ALIBABA CLOUD

阿里云

大数据计算服务

规范

文档版本：20201014

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.数仓建设指南

1.1. 数据模型架构规范

本文为您介绍数据模型架构规范。

数据模型架构规范

声明

本文以及后续章节中介绍的非功能性规范均为建议性规范，产品功能上并未强行限制，仅供指导。

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。它相当于一个数据准备区，同时又记录基础数据及历史变化。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。

数据仓库的具体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名称，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。在后续数据开发过程中，为项目空间、表、字段等命名作为重要参照。

- 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项目。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，数据仓库开始工作时的第一步。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中的记录和字段组成，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。当核心模型与扩展模型进行关联时，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，不能让公用的处理逻辑暴露给应用层实现，不能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的性能，但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

本文为您介绍建设MaxCompute数据仓库的公共规范。

数仓建设 公共规范

层次调用约定

应用层应优先调用公共层数据，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。

中间层需要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，为其他层提供数据服务。应用层需要积极配合中间层持续改造公共层。必须避免出现过度的引用ODS层、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。
- 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，需要针对性地建设好CDM公共汇总层。

MaxCompute项目分配

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

- ODS层项目，按业务部门的粒度建立。
- CDM层项目，按业务部门的粒度建立。
- ADS层项目，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层项目名称以ods为后缀，例如tbods。
- 中间层项目名称以cdm为后缀，例如tbcdm。
- 应用层项目中，数据报表、数据分析等应用名称以bi为后缀，例如tbbi；而数据产品等应用名称以app为后缀，例如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。例如，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

MySQL数据类型	MaxCompute数据类型
DOUBLE	DOUBLE
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。以Y和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

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。

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数据值、零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。

ODS层设计

数据同步及处理规范

- 数据同步方式的选择

  数据同步规范如下。基本规范通过需求形式落地到DataWorks的数据集成，规范落地情况依赖工具的推进节奏：

  - 一个系统的源表只允许同步一次到MaxCompute。
  - 数据集成只同步离线全量数据。增量数据同步可以通过数据传输服务DTS实现。

- 数据加载与处理
  - 数据集成同步全量数据时会直接进入全量表的当日分区。
  - DTS同步数据处理方式如下：
    - 全量初始化：对于每个同步表，全量初始化的数据都会独立存储在MaxCompute的全量基线表中，表名的默认格式为源表名_base。
    - 增量数据同步：增量数据会实时同步到MaxCompute，并存储在增量日志表中，表名的默认格式为源表名_log。每个同步表对应一个增量日志表。在同步增量数据时，DTS会将多条记录合并到一个文件并写入MaxCompute。
  - DTS全量数据合并。DTS根据全量基线表和增量日志表得到某个时刻表的全量数据，并通过MaxCompute SQL合并全量数据。您可以通过DataWorks配置一个全量数据Merge节点，当全量数据完成Merge后，可自动调度后续的计算分析节点。您也可以配置调度周期，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控制。
  - 如果源系统新增了字段，您需要重新配置数据集成同步作业。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}。
  - 按小时同步的增量表：{project_name}.s{源系统表名}{delta}_{hh}。

- 按小时同步的全量表：{project_name}.s{源系统表名}{hh}。
- 当不同源系统同步到同一个Project下的表命名冲突时，您需要给同步较晚的表名加上源系统的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。
  - 字段名与MaxCompute关键字冲突时，在源字段名后加上col，即源字段名col。MaxCompute关键字详情请参见保留字与关键字。
- 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为{源系统表名}{tddl的appname}[_delta]。

  - 任务的输出名称，即输出表的名称，需要与数据存储及生命周期管理规范保持一致。详情请参见数据存储及生命周期管理规范。

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（数据量非常大，例如一天数据量大于100 GB）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为2天，支持通过 `ds=max_pt(tablename)` 方式访问数据。
ODS增量表	按天分区	• 有对应全量表，最多保留最近14天分区数据。 • 无对应全量表，需要永久保留数据。
ODS ETL过程临时表	按天分区	最多保留最近7天分区。
DBSync非去重数据	按天分区	由应用通过中间层保留历史数据，默认ODS层不保留历史数据。

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全量表必须有注释。
- 每个ODS全量表必须监控分区空数据。
- 仅有监控要求的ODS表才需要创建数据质量监控规则。您可以通过DataWorks配置数据质量监控规则，详情请参见配置数据质量监控。
- 建议对重要表的重要枚举类型字段进行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周同环比监控，如果周同环比无变化，表示源系统已迁移或下线。

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。

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。除了以下情况：

  - 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，其他名称也必须是规范的维度属性的别名。例如，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。相关性强是指经常需要一起查询或进行报表展现、两个维度属性间是否存在天然的关系等。例如，商品基本属性和所属品牌。
    - 无相关性的维度可以适当考虑杂项维度（例如交易），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（例如会员表），可以做适当拆分：
      - 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（例如商品表），可以适当冗余一些子集合，以减少下游扫描数据量：
      - 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}]，其中的pub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。

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明细层的表、数据存储与生命周期管理和各种事实表的设计规范。

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}。

命名说明：

- 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。周期快照型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，例如浏览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。例如，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。

累计快照事实表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。例如，在当前事务型事实表不支持，且只有少量的统计指标时，需要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。

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}。

命名说明：

- 在默认情况下，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nd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。例如，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。
- 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。

事务型事实表一般会永久保存。周期快照型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. CDM接口数据层设计规范

本文为您介绍CDM接口数据层设计规范。

CDM接口数据层设计规范
接口数据层将不同数据域的汇总数据预关联在一个物理表，开放给应用层使用，以减少应用层多次重复JOIN的成本开销，CDM接口数据层更适用于实时计算。

命名规范

命名规则：{project_name}.dwi{业务 BU 缩写/pub}{数据域/hbd}{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}。

命名说明：

- 关于统计时间周期范围缩写，默认情况下，离线计算包括最近一天（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。例如，最近7天（1week）拆分一个表，不允许拆分出一个表存储多个统计周期。
- 如果一个汇总表出现混合多个数据域时，表名称中需要使用hbd（hybrid 缩写）进行标识，这种情况当前只用于准实时情况，离线计算不建议跨数据域存储数据。

1.8. 其他命名规范

本文为您介绍视图、中间表以及其它表的命名规范。

视图 临时表 中间表 下线表

视图命名规范

视图命名规范如下：

- ODS层直接以视图形式开放到CDM层：{project_name}.dwd_{ODS 表名}。
- 中间层视图命名规范：遵循中间层命名规范，且加上后缀{project_name}.{dws/dwd}_{中间层表命名规范要求}。

脚本间可复用的中间表命名规范

因为所有表都是以分区表形式存在的，因此中间表不设置命名规范，全部以正式表方式处理。

临时表命名规范

不同场景下临时表命名规范不同：

- 测试、数据探查、临时取数等场景下产生的临时表命名规范为：{project_name}.tmp{工号/操作人名标识}{产出表表名}_{n}。
- 脚本内临时表命名规范：{project_name}.tmp{产出表表名}{n}。

下线表命名规范

下线表，统一后缀 _retireyyyymmdd，生产任务下线后的表重命名为YYYYMMDD。三个月后需要与表拥有者确认是否能删除。

1.9. MaxCompute数据开发规范

本文为您介绍MaxCompute数据开发规范，包括项目空间、表、视图、工作流节点和编码规范。

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项目分配、项目命名规范。
- 关于项目安全管理规范的详解，请参见安全模型。
- 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表设计规范。

> 说明 建议通过DataWorks数据管理中的管理配置模块进行表的分类管理，同时通过DataWorks数据开发中的表管理模块将表与对应分类关联。

- 视图设计规范
  - 视图的命名规范与表保持一致。
  - 建议创建独立的刷新任务以产生视图，创建视图的脚本如下。

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

DataWorks工作流节点设计规范

- 工作流节点类型和命名
  - 工作流节点的输出表命名规范

```
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}]	如果多个源库存在表名重复的情况，可以增加源库标识的后缀。
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，可以增加目标库标识的后缀。
数据处理节点	{输出表名}	■ 多个目标表输出任务时，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，可以建一个虚拟目标表任务。
Shell节点	sh_{脚本命名}	不涉及
MapReduce节点	mr_{脚本命名}	不涉及

- 资源文件命名规范

  资源名称需有后缀表示资源类型，例如.JAVA、.PY、.SH等。

- 任务设计规范

  SQL任务：

  - 每个MaxCompute SQL任务至少有一个输出表。
  - 脚本需支持重跑，例如使用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。
  - 代码如果有时间或日期参数，需采用类似 `{bdp.system.cyctime}` 的调度参数，以方便调试。
  - 自定义参数，采用${变量名}在发布任务时进行配置。变量名即为调度参数。

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。
- 基本要求
  - 代码中应用到的所有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请谨慎使用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执行任何四舍五入操作，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

  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。并提供修改日志及标题栏的功能，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中针对不同的任务类型提供了不同的代码头部模板，也可以在配置中心自定义头部模板。例如，SQL类型任务头部模板默认为如下。

    ```
    --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。例如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CASE语句，正确的编排CASE语句对加强代码行的可读性也是很关键的一部分。对CASE语句编排的约定如下：

  - WHEN子句在CASE语句的同一行并缩进一个缩进量后开始编写。
  - 每个WHEN子句单独一行编写，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

  建议对所有的表加上别名。一旦在SELECT语句中对表定义了别名，在整个语句中对此表的引用都必须以别名替代。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。 表别名定义约定如下：

  - 表别名采用简单字符命名。
  - 多层次的嵌套子查询，在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项目空间名称的编写

  本项目空间（Project）的名称不需在编码中体现。如果引用了其他项目空间的表则需要在表名前加上项目空间名称。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人员参与的项目，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人员参与的项目，开发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. 表概述

本文为您介绍表的基本概念、数据类型和使用限制。

表原理 表基本概念

系统架构

您可以通过如下系统架构图了解MaxCompute数据的处理流程。

MaxCompute中表的类型如下。

基本概念

- 项目

  项目（Project）是MaxCompute的基本组织单元，类似于传统数据库的Database或Schema。项目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的权限。您通过安全授权可以跨项目访问对象，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，各个列的名称和数据类型构成表的Schema。

  MaxCompute的表有两种类型：

  - 内部表：所有数据都存储在MaxCompute中，内部表的列可以是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。
  - 外部表：数据不存储在MaxCompute中，表数据可以存放在OSS或OTS 。MaxCompute仅会记录表格的Meta信息。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、气象或地理信息等。本文所介绍的相关规范不包含此类表。

- 分区

  您需要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。

  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。

  分区表用于优化查询，查询表时通过WHERE语句指定待查询的分区，避免全表扫描，提高处理效率，降低费用。

- 建表语法如下，详情请参见表操作。

```
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 | DES
C] ...])] 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	32位有符号整形。范围：-2^{31} ~2^{31} -1。
BIGINT	否	100000000000L、-1L	64位有符号整形。范围：-2^{63} +1~2^{63} -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^{36} +1~10^{36} -1。小数部分精确到10^{-18} 。
VARCHAR(n)	是	无	变长字符类型，n为长度。取值范围：1~65535。
STRING	否	"abc"、'bcd'、"alibaba"、"inc"	字符串类型。长度限制为8 MB。
BINARY	是	无	二进制数据类型。长度限制为8 MB。
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：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：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'))
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))

您可以执行如下命令打开复杂数据类型。

- Session级别，需要将set命令和SQL命令一起提交。

```
set odps.sql.type.system.odps2=true;
```

- Project级别，需要Project Owner执行。

```
setproject odps.sql.type.system.odps2=true;
```

> 说明
> - 数据精确的场景，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。
> - 当您在函数Round、Trunc、Floor、Ceil和Bround中使用DOUBLE类型数据时，需要注意精度问题。

表的限制

边界名	最大值	限制类别	说明
表名长度	128字节（byte）	长度限制	表名和列名中不能出现特殊字符，只能出现英文字母a~z、A~Z、数字和下划线，且必须以字母开头。
注释长度	1024字节（byte）	长度限制	注释内容为长度不超过1024字节（byte）的有效字符串。
表的列定义	1200个	数量限制	单表的列个数最多为1200个。

边界名	最大值	限制类别	说明
单表分区数	60000个	数量限制	单标最多允许有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。

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。

声明

本文中介绍的非功能性规范均为建议性规范，产品功能无限制，仅供参考。

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您优化数据的读取，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。

- 降低维护成本

  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，降低表和分区的维护数量。

表设计主要影响

表设计影响的操作有：创建表、导入数据、更新表、删除表及管理表。

其中，导入数据场景按照实时数据采集和离线导入批量数据的方式分为如下三种：

- 导入立即查询与计算。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。
- 多次导入并定时查询与计算。
- 导入后生成中间表进行计算。

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。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。MaxCompute优先按照业务逻辑对批量数据进行计算，例如，按照分区进行计算。

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. 创建表和转换表。

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、周期性条式数据插入）。
7. 定义表和分区的生命周期。

> 说明
> - 创建完表后，您可以根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、事实表和原始采集表等，并按照时间进行分区。
> - 注意表和分区的限制条件。

表数据存储规范

- 按数据层规划数据的生命周期：
  - 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。
- 按数据变更规划数据的保存方式：
  - 记录客户属性、产品属性的历史变化情况，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。
  - 任意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

通道类型有以下几种：

- DataHub

  规划写入的分区与写入流量之间的关系。数据达到64 MB会执行1次Commit。
- 数据集成或DataX

  规划写入表分区的频率。数据达到64 MB会执行1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。
- 客户端（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

  SDK执行INSERT INTO语句，数据上传至表或者分区后，使用MERGE语句整理小文件。

> 说明
>
> - MaxCompute导入数据的通道只能是Tunnel SDK或执行INSERT INTO语句，请避免流式插入数据。
> - 以上各通道本身均由自身逻辑进行流式数据写入、批量数据写入和周期调度写入。
> - 当使用数据通道写入表或分区时，需将1次写入的数据量控制在合理范围，例如，64 MB以上。

分区设计与存储逻辑

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。

分区设计需要注意：

- 设置分区的数量上限。
- 避免每个分区中只存少量数据。
- 以方便数据查询和计算为前提设置分区列。
- 避免每个分区中出现多次数据写入。

表和分区设计的基本规则

- 所有的表和字段名要使用统一的命名规范。命名要求如下：
  - 能区分该表的业务类型。
  - 能区分该表是事实表、维度表、日志表或极限存储表。
  - 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如下：
  - 支持新增分区，不支持新增分区字段。
  - 单表支持的分区数量上限为6万个。
  - 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

在计算的时候可以使用分区裁剪是分区的优势。分区设计需要关注如下内容：

- 分区字段和普通字段选择

  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。

  您可以在创建表时设置普通字段和分区字段。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。表的存储空间主要是普通字段占用的空间。设置分区字段时，您可以从数据管理和数据扫描方面考虑，来选择对应的字段。不具备规律、类型数量大于10000且不经常作为查询条件的字段，应该被设置成普通字段。

  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，在计算时如果指定具体的分区，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。因此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。

- 分区字段定义依据

  按优先级高低排序如下：

  - 分区列的选择应充分考虑时间因素，尽量避免更新存量分区。
  - 如果有多个事实表（不包括维度表）进行JOIN，应将作为查询条件的字段置为分区列。
  - 选择GROUP BY或DISTINCT包含的列作为分区列。
  - 选择值分布均匀的列，而不要选择数据倾斜的列作为分区列。
  - 常用SQL语句中如果包含某列的等值或IN的查询条件，则选择该列作为分区列。

    ```
    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

    如果对省、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的小时进行分区。
  - 单分区与多级分区

    在单分区下，建议每次提交64 MB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
  - 建议单个分区中的数据量不要太大。
  - 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。
  - 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执行任务时会进行分片处理而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性能的下降。
  - 采用多级分区时，建议先按日期分区，然后按交易类型分区。
  - 多种交易类型混合的表，建议您拆表，一种交易类型独立成一张表，然后每张表按日期分区。
  - 维度表不进行分区。

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，为实际开发提供指导和依据。

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。会产生大量小文件的操作如下：

- 使用MaxCompute Tunnel SDK上传数据时，每1次Commit会产生1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上传数据时，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（在服务侧等同于小文件）。
- 使用MaxCompute客户端执行Tunnel命令上传时，将本地大文件切分过小会导致上传后产生大量小文件。
- 通过DataHub执行数据归档，DataHub的每个Shard写入MaxCompute时存在条件限制，即数据总量达到64 MB就Commit 1次，或每隔5分钟Commit 1次，形成1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64 MB（例如几百KB），就会产生大量小文件。相应地，一天会产生2412×20=5760个小文件。
- 通过DataWorks等数据开发工具将数据增量插入（ `INSERT INTO` ）MaxCompute表（或表分区）时，每次进行数据增量插入都会产生1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1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合并全量极限表和增量数据表。
- 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持续高频地向MaxCompute中传入文件时，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最上层的对象。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，如果多个应用所需的数据是无关的，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。

维度表的设计

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，但是对单表个数有所限制。通常要求维度表的单表量不超过1000万个。

> 说明
>
> - 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。

拉链表的设计

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、订单的状态和手机号码等。
- 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。
- 变化的比例不大或频率不高。

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src_tbl。

```
CREATE TABLE src_tbl (key0 STRING, key1 STRING, col0 STRING, col1 STRING, col2 STRING) PARTITIO
N (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、批量数据写入和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表设计；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，相关采集通道应该进行有效区分。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过多导致采集通道流量较小、通道数量较多的问题。
- 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。
  - 通过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，所以通常需要按照终端设计多个明细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1 TB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64 MB提交1次）。
> - 日志数据很少会对原来分区执行更新操作，可以用INSERT操作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，需要采用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。

周期快照表中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。建议您建立一个事务性事实表，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，以降低资源的消耗。

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：

- 准备表

```
//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 t3.key1 IS NULL
 UNION ALL
//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，面积和基本人口数量等）和天气信息。
  - 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。

> 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。
> - 在插入数据或覆盖全表时，应尽量选用 `INSERT OVERWRITE` 而并非选择 `INSERT INTO` 。

2.4. MaxCompute表的高级功能

本文为您介绍MaxCompute表的生命周期、避免全表扫描、小文件以及Hash Clustering表等高级功能。

生命周期 全表扫描 小文件 Hash Clustering

生命周期

MaxCompute为表和分区提供数据生命周期管理功能。表（分区）数据从最后一次更新时间算起，在指定的时间段（即生命周期）内如果没有变动，则此表（分区）将被MaxCompute自动回收。生命周期只能以表为单位进行设置。

```
--创建表test_lifecycle，指定其生命周期为100天。
create table test_lifecycle(key string) lifecycle 100;
--修改表test_lifecycle的生命周期为50天。
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，判断是否要进行回收操作。

MaxCompute SQL提供TOUCH操作，将表或分区的LastDataModifiedTime值修改为当前时间。如果您使用TOUCH操作，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，举例如下。

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

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

避免全表扫描

- 在表设计时避免全表扫描。表设计是指建立分区表或者对扫描条件进行列设计，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，然后再扫描此中间表的一整年分区，从而减少扫描的数据量。

避免小文件

您可以参考如下建议避免产生小文件：

- Reduce计算过程产生的小文件：只需要Insert Overwrite源表（或分区）即可，或者写入到新表中再删除源表。

- Tunnel数据采集过程中产生小文件建议：
  - 调用Tunnel SDK时，每当缓存达到64MB时，进行一次提交。
  - 使用客户端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。
  - 使用Insert Overwrite语句对源表（或分区）进行操作。
  - 使用ALTER合并模式时，通过客户端命令进行合并。
- 建议为临时表设置生命周期，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，可以按照此数据分配实际的Shard数目（您也可以在此基础上多加几个）。
  - MaxCompute的每个Shard会有一个单独的Task，每隔5分钟或每满64MB，此Task会Commit一次，以提高在MaxCompute上查询数据的速度。当按照小时划分分区时，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MaxCompute里就会出现很多小文件。
  - 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值请选择重复键值少的列。

转化为Hash Clustering表的方法如下。

```
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，新的分区将进行Hash Clustering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。

> 注意 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。
> - 由于Tunnel方式上传的数据是无序的，因此不支持直接使用Tunnel上传数据到Range Cluster表。