列。索引过滤具备左匹配原则，建议设置不超过2列。	`select sum(a) from tb1 where a > 100 and a < 200;`
bitmap_columns	位图列	文件内位图索引，数据在文件内按该索引列建立位图。 对于等值查询，Hologres可以按照数值对每一行的数据做编码，通过位操作快速索引到对应行，时间复杂度为O(1)。	将等值查询列作为Bitmap列。	`select * from tb1 where a =100;`
segment_key（也称为event_time_column）	分段列	文件索引，数据按Append Only方式写入文件，随后文件间按该索引键合并小文件。 Segment_key标识了文件的边界范围，您可以通过Segment Key快速索引到目标文件。 Segment_key是为时间戳、日期等有序，范围类数据场景设计的，因此与数据的写入时间有强相关性。	您需要先通过Segment_key进行快速过滤，再通过Bitmap或Cluster索引进行文件内范围或等值查询。具备最左匹配原则，一般只有1列。 建议将第一个非空的时间戳字段设置为Segment_key。	`select sum(a) from tb1 where ts > '2020-01-01' and a < '2020-03-02';`

clustering_key和segment_key都需要满足传统数据库（例如MySQL）的最左前缀匹配原则，即按照Index书写的最左列排序进行索引。如果最左列为有序的场景，则按照左边第二列进行排序。示例如下。

```
call set_table_property('tmp', 'clustering_key', 'a,b,c');
select * from tmp where a > 1 ;  --可以使用Cluster索引。
select * from tmp where a > 1 and c > 2 ;   --只有a可以使用Cluster索引。
select * from tmp where a > 1 and b > 2 ;  --a,b均可以使用Cluster索引。
select * from tmp where a > 1 and b > 2 and c > 3 ; --a,b,c均可以使用Cluster索引。
select * from tmp where b > 1 and c > 2 ;   --b,c均不能使用Cluster索引。
```

Bitmap Index支持多个列的and及or查询，示例如下。

```
call set_table_property('tmp', 'bitmap_columns', 'a,b,c');
select * from tmp where a = 1 and b = 2 ;  -- 可以使用Bitmap索引。
select * from tmp where a = 1 or b = 2 ;  -- 可以使用Bitmap索引。
```

> **说明** bitmap_columns可以在创建表后添加，clustering_key和segment_key则在创建表时已经指定，后续无法再添加。

- 查看是否使用Index

  创建tmp表并指定索引字段，语句如下。

```
begin;
create table tmp(a int not null, b int not null, c int not null);
call set_table_property('tmp', 'clustering_key', 'a');
call set_table_property('tmp', 'segment_key', 'b');
call set_table_property('tmp', 'bitmap_columns', 'a,b,c');
commit;
```

  - 查看是否使用Cluster Index，语句如下。

```
explain select * from tmp where a > 1;
```

```
postgres=# explain select * from tmp where a > 1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Gather Motion  (cost=0.00..1.16 rows=1000 width=12)
   ->  Parallelism (Gather Exchange)  (cost=0.00..1.11 rows=1000 width=12)
         ->  DecodeNode  (cost=0.00..1.11 rows=1000 width=12)
               ->  Index Scan using holo_index:[1] on tmp  (cost=0.00..1.00 rows=1000 width=12)
                     Cluster Filter: (a > 1)
 Optimizer: HQO version 0.8.0
(6 rows)
```

  - 查看是否使用Bitmap Index，语句如下。

```
explain select * from tmp where c = 1;
```

```
postgres=# explain select * from tmp where c = 1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Gather Motion  (cost=0.00..1.16 rows=1000 width=12)
   ->  Parallelism (Gather Exchange)  (cost=0.00..1.11 rows=1000 width=12)
         ->  DecodeNode  (cost=0.00..1.11 rows=1000 width=12)
               ->  Index Scan using holo_index:[1] on tmp  (cost=0.00..1.00 rows=1000 width=12)
                     Bitmap Filter: (c = 1)
 Optimizer: HQO version 0.8.0
(6 rows)
```

- 查看是否使用Segment Key，语句如下。

```
explain select * from tmp where b > 1;
```

```
postgres=# explain select * from tmp where b > 1;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------
 Gather Motion  (cost=0.00..1.16 rows=1000 width=12)
   ->  Parallelism (Gather Exchange)  (cost=0.00..1.11 rows=1000 width=12)
         ->  DecodeNode  (cost=0.00..1.11 rows=1000 width=12)
               ->  Index Scan using holo_index:[1] on tmp  (cost=0.00..1.00 rows=1000 width=12)
                     Segment Filter: (b > 1)
 Optimizer: HQO version 0.8.0
(6 rows)
```

12.2. Key/Value查询场景最佳实践

Hologres是一款服务分析一体化的实时数仓，支持海量数据集（百亿以上），基于SQL提供低延时（10ms以下）、高QPS（上百万每秒）的Key/Value点查服务。本文主要指导您在点查场景的最佳实践，包括建表和查询方式等。

创建表

创建点查场景的表时，您需要注意如下几点：

- Key/Value点查的Key字段需要设置为主键（Primary Key）。
- Primary Key与Clustering Key保持一致。
- 需要将查询条件中的列设置为Distribution Key，一般默认主键为Distribution Key。
- 表的存储类型需要设置为行存。
- 建议连接Hologres实例时使用VPC网络的域名。
- 当有TEXT、VARCHAR、CHAR类型时，建议使用TEXT类型，而不是VARCHAR或者CHAR类型。

具体创建表的示例语句如下：

```
--创建一张行存表test_kv_table并将key字段设置为主键
begin;
create table test_kv_table(
  key text primary key,
  value text
);
call set_table_property('test_kv_table', 'orientation', 'row');
call set_table_property('test_kv_table', 'clustering_key', 'key');
call set_table_property('test_kv_table', 'distribution_key', 'key');
commit;
```

数据查询

当您将数据导入表之后就可以对表数据进行点查询。具体查询方式如下所示：

- 一次查询单个Key

```
select * from test_kv_table where key  = '1';
```

- 一次查询多个Key

```
select * from test_kv_table where key  in ('1', '2', '3');
```

- 使用Java查询

  以Java为例，您可以使用Prepared Statement来进行Key/Value查询，示例如下。

  > **说明**
  > - 建议使用Hologres的VPC网络的域名连接。
  > - 使用Prepared Statement进行Key/Value查询会得到更好的查询性能。
  > - Prepared Statement可以复用，不需要每次查询都新建一个Prepared Statement对象。

```
//查询取值1-100的多个key
 private static void testKV(Connection conn) throws Exception {
        String sql = "select * from test_kv_table where key = ?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            for (int i = 0; i < 100; ++i) {
                stmt.setString(1, Integer.toString(i));
                long begin = System.currentTimeMillis();
                try (ResultSet rs = stmt.executeQuery()) {
                long cost = System.currentTimeMillis() - begin;
                    while(rs.next()) {
                        System.out.println("data => " + rs.getObject(1).toString() + " "
+ rs.getObject(2).toString() + " latency => [" + cost + "]ms");
                    }
                }
            }
        }
    }
```

- 使用HoloClient查询

  HoloClient会将多个查询合并为一个SQL语句，简化开发复杂度，示例如下，建议使用Maven上发布的最新版本。

```
<dependency>
  <groupId>com.alibaba.hologres</groupId>
  <artifactId>holo-client</artifactId>
  <version>{1.2.16.5}</version>
</dependency>
// 配置参数,url格式为 jdbc:postgresql://host:port/db
HoloConfig config = new HoloConfig();
config.setJdbcUrl(url);
config.setUsername(username);
config.setPassword(password);
config.setReadThreadCount(10);//读并发，最多占用10个jdbc连接
try (HoloClient client = new HoloClient(config)) {
    //create table t0(id int not null,name0 text,address text,primary key(id))
    TableSchema schema0 = client.getTableSchema("t0");
    Get get = Get.newBuilder(schema).setPrimaryKey("id", 0).build(); // where id=0;
    client.get(get).thenAcceptAsync((record)->{
        // do something after get result
    });
    Get get1 = Get.newBuilder(schema).setPrimaryKey("id", 1).build(); // where id=1;
    client.get(get1).thenAcceptAsync((record)->{
        // do something after get result
    });
catch(HoloClientException e){
}
```

Java使用示例

如下内容为完整的Java使用示例，指导您创建一张行存表test_kv_table并将key字段设置为主键，一次查询多个Key并输出查询结果。

```
package test;
import org.postgresql.jdbc.PgConnection;
import java.sql.*;
//创建一张行存表test_kv_table并将key字段设置为主键
public class TestPointQuery {
    private static void init(Connection conn) throws Exception {
        try (Statement stmt = conn.createStatement()) {
            stmt.execute("drop table if exists test_kv_table;");
            stmt.execute("begin;");
            stmt.execute("create table if not exists test_kv_table(key text primary key, va
lue text);");
            stmt.execute("call set_table_property('test_kv_table', 'orientation', 'row');")
;
            stmt.execute("call set_table_property('test_kv_table', 'shard_count', '20');");
            stmt.execute("end;");
            stmt.execute("insert into test_kv_table select i, i from generate_series(1, 100
00)i");
        }
    }
//查询取值1-100的多个key
    private static void testKV(Connection conn) throws Exception {
        String sql = "select *from test_kv_table where key = ?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            for (int i = 0; i < 100; ++i) {
                stmt.setString(1, Integer.toString(i));
                long begin = System.currentTimeMillis();
                try (ResultSet rs = stmt.executeQuery()) {
                long cost = System.currentTimeMillis() - begin;
                    while(rs.next()) {
                        System.out.println("data => " + rs.getObject(1).toString() + " " +
rs.getObject(2).toString() + " latency => [" + cost + "]ms");
                    }
                }
            }
        }
    }
//输出查询结果
    public static void main(String[] args) throws Exception {
        Class.forName("org.postgresql.Driver").newInstance();
        String host = "";
        String db = "";
        String user = "";
        String password = "";
        String url = "jdbc:postgresql://" + host + "/" + db;
        try (PgConnection conn = (PgConnection) DriverManager.getConnection(url, user, pass
word)) {
            System.out.println("init the test_kv_table for testing");
            init(conn);
            System.out.println("run test on test_kv_table");
            testKV(conn);
        }
    }
}
```

12.3. Hologres Clustering Key最佳实践

本文将为您介绍在Hologres中，Clustering Key的概念以及实现快速查询的设置方法。

Clustering Key概述

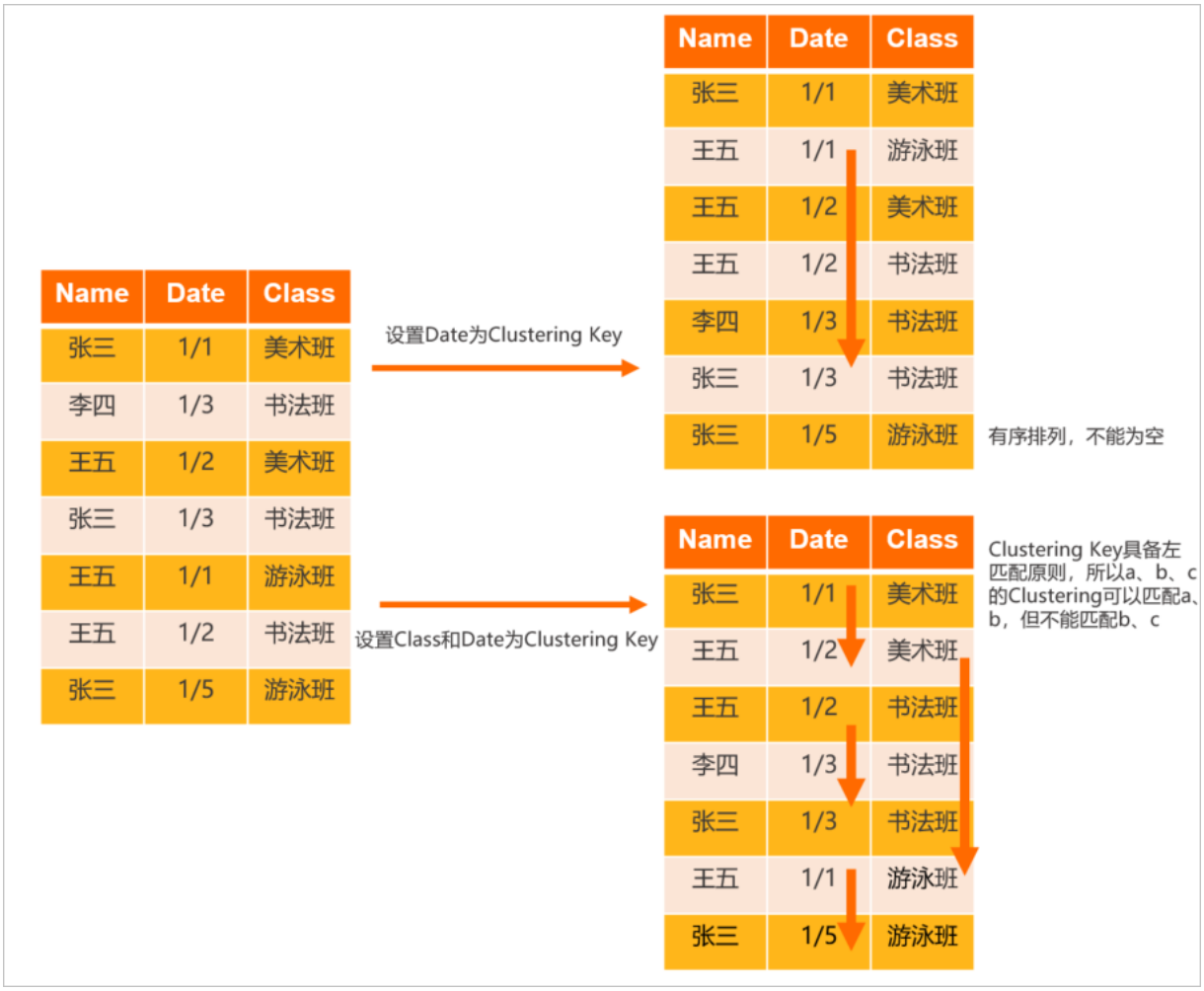
Clustering Key即文件内聚簇布局，表示文件内按照指定key的顺序来排序数据。与传统数据库（MySQL、SQLServer）中的聚簇索引不同，Hologres的排序仅做到了文件内的排序，并非是全表数据的排序，因此在Clustering Key上做排序操作仍然有一定的代价。每个表只能设置一次Clustering Key。如果修改了Clustering Key，需要重新进行数据导入。

- 对于行存表，如果设置了主键（PK），则PK默认就是Clustering Key，如果不设置PK，则默认不设置Clustering Key。
- 对于列存表，默认不设置Clustering Key。

逻辑布局

Clustering Key查询具备左匹配原则，不匹配则无法使用Clustering Key查询加速。如下场景示例将为您说明Hologres中Clustering Key的逻辑布局。

准备一张表，其字段分别包括Name、Date、Class。设置不同的字段为Clustering Key，其最终的呈现结果也不同，具体如下图所示。

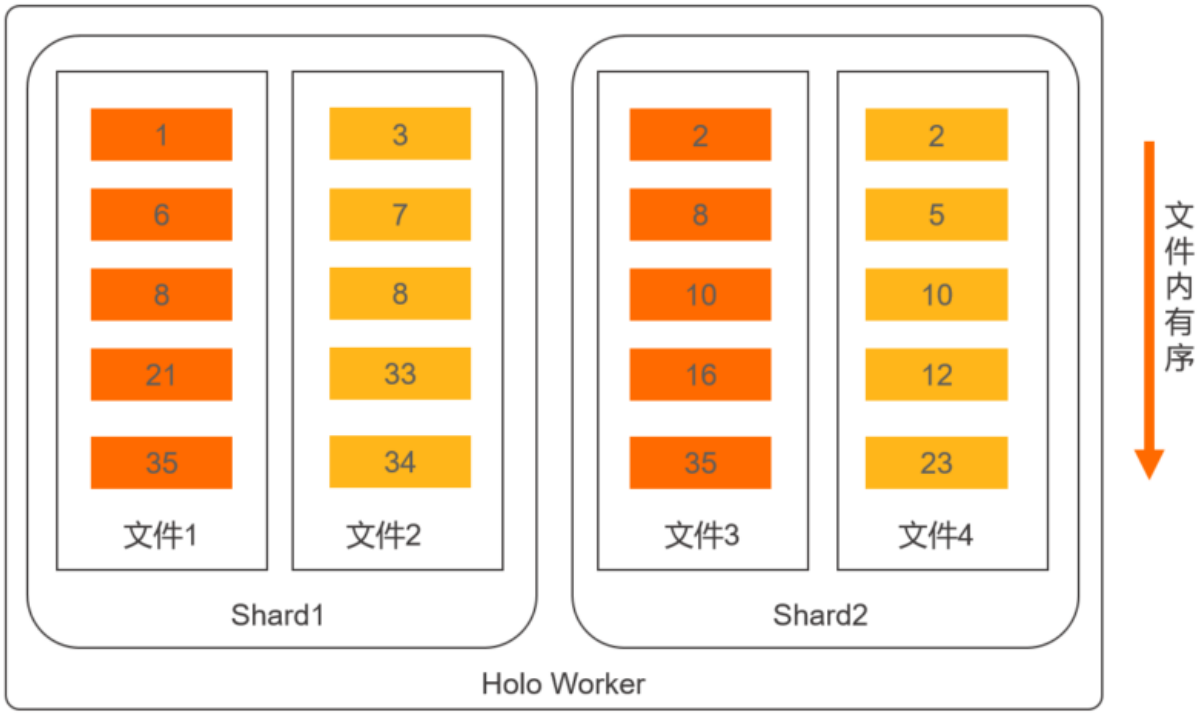


- 设置Date为Clustering Key，会将表内的数据按照Date进行排序。

- 设置Class和Date为Clustering Key，会对表先按照Class排序后再按照Date进行排序。由于Clustering Key查询具备左匹配原则，所以a、b、c的Clustering可以匹配a、b，但不能匹配b、c。

物理存储布局

Clustering Key的物理存储布局如下图所示。



Clustering Key应用

Clustering Key主要适用于点查以及范围查询，对于filter操作有比较好的性能提升，即对于 `where a = 1` 或者 `where a > 1 and a < 5` 的场景加速效果比较好。需要特别注意的是，Clustering Key和Bitmap可以同时设置，以达到最佳的点查性能。

同Bitmap相比，Hologres每个文件中按照各个Block来存储，Clustering Key可以帮助快速定位到对应的block上，而Bitmap则用于block内部数据的快速定位。因此，Bitmap和Clustering Key可以结合起来使用，以达到最佳的文件过滤效果。其异同点具体如下。

- 相同：都能对等值过滤查询有加速效果，具体示例如下所示。

```
select a,b,c from test_b where b = 10;
```

- 不同：Clustering Key每个表只能设置一个，对于range filter也有加速效果，而Bitmap每个表可以设置多个，对于range filter目前没有加速效果。因此，当用户有多种等值过滤查询场景时，可以考虑将使用次数最多的场景的key设置为Clustering Key，剩余的过滤条件设置为Bitmap。具体示例如下所示：

```
select a,b,c from test_b where a = 10;
select a,b,c from test_b where a = 20;
select a,b,c from test_b where b = 20;
select a,b,c from test_b where c = 20;
--建议设置a为clustering_key,a,b,c都设置为bitmap_column
--call set_table_property('test_b', 'clustering_key', 'a');
--call set_table_property('test_b', 'bitmap_columns', 'a,b,c');
```

使用示例

Bitmap和Clustering Key可以结合起来使用，以达到最佳的文件过滤效果。具体示例如下：

```
select a,b,c from test_a where a > 10 and a < 100;
--建议设置a为clustering_key
--call set_table_property('test_a', 'clustering_key', 'a');
select a,b,c from test_b where b = 10;
--建议同时设置b为clustering_key和bitmap_column
--call set_table_property('test_b', 'clustering_key', 'b');
--call set_table_property('test_b', 'bitmap_columns', 'b');
```

高级调优手段

Hologres从V1.3版本开始针对Clustering Key的场景使用做了较多的性能优化，实现在使用Clustering Key时有更好的性能，主要包含如下两个场景优化。如果您的版本低于1.3版本，请您提交工单升级实例。

- 针对Clustering Keys做Order By场景

  在Hologres中，文件内是按照Clustering Keys定义排序的，但在V1.3版本之前，优化器无法利用文件内的Clustering Keys有序性生成最优执行计划；同时经过Shuffle节点时也无法保障数据有序输出（多路归并），这就容易导致实际的计算量更大，耗时较久。在Hologres V1.3版本针对上面的情况进行优化，保证了生成的执行计划能够利用Clustering Keys的有序性，并能保障跨Shuffle保序，从而提高查询性能。但要注意：

  - 当表没有对Clustering Keys做过滤时，默认走的是SeqScan，而不是IndexScan（只有IndexScan才会利用Clustering Keys的有序属性）。
  - 优化器并不保障总是生成基于Clustering Keys有序的执行计划，因为利用Clustering Keys有序性是有些代价的（文件内有序但内存中需要额外排序的）。

  示例如下。

  - 表的DDL如下。

    ```
    drop table if exists test_use_sort_info_of_clustering_keys;
    begin;
    create table test_use_sort_info_of_clustering_keys
    (
              a int,
              b int,
              c text
    );
    call set_table_property('test_use_sort_info_of_clustering_keys', 'distribution_key', 'a
    ');
    call set_table_property('test_use_sort_info_of_clustering_keys', 'clustering_key', 'a,b
    ');
    commit;
    insert into test_use_sort_info_of_clustering_keys select i%500, i%100, i::text from gen
    erate_series(1, 1000) as s(i);
    analyze test_use_sort_info_of_clustering_keys;
    ```

  - 查询语句。

    ```
    explain select * from test_use_sort_info_of_clustering_keys where a > 100  order by a,
    b;
    ```

- 执行计划对比
  - V1.3之前版本（V1.1）的执行计划（执行 `explain` SQL）如下。

```
 Sort  (cost=0.00..0.00 rows=797 width=11)
   ->  Gather  (cost=0.00..2.48 rows=797 width=11)
         Sort Key: a, b
         ->  Sort  (cost=0.00..2.44 rows=797 width=11)
               Sort Key: a, b
               ->  Exchange (Gather Exchange)  (cost=0.00..1.11 rows=797 width=11)
                     ->  Decode  (cost=0.00..1.11 rows=797 width=11)
                           ->  Index Scan using holo_index:[1] on test_use_sort_info_
of_clustering_keys  (cost=0.00..1.00 rows=797 width=11)
                                 Cluster Filter: (a > 100)
```

  - V1.3版本的执行计划如下。

```
 Gather  (cost=0.00..1.15 rows=797 width=11)
   Merge Key: a, b
   ->  Exchange (Gather Exchange)  (cost=0.00..1.11 rows=797 width=11)
         Merge Key: a, b
         ->  Decode  (cost=0.00..1.11 rows=797 width=11)
               ->  Index Scan using holo_index:[1] on test_use_sort_info_of_clusterin
g_keys  (cost=0.00..1.01 rows=797 width=11)
                     Order by: a, b
                     Cluster Filter: (a > 100)
```

  V1.3版本的执行计划相较于之前版本，利用表Clustering Keys的有序性直接做归并输出，整个执行可Pipeline起来，不用再担心数据量大的时候排序慢的问题。从执行计划对比中可以看到，V1.3版本生成的是Groupagg，相比Hashagg，处理复杂度更低，性能会更好。

- 针对Clustering Keys做Join的场景（Beta）

  Hologres在V1.3版本新增了SortMergeJoin类型，以保证生成的执行计划能够利用Clustering Keys的有序性，减少计算量，从而提高性能。但需要注意：

  - 当前该功能还处于Beta版本，默认不开启，需要在Query前添加如下参数开启。

```
-- 开启merge join
set hg_experimental_enable_sort_merge_join=on;
```

  - 当表没有对Clustering Keys做过滤时，默认走的是SeqScan，而不是IndexScan（只有IndexScan才会利用Clustering Keys的有序属性）。
  - 优化器并不保障总是生成基于Clustering Keys有序的执行，因为利用Clustering Keys有序性是有些代价的（文件内有序但内存中需要额外排序）。

  示例如下。

- 表的DDL如下。

```
drop table if exists test_use_sort_info_of_clustering_keys1;
begin;
create table test_use_sort_info_of_clustering_keys1
(
  a int,
  b int,
  c text
);
call set_table_property('test_use_sort_info_of_clustering_keys1', 'distribution_key', '
a');
call set_table_property('test_use_sort_info_of_clustering_keys1', 'clustering_key', 'a,
b');
commit;
insert into test_use_sort_info_of_clustering_keys1 select i%500, i%100, i::text from ge
nerate_series(1, 10000) as s(i);
analyze test_use_sort_info_of_clustering_keys1;
drop table if exists test_use_sort_info_of_clustering_keys2;
begin;
create table test_use_sort_info_of_clustering_keys2
(
  a int,
  b int,
  c text
);
call set_table_property('test_use_sort_info_of_clustering_keys2', 'distribution_key', '
a');
call set_table_property('test_use_sort_info_of_clustering_keys2', 'clustering_key', 'a,
b');
commit;
insert into test_use_sort_info_of_clustering_keys2 select i%600, i%200, i::text from ge
nerate_series(1, 10000) as s(i);
analyze test_use_sort_info_of_clustering_keys2;
```

- 查询语句如下。

```
explain select * from test_use_sort_info_of_clustering_keys1 a join test_use_sort_info_
of_clustering_keys2 b on a.a = b.a and a.b=b.b where a.a > 100 and b.a < 300;
```

- 执行计划对比

- V1.3之前版本（V1.1）的执行计划如下。

```
 Gather  (cost=0.00..3.09 rows=4762 width=24)
   ->  Hash Join  (cost=0.00..2.67 rows=4762 width=24)
         Hash Cond: ((test_use_sort_info_of_clustering_keys1.a = test_use_sort_info_o
f_clustering_keys2.a) AND (test_use_sort_info_of_clustering_keys1.b = test_use_sort_i
nfo_of_clustering_keys2.b))
         ->  Exchange (Gather Exchange)  (cost=0.00..1.14 rows=3993 width=12)
               ->  Decode  (cost=0.00..1.14 rows=3993 width=12)
                     ->  Index Scan using holo_index:[1] on test_use_sort_info_of_clu
stering_keys1  (cost=0.00..1.01 rows=3993 width=12)
                           Cluster Filter: ((a > 100) AND (a < 300))
         ->  Hash  (cost=1.13..1.13 rows=3386 width=12)
               ->  Exchange (Gather Exchange)  (cost=0.00..1.13 rows=3386 width=12)
                     ->  Decode  (cost=0.00..1.13 rows=3386 width=12)
                           ->  Index Scan using holo_index:[1] on test_use_sort_info_
of_clustering_keys2  (cost=0.00..1.01 rows=3386 width=12)
                                 Cluster Filter: ((a > 100) AND (a < 300))
```

- V1.3版本的执行计划如下。

```
  Gather  (cost=0.00..2.88 rows=4762 width=24)
   ->  Merge Join  (cost=0.00..2.46 rows=4762 width=24)
         Merge Cond: ((test_use_sort_info_of_clustering_keys2.a = test_use_sort_info_
of_clustering_keys1.a) AND (test_use_sort_info_of_clustering_keys2.b = test_use_sort_
info_of_clustering_keys1.b))
         ->  Exchange (Gather Exchange)  (cost=0.00..1.14 rows=3386 width=12)
               Merge Key: test_use_sort_info_of_clustering_keys2.a, test_use_sort_inf
o_of_clustering_keys2.b
               ->  Decode  (cost=0.00..1.14 rows=3386 width=12)
                     ->  Index Scan using holo_index:[1] on test_use_sort_info_of_clu
stering_keys2  (cost=0.00..1.01 rows=3386 width=12)
                           Order by: test_use_sort_info_of_clustering_keys2.a, test_u
se_sort_info_of_clustering_keys2.b
                           Cluster Filter: ((a > 100) AND (a < 300))
         ->  Exchange (Gather Exchange)  (cost=0.00..1.14 rows=3993 width=12)
               Merge Key: test_use_sort_info_of_clustering_keys1.a, test_use_sort_inf
o_of_clustering_keys1.b
               ->  Decode  (cost=0.00..1.14 rows=3993 width=12)
                     ->  Index Scan using holo_index:[1] on test_use_sort_info_of_clu
stering_keys1  (cost=0.00..1.01 rows=3993 width=12)
                           Order by: test_use_sort_info_of_clustering_keys1.a, test_u
se_sort_info_of_clustering_keys1.b
                           Cluster Filter: ((a > 100) AND (a < 300))
```

V1.3版本的执行计划相较于之前版本的执行计划，利用Clustering Index的有序性，在Shard内做归并排序后直接进行Sort MergeJoin，让整个执行Pipeline起来；可规避数据量大较大时，HashJoin需将Hash Side填充至内存而导致的OOM问题。

12.4. 优化MaxCompute外部表的查询性能

本文为您介绍在Hologres中如何优化查询MaxCompute外部表数据的性能。

Hologres与MaxCompute在底层资源无缝打通，您可以通过以下方式加速查询MaxCompute的数据：

- **新建外部表直接加速查询**

  在Hologres中新建外部表，即可直接加速查询外部表数据。无需数据导入导出、无冗余存储。该方式适用于单次查询的数据量小于200 GB的表（与查询字段的大小无关）。

- **导入数据至Hologres进行加速查询**

  当需要大量分析计算外部表数据并建立与内部表的连接时，您可以在Hologres中新建内部表并导入外部表数据。根据业务需求，为内部表指定合适的Distribute Key索引属性，加快查询速度。

  导入外部表数据相比新建外部表方式查询速度更快。该方式适用于单次查询的数据量大于等于200 GB的表，以及使用复杂查询、包含索引查询、更新数据或插入数据的场景。

  导入MaxCompute外部表数据至Hologres的操作请参见使用SQL导入MaxCompute的数据至Hologres。

您还可以根据实际业务需求，通过优化查询语句、修改MaxCompute数据源表、合理配置资源和参数，优化查询外部表数据的性能。

优化查询语句

使用如下方式优化查询语句，避免查询外部表数据时扫描全表：

- 查询数据时使用 `select a from xx` 语句查询指定内容，不推荐使用 `select * from xx` 。
- 增加过滤分区的条件或减少扫描的分区数，实现减少扫描的数据量。

修改MaxCompute数据源表

通过修改MaxCompute数据源表优化查询数据的性能，方式如下：

- **转换MaxCompute表为Hash Clustering表**

  Hash Clustering表可以优化Bucket Pruning、Aggregation以及存储。

  创建MaxCompute表时，如果使用 `Clustered By` 指定了Hash Key，则MaxCompute对指定列进行Hash运算，并分散Hash值至各个Bucket中。请选择重复键值少的列作为Hash Key。

  如果没有指定Hash Key，则使用如下语句指定。

  ```
  ALTER TABLE table_name [CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC] ...])] INTO number_of_buckets BUCKETS]
  ```

  `ALTER TABLE` 语句适用于指定存量表的Hash Key。

  新增聚集属性后，新的分区数据写入MaxCompute时直接执行Hash Clustering计算。同时，您可以使用 `INSERT OVERWRITE` 语句对原有的源表数据执行Hash Clustering计算。

  > **说明**
  > - Hash Clustering表不支持 `INSERT INTO` 语句，您需要使用 `INSERT OVERWRITE` 语句添加数据。
  > - Hash Clustering表不支持通过Tunnel方式上传数据至Range Cluster表。

- **合并小文件**

  当MaxCompute中的小文件数量较多时，会降低查询表数据的速度。

  您可以在Hologres中执行如下语句，查看Query命中的文件数量：

  ```
  explain analyze <query>;
  ```

查询结果中的file_count表示MaxCompute中的文件数。如果当前小文件数量较多，影响查询速度，您可以在MaxCompute中对小文件进行合并，具体操作请参见合并小文件。

合理配置参数

查询外部表时，Hologres会设置一些默认的参数来提高读取数据的并发度，从而提高查询效率。如果您具有有特殊需求，可以按照业务场景合理配置如下参数（如下参数是经过内部调校和实验的最佳规格，一般情况下，不太建议更改）。

> **说明** `hg_foreign_table_executor_max_dop` 建议不要设置太低的值，例如设置为1。尤其是在实例既有写入又有查询的时候，不建议直接设置为1，会导致负载集中在某些Worker上引发实例OOM。

```
--调整每次query命中的分区数大小，默认512，最大为1024,不建议调整太大，会影响查询性能
set hg_foreign_table_max_partition_limit = 128;
--调整每次读取MaxCompute表batch的大小，默认8192。
set hg_experimental_query_batch_size = 4096;
--设置MaxCompute表访问切分split的数目，可以调节并发数目，默认64MB，当表很大时需要调大，避免过多的split影响性能。该参数在Hologres 1.1中生效。
set hg_foreign_table_split_size = 128;
--设置访问外表时的最大并发度，默认为实例的Core数，最大为128，不建议设置大，避免外表query（特别是数据导入场景）影响其它query，导致系统繁忙导致报错。该参数在Hologres 1.1中生效。
set hg_foreign_table_executor_max_dop = 32;
--设置访问外表时执行DML语句的最大并发度，默认值为32，针对数据导入导出场景专门优化的参数，避免导入操作占用过多系统资源，该参数在Hologres 1.1中生效。
set hg_foreign_table_executor_dml_max_dop = 16;
```

采用全新外部表查询引擎

从Hologres V0.10版本开始，将会采用全新的MaxCompute查询引擎，相比低于V0.10版本的实例，查询性能约有 30% ~ 100%的提升。

- **使用限制**
  - 该功能仅Hologres V0.10及以上版本支持，请在Hologres管理控制台的实例详情页查看当前实例版本，如果您的实例是V0.10以下版本，请您提交工单升级实例。
  - 从Hologres V1.1版本开始，查询MaxCompute数据时，默认使用全新外表查询引擎。
  - 该功能仅适用于独享实例，不适用于共享实例。
  - 当前仅对MaxCompute ORC类型的表有加速效果，暂不支持对Cfile等文件进行加速。
  - 请确保MaxCompute与Hologres的数据类型映射正确，否则加速效果不明显。
- **使用方式**
  - Hologres实例升级到V0.10版本之后，您可以使用如下开关参数开启全新外表查询引擎。

```
--session级别
set hg_experimental_enable_access_odps_orc_via_holo = on;
--DB级别
alter database <databasename> set hg_experimental_enable_access_odps_orc_via_holo = on;
```

- Hologres在V1.1及之后版本中，新外表查询引擎默认启用，可以通过以下参数配置。

```
--session级别
set hg_enable_access_odps_orc_via_holo = on;
--DB级别
alter database <databasename> set hg_enable_access_odps_orc_via_holo = on;
```

12.5. OOM常见问题排查指南

OOM（Out of Memory）描述的是Query的内存消耗超出了系统当前的供给，系统做出的一种异常提示。本文将会为您介绍Hologres中出现OOM情况的原因及对应处理方法。

产生OOM的基本原因

有的系统在内存资源不足时会采用磁盘缓存的方式进行算子降级（Spill to Disk），Hologres为了保障查询的效率，默认所有算子都采用内存资源进行计算，因此会存在内存不足导致OOM的问题。

- 内存的分配和上限

  一个Hologres实例是由多个节点组成的分布式系统，不同的实例规格对应不同的节点数，详情请参见实例规格概述。

  在Hologres中一个节点的规格是16Core 64GB，即内存上限是64GB，一个Query运行时涉及到的任意节点的内存不足，都会产生OOM异常。内存会分为三部分，三分之一分配给计算运行时分配，三分之一分配给缓存，三分之一分配给元数据及常驻执行进程。在早期版本中，计算节点（Worker Node）的内存上限是20GB，但Hologres从 V1.1.24版本开始，计算节点运行时内存取消单节点20GB的限制，采用动态调整节点内存，定期检查内存水位，如果元数据较少时，会尽量将剩余可用内存都分配给查询运行时使用，尽量保证运行时内存最大化分配，保障Query获得足够内存分配。

- 识别OOM报错

  当计算内存超出上限时（大于等于20GB），就会出现OOM的情况。常见的报错如下。

```
Total memory used by all existing queries exceeded memory limitation.
memory usage for existing queries=(2031xxxx,184yy)(2021yyyy,85yy)(1021121xxxx,6yy)(2021xx
x,18yy)(202xxxx,14yy); Used/Limit: xy1/xy2 quota/sum_quota: zz/100
```

  报错解读如下。

  - `queries=(2031xxxx,184yy)`

    指`queries=(query_id,query使用的内存)`，例如`queries=(2031xxxx,18441803528)`，代表`query_id=2031xxxx`的Query，在运行时单个节点使用了18GB的内存。异常信息里面会列出消耗内存的Top 5的Query，可以通过报错找到内存消耗最大的Query，并在慢Query日志查看与分析中查看详细的Query信息。

  - `Used/Limit: xy1/xy2`

    指**单个节点使用的计算内存/单个节点的计算内存上限**，单位为Byte。单个节点使用的计算内存是指当前时刻所有Query运行时在该节点使用的计算内存总和。例如`Used/Limit: 33288093696/33114697728`，代表所有Query在该节点运行时的内存使用了33.2GB，但是单个计算节点的弹性内存只能配33.1GB，因此出现OOM。

  - `quota/sum_quota: zz/100`

    `quota`代表资源组，其中`zz`对应资源组分配的资源。例如`quota/sum_quota: 50/100`代表设置了资源组，其分配的资源是实例总资源的50%。

- 查看内存消耗
  - 管理控制台提供整个实例的内存消耗情况，即多个节点的内存汇总值，详情请参见Hologres管控台的监控指标。
  - 慢Query日志中memory_bytes字段提供单个Query的内存消耗情况，是非精确的值，存在一定的误差，详情请参见慢Query日志查看与分析。

内存水位高

可以通过Hologres管理控制台内存指标了解实例的内存综合使用率。当内存使用率长期超过80%，可以认为属于内存水位高的情况。Hologres的内存资源采用预留模式，即使没有执行查询操作，也会有部分Meta、Index元数据和缓存加载到内存中，该类元数据用于提升计算速度，无任务运行时内存使用率可能会到达30%~50%左右，属于正常现象。内存使用率持续升高，甚至接近100%，通常会影响系统的运行，影响实例的稳定性和性能。内存水位高产生的原因、主要影响和解决方法如下所示。

- 产生原因
  - 元数据占用内存多

    表数据量增加，数据总量也随之增加，元数据占用内存多，当没有任务运行时，内存水位也会高，通常不建议一个Table Group下超过3000张表（包括分区子表），Table Group的Shard数高，也会造成更多的文件碎片和积累更多的元数据，占用元数据内存。
  - 索引设置不合理

    例如表的字段大多数是TEXT类型，但是设置了过多的Bitmap或Dictionary索引。
  - 计算内存高

    运行任务时扫描大数据量或者计算逻辑非常复杂，例如有非常多的Count Distinct函数、复杂的Join操作、多字段Group By、窗口操作等。
- 主要影响
  - 影响稳定性

    当存在元数据过大等问题时，会超额占据正常Query可用的内存空间，导致在查询过程中，可能会偶发`SERVER_INTERNAL_ERROR`、`ERPC_ERROR_CONNECTION_CLOSED`、`Total memory used by all existing queries exceeded memory limitation`等报错。
  - 影响性能

    当存在元数据过大等问题时，会超额占据正常Query可用的缓存空间，从而导致缓存命中减少，Query延迟增加。
- 解决方法
  - 元数据过多导致内存水位较高时，建议删除不再查询的数据或者表和减少不必要的分区表设计，以释放占用的内存。
  - 索引设置不合理导致内存水位较高时，建议根据业务场景具体分析，删除不涉及的Bitmap和Dictionary，以设置合理的索引。需要注意的是修改Bitmap和Dictionary时数据会进行合并，将会耗费资源，建议业务低峰期进行。
  - 计算导致内存水位较高时，建议区分写入和查询场景，进行SQL优化，详情请参见如何解决查询时OOM和如何解决导入导出时OOM。
  - 扩容，对实例的计算和存储资源进行升配，详情请参见实例列表。

如何解决查询时OOM

- 当出现查询OOM时，其原因通常有如下几类。
  - 执行计划错误：统计信息不正确、 Join Order不正确等。

- Query并发度高，且每个Query消耗的内存很大。
- Query本身复杂或者扫描的数据量很大。
- Query中包含 `union all` ，增加执行器的并行度。
- 设置了资源组，但是给资源组分配的资源较少。
- 数据倾斜或者shard pruning导致负载不均衡，个别节点的内存压力较大。

- 以上原因的具体分析以及相应的解决方法如下。
  - 检查执行计划是否合理
    - 类型1：统计信息不准确

      通过执行 `explain <SQL>` 可以查询执行计划，如下图所示 `rows=1000` 表示缺少统计信息或者统计信息不准确，会导致生成不准确的执行计划，从而使用更多的资源进行计算造成OOM。

      ```
      tt=# explain  select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
                                          QUERY PLAN
      ------------------------------------------------------------------------------------------------
       Partial Aggregate  (cost=0.00..10.11 rows=1 width=8)
         ->  Gather Motion  (cost=0.00..10.11 rows=10 width=8)
               ->  Partial Aggregate  (cost=0.00..10.11 rows=10 width=8)
                     ->  Hash Join  (cost=0.00..10.11 rows=1 width=1)
                           Hash Cond: (tmp.a = tmp1.b)
                           ->  Parallelism (Gather Exchange)  (cost=0.00..5.04 rows=1000 width=1)
                                 ->  DecodeNode  (cost=0.00..5.04 rows=1000 width=1)
                                       ->  Seq Scan on tmp  (cost=0.00..5.01 rows=1000 width=1)
                           ->  Hash  (cost=5.04..5.04 rows=1000 width=1)
                                 ->  Parallelism (Gather Exchange)  (cost=0.00..5.04 rows=1000 width=1)
                                       ->  DecodeNode  (cost=0.00..5.04 rows=1000 width=1)
                                             ->  Seq Scan on tmp1  (cost=0.00..5.01 rows=1000 width=1)
       Optimizer: HQO version 0.8.0
      (13 rows)
      ```

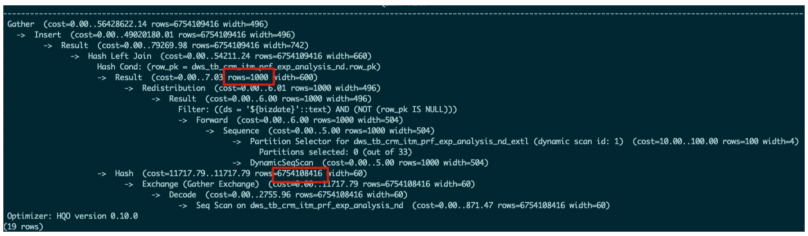
      解决方法如下。

      - 执行 `analyze <tablename>` 命令，更新表的统计信息。
      - 设置AUTO ANALYZE自动更新统计信息，详情请参见ANALYZE和AUTO ANALYZE。

- 类型2：Join Order不正确

  当两个表通过Hash算子执行连接时，合理的连接方式是数据量较小的表构建Hash表。通过执行 `explain <SQL>` 命令查看执行计划，如果数据量更大的表在下方，小表在上方时，表示使用更大的表构建Hash表，这种Join Order容易导致OOM。Join Order不正确的原因通常如下。

  - 表未及时更新统计信息，例如下图中上面的表没有更新统计信息，导致 `rows=1000` 。

  - 优化器未能生成更好的执行计划。

  解决方法如下。

  - 对参与Join的表都执行 `analyze <tablename>` 命令，及时更新统计信息，使其生成正确的Join Order。
  - 执行 `analyze tablename` 命令之后，Join Order还是不正确，可以通过修改GUC参数进行干预。如下所示设置 `optimizer_join_order=query` ，使优化器按照SQL的书写顺序确定Join Order，适用于复杂Query。

    ```
    set optimizer_join_order = query;
    select * from a join b on a.id = b.id; -- 会将b作为HashTable的build side
    ```

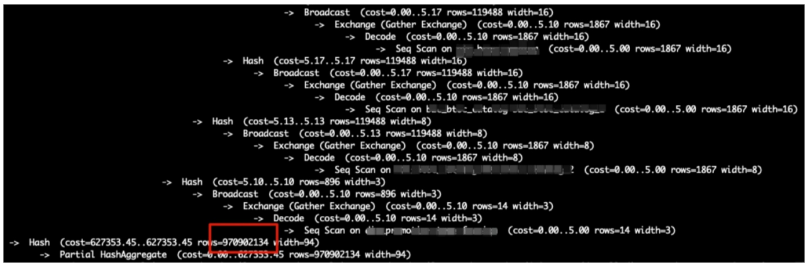
    同时也可以根据业务情况，调整Join Order策略。

参数	说明
set optimizer_join_order = <value>	优化器Join Order算法，values有如下三种。 ■ query：不进行Join Order转换，按照SQL书写的连接顺序执行，优化器开销最低。 ■ greedy：通过贪心算法进行Join Order的探索，优化器开销适中。 ■ exhaustive（默认）：通过动态规划算法进行Join Order转换，会生成最优的执行计划，但优化器开销最高。

- 类型3：Hash表预估错误

  当有Join操作时，通常是会把小表或者数据量小的子查询作为build side构建Hash表，这样既能优化性能，又能节省内存。但是有时候因为查询过于复杂，或者统计信息的问题，数据量会估错，就导致把数据量大的表或者子查询做了buildi side，这样一来，构建Hash表会消耗大量的内存，导致OOM。

  如下图所示，执行计划中 `Hash (cost=727353.45..627353.35 , rows=970902134 witdh=94)` 即为build side， `rows=970902134` 就是构建Hash表的数据量，若是实际表数据量比这个少，说明Hash表预估不准确。



  解决方法如下。

  - 查看子查询的表是否更新统计信息或者统计信息是否准确，若是不准确，请执行 `analyze <table name>` 命令。
  - 通过以下参数关闭执行引擎对Hash表的预估。

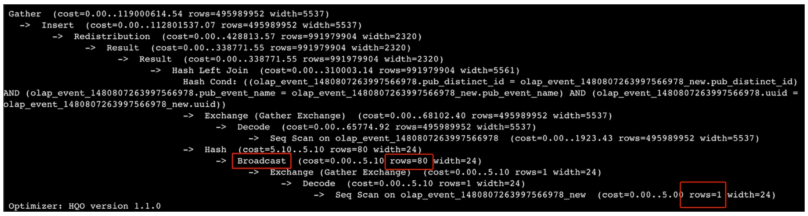
    > **说明** 该参数默认关闭，但是可能在某些调优场景被打开过，若是查看时打开的，可以进行关闭。

    ```
    set hg_experimental_enable_estimate_hash_table_size =off;
    ```

- 类型4：大表被Broadcast

  Broadcast是指将数据复制至所有Shard。仅在Shard数量与广播表的数量均较少时，Broadcast Motion的优势较大。在Join场景中，执行计划先进行Broadcast，即将build side的数据广播完再构建Hash表，这就意味着每个Shard内构建Hash表的数据都是build side的全量数据，若是Shard多或者数据量较大，则会消耗很多内存，造成OOM。

  假如表数据量是8000万行，如下图执行计划所示，预估表只有1行，参与Broadcast只有80行，与真实情况不符合，真实执行时需要8000万行数据参与Broadcast，导致消耗过多内存从而出现oom。



  解决方法如下。

  - 检查执行计划中预估行数是否正确，不正确的话，请执行 `analyze tablename` 命令更新统计信息。
  - 通过以下GUC关闭Broadcast，直接改写为redistribution算子。

    ```
    set optimizer_enable_motion_broadcast = off;
    ```

- Query并发大

  监控指标上QPS增加明显，或者OOM中报错：`HGERR_detl memory usage for existing queries=(2031xxxx,184yy)(2021yyyy,85yy)(1021121xxxx,6yy)(2021xxx,18yy)(202xxxx,14yy);` 且每个Query使用的内存较少，说明当前Query并发较大，可以通过以下方式解决。

  - 若是有写入，可以降低写入并发，详情请参见如何解决导入导出时OOM。
  - 采用多实例读写分离高可用部署（共享存储）。
  - 扩容实例计算规格。

- 复杂Query

  若是Query本身比较复杂或者扫描数据量较多，一个Query就出现OOM，可以通过以下方法解决。

  - 计算前置，将清洗好的数据写入Hologres，避免在Hologres进行大型ETL操作。
  - 增加过滤条件。
  - SQL优化：例如使用Fixed Plan、count distinct优化等，详情请参见优化内部表的性能。

- UnionALL

  如下所示，当SQL中含有大量`union all subquery`时，执行器会并行处理每个`subquery`，导致内存压力变大，从而出现OOM。

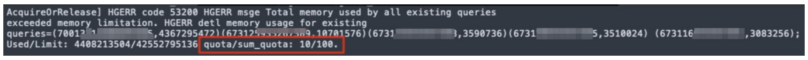
  ```
  subquery1 union all subquery2 union all subquery3 ...
  ```

  可以通过如下参数强制执行器串行，减少OOM情况，但查询速度会变慢。

  ```
  set hg_experimental_hqe_union_all_type=1;
  set hg_experimental_enable_fragment_instance_delay_open=on;
  ```

- 资源组配置不合理

  OOM时出现报错：`memory usage for existing queries=(3019xxx,37yy)(3022xxx,37yy)(3023xxx,35yy)(4015xxx,30yy)(2004xxx,2yy); Used/Limit: xy1/xy2 quota/sum_quota: zz/100` 。其中zz的取值较小，如下图所示为10，说明资源组只拥有实例10%的资源。

  解决方法：重新设置资源组配额，每个资源组都不应该小于`30%`。

- 数据倾斜或Shard Pruning

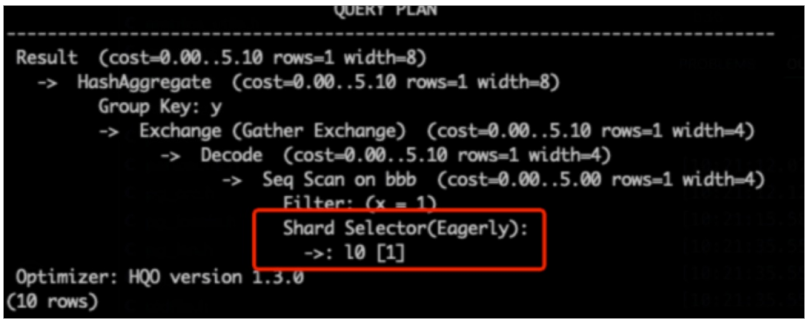
  当实例整体内存水位不高，但仍然出现OOM的情况，一般原因为数据倾斜或者Shard Pruning导致某个或者某几个节点的内存水位较高，从而出现OOM。

  > **说明** Shard Pruning是指通过查询剪枝技术，只扫描部分Shard。

  - 通过以下SQL排查数据倾斜，`hg_shard_id`是每个表的内置字段，表示数据所在的Shard。

    ```
    select count(1) from t1 group by hg_shard_id;
    ```

  - 从执行计划查看Shard Pruning，例如执行计划中shard selector为`10 (1)`，说明只选中了一个Shard数据进行查询。

  解决方法如下。

  - 设计合理的Distribution Key避免数据倾斜。
  - 若是业务有数据倾斜，需要对业务进行改造。

如何解决导入导出时OOM

导入导出OOM是指数据在Hologres表之间导入导出，也包括内部表和外部表之间导入导出，常见于MaxCompute导入到Hologres时出现OOM。

- 大宽表或者宽列且有高Scan并行度

  通常在MaxCompute导入场景会出现大宽表或者比较宽的列有比较大的Scan并行度，导致写入出现OOM。可以通过以下参数控制导入并行度减少OOM。

  - 大宽表导入（常用场景）

    > **说明** 以下参数与SQL一起执行（优先选择前两个参数，若是仍然出现OOM，可以适当调低参数取值）。

    ```
    -- 设置访问外部表时的最大并发度，默认为实例的Core数，最大为128，不建议设置大，避免外部表Query（特别是数据导入场景）影响其它Query，导致系统繁忙导致报错。该参数在Hologres V1.1及以上版本中生效。
    set hg_foreign_table_executor_max_dop = 32;
    -- 调整每次读取MaxCompute表batch的大小，默认为8192。
    set hg_experimental_query_batch_size = 4096;
    -- 设置访问外部表时执行DML语句的最大并发度，默认值为32，针对数据导入导出场景专门优化的参数，避免导入操作占用过多系统资源，该参数在Hologres V1.1及以上版本中生效。
    set hg_foreign_table_executor_dml_max_dop = 16;
    -- 设置MaxCompute表访问切分split的数目，可以调节并发数目，默认64MB，当表很大时需要调大，避免过多的split影响性能。该参数在Hologres V1.1及以上版本中生效。
    set hg_foreign_table_split_size = 128;
    ```

- 比较宽的列有比较大的scan并行度

  若是已经调整过大宽表的导入参数，但是仍然出现OOM，可以排查业务是否有比较宽的列，若有可以通过调整以下参数解决。

  ```
  -- 调整宽列的shuffle并行度，减少宽列数据量的堆积
  set hg_experimental_max_num_record_batches_in_buffer = 32;
  -- 调整每次读取MaxCompute表batch的大小，默认8192。
  set hg_experimental_query_batch_size=128;
  ```

- 外部表数据重复

  若是外部表的重复数据较多，导致导入速度慢或者出现OOM。重复数是相对而言，并没有统一标准，例如有1亿行数据，有8000万行数据都是重复的，则认为重复数据较多，请根据实际业务情况进行判断。

  解决方法：导入之前先对数据进行去重再进行导入或者分批次导入，避免一次性导入大量重复数据。

13.性能测试

13.1. 参考TPC-H测试说明

13.1.1. 测试方案介绍

本文将为您介绍如何使用TPC-H（商业智能计算测试）对OLAP查询场景和Key/Value点查场景进行性能测试。

TPC-H简介

以下文字描述引用自TPC Benchmark™ H (TPC-H)规范：

TPC-H是一个决策支持基准，由一套面向业务的临时查询和并发数据修改组成。选择的查询和填充数据库的数据具有广泛的行业相关性。该基准测试说明了决策支持系统可以检查大量数据，执行高度复杂的查询，并解答关键的业务问题。

详情请参见TPCH Specification。

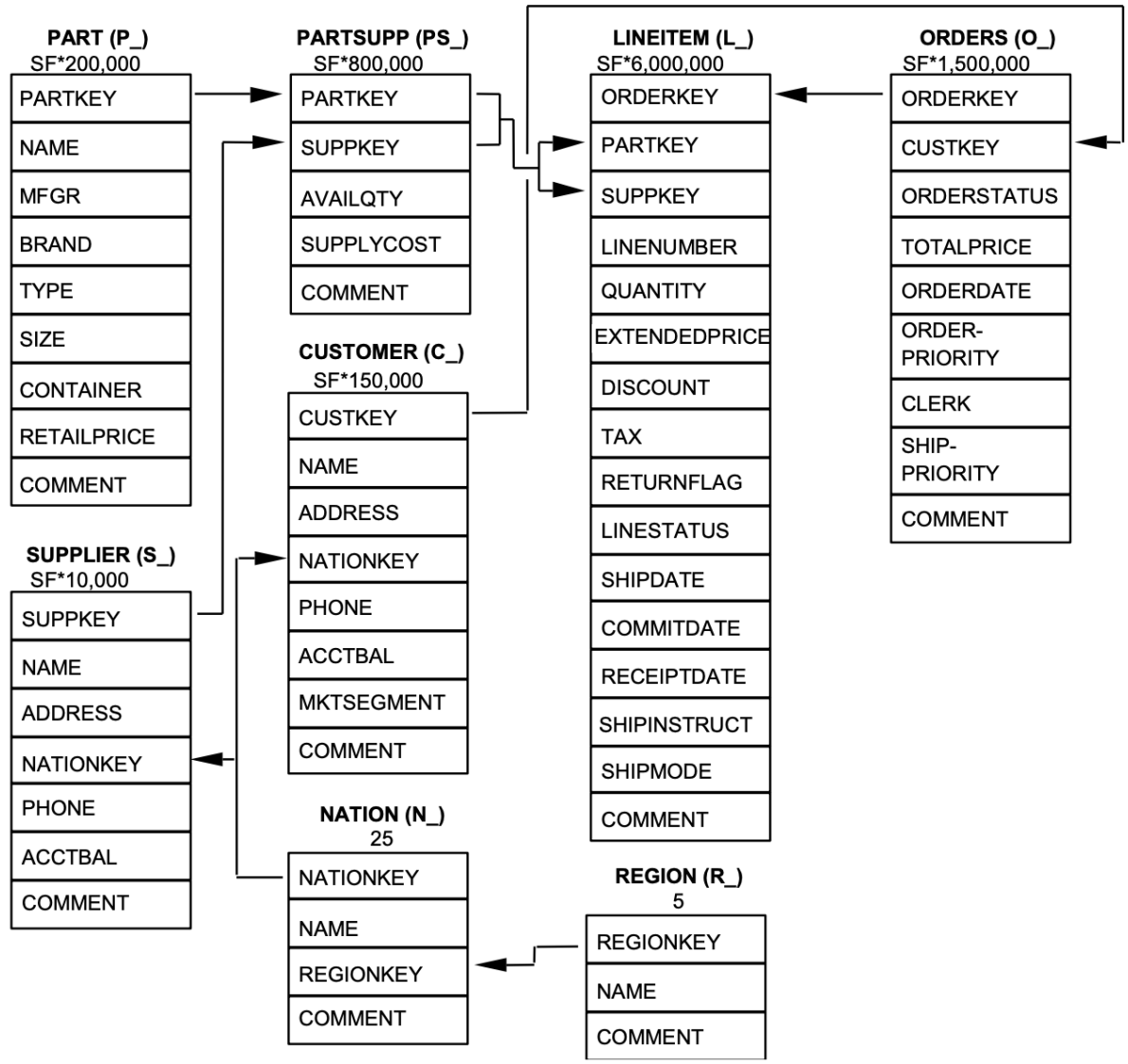
> **说明** 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

数据集简介

TPC-H（商业智能计算测试）是美国交易处理效能委员会（TPC,Transaction Processing Performance Council）组织制定的用来模拟决策支持类应用的一个测试集。目前在学术界和工业界普遍采用它来评价决策支持技术方面应用的性能。

TPC-H是根据真实的生产运行环境来建模，模拟了一套销售系统的数据仓库。其共包含8张表，数据量可设定从1 GB~3 TB不等。其基准测试共包含了22个查询，主要评价指标各个查询的响应时间，即从提交查询到结果返回所需时间。其测试结果可综合反映系统处理查询时的能力。详情请参见TPC-H基准。

该数据集包含如下 8 张表，互相间的关系如下图所示。



场景说明

本测试场景主要包含如下内容：

- OLAP查询场景测试，主要使用列存表，直接使用TPC-H测试中的22条查询语句进行测试。
- Key/Value点查场景测试，主要使用行存表，针对ORDERS使用行存表后，进行主键过滤的点查。
- 数据更新场景，主要用于测试OLAP引擎在有主键的情况下数据更新的性能。

测试数据量会直接影响测试结果，TPC-H的生成工具中使用SF（scale factor）控制生成数据量的大小，1SF对应1GB。

> **说明** 以上的数据量仅针对**原始数据的数据量**，不包括索引等空间占用，因此在准备环境时，您需要预留更多的空间。

准备工作

您需要准备OLAP查询场景和Key/Value点查场景所需的基础环境和TPC-H数据。具体内容如下：

> **说明** 为了减少可能对测试结果有影响的变量，建议每次使用新创建的实例进行测试，不要使用升降配的实例。

- **基础环境准备**

操作步骤	操作说明
1.创建ECS实例	登录阿里云，创建一个ECS实例，用于数据生成、向Hologres导入数据和客户端测试。建议规格具体如下： ○ ecs.g6.4xlarge规格。 ○ CentOS 8.4系统。 ○ ESSD数据盘，具体数据容量根据需要测试的数据量大小决定。 ○ 建议ECS与Hologres实例在相同地域，使用相同的VPC网络。 更多关于创建ECS的操作，请参见创建实例。
2.创建Hologres实例	本次测试环境使用了独享（按量付费）的实例，由于该实例仅用于测试使用，计算资源配置选择64核256GB。您可以根据实际业务需求，选择计算资源配置。 登录阿里云，进入Hologres管理控制台，单击**新增引擎实例**。 更多关于创建Hologres实例的操作，请参见购买Hologres。
3.创建测试数据库	您需要登录创建的Hologres实例，创建一个数据库。本文示例中命名数据库为**tpch_1sf**。 更多关于创建数据库的操作，请参见新建数据库。

- **生成TPC-H数据**

  您需要准备数据生成工具，即通过远程连接ECS实例，实现下载、编译并生成TPC-H数据。具体说明如下：

操作步骤	操作说明
1.准备数据生成工具	准备数据生成工具，您需要执行如下内容： i. 远程连接ECS实例。 更多关于连接方式的说明，请参见连接方式概述。 ii. 连接实例后，执行 `yum update` ，更新所有库。 iii. 执行 `yum install git` ，安装git。 iv. 执行 `yum install gcc` ，安装gcc。 v. 执行 `git clone https://github.com/gregrahn/tpch-kit.git` ，下载TPC-H数据生成代码。 vi. 执行 `cd tpch-kit/dbgen` ，进入数据生成工具代码目录。 vii. 执行 `make` ，编译数据生成工具代码。

操作步骤	操作说明

2.生成数据	编译成功后，您可以执行 `./dbgen --help` 查看代码生成工具的相关参数。您可以执行如下内容生成TPC-H数据用于下文的测试，同时您也可以查看生成的文件：
	○ 本文示例仅生成1 GB数据，您可以运行如下代码生成数据。
	`./dbgen -vf -s 1`
	如果您需要生成更多数据量的数据，可以调整SF的参数。例如，您可以使用如下代码生成1 TB数据。
	`./dbgen -vf -s 1000`
	一般情况下，32CU可以运行TPC-H SF10的数据量，256CU可以运行TPC-H SF50的数据量。
	○ 生成的数据存储在ECS中，您可以在数据生成后使用如下代码查看生成的文件。可以看到生成工具生成了8个数据文件，每个数据文件都对应一张数据集中的表。
	`ls | grep '.*.tbl'`

OLAP查询场景测试

OLAP查询场景测试，您需要创建表并导入数据。数据导入完成后，为更好的执行查询，您可以收集各张表的特征信息，然后使用pgbench工具进行查询测试。具体步骤如下：

1. **创建表**

   i. 本文内容主要基于psql进行数据导入操作，您需要先在ECS中运行如下命令安装psql。

   ```
   yum install postgresql-server -y
   ```

   ii. 安装psql后，您可以使用如下命令登录Hologres实例。更多关于参数的说明，请参见连接psql参数说明。

   ```
   PGUSER=<AccessID> PGPASSWORD=<AccessKey> psql -p <Port> -h <Endpoint> -d <Database>
   ```

   iii. 使用psql连接Hologres后，您可以使用如下建表语句创建数据库表。

   ```
   DROP TABLE IF EXISTS LINEITEM;
   BEGIN;
   CREATE TABLE LINEITEM
   (
       L_ORDERKEY      BIGINT      NOT NULL,
       L_PARTKEY       INT         NOT NULL,
       L_SUPPKEY       INT         NOT NULL,
       L_LINENUMBER    INT         NOT NULL,
       L_QUANTITY      DECIMAL(15,2) NOT NULL,
       L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
       L_DISCOUNT      DECIMAL(15,2) NOT NULL,
       L_TAX           DECIMAL(15,2) NOT NULL,
       L_RETURNFLAG    TEXT        NOT NULL,
       L_LINESTATUS    TEXT        NOT NULL,
       L_SHIPDATE      TIMESTAMPTZ NOT NULL,
   ```

```
    L_COMMITDATE    TIMESTAMPTZ NOT NULL,
    L_RECEIPTDATE   TIMESTAMPTZ NOT NULL,
    L_SHIPINSTRUCT  TEXT        NOT NULL,
    L_SHIPMODE      TEXT        NOT NULL,
    L_COMMENT       TEXT        NOT NULL,
    PRIMARY KEY (L_ORDERKEY,L_LINENUMBER)
);
CALL set_table_property('LINEITEM', 'clustering_key', 'L_SHIPDATE,L_ORDERKEY');
CALL set_table_property('LINEITEM', 'segment_key', 'L_SHIPDATE');
CALL set_table_property('LINEITEM', 'distribution_key', 'L_ORDERKEY');
CALL set_table_property('LINEITEM', 'bitmap_columns', 'L_ORDERKEY,L_PARTKEY,L_SUPPK
EY,L_LINENUMBER,L_RETURNFLAG,L_LINESTATUS,L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT');
CALL set_table_property('LINEITEM', 'dictionary_encoding_columns', 'L_RETURNFLAG,L_
LINESTATUS,L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT');
CALL set_table_property('LINEITEM', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS ORDERS;
BEGIN;
CREATE TABLE ORDERS
(
    O_ORDERKEY      BIGINT      NOT NULL PRIMARY KEY,
    O_CUSTKEY       INT         NOT NULL,
    O_ORDERSTATUS   TEXT        NOT NULL,
    O_TOTALPRICE    DECIMAL(15,2) NOT NULL,
    O_ORDERDATE     timestamptz NOT NULL,
    O_ORDERPRIORITY TEXT        NOT NULL,
    O_CLERK         TEXT        NOT NULL,
    O_SHIPPRIORITY  INT         NOT NULL,
    O_COMMENT       TEXT        NOT NULL
);
CALL set_table_property('ORDERS', 'segment_key', 'O_ORDERDATE');
CALL set_table_property('ORDERS', 'distribution_key', 'O_ORDERKEY');
CALL set_table_property('ORDERS', 'bitmap_columns', 'O_ORDERKEY,O_CUSTKEY,O_ORDERST
ATUS,O_ORDERPRIORITY,O_CLERK,O_SHIPPRIORITY,O_COMMENT');
CALL set_table_property('ORDERS', 'dictionary_encoding_columns', 'O_ORDERSTATUS,O_O
RDERPRIORITY,O_CLERK,O_COMMENT');
CALL set_table_property('ORDERS', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS PARTSUPP;
BEGIN;
CREATE TABLE PARTSUPP
(
    PS_PARTKEY    INT    NOT NULL,
    PS_SUPPKEY    INT    NOT NULL,
    PS_AVAILQTY   INT    NOT NULL,
    PS_SUPPLYCOST DECIMAL(15,2) NOT NULL,
    PS_COMMENT    TEXT   NOT NULL,
    PRIMARY KEY(PS_PARTKEY,PS_SUPPKEY)
);
CALL set_table_property('PARTSUPP', 'distribution_key', 'PS_PARTKEY');
CALL set_table_property('PARTSUPP', 'bitmap_columns', 'PS_PARTKEY,PS_SUPPKEY,PS_AVA
ILQTY,PS_COMMENT');
CALL set_table_property('PARTSUPP', 'dictionary_encoding_columns', 'PS_COMMENT');
CALL set_table_property('PARTSUPP', 'time_to_live_in_seconds', '31536000');
COMMIT;
```

```
COMMIT;
DROP TABLE IF EXISTS PART;
BEGIN;
CREATE TABLE PART
(
    P_PARTKEY     INT    NOT NULL PRIMARY KEY,
    P_NAME        TEXT   NOT NULL,
    P_MFGR        TEXT   NOT NULL,
    P_BRAND       TEXT   NOT NULL,
    P_TYPE        TEXT   NOT NULL,
    P_SIZE        INT    NOT NULL,
    P_CONTAINER   TEXT   NOT NULL,
    P_RETAILPRICE DECIMAL(15,2) NOT NULL,
    P_COMMENT     TEXT   NOT NULL
);
CALL set_table_property('PART', 'distribution_key', 'P_PARTKEY');
CALL set_table_property('PART', 'bitmap_columns', 'P_PARTKEY,P_SIZE,P_NAME,P_MFGR,P
_BRAND,P_TYPE,P_CONTAINER,P_COMMENT');
CALL set_table_property('PART', 'dictionary_encoding_columns', 'P_NAME,P_MFGR,P_BRA
ND,P_TYPE,P_CONTAINER,P_COMMENT');
CALL set_table_property('PART', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS CUSTOMER;
BEGIN;
CREATE TABLE CUSTOMER
(
    C_CUSTKEY    INT    NOT NULL PRIMARY KEY,
    C_NAME       TEXT   NOT NULL,
    C_ADDRESS    TEXT   NOT NULL,
    C_NATIONKEY  INT    NOT NULL,
    C_PHONE      TEXT   NOT NULL,
    C_ACCTBAL    DECIMAL(15,2) NOT NULL,
    C_MKTSEGMENT TEXT   NOT NULL,
    C_COMMENT    TEXT   NOT NULL
);
CALL set_table_property('CUSTOMER', 'distribution_key', 'C_CUSTKEY');
CALL set_table_property('CUSTOMER', 'bitmap_columns', 'C_CUSTKEY,C_NATIONKEY,C_NAME
,C_ADDRESS,C_PHONE,C_MKTSEGMENT,C_COMMENT');
CALL set_table_property('CUSTOMER', 'dictionary_encoding_columns', 'C_NAME,C_ADDRES
S,C_PHONE,C_MKTSEGMENT,C_COMMENT');
CALL set_table_property('CUSTOMER', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS SUPPLIER;
BEGIN;
CREATE TABLE SUPPLIER
(
    S_SUPPKEY   INT    NOT NULL PRIMARY KEY,
    S_NAME      TEXT   NOT NULL,
    S_ADDRESS   TEXT   NOT NULL,
    S_NATIONKEY INT    NOT NULL,
    S_PHONE     TEXT   NOT NULL,
    S_ACCTBAL   DECIMAL(15,2) NOT NULL,
    S_COMMENT   TEXT   NOT NULL
);
CALL set table property('SUPPLIER', 'distribution key', 'S SUPPKEY');
```

```
CALL set_table_property('SUPPLIER', 'bitmap_columns', 'S_SUPPKEY,S_NAME,S_ADDRESS,S
_NATIONKEY,S_PHONE,S_COMMENT');
CALL set_table_property('SUPPLIER', 'dictionary_encoding_columns', 'S_NAME,S_ADDRES
S,S_PHONE,S_COMMENT');
CALL set_table_property('SUPPLIER', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS NATION;
BEGIN;
CREATE TABLE NATION(
  N_NATIONKEY INT NOT NULL PRIMARY KEY,
  N_NAME text NOT NULL,
  N_REGIONKEY INT NOT NULL,
  N_COMMENT text NOT NULL
);
CALL set_table_property('NATION', 'distribution_key', 'N_NATIONKEY');
CALL set_table_property('NATION', 'bitmap_columns', 'N_NATIONKEY,N_NAME,N_REGIONKEY
,N_COMMENT');
CALL set_table_property('NATION', 'dictionary_encoding_columns', 'N_NAME,N_COMMENT'
);
CALL set_table_property('NATION', 'time_to_live_in_seconds', '31536000');
COMMIT;
DROP TABLE IF EXISTS REGION;
BEGIN;
CREATE TABLE REGION
(
    R_REGIONKEY INT  NOT NULL PRIMARY KEY,
    R_NAME      TEXT NOT NULL,
    R_COMMENT   TEXT
);
CALL set_table_property('REGION', 'distribution_key', 'R_REGIONKEY');
CALL set_table_property('REGION', 'bitmap_columns', 'R_REGIONKEY,R_NAME,R_COMMENT')
;
CALL set_table_property('REGION', 'dictionary_encoding_columns', 'R_NAME,R_COMMENT'
);
CALL set_table_property('REGION', 'time_to_live_in_seconds', '31536000');
COMMIT;
```

数据表创建完毕后，您可以在psql中执行 `tpch_1sf=# \dt` 查看是否创建成功。

```
tpch_1sf=# \dt
               List of relations
 Schema |   Name   | Type  |       Owner
--------+----------+-------+--------------------
 public | customer | table | tpch_1sf_developer
 public | lineitem | table | tpch_1sf_developer
 public | nation   | table | tpch_1sf_developer
 public | orders   | table | tpch_1sf_developer
 public | part     | table | tpch_1sf_developer
 public | partsupp | table | tpch_1sf_developer
 public | region   | table | tpch_1sf_developer
 public | supplier | table | tpch_1sf_developer
(8 rows)
```

2. **导入数据**

您可以使用如下方式导入数据：

- 本文主要使用 `COPY FROM STDIN` 的方式导入数据。更多关于 `COPY FROM STDIN` 的详细操作，请参见使用COPY命令导入或导出本地数据。此处会将tbl数据文件导入Hologres创建的表中，tbl为准备工作中生成的TPC-H数据。
- 您也可以在数据生成工具的目录中，使用如下Shell脚本导入数据。更多关于参数的说明，请参见参数说明。

```
for i in `ls *.tbl`; do
    echo $i;
    name=`echo $i| cut -d'.' -f1`;
    PGUSER=<AccessID> PGPASSWORD=<AccessKey> psql -p <Port> -h <Endpoint> -d <Databas
e> -c "COPY $name from stdin with delimiter '|' csv;" < $i;
done
```

3. **收集统计信息**

为了更好的执行查询，您可以在psql中使用如下语句，收集各张表的特征信息。

```
--清理写入文件
vacuum  region;
vacuum  nation;
vacuum  supplier;
vacuum  customer;
vacuum  part;
vacuum  partsupp;
vacuum  orders;
vacuum  lineitem;
--收集表的统计信息
analyze nation;
analyze region;
analyze lineitem;
analyze orders;
analyze customer;
analyze part;
analyze partsupp;
analyze supplier;
--针对非主键的JOIN KEY收集统计信息
analyze lineitem (l_orderkey,l_partkey,l_suppkey);
analyze orders (o_custkey);
analyze partsupp(ps_partkey,ps_suppkey);
```

4. **执行查询**

i. 为了方便统计查询信息，您需要使用pgbench工具。您可以使用如下命令安装pgbench工具。

```
yum install postgresql-contrib -y
```

为了避免因工具不兼容影响测试，请您安装版本为13及以上的pgbench工具。如果您本地已经安装pgbench工具，请确保其版本为9.6以上。您可以通过执行如下命令查看当前工具版本。

```
pgbench --version
```

ii. 为了方便查询，您可以直接单击下载TPCH 22条查询语句，并将下载的语句上传至ECS。更多关于上传数据的操作，请参见上传本地文件到ECS实例。

iii. 进入ECS，并访问上传文件的目录，使用如下Shell命令解压缩文件。

```
unzip tpch_query_updated_20210721.zip
```

iv. 您可以使用如下方式执行TPC-H的查询语句。查询语句具体内容，请参见TPC-H 22条查询语句。

- 使用pgbench工具，执行单条查询。

  TPC-H共有22条语句，例如您需要执行Q6语句，则具体语句为 `6.sql`

```
PGUSER=<AccessID> PGPASSWORD=<AccessKey> PGDATABASE=<Database> pgbench -h <Endpoi
nt> -p <Port> -c <Client_Num> -t <Query_Num> -n -f xxx.sql
```

- 使用Shell脚本，批量执行22条查询，并将结果输出到文件hologres_tpch_test.out中。

```
rm -f hologres_tpch_test.out
echo `date +"%Y-%m-%d %H:%M:%S"` begin >> ./hologres_tpch_test.out
for i in {1..22}
do
    PGUSER=<AccessID> PGPASSWORD=<AccessKey> PGDATABASE=<Database> pgbench -h <En
dpoint> -p <Port> -c <Client_Num> -t <Query_Num> -n -f ./tpch_query_updated_20210
721/q${i}.sql >> ./hologres_tpch_test.out
done
```

具体参数说明如下所示：

参数	说明
PGUSER	当前阿里云账号的AccessKey ID。 您可以单击AccessKey 管理，获取AccessKey ID。
PGPASSWORD	当前阿里云账号的AccessKey Secret。 您可以单击AccessKey 管理，获取AccessKey Secret。
-h	Hologres实例的网络地址（Endpoint）。 您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取网络地址。
-p	Hologres实例的端口（Port）。 您可以进入Hologres管理控制台的实例详情页，从实例配置页签获取实例端
-d	Hologres指定实例中的数据库名（Database）。
-c	客户端数目，即并发度（Client_Num）。 本示例取值为1，由于该测试仅测试查询性能，不测试并发，将并发度置为1即可。
-t	每个客户端需要执行的压测Query数量（Query_Num）。例如本示例取值为50。

参数	说明
-f	压力测试的SQL语句。 ■ 当您执行单条查询时，执行语句为 `xxx.sql` TPC-H共有22条语句，如您需要执行Q6语句，则具体语句为 `6.sql` ■ 当您执行批量查询时，批量脚本中该参数的取值无需修改。 TPC-H 22条查询语句具体内容，请参见TPC-H 22条查询语句。

v. 您可以通过查看hologres_tpch_test.out得到查询结果。其中，`transaction type` 表示执行具体的SQL文件。`latency average` 表示对应SQL文件的3次查询的平均时间。

```
2021-03-23 03:50:54 begin
pghost: hgpostcn-cn-oew21c935002-cn-hangzhou.hologres.aliyuncs.com pgport: 80 nclie
nts: 1 nxacts: 3 dbName: tpch_100
transaction type: ./tpch_data_tpch_query/1.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 3
number of transactions actually processed: 3/3
latency average = 76.936 ms
tps = 12.997850 (including connections establishing)
tps = 15.972757 (excluding connections establishing)
...
```

Key/Value点查场景测试

Key/Value点查场景测试，您需要创建表并导入数据。数据导入完成后，可以生成查询语句并进行查询测试。具体步骤如下：

1. **创建表**

   Key/Value点查场景测试继续使用OLAP查询场景创建的数据库，使用TPCH数据集中的ORDERS表进行测试。您在使用psql连接Hologres后即可运行如下命令创建表。psql连接Hologres操作，请参见连接psql参数说明。

   > **说明** 由于点查场景需要使用行存表，所以不能使用OLAP查询场景中使用的表，需要新创建一张表。

```
DROP TABLE IF EXISTS public.orders_row;
BEGIN;
CREATE TABLE public.orders_row(
    O_ORDERKEY       INT            NOT NULL PRIMARY KEY
    ,O_CUSTKEY       INT            NOT NULL
    ,O_ORDERSTATUS   TEXT           NOT NULL
    ,O_TOTALPRICE    DECIMAL(15,2)  NOT NULL
    ,O_ORDERDATE     TIMESTAMPTZ    NOT NULL
    ,O_ORDERPRIORITY TEXT           NOT NULL
    ,O_CLERK         TEXT           NOT NULL
    ,O_SHIPPRIORITY  INT            NOT NULL
    ,O_COMMENT       TEXT           NOT NULL
);
CALL SET_TABLE_PROPERTY('public.orders_row', 'orientation', 'row');
CALL SET_TABLE_PROPERTY('public.orders_row', 'clustering_key', 'o_orderkey');
CALL SET_TABLE_PROPERTY('public.orders_row', 'time_to_live_in_seconds', '3153600000');
CALL SET_TABLE_PROPERTY('public.orders_row', 'distribution_key', 'o_orderkey');
COMMIT;
```

2. **导入数据**

   您可以使用如下方式导入数据：

   - COPY方式导入数据
     - 本文主要使用 `COPY FROM STDIN` 的方式导入数据。更多关于 `COPY FROM STDIN` 的详细操作，请参见使用COPY命令导入或导出本地数据。此处会将**tbl**数据文件导入Hologres创建的表中，**tbl**为准备工作中生成的TPC-H数据。
     - 您也可以在数据生成工具的目录中，使用如下Shell脚本导入数据。更多关于参数的说明，请参见参数说明。

       ```
       PGUSER=<AccessID> PGPASSWORD=<AccessKey> psql -p <Port> -h <Endpoint> -d <Database>
       -c "COPY public.orders_row from stdin with delimiter '|' csv;" < orders.tbl
       ```

   - INSERT INTO方式导入数据

     在OLAP场景测试时已经导入了ORDERS表的数据，您可以运行如下SQL语句导入数据。

     ```
     INSERT INTO public.orders_row
     SELECT  *
     FROM    public.orders;
     ```

3. **执行查询**

   i. 生成查询语句

   Key/Value点查场景主要有两种查询场景，具体查询语句如下：

查询方式	查询语句	说明

查询方式	查询语句	说明
单值筛选	SELECT column_a ,column_b ,... ,column_x FROM table_x WHERE pk = value_x ;	此查询语句主要用于单值筛选，即 WHERE 的SQL语句取值唯一。
多值筛选	SELECT column_a ,column_b ,... ,column_x FROM table_x WHERE pk IN (value_a, value_b,..., value_x) ;	此查询语句主要用于多值筛选，即 WHERE 的SQL语句可以取多个值。

您可以使用如下脚本生成所需的SQL语句，该脚本会生成2条SQL语句。其中 kv_query_single.sql 表示单值筛选的SQL。 kv_query_in.sql 表示多值筛选的SQL，该脚本会随机生成一个针对10个值筛选的SQL。

```
rm -rf kv_query
mkdir kv_query
cd kv_query
echo "
\set column_values random(1,99999999)
select O_ORDERKEY,O_CUSTKEY,O_ORDERSTATUS,O_TOTALPRICE,O_ORDERDATE,O_ORDERPRIORITY,
O_CLERK,O_SHIPPRIORITY,O_COMMENT from public.orders_row WHERE o_orderkey =:column_v
alues;
" >> kv_query_single.sql
echo "
\set column_values1 random(1,99999999)
\set column_values2 random(1,99999999)
\set column_values3 random(1,99999999)
\set column_values4 random(1,99999999)
\set column_values5 random(1,99999999)
\set column_values6 random(1,99999999)
\set column_values7 random(1,99999999)
\set column_values8 random(1,99999999)
\set column_values9 random(1,99999999)
select O_ORDERKEY,O_CUSTKEY,O_ORDERSTATUS,O_TOTALPRICE,O_ORDERDATE,O_ORDERPRIORITY,
O_CLERK,O_SHIPPRIORITY,O_COMMENT from public.orders_row WHERE o_orderkey in(:column
_values1,:column_values2,:column_values3,:column_values4,:column_values5,:column_va
lues6,:column_values7,:column_values8,:column_values9);
" >> kv_query_in.sql
```

ii. 为了方便统计查询信息，您需要使用pgbench工具。您可以使用如下命令安装pgbench工具。

```
yum install postgresql-contrib -y
```

为了避免因工具不兼容影响测试，请您安装版本为13及以上的pgbench工具。如果您本地已经安装pgbench工具，请确保其版本为9.6以上。您可以通过执行如下命令查看当前工具版本。

```
pgbench --version
```

iii. 执行测试语句

- 针对单值筛选的场景，使用pgbench工具进行压测。您需要在生成SQL的目录执行如下命令，更多参数说明，请参见参数说明。

```
PGUSER=<AccessID> PGPASSWORD=<AccessKey> PGDATABASE=<Database> pgbench -h <Endpoi
nt> -p <Port> -c <Client_Num> -T <Query_Seconds> -M prepared -n -f kv_query_singl
e.sql
```

- 针对多值筛选的场景，使用pgbench工具进行压测。您需要在生成SQL的目录执行如下命令，更多参数说明，请参见参数说明。

```
PGUSER=<AccessID> PGPASSWORD=<AccessKey> PGDATABASE=<Database> pgbench -h <Endpoi
nt> -p <Port> -c <Client_Num> -T <Query_Seconds> -M prepared -n -f kv_query_in.sq
l
```

数据更新场景

该场景用于测试OLAP引擎在有主键情况下数据更新的性能，以及在主键冲突时更新整行数据。

- 生成查询

```
echo "
\set O_ORDERKEY random(1,99999999)
INSERT INTO public.orders_row(o_orderkey,o_custkey,o_orderstatus,o_totalprice,o_orderdate
,o_orderpriority,o_clerk,o_shippriority,o_comment) VALUES (:O_ORDERKEY,1,'demo',1.1,'2021
-01-01','demo','demo',1,'demo') on conflict(o_orderkey) do update set (o_orderkey,o_custk
ey,o_orderstatus,o_totalprice,o_orderdate,o_orderpriority,o_clerk,o_shippriority,o_commen
t)= ROW(excluded.*);
" > /root/insert_on_conflict.sql
```

- 插入及更新

```
PGUSER=<AccessID> PGPASSWORD=<AccessKey> PGDATABASE=<Database> pgbench -h <Endpoint> -p 8
0 -c <Client_Num> -T <Query_Seconds> -M prepared -n -f /root/insert_on_conflict.sql
```

- 示例结果

```
transaction type: Custom query
scaling factor: 1
query mode: prepared
number of clients: 249
number of threads: 1
duration: 60 s
number of transactions actually processed: 1923038
tps = 32005.850214 (including connections establishing)
tps = 36403.145722 (excluding connections establishing)
```

Flink实时写入场景

该场景用于测试实时数据写入能力。

- Hologres DDL

  该场景Hologres的表拥有10列，其中`key`列为主键，Hologres DDL如下。

```
DROP TABLE IF EXISTS flink_insert;
BEGIN ;
CREATE TABLE IF NOT EXISTS flink_insert(
  key INT PRIMARY KEY
  ,value1 TEXT
  ,value2 TEXT
  ,value3 TEXT
  ,value4 TEXT
  ,value5 TEXT
  ,value6 TEXT
  ,value7 TEXT
  ,value8 TEXT
  ,value9 TEXT
);
CALL SET_TABLE_PROPERTY('flink_insert', 'orientation', 'row');
CALL SET_TABLE_PROPERTY('flink_insert', 'clustering_key', 'key');
CALL SET_TABLE_PROPERTY('flink_insert', 'distribution_key', 'key');
COMMIT;
```

- Flink作业脚本

使用Flink全托管自带的随机数发生器向Hologres写入数据，当主键冲突时选择整行更新，单行数据量超过512 B，Flink作业脚本如下。

```
CREATE TEMPORARY TABLE flink_case_1_source (
    key INT,
    value1 VARCHAR,
    value2 VARCHAR,
    value3 VARCHAR,
    value4 VARCHAR,
    value5 VARCHAR,
    value6 VARCHAR,
    value7 VARCHAR,
    value8 VARCHAR,
    value9 VARCHAR
  )
WITH (
    'connector' = 'datagen',
     -- optional options --
    'rows-per-second' = '1000000000',
    'fields.key.min'='1',
    'fields.key.max'='2147483647',
    'fields.value1.length' = '57',
    'fields.value2.length' = '57',
    'fields.value3.length' = '57',
    'fields.value4.length' = '57',
    'fields.value5.length' = '57',
    'fields.value6.length' = '57',
    'fields.value7.length' = '57',
    'fields.value8.length' = '57',
    'fields.value9.length' = '57'
  );
-- 创建 Hologres 结果表
CREATE TEMPORARY TABLE flink_case_1_sink (
    key INT,
    value1 VARCHAR,
    value2 VARCHAR,
    value3 VARCHAR,
    value4 VARCHAR,
    value5 VARCHAR,
    value6 VARCHAR,
    value7 VARCHAR,
    value8 VARCHAR,
    value9 VARCHAR
  )
WITH (
    'connector' = 'hologres',
    'dbname'='<yourDbname>',  --Hologres的数据库名称。
    'tablename'='<yourTablename>',  --Hologres用于接收数据的表名称。
    'username'='<yourUsername>',  --当前阿里云账号的AccessKey ID。
    'password'='<yourPassword>',  --当前阿里云账号的AccessKey Secret。
    'endpoint'='<yourEndpoint>',  --当前Hologres实例VPC网络的Endpoint。
    'connectionSize' = '10',
    'jdbcWriteBatchSize' = '1024',
    'jdbcWriteBatchByteSize' = '2147483647',
```

```
    'mutatetype'='insertorreplace'
  );
-- 进行 ETL 操作并写入数据
insert into flink_case_1_sink
select key,
  value1,
  value2,
  value3,
  value4,
  value5,
  value6,
  value7,
  value8,
  value9
from
  flink_case_1_source
;
```

参数说明请参见Hologres结果表。

- 示例结果

  在Hologres的管理控制台的**监控信息**页面，即可看到RPS的数值。

TPC-H 22条查询语句

TPCH 22条查询语句如下所示，您可以单击表格中的链接进行查看。

名称	查询语句			
TPCH 22条查询语句	Q1	Q2	Q3	Q4
	Q5	Q6	Q7	Q8
	Q9	Q10	Q11	Q12
	Q13	Q14	Q15	Q16
	Q17	Q18	Q19	Q20
	Q21	Q22	-	-

- Q1

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= date '1998-12-01' - interval '120' day
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

- Q2

```
select
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from
        part,
        supplier,
        partsupp,
        nation,
        region
where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = 48
        and p_type like '%STEEL'
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'EUROPE'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp,
                        supplier,
                        nation,
                        region
                where
                        p_partkey = ps_partkey
                        and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey
                        and n_regionkey = r_regionkey
                        and r_name = 'EUROPE'
        )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey
limit 100;
```

- Q3

```
select
        l_orderkey,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = 'MACHINERY'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '1995-03-23'
        and l_shipdate > date '1995-03-23'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate
limit 10;
```

- Q4

```
select
        o_orderpriority,
        count(*) as order_count
from
        orders
where
        o_orderdate >= date '1996-07-01'
        and o_orderdate < date '1996-07-01' + interval '3' month
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
        )
group by
        o_orderpriority
order by
        o_orderpriority;
```

- Q5

```
select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'EUROPE'
        and o_orderdate >= date '1996-01-01'
        and o_orderdate < date '1996-01-01' + interval '1' year
group by
        n_name
order by
        revenue desc;
```

- Q6

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date '1996-01-01'
        and l_shipdate < date '1996-01-01' + interval '1' year
        and l_discount between 0.02 - 0.01 and 0.02 + 0.01
        and l_quantity < 24;
```

- Q7

```
set hg_experimental_enable_double_equivalent=on;
select
        supp_nation,
        cust_nation,
        l_year,
        sum(volume) as revenue
from
        (
                select
                        n1.n_name as supp_nation,
                        n2.n_name as cust_nation,
                        extract(year from l_shipdate) as l_year,
                        l_extendedprice * (1 - l_discount) as volume
                from
                        supplier,
                        lineitem,
                        orders,
                        customer,
                        nation n1,
                        nation n2
                where
                        s_suppkey = l_suppkey
                        and o_orderkey = l_orderkey
                        and c_custkey = o_custkey
                        and s_nationkey = n1.n_nationkey
                        and c_nationkey = n2.n_nationkey
                        and (
                                (n1.n_name = 'CANADA' and n2.n_name = 'BRAZIL')
                                or (n1.n_name = 'BRAZIL' and n2.n_name = 'CANADA')
                        )
                        and l_shipdate between date '1995-01-01' and date '1996-12-31'
        ) as shipping
group by
        supp_nation,
        cust_nation,
        l_year
order by
        supp_nation,
        cust_nation,
        l_year;
```

- Q8

```
set hg_experimental_enable_double_equivalent=on;
select
        o_year,
        sum(case
                when nation = 'BRAZIL' then volume
                else 0
        end) / sum(volume) as mkt_share
from
        (
                select
                        extract(year from o_orderdate) as o_year,
                        l_extendedprice * (1 - l_discount) as volume,
                        n2.n_name as nation
                from
                        part,
                        supplier,
                        lineitem,
                        orders,
                        customer,
                        nation n1,
                        nation n2,
                        region
                where
                        p_partkey = l_partkey
                        and s_suppkey = l_suppkey
                        and l_orderkey = o_orderkey
                        and o_custkey = c_custkey
                        and c_nationkey = n1.n_nationkey
                        and n1.n_regionkey = r_regionkey
                        and r_name = 'AMERICA'
                        and s_nationkey = n2.n_nationkey
                        and o_orderdate between date '1995-01-01' and date '1996-12-31'
                        and p_type = 'LARGE ANODIZED COPPER'
        ) as all_nations
group by
        o_year
order by
        o_year;
```

- Q9

```
set hg_experimental_enable_double_equivalent=on;
select
        nation,
        o_year,
        sum(amount) as sum_profit
from
        (
                select
                        n_name as nation,
                        extract(year from o_orderdate) as o_year,
                        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity a
s amount
                from
                        part,
                        supplier,
                        lineitem,
                        partsupp,
                        orders,
                        nation
                where
                        s_suppkey = l_suppkey
                        and ps_suppkey = l_suppkey
                        and ps_partkey = l_partkey
                        and p_partkey = l_partkey
                        and o_orderkey = l_orderkey
                        and s_nationkey = n_nationkey
                        and p_name like '%maroon%'
        ) as profit
group by
        nation,
        o_year
order by
        nation,
        o_year desc;
```

- Q10

```
select
        c_custkey,
        c_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,
        n_name,
        c_address,
        c_phone,
        c_comment
from
        customer,
        orders,
        lineitem,
        nation
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate >= date '1993-02-01'
        and o_orderdate < date '1993-02-01' + interval '3' month
        and l_returnflag = 'R'
        and c_nationkey = n_nationkey
group by
        c_custkey,
        c_name,
        c_acctbal,
        c_phone,
        n_name,
        c_address,
        c_comment
order by
        revenue desc
limit 20;
```

- Q11

```
select
        ps_partkey,
        sum(ps_supplycost * ps_availqty) as value
from
        partsupp,
        supplier,
        nation
where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'EGYPT'
group by
        ps_partkey having
                sum(ps_supplycost * ps_availqty) > (
                        select
                                sum(ps_supplycost * ps_availqty) * 0.0001000000
                        from
                                partsupp,
                                supplier,
                                nation
                        where
                                ps_suppkey = s_suppkey
                                and s_nationkey = n_nationkey
                                and n_name = 'EGYPT'
                )
order by
        value desc;
```

- Q12

```
select
        l_shipmode,
        sum(case
                when o_orderpriority = '1-URGENT'
                        or o_orderpriority = '2-HIGH'
                        then 1
                else 0
        end) as high_line_count,
        sum(case
                when o_orderpriority <> '1-URGENT'
                        and o_orderpriority <> '2-HIGH'
                        then 1
                else 0
        end) as low_line_count
from
        orders,
        lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode in ('FOB', 'AIR')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= date '1997-01-01'
        and l_receiptdate < date '1997-01-01' + interval '1' year
group by
        l_shipmode
order by
        l_shipmode;
```

- Q13

```
select
        c_count,
        count(*) as custdist
from
        (
                select
                        c_custkey,
                        count(o_orderkey) as c_count
                from
                        customer left outer join orders on
                                c_custkey = o_custkey
                                and o_comment not like '%special%deposits%'
                group by
                        c_custkey
        ) c_orders
group by
        c_count
order by
        custdist desc,
        c_count desc;
```

- Q14

```
select
        100.00 * sum(case
                when p_type like 'PROMO%'
                        then l_extendedprice * (1 - l_discount)
                else 0
        end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= date '1997-06-01'
        and l_shipdate < date '1997-06-01' + interval '1' month;
```

- Q15

```
with revenue0(SUPPLIER_NO, TOTAL_REVENUE)  as
    (
    select
        l_suppkey,
        sum(l_extendedprice * (1 - l_discount))
    from
        lineitem
    where
        l_shipdate >= date '1995-02-01'
        and l_shipdate < date '1995-02-01' + interval '3' month
    group by
        l_suppkey
    )
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue0
    )
order by
    s_suppkey;
```

- Q16

```
select
        p_brand,
        p_type,
        p_size,
        count(distinct ps_suppkey) as supplier_cnt
from
        partsupp,
        part
where
        p_partkey = ps_partkey
        and p_brand <> 'Brand#45'
        and p_type not like 'SMALL ANODIZED%'
        and p_size in (47, 15, 37, 30, 46, 16, 18, 6)
        and ps_suppkey not in (
                select
                        s_suppkey
                from
                        supplier
                where
                        s_comment like '%Customer%Complaints%'
        )
group by
        p_brand,
        p_type,
        p_size
order by
        supplier_cnt desc,
        p_brand,
        p_type,
        p_size;
```

- Q17

```
select
        sum(l_extendedprice) / 7.0 as avg_yearly
from
        lineitem,
        part
where
        p_partkey = l_partkey
        and p_brand = 'Brand#51'
        and p_container = 'WRAP PACK'
        and l_quantity < (
                select
                        0.2 * avg(l_quantity)
                from
                        lineitem
                where
                        l_partkey = p_partkey
        );
```

- Q18

```
select
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice,
        sum(l_quantity)
from
        customer,
        orders,
        lineitem
where
        o_orderkey in (
                select
                        l_orderkey
                from
                        lineitem
                group by
                        l_orderkey having
                                sum(l_quantity) > 312
        )
        and c_custkey = o_custkey
        and o_orderkey = l_orderkey
group by
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice
order by
        o_totalprice desc,
        o_orderdate
limit 100;
```

- Q19

```
select
        sum(l_extendedprice* (1 - l_discount)) as revenue
from
        lineitem,
        part
where
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#52'
                and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                and l_quantity >= 3 and l_quantity <= 3 + 10
                and p_size between 1 and 5
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#43'
                and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
                and l_quantity >= 12 and l_quantity <= 12 + 10
                and p_size between 1 and 10
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#52'
                and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
                and l_quantity >= 21 and l_quantity <= 21 + 10
                and p_size between 1 and 15
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        );
```

- Q20

```
select
        s_name,
        s_address
from
        supplier,
        nation
where
        s_suppkey in (
                select
                        ps_suppkey
                from
                        partsupp
                where
                        ps_partkey in (
                                select
                                        p_partkey
                                from
                                        part
                                where
                                        p_name like 'drab%'
                        )
                        and ps_availqty > (
                                select
                                        0.5 * sum(l_quantity)
                                from
                                        lineitem
                                where
                                        l_partkey = ps_partkey
                                        and l_suppkey = ps_suppkey
                                        and l_shipdate >= date '1996-01-01'
                                        and l_shipdate < date '1996-01-01' + interval '1'
year
                        )
        )
        and s_nationkey = n_nationkey
        and n_name = 'KENYA'
order by
        s_name;
```

- Q21

```
select
        s_name,
        count(*) as numwait
from
        supplier,
        lineitem l1,
        orders,
        nation
where
        s_suppkey = l1.l_suppkey
        and o_orderkey = l1.l_orderkey
        and o_orderstatus = 'F'
        and l1.l_receiptdate > l1.l_commitdate
        and exists (
                select
                        *
                from
                        lineitem l2
                where
                        l2.l_orderkey = l1.l_orderkey
                        and l2.l_suppkey <> l1.l_suppkey
        )
        and not exists (
                select
                        *
                from
                        lineitem l3
                where
                        l3.l_orderkey = l1.l_orderkey
                        and l3.l_suppkey <> l1.l_suppkey
                        and l3.l_receiptdate > l3.l_commitdate
        )
        and s_nationkey = n_nationkey
        and n_name = 'PERU'
group by
        s_name
order by
        numwait desc,
        s_name
limit 100;
```

- Q22

```
select
        cntrycode,
        count(*) as numcust,
        sum(c_acctbal) as totacctbal
from
        (
                select
                        substring(c_phone from 1 for 2) as cntrycode,
                        c_acctbal
                from
                        customer
                where
                        substring(c_phone from 1 for 2) in
                                ('24', '32', '17', '18', '12', '14', '22')
                        and c_acctbal > (
                                select
                                        avg(c_acctbal)
                                from
                                        customer
                                where
                                        c_acctbal > 0.00
                                        and substring(c_phone from 1 for 2) in
                                                ('24', '32', '17', '18', '12', '14', '22'
)
                        )
                        and not exists (
                                select
                                        *
                                from
                                        orders
                                where
                                        o_custkey = c_custkey
                        )
        ) as custsale
group by
        cntrycode
order by
        cntrycode;
```

13.1.2. 测试结果参考

本文将为您介绍测试方案介绍文档中介绍测试方法的参考结果。

背景信息

本文的参考结果是针对Hologres公有云实例进行测试的结果。

测试所用的数据量及相关集群规格说明如下：

基于100GB数据集性能参考

- 测试所用的数据量及相关集群规格说明如下：

- 测试数据量说明：

表名称	表中包含的数据行数
LINEITEM	600,037,902
ORDERS	150,000,000
PARTSUPP	80,000,000
PART	20,000,000
CUSTOMER	15,000,000
SUPPLIER	1,000,000
NATION	25
REGION	5

- 集群规则说明如下：

计算资源	存储容量	软件版本	备注
64 CU （CPU：64 Core，内存：256 GB）	100 GB	r0.10.20	使用集群默认配置，Shard数量为40。
128 CU （CPU：128 Core，内存：512 GB）	100 GB	r0.10.20	使用集群默认配置，Shard数量为80。

- 测试结果
  - 导入时间数据

    导入时间指将数据导入Hologres内表的时间，数据导入时间以秒（s）为单位。下表以64 CU计算资源为例，为您展示各类导入时间的具体数值。

    > **说明** 在使用COPY方法导入数据时，一张表对应一个数据文件，并未使用并发导入方式。

表名	数据行数	数据量	在公共网络下使用COPY方式导入	在VPC网络下使用COPY方式导入	使用MaxCompute外表导入
LINEITEM	600,037,902	73.6 GB	3,070.453	694.364	148.165
ORDERS	150,000,000	16.4 GB	691.060	172.529	37.741
PARTSUPP	80,000,000	2.3 GB	468.560	107.092	18.488
PART	20,000,000	11.3 GB	96.342	24.020	8.083
CUSTOMER	15,000,000	2.3 GB	95.190	22.937	10.363

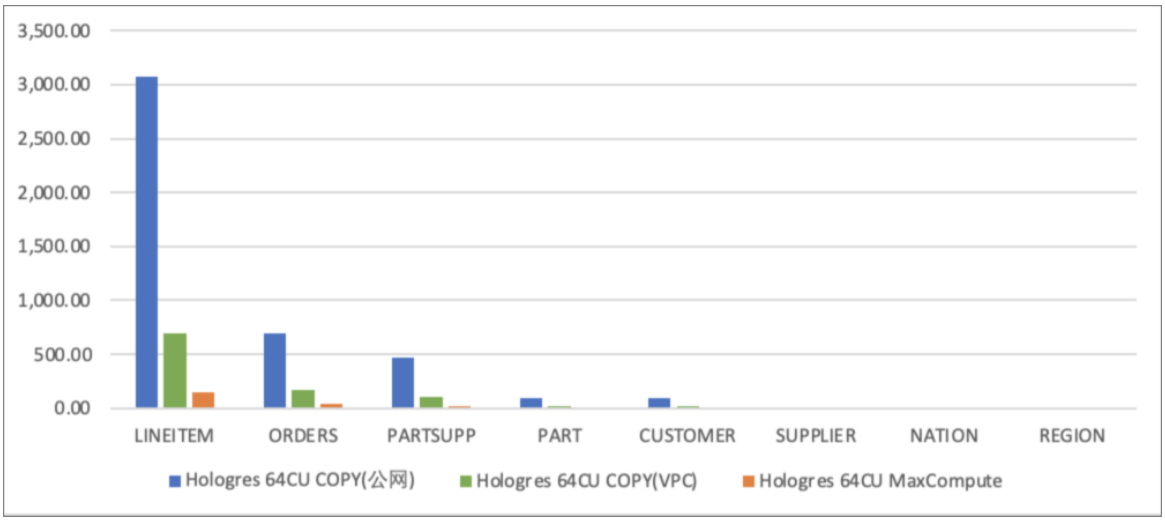
表名	数据行数	数据量	在公共网络下使用COPY方式导入	在VPC网络下使用COPY方式导入	使用MaxCompute外表导入
SUPPLIER	1,000,000	132 MB	5.057	1.803	1.503
NATION	25	2 KB	0.580	0.584	0.747
REGION	5	0.375 KB	0.168	0.153	0.430
ORDERS_ROW	150,000,000	16.4 GB	717,653	143.800	25.676
Total		122.4 GB	722,080.410	1,167.282	251.196

导入时间图示如下图所示，图中蓝色为使用COPY方式在公网条件下导入数据的时间，绿色为使用COPY方式在VPC网络条件下导入数据的时间，灰色为使用MaxCompute外表方式导入的时间。其中纵坐标数值越低，表示导入速度越快。

> **说明** 横轴：表名。纵轴：数据导入时间（s）。

结合下图内容可以看出：

- 使用COPY方式导入本地文件数据时，由于网络带宽影响，使用VPC网络的导入数据时间明显短于使用公共网络的导入数据时间。
- 使用MaxCompute导入数据时间明显短于使用COPY方式导入本地文件数据时间。



- OLAP场景

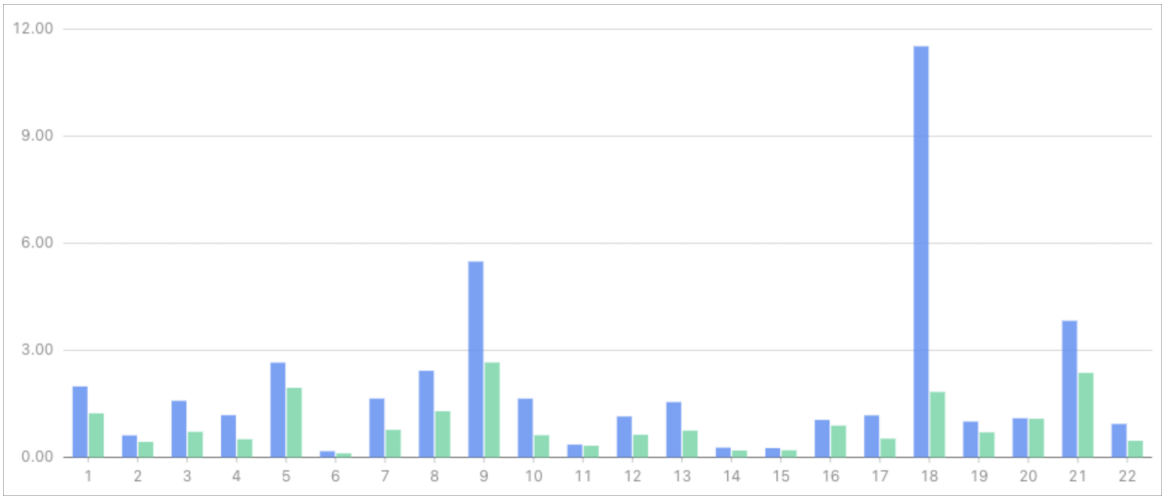
  查询执行时间以秒（s）为单位，结果均基于Hologres内表，具体数值如下所示。

TPCH Query编号	Hologres 64CU	Hologres 128CU
1	1.99	1.23
2	0.61	0.43
3	1.58	0.72

TPCH Query编号	Hologres 64CU	Hologres 128CU
4	1.18	0.51
5	2.65	1.95
6	0.17	0.11
7	1.65	0.77
8	2.43	1.29
9	5.49	2.66
10	1.65	0.62
11	0.36	0.32
12	1.15	0.63
13	1.55	0.75
14	0.27	0.19
15	0.26	0.20
16	1.05	0.89
17	1.18	0.52
18	11.52	1.83
19	1.00	0.70
20	1.10	1.08
21	3.83	2.37
22	0.94	0.46
Total	43.58	20.23

查询时间图示如下图所示，图中蓝色数据为64 CU的实例查询结果，绿色为128 CU实例的查询结果。其中纵坐标数值越低，表示TPC-H的性能越好。随着实例规模的成本增长，查询时间也在成线性下降趋势。

> **说明** 横轴：query在文档中的编号。纵轴：query执行时间（s）。



- Key/Value点查场景

  结果均基于Hologres内部表，软件版本为 V1.1.42，每类查询均连续运行5分钟，Hologres具体数值如下所示。

具体场景	样例SQL	实例规格	并发客户端数量	QPS（包含连接时间）	QPS（不包含连接时间）	平均查询延迟

1	<pre>SELECT O_ORDERKEY
	        ,O_CUSTKEY
	        ,O_ORDERSTATUS
	        ,O_TOTALPRICE
	        ,O_ORDERDATE
	
	,O_ORDERPRIORITY
	        ,O_CLERK
	        ,O_SHIPPRIORITY
	        ,O_COMMENT
	FROM
	public.orders_row
	WHERE o_orderkey =
	{parameter}
	;</pre>
		64CU	500	112,435	112,443	4.447 ms

		128CU	1,000	242,755	242,772	4.119 ms

具体场景	样例SQL	实例规格	并发客户端数量	QPS（包含连接时间）	QPS（不包含连接时间）	平均查询延迟
2	SELECT O_ORDERKEY ,O_CUSTKEY ,O_ORDERSTATUS ,O_TOTALPRICE ,O_ORDERDATE ,O_ORDERPRIORITY ,O_CLERK ,O_SHIPPRIORITY ,O_COMMENT FROM public.orders_row WHERE o_orderkey in ({parameter1}, {parameter2}, {parameter3}, {parameter4}, {parameter5}, {parameter6}, {parameter7}, {parameter8}, {parameter9});	64CU	500	27,632	27,634	18.094 ms
		128CU	1,000	90,086	90,092	11.100 ms

由测试结果可以看出：随着实例规模的增长，QPS基本呈线性增长趋势。

- 数据更新场景

  结果均基于Hologres内部表，软件版本 V1.1.42，每类查询均连续运行5分钟，Hologres具体数值如下所示。

样例SQL	实例规格	并发客户端数量	QPS（包含连接时间）	QPS（不包含连接时间）	平均查询延迟
 INSERT INTO public.orders_row(o_orderkey ,o_custkey ,o_orderstatus ,o_totalprice ,o_orderdate ,o_orderpriority ,o_clerk ,o_shippriority ,o_comment) VALUES ({parameter} ,1 ,'demo' ,1.1 ,'2021-01-01' ,'demo' ,'demo' ,1 ,'demo') on conflict(o_orderkey) do update set (o_orderkey ,o_custkey ,o_orderstatus ,o_totalprice ,o_orderdate ,o_orderpriority ,o_clerk ,o_shippriority ,o_comment)= ROW(excluded.*);	64CU	500	92,006	92,000	5.435 ms
	128CU	1,000	191,761	191,771	5.215 ms

由测试结果可以看出：随着实例规模的增长，QPS基本呈线性增长趋势。

基于1TB数据集性能参考

- 测试所用的数据量及相关集群规格说明如下：

- 测试数据量说明：

表名称	表中包含的数据行数	数据量
LINEITEM	5,999,989,709	752.27 GB
ORDERS	1,500,000,000	167.11 GB
PARTSUPP	800,000,000	114.45 GB
PART	150,000,000	22.94 GB
CUSTOMER	200,000,000	22.85 GB
SUPPLIER	10,000,000	1.33 GB
NATION	25	2.15 KB
REGION	5	0.38 KB

- 集群规则说明如下：

计算资源	存储容量	软件版本	备注
96 CU （CPU：96 Core，内存：384 GB）	1000 GB	r1.1.24	使用集群默认配置，Shard数量为60。

- 测试结果：查询时间数据

  查询执行时间以秒（s）为单位，结果均基于Hologres内表，具体数值如下所示。

TPCH Query编号	查询耗时
1	12.66
2	1.43
3	10.79
4	8.13
5	14.51
6	1.48
7	8.10
8	12.62
9	28.20
10	12.19

TPCH Query编号	查询耗时
11	3.27
12	8.96
13	12.78
14	1.10
15	1.36
16	5.67
17	8.37
18	27.78
19	9.25
20	4.11
21	25.80
22	4.51
Total	223.08

Flink实时数据写入场景性能参考

- 数据集基本信息
  - Hologres实例规格

计算资源	软件版本	备注
64 CU（CPU：64 Core，内存：256 GB）	r1.1.53	使用集群默认配置，Shard数量为40。

  - Flink全托管规格

引擎版本	vvr-4.0.12-flink-1.13
配置模式	基础模式
并发度	15
Job Manager CPU	1
Job Manager Memory	1 Gi
Task Manager CPU	1
Task Manager Memory	6 Gi

- 测试结果

  测试结果如下所示，可以看出从测试开始的2022年4月8日19:35至2022年4月8日19:50，RPS最大值为 `580,234` ，最小值为 `357,729` 。

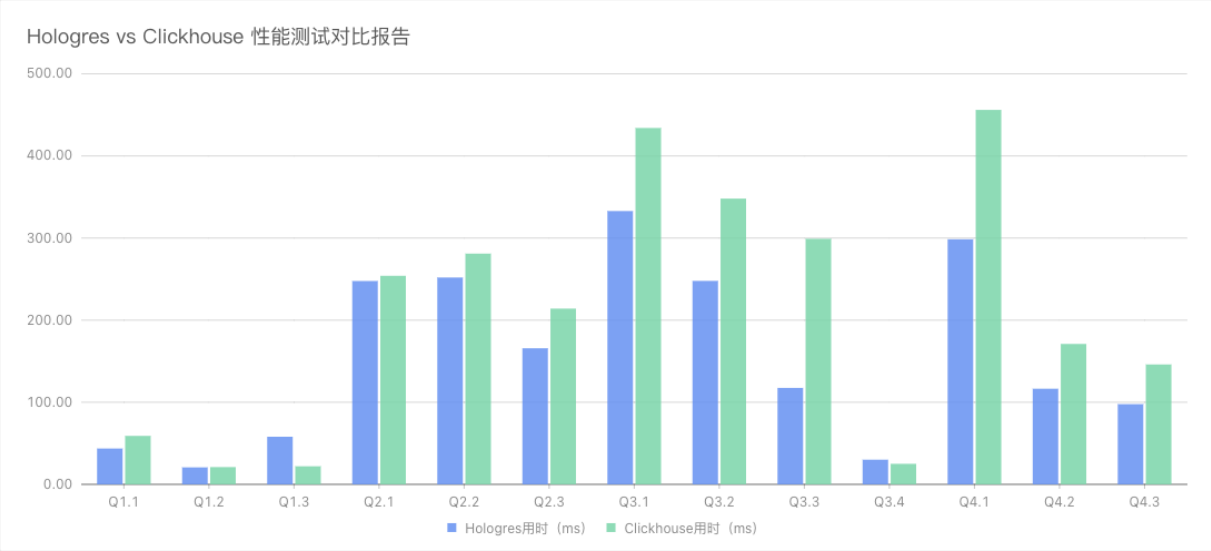
13.2. Hologres vs Clickhouse性能对比参考测试

本文将为您介绍Hologres和Clickhouse在SSB单表数据集上进行了性能对比测试的结果。

背景信息

Star schema benchmark（以下简称SSB）是学术界和工业界广泛使用的一个星型模型测试集，该测试集用于对比各种OLAP产品的基础性能指标。Clickhouse官方将SSB的星型模型打平转化成宽表，改造成了一个单表测试集，详情请参见Clickhouse官方链接。

本报告记录了Hologres和Clickhouse在SSB单表数据集上进行了性能对比测试的结果，测试结论如下。



- 在单表测试的13个查询中，11个查询Hologres比Clickhouse更快。
- 在单表测试场景中，13个查询Clickhouse总耗时是Hologres的1.35倍。

测试环境介绍

为了消除网络带宽的影响，本测试使用同一台ECS向Hologres和Clickhouse发送查询请求且使用VPC连接，其中Hologres测试关闭了result cache。具体环境信息如下。

- **Clickhouse测试环境**

配置项	详情配置信息
机器	1台阿里云ECS主机
CPU	Intel Xeon(Ice Lake) Platinum 8369B 64 vCore
内存	256 GiB
内网带宽	32 Gbps
磁盘	ESSD高效云盘200 GB PL1 单盘IOPS上限5万
操作系统	CentOS 8.4 64位
硬件费用（不含公网带宽）	9,032 元/月
公网IPv4带宽	200 Mbps
网络费用	15,725 元/月
Clickhouse版本	21.8.3.44

- **Hologres测试环境**

配置项	具体配置信息
计算资源	CPU：64 Core，内存：256 GB
存储资源	200 GB（逻辑存储）
公网带宽	大于5 Gbps
总费用	11,080 元/月
Hologres版本	0.10.33

测试数据

表名	行数	解释
lineorder	6亿	SSB商品订单表
customer	300万	SSB客户表
part	140万	SSB 零部件表
supplier	20万	SSB 供应商表
dates	2556	日期表
lineorder_flat	6亿	SSB打平后的宽表

相关SQL命令

- Clickhouse命令

  所使用的DDL与查询SQL与Clickhouse官网测试方式一致，详情请参见Clickhouse官方链接。

- Hologres命令
  - DDL

```
DROP TABLE IF EXISTS lineorder_flat;
BEGIN;
CREATE TABLE IF NOT EXISTS lineorder_flat (
  lo_orderdate     date NOT NULL ,
  lo_orderkey      int NOT NULL ,
  lo_linenumber    int NOT NULL ,
  lo_custkey       int NOT NULL ,
  lo_partkey       int NOT NULL ,
  lo_suppkey       int NOT NULL ,
  lo_orderpriority text NOT NULL ,
  lo_shippriority  int NOT NULL ,
  lo_quantity      int NOT NULL ,
  lo_extendedprice int NOT NULL ,
  lo_ordtotalprice int NOT NULL ,
  lo_discount      int NOT NULL ,
  lo_revenue       int NOT NULL ,
  lo_supplycost    int NOT NULL ,
  lo_tax           int NOT NULL ,
  lo_commitdate    date NOT NULL ,
  lo_shipmode      text NOT NULL ,
  c_name           text NOT NULL ,
  c_address text NOT NULL ,
  c_city text NOT NULL ,
  c_nation text NOT NULL ,
  c_region text NOT NULL ,
  c_phone text NOT NULL ,
  c_mktsegment text NOT NULL ,
  s_region text NOT NULL ,
  s_nation text NOT NULL ,
  s_city text NOT NULL ,
  s_name text NOT NULL ,
  s_address text NOT NULL ,
  s_phone text NOT NULL ,
  p_name text NOT NULL ,
  p_mfgr text NOT NULL ,
  p_category text NOT NULL ,
  p_brand text NOT NULL ,
  p_color text NOT NULL ,
  p_type text NOT NULL ,
  p_size int NOT NULL ,
  p_container text NOT NULL,
  PRIMARY KEY (lo_orderkey,lo_linenumber)
);
CALL set_table_property('lineorder_flat', 'distribution_key', 'lo_orderkey');
CALL set_table_property('lineorder_flat', 'segment_key', 'lo_orderdate');
CALL set_table_property('lineorder_flat', 'clustering_key', 'lo_orderdate');
CALL set_table_property('lineorder_flat', 'bitmap_columns', 'p_category,s_region,c_regi
on,c_nation,s_nation,c_city,s_city,p_mfgr,p_brand');
CALL set_table_property('lineorder_flat', 'time_to_live_in_seconds', '31536000');
COMMIT;
```

- 查询SQL

- Q1.1

```
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS REVENUE
FROM LINEORDER_FLAT
WHERE LO_ORDERDATE >= DATE '1993-01-01'
AND LO_ORDERDATE <= DATE '1993-12-31'
AND LO_DISCOUNT BETWEEN 1 AND 3
AND LO_QUANTITY < 25;
```

- Q1.2

```
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS REVENUE
FROM LINEORDER_FLAT
WHERE LO_ORDERDATE >= DATE '1994-01-01'
AND LO_ORDERDATE <= DATE '1994-01-31'
AND LO_DISCOUNT BETWEEN 4 AND 6
AND LO_QUANTITY BETWEEN 26 AND 35;
```

- Q1.3

```
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS REVENUE
FROM LINEORDER_FLAT
WHERE EXTRACT(WEEK FROM LO_ORDERDATE ) = 6
AND LO_ORDERDATE >= DATE '1994-01-01'
AND LO_ORDERDATE <= DATE '1994-12-31'
AND LO_DISCOUNT BETWEEN 5 AND 7
AND LO_QUANTITY BETWEEN 26 AND 35;
```

- Q2.1

```
SELECT SUM(LO_REVENUE),
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
P_BRAND
FROM LINEORDER_FLAT
WHERE P_CATEGORY = 'MFGR#12'
AND S_REGION = 'AMERICA'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;
```

- Q2.2

```
SELECT SUM(LO_REVENUE),
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
P_BRAND
FROM LINEORDER_FLAT
WHERE P_BRAND BETWEEN 'MFGR#2221' AND 'MFGR#2228'
AND S_REGION = 'ASIA'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;
```

- Q2.3

```
SELECT SUM(LO_REVENUE),
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
P_BRAND
FROM LINEORDER_FLAT
WHERE P_BRAND = 'MFGR#2239'
AND S_REGION = 'EUROPE'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;
```

- Q3.1

```
SELECT C_NATION,
S_NATION,
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
SUM(LO_REVENUE) AS REVENUE
FROM LINEORDER_FLAT
WHERE C_REGION = 'ASIA'
AND S_REGION = 'ASIA'
AND LO_ORDERDATE  >= DATE '1992-01-01'
AND LO_ORDERDATE  <= DATE '1997-12-31'
GROUP BY C_NATION,S_NATION,YEAR
ORDER BY  YEAR ASC,REVENUE DESC;
```

- Q3.2

```
SELECT C_CITY,
S_CITY,
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
SUM(LO_REVENUE) AS REVENUE
FROM LINEORDER_FLAT
WHERE C_NATION = 'UNITED STATES'
AND S_NATION = 'UNITED STATES'
AND LO_ORDERDATE >= DATE '1992-01-01'
AND LO_ORDERDATE <= DATE '1997-12-31'
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, REVENUE DESC;
```

- Q3.3

```
SELECT C_CITY,
S_CITY,
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
SUM(LO_REVENUE) AS REVENUE
FROM LINEORDER_FLAT
WHERE C_CITY IN ( 'UNITED KI1' ,'UNITED KI5')
AND S_CITY IN ( 'UNITED KI1' ,'UNITED KI5')
AND LO_ORDERDATE >= DATE '1992-01-01'
AND LO_ORDERDATE <= DATE '1997-12-31'
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, REVENUE DESC;
```

- Q3.4

```
SELECT C_CITY,
S_CITY,
EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
SUM(LO_REVENUE) AS REVENUE
FROM LINEORDER_FLAT
WHERE C_CITY IN ('UNITED KI1', 'UNITED KI5')
AND S_CITY IN ( 'UNITED KI1',  'UNITED KI5')
AND LO_ORDERDATE >= DATE '1997-12-01'
AND LO_ORDERDATE <= DATE '1997-12-31'
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, REVENUE DESC;
```

- Q4.1

```
SELECT EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
C_NATION,
SUM(LO_REVENUE - LO_SUPPLYCOST) AS PROFIT
FROM LINEORDER_FLAT
WHERE C_REGION = 'AMERICA'
AND S_REGION = 'AMERICA'
AND P_MFGR IN ( 'MFGR#1' , 'MFGR#2')
GROUP BY YEAR, C_NATION
ORDER BY YEAR ASC, C_NATION ASC;
```

- Q4.2

```
SELECT EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
S_NATION,
P_CATEGORY,
SUM(LO_REVENUE - LO_SUPPLYCOST) AS PROFIT
FROM LINEORDER_FLAT
WHERE C_REGION = 'AMERICA'
AND S_REGION = 'AMERICA'
AND LO_ORDERDATE >= DATE '1997-01-01'
AND LO_ORDERDATE <= DATE '1998-12-31'
AND  P_MFGR IN ( 'MFGR#1' , 'MFGR#2')
GROUP BY YEAR, S_NATION, P_CATEGORY
ORDER BY  YEAR ASC, S_NATION ASC, P_CATEGORY ASC;
```

- Q4.3

```
SELECT EXTRACT(YEAR FROM LO_ORDERDATE) AS YEAR,
S_CITY,
P_BRAND,
SUM(LO_REVENUE - LO_SUPPLYCOST) AS PROFIT
FROM LINEORDER_FLAT
WHERE S_NATION = 'UNITED STATES'
AND LO_ORDERDATE >= DATE '1997-01-01'
AND LO_ORDERDATE <= DATE '1998-12-31'
AND P_CATEGORY = 'MFGR#14'
GROUP BY  YEAR,  S_CITY, P_BRAND
ORDER BY YEAR ASC,  S_CITY ASC,  P_BRAND ASC;
```

测试结果

SQL	Hologres用时（ms）	Clickhouse用时（ms）	Clickhouse/Hologres用时
Q1.1	43.66	59.00	1.35
Q1.2	20.68	21.00	1.02
Q1.3	57.98	22.00	0.38
Q2.1	247.63	254.00	1.03
Q2.2	251.90	281.00	1.12
Q2.3	165.73	214.00	1.29
Q3.1	332.84	434.00	1.30
Q3.2	247.79	348.00	1.40
Q3.3	117.46	299.00	2.55
Q3.4	30.05	25.00	0.83
Q4.1	298.48	456.00	1.53
Q4.2	116.47	171.00	1.47
Q4.3	97.68	146.00	1.49
合计	2,028.35	2,730.00	1.35

14.常见问题

14.1. 约束和限制

本文为您介绍使用交互式分析Hologres服务的约束与限制。

MaxCompute直接查询的约束限制

- 查询MaxCompute分区表时，扫描分区数的最大值为512。
- 每个查询中最大的底层数据扫描量为200GB，与表以及字段的数量无关。
- Hologres中不支持使用MaxCompute的内建函数。

UDF约束限制

- Hologres中不支持使用MaxCompute创建的UDF。
- Hologres中不支持创建和使用UDF，但是支持使用PostgreSQL的内建函数。

数据类型约束

Hologres兼容PostgreSQL生态，支持的数据类型与PostgreSQL兼容，但仅为PostgreSQL的一个子集。Hologres目前支持的数据类型请参见数据类型汇总。

HoloStudio使用限制

- 一个实例可以创建多个数据库，在绑定HoloStudio工作空间时，一个数据库可以重复绑定在多个相同地域的工作空间。
- 在Holostudio中，**前往DataWorks调度**默认使用**公共资源组**，如果您需要使用**独享资源组**，请联系对应的技术支持或者提交工单开通配置。

购买和付费约束

- 一个阿里云主账号每天最多只能买3个实例。
- 购买实例分为**包年包月**和**按量付费**两种方式，当前仅支持将**按量付费**转为**包年包月**，不支持将**包年包月**转换为**按量付费**。
- 如果您使用的是**按量付费**方式，并在Hologres管控台执行手动停机操作，计算资源将会停止收费，但存储资源仍然会继续收费，直到实例删除停止收费。
- 如果您购买Hologres实例使用的**包年包月**方式，如果使用时存储资源超过购买额度，则超出部分将会按照**按量付费**的方式收费。

14.2. Blink和Flink常见问题及诊断

本文为您介绍使用Hologres过程中关于Blink和Flink的常见问题。

基本概念

- Hologres性能

- 写入性能
  - 列存表： InsertOrIgnore > InsertOrReplace > InsertOrUpdate
  - 行存表： InsertOrReplcae = InsertOrUpdate > InsertOrIgnore

参数	说明
InsertOrIgnore	结果表有主键，实时写入时如果主键重复，丢弃后到的数据。
InsertOrReplace	结果表有主键，实时写入时如果主键重复，按照主键更新，如果写入的一行数据不包含所有列，缺失的列的数据补Null。
InsertOrUpdate	结果表有主键，实时写入时如果主键重复，按照主键更新，如果写入的一行数据不包含所有列，缺失的列不更新。

- 点查性能

  行存 = 行列混存 > 列存。

- Blink、Flink（VVP）、开源Flink支持情况

产品形态	数据存储类型					描述
	源表	结果表	维表	Binlog	Hologres Catalog	
Flink全托管	支持行存储及列存储。	支持行存储及列存储。	建议使用行存储。	支持	支持	无
Blink独享	支持行存储及列存储。	支持行存储及列存储。	建议使用行存储。	Hologres V0.8版本只支持行存储，V0.9及以上版本支持行存储及列存储。建议使用行存储。	不支持	已开始逐步下线，推荐使用阿里云Flink全托管。
开源Flink1.10	支持行存储及列存储。	支持行存储及列存储。	无	不支持	不支持	无
开源Flink1.11	支持行存储及列存储。	支持行存储及列存储。	建议使用行存储。	不支持	不支持	从开源Flink1.11版本开始，Hologres代码已开源。详细内容请参见GitHub。
开源Flink1.12	支持行存储及列存储。	支持行存储及列存储。	建议使用行存储。	不支持	不支持	

- Blink、Flink 映射Hologres的SQL示例如下。

```
create table holo_source(
'hg_binlog_lsn' BIGINT HEADER,
'hg_binlog_event_type' BIGINT HEADER,
'hg_binlog_timestamp_us' BIGINT HEADER,
A int,
B int,
C timestamp )
with (
type = 'hologres',
'endpoint' = 'xxx.hologres.aliyuncs.com:80',    --Hologres实例的Endpoint。
'userName' = '',                                --当前阿里云账号的AccessKey ID。
'password' = '',                                --当前阿里云账号的AccessKey Secret。
'dbName' = 'binlog',                            --Hologres实例的数据库名称。
'tableName' ='test'                             --Hologres实例的表名称。
'binlog' = 'true',
);
```

Blink、VVP、Flink SQL，都是在Flink侧声明一张表，然后根据参数映射至Hologres的一张具体的物理表，所以不支持映射至外部表。

实时写入慢问题排查流程

1. 确认写入相关配置

   需要确认以下配置信息。

   - 目标表的存储格式，包括行存表、列存表和行列共存表。
   - Insert模式，包括InsertOrIgnore、InsertOrUpdate和InsertOrReplace。
   - 目标表的Table Group及Shard Count。

2. 查看监控指标的实时写入延迟

   如果平均写入延迟偏高，在百毫秒甚至秒级别，通常便是后端达到了写入瓶颈，这时候可能会存在如下问题。

   - 使用了列存表的InsertOrUpdate，即局部更新，且流量较高，这种情况下会导致实例的CPU负载和写入延迟偏高。

     解决方法：建议更换表的类型，使用行存表，如果您的实例是V1.1及以上版本可以选择行列混存表。

   - 云监控查看实例的CPU负载，如果CPU水位接近100%，但没有列存表的局部更新，那么通常情况下是由于高QPS的查询，或者本身写入量较高导致的。

     解决方法：扩容Hologres实例。

   - 确认是否有不断的 `Insert into select from` 命令，触发了该表的BulkLoad写入，当前BulkLoad写入会阻塞实时写入。

     解决方法：将BulkLoad写入转换成实时写入，或者错峰执行。

3. 确认是否有数据倾斜

   使用如下SQL命令查看是否有数据倾斜。

   ```
   select count(1) from t1 group by hg_shard_id;
   ```

   解决方法：修改Distribution Key。

4. 确认后端是否有压力

   如果以上步骤排查完没有问题，写入性能突然下降，一般情况是后端集群压力比较大，存在瓶颈。请联

系技术支持人员确认情况，详情请参见如何获取更多的在线支持？。

5. 查看Blink/Flink侧的反压情况

   上述步骤排查完后，发现Hologres侧没有明显的异常，通常情况下是客户端慢了，也就是Blink/Flink侧本身就慢了，这时候确认是否是Sink节点反压了。如果作业只有一个节点，就无法看出是否反压了，这时候可以将Sink节点单独拆开再观察。具体请联系Flink技术支持。

写入数据有问题排查流程

这种情况通常是由于数据乱序引起的，比如相同主键的数据分部在不同的Flink Task上，写入的时候无法保证顺序。需要确认Flink SQL的逻辑，最后写出到Hologres的时候，是否按照Hologres表的主键进行Shuffle了。

维表查询问题排查流程

- 维表Join和双流Join

  对于读Hologres的场景，需要首先确认用户是否使用对了维表Join，是否错将双流Join当成维表Join来使用了。以下是Hologres作为维表的使用示例，如果少了 `proctime AS PROCTIME()` 和 `hologres_dim FOR SYSTEM_TIME AS` 两处关键字，则会变成双流Join。

```
CREATE TEMPORARY TABLE datagen_source (
   a INT,
   b BIGINT,
   c STRING,
   proctime AS PROCTIME()
) with (
   'connector' = 'datagen'
);
CREATE TEMPORARY TABLE hologres_dim (
   a INT,
   b VARCHAR,
   c VARCHAR
) with (
   'connector' = 'hologres',
   ...
);
CREATE TEMPORARY TABLE blackhole_sink (
   a INT,
   b STRING
) with (
   'connector' = 'blackhole'
);
insert into blackhole_sink select T.a,H.b
FROM datagen_source AS T JOIN hologres_dim FOR SYSTEM_TIME AS OF T.proctime AS H ON T.a =
H.a;
```

- 维表查询

  i. 确认维表存储格式

     确认维表的存储格式是行存表、列存表还是行列共存。

  ii. 维表查询延迟高

     维表的使用，最常见的问题就是Flink/Blink侧用户反馈Join节点有反压，导致整个作业的吞吐上不去。

a. 确认Flink维表Join的模式

当前Hologres Flink Connector的维表Join功能支持同步和异步模式两种，异步模式性能要优于同步模式，具体需要看Flink SQL进行区分，以下是一个开启异步维表查询功能的SQL示例。

```
CREATE TABLE hologres_dim(
 id INT,
 len INT,
 content VARCHAR
) with (
  'connector'='hologres',
  'dbname'='<yourDbname>',  --Hologres的数据库名称。
  'tablename'='<yourTablename>',  --Hologres用于接收数据的表名称。
  'username'='<yourUsername>',  --当前阿里云账号的AccessKey ID。
  'password'='<yourPassword>',  --当前阿里云账号的AccessKey Secret。
  'endpoint'='<yourEndpoint>'  --当前Hologres实例VPC网络的Endpoint。
  'async' = 'true'--异步模式
);
```

b. 确认后端查询延迟

查看监控指标的实时写入延迟：

- 确认是否是列存表在做维表，列存表的维表在高QPS场景下开销很高。
- 如果是行存表，且延迟高，通常情况下是实例整体负载较高导致的，需要进行扩容。

iii. 确认Join的Key是否是Hologres表的主键

自VVR 4.x (Flink 1.13) 版本开始，Hologres Connector基于Holo Client实现了Hologres表的非主键查询，这种情况通常性能会比较差、实例负载也比较高，尤其是建表没有特别优化过的情况。这时候需要引导优化表结构，最常见的就是将Join的key设置成Distribution Key，这样就能实现Shard Pruning。

iv. 查看Blink侧的反压情况

如果上述步骤排查完成，发现Hologres侧没有明显的异常，通常情况下是客户端慢了，也就是Blink侧本身就慢了，这时候可以确认是否是Sink节点反压了。如果作业只有一个节点，就无法看出是否反压了，这时候可以将Sink节点单独拆开再观察。同样可以排查是否是Join节点导致的反压。具体请联系Flink技术支持排查。

常见报错

- `ERPC TIMEOUT` 或者 `ERPC CONNECTION CLOSED` 。
  - 报错现象：出现 `com.alibaba.blink.store.core.rpc.RpcException: request xx UpsertRecordBatchRequest failed on final try 4, maxAttempts=4, errorCode=3, msg=ERPC_ERROR_TIMEOUT` 报错。
  - 可能原因：写入时压力过大写入失败或者集群比较繁忙，可以观察Hologres实例的CPU负载是否打满。`CONNECTION CLOSED` 可能是负载过大导致后端节点挂掉了，出现OOM（Out Of Memory）或者Coredump。
  - 解决方法：请先重试写入，如果不能恢复请找Hologres技术支持人员排查原因。
- 报错： `BackPresure Exceed Reject Limit` 。
  - 可能原因：通常是Hologres后端写入压力过大，导致Memtable来不及刷盘导致写入失败。
  - 解决方法：如偶发失败可忽略该问题，或者Sink加上参数rpcRetries = '100' 来调大写入重试次数。如果一直报该错误，请联系Hologres技术支持人员确认后端实例状态。

- 报错：`The requested table name xxx mismatches the version of the table xxx from server/org.postgresql.util.PSQLException: An I/O error occurred while sending to the backend.Caused by: java.net.SocketTimeoutException: Read timed out` 。
  - 可能原因：通常是用户做了Alter Table导致Blink写入所带表的Schema版本号低于Server端版本号导致的，并且超过了客户端的重试次数。
  - 解决方法：如偶发报错可忽略该问题。如果一直报该错误，请联系Hologres技术支持人员。
- 报错：`Failed to query table meta for table` 。
  - 可能原因：一种可能是用户读写了一张Hologres的外部表，Hologres Connector不支持读写外部表。如果不是，可能是Hologres实例 Meta出现了问题。
  - 解决方法：请联系Hologres技术支持人员。
- 报错：`Cloud authentication failed for access id` 。
  - 可能原因：该报错通常是用户配置的AccesKey信息不对，或者用户没有添加账号至Hologres实例。
  - 解决方法：
    - 请检查当前账户的AccessKey ID和AccessKey Secret填写是否正确，一般是AccessKey Secret错误或者有空格。
    - 检查不出原因可以用当前AccesKey连接HoloWeb（使用账号密码方式登录），在测试联通性时看报错是什么，还是一样的报错说明AccesKey有问题，如果报错为 `FATAL: role"ALIYUN$xxxx"does not exist` 说明账号没有实例的权限，需要管理员给该账号授予权限。
- Hologres维表Join不到数据。
  - 可能原因：Hologres维表使用了分区表，Hologres维表暂不支持分区表。
  - 解决方法：请将分区表转为普通表。
- 报错：`Modify record by primary key is not on this table` 。
  - 可能原因：实时写入的时候选择了更新模式，但是Hologres的结果表没有主键。
  - 解决方法：请设置主键。
- 报错：`shard columns count is no match` 。
  - 可能原因：写入Hologres的时候，没有写入完整的Distribution Key的列（默认是主键）。
  - 解决方法：请写入完整的Distribution Key列。
- 报错：`Full row is required, but the column xxx is missing` 。
  - 可能原因：Hologres老版本的报错信息，通常是用户没有写某列数据，而那一列是不能为空的。
  - 解决方法：请为不能为空的列赋值。
- VVP用户读写Hologres导致JDBC连接数暴涨。
  - 可能原因：VVP Hologres Connector读写Hologres（除了Binlog），采用JDBC模式，最大占用 `读写Hologres表数量*并发度 * connectionSize(VVP表的参数，默认为3)` 个连接。
  - 解决方法：合理规划任务连接数，降低并发度或者connectionSize。如无法调低并发度或connectionSize，可以为表设置参数useRpcMode = 'true' 切回至Rpc模式。
- Blink/VVP用户读写Hologres报错显示无法连接Hologres。
  - 可能原因：Blink/VVP集群默认访问公网很慢或者无法访问。
  - 解决方法：需要保证和Hologres实例在相同Region，且使用VPC的Endpoint。

- 报错：`Hologres rpc mode dimension table does not support one to many join`。
  - 可能原因：Blink和VVP的RPC模式维表必须是行存表，且Join的字段必须是主键，报错的原因往往是以上两个条件不满足
  - 解决方法：建议使用JDBC模式，且维表使用行存表或者行列共存表。
- DatahubClientException
  - 报错现象：出现`Caused by: com.aliyun.datahub.client.exception.DatahubClientException: [httpStatus:503, requestId:null, errorCode:null, errorMessage:{"ErrorCode":"ServiceUnavailable","ErrorMessage":"Queue Full"}]`报错。
  - 可能原因：大量消费Binlog作业由于某种原因同时重启导致线程池被占满。
  - 解决方法：分批进行消费Binlog作业。
- Error occurs when reading data from datahub
  - 报错现象：出现`Error occurs when reading data from datahub, msg: [httpStatus:500, requestId:xxx, errorCode:InternalServerError, errorMessage:Get binlog timeout.]`报错。
  - 可能原因：Binlog每条数据太大，乘上攒批之后，每个RPC请求的大小超过最大限制。
  - 解决方法：在每行数据字段较多且有很长的字符串等字段时，可以减小攒批配置。
- 报错：`Caused by: java.lang.IllegalArgumentException: Column: created_time type does not match: flink row type: TIMESTAMP(6) WITH LOCAL TIME ZONE, hologres type: timestamp`。
  - 可能原因：在Flink中字段使用了TIMESTAMP(6)类型，当前不支持映射至Hologres。
  - 解决方法：修改字段类型为TIMESTAMP。

14.3. 对接MaxCompute常见问题与诊断

基本概念

- Hologres与MaxCompute的对比。

对比项	MaxCompute	Hologres
使用场景	ETL（Extract-Transform-Load）加工，面向数据明细层（DWD，Data Warehouse Detail）和数据服务层（DWS，Data WareHouse Service）。	交互式查询、在线数据服务，面向应用的数据服务（ADS）。
用户使用	异步的执行作业。	同步的Query。
集群资源	共享大集群，SaaS模式。	独享集群，PaaS模式。
计算引擎	基于Job Execution模型，将作业转化为Stage，每个Stage按需申请计算资源，执行过程中通过File持久化。	基于MPP模型，精细化内存资源管理，执行引擎常驻内存，用户态细粒度SQL算子调度，计算不落盘。
调度方式	进程级别，运行时按需申请、分配。	轻量级线程，资源启动时预留。
扩展性	几乎不受限制。	复杂查询尽量避免跨多节点数据Shuffle。
存储格式	列式。	行式、列式、行列共存。

对比项	MaxCompute	Hologres
存储成本	基于Pangu，成本低。	基于Pangu，利用SSD做缓存加速，成本相对高。
接口标准	MaxCompute SQL，类Hive。	PostgreSQL。

- Hologres外部表和内部表的适用场景
  - 新建外部表直接加速查询

    外部表不存储数据，数据存储在MaxCompute中，且外部表没有索引，全靠CPU资源进行计算，因此外部表比较适用于小数据量，低QPS（Queries-per-second）的查询。
  - 导入数据至Hologres内部表进行加速查询

    内部表的数据存储在Hologres中，当有数据更新、复杂查询、高QPS的查询时，建议导入内部表，能充分发挥Hologres底层的性能优势。

常见报错

- 报错：`specified partitions count in MaxCompute table: exceeds the limitation of 512, please add stricter partition filter or set axf_MaxCompute_partition_limit.` 或者 `Build desc failed: Exceeds the partition limitation of 512, current match xxx partitions.` 。
  - 报错原因：

    当前Hologres支持查询最多分区数为512个，查询超过此限制。
  - 解决方法：
    - 请添加分区过滤条件，使一次查询不超过512个分区。
    - 请将数据导入Hologres内部表，则没有分区限制，详情请参见使用SQL导入MaxCompute的数据至Hologres。
    - 使用如下命令调整每次Query命中的分区数，默认为512，最大为1024，不建议调整太大，否则会影响查询性能。

      ```
      -- V1.1及以上版本
      set hg_foreign_table_max_partition_limit = 128;
      -- V0.10版本
      set hg_experimental_foreign_table_max_partition_limit = xxx ;
      ```

      > **说明** 如果MaxCompute配置了多级分区，会按照最细粒度分区单位进行分区命中计数。

- 报错：`Build desc failed: Exceeds the scan limitation of 200 GB, current scan xxx GB.` 。
  - 报错原因：

    Hologres中默认最大的底层数据扫描量为200GB，此数据量是命中MaxCompute分区之后的扫描数据量，和MaxCompute自身存储数据量无关，查询超出此限制导致报错。

- 解决方法：
  - 增加过滤条件，命中更少的分区，使一次Query的扫描数据量在200GB以内。
  - 请将MaxCompute表数据导入至Hologres中，再进行查询，详情请参见使用SQL导入MaxCompute的数据至Hologres。
  - （不推荐）使用 `set hg_experimental_foreign_table_max_scan_size = xxx;` 命令设置参数调大数据量限制（其中xxx可以替换为业务的数据量，如400，单位为GB）。但是过分调大外部表数据量限制，可能无法得到预期的性能，也可能造成实例OOM（Out Of Memory），影响正常使用。

- 报错：`query next from foreign table executor failed, Not implemented` 。
  - 报错原因：MaxCompute表数据写入时使用Streaming Tunnel的方式写入，写入命令为 `tunnel.creatStreamUploadSession` 。Hologres暂时不支持这种Streaming Tunnel方式写入数据到MaxCompute表。
  - 解决方法：
    - 方法一：重新创建MaxCompute表，并且写入时将 `tunnel.createStreamUploadSession` 命令改为 `tunnel.createUploadSession` 。
    - 方法二：在MaxCompute中执行以下命令将MaxCompute表进行合并，详情请参见MaxCompute什么情况下会产生小文件？如何解决小文件问题？。同时写入时将 `tunnel.createStreamUploadSession` 命令改为 `tunnel.createUploadSession` 。

      ```
      ALTER TABLE tablename [PARTITION] MERGE SMALLFILES;
      ```

- 报错：`Build desc failed: failed to check permission: Currently not supported table type "view"` 。

  报错原因：目前暂时不支持MaxCompute的View。

- 报错：`Build desc failed: failed to get foregin table split:MaxCompute-0010000: System internal error - get input pangu dir meta fai` 。
  - 报错原因：

    Hologres读取MaxCompute的配置未及时更新。
  - 解决方法：

    请过几分钟重试，若是重试好几次都未成功，请联系技术支持处理。

- 报错：`ERROR: Query:[xxx] Build desc failed: failed to get foregin table split:ERPC_ERROR_CONNECTION_CLOSED` 。
  - 报错原因：

    MaxCompute小文件过多，导致请求的META超过远程过程调用协议（RPC，Remote Procedure Call Protocol)1GB的最大限制。
  - 解决方法：
    - 请执行以下命令进行小文件合并。

      ```
      set MaxCompute.merge.task.mode=sql;
      set MaxCompute.merge.lock.expire.time=0;
      ALTER TABLE <tablename> [PARTITION] MERGE SMALLFILES;
      ```

    - HologresV0.10.21及以上版本已优化，请升级Hologres实例，详情请参见实例升级。
    - 请联系MaxCompute技术支持从源头解决，如果数据量不大可直接将数据写入Hologres。

- 报错：`ERROR: status { code: SERVER_INTERNAL_ERROR message: "hos_exception: IO error: Failed to execute pangu open normal file , err: PanguParameterInvalidException" }`。
  - 报错原因：

    Hologres V1.1版本外部表的执行引擎直读MaxCompute Pangu加密数据存在问题。
  - 解决方法：

    请执行`set hg_experimental_enable_access_MaxCompute_orc_via_holo = off;`命令或者升级Hologres实例至最新版本。
- 报错：`failed to import foregin schema:Failed to get MaxCompute table:Not enable schema evolution`。
  - 报错原因：

    对MaxCompute表的元数据做了修改。
  - 解决方法：
    - 更新了MaxCompute外部表Schema之后（例如增加列、删除列操作），请执行IMPORT FOREIGN SCHEMA来做刷新。
    - 如果执行了`IMPORT FOREIGN SCHEMA`还报错的话，请重新建一次MaxCompute的表，再建外部表，因为MaxCompute修改Schema之后进入到`schema evolution`状态，Hologres无法读取这种状态的表。
- 报错：`Open ORC file failed for schema mismatch. Reader schema:`。
  - 报错原因：

    MaxCompute的表为ORC格式，表的DECIMAL类型存储方式改变（一般是MaxCompute新加了DECIMAL字段或者MaxCompute做了灰度配置变更），导致Hologres读MaxCompute的DECIMAL类型出错。
  - 解决方法：
    - 请执行`set MaxCompute.storage.orc.enable.binary.decimal=false`命令，重新导下MaxCompute数据。
    - 请将MaxCompute的表的DECIMAL类型改为DOUBLE类型，再重新刷新一遍数据。
- 报错：`failed to import foreign schema:Failed to get MaxCompute table:Not enable acid table`。
  - 报错原因：

    MaxCompute表是事务（Transactional）表。
  - 解决方法：

    当前不支持MaxCompute的Transactional表，建议改为普通表。
- 报错：`Request denied, may caused by server busy.`。
  - 报错原因：

    外部表资源占满，CPU用量严重超出。

- 解决方法：
  - 请优化SQL，使SQL更加充分合理的使用资源，详情请参见优化MaxCompute外部表的查询性能。
  - 降低并发度。
    a. 使用 `show hg_foreign_table_executor_max_dop;` 命令查看当前配置。
    b. 使用如下命令降低并发度，推荐调整为当前配置的一半。

```
-- 语法示例
set hg_foreign_table_executor_max_dop = <并发数>;
-- 使用示例
set hg_foreign_table_executor_max_dop = 18;
```

    并发数：外部表单个执行节点读取外部表数据的并发度，默认值为256，取值范围为0-1024。修改后的风险： 并发度太大可能造成实例OOM，导致导入、查询失败，甚至实例重启，以至于服务不可用。并发度太小会导致外表查询、外表导入内表性能较差。
  - 请导入数据至Hologres内部表，内部表可以设置索引，使查询性能更好，详情请参见使用SQL导入MaxCompute的数据至Hologres。

- 导入数据报错：`Query executor exceeded total memory limitation xxxxx: yyyy bytes used` 。
  - 报错原因：

    数据量太大或者导入逻辑太复杂，导致超出了内存限制。（实例由多个节点组成，一个节点标准的内存上限是64GB，节点内存会分为三部分，三分之一用于计算，三分之一用于缓存，三分之一用于元数据。这里的报错是计算内存超出了限制。）

- 解决方法：

  a. 查看执行计划

  可以执行 `explain analyze sql;` 命令查看执行计划中具体的数据行数。当导入Query包含查询，但部分表没有 `analyze` ，或者 `analyze` 过，但数据又有更新导致不准确，导致查询优化器决策连接顺序有误，会引起内存开销过高。

  对所有参与的内表、外表执行 `analyze tablename;` 命令，更新表的统计元信息，可以帮助查询优化器生成更优的执行计划。

  b. 设置单行导入条数

  当表的列数较多，单行数据量较大时，单次读取的数据量会更大，通过在SQL前加以下参数来控制单次读取数据的行数，可以有效减少OOM情况。

  ```
  set hg_experimental_query_batch_size = 1024;  -- 默认值为8192
  insert into holo_table select * from mc_table;
  ```

  c. 降低导入的并发度。

  降低导入并发度，也会有效减少导入过程中的内存开销，并发度通过参数 `hg_foreign_table_executor_max_dop` 控制，默认为实例的Core数，可以在导入时设置更小的参数，降低导入的内存使用。

  ```
  set hg_foreign_table_executor_max_dop = 8;
  insert into holo_table select * from mc_table;
  ```

  d. 排查外表重复数据是否过多。

  如果以上操作都做完了，还是导入不了，如果使用的是 `insert on conflict` 命令，请排查是否外表重复数据太多，重复数据太多也会导致导入性能不好，可以在MaxCompute做重复数据去重，再进行导入，详情请参见多行数据合并为一行数据。

  e. 升级新版本动态调整内存。

  Hologres从V1.1.24版本开始，会对内存进行动态调整，后台会实时刷新当前内存水位，若是有空闲，则会分配更多内存给计算使用，请升级Hologres至最新版本，具体操作请参见实例升级。

  f. 扩容。

  如果以上操作都做完了，导入数据还是不成功，请对Hologres扩容，详情请参见升配。

- 报错： `Timestamp overflow detected while converting timestampfrom orc VectorBatch to arrow` 。
  - 报错原因：

    在MaxCompute表中有TIMESTAMP类型，使用Tunnel写入后TIMESTAMP精度会变成纳秒，但Hologres支持的精度是微秒，造成转换失败。

  - 解决方法：

    升级版本到 V1.1.70及以上版本；或者在MaxCompute中将TIMESTAMP类型转换为DateTime类型。

- 报错： `query next from foreign table executor failed, userinfao fail` 。
  - 报错原因：

    当前MaxCompute表是存储加密的表，在HologresV1.1以下版本还不支持读取。

- 解决方法：

  请将Hologres实例升级至V1.1以上版本，详情请参见实例升级。

- 报错：`You have NO privilege 'MaxCompute:Select' on xxx`。
  - 报错原因：

    当前账号不具备MaxCompute表的查询（Select）权限。

  - 解决方法：

    请联系MaxCompute管理员在MaxCompute中授予当前账号查询表（Select）的权限，具体操作请参见权限列表。

- 报错：`The sensitive label of column 'xxx' is 2, but your effective label is 0`。
  - 报错原因：

    当前账号只有MaxCompute表的部分字段权限。

  - 解决方法：
    - 核对有权限的账号和报错的账号是否为同一个账号，若是真的没有权限，可以去申请MaxCompute的权限，或者只过滤有权限的字段查询。获取MaxCompute表全部字段的权限，具体操作请参见授权。
    - 若是有权限，并且也只查询了有权限的字段，在实例比较老的版本可能存在缺陷，您可以在执行的Query前增加如下命令解决报错问题。

      ```
      set hg_experimental_enable_MaxCompute_executor=on;
      set hg_experimental_enable_query_master=on;
      ```

      更多关于MaxCompute的权限问题，请参见MaxCompute权限相关。

查询外部表速度慢如何解决？

建议优化SQL，详情请参见优化MaxCompute外部表的查询性能。

14.4. 监控指标常见问题

本文为您介绍Hologres监控指标相关的常见问题。

- 连接数过多时如何查看有哪些连接以及Kill连接？
- 延迟过高时如何查看慢Query？
- 当前实例水位很高，如何查看正在运行的任务？
- Query长时间不结束，如何Kill以及设置超时时间？
- 内存使用率高如何解决？
- 为什么只有一个任务，Hologres实例CPU使用率就达到100%？
- CPU使用率长期达到100%如何解决？

连接数过多时如何查看有哪些连接以及Kill连接？

连接数包括实例中总的SQL连接数，包括Active、Idle状态的JDBC/PSQL等连接。实例的连接数通常与实例的规格有关，如果您发现连接数过多，甚至出现超出实例最大连接数的情况，或者遇到如下报错：

- 产生`FATAL: sorry, too many clients already connection limit exceeded for superusers`报错。
- 产生`FATAL: remaining connection slots are reserved for non-replication superuser connections`报错。

说明实例连接数已达上限，意味着您需要检查您的实例是否存在连接数泄漏情况。通过HoloWeb，在**活跃连接管理**页面，可以对不符合预期或者Idle连接进行Kill操作，详情请参见HoloWeb可视化管理连接。

> **说明** Superuser账号可以看到实例全部连接。

延迟过高时如何查看慢Query？

通过在HoloWeb**诊断与优化**模块的**历史慢Query**页面可视化查看慢Query，详情请参见慢Query日志查看与分析。

常见降低查询延迟的方法如下。

- 让写入在查询低峰期进行，或者降低写入的并发度，提高查询效率，如果是外表写入，可以使用以下参数降低并发度。

```
--设置MaxCompute执行的最大并发度，默认为128，建议数值设置小一些，避免一个Query影响其他Query，导致系统繁忙导致报错。
set hg_experimental_foreign_table_executor_max_dop = 32; --优先考虑设置
--调整每次读取MaxCompute表batch的大小，默认8192。
set hg_experimental_query_batch_size = 1024;
--直读orc
set hg_experimental_enable_access_odps_orc_via_holo = on;
--设置MaxCompute表访问切分spilit的数目，可以调节并发数目，默认64MB，当表很大时需要调大，避免过多的split影响性能。
set hg_experimental_foreign_table_split_size = 512MB;
```

- 优化查询，使查询的效率更高，可以更加快速的获取查询结果。
- 扩容实例，详情请参见实例升配。

当前实例水位很高，如何查看正在运行的任务？

通过在HoloWeb**诊断与优化**模块的**活跃Query**页面可视化查看，详情请参见查看活跃Query。

Query长时间不结束，如何Kill以及设置超时时间？

您可以通过在HoloWeb**诊断与优化**模块的**活跃Query**页面可视化查看活跃Query，并对Query进行Kill操作；在**SQL编辑器**模块使用SQL命令设置Query超时时间，详情请参见Query管理。

内存使用率高如何解决？

Hologres实例的内存使用率为内存综合使用率。Hologres的内存资源采用预留模式，在没有查询的时候，也会有数据表的元数据、索引、数据缓存等加载到内存中，以便加快检索和计算，此时内存使用率不为零是正常情况。理论上，在无查询的情况，内存使用率达到30%~40%左右都属于正常现象。

一些情况下，会使得内存使用率持续升高，甚至接近100%。主要原因如下：

- 表越来越多，数据总量越来越大，以至于数据规模远大于当前计算规格。由于内存使用率和元数据、索引量存在正相关关系，因此，表的数量越多，数据量越大，索引越多，都会导致内存使用率升高。
- 索引不合理，例如表的列特别多，TEXT列居多，设置了过多的Bitmap或Dictionary索引。此情况可以考虑去掉一些Bitmap或者Dictionar索引，详情请参见ALTER TABLE。

但是当内存使用率稳定增长，长期接近100%时，通常意味着内存资源可能成为了系统的瓶颈，可能会影响实例的稳定性和或性能。稳定性影响体现在当元数据等过大，超额占据了正常Query可用的内存空间时，在查询过程中，可能会偶发`SERVER_INTERNAL_ERROR`、`ERPC_ERROR_CONNECTION_CLOSED`、`Total memory used by all existing queries exceeded memory limitation`等报错。性能影响体现在当元数据等过大，超额占据了正常Query本来能够使用的缓存空间，从而缓存命中会减少，Query延迟会增加。

因此当内存长期接近100%时，有如下几个操作建议：

- 删除不再查询的数据，以释放元数据等占的内存。
- 设置合理的索引，若是业务场景用不上的bitmap和dictionary，可以去掉，但不建议直接去掉，需要根据业务情况具体分析。
- 升配实例的计算和存储资源。对于升配的建议是：
  - 普通场景：可以容许读磁盘数据的延迟，响应时间要求不严格，1CU（1Core+4GB内存）可以支持50~100GB的数据存储。
  - 响应时间要求低的Serving场景：最好查询热点数据全在内存的缓存中。内存中缓存的比例默认占总内存的30%，即1CU（1Core+4GB内存）其中1.3GB用于数据缓存，同时数据缓存还会被表的元数据所使用一些。举个例子，低响应要求的场景，热点数据如果是100GB，那么最好要求100GB 在缓存可用（实际上数据读出来解压后，占用内存不止100GB），因此至少需要约320GB内存以上，从而推算计算资源至少需要96CU左右。

为什么只有一个任务，Hologres实例CPU使用率就达到100%？

Hologres实例的CPU使用率为实例的CPU综合使用率。Hologres因其可以充分发挥多核并行计算的能力，通常来说单个查询可以迅速将CPU使用率提高到100%，这说明计算资源得到了充分利用。CPU使用率高不是问题，CPU使用率高了之后，查询慢写入慢才是问题，需要综合分析。

CPU使用率长期达到100%如何解决？

当Hologres实例CPU使用率长期接近100%时（例如CPU使用率连续3小时满载100%，或者连续12小时达到90%以上等），说明实例负载非常高，这通常意味着CPU资源成为了系统的瓶颈，需要分析具体的业务场景和查询，以判断原因。可以从以下几方面进行排查：

- 查看是否有比较大的离线数据导入（INSERT），且数据规模还在日渐增长。

  可以考虑降低写入并发，或者错峰写入，不至于影响查询。

- 查看是否有高QPS的查询或写入，共同用满了CPU使用率。

  可以考虑降低写入并发，让查询先完成。

- 查看活跃Query是否有数据量大的Query。

  可以在HoloWeb**活跃Query**页面查看是否有Query一直没有结束，若是不符合业务预期的可以执行KILL操作，详情请参见Query管理。

- 如果确定是业务场景需要，将CPU资源用满，可以对实例进行扩容，以适应更复杂的查询或更大的数据量，详情请参见实例升配。

14.5. SQL语句的常见问题

如何处理数字开头的字段？

交互式分析Hologres兼容PostgreSQL，使用语法同PostgreSQL，不支持数字开头的字段。

如果您使用Hologres时遇到数字开头的字段，查询时需要为该字段增加双引号，示例如下。

```
select bizdate,"1_day_active_users","7_day_active_users" from t_active_users;
```

执行TRUNCATE语句报错

- 问题现象

执行 `TRUNCATE TABLE <table name>;` 语句，出现 `ERROR: TRUNCATE TABLE is not supported now.` 报错。

- 问题原因

  当前使用的Hologres版本较低。

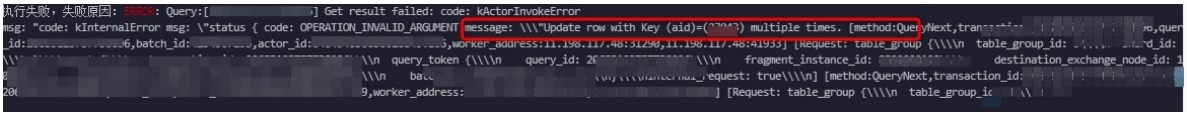
- 解决方法

  请提交工单，升级Hologres至0.8及以上版本。

执行INSERT ON CONFLICT语句报错

- 问题现象

  对数据源执行 `INSERT ON CONFLICT` 语句时出现如下报错。



- 问题原因

  Hologres兼容PostgreSQL，使用的也是标准PostgreSQL语法。在标准的PostgreSQL语义中，对数据源执行 `INSERT ON CONFLICT` 语句时，数据源不能包含重复数据，如果包含重复数据则会产生上述报错。

  > **说明** 数据源重复是指待插入的数据中包含重复数据，不是指待插入的数据与表里的数据重复。

  使用 `INSERT ON CONFLICT` 语句插入数据时包含重复数据，示例语句如下。

```
insert into tmp1_on_conflict values(1,2,3),(1,2,3) on conflict(a) do update set (a, b ,c )= ROW(excluded.*);
```

- 解决方法

  如果数据源包含重复数据，可以配置 `set hg_experimental_affect_row_multiple_times_keep_first = on;` ，保留重复数据的第一条数据。

14.6. 权限相关问题

14.6.1. RAM用户授权相关

Hologres管理控制台主要集成了RAM鉴权和实例的部分开发权限，本文内容将为您介绍Hologres管理控制台权限相关的常见问题解决方法。

问题汇总

您需要具备Hologres管理控制台操作权限，才能管理Hologres实例时。与Hologres管理控制台权限相关的常见问题，请单击如下链接查看解决方法。

- RAM用户如何购买交互式分析Hologres
- RAM用户无法查看实例列表及实例ID
- 被授权为Superuser的RAM用户无法新增用户
- 无法查看用户管理和DB管理模块的信息
- RAM用户无实例的操作权限

RAM用户如何购买交互式分析Hologres

RAM用户需要被主账号授权，才可以购买Hologres实例，详情请参见授予RAM用户权限。

> **说明** RAM用户成功购买实例后，您还需要使用主账号授予RAM用户实例的开发权限，才能正常使用实例，详情请参见授予RAM用户实例的开发权限。

RAM用户无法查看实例列表及实例ID

- 问题现象

  RAM用户选择了正确的地域，但无法看到已购买的实例，并提示**暂无权限查看所有实例，请让主账号前往RAM中心授予当前用户“xxx/*”资源的“hologram:ListInstances”权限**报错。

- 问题原因

  当前RAM用户没有查看实例列表的权限。

- 解决方法

  主账号登录RAM控制台，授予RAM用户显示实例列表的权限AliyunHologresReadOnlyAccess。

被授权为Superuser的RAM用户无法新增用户

- 问题现象

  RAM用户已经被授权为Superuser，但是在Hologres管理控制台的**用户管理**模块无法看到用户列表并新增用户。提示**暂无权限列出所有子账号，请让主账号前往RAM中心授予当前用户ram:ListUsers权限**报错。

- 问题原因

  当前RAM用户在Hologres管理控制台没有查看用户列表的权限。

- 解决方法

  主账号登录RAM控制台，授予RAM用户显示用户列表的权限**AliyunRAMReadOnlyAccess权限策略**。

无法查看用户管理和DB管理模块的信息

- 问题现象

  RAM用户登录Hologres管理控制台后，无法查看**用户管理**和**DB管理**模块的信息。提示**暂无权限，请找Superuser将当前账户添加到实例中**报错。

- 问题原因

  当前RAM用户没有实例的开发权限。

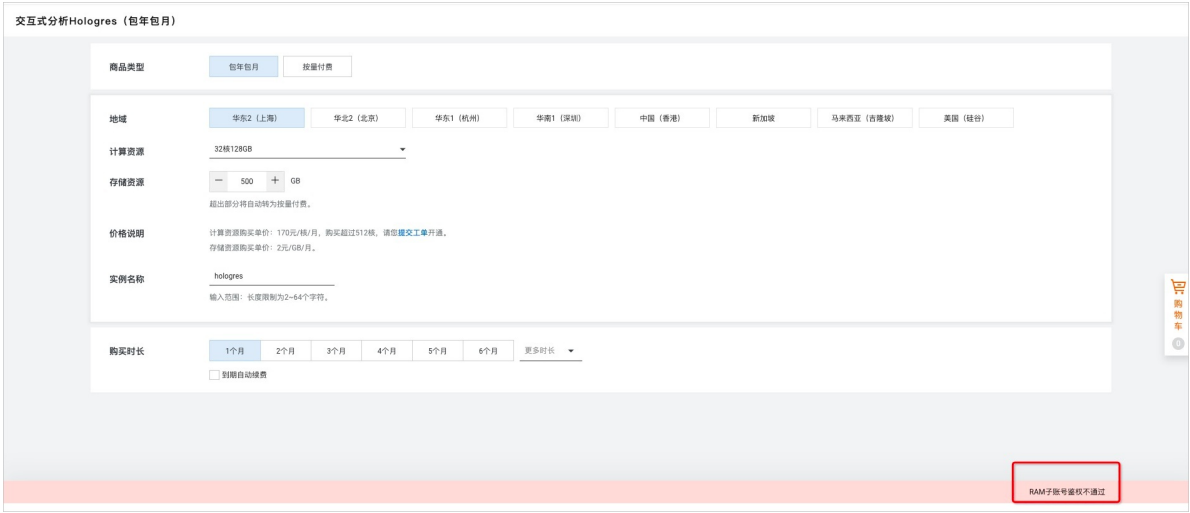
- 解决方法

  主账号或具有Superuser权限的账号为RAM用户授予实例的开发权限，详情请参见授予RAM用户实例的开发权限。

RAM用户无实例的操作权限

- 问题现象

  被授权为Superuser的RAM用户无法购买、升配和降配实例，以及转换实例的付费方式由**按量付费**为**包年包月**。提示**RAM子账号鉴权不通过**报错。



- 问题原因

  购买、升配、降配及转换付费方式等操作涉及费用账单，由主账号统一控制，当前RAM用户没有相关权限。

- 解决方法

  主账号登录RAM控制台，授予RAM用户与实例费用相关的权限AliyunHologresFullAccess及AliyunBSSOrderAccess。

14.6.2. Hologres权限相关

在使用Hologres开发时，可能会因为权限问题产生报错。本文内容将为您介绍几个Hologres实例开发常见权限问题的解决方法。

问题汇总

在使用Hologres开发时，需要有实例的具体开发权限，才能在实例内进行操作。与实例权限相关的常见问题，请单击如下链接查看解决方法。

- 如何选择合适的Hologres权限模型？
- 在Hologres实例内操作报错：role “RAM$xxx” doesn't not exsit
- 连接实例时报错：Cloud authentication failed for access id "xxxx"
- 查询表时报错： permission denied for table xxxx
- 在Hologres实例内操作报错：permission denied for database “xxx” detail：user does not have CONNECT privilege
- 使用call spm_enable()命令时，提示：because roles conflict
- 授权时报错：current database is NOT in simple privilege mode
- 操作表时报错：must be the owner of table xxxx

如何选择合适的Hologres权限模型？

Hologres具有专家权限模型、简单权限模型和基于Schema级别的简单权限模型（简称SLPM），那么该如何选择合适的Hologres权限模型，具体操作场景说明如下：

- 专家权限模型指的是Postgres原生的权限模型，若您对Postgres及其权限管理已经比较熟悉，可以零学习成本使用专家权限模型实现授权操作。如果您需要细粒度到表级别的权限管理，并且有充足的时间、精力和必要性对每个表、每个用户进行权限赋予、回收等操作可以使用专家权限模型。
- 简单权限模型（简称SPM）指的是DB级别的简单权限管理模型，所有需要访问DB的用户都需要加入到某个用户组中，每个用户组都有DB中任意Schema下对象特定的访问权限。如果您很少采用Schema进行开发，或者只是像使用目录一样采用Schema对表对象进行分类，而无需Schema级别的权限隔离，推荐您使用简单权限模型SPM。
- 基于Schema级别的简单权限模型（简称SLPM），每个Schema都有自己的developer、writer、viewer用户组。若您强依赖Schema级别的用户隔离，表权限隔离，推荐您使用SLPM。

在Hologres实例内操作报错：role “RAM$xxx” doesn't not exsit

- 问题现象

  连接开发工具之后，在实例内进行查询等操作时报错提示：role “RAM$xxx” doesn't not exsit。

- 问题原因

  当前RAM用户账号未被创建进实例中。

- 解决方法

  您可以根据业务情况，为当前RAM用户账号授予实例的相关权限，如Superuser。更详细的授权请参见授予RAM用户实例的开发权限。

连接实例时报错：Cloud authentication failed for access id "xxxx"

- 问题现象

  在连接开发工具时报错提示Cloud authentication failed for access id "xxxx"。

- 问题原因

  当前账号填写的密码错误。

- 解决方法

  您需要检查当前连接的密码，需要确保密码为Access key且不包含空格。更多关于Access key的详细内容请参见创建访问密钥。

查询表时报错： permission denied for table xxxx

- 问题现象

  在实例内进行查询等操作时报错提示Execution failed：ERROR: permission denied for table xxxx。

- 问题原因

  当前账号没有对应表的查看权限。

- 解决方法

  您可以根据当前授权模式的不同，选择对应的解决方法。

  - 专家权限模式授权：执行如下命令为用户授权，其中p4_UID为RAM用户的账号信息。更多专家权限模式的信息，请参见专家权限模型。

    ```
    grant select on table tablename to "p4_UID";
    ```

  - 简单权限模型：将当前用户加入viewer及以上用户组。具体操作请参见用户授权。

在Hologres实例内操作报错：permission denied for database “xxx” detail：user does not have CONNECT privilege

- 问题现象

  连接开发工具之后，在实例内进行查询等操作时报错提示FATAL:permission denied for database “xxx” detail：user does not have CONNECT privilege。

- 问题原因

  当前账号只被创建进了实例中，但是没有对应实例的开发权限。

- 解决方法

  您可以根据业务情况，为当前用户账号授予实例的相关权限，如Superuser。更详细的授权请参见授予RAM用户实例的开发权限。

使用 `call spm_enable()` 命令时，提示：because roles conflict

- 问题现象

  在专家模式下，执行 `call spm_enable()` 会提示：because roles conflict类似报错。

- 问题原因

  是因为当前数据库之前开启过简单模式，存在残留信息。

- 解决方法

您需要执行 `call spm_enable ('t');` 开启。

授权时报错：current database is NOT in simple privilege mode

- 问题现象

  进行授权操作时，报错：current database is NOT in simple privilege mode。

- 问题原因

  当前数据库未开启简单权限模型。

- 解决方法

  i. 请执行 `show hg_experimental_enable_spm;` 命令检查是否关闭了简单权限模型。

  ii. 使用如下命令开启当前数据库的简单权限模型。

  ```
  -- 开启当前DB的简单权限模型
  call spm_enable ('t');
  -- 将DB中已有的对象change owner到develoepr，使用SPM管理
  call spm_migrate ();
  ```

  `call spm_enable ('t');` 括号里't'的含义：将 `call spm_disable` 关掉后，原系统角色不会删除，会继续保留这些角色和权限。如果下次使用 `spm_enable()` 不带t，就会显示角色冲突，而无法开启。 `spm_enable ('t')` 就是忽略这个冲突，继续重用这些系统角色。

操作表时报错：must be the owner of table xxxx

- 问题现象

  在实例内进行操作时报错提示：must be the owner of table xxxx。

- 问题原因

  当前RAM用户不是表的owner，无法创建分区子表或者执行删除表的操作。

- 解决方法

  您可以根据当前授权模式的不同，选择对应的解决方法。

  - 专家权限模式授权：执行如下命令将表的owner授予当前账号，其中p4_UID为RAM用户的账号信息。更多专家权限模式的信息，请参见专家权限模型。

    ```
    alter table tablename owner to "p4_UID";
    ```

  - 简单权限模型：将当前用户加入developer及以上用户组。具体操作请参见用户授权。

14.6.3. DataWorks权限相关

Hologres与DataWorks深度集成，在使用DataWorks进行Hologres开发时，可能会因为权限问题产生报错。本文内容将为您介绍几个常见权限问题的解决方法。

问题汇总

与DataWorks权限相关的问题及解决方法，请单击如下链接查看。

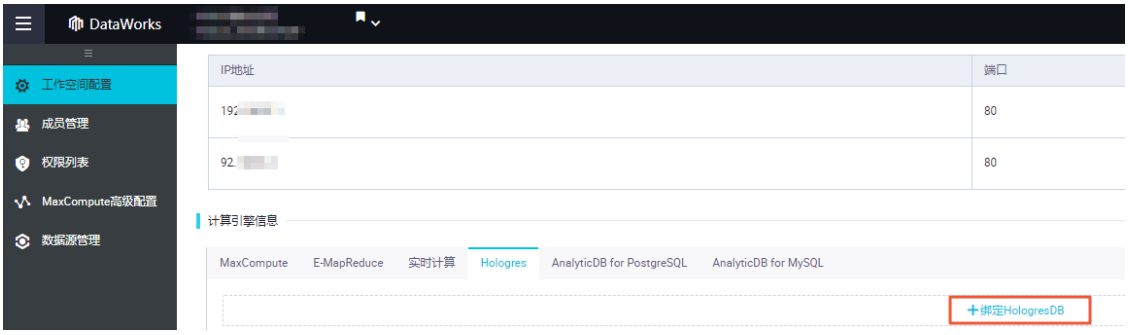
- 工作空间配置绑定HologresDB按钮为灰色
- 绑定HologresDB时测试连通性不通过
- 绑定HologresDB时报错“同名计算引擎实例已经存在”或者“租户系统内部错误”

- 数据服务查询报错“permission denied”
- DataWorks生产环境报错“permission denied for schema xxx”

工作空间配置绑定HologresDB按钮为灰色

- 问题现象

  当您在DataWorks管理控制台**工作空间配置**页面的计算引擎信息页签绑定HologresDB时，**+绑定HologresDB**按钮置灰，无法执行绑定操作。



- 问题原因
  - RAM用户不具有工作空间的管理权限。
  - RAM用户不具有实例的开发权限。
- 解决方法
  - 如果RAM用户不具有工作空间的管理权限，您可以在**成员管理**页面为当前用户授权，具体操作请参见角色及成员管理：空间级。
  - 如果RAM用户不具有实例的开发权限，您可以根据用户权限模式的不同，前往Hologres进行授权。
    - 简单权限模型授权，请参见新增用户。
    - 专家权限模型授权，请参见专家权限模型授权。

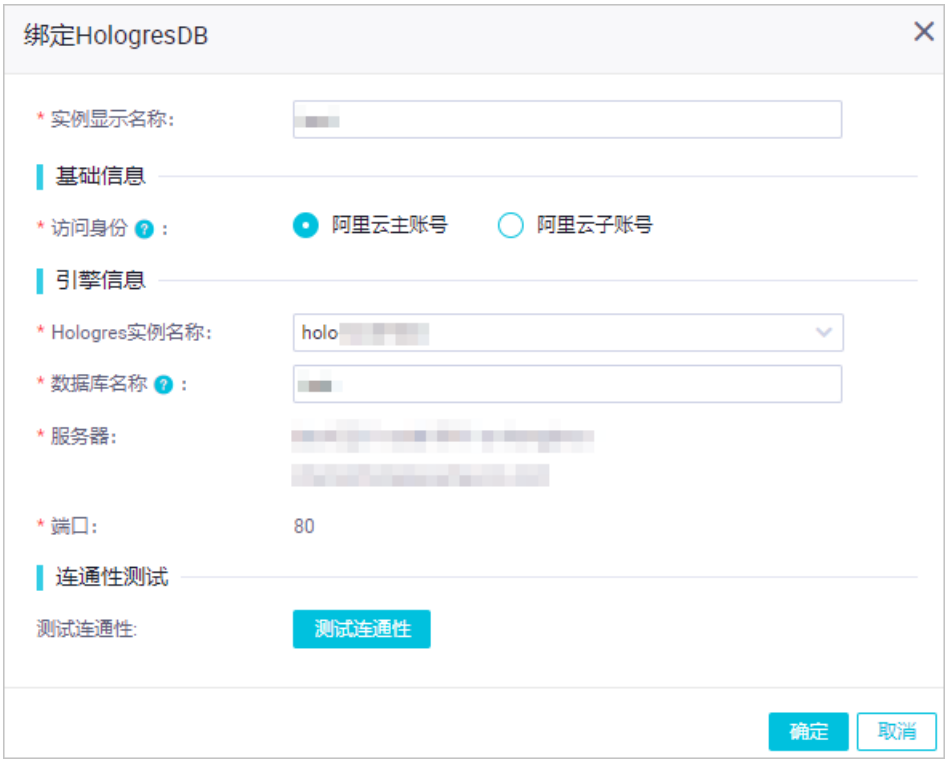
绑定HologresDB时测试连通性不通过

- 问题现象

  登录DataWorks管理控制台配置完成**绑定HologresDB**弹框参数时，单击**测试连通性**按钮，结果提示 测试不通过。

- 问题原因

如下图所示，在绑定HologresDB时需要配置相关信息，包括访问身份、Hologres实例名称和数据库名称等信息，信息填写不正确或账号权限不足均会导致报错。



- 解决方法

  i. 检查Hologres实例运行状态。

  登录Hologres管理控制台，在左侧导航栏单击实例列表进入**Hologres引擎管理**页面，您需要检查，确保从下拉框选取的实例为正常运行状态。

  ii. 检查数据库是否存在。

  实例运行状态正常的情况下，您可以前往实例详情页的**数据库管理**页签查看当前输入的数据库名称是否存在。否则您需要重新输入已经存在的数据库名称或者新增数据库，具体操作请参见DB管理。

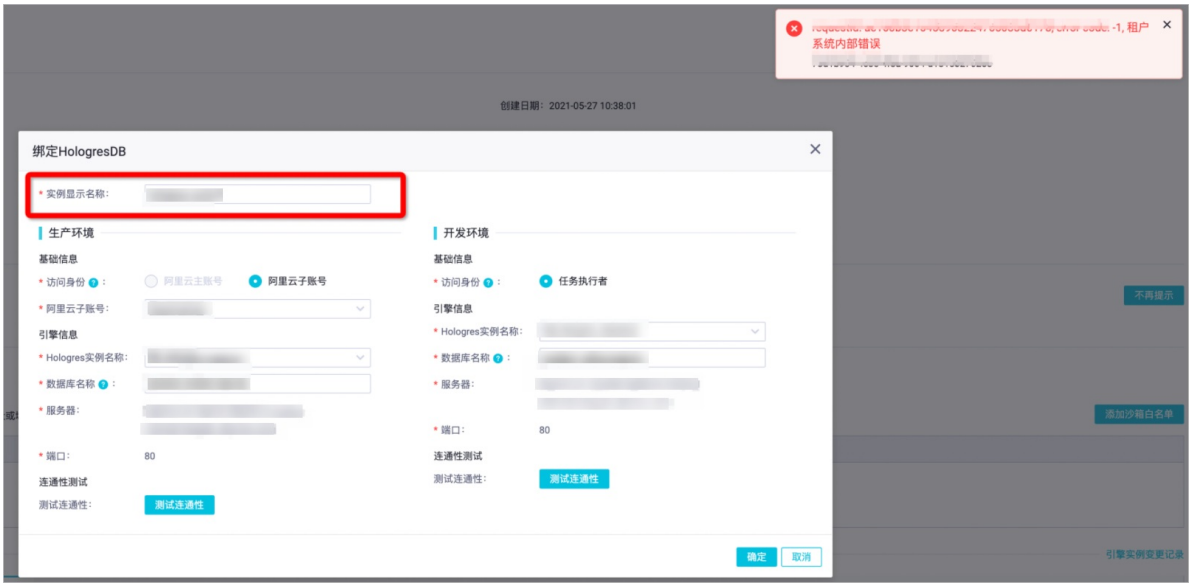
  iii. 检查当前访问身份权限。

  您可以前往实例详情页的**用户管理**页签查看当前访问身份的权限。否则您需要添加用户并授权，具体操作请参见用户管理。

绑定HologresDB时报错“同名计算引擎实例已经存在”或者“租户系统内部错误”

- 问题现象

  当您在**绑定HologresDB**弹框配置完参数，单击**确定**，报错提示 同名计算引擎实例已经存在 或者“租户系统内部错误”。



- 问题原因

  Hologres的实例显示名称与之前的实例名称重复。

- 解决方法

  修改实例显示名称后再次单击**确定**。

数据服务查询报错“permission denied”

- 问题现象

  登录DataWorks管理控制台在**数据服务**页面查询表，报错提示“Execution failed：ERROR: permission denied for table xxxx”。

- 问题原因

  当前执行数据服务配置的用户账号没有查询表的权限。

- 解决方法

  i. 检查数据服务Hologres数据源配置的账号是否正确。

  您可以登录DataWorks管理控制台，进入**数据源管理**页面，检查数据源用户账号配置。具体操作请参见配置Hologres数据源。

  ii. 检查数据服务Hologres数据源配置的账号是否有查看表的权限。

  您可以登录DataWorks管理控制台，进入**数据源管理**页面，检查数据源账号权限配置。具体操作请参见通过RAM角色授权模式配置数据源。

  更多关于DataWorks的权限以及相关授权操作，请参见附录：预设角色权限列表（空间级）。

DataWorks生产环境报错 `permission denied for schema xxx`

- 问题现象

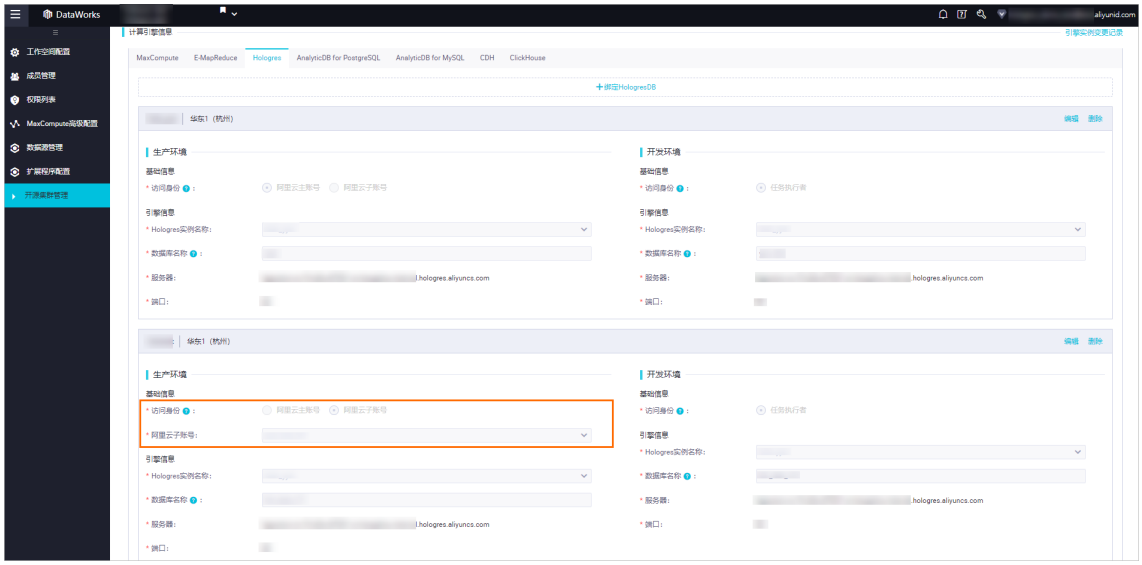
  在DataWork中，将SQL发布至生产环境中运行时报错：`permission denied for schema xxx`。

- 问题原因

  当前生产账号的权限没有实例对应Schema的访问权限。

- 解决方法

i. 前往DataWorks工作空间配置页面，查看生产环境绑定的是哪个账号访问，详情请参见配置工作空间。



ii. 登录HoloWeb，在**安全中心**页面的**用户管理**或者**DB授权**查看访问生产环境账号的权限。

iii. 如果账号没有权限，即可在**用户管理**或者**DB授权**页面对绑定的生产环境账号进行授权，详情请参见DB管理。

14.6.4. MaxCompute权限相关

使用Hologres加速查询MaxCompute表数据时，可能会因为权限问题产生报错。本文内容将为您介绍几个常见权限问题的解决方法。

问题汇总

与MaxCompute权限相关的问题及解决方法，请单击如下链接查看。

- 查询外部表报错：You have NO privilege 'odps:Select' on xxx
- 查询外表报错：The sensitive label of column 'xxx' is 2, but your effective label is 0
- 跨project访问MaxCompute表报错：You have NO privilege 'odps:Select' on xxx
- 创建外部表报错：You have NO privilege 'odps:List' on xxx
- 创建外部表时报错：Access denied by project ip white list: sourceIP:'xxxx' is not in white list. project: xxxx
- 创建外部表时报错：You don't exist in project xxx

查询外部表报错：You have NO privilege 'odps:Select' on xxx

- 问题现象

  当您在Hologres管理控制台创建外部表之后，查询外部表时报错提示“You have NO privilege 'odps:Select' on xxx”。

- 问题原因

  当前账号不具备MaxCompute表的查询（Select）权限。

- 解决方法

  需要MaxCompute管理员在MaxCompute中授予当前账号查询表（Select）的权限，具体操作请参见权限列表。

查询外表报错：The sensitive label of column 'xxx' is 2, but your effective label is 0

- 问题现象

  当您在Hologres管理控制台创建外部表之后，查询外部表时报错提示“The sensitive label of column 'xxx' is 2, but your effective label is 0”。

- 问题原因

  当前账号只有MaxCompute表的部分字段权限。

- 解决方法

  您可以选择如下三种方法中的一种来解决该问题：

  - （推荐）建议您提交工单将当前实例版本升级至V0.8。
  - 您可以在执行的Query前增加如下参数解决报错问题。

    ```
    set hg_experimental_enable_odps_executor=on;
    set hg_experimental_enable_query_master=on;
    ```

  - 获取MaxCompute表全部字段的权限，具体操作请参见权限列表。

跨project访问MaxCompute表报错：You have NO privilege 'odps:Select' on xxx

- 问题现象

  当前账号已经具备MaxCompute表查询权限，但是跨project访问MaxCompute表报错”You have NO privilege 'odps:Select' on xxx“。

- 问题原因

  若是当前账号已经具备MaxCompute已经有表的查询权限，跨project访问MaxCompute表还是报错，则MaxCompute当前可能采用的是package授权，您需要添加SQL语句解决该问题。

- 解决方法

  当前如果MaxCompute是project授权方式，在Hologres中，您可以在SQL前添加如下参数解决。

  ```
  // V0.7版本的实例请执行以下语句授权
  set seahawks.seahawks_internal_current_odps_project='holoprojectname';
  //V0.8版本的实例请执行以下语句授权
  set hg_experimental_odps_current_project_name = 'holoprojectname';
  ```

创建外部表报错：You have NO privilege 'odps:List' on xxx

- 问题现象

  当您在Hologres管理控制台使用HoloWeb或HoloStudio可视化创建外部表时报错提示“You have NO privilege 'odps:List' on xxx”。

- 问题原因

  当前账号在MaxCompute中不具备查看所有表（List）的权限。

- 解决方法
  - 需要MaxCompute管理员在MaxCompute中授予当前账号查看所有表（List）的权限，具体操作请参见权限列表。
  - 使用SQL语句创建外部表查询数据，具体操作请参见通过创建外部表加速查询MaxCompute数据。

创建外部表时报错：Access denied by project ip white list: sourceIP:'xxxx' is not in white list. project: xxxx

- 问题现象

  当您在Hologres管理控制台使用HoloWeb创建外部表时报错提示“Access denied by project ip white list: sourceIP:'xxxx' is not in white list. project: xxxx”。

- 问题原因

  当前MaxCompute集群设置了白名单访问，Holoweb不在白名单内。

- 解决方法

  当MaxCompute项目开启白名单功能时，仅允许白名单内的设备访问项目空间；非白名单内的设备访问项目空间时，即使拥有正确的AccessKey ID及AccessKey Secret，也无法通过鉴权。因此需要将报错信息中的IP设置为白名单才可以创建外表，具体操作请参见管理IP白名单。

创建外部表时报错：You don't exist in project xxx

- 问题现象

  当您在创建外部表时报错提示“You don't exist in project xxx”。

- 问题原因

  执行创建外部表的账号不具有访问对应MaxCompute Project的权限。

- 解决方法

  请先确认需要访问的MaxCompute Project名称，如果Project名称错误请先换成正确的Project名。如果Project名称正确仍然报同样的错误，需要前往MaxCompute中给报错的账号授权，详情请参见权限概述。

14.7. 其他问题

14.7.1. 如何获取更多的在线支持？

Q：如何获取更多关于交互式分析的在线支持？

您可以通过以下途径获取更多关于交互式分析Hologres的在线支持：

1. **云栖社区团队号--阿里云交互式分析**

   交互式分析有官方的阿里云云栖社区团队号--**阿里云交互式分析**，团队会定期发布关于产品的最新咨询、学习路径、技术交流等，详情请关注：阿里云交互式分析。

2. **钉钉交流群**

   使用钉钉搜索钉钉群号32314975加入钉钉群进行反馈，会有产品技术专家在线提供服务，同时定期发布优惠活动，及新特性预告等。

15.相关协议

15.1. 实时数仓Hologres服务等级协议（SLA）

实时数仓Hologres服务等级协议（SLA）的详情，请参见实时数仓服务等级协议。

15.2. 产品生命周期策略与版本

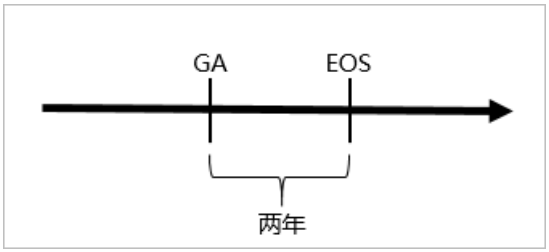
本文为您介绍Hologres的产品生命周期策略及产品终止策略、产品版本号及含义、版本对应关系和产品生命周期重要事件点。

背景信息

产品的更新换代是基础技术软件领域的普遍规律。Hologres正在处于快速发展中，新版本带来了更加丰富强大易用的产品特性，随之而来的是已经服务于用户的诸多版本与产品形态会逐步老化，功能逐渐不能满足用户不断丰富的业务需求。在应用的稳定性、产品易用性、系统兼容性等多方面，将面临越来越多的挑战，老版本会逐步被新版本所替代。对产品进行生命周期管理，有节奏的引入产品新版本，更加有利于用户技术的与时俱进，增强用户使用产品的效能。考虑到用户的迁移、验证新版本的内部成本与稳定性，以及整体长期IT规划等方面，为了更好地为您提供企业级服务、加强对产品未来演进的洞见，Hologres构建了对应的生命周期管理体系，明确了产品生命周期策略及产品终止策略，帮助您知晓详细规则，提早做好相应准备。

产品生命周期策略

产品生命周期服务策略（Runtime Lifecycle Policy，RLP）描述Hologres从正式交付用户使用，到最终停止对外服务的过程。Hologres的RLP详情如下图所示。



里程碑	全称	定义	阶段时间点	说明
GA	General Availability	正式交付	T月	可以正式发布交付给用户在生产环境使用。
EOS	End of Service	服务中止	T月+2年	停止发布新版本，包括补丁版本。 **说明** 在EOS阶段，会停止用户答疑、问题解决、SLA保障赔付等服务。

> 说明
> - 各阶段时间点不会早于承诺时间发起，请您放心使用。具体发起时间会由通知渠道至少提前三个月告知给用户。
> - 如果遇到某个版本出现重大缺陷，会造成用户的重大损失（例如安全问题或数据准确性等），Hologres有权对此版本进行紧急修复，并提供保证兼容的新版本给对应用户升级。

版本号及其含义

Hologres使用两位编号的方案来指定产品的发布版本。版本的格式为 `Major.Minor` 。版本号中的Major和Minor含义详情如下。

- Major部分：表示Hologres功能的根本变化和增加。根据每个版本中更改的大小和规模，增加对应Major部分的数字。
- Minor部分：表示质量改进和对现有功能的修复。当许多质量改进被添加到版本中时，递增Minor部分的数字。

版本	含义	Hologres升级策略	示例
大版本	Hologres的新功能与缺陷修复。	升级Major部分。	`0.10.33` 升级为 `1.1.21` ，其中Major部分为 `0.10` 到 `1.1` 。
小版本	Hologres的兼容性功能与缺陷修复。	升级Minor部分。	`1.1.3` 升级为 `1.1.4` ，其中Minor部分为 `3` 增加到 `4` 。

版本对应关系和生命周期重要时间点

Hologres各版本对应的GA、EOS时间点如下表所示。

Hologres版本	GA	EOS
0.7	2020-04	2022-04
0.8	2020-10	2022-10
0.9	2021-01	2023-01
0.10	2021-05	2023-05
1.1	2021-10	待定

服务承诺与建议

- 兼容性
  - 阿里云Hologres不承诺大版本之间的兼容性。阿里云Hologres会尽最大努力保持产品在不同大版本间程序逻辑、SQL语法、接口等兼容性，并对可能出现的不兼容进行文档或售后服务告知。
  - 阿里云Hologres承诺同一个大版本中的小版本之间是兼容的，兼容包括SQL语法、接口和引擎框架等方面。兼容不包括程序的业务逻辑和处理数据等方面。因此，我们建议您在兼容性的基础上，将产品版本升级到同一个大版本的最新小版本。

- 缺陷

  如果您使用的不是最新版本，由于兼容性的服务承诺，阿里云Hologres建议您通过升级版本（包括大版本或者小版本）实现相关缺陷的修复。例如您作业使用的是0.10.3版本，而目前Hologres版本的最新版是1.1.5，则建议您通过升级引擎版本到1.1.5，来实现相关缺陷的修复。

  功能发布记录等公告渠道会通知您相关功能发布和缺陷修复情况，您也可以通过技术支持或提交工单进行咨询。

相关服务与通知

基于以上生命周期策略规则和版本对应关系，我们将会：

- 规划Hologres每一个版本的生命周期计划，并在产品交流与服务过程中向您传达。
- 在产品生命周期关键里程碑时间点到来之前的至少3个月，通过公司公告、站内信、邮件或短信等途经通知到购买产品的主账号。
- 在停止版本服务之前愿意帮助您一起评估作业运行风险并商讨解决方案。

16.视频专区

视频专区内容，请参见视频专区。

17.用户案例合集

用户案例合集内容，请参见Hologres最佳实践与案例合集。