

阿里云

云原生数据仓库AnalyticDB
MySQL版
最佳实践

文档版本：20220531

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 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.表结构设计	05
2.SQL开发规范	10
3.数据变更优化	13
4.数据导入性能优化	14
5.优化器	17
5.1. Persist plan and Query-Blocker	17
6.通过视图管控数据权限	20

1.表结构设计

本文介绍如何设计AnalyticDB MySQL版的表结构（包括选择表类型、分布键、分区键、主键和聚集索引键等），从而实现表性能的优化。

选择表类型

AnalyticDB MySQL版支持复制表和普通表两种类型。在选择表类型时，需要注意以下几点：

- 复制表会在集群的每个节点存储一份数据，因此建议复制表中的数据量不宜太大，每张复制表存储的数据不超过2万行。
- 普通表（即分区表）能够充分利用分布式系统的查询优势，提高查询效率。普通表可存储的数据量较大，通常可以存储千万条甚至上亿条数据。

选择分布键

如果业务明确有增量数据导入需求，创建普通表时可以同时指定分布键和分区键，来实现数据的增量同步。您可以在创建表时，通过 `DISTRIBUTED BY HASH(column_name,...)` 指定分布键，按照 `column_name` 的Hash值进行分区。更多详情，请参见[CREATE TABLE](#)。

● 语法

```
DISTRIBUTED BY HASH(column_name,...)
```

● 注意事项

- 尽可能选择值分布均匀的字段作为分布键，例如交易ID、设备ID、用户ID或者自增列作为分布键。

 **说明** 尽量不要选择日期、时间和时间戳类型的字段作为分布键。这些字段虽然是均匀分布的，但多数查询往往是限定了日期或者时间段，如：查询最近一天或者一个月的数据，可能会导致要查询的数据只存在于一个节点上，无法充分利用分布式数据库中所有节点的处理能力。日期、时间类型的字段建议作为二级分区来考虑，具体请参见[选择分区键](#)。

- 尽可能将需要Join的字段作为分布键，可以有效减少数据Shuffle。例如，需要按照顾客维度查看历史订单信息，可以选择 `customer_id` 作为分布键。
- 尽可能选择高频率出现查询条件的字段作为分布键，从而实现按分布键做裁剪。
- 每张表只能选取一个分布键，一个分布键可以包含一个字段或者多个字段，尽可能选取少的字段，使得分布键在各种复杂查询中更加通用。
- 若在创建表时，未指定分布键，系统会根据MySQL表是否含有主键进行如下处理：
 - 如果MySQL表含有主键，AnalyticDB MySQL版默认将主键作为分布键。
 - 如果MySQL表不含有主键，AnalyticDB MySQL版将添加一个 `__adb_auto_id__` 字段作为主键和分布键。
- AnalyticDB MySQL版 3.0弹性模式集群版（新版）8核和16核规格的集群，默认分片数为32。

- AnalyticDB MySQL版 3.0弹性模式集群版（新版）32核及以上规格的集群，默认分片数与EIU数量有关，对应关系如下：

EIU数量	分片数
EIU=1	32
2<=EIU<4	64
4<=EIU<8	128
8<=EIU<16	192
16<=EIU<32	256
32<=EIU<64	512
EIU>=64	1024

- AnalyticDB MySQL版 3.0预留模式集群版的默认分片数与节点组数量有关，对应关系如下：

节点组数	分片数
节点组数=1	16
2<=节点组数<4	32
4<=节点组数<8	64
8<=节点组数<16	128
16<=节点组数<32	256
32<=节点组数<64	512
节点组数>=64	1024

 **说明** 关于默认分片数的常见问题，请参见[常见问题](#)。

选择分区键

如果设置了分布键后，单个分片的数据量较大，您可以通过分区键在分片内进一步设置分区，以提高数据访问的性能。您可以在创建表时，通过 `PARTITION BY` 来定义二级分区，数据会将按照指定方式进行切分。更多详情，请参见[CREATE TABLE](#)。

● 语法

- 使用 `column_name` 的值做分区，语法如下：

```
PARTITION BY VALUE(column_name)
```

- 将 `column_name` 的值转换为 `%Y%m%d` 的日期格式（类似 20210101）做分区，语法如下：

```
PARTITION BY VALUE(DATE_FORMAT(column_name, '%Y%m%d'))
```

- 将 `column_name` 的值转换为 `%Y%m` 的日期格式（类似 202101）做分区，语法如下：

```
PARTITION BY VALUE (DATE_FORMAT(column_name, '%Y%m'))
```

- 将 `column_name` 的值转换为 `%Y` 的日期格式（类似 2021）做分区，语法如下：

```
PARTITION BY VALUE (DATE_FORMAT(column_name, '%Y'))
```

● 注意事项

- 当数据量较大时，二级分区的选择至关重要，如果数据量大的表中没有二级分区或者二级分区切分不合理，将严重影响AnalyticDB MySQL版集群性能。
- 目前切分粒度只支持年、月、日或原始值。切分粒度太大或太小都会影响查询性能和写入性能，甚至影响AnalyticDB MySQL版集群的稳定性。
- 一个二级分区的数据量在3亿~10亿之间视为合理切分。如果小于3亿，表示切分粒度太小，可以增大切分粒度（例如将切分粒度由日改为月）；如果大于10亿，表示切分粒度太大，可以减小切分粒度（例如将切分粒度由月改为日）。
- 尽量使二级分区维持静态状态，不建议频繁更新二级分区，例如如果有每天频繁更新多个历史二级分区场景，应考虑使用的二级分区字段是否合理。
- 您可以通过 `LIFECYCLE N` 关键字实现表生命周期管理，即对分区进行排序，超出 `N` 的分区会被过滤。

 **说明** 由于每张表中支持的最大分区数存在上限，因此分区表中的数据无法永久保留。关于分区数量限制的详情，请参见[约束和限制](#)。

选择主键

主键可以作为每一条记录的唯一标识。您可以在创建表时，通过 `PRIMARY KEY` 来定义主键。更多详情，请参见[CREATE TABLE](#)。

● 语法

```
PRIMARY KEY (column_name,...)
```

● 注意事项

- 只有定义过主键的表支持数据更新操作（包括DELETE和UPDATE）。
- AnalyticDB MySQL版的主键可以是单个字段或多个字段的组合。推荐使用数值类型的单个字段作为主键，以获得较好的表性能。
- 主键中必须包含分布键和分区键，建议将分布键和分区键放在组合主键的前部。

选择聚集索引键

聚集索引中键值的逻辑顺序决定了表中相应行的物理顺序。选择聚集索引键时需要注意如下几点：

- 每个表仅支持创建一个聚集索引。创建方式，请参见[CREATE TABLE](#)。
- 建议将查询一定会携带的字段作为聚集索引键。例如，每个学生在校教务系统中，只需查看自己的期末成绩，那么可以将学生的学号ID定义为聚集索引，来保证数据的局部性，提升数据查询性能。
- 由于聚集索引会进行全表排序，导致数据写入性能下降、CPU占用较高，因此一般不建议使用聚集索引。

示例

创建一张 `customer` 表，需要满足如下要求：

- 根据顾客的登录时间（即 `login_time` 列）进行数据分区，且需要将登录时间转换为 `%Y%m%d` 日期格式。
- 仅保留最近30个分区（即生命周期为30）的数据。
- 根据顾客ID（即 `customer_id` 列）进行数据分布。
- 将 `login_time, customer_id, phone_num` 设置为组合主键。

建表语句如下：

```
CREATE TABLE customer (
  customer_id bigint NOT NULL COMMENT '顾客ID',
  customer_name varchar NOT NULL COMMENT '顾客姓名',
  phone_num bigint NOT NULL COMMENT '电话',
  city_name varchar NOT NULL COMMENT '所属城市',
  sex int NOT NULL COMMENT '性别',
  id_number varchar NOT NULL COMMENT '身份证号码',
  home_address varchar NOT NULL COMMENT '家庭住址',
  office_address varchar NOT NULL COMMENT '办公地址',
  age int NOT NULL COMMENT '年龄',
  login_time timestamp NOT NULL COMMENT '登录时间',
  PRIMARY KEY (login_time, customer_id, phone_num)
)
DISTRIBUTED BY HASH(customer_id)
PARTITION BY VALUE (DATE_FORMAT(login_time, '%Y%m%d')) LIFECYCLE 30
COMMENT '客户信息表';
```

常见问题

- Q：进行二级分区后，如何查看表的所有二级分区及分区的统计信息？
A：执行以下SQL，可以查看表的所有二级分区及分区的统计信息：

```
SELECT partition_id, --分区名
       row_count, -- 分区总行数
       local_data_size, --分区本地存储所占空间大小
       index_size, -- 分区的索引大小
       pk_size, --分区的主键索引大小
       remote_data_size --分区的远端存储所占空间大小
FROM information_schema.kepler_partitions
WHERE schema_name = '$DB'
  AND table_name = '$TABLE'
  AND partition_id > 0;
```



注意 针对增量数据中还未触发compaction的分区不予展示。如果需要查看实时的所有二级分区列表，可以通过 `select distinct $partition_column from $db.$table;` 进行查询。

- Q：分片数跟什么因素有关系？用户是否可以自己调整？
A：分片数是创建集群时根据集群初始规格自动计算得出的。不支持用户调整分片数。
- Q：在集群变配时，是否会对分片数产生影响？
A：集群变配时，不会影响分片数。
- Q：AnalyticDB MySQL版是否支持修改分布键/分区键？

A：不支持。如果需要修改分布键/分区键，请参见[更改分区键/分布键](#)。

2.SQL开发规范

在云原生数据仓库AnalyticDB MySQL版中编写和优化SQL时，需要充分考虑其分布式特性，本文汇总了SQL编写和优化时的经验和技巧供您参考。

经验总结

编写和优化SQL的经验总结如下：

- SQL编写原则为追求简单

一般情况下，数据库性能会随SQL复杂度而下降。例如，单表查询（冗余设计）优于表关联查询。

- SQL优化核心方法是减少I/O

尽可能少的进行列扫描，返回最小数据量，减少I/O同时也减少内存开销。

- 分布式计算，本地计算&并行计算

大数据计算情况下，本地计算时充分利用分布式多计算资源的能力，避免数据跨节点。

- 高QPS，分区裁剪

业务系统要求高QPS、毫秒级RT时，表和SQL必须设计为分区裁剪模式。

SQL优化技巧

- 去掉不必要的列

云原生数据仓库AnalyticDB MySQL版是行列混存数据库，返回的列的数量直接影响性能，在编写SQL时一定要确认业务需要返回的列，不要直接使用星号（*）进行查询。

SQL示例。

- 错误SQL:

```
select * from tab1 where c1>100 and c1<1000;
```

- 正确SQL:

```
select col1, col2 from table_name where c1>100 and c1<1000;
```

- 索引和扫描

当SQL包含多个查询条件时，优先选择高筛选条件，其他条件可以通过扫描实现。

云原生数据仓库AnalyticDB MySQL版内部采用行列混存方式，通过单列高效过滤后，可直接通过内部记录指针扫描其他列值，减少其他列的索引查询开销。

云原生数据仓库AnalyticDB MySQL版支持在查询级别和集群级别针对特定字段关闭索引过滤，请参见[过滤条件不下推](#)。

示例

 说明 以下示例适用于3.14以下的内核版本。如果内核版本为3.14及以上，请使用 `filter_not_pushdown_columns` Hint。

o time条件通过内部扫描

在SQL命令中通过条件 `c1=3` 可快速查询到少量记录（假设10000），单独使用 `time>'2010-01-01 00:00:00'` 时返回的记录数又非常大，SQL示例如下：

```
select c1,c2 from tab1 where c1=3 and time >='2010-01-01 00:00:00';
```

此时可只通过c1进行索引，time通过内部扫描方式执行，查询更快，返回更多有效记录数。SQL示例如下：

```
/*+ no_index_columns=[tab1.time] */
select c1,c2 from tab1
where c1=3 and time>='2010-01-01 00:00:00';
```

Hint表示强制 `time>='2010-01-01 00:00:00'` 条件走扫描。

在上述SQL中，计算引擎首先检索列c1的索引，得出满足条件 `c1=3` 的行集合，然后读取每行所对应的time列数据。如果满足 `time>='2010-01-01 00:00:00'`，则将该行数据加入返回结果。

o 不等于条件通过内部扫描

例如在查询中使用 `c2<>100`，不通过索引扫描时，`c2<>100` 无法有效过滤掉无效记录。SQL示例如下：

```
select c1,c2 from tab1 where c1=3 and c2<>100;
```

增加 `no_index_columns` Hint，使不等于条件通过内部扫描执行，SQL示例如下：

```
/*+ no_index_columns=[tab1.c2] */
select c1,c2 from tab1 where c1=3 and c2<>100;
```

o like条件通过内部扫描

中缀或后缀查询，例如：`like '%abc'` 或 `like '%abc%'`。

增加 `no_index_columns` Hint，使like条件通过内部扫描执行，更加快速地查询有效记录，SQL示例如下：

```
/*+ no_index_columns=[tab1.c3] */
select c1,c2 from tab1 where c1=3 and c3 like '%abc%';
```

● 索引失效

索引失效时，SQL语句直接以扫描的方式进行查询，如果表记录数非常大，会导致查询缓慢。

以下情形容易引起索引失效：

- o 函数转换（列）。
- o 类型转换。
- o like条件，例如：`like '%abc%'`。

SQL示例：

- o 以下SQL中的函数转换导致索引失效。time为timestamp类型，存储时间 `2017-12-10 10:00:23`。

```
select c1,c2 from tab1 where substr(cast(time as varchar),1,10)='2017-10-12';
```

- 正确SQL示例：

```
select c1,c2 from tab1 where time>='2017-10-12 00:00:00' and time<='2017-10-12 23:59:59';
```

- 去掉不必要的 `is not null` 过滤条件

SQL示例：

- 错误SQL示例：

```
Select c1, c2 from tab1 where c1>100 and c1<1000 and c1 is not null;
```

- 删除上述SQL中的多余条件 `and c1 is not null`，优化后SQL如下：

```
Select c1,c2 from tab1 where c1>100 and c1<1000;
```

- 多表关联

不同多表关联场景下，SQL优化原则不同，其中：

- 普通表JOIN普通表，尽量包含分区列条件。如果不包含，则尽量通过WHERE条件过滤掉多余的数据。
- 维度表JOIN普通表，没有限制。

多表关联查询WHERE条件中，需要明确写明每一个表的过滤条件。通常在传统数据库中，都是通过索引字段关联来快速检索数据。例如：

```
Select count(*)
from customer_table C join
order_table O on C.customer_id= O.customer_id
where O.order_time between '2018-07-20 10:00:11'
and '2018-09-30 10:00:11'
and O.order_amount=100;
```

在明确t2与t1表都有同样的time和type过滤条件情况下，建议修改为如下SQL：

```
Select count(*)
from t1 join t2 on t1.id=t2.id
where t1.time between '2017-12-10 00:00:00' and '2017-12-10 23:59:59'
and t1.type= 100
and t2.time between '2017-12-10 00:00:00' and '2017-12-10 23:59:59'
and t2.type=100
```

3.数据变更优化

本文介绍了更新、删除和导入的方法。

向表中写入数据时，可以通过批量打包方式INSERT和REPLACE INTO提高数据写入性能。建议如下：

- 通过每条INSERT或者REPLACE语句写入的数据行数大于1000行，但写入的总数据量不宜太大，不超过16MB。
- 通过批量打包方式写入数据时，单个批次的写入延迟相对会高一些。
- 写入报错时，需要做重试确保数据被写入，重试导致的数据重复可以通过表的主键来消除。

更新数据

AnalyticDB MySQL版提供多种数据更新方式，建议如下：

- 数据更新频率高、基于主键的行级覆盖更新且应用可以补齐所有列，可通过REPLACE INTO批量更新数据。
- 数据更新频率低、基于主键更新，可通过REPLACE INTO或者UPDATE单条更新数据。
- 数据更新频率低、基于任意条件的更新，可通过UPDATE更新数据。

删除数据、分区、表

AnalyticDB MySQL版有多种删除数据的方式，建议如下：

- 数据删除频率低、基于主键为条件的删除，可通过 `DELETE FROM WHERE PK='xxx'` 删除数据。
- 数据删除频率低、基于任意条件的删除，可通过DELETE删除数据。
- 通过 `TRUNCATE TABLE db_name.table_name PARTITION partition_name` 删除指定二级分区。
- 通过 `TRUNCATE TABLE db_name.table_name` 删除指定表（包括所有二级分区）数据。

批量和实时导入数据

- 批量导入：适用于大数据量导入的场景，导入过程中可以查询旧数据，导入操作完成后一键切换新数据。导入失败时，支持回滚新数据，不影响查询旧数据。

例如，从MaxCompute、OSS中导入数据到AnalyticDB MySQL版时，推荐使用INSERT OVERWRITE INTO SELECT批量导入方式，详情请参见[通过外表将OSS数据导入至AnalyticDB MySQL](#)。

 **说明** 批量导入单个表时，导入任务会在系统中串行执行；批量导入多个表时，导入任务会在系统中并行执行，默认并行执行2个任务。

例如将MaxCompute中的数据导入AnalyticDB MySQL版时，MaxCompute中同一个表的不同分区中的数据会在系统中排队串行导入。

- 实时导入：适用于小数据量导入的场景，例如百万级别的表数据。例如，从RDS for MySQL、ECS自建MySQL数据导入AnalyticDB MySQL版时，推荐使用INSERT INTO，详情请参见[通过外表将RDS MySQL数据导入至AnalyticDB MySQL](#)。

 **说明** 导入数据到AnalyticDB MySQL版时，导入任务使用AnalyticDB MySQL版中的资源，如果查询数据期间需要导入数据，建议在QPS（Query Per Second，每秒查询率）较低时进行导入操作。

4.数据导入性能优化

云原生数据仓库AnalyticDB MySQL版提供的多种数据导入方法，满足不同场景下的数据导入需求。然而数据导入性能依然受各种各样的因素影响，如表的建模不合理导致长尾、导入配置低无法有效利用资源等。本文介绍不同场景下的数据导入调优方法。

通用外表导入数据调优

检查分布键

分布键决定着数据导入的一级分区，每个表在导入时以一级分区为粒度并发导入。当数据分布不均匀时，导入数据较多的一级分区将成为长尾节点，影响整个导入任务的性能，因此要求导入时数据均匀分布。如何选择分布键，请参见[选择分布键](#)。

判断分布键合理性：

- 导入前，根据导入数据所选分布键的业务意义判断是否合理。以表Lineitem为例，当选择l_discount列为分布键，订单折扣值区分度很低，仅有11个不同值，l_discount值相同的数据会分布到同一分区，造成严重倾斜，导入会有长尾，影响性能。选择l_orderkey列则更为合适，订单ID互不相同，数据分布相对较为均匀。
- 导入后，数据建模诊断中如有分布字段倾斜，则说明选择的分布键不均匀。如何查看分布键诊断信息，请参见[分布字段合理性诊断](#)。

检查分区键

`INSERT OVERWRITE INTO SELECT` 导入数据的基本特性为分区覆盖，即导入的二级分区会覆盖原表的同名二级分区。每个一级分区内的数据会再按二级分区定义导入各个二级分区。导入时需要避免一次性导入过多二级分区，多个二级分区同时导入可能引入外排序过程，影响导入性能。如何选择分区键，请参见[选择分区键](#)。

判断分区键合理性：

- 导入前，根据业务数据需求及数据分布判断分区键是否合理。如Lineitem表按l_shipdate列做二级分区，数据范围横跨7年，按年做分区有7个分区，按日做分区有2000多个分区，单分区约3000万条记录，选择按月或者按年做分区则更合适。
- 导入后，数据建模诊断中如有不合理的二级分区，则选择的分区键不合适。如何查看分区键诊断信息，请参见[分区字段合理性诊断](#)。

检查索引

AnalyticDB MySQL建表时默认全列索引，而构建宽表的全列索引会消耗部分资源。导入数据到宽表时，建议使用主键索引。主键索引用于去重，主键列数过多影响去重性能。如何选择主键索引，请参见[选择主键](#)。

判断索引合理性：

- 离线导入场景通常已经通过离线计算进行去重，无需指定主键索引。
- 在[监控信息 > 表信息统计](#)页签，查看表数据量、索引数据量和主键索引数据量。当索引数据量超过表数据量时，需要检查表中是否有较长的字符串列，这种索引列不仅构建耗时，还占用存储空间，可以删除索引，请参见[删除索引](#)。

 说明 主键索引无法删除。需要重建表。

增加Hint加速导入

在导入任务前增加Hint（`direct_batch_load=true`）可以加速导入任务。

说明

该Hint仅数仓版（3.0）弹性模式集群3.1.5版本支持，若使用后导入性能无明显提升，请提交工单。

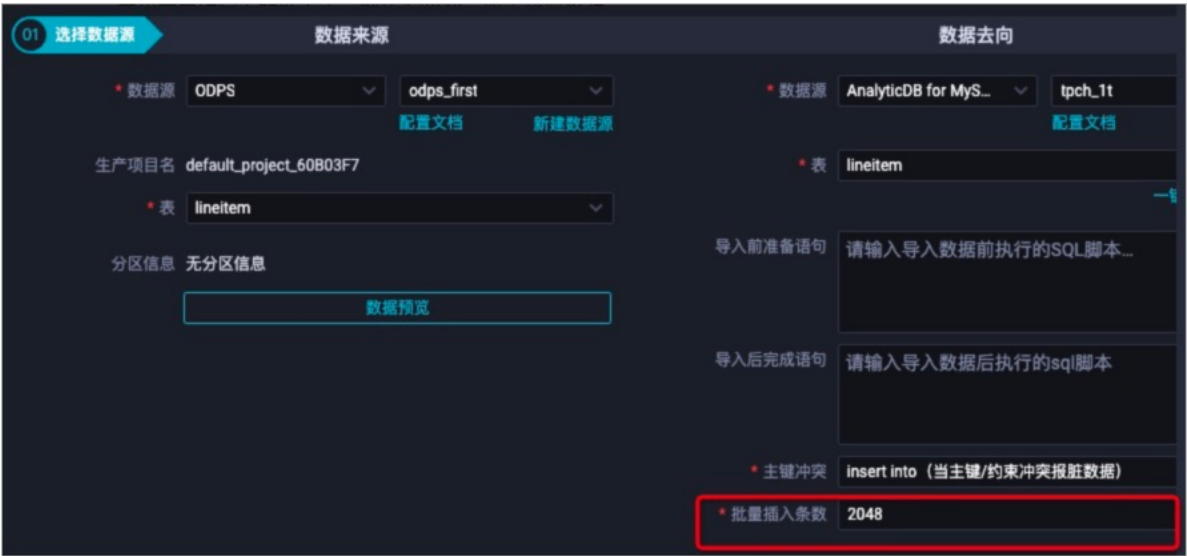
示例如下：

```
submit job /* direct_batch_load=true*/insert overwrite adb_table
select * from adb_external_table;
```

通过DataWorks导入数据调优

优化任务配置

- 优化批量插入条数
表示单次导入的批大小，默认为2048，一般不建议修改。



如果单条数据量过大达到数百KB，如高达512 KB，则建议修改此配置为16，保证单次导入量不超过8 MB，防止占用过多前端节点内存。

- 优化通道控制
 - 数据同步性能与任务期望最大并发数配置项大小成正比，建议尽可能增加任务期望最大并发数。

注意

任务期望最大并发数越高，占用DataWorks资源会越多，请合理选择。

- 建议打开分布式处理能力，以取得更好的同步性能。



常见问题及解决方法

- 当客户端导入压力不足时，会导致集群CPU使用率、磁盘IO使用率及写入响应时间处于较低水位。数据库服务器端虽然能够及时消费客户端发送的数据，但由于总发送量较小，导致写入TPS不满足预期。

解决方法：调大单次导入的批量插入条数及增加任务期望最大并发数，数据导入性能会随着导入压力的增加而线性增加。

- 当导入的目标表存在数据倾斜时，集群部分节点负载过高，影响导入性能。此时，集群CPU使用率、磁盘IO使用率处于较低水位，但写入响应时间较高，同时您可以在[诊断优化 > 数据建模诊断](#)页面的倾斜诊断表中发现目标表。

解决方法：重新设计表结构后再导入数据，详情请参见[表结构设计](#)。

通过JDBC使用程序导入数据调优

客户端优化

- 应用端攒批，多条批量导入
 - 在通过JDBC使用程序导入数据过程中，为减少网络和链路上的开销，建议攒批导入。无特殊要求，请避免单条导入。
 - 批量导入条数建议为2048条。如果单条数据量过大达到数百KB，建议攒批数据大小不超过8 MB，可通过8 MB/单条数据量得到攒批条数。否则单批过大容易占用过多前端节点内存，影响导入性能。
- 应用端并发配置
 - 应用端导入数据时，建议多个并发同时导入数据。单进程无法完全利用系统资源，且一般客户端需要处理数据、攒批等操作，难以跟上数据库的导入速度，通过多并发导入可以加快导入速度。
 - 导入并发受攒批、数据源、客户端机器负载等影响，没有最合适的数值，建议通过测试逐步计算合适的并发能力。如导入不达预期，请翻倍加大并发，导入速度下降再逐步降低并发，寻找最合适的并发数。

常见问题及解决方法

当通过程序导入数据到AnalyticDB MySQL性能不佳时，首先排查客户端性能是否存在瓶颈。

- 保证数据源的数据生产速度足够大，如果数据源来自其他系统或文件，排查客户端是否有输出瓶颈。
- 保证数据处理速度，排查数据生产消费是否同步，保证有足够的等待数据导入AnalyticDB MySQL。
- 保证客户端机器负载，检查CPU使用率或磁盘IO使用率等系统资源是否充足。

5. 优化器

5.1. Persist plan and Query-Blocker

Persist plan功能支持在SQL参数化级别使Hints持久化。Query-Blocker功能支持参数化SQL的拦截，实现SQL防火墙的功能。本文适用于AnalyticDB MySQL版3.1.4及以上版本的实例。

功能介绍

在集群级别开启高级优化特性，可能影响范围较大。通过Persist plan设置指定SQL的Hints，可以使Hints只对相同Pattern（即参数化后的SQL）的SQL生效。Persist plan规则支持通过系统表查看和删除。

? 说明

- 通过Persist plan设置指定SQL的Hints时，不会执行当前SQL。
- 相同Pattern指用 ? 替代SQL里的常量，例如select、where中的常量条件，limit m、n等。

语法结构

- 计算SQL参数化后的Pattern和sign: `PARAMETERIZE $sql`
- 针对指定SQL Pattern，添加Hints: `/*+hints*/ PERSIST_PLAN $sql`
- 针对指定SQL Pattern，删除Hints: `DELETE_PLAN $sql` 或 `DELETE_PLAN_BY_SIGN $sign`
- 查询指定SQL是否配置过Hints:
 - `PERSIST_PLAN_CHECK $sql`
 - `SELECT * FROM INFORMATION_SCHEMA.KEPLER_PERSIST_PLAN_SUMMARY`
- 查询配置过Hints的所有SQL Pattern: `SELECT * FROM INFORMATION_SCHEMA.KEPLER_PERSIST_PLAN_SUMMARY`

典型应用：拦截SQL Pattern

使用 `/*+query_blocker=true*/ persist_plan + SQL`，针对指定Pattern拦截同类型的Bad SQL。

示例

- 计算SQL参数化后的pattern&sign

针对以下SQL计算其对应的pattern和sign值，示例如下：

```
PARAMETERIZE
SELECT
  t1.c1
FROM
  t1
  INNER JOIN t2 ON t1.c1 = t2.c1
WHERE
  t1.c2 < 9999;
```

执行命令的结果如下：

sign	sql
2506ed2c1f53ea59a1ef996a98a50411	SELECT t1.c1 FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1 WHERE t1.c2 < ?

● 针对指定SQL Pattern，添加Hints

为相同Pattern的SQL，执行 `nested loop join` 策略，示例如下：

```
/*+nested_loop_join=true*/
PERSIST_PLAN
SELECT t1.c1
FROM t1
      INNER JOIN t2 ON t1.c1 = t2.c1
WHERE t1.c2 < 9999;
```

● 查询指定SQL是否配置过Hints

```
PERSIST_PLAN_CHECK
SELECT t1.c1
FROM t1
      INNER JOIN t2 ON t1.c1 = t2.c1
WHERE t1.c2 < 9999;
```

执行命令的结果如下：

sign	Hints	hitApplied	sql
2506ed2c1f53ea59a1ef996a98a50411	nested_loop_join=true	12	SELECT t1.c1 FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1 WHERE t1.c2 < ?

② 说明

- `PERSIST_PLAN_CHECK $sql` 命令在DMS数据管理服务系统中执行时，执行命令成功，但不会显示上表中的结果。若您想看到具体返回信息，请登录MySQL客户端执行。
- 返回结果中，hitApplied字段表示创建该规则后被应用的次数，若更新该规则，hitApplied字段会清零，重新开始计数。

● 查询配置过Hints的所有SQL Pattern

```
SELECT * FROM INFORMATION_SCHEMA.KEPLER_PERSIST_PLAN_SUMMARY;
```

● 针对指定SQL Pattern，删除Hints

AnalyticDB MySQL版支持删除指定SQL Pattern的Hints，您可以选择以下任意一种实现方式：

- 使用 `DELETE_PLAN $sql` 命令，删除Hints。

```
DELETE_PLAN
SELECT t1.c1
FROM t1
      INNER JOIN t2 ON t1.c1 = t2.c1
WHERE t1.c2 < 9999;
```

- 使用 `DELETE_PLAN_BY_SIGN` 命令，根据sign值删除Hints。

```
DELETE_PLAN_BY_SIGN 2506ed2c1f53ea59a1ef996a98a50411;
```

② 说明 查询SQL Pattern的sign值有以下两种方法：

- 通过 `PARAMETERIZE $sql`，在返回结果中查看sign值。
- 通过 `SELECT * FROM INFORMATION_SCHEMA.KEPLER_PERSIST_PLAN_SUMMARY`，在返回结果中查看sign值。

6.通过视图管控数据权限

本文介绍如何通过视图实现给不同的账号授予查询同一表中不同数据的权限，来提供数据的安全性。

场景需求

已创建了一张包含不同省份客户的表 `customer`，建表语句如下：

```
Create Table `customer` (  
  `id` bigint AUTO_INCREMENT,  
  `province_id` bigint NOT NULL,  
  `user_info` varchar,  
  primary key (`id`)  
) DISTRIBUTE BY HASH(`id`);
```

往表 `customer` 中插入测试数据，语句如下：

```
INSERT INTO customer(province_id,user_info) VALUES (1,'Tom'), (1,'Jerry'), (2,'Jerry'), (3,'Mark');
```

通过SELECT语句可查询到表中当前数据如下：

id	province_id	user_info
1369417242420617216	1	Tom
1369417242424811520	1	Jerry
1369417242424811522	3	Mark
1369417242424811521	2	Jerry

要求 `user1` 仅能查看省份1（即 `province_id=1`）的数据，而 `user2` 仅能查看省份2（即 `province_id=2`）的数据。

实现方式

您可以通过如下操作步骤实现上述场景需求：

1. 为省份1和省份2分别创建一个视图 `v1` 和 `v2`，语句如下：

- 查询省份1数据的视图创建语句。

```
CREATE SQL SECURITY DEFINER VIEW v1 AS SELECT * FROM customer WHERE province_id=1;
```

- 查询省份2数据的视图创建语句。

```
CREATE SQL SECURITY DEFINER VIEW v2 AS SELECT * FROM customer WHERE province_id=2;
```

 **说明** 创建视图语句中各参数的详细介绍，请参见[CREATE VIEW](#)。

2. 视图创建成功后，通过GRANT语句将对应的数据查询权限授予给目标用户，语句如下：

 **说明** 创建账号的方法，请参见[CREATE USER](#)。

- 授予 `user1` 通过 `v1` 视图查询省份1数据的权限：

```
GRANT SELECT ON v1 TO user1;
```

- 授予 `user2` 通过 `v2` 视图查询省份2数据的权限：

```
GRANT SELECT ON v2 TO user2;
```

查询结果验证

- 使用 `user1` 账号连接AnalyticDB MySQL版集群的 `adb_demo` 数据库后，`user1` 仅能查询到视图 `v1` 数据，查询语句如下：

```
SELECT * FROM v1;
```

查询结果如下：

ID	PROVINCE_ID	USER_INFO
1369417242420617216	1	Tom
1369417242424811520	1	Jerry

若 `user1` 想要查询视图 `v2` 中的数据，则会出现如下错误：

```
ERROR 1815 (HY000): [9001, 2021083114191719216818804803453965343] : Access Denied
```

- 使用 `user2` 账号连接AnalyticDB MySQL版集群的 `adb_demo` 数据库后，`user2` 仅能查询到视图 `v2` 数据，查询语句如下：

```
SELECT * FROM v2;
```

查询结果如下：

ID	PROVINCE_ID	USER_INFO
1369417242424811521	2	Jerry

若 `user2` 想要查询视图 `v1` 中的数据，也会出现如下错误：

```
ERROR 1815 (HY000): [9001, 2021083114191719216818804803453965343] : Access Denied
```