阿里云
MaxCompute

规范

文档版本：20190521

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 数仓建设指南

1.1 数据模型架构规范

数据层次的划分

- ODS：Operational Data Store，操作数据层，在结构上其与源系统的增量或者全量数据基本保持一致。它相当于DW数据的一个数据准备区，同时又承担着基础数据的记录以及历史变化。其主要作用是把基础数据引入到MaxCompute。
- CDM：Common Data Model，公共维度模型层，又细分为DWD和DWS。它的主要作用是完成数据加工与整合，建立一致性的维度，构建可复用的面向分析和统计的明细事实表，以及汇总公共粒度的指标。
  - DWD：Data Warehouse Detail，明细数据层。
  - DWS：Data Warehouse Summary，汇总数据层。
- ADS：Application Data Service，应用数据层。

具体仓库的分层情况需要结合业务场景、数据场景、系统场景进行综合考虑。

数据分类架构

该数据分类架构在ODS层分为三部分：数据准备区，离线数据和准实时数据区。在进入到CDM层后，由以下几部分组成：

- 公共维度层：基于维度建模理念思想，建立整个企业的一致性维度。
- 明细粒度事实层：以业务过程为建模驱动，基于每个具体业务过程的特点，构建最细粒度的明细层事实表。您可以结合企业的数据使用特点，将明细事实表的某些重要维度属性字段做适当的冗余，即宽表化处理。
- 公共汇总粒度事实层：以分析的主题对象为建模驱动，基于上层的应用和产品的指标需求，构建公共粒度的汇总指标事实表，以宽表化手段来物理化模型。

数据处理流程架构

数据划分及命名空间约定

请根据业务划分数据并约定命名，建议针对业务名称结合数据层次约定相关命名的英文缩写，这样可以给后续数据开发过程中，对项目空间、表、字段等命名做为重要参照。

- 按业务划分：命名时按主要的业务划分，以指导物理模型的划分原则、命名原则及使用的ODS project。例如，按业务定义英文缩写，阿里的“淘宝”英文缩写可以定义为'tb'。
- 按数据域划分：命名时按照CDM层的数据进行数据域划分，以便有效地对数据进行管理，以及指导数据表的命名。例如，“交易”数据的英文缩写可定义为'trd'。
- 按业务过程划分：当一个数据域由多个业务过程组成时，命名时可以按业务流程划分。业务过程是从数据分析角度看客观存在的或者抽象的业务行为动作。例如，交易数据域中的“退款”这个业务过程的英文缩写可约定命名为'rfd_ent'。

数据模型

模型是对现实事物的反映和抽象，能帮助我们更好地了解客观世界。数据模型定义了数据之间关系和结构，使得我们可以有规律地获取想要的数据。例如，在一个超市里，商品的布局都有特定的规范，商品摆放的位置是按照消费者的购买习惯以及人流走向进行摆放的。

- 数据模型的作用

  数据模型是在业务需求分析之后，数据仓库工作开始时的第一步。良好的数据模型可以帮助我们更好地存储数据，更有效率地获取数据，保证数据间的一致性。
- 模型设计的基本原则
  - 高内聚和低耦合

    一个逻辑和物理模型由哪些记录和字段组成，应该遵循最基本的软件设计方法论中的高内聚和低耦合原则。主要从数据业务特性和访问特性两个角度来考虑：将业务相近或者相关的数

据、粒度相同数据设计为一个逻辑或者物理模型；将高概率同时访问的数据放一起，将低概率同时访问的数据分开存储。

- 核心模型与扩展模型分离

  建立核心模型与扩展模型体系，核心模型包括的字段支持常用核心的业务，扩展模型包括的字段支持个性化或是少量应用的需要。在必须让核心模型与扩展模型做关联时，不能让扩展字段过度侵入核心模型，以免破坏了核心模型的架构简洁性与可维护性。

- 公共处理逻辑下沉及单一

  底层公用的处理逻辑应该在数据调度依赖的底层进行封装与实现，不要让公用的处理逻辑暴露给应用层实现，不要让公共逻辑在多处同时存在。

- 成本与性能平衡

  适当的数据冗余可换取查询和刷新性能，不宜过度冗余与数据复制。

- 数据可回滚

  处理逻辑不变，在不同时间多次运行数据的结果需确定不变。

- 一致性

  相同的字段在不同表中的字段名必须相同。

- 命名清晰可理解

  表命名规范需清晰、一致，表命名需易于下游的理解和使用。

说明:

- 一个模型无法满足所有的需求。
- 需合理选择数据模型的建模方式。
- 通常，设计顺序依次为：概念模型->逻辑模型->物理模型。

1.2 公共规范

本文为您介绍MaxCompute数仓建设的公共规范。

层次调用约定

应用层应优先调用DW公共层数据，必须存在中间层数据，不允许应用层跨过中间层从ODS层重复加工数据。一方面，中间层团队应该积极了解应用层数据的建设需求，将公用的数据沉淀到公共层，为其他团队提供数据服务；另一方面，应用层团队也需积极配合中间层团队进行持续的数据公共建设的改造。必须避免出现过度的ODS层引用、不合理的数据复制以及子集合冗余。

- ODS层数据不能被应用层任务引用，中间层不能有沉淀的ODS层数据，必须通过CDM层的视图访问。CDM层视图必须使用调度程序进行封装，保持视图的可维护性与可管理性。
- CDM层任务的深度不宜过大（建议不超过10层）。
- 原则上一个计算刷新任务只允许一个输出表，特殊情况除外。
- 如果多个任务刷新输出一个表（不同任务插入不同的分区），DataWorks上需要建立一个依赖多个刷新任务的虚拟任务，通常下游应该依赖此虚拟任务。
- CDM汇总层应优先调用CDM明细层。在调用可累加类指标计算时，CDM汇总层尽量优先调用已经产出的粗粒度汇总层，以避免大量汇总都直接从海量的明细数据层计算。
- CDM明细层累计快照事实表优先调用CDM事务型事实表，以保持数据的一致性产出。
- 避免应用层过度引用和依赖CDM层明细数据，需有针对性地建设好CDM公共汇总层。

MaxCompute Project分配

按实际的需求分配不同的ODS和CDM项目。一个ODS层项目对应一个CDM项目。例如：

- ODS层项目，需按业务部门的粒度建立。
- CDM层项目，需按业务部门的粒度建立。
- ADS层项目，需按应用的粒度建立。

一个项目的划分结构如下图所示。

项目命名规范

- ODS层项目以ods为后缀，如tbods。
- 中间层项目以cdm为后缀，如tbcdm。
- 应用层项目中，数据报表、数据分析等应用以bi为后缀，如tbbi；而数据产品等应用以app为后缀，如sycmapp。

数据类型规范

ODS层的数据类型应基于源系统数据类型转换。如源数据为MySQL时的转换规则如下：

MySQL数据类型	MaxCompute数据类型
TINYINT	TINYINT
SMALLINT/MEDIUMINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE

MySQL数据类型	MaxCompute数据类型
DECIMAL	DECIMAL
CHAR/VARCHAR	VARCHAR
LONGTEXT/TEXT	STRING
DATE/TIMESTAMP/TIME/YEAR	STRING
DATETIME	DATETIME

CDM数据公共层如果是引用ODS层数据，则默认使用ODS层字段的数据类型。其衍生加工数据字段按以下标准执行：

- 金额类及其它小数点数据使用DOUBLE类型。
- 字符类数据使用STRING类型。
- ID类和整形数值使用BIGINT类型。
- 时间类型数据使用STRING类型（如果有特殊的格式强制要求，可以选择性使用DATETIME类型）。
- 状态使用STRING类型。

公共字段定义规范

数据统计日期的分区字段按以下标准：

- 按天分区：ds(YYYYMMDD)
- 按小时分区：hh(00-23)
- 按分钟：mi (00-59)
- is_{业务}：表示布尔型数据字段。”Y”,”N”表示，不允许出现空值域。
- 原则上不需要冗余分区字段。

数据冗余

- 相关数据冗余

  一个表做宽表冗余维度属性时，应该遵循以下建议准则：

  - 冗余字段与表中其它字段高频率（大于3个下游应用SQL）同时访问。
  - 冗余字段的引入不应造成其本身的刷新完成时间产生过多后延。
  - 公共层数据不允许字段重复率大于60%的相同粒度数据表冗余，可以选择在原表基础上拓宽或者在下游应用中通过JOIN方式实现。

- 子集合冗余

  当需要从一个集合中冗余一部分记录作为另外一张表存在时，可以优先考虑子分区方式，但多级子分区不应超过（5级）。只有以下情况才考虑冗余：

  - 子类型表有较多（大于10）个字段，而父类型表并不存在。
  - 子集合的过滤条件会被多次（大于5次）应用。

数据拆分

数据的水平和垂直拆分是按照访问热度分布和数据表”非空或者0值”的数据值在行列二维空间上分布情况进行划分的。

- 在物理上划分核心模型和扩展模型，将其字段进行垂直划分。
- 将访问相关度较高的列在一个表存储，将访问相关度较低的字段分开存储。
- 将经常用到的where条件按记录行进行水平切分或者冗余；水平切分可以考虑二级分区手段，以避免多余的数据复制与冗余。
- 将出现大量空值和零值的统计汇总表，依据其空值和零值分布状况可以做适当的水平和垂直切分，以减少存储和下游的扫描数据量。

空值处理原则

- 汇总类指标的空值：空值处理，填充为零，当前MaxCompute基于列存储的压缩技术不会由于填充大量空值导致存储成本上升。
- 维度属性值为空：在汇总到对应维度上时，对于无法对应的统计事实，记录行会填充为-99（未知），对应维表会出现一条-99（未知）的记录。

1.3 ODS层设计规范

本文为您介绍ODS层设计规范。

数据同步及处理规范

- 数据同步方式的选择

  以下是数据同步的相关规范，基本规范以需求形式落地到DataWorks的数据集成实现，规范落地情况需要依赖工具的推进节奏：

  - 一个系统的源系统表只允许同步一次到MaxCompute。
  - 数据集成只做离线全量数据同步，增量数据同步可使用数据传输服务DTS的方式实现。

- 数据加载与处理
  - 数据集成同步时全量数据会直接进入全量表的当日分区。
  - DTS同步数据主要有两步：
    - 全量初始化。对于同步的每个表，全量初始化的数据都会独立存储在MaxCompute中的全量基线表中，这个表名的默认格式为：源表名_base。
    - 增量数据同步。增量数据数据实时同步到 MaxCompute中。并存储在增量日志表中，每个同步表对应一个增量日志表。在增量数据同步时，会使用合并多条记录到一个文件的方式写入到 MaxCompute中。增量日志表在MaxCompute中存储的表名的默认格式为：源表名_log。
  - DTS全量数据合并。根据同步到MaxCompute中的全量基线表和增量日志数据得到某个时刻表的全量数据，DTS 提供通过MaxCompute SQL实现全量数据合并的能力。通过DataWorks可以配置一个全量数据merge节点，当全量数据merge完成后，可自动调度起后续的计算分析节点。同时可以配置调度周期，进行周期性数据的离线分析。
  - 所有ODS层的表都以统计日期及时间分区表方式存储，数据成本由存储管理和策略部分控制和管理。
  - 数据集成会自适应处理源系统的字段变更，具体策略是：如果源系统的字段在MaxCompute目标表不存在，由数据集成自动添加字段；如果目标表的字段在源系统中不存在，数据集成自动填充NULL。

命名规范

- 表命名规范

  表命名规则：{层次}{源系统表名}{保留位/delta与否}
  - 增量数据：{project_name}.s{源系统表名}delta
  - 全量数据: {project_name}.s_{源系统表名}
  - ODS ETL过程的临时表：{project_name}.tmp{临时表所在过程的输出表}{从0开始的序号}
  - 按小时的增量表：{project_name}.s{源系统表名}{delta}_{hh}
  - 按小时的全量表：{project_name}.s{源系统表名}{hh}
  - 当从不同源系统同步到一个project下造成表命名冲突时，后进来的表的命名需加上源系统的dbname以解决冲突。
- 字段命名规范
  - 字段默认使用源系统的字段名称。
  - 字段名与MaxCompute关键字冲突时的处理规则：在源字段名后加一个”col”，即源字段名col。

- 同步任务命名规范
  - 任务名：{源系统表名}[delta]

    

    说明:
    同一project下异库同名表的任务名：{源系统表名}{tddl的appname}[_delta]。
  - 任务的输出名称需与数据存储及生命周期管理规范保持一致，即为输出表的名称。

数据存储及生命周期管理规范

数据表类型	存储方式	最长存储保留策略
ODS流水型全量表	按天分区	· 不可再生情况下永久保存。 · 日志（非常大:如一天数据大于100G）数据保留24个月。 · 自主设置是否保留历史月初数据。 · 自主设置是否保留特殊日期数据。
ODS镜像型全量表	按天分区	· 重要的业务表及需要保留历史的表视情况保存。 · ODS全量表的默认生命周期管理为保留2天，并采用`ds=max_pt(tablename)`方式进行数据访问。
ODS增量表	按天分区	有对应的全量表则最多保留最近14天分区数据。无全量表的增量数据需永久保留。
ODS ETL过程临时表	按天分区	最多保留最近7天分区
BDSync非去重数据	按天分区	由应用通过中间层做保留历史处理，默认ODS层不保留历史。

数据质量规范

- 每个ODS全量表必须配置唯一性字段标识。
- 每个ODS全量表必须做分区空数据监控。
- 建议对重要表的重要枚举类型字段做枚举值变化及枚举值分布监控。

- 建议对ODS表的数据量及数据记录数设置上周同比无变化监控，用于监控源系统是否下线或者已迁移。
- 只有有监控要求的表才创建数据质量管控层，应由DataWorks的数据质量配置完成。
- 每个ODS层全表都必须要有注释。

1.4 CDM公共维度层设计规范

本文为您介绍CDM公共维度层设计规范。

设计准则

- 一致性维度规范

  公共层的维度表中相同维度属性在不同物理表中的字段名称、数据类型、数据内容必须保持一致。除了以下特例：

  - 在不同的实际物理表中，如果由于维度角色的差异，需要使用其他的名称，其名称也必须是规范的维度属性的别名。比如：定义一个标准的会员ID时，如果在一个表中，分别要表示买家ID，卖家ID，那么设计规范阶段就预先对会员ID分别定义买家ID和卖家ID。
  - 如果由于历史原因，在暂时不一致的情况下，必须在规范的维度定义一个标准维度属性，不同的物理名也必须是来自标准维度属性的别名。

- 维度的组合与拆分
  - 组合原则
    - 将维度所描述业务相关性强的字段在一个物理维表实现，相关性一般指：经常需要一起查询、报表展现，比如商品基本属性和所属品牌；两个维度属性间是否存在天然的关系等。
    - 无相关性的维度可以适当考虑杂项维度，比如交易，可以构建一个交易杂项维度收集交易的特殊标记属性、业务分类等信息。也可以将杂项维度退化在事实表中处理，不过容易造成事实表相对庞大，加工处理较为复杂。
    - 所谓的行为维度是经过汇总计算的指标，在下游的应用使用时将其当维度处理。如果有需要，度量指标可以作为行为维度冗余到维度表中。
  - 拆分与冗余
    - 对于维度属性过多，涉及源较多的维度表，可以做适当拆分。
      - 比如会员表，建议拆分为核心表和扩展表。核心表相对字段较少，刷新产出时间较早，优先使用。扩展表字段较多，且可以冗余核心表部分字段，刷新产出时间较晚，适合数据分析人员使用。
      - 根据维度属性的业务不相关性，将相关度不大的维度属性拆分为多个物理表存储。
    - 数据记录数较大的维度表，可以适当冗余一些子集合，以减少下游扫描数据量：
      - 比如商品表，可以根据当天是否有行为，产出一个有活跃行为的相关维表，以减少应用的数据扫描量。
      - 可根据所属业务扫描数据范围大小的不同，进行适当子集合冗余。

表命名规范

命名规则：{project_name}.dim{业务/pub}{维度定义}[_{自定义命名标签}]，所谓的pub是类似与具体业务无关，各个业务部都可以共用，例如时间维度。

数据存储及生命周期管理规范

CDM公共维度层的表的类型为维度表，存储方式为按天分区。

最长存储保留策略

模型设计者根据自身业务需求设置表的生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具体计算方式如下：

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。
- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时， 建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。

- 当3个月内的最大访问跨度小于或等于180天时，建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

1.5 CDM明细层设计规范

本文为您介绍CDM明细层设计规范。

表命名规范

命名规则：{project_name}.dwd{业务缩写/pub}{数据域缩写}{业务过程缩写}[{自定义表命名标签缩写}]{刷新周期标识}{单分区增量全量标识}。

说明:

- pub表示数据包括多个业务的数据。
- 单分区增量全量标识：i表示增量，f表示全量。

数据存储及生命周期管理规范

CDM明细层的表的类型为事实表，存储方式为按天分区。

事务型事实表一般永久保存。周期性快照事实表根据业务需求设置生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具体计算方式如下：

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。
- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时，建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。
- 当3个月内的最大访问跨度小于或等于180天时，建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

事务型事实表设计准则

事务型事实表主要用于分析行为与追踪事件。事务事实表获取业务过程中的事件或者行为细节，然后通过事实与维度之间关联，可以非常方便地统计各种事件相关的度量，比如浏览UV，搜索次数等等。

- 基于数据应用需求的分析设计事务型事实表，如果下游存在较大的针对某个业务过程事件的分析指标需求，可以考虑基于某一个事件过程构建事务型事实表。
- 事务型事实表一般选用事件发生日期或时间作为分区字段，这种分区方式可以方便下游的作业数据扫描执行分区裁剪。
- 明细层事实表的冗余子集的原则能有利于降低上层数据访问的IO开销。

- 明细层事实表维度退化到事实表原则能有利于减少上层数据访问的JOIN成本。

周期快照型事实表

周期快照型事实表主要用于分析状态型或者存量型事实，快照是指以预定的时间间隔来采样状态度量。

累计快照事实表

累计快照事实表是基于多个业务过程联合分析从而构建的事实表，如采购单的流转环节等。

累计快照事实表主要用于分析事件之间的时间间隔与周期，比如用交易的支付与发货之间的间隔，来分析发货速度，或在支付和退款环节分析支付退款率等等；同时也可以用于帮助分析一些少量的、且对刷新时间不是非常敏感的指标统计，比如，在当前事务型事实表不支持，且只有少量的统计指标时，需分析交易的关闭和发货，就可以基于累计快照事实表进行计算。

1.6 CDM汇总层设计规范

本文为您介绍CDM汇总层设计规范。

命名规范

命名规则：{project_name}.dws{业务缩写/pub}{数据域缩写}{数据粒度缩写}[{自定义表命名标签缩写}]{统计时间周期范围缩写}{刷新周期标识}{单分区增量全量标识}。

说明:

- 关于统计时间周期范围缩写，在缺省情况下，离线计算应该包括最近一天（1d）,最近N天（nd）和历史截至当天(td)三个表，如果出现nd的表的字段过多，需要拆分时，只允许以一个统计周期单元作为原子拆分，即一个统计周期拆分一个表，比如最近7天(_1w)拆分一个表；不允许拆分出来的一个表存储多个统计周期的。
- 对于{刷新周期标识}和{单分区增量全量标识}在汇总层不做强制要求。单分区增量全量标识：i：表示增量，f表示全量。
- 对于小时表不管是按天刷新还是按小时刷新, 都用_hh 来表示。
- 对于分钟表不管是按天刷新还是按小时刷新,都用_mm来表示。

数据存储及生命周期管理规范

CDM汇总层的表的类型为事实表，存储方式为按天分区。

事务型事实表一般永久保存。 周期性快照事实表根据业务需求设置生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具体计算方式如下：

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。

- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时， 建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。
- 当3个月内的最大访问跨度小于或等于180天时， 建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

1.7 MaxCompute数据开发规范

本文为您介绍MaxCompute数据开发规范。

在进行数据开发前，请做好数据仓库研发流程的阶段规划，了解各种角色及其职责。具体内容请参见数据仓库研发规范概述。

项目空间管理规范

关于项目划分和命名规范的详解，请参见MaxCompute Project分配、项目命名规范。

关于项目安全管理规范的详解，请参见#unique_13。

DataWorks项目空间目录构建建议：

- 数据开发时建议建立两层目录，第一层目录表示数据域（对于中间层项目）或业务线（对于应用层项目），例如日志、会员等；第二层目录表示层次，如DWD、DWS、DIM以及数据同步任务。
- 为避免临时查询文件列表过多，建议以开发人员姓名作为文件夹的名称进行管理。

表和视图相关规范

- 表设计规范
  - 表(table)和字段命名规范请参见ODS层设计规范、CDM公共维度层设计规范、CDM明细层设计规范、CDM汇总层设计规范。
  - 表(table)整体设计规范请参见#unique_16。

> 说明:
> 建议通过DataWorks的数据管理中管理配置模块进行表的分类管理规范，同时通过DataWorks的Data Studio里的表管理模块将表与对应分类关联。

- 视图设计规范
  - 视图的命名应与表保持一致。
  - 应创建独立的刷新任务以产生视图，创建视图的脚本如下所示。

```
create or replace view ***
```

DataWorks工作流节点设计规范

- 工作流节点类型和命名
  - 所有工作流节点的输出命名规范

```
projectname.tablename
```

  - 工作流节点命名规范

节点类型	命名规范	备注
虚拟节点	vt_{虚拟节点含义}	任务根节点
同步节点导入任务	imp{表名}[{源库标示}]	如果存在多个源库表名重复的情况，可以增加源库标示的后缀。
同步节点导出任务	exp{表名}[{目标库标示}]	如果存在多个目标库，可以增加目标库标示的后缀。
数据处理节点	{输出表名}	■ 多个目标表输出的任务时选定一个主要表名作为节点名。 ■ 多个任务插入同一张表的不同分区时，可以在后面建一个虚拟目标表任务。
Shell节点	sh_{脚本命名}	
MR节点	mr_{脚本命名}	

- 资源文件命名规范

  资源名称需有后缀表示资源类型，如.java、.py、.sh等。

- 任务设计规范

  SQL任务：

  - 每个ODPS SQL任务至少有一个输出表。
  - 脚本需支持重跑，如使用insert overwrite等语句，以便在系统错误时，重跑任务不会出现重复数据等脏数据。
  - 代码如有时间或日期参数，需采用类似`{bdp.system.cyctime}`的调度参数体系，以方便调试。
  - 自定义参数，采用${变量名}体系，在发布任务时进行配置。变量名=调度参数。

编码规范

- 编写原则

  - 代码行清晰、整齐，具有一定的可观赏性。
  - 代码编写要充分考虑执行速度最优原则。
  - 代码行整体层次分明、结构化强。
  - 代码中应有必要的注释以增强代码的可读性。
  - 规范要求非强制性地约束代码开发人员的代码编写行为，在实际应用中，只要不违反常规要求，允许存在可理解的偏差。
  - 本规范在对日常的代码开发工作起到指导作用的同时也将得到不断的完善和补充。

- 基本要求

  - 代码段中应用到的所有SQL关键字、保留字都需使用全大写或小写，例如select/SELECT、from/FROM、where/WHERE、and/AND、or/OR、union/UNION、insert/INSERT、delete/DELETE、group/GROUP、having/HAVING、count/COUNT等。不能使用大小写混合的方式，例如Select或seLECT等方式。
  - 代码段中应用到的除关键字、保留字之外的代码，都要求使用小写。
  - 四个空格为一个缩进量，所有的缩进均为一个缩进量的整数倍。
  - 禁止使用`select *`操作，所有操作必须明确指定列名。
  - 通常要求对应的括号在同一列上。

  

  说明：

  通过DataWorks或MaxCompute Studio编码时，可以用格式化工具对SQL代码进行格式化。

- 数据类型
  - MaxCompute Project的表字段类型应尽量与业务系统一致。
  - 不推荐大量使用STRING类型，以免数据加工环节的数据质量问题无法及时暴露。
  - 在对精度要求极其严格的场景下谨慎使用DECIMAL类型。
  - 关于货币类型
    - 中国货币单位统一为人民币元，国际货币单位统一为美元。
    - 除非模型有特殊说明，否则中间层金额相关的数据不做任何四舍五入操作，以避免后续的汇总计算中出现不同口径的汇总结果不一致的情况。
- DataWorks编码规范

  主要是针对通过DataWorks进行数据开发时，在DataWorks的数据开发工作台上进行代码编辑的规范。
  - 代码头部

    代码头部添加主题、功能描述、作者、日期等信息，并预留修改日志及标题栏，以便后续修改人员添加修改记录。每一行不能超过80个字符。

    DataWorks的配置中心>模板管理中可针对不同任务类型配置代码头部模板。例如，SQL类型任务头部模板默认为：

```
--odps sql
--
********************************************************************--
--author:${author}
--create time:${createTime}
```

```
--
************************************************************************--
```

- 字段排列要求

  ■ SELECT语句选择的字段按每行一个字段方式编排。

  ■ SELECT单字后面一个缩进量后应直接跟首个选择的字段，即字段离首起二个缩进量。

  ■ 其它字段前导二个缩进量再跟一个逗号（‘，’）后放置字段名。

  ■ 两个字段之间的逗号（‘，’）分割符紧跟在第二个字段的前面。

  ■ ‘AS‘语句应与相应的字段在同一行，多个字段的‘AS’建议尽量对齐在同一列上。

- SELECT子句排列要求

  SELECT语句中所用到的FROM、WHERE、GROUP BY、HAVING、ORDER BY、JOIN、UNION等子句，需遵循如下要求：

  ■ 换行编写

  ■ 与相应的SELECT语句左对齐编排。

  ■ 子句后续的代码离子句首字母二个缩进量起编写。

  ■ WHERE子句下的逻辑判断符AND、OR等与WHERE左对齐编排。

  ■ 超过两个缩进量长度的子句加一空格后编写后续代码，如：ORDER BY、 GROUP BY等。

- 运算符前后间隔要求

  算术运算符、逻辑运算符的前后要保留一个空格。

- CASE语句的编写

  SELECT语句中对字段值进行判断取值的操作将用到的CASE语句，正确的编排CASE语句的写法对加强代码行的可阅读性也是很关键的一部分。对CASE语句编排的约定如下：

  ■ WHEN子语在CASE语句的同一行并缩进一个缩进量后开始编写。

  ■ 每个WHEN子语一行编写，如果语句较长可换行编写。

  ■ CASE语句必须包含ELSE子语，ELSE子句与WHEN子句对齐。

- 子查询嵌套编写规范

  在数据仓库系统ETL开发中经常需要用到子查询嵌套，因此代码的分层编排变得非常重要。

- 表别名定义约定

  建议将所有的表加上别名。一旦在SELECT语句中给操作表定义了别名，在整个语句中对此表的引用都必须惯以别名替代。考虑到编写代码的便捷性，约定别名尽量简洁，同时避免使用关键字。

  ■ 表别名采用简单字符命名。

  ■ 多层次的嵌套子查询别名之前要体现层次关系，SQL语句别名或分层的命 名，从第一层次至第四层次，分别用P、S、U、D表示，取意为Part，Segment，Unit，Detail。也可用a、b、c、d来表示第一层次到第四层次。对于同一层次的多个子句，可以在字母后加1、2、3、4…区分。

  ■ 必要时，为表别名添加注释。

- SQL注释

  ■ 每条SQL语句均应添加注释说明。

  ■ 每条SQL语句的注释单独成行并置于语句前面。

  ■ 字段注释紧跟在字段后面。

  ■ 应为不易理解的分支条件表达式添加注释。

  ■ 应说明重要计算的功能。

  ■ 过长的函数实现，应将其语句按实现的功能分段加以概括性说明。

  ■ 常量及变量注释时，必须注释被保存值的含义，按需注释合法的取值范围 。

- MaxCompute Project名称的编写

  本项目（Project）的项目名称不需在编码中体现，如引用了其他项目（Project） 的表则需带上项目名称。示例如下。

```
--当前Project为 prj_bi
INSERT OVERWRITE TABLE test_2
SELECT  c1
       ,c2
       ,c3
FROM    prj_ods.test_1
WHERE   pt = 20181212
```

```
;
```

- DataWorks任务发布规范
  - 发布上线的时间根据业务定义。
  - 无QA参与的项目，应由开发负责自测，开发测试环境通过后再自行发布到生产环境。
  - 有QA参与的项目，开发应负责提交到调度开发环境并测试通过，而正式上线则由 QA负责打包发布到生产环境。

2 表设计指南

2.1 表的基本概念和原理

MaxCompute以表（Table）作为数据存储单元。表在逻辑上是由行和列组成的二维结构，每行代表一条记录，每列表示相同数据类型的一个字段。一条记录可以包含一个或多个列，各个列的名称和类型构成这张表的Schema。

系统架构

MaxCompute的应用架构如下图所示。您可以通过该图了解数据的处理流程。

MaxCompute项目空间的逻辑对象如下图所示。

基本概念

- 项目空间

  项目空间（Project）是MaxCompute的基本组织单元，类似于传统数据库的Database或Schema的概念。它是进行多用户隔离和访问控制的主要边界。一个用户可以同时拥有多个项目空间的权限。通过安全授权，用户可以在一个项目空间中访问另一个项目空间中的对象，例如表（Table）、资源（Resource）、函数（Function）和实例（Instance）。

- 表

  表（Table）是MaxCompute的数据存储单元。它在逻辑上是由行和列组成的二维结构，每行代表一条记录，每列表示相同数据类型的一个字段。一条记录可以包含一个或多个列，各个列的名称和类型构成这张表的Schema。

  MaxCompute的表有两种类型：内部表和外部表（MaxCompute2.0版本已支持外部表）。

  - 对于内部表，所有的数据都被存储在MaxCompute中，表中的列可以是MaxCompute支持的任意一种数据类型。本文所介绍的相关规范只针对此类表。
  - 对于外部表，MaxCompute并不真正持有数据，表格的数据可以存放在OSS或OTS中 。MaxCompute仅会记录表格的Meta信息，但您可以通过MaxCompute的外部表机制处理OSS或OTS上的非结构化数据，例如视频、音频、基因、气象、地理信息等。本文所介绍的相关规范不包含此类表。

- 分区

  分区表是指在创建表时指定分区空间，即指定表内的某几个字段作为分区列。大多数情况下，您可以将分区类比为文件系统下的目录，该文件夹存放该分区所有数据文件。分区的作用类似于分类，即通过分类把不同类型的数据放到不同的目录下。分类的标准是分区的字段，即不同的字段代表不同的分类标准。分区字段的个数没有限制，您可以设置一个或多个分区字段。分区表的意义在于优化查询。查询表时通过where语句查询指定所需查询的分区，避免全表扫描，提高处理效率，降低费用。

- 建表语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [
ASC | DESC] [, col_name [ASC | DESC] ...])] INTO number_of_buckets
BUCKETS] -- 用于创建Hash Clustering表时设置表的Shuffle和Sort属性。
[STORED BY StorageHandler] -- 仅限外部表
[WITH SERDEPROPERTIES (Options)] -- 仅限外部表
[LOCATION OSSLocation]-- 仅限外部表
[LIFECYCLE days]
[AS select_statement];
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name;
```

- 数据类型

  基础数据类型

类型	是否新增	常量定义	描述
TINYINT	是	1Y、-127Y	8位有符号整形，范围：-128~127。
SMALLINT	是	32767S、-100S	16位有符号整形，范围：-32768~32767。
INT	是	1000、-15645787（注释1）	32位有符号整形，范围：-2 ~2 -1。
BIGINT	否	100000000000L、-1L	64位有符号整形，范围-2 +1~2 -1。
FLOAT	是	无	32位二进制浮点型
DOUBLE	否	3.1415926 1E+7	8字节双精度浮点数或64位二进制浮点型
DECIMAL	否	3.5BD、99999999999.9999999BD	10进制精确数字类型，整形部分范围-10 +1~10 -1，小数部分精确到10 。

类型	是否新增	常量定义	描述
VARCHAR(n)	是	无 (注释2)	变长字符类型，n为长度，取值范围：1~65535。
STRING	否	"abc"、'bcd'、"alibaba"、”inc“ (注释3)	字符串类型，目前长度限制为8M。
BINARY	是	无	二进制数据类型，目前长度限制为8M。
DATETIME	否	DATETIME '2017-11-11 00:00:00'	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~9999年12月31日，精确到毫秒 。
TIMESTAMP	是	TIMESTAMP '2017-11-11 00:00:00.123456789'	与时区相关的时间戳类型，范围从0000年1月1日~9999年12月31日23.59:59.999999999, 精确到纳秒。
BOOLEAN	否	TRUE、FALSE	布尔类型, 值为TRUE或FALSE。

- 复杂数据类型

类型	定义方法	构造方法
ARRAY	array<int>、array<struct<a:int，b:string>>	array(1, 2, 3)、array(array(1, 2)、array(3, 4))
MAP	map<string, string>、map<smallint, array<string>>	map("k1", "v1", "k2", "v2")、map(1S, array('a', 'b'), 2S, array('x', 'y’))

类型	定义方法	构造方法
STRUCT	struct<x:int, y:int>、struct<field1:bigint, field2:array<int>, field3:map<int, int>>	named_struct('x', 1, 'y', 2)、named_struct('field1', 100L, 'field2', array(1, 2), 'field3', map(1, 100, 2, 200)

当使用新基础数据类型和复杂数据类型时，应该一起执行set命令和SQL命令。set命令如下：

- `set odps.sql.type.system.odps2=true;--session级别`
- `setproject odps.sql.type.system.odps2=true;--project级别，需project owner执行。`

说明：

- 对于精确的数据场景，不建议使用DOUBLE类型。
- round函数对DOUBLE类型字段的处理结果不一定是准确的四舍五入结果。
- 当在函数round、trunc、floor、ceil、bround中使用DOUBLE类型数据时，需要注意精度问题。

表的限制项

边界名	最大值/限制条件	分类	说明
表名长度	128字节	长度限制	表名和列名中不能出现特殊字符，只能出现英文的a-z、A-Z及数字和下划线，且必须以字母开头。
注释长度	1024字节	长度限制	注释内容为长度不超过1024字节的有效字符串。
表的列定义	10000个	数量限制	单表的列定义个数最多10000个。
单表分区数	60000个	数量限制	一张表最多允许60000个分区。
表的分区层级	6级	数量限制	表的分区层级不能超过6个层级。

2.2 表设计规范

MaxCompute中不同类型计算任务的操作对象（输入、输出）都是表。表设计是否合理将影响存储和计算的性能，进而影响到存储和计算的计费。

表设计主要目标

- 降低存储成本

  合理的表设计可以降低数据分层设计上的冗余存储，减少中间表的数据量大小。对表数据的生命周期进行正确地管理，也能够直接降低存储的数据量及存储成本。
- 降低计算成本

  规范化的表设计可以帮助使用者优化数据的读取，从而减少计算过程中的冗余读写和计算，提升计算性能，降低计算成本。
- 降低维护复杂度

  规范化的表分层设计能够直接体现业务的特点。例如，在规范化设计表的同时对数据通道中的数据采集方式进行优化，可以减少分布式系统中小文件的问题，降低表和分区维护的数量等复杂度。

表设计的影响

表设计所影响的操作：表创建、入数据、表更新、表删除、表管理。

导入数据场景（区分要做实时数据采集还是离线批量数据写入）

- 导入即查询与计算
- 多次导入，定时查询与计算
- 导入后生成中间表进行计算

> 说明:
>
> - 合理的表设计和数据集成周期管理能够降低数据在存储期间的成本。
> - MaxCompute优先计算批量数据集成库并按业务逻辑进行计算，例如按照分区进行计算。
> - 导入后立即查询与计算，需要考虑每次导入的数据量，减少流式小量数据导入。
> - 不合理的数据导入及存储（小文件）会影响整体的存储性能、计算性能、运维稳定性。

表设计步骤

1. 确定所属项目空间，依据业务过程规划表类型，分析数据层次。
2. 定义表描述，进行权限定义与Owner定义。
3. 依据数据量、数据集成特点定义分区表或非分区表。

4. 定义字段或分区字段。
5. 进行表创建、表转换。
6. 明确导入数据场景的相关因素（包括批量数据写入、流式数据写入、条式数据插入）。
7. 定义表和分区数据生命周期。

说明:

- 创建完表后，您可以依据业务变化修改表的schema，例如设置生命周期：RangeClustering。
- 在表设计阶段，需要特别注意区分数据的场景（批量数据写入、流式数据写入、周期性条式数据插入）。
- 合理使用非分区表和分区表。建议采用分区表来设计日志表、事实表，原始采集表等，并按照时间进行分区。
- 注意各种表和分区的限制条件。

表数据存储规范

- 按数据分层规范数据的生命周期
  - 源表ODS层： 每天从业务系统同步过来的数据，全部保留，生命周期定义永久保存。当下游数据受损时，可以从ODS恢复数据。若ODS每天同步过来的是全量表，则可以通过全表拉链的方式来压缩存储。
  - 数据仓库（基础）层： 至少保留一份完整的全量数据（不必像ODS那样存储冗余的全量表）。您可以通过拆表或者做分区来提升性能。
  - 数据集市层： 数据将被按需保留1~3年。数据集市的数据比较容易生成，所以无需保留久远的历史数据。
- 按数据的变更和历史规范数据的保存
  - 客户属性、产品属性不断在变更。将这些属性的历史变化情况记录下来，以便追溯某个时点的值。
  - 在事实表里冗余维表的字段，即把”事件发生时“的各种维度属性值与该事件绑定起来。 使用者无需关联多张表就可以使用数据。此方式仅可应用于数据应用层。
  - 用拉链表或者日快照的形式，记录维表的变化情况。这使得数据结构变得灵活、易于扩展，数据一致性得到了增强，数据加工者可以更加方便地管理数据。此方式仅可应用于数据基础层。

数据导入通道与表设计

通道类型

- Datahub

  规划写入的分区与写入流量之间的关系，做到每64M进行一次commit。

- 数据集成或DataX

  规划写入的表分区的频率，做到每64M进行一次commit，以免commit空目录。

- DTS

  规划写入的表存量分区与增量分区的关系，设置commit频率。

- Console（Run SQL or Tunnel upload）

  需要避免高频小数据量文件的插入或者上传。

- SDK执行SQL的insert into语句

  对表或者分区上传时需要注意在插入到分区后使用merge语句进行小文件整理操作，以免对一个分区或者非分区表插入多次。

说明:

- MaxCompute导入数据的通道只能是Tunnel SDK或执行SQL的insert into语句，请避免流式插入。
- 以上各通道本身均由自身逻辑进行流式数据写入、批量数据写入、周期调度写入。
- 当使用数据通道写入表或分区时，需将一次写入的数据量控制在合理范围，例如64M以上。

分区设计与逻辑存储的对应

一张表里有很多个一级分区，每个一级分区都会按时间存储二级分区，每个二级分区都会存储所有的列，如下图所示。

说明:

- 请设置分区的数量上限。
- 请避免每个分区中只存少量数据。
- 分区的条件设置应以方便数据的查询和计算为前提。
- 避免每个分区中出现多次的数据写入。

表和分区设计基本规则

- 所有的表、字段名要使用统一的命名规范。
  - 命名应能区分该表的业务类型。
  - 命名应能区分该表是“事实表”或“维度表”、“日志表”、“极限存储表”（待发布的功能）。
  - 命名应能区分该表的实体信息。
- 不同表中具有相同业务含义的字段要定义成统一的数据类型，避免不必要的类型转换。
- 分区设计及使用规则：
  - 支持新增分区，不支持新增分区字段。
  - 单表支持分区数量为6万。
  - 对于多级分区的表，如果想添加新的分区，则必须指明全部的分区值。
  - 不支持修改分区列的列名，只能修改分区列对应的值。修改多级分区的一个或者多个分区值时，多级分区的每一级的分区值都必须写上。

分区设计

- 分区字段和普通字段的选择

  通过分区字段，您可以划分数据扫描范围，更加方便地管理数据。

  您在可以在创建表时设置普通字段和分区字段。通常，普通字段可以被理解为数据文件的数据，而分区字段可以被理解为文件系统的目录。表的存储空间的占用主要是普通字段的空间占用。

  分区列虽不直接存储数据，但如同文件系统里的目录，可以方便您管理数据。例如，在计算时若指定具体的分区，则计算过程中只需查询对应分区，从而减少计算输入量。

  分区表的分区列的级数不能超过6级，即底层存储数据的目录层数不能超过6层。您应为分区表设置合适的生命周期。当部分数据的生命周期与其它数据不同时，您可以通过细粒度分区实现对部分数据的管理。

   说明：

  - 设置分区字段时，您可以从数据管理和常用的数据扫描方面考虑，来选择对应的字段。
  - 不具备规律或类型数量大于10000且不经常作为查询条件的字段，应被设置成普通字段。

- 分区字段定义依据

  按优先级高低排序：

  - 分区列的选择应充分考虑时间因素，尽量避免对于存量分区进行更新。
  - 如果有多个事实表（不包括维度表）进行join，应将查询条件where范围的列作为分区列。
  - 选择group by或distinct包含的列作为分区列。
  - 选择值分布均匀的列，而不要选择分区倾斜的列作为分区列。
  - 常用SQL语句中若经常包含某列的等值或in的查询条件，则选择该列作为分区列。例如：

    ```
    select … from table where id=123 and ….;
    ```

- 分区个数定义依据

  - 时间分区

    建议按天或月进行分区。如果按小时进行分区，则二级分区的平均数量不应超过8个。

  - 地域分区

    若对省、市、县进行分区，则应考虑进行多级分区。23个省，5个自治区，4个直辖市，2个特别行政区，50个地区（州、盟），661个市（其中直辖市4个、地级市283个、县级市374个），1636个县（自治县、旗、自治旗、特区和林区），按照最细粒度县进行分区后，不应再按照更细粒度小时进行分区。

  - 单分区与多级分区

    在单分区下，建议每次提交64M数据。如果为多级分区，则需保证每个最细粒度级分区下的二级分区的数据都遵循单分区个数规则。

  - 单表分区

    单表分区数（包括下级分区）不能超过6万。

- 分区数量和数据量建议

  在计算的时候可以使用分区裁剪是分区的优势。

  - 建议单个分区中数据量不要太大。
  - 应尽量避免分区数据倾斜，避免单个表不同分区的数据量差异超过100万。
  - 做分区设计时应合理规划分区个数，较细粒度的分区在跨分区扫描时会影响到SQL的执行性能。
  - 单个分区中数据量较大的情况下，MaxCompute执行任务时会做分片处理不影响分区裁剪的优势。
  - 单个分区中文件数较多时，会影响MaxCompute Instance数量，造成资源浪费和SQL性能的影响。
  - 采用多级分区，先按日期分区，然后按交易类型分区。
  - 拆表，一种交易类型独立成一张表，然后每张表按日期分区。
  - 维度表不做分区。

2.3 表设计最佳实践

本文为您介绍表设计的最佳实践方式，为实际开发提供依据和指导。

产生大量小文件的操作

MaxCompute表的小文件会影响存储和计算性能。在进行表设计时，应考虑避开产生大量小文件的操作。

- 使用MaxCompute Tunnel SDK上传数据的过程中，每commit一次就会产生一个文件。这时每个文件过小（例如几KB），并且频繁上传（例如每5秒上传一次），则一小时就会产生720个小文件，一天就会产生17280个小文件。
- 使用MaxCompute Tunnel SDK上传数据，如果创建了session却没有上传数据，而是直接commit，则会产生大量空目录（服务侧等同于小文件）。
- 使用MaxCompute Console命令行工具Tunnel命令上传时，将本地大文件切分过小会导致上传后产生的文件数过多，文件过小。
- 通过DataHub做数据归档，DataHub的每个shard写入MaxCompute时存在条件限制，即数据总量每到64MB，commit一次，或每隔5分钟commit一次，形成一个文件。当开启的shard数过多（例如20个shard）时，每个shard数据在5分钟内都远达不到64M（例如几百KB），就会产生大量小文件。相应地，一天会产生24*12*20=5760个小文件。

- 通过Dataworks等数据开发工具进行数据增量插入（insert into）到MaxCompute表（或表分区）时，每次进行数据增量插入都会产生一个文件。若每次插入10条，则每天累计插入10000条记录，即会产生1000个小文件。
- 使用阿里云DTS将数据从RDS等数据库同步到MaxCompute时，会创建全量表和增量表。在增量表进行数据插入的过程中，会因为每次数据插入条数较少而造成增量表中的小文件问题。例如每隔5分钟执行一次同步，每次同步的数据量为10条，一天内的增量为10000条，则会产生1000个小文件。此种场景下，需要在数据同步完成后进行全量极限表和增量数据表的merge。
- 源数据采集客户端太多时，如果源数据通过Tunnel直接进入到一个分区，则每个源数据采集客户端提交一次数据，都会在同一分区下产生一个独立的文件，从而导致大量小文件的出现。
- 当SLS触发FunctionCompute持续高频地往MaxCompute中心传入文件时，小文件流式数据会进入MaxCompute。

根据数据划分项目空间

项目空间（Project）是MaxCompute最高层的对象。按项目空间进行资源的分配、隔离和管理，实现了多租户的管理能力。如果多个应用需要共享“数据”，则推荐使用同一个项目空间；反之，如果多个应用所需“数据”是无关的，则推荐使用不同的项目空间。项目空间的表和分区可以通过Package授权的方式进行交换。

维度表设计的最佳实践

描述属性的表通常被设计为维度表。维度表可与任意表组中的任意表进行关联，且创建时无需配置分区信息，但是对单表数据量大小有所限制。

> 说明:
>
> - 通常要求维度表的单表量不超过1000万个。
> - 维度表的数据不应被大量更新。
> - 可以使用mapjoin语句进行维度表和其它表的join操作。

拉链表设计

在数据仓库的数据模型设计过程中，经常会遇到如下需求：

- 数据量较大。
- 表中的部分字段被更新，例如用户的地址、产品的描述信息、订单的状态、手机号码等。
- 需要查看某一个时间点或时间段的历史快照信息。例如，查看某一个订单在某一个历史时间点的状态，或查看某一个用户在过去某段时间内更新过几次等。

- 变化的比例不大、频率不高。假设总共有1000万个会员，且每天新增和发生变化的会员只有10万左右，如果每天都在表中保留一份全量，那么每次全量中会保存很多不变的信息，极大地浪费了存储资源。

MaxCompute提供了将不同表转化为极限存储表的方法。极限存储操作示例如下：

1. 创建源表。

```
create table src_tbl (key0 STRING, key1 STRING, col0 STRING, col1
STRING, col2 STRING) PARTITION (datestamp_x STRING, pt0 STRING);
```

2. 导入数据。
3. 将src_tbl转变为极限存储的表。

```
set odps.exstore.primarykey=key0,key1;
[set odps.exstore.ignorekey=col0;]
EXSTORE exstore_tbl PARTITION (datestamp_x='20140801');
EXSTORE exstore_tbl PARTITION (datestamp_x='20140802');
```

说明：

极限存储功能待发布，在此主要介绍其设计思想。

采集源表的设计

数据采集方式包括流式数据写入、 批量数据写入、周期调度条式数据插入。数据量较大时，需确保同一个业务单元的数据使用分区和表进行分；数据量较小时，需优化采集频率。

- 流式数据写入
  - 对于流式写入的数据，采集的通道通常较多，相关采集通道应做有效区分。在单个数据通道写入量较大的情况下，应该按照时间进行分区设计。
  - 在采集通道数据量较小的情况下，适合采取非分区表设计，将终端类型和采集时间设计成标准列字段。
  - 采用DataHub进行数据写入时，应该合理规划shard数量，避免出现由于shard过多而导致的采集通道流量较小而通道较多的问题。
- 批量数据写入

  使用批量数据写入方式时，应重点关注写入周期。
- 周期调度条式数据插入

  应避免使用周期调度条式数据插入的方法。若无法避免使用此方法，则需建立分区表，在新分区进行插入操作，减小对于原来分区的影响。

日志表的设计

日志的本质是个流水表，不涉及记录的更新。日志表设计需注意以下几点：

- 考虑是否需要对日志进行去重处理。
- 考虑是否需要扩展维度属性。
  - 您需要考虑业务使用的频次以及关联是否会造成产出的延迟，来确定是否需要关联维度表、扩展维度属性字段。
  - 需要谨慎选择是否对维度表进行扩展。
- 考虑区分终端类型。
  - 日志表中的数据量很庞大，在业务分析使用时，通常会按PC端、APP端来统计分析。由于PC端、APP端采用不同的体系采集数据，所以通常的做法是按终端设计多个明细DWD表。
  - 如果终端较多但数据量不大，例如一个终端的数据量小于1TB但采集次数较多，则可以不对终端进行分区，设置终端信息为普通列。

说明:

- 对日志表进行分区设计时，可以按照日志采集的时间进行分区。在写入数据前进行数据的采集和整合，整合好后，一次性提交数据（通常是每64M提交一次）。
- 日志数据很少有对原来分区的更新操作，可以用insert进行少量数据的插入，但通常需要限制插入次数。
- 如果有大量的更新操作，则需采用insert overwrite操作避免小文件问题。
- 为日志表设置合理的分区，并对长久不被访问的冷热数据配置归档操作。

互动明细表的设计

周期快照表中存放的是每天收藏的所有记录的快照。

- 面临的问题：历史累计的记录有很多，每天需要将当天增量表与前一天的全量表合并才能生成快照，非常耗资源。统计最近1天的新增收藏数，需要扫描全量表。如何才能降低资源?
- 建议：建立一个事务性事实表，在建立一个存放当前有效收藏的周期快照表，以满足各种不同业务的统计分析需要。

说明:

- 设计互动明细表时，区分存量数据和增量数据之间的关系非常重要。
- 新增数据应作为增量数据写入新分区。
- 应尽量减少对已有分区中的数据进行修改和插入新数据。
- 在数据插入和全表覆盖写种选择时应尽量选用insert overwrite而并选择insert into。

MaxCompute表数据更新与删除操作

MaxCompute实现关系型数据库所支持的delete/update/merge SQL的示例如下：

- 表准备

```
-- 上日全量表
table1(key1 string,key2 string,col1 string,col2 string);
-- 今日增量表
table2(key1 string,key2 string,col1 string,col2 string);
-- 今日增量表（删除）
table3(key1 string,key2 string,col1 string,col2 string);
```

- update（将table2表中的记录的值更新到table1表中）

```
insert overwrite table table1
select t1.key1
      ,t1.key2
      ,case when t2.key1 is not null then t2.col1 else t1.col1 end
as col1
      ,case when t2.key1 is not null then t2.col2 else t1.col2 end
as col2
  from table1 t1
  left outer join table2 t2 on t1.key1 = t2.key1 and t1.key2 = t2.
key2
;
```

- delete（从table1表中删除table2表中的记录）

```
insert overwrite table table1
select t1.key1
      ,t1.key2
      ,t1.col1
      ,t1.col2
  from table1 t1
  left outer join table2 t2 on t1.key1 = t2.key1 and t1.key2 = t2.
key2
  where t2.key1 is null
;
```

- merge（当日发生过删除操作）

```
insert overwrite table table1
select
  from(
-- 先把上日和今日都存在的记录从上日表中排除，再把今日删除的记录排除。剩下的就是今
日没有更新的记录。
select t1.key1
      ,t1.key2
      ,t1.col1
      ,t1.col2
  from table1 t1
  left outer join table2 t2 on t1.key1 = t2.key1 and t1.key2 = t2.
key2
  left outer join table3 t3 on t1.key1 = t3.key1 and t1.key2 = t3.
key2
  where t2.key1 is null or t2.key1 is null
  union all
-- 再合并上今日增量，就是今日的全量。
select t2.key1
      ,t2.key2
      ,t2.col1
      ,t2.col2
  from table2 t2)tt
```

```
;
```

- merge（当日没有发生过删除操作）

```
insert overwrite table table1
select
  from(
-- 先把上日存在，今日也存在的记录从上日表中排除。剩下的就是今日没有更新的记录。
select t1.key1
      ,t1.key2
      ,t1.col1
      ,t1.col2
  from table1 t1
  left outer join table2 t2 on t1.key1 = t2.key1 and t1.key2 = t2.
key2
  where t2.key1 is null
  union all
-- 再合并上今日增量，就是今天的全量。
select t2.key1
      ,t2.key2
      ,t2.col1
      ,t2.col2
  from table2 t2)tt
;
```

表创建设计示例

- 场景

  天气情况信息采集。

- 基本信息

  - 数据信息包括地名、关于此地的属性信息（例如面积、基本人口数量等）、天气信息。
  - 属性的数据变化较小，但天气信息数采用多个终端采集，且数据量较大。
  - 天气信息变化较大，但在终端数量稳定的情况下流量基本稳定。

- 表设计指南

  - 建议将数据信息划分为基本属性表和天气日志表，分别用于存储变化小和变化大的数据。
  - 因为天气信息的数据量巨大，在对天气日志表按照地域进行分区后，可以按照时间（例如天）进行二级分区。此种分区方式可避免发生因某一个地点或某一个时间的天气变化而造成其他无关数据变化。
  - 建议采集终端上使用DataHub进行数据汇聚，然后依据稳定的流量值选择合适的shard通道数量，以批量数据传输的方式写入到天气日志表中，而非insert into。

2.4 MaxCompute表的特殊功能

本文为您介绍MaxCompute表的生命周期、避免全表扫描和小文件以及Hash Clustering表等特殊功能。

生命周期

MaxCompute为表和分区提供数据生命周期管理。表（分区）数据从最后一次更新时间算起，在经过指定的时间后如没有变动，则此表（分区）将被MaxCompute自动回收，这个指定的时间就是生命周期。生命周期只能以表为单位进行设置。

```
create table test_lifecycle(key string) lifecycle 100;/alter table
test_lifecycle set lifecycle 50;
```

MaxCompute会根据每张非分区表或者分区表中分区的LastDataModifiedTime和lifecycle的设置，进行判断是否要回收他们。

MaxCompute SQL提供touch操作用来修改分区的LastDataModifiedTime。此操作会将分区的LastDataModifiedTime值修改为当前时间，这样MaxCompute会认为表或分区的数据有变动，生命周期的计算会重新开始。

```
ALTER TABLE table_name TOUCH PARTITION(partition_col='partition_
col_value', ...);
```

说明:

- 合理规划表的生命周期，在创建表时即设置生命周期，可有效减少存储压力。
- 对表数据的任何变动都会影响生命周期回收数据的判断时间，包括小文件合并。

避免全表扫描

- 在表设计时如何避免全表扫描

  表设计是指建立分区表或者对扫描条件进行列设计。在设计时，需要注意以下几点：

  - 对数据表进行合理的分区。
  - 把常用查询条件设置成列名。
  - 对常用查询条件进行hash clustering。

  在数据计算时如何避免全表扫描

- 您可以增加分区过滤的条件或减少扫描的分区数，实现减少数据扫描量。
- 把全局扫描表的中间结果进行存储，形成中间表。
- 如果每天都需扫描某表一整年的分区，则计算消耗是非常大的。因此，建议您拆出一张中间表，每天做一次汇总，然后再扫描这张中间表的一整年的分区，从而减少扫描的数据量。

避免小文件

- Reduce计算过程产生的小文件：只需要`insert overwrite`源表（或分区）即可，或者写入到新表删除源表。
- Tunnel数据采集过程中产生的小文件建议：
  - 调用Tunnel SDK时，每当buffer达到64M时，进行一次提交。
  - 使用console时应避免频繁上传小文件，建议积累至一定的数量后再一次性上传。
  - 如果导入的是分区表，建议给分区表设置生命周期，过期不用的数据将会被自动清理。
  - 也可以同第一种方案，使用`insert overwrite`语句对源表（或分区）进行操作。
  - 使用alter合并模式时，通过console命令进行合并。
- 建议创建临时表时加上生命周期，在到期后垃圾回收机制会自动回收临时表。
- 申请过多的DataHub shard将会产生小文件问题，以下是申请DataHub shard数目时的策略。
  - 单个shard的默认吞吐量是1MB/s，可以按照这个分配实际的shard数目（可以在此基础上多加几个）。
  - ODPS的同步逻辑是每个shard会有一个单独的task（每隔5分钟或每满64MB，此task会commit一次），默认设置5分钟是为了能在ODPS上尽快查到数据。当按照小时建partition时，一个shard每个小时将会产生12个文件。若此时数据量很少而shard很多，则ODPS里就会出现很多小文件（shard*12/hour）。
  - 应按需分配shard，避免过度分配。

转化Hash Clustering表

Hash Clustering表的优势在于可以实现Bucket Pruning优化、Aggregation优化以及存储优化。在创建表时，使用clustered by指定Hash Key后，MaxCompute将对指定列进行Hash运算，按照Hash值分散到各个Bucket里。Hash Key值的选择原则为选择重复键值少的列。

如何转化为Hash Clustering表：

```
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`语句适用于存量表，在增加了新的聚集属性之后，新的分区将做hash cluster存储。

创建完Hash Clustering表后，您可以使用`insert overwrite`语句将源表转化为Hash Clustering表。

说明:

Hash Clustering表存在以下限制：

- **不支持`insert into`语句，只能通过`insert overwrite`来添加数据。**
- **不支持直接使用tunnel upload数据到range cluster表，因为tunnel上传的数据是无序的。**