

# 分析型数据库

## 使用手册

# 使用手册

## 前言：极速自由的大数据OLAP体验

随着企业IT和互联网系统的发展，越来越多的数据被产生了。数据的量的积累带来了质的飞跃，使得数据系统从业务系统的一部分演变得愈发独立，通过对数据的分析和挖掘产生自己独特的价值。在业务系统中，我们通常使用的是OLTP（OnLine Transaction Processing，联机事务处理）系统，如MySQL, MicroSoft SQL Server等关系数据库系统。这些关系数据库系统擅长事务处理，在数据操作中保持着很强的一致性和原子性，能够很好的支持频繁的数据插入和修改，但是，一旦需要进行计算的数据量过大，达到数千万甚至数十亿条，或需要进行的计算非常复杂的情况下，OLTP类数据库系统便力不从心了。

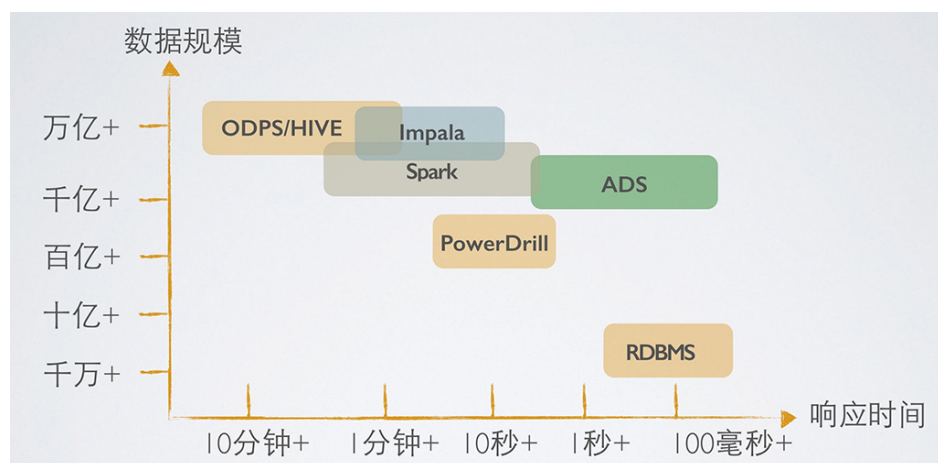


图0-1 分析型数据库和主流

数据系统的对比（数据未经严格测试，仅供参考）

这个时候，我们便需要OLAP（On-Line Analytical Processing，联机分析处理）系统，来进行处理。从广义上，OLAP系统是针对OLTP系统而言的，暨不特别关心对数据进行输入、修改等事务性处理，而是关心对已有的大量数据进行多维度的、复杂的分析的一类数据系统。而在具体的产品中，我们通常将OLAP系统分为MOLAP、ROLAP和HOLAP三种。多维OLAP（Multi-Dimensional OLAP，简称MOLAP），是预先根据数据需要分析的维度进行建模，在数据的物理存储层面以“立方体”（Cube）的结构进行存储，具有查询速度快等的优点，但是数据必须预先建模，无法依据使用者的意愿进行即时的修改。而关系型OLAP（Relational OLAP，简称ROLAP），则使用类似关系数据库的模型进行数据存储，通过类似SQL等语言进行查询和计算，优点是数据查询计算自由，可以灵活的根据使用者的要求进行分析，但是缺点是在海量数据的情况下分析计算缓慢。至于HOLAP，则是MOLAP和ROLAP的混合模式。而阿里云分析型数据库（原名：分析数据库服务ADS），则是一套RT-OLAP（Realtime OLAP，实时OLAP）系统。在数据存储模型上，采用自由灵活的关系模型存储，可以使用SQL进行自由灵活的计算分析，无需预先建模，而利用云计算技术，分析型数据库可以在处理百亿条甚至更多量级的数据上达到甚至超越MOLAP类系统的处理性能，真正实现百亿数据毫秒级计算。

阿里云分析型数据库让海量数据和实时与自由的计算可以兼得，实现了速度驱动的大数据商业变革。一方面，分析型数据库拥有快速处理百亿级别的大数据的能力，使得数据分析中使用的数据可以不再是抽样的，而是业务系统中产生的全量数据，使得数据分析的结果具有最大的代表性。而更重要的是，分析型数据库采用云计算技术，拥有强大的实时计算能力，通常可以在数百毫秒内完成十亿百亿的数据计算，使得使用者可以根据自己的想法在海量数据中自由的进行探索，而不是根据预先设定好的逻辑查看已有的数据报表。同时，由于分析型数据库能够支撑较高并发查询量，并且通过动态的多副本数据存储计算技术来保证较高的系统可用性，所以能够直接作为面向最终用户（End User）的产品（包括互联网产品和企业内部的分析产品）的后端系统。如淘宝数据魔方、淘宝指数、快的打车、阿里妈妈达摩盘（DMP）、淘宝美食频道等拥有数十万至上千万最终用户的互联网业务系统中，都使用了分析型数据库。分析型数据库作为海量数据下的实时计算系统，给使用者带来极速自由的大数据OLAP分析体验，最终期望为大数据行业带来巨大的变革。我们欢迎有各类相关需求的用户使用分析型数据库并向我们提出宝贵的建议。

# 第一章 快速开始

## 1.1 开通阿里云分析型数据库服务

在公共云上，满足开通条件的用户可以在 <https://buy.aliyun.com/ads> 上进行按量付费开通，或访问 [https://common-buy.aliyun.com/?commodityCode=prepaid\\_ads#/buy](https://common-buy.aliyun.com/?commodityCode=prepaid_ads#/buy) 购买包月套餐。

在专有云中，开通分析型数据库服务的方式请咨询您的系统管理员或运维人员。

## 1.2 创建数据库

分析型数据库中，需要通过DMS for Analytic DB页面进行创建数据库。

在目前的分析型数据库版本中，创建数据库时，需要填写数据库名，注意这个数据库名称需要在分析型数据库全部集群上全局唯一。然后选择分析型数据库的Region所在地，如杭州、北京等。

分析型数据库以ECU（弹性计算单元）作为资源计量的最小单位。ECU（弹性计算单元）拥有多种型号，每种型号的ECU，标识着不同的vCPU核数、内存大小、磁盘空间大小。用户在创建数据库时需要根据自己的需求选择这个数据库的ECU型号，以及初始的ECU数量（必须是偶数个，至少两个），ECU型号DB创建后不可修改，ECU数量可以在使用中随时调整（扩容/缩容），关于ECU的详细信息，详见 2.4节 ECU详解。

填好所有选项后，点击创建数据库，若返回错误，则根据错误提示进行修正（通常是数据库名称重复或不符合规范，或提交的ECU资源量超过了分析型数据库允许的最大限制），否则则创建成功。十分钟以内DMS界面中会显示出新的数据库的连接地址。

## 1.3 使用DMS创建和管理表

前文中，我们已经创建了一个分析型数据库数据库，分析型数据库采用关系模型存储数据，也就是使用二维表来进行数据的组织和存储。像MySQL一样，将数据灌入分析型数据库前需要需要建立对应的数据表。而分析型数据库为了管理相关联的数据表，又引入了表组的概念。

表组是数据库的下一级实体，也是表的上一级。在分析型数据库中一个表必须从属于一个表组。关于表组的具体介绍我们会在3.2节中进行。在这里，我们首先创建一个表组。

在DMS for 分析型数据库中，右击左侧表组对象，选择新建表组，弹出新建表组对话框，弹出如下图所示内容，我们填写表组名为test\_group，其余参数先暂时使用默认值。

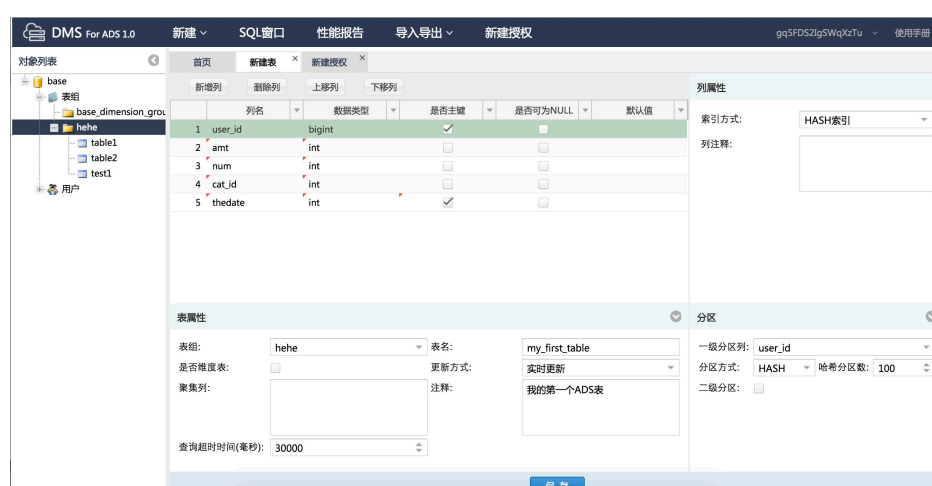


新建表组对话框，包含以下字段：

- 表组名: test\_group
- 最小副本数: 2
- 超时时间(毫秒): 30000

底部有提交和取消按钮。

点击确定建立好表组后，我们右击表组，选择新建表。在这里，我们根据测试数据的情况，建立一张有五列和一级分区的数据表。如下图所示：



DMS新建表界面，显示了表属性、分区、列属性等信息。

表属性：

- 表组: hehe
- 表名: my\_first\_table
- 是否维度表: ☐
- 更新方式: 实时更新
- 查询超时时间(毫秒): 30000

分区：

- 一级分区列: user\_id
- 分区方式: HASH
- 哈希分区数: 100
- 二级分区: ☐

列属性：

| 列名        | 数据类型   | 是否主键                                | 是否可为NULL                 | 默认值 |
|-----------|--------|-------------------------------------|--------------------------|-----|
| 1 user_id | bigint | <input checked="" type="checkbox"/> | <input type="checkbox"/> |     |
| 2 amt     | int    | <input type="checkbox"/>            | <input type="checkbox"/> |     |
| 3 num     | int    | <input type="checkbox"/>            | <input type="checkbox"/> |     |
| 4 cat_id  | int    | <input type="checkbox"/>            | <input type="checkbox"/> |     |
| 5 thedate | int    | <input checked="" type="checkbox"/> | <input type="checkbox"/> |     |

索引方式: HASH索引

列注释:

在分析型数据库中，数据根据分区列进行分布式的存储和计算。举例来讲，我们在这里的原始数据是按照user\_id较均匀的进行分布的，所以我们指定按照user\_id进行hash分区，分区数调整为40个（一般来讲，每个

分区的数据不超过800万条为宜，当然也不绝对，分区数不能超过256个）。表名和列名根据实际情况填写，目前需要和源头数据表的字段名称一致。

另外，如果这个表的数据来源是批量的从其他系统导入（例如从ODPS），那么在更新方式一项，则选择批量更新，随后阅读1.4节中的导入数据部分。如果这个表的数据来源来自于直接的insert插入，那么在更新方式一项选择实时更新，随后阅读1.4节中的插入数据一项。

分析型数据库拥有强大的自动索引功能，用户在创建表时通常无需关心一个列的索引情况，分析型数据库会根据实际数据的分布情况来自动进行索引。所以这里我们先不调整列的索引设置。而表名任意填写，表组名选择我们刚刚创建的test\_group，然后点击保存，弹出实际的建表DDL供校验。

有关表和列的详细说明，我们会在3.3和3.4节中稍后叙述。创建表完毕后，右击已有的表可以进行编辑。

## 1.4 导入数据

分析型数据库支持多种接入数据的方式，您可以直接将数据通过insert/delete SQL写入实时表（详见使用手册第四章），或通过Kettle等ETL工具将本地文件写入分析型数据库，或是通过阿里云数据传输从阿里云RDS中实时同步数据变更（见使用手册8.5节），或者建立批量导入表从阿里云MaxCompute（原名ODPS）大批量的导入数据。

如果在建立表时选择数据来源是批量导入，则分析型数据库提供多种数据导入的方式，如通过data pipeline系列命令（详见5.1），等方式。在这里，作为测试使用，我们通过控制台界面进行数据导入。

在操作导入数据之前，我们需要对数据的来源表进行授权，例如数据的来源表在odps上，在公有云上则需要ODPS上对 garuda\_build@aliyun.com 授予describe和select权限（各个专有云授权的账号名参照专有云的相关配置文档，不一定是这个账号）。另外要注意，分析型数据库目前仅允许操作者导入自身为Project Owner的ODPS Project中，或者操作者是ODPS表的Table Creator的数据。

进入DMS页面，选择菜单栏上的导入按钮，弹出导入对话框。这里我们的数据源表在阿里云ODPS上。因此数据导入路径按照“odps://project\_name/table\_name/partition\_spec”的格式来填写。关于导入数据的分区信息，在仅有Hash分区的情况下iDB Cloud会帮我们自动识别并填写。填写完毕后，如下图所示，点击“确定”按钮。

### 导入数据

数据源：

odps

是否覆盖：

否

数据源路径：

odps://ads\_test\_project/my\_first\_table/pt=2014

目标表：

my\_first\_table

一级分区列：

USER\_ID

填写版本号：

☐ 是 ☒ 否

确定

关闭

之后页面会展示导入状态一览，分析型数据库会对导入任务进行调度，根据当前系统繁忙情况和待导入数据的大小和结构不同，二十分钟至数个小时内数据导入会结束。

如果建表时数据来源选择的是实时写入，那么则可以在建表后直接在SQL窗口中编写SQL:

```
insert into my_first_table (user_id,amt,num,cat_id,theDate) values (12345, 80, 900, 1555, 20140101);
```

实时写入的表刚刚建立后，会有一分钟左右（公共云当前版本：0.9.\*版本数据，若使用0.8版本的ADS一般为十五分钟左右）的准备时间，这时候写入的数据需要在准备完成后才能查询，否则查询会报错。在准备完成后，实时进行insert/delete的数据变更一般延迟一分钟内可查。

需要注意的是，分析型数据库进行实时插入和删除时，不支持事务，并且仅遵循最终一致性的设计，所以分析型数据库并不能作为OLTP系统使用。

若表的数据是离线批量导入，但是数据源是RDS等其它的云上系统，那么我们可以通过阿里云的CDP产品进行数据同步，在 <http://www.aliyun.com/product/cdp/> 上开通CDP（可能需要申请公测）后，按照如下示例配置同步Job：

```
{
  "type": "job",
  "traceId": "rds to ads job test",
  "version": "1.0",
  "configuration": {
    "setting": {
    },
    "reader": {
      "plugin": "mysql",
```

```
"parameter": {
  "instanceName": "你的RDS的实例名",
  "database": "RDS数据库名",
  "table": "RDS表名",
  "splitPk": "任意一个列的名字",
  "username": "RDS用户名",
  "password": "RDS密码",
  "column": ["*"],
}
},
"writer": {
  "plugin": "ads",
  "parameter": {
    "url": "在分析型数据库的控制台中选择数据库时提供的连接信息",
    "schema": "分析型数据库数据库名",
    "table": "分析型数据库表名",
    "username": "你的access key id",
    "password": "你的access key secret",
    "partition": "",
    "lifeCycle": 2,
    "overWrite": true
  }
}
}
```

需要注意的是，在运行这个任务之前，需要在分析型数据库中对cloud-data-pipeline@aliyun-inner.com 至少授予表的Load Data权限。

若表是实时写入的，但是仍需要通过数据集成CDP从其他数据源导入到分析型数据库中，则>上述reader配置不变，但是writer配置需要变为（以原表列和目标表列相同为例，若不同需要在column选项中配置Mapping）：

```
"writer": {
  "name": "adswriter",
  "parameter": {
    "writeMode": "insert",
    "url": "在分析型数据库的控制台中选择数据库时提供的连接信息",
    "schema": "分析型数据库数据库名",
    "table": "分析型数据库表名",
    "column": ["*"],
    "username": "你的access key id",
    "password": "你的access key secret",
    "partition": "id,ds=2015"
  }
}
```

## 1.5 在应用中连接和使用分析型数据库

首次成功导入数据到分析型数据库后，我们便希望我们的应用系统能够连接到分析型数据库来进行数据查询。

分析型数据库可以通过任何支持 5.1.x 5.4.x 5.6.x协议的客户端进行连接。连接所使用的域名和端口号可以在 iDB Cloud的右上角进行查看。连接使用的用户名和密码为用户在阿里云的Access Key，可以在 [https://i.aliyun.com/access\\_key/](https://i.aliyun.com/access_key/) 查看和管理。其中Access Key ID为用户名，Access Key Secret为密码（分析型数据库承诺不会保存用户的Access Key信息）。

若需要使用阿里云访问控制(RAM)子账号连接分析型数据库，请参阅使用手册的8.6节。

## 在PHP中连接分析型数据库

在PHP环境下，假设我们已经安装好了php-mysql 5.1.x模块（Windows下为php\_MySQL.dll），那么我们新建一个ads\_conn.php，内容如下：

```
$ads_server_name="mydbname-xxxx.ads-cn-hangzhou-1.aliyuncs.com "; //数据库的连接url，请在控制台中的连接信息中获取
$ads_username="my_access_key_id"; // 连接数据库用户名
$ads_password="my_access_key_secret"; // 连接数据库密码
$ads_database="my_ads_db"; // 数据库的名字
$ads_port=3003; //数据库的端口号，请在控制台中的连接信息中获取
// 连接到数据库
$ads_conn=mysqli_connect($ads_server_name, $ads_username, $ads_password, $ads_database, $ads_port);
```

执行查询时，可以使用：

```
$strsql="SELECT user_id FROM my_ads_db.my_first_table limit 20;"; $result=mysqli_query($ads_conn, $strsql);

while($row = mysqli_fetch_array($result)) {
    echo $row["user_id"]; //user_id为列名
}
```

上述代码即可取出任意十条记录的user\_id并打印出。注意分析型数据库在数据查询中是不支持SELECT \*方式查询所有列的。

## 在JAVA中连接分析型数据库

通常，在JAVA中，我们通过连接池来使用分析型数据库。在这里我们以国产的高性能连接池Druid为例来演示连接分析型数据库的方式。

```
import com.alibaba.druid.pool.*;
DruidDataSource dataSource = new DruidDataSource();
dataSource.setDriverClassName("com.mysql.jdbc.Driver");
dataSource.setUsername("my_access_key_id");
dataSource.setPassword("my_access_key_secret");
dataSource.setUrl("jdbc:mysql://mydbname-xxxx.ads-hz.aliyuncs.com:5544/my_ads_db");
// 连接数配置
dataSource.setInitialSize(5);
dataSource.setMinIdle(1);
dataSource.setMaxActive(10);
// 启用监控统计功能
dataSource.setFilters("stat");
```

```
// for mysql
dataSource.setPoolPreparedStatements(false);
// 使用心跳语句检测空闲连接
dataSource.setValidationQuery('show status like "%Service_Status%";');
dataSource.setTestWhileIdle(true);
```

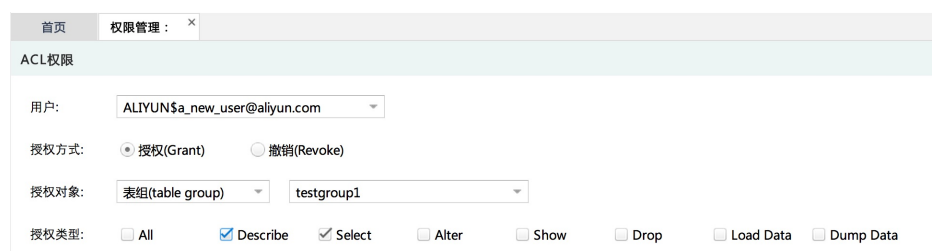
如上，需要注意的是，若是在任何语言中需要使用心跳SQL来进行分析型数据库服务状态检测，请使用 `show status like "%Service_Status%"` 语句，若返回一行两列且第二列为1，则分析型数据库服务正常。

## 1.6 新增用户和授权

在分析型数据库中，创建一个数据库的账号为这个数据库的owner账号，拥有最大化的权限。在很多情况下，我们不希望这个账号被更多的人掌握，所以需要授权其他人仅使用较小的权限连接分析型数据库。这时我们便需要进行权限管理，完整的权限体系我们将会在第六章详细描述，这里先给出一个简单的例子。

在DMS中，左侧的对象列表中，点击用户可以看到可以访问该数据库的全部用户列表。若要新增一个授权，则我们点击菜单栏上的“权限”按钮。

在打开的窗口中，我们输入待授权的用户账号，注意由于分析型数据库将会支持多种账号来源，所以需要在输入账号时声明账号来源，例如 `ALIYUN$a_new_user@aliyun.com` 前缀代表该账号是阿里云账号。我们输入一个待授权的云账号例如 `ALIYUN$a_new_user@aliyun.com`。然后假设我们希望这个账号对testgroup1表组下的所有表有查看元信息和只读权限，那么我们进行如下选项操作：



ACL权限

用户:

授权方式: ☒ 授权(Grant) ☐ 撤销(Revoke)

授权对象:

授权类型: ☐ All ☒ Describe ☒ Select ☐ Alter ☐ Show ☐ Drop ☐ Load Data ☐ Dump Data

然后点击保存，完成授权操作。随后 `a_new_user@aliyun.com` 该用户便可使用自己的Access Key登录分析型数据库执行有权限的操作了。

## 第二章 分析型数据库概览

### 2.1 数据库、表组和表

在传统的关系数据库系统中，表直接隶属于某个数据库。而在分析型数据库中，为了方便的进行数据的关联的管理，以及进行资源分配，我们引入了表组的概念。并且在分析型数据库中数据库、表组、表的概念有其独特

的地方。

在分析型数据库中，数据库是用户所关心的最大单元，也是用户和分析型数据库系统管理员的管理职权的分界点。分析型数据库系统管理员最小可管理数据库粒度的参数，而无法未经用户授权来查看和管理数据库内部的结构和信息。而数据库级别的参数对于用户来讲大多只能查看而不能修改。在分析型数据库中，一个数据库对应着一个用于访问的域名和端口号，并且有且只有一个owner暨为数据库的创建者。另外对用户的宏观的资源配置是以数据库为粒度进行的，因此创建数据库时用户需要输入业务预估的QPS、数据量、Query类型等信息用于智能的判断初始的资源分配。

表组是一系列可发生关联的数据表的集合。分析型数据库中表组分为两类：普通表组和维度表组。其中维度表组用于存放维度表（特征上是一种数据量较小但是需要和任何表进行关联的表），目前有且仅有一个，并且在分析型数据库数据库建立时会自动创建，用户不可修改和删除。而普通表组有如下特征：

- 表组是数据物理分配的最小单元，数据的物理分布情况通常无需用户关心，但是因此数据的副本数（指数据在分析型数据库中同时存在的份数）必须在表组上进行设定，一个表组所有表副本数一致。
- 一个表组内的表才能够进行快速的hash join。0.9版本之前分析型数据库对非维度表仅支持同表组内的表进行Join操作（维度表除外），0.9.15版本后开启Full MPP Mode（目前为邀请测试功能，后续全面开放）支持非相同表组内的表进行Join关联，但是性能较差。
- 一个表组的表可以共享一些配置，例如查询超时时间，如果表组中的单表也进行了这些配置的个性化，那么进行表关联时会通表组级别的配置进行覆盖。
- 一个表组中的所有表的一级Hash分区数建议一致（非强制）。

分析型数据库中的表分为普通表和维度表。普通表创建时需要指定至少一级分区列和相关分区信息，并且需要指定存放在一个表组中。维度表可以和任意表组的任意表进行关联，并且创建时不需要配置分区信息，但是对单表数据量大小有所限制，并且需要消耗更多的存储资源。

在分析型数据库中，普通表的分区目前最多可以有两级。分区种类有Hash分区和List分区两种，其中Hash分区是根据导入数据时已有的一列的内容进行散列后进行分区的，因此目前多张事实表进行快速的Hash Join时Join Key必须有分区列参与，并且这些表的Hash分区数必须一致；0.9版本之前，不支持事实表在没有Hash分区键参与关联条件的情况下进行关联，0.9.15版本后，通过Full MPP Mode（目前为邀请测试功能，后续全面开放）或小表广播进行计算时无此限制，关于join的详细内容请看4.4节。

另外仅采用Hash分区的数据表，在数据装载时，是进行全量覆盖历史数据的。而List分区则是根据进行导入操作时所填写的分区列值来进行分区的，暨一次导入的数据会进入同一个List分区中，因此List分区是支持增量的数据导入的。目前分析型数据库支持将表的一级分区设置为Hash分区，二级分区设置为List分区，即可支持Hash join又可支持增量数据导入。另外无论采用何种分区形态，分析型数据库均不需要在用户查询时指定分区列，但是指定分区列或分区列的范围进行查询可能会提高查询性能。

另外根据表的更新方式不同，分析型数据库的表分为离线批量更新的表（适合从离线系统如ODPS产出的数据结果导入到分析型数据库供在线系统使用），以及实时更新的表（可以直接insert/delete单条数据，适合业务系统直接写入数据），实时更新的表不提供二级分区功能（因为天生支持增量，无需二级分区）。

另外请注意，分析型数据库不支持读写事务，并且数据实时更新时一分钟左右才可查询，另外在一致性方面分析型数据库遵循最终一致性。

## 2.2 数据类型支持

### 1、分析型数据库支持的数据类型

**boolean** 布尔类型，值只能是0或1；取值0的逻辑意义为假，取值1的逻辑意义为真；存储字节数1比特位

**tinyint** 微整数类型，取值范围-128到127；存储字节数1字节

**smallint** 整数类型，取值范围-32768到32767；存储字节数2字节

**int** 整数类型，取值范围-2147483648到2147483647；存储字节数4字节

**bigint** 大整数类型，取值范围-9223372036854775808到9223372036854775807; 存储字节数8字节

**float** 单精度浮点数，取值范围-3.402823466E+38到-1.175494351E-38, 0, 1.175494351E-38到3.402823466E+38, IEEE标准; 存储字节数4字节

**double** 双精度浮点数，取值范围-1.7976931348623157E+308到-2.2250738585072014E-308, 0, 2.2250738585072014E-308 到 1.7976931348623157E+308. IEEE标准; 存储字节数4字节

**varchar** 变长字符串类型

**date** 日期类型，取值范围：'1000-01-01' 到 '9999-12-31', 支持的数据格式为'YYYY-MM-DD' 存储字节数为4字节

**timestamp** 时间戳类型，取值范围：1970-01-01 00:00:01.000' UTC 到 '2038-01-19 03:14:07.999' UTC., 支持的的数据格式为：'YYYY-MM-DD HH:MM:SS' 存储字节数为4字节

**multivalue** 多值列类型，即一个cell中包含多个值，多个值直接的分隔符默认是, 当然也可以通过delimiter关键字指定。

### 2、分析型数据库数据类型和mysql数据类型对比

注：分析型数据库所有的数据类型都不支持unsigned，下表不包含该差异

| 分析型数据库数据类型 | MySQL数据类型 | 差异 |
|------------|-----------|----|
|------------|-----------|----|

|            |               |                               |
|------------|---------------|-------------------------------|
| boolean    | bool,boolean  | 一致                            |
| tinyint    | tinyint       | 一致                            |
| smallint   | smallint      | 一致                            |
| int        | int,integer   | 一致                            |
| bigint     | bigint        | 一致                            |
| float      | float[(m,d)]  | 分析型数据库不支持自定义m和d，而mysql可以      |
| double     | double[(m,d)] | 分析型数据库不支持自定义m和d，而mysql可以      |
| varchar    | varchar       | 一致                            |
| date       | date          | 一致                            |
| timestamp  | timestamp     | 分析型数据库只支持到精确到毫秒，而mysql是可以定经度的 |
| multivalue | -             | 分析型数据库特有的，MYSQL无此类型           |

## 2.3 分析型数据库特色功能

作为创新型的实时OLAP服务，分析型数据库提供很多独有的特色功能，这里便就其中常用的部分进行简要的介绍。

### 聚集列

在创建表时，用户可以指定一列或者若干列作为聚集列。在物理上，一个分区内聚集列内容相同的数据会尽可能的分布在同样的区块内存放，因此如果用户的查询Query的条件中会指定聚集列的内容或范围，那么这样的查询性能便会有较大的提升。需要注意的是如果用户指定多列为聚集列，那么指定的聚集列的顺序便是比较数据是否相同的顺序。

聚集列可以在建表后进行修改，但是目前修改后下次装载数据完毕后生效。

### 多值列

分析型数据库拥有一个特殊的数据类型，叫做多值列。顾名思义，多值列中可以存入String类型的多个值。在原始数据中多值列为一列用分隔符（默认为半角逗号，可以建表时进行配置）分隔的String类型。多值列数据存入分析型数据库后，可使用in, contains条件对该列的单个值进行查询，枚举查询后该列的每个值可像一个普通列一样进行各类操作。但是不允许在没有进行枚举查询时对该列直接select或在group by中使用该列。

多值列的通常使用场景是，已有一个实体属性表均为普通列并以实体编号为主键的情况下，需要新增一个用于进行实体筛选的属性，而这个属性和实体编号为多对多的对应关系，按照通常的做法，则新建一张该实体编号和属性两列的数据表，和原有的实体属性表进行Join操作查询。而使用多值列后，性能会被进行Join操作计算高数倍。一个具体的例子：一个张用户表中已有用户id和性别、年龄、职业、籍贯的数据，而现在需要增加用

户购买的商品品类的数据用于筛选，于是可以将这个商品的品类id存放在用户表的一个新增的多值列中使用。

## 智能自动索引

分析型数据库拥有智能的自动索引创建机制，通常情况下，分析型数据库会根据导入数据中每一列的分布情况，如该列的枚举值数量的多少，该列的数据是连续的或离散的，自动为导入数据的每一列创建符合该列情况的索引类型，无需用户显式指定创建索引或索引类型。而在有些情况下，用户如果认为某一列不需要进行筛选查询，可以指定该列不需要创建索引来节省数据存储空间。另外，0.9版本之前如果一个列需要作为Hash Join的Join Key，那么用户需要人工指定该列创建Hash索引，0.9版本（公共云当前版本）已废弃Hash索引，可动态根据Join SQL进行优化，完全无需用户干预。

## 查询CBO优化

分析型数据库拥有高度智能的CBO(Cost-Based Optimization)优化策略。在用户发起的一个Query达到分析型数据库后，分析型数据库会智能的判断该Query涉及的数据的分布情况、索引使用情况、缓存使用情况、Query条件分布等，进行逻辑和物理执行计划优化和重组，尽可能的以最优的路径执行Query查询。因此用户大部分情况下无需关心Query的具体写法，只要保证语义正确即可。另外分析型数据库提供一些Hint参数，可以在Query时对执行计划进行部分调整，详见4.4节。

## 2.4 ECU详解

ECU（弹性计算单元），是分析型数据库中存储和计算资源的分配单位。

分析型数据库对每个用户的每一个DB会分配若干个计算节点（COMPUTENODE），以及若干个接入节点（BUFFERNODE），接入节点用于接收用户的应用前端连接等工作，计算节点用于存储用户的数据和进行计算，另外还有若干个用于放置实时化数据写入缓冲的缓冲节点（BUFFERNODE）。目前分析型数据库仅计算节点是用户可按ECU模式配置，分析型数据库会自动根据用户的计算节点的量来配置接入节点等其它角色的数量。

计算节点的ECU具有如下属性：

- 内存容量：该ECU的内存大小
- 磁盘容量：该ECU的磁盘容量，用户在一个DB中存储的物理数据总量不能超过该DB的全部的磁盘容量，并且由于分析型数据库将用户的数据分布到每一个ECU中，若用户的数据倾斜导致单个ECU的磁盘空间占满，也会导致数据无法再进入分析型数据库。各个ECU的磁盘使用情况可以在云监控（<http://cms.console.aliyun.com/>）中查看

目前分析型数据库公共云提供的ECU规格为（专有云参照此标准灵活执行）：

| 型号 | 内存    | 磁盘类型 | 磁盘容量(SSD) | 磁盘容量(SATA) |
|----|-------|------|-----------|------------|
| c1 | 7.5GB | SSD  | 60GB      |            |
| c8 | 45GB  | SSD  | 480GB     |            |

0.9版本的分析型数据库，提供基于SATA存储的大容量实例（目前为邀请测试功能，后续开放购买），采用SATA和SSD混合存储，能够大幅度降低存储成本，但是同时查询性能也以数量级而下降。大容量实例的ECU型

号通常以字母s开头。专有云中原则上仅万兆网物理机能够运行大容量实例。

ECU数量，可以通过DMS for 分析型数据库界面的扩容/缩容功能，或相应DDL动态修改。

## 第三章 DDL

### 3.1 创建数据库与配置

目前，公共云计算上分析型数据库的用户不能直接通过CREATE DATABASE的DDL语句创建数据库，只能通过DMS控制台界面来创建需要的业务数据库。

用户需要根据自己的业务需求来估算使用分析型数据库时所需的查询类型等配置参数，并在DMS的界面上进行填写。由于已在1.2节中叙述过相关流程，所以再次不再叙述。

专有云中若需要创建数据库，请联系您的运维管理员或DBA。

### 3.2 创建和修改表组

在第一章中，我们已经介绍了如何通过DMS界面创建和修改表组，本节中，我们详细描述如何通过DDL来创建和修改表组以及表组各属性的含义。

创建表组时，我们提交如下SQL：

```
create tablegroup db_name.tablegroup_name options(minRedundancy=2 executeTimeout=30000);
```

其中db\_name为数据库名称，tablegroup\_name为表组名称（不能和现有表组重叠）。

options部分为可选项：minRedundancy表示该表组的副本数，默认为2，可配置为1、2、4、8。需要注意的是，如果将一个表组配置为1副本，那么这个表组中的表在数据导入时会有不可用的时间。而将表组副本数配置为4或更高，可以一定程度的增加分析型数据库的最大承受的QPS，但是数据存储费用也会相应增加。在绝大部分情况下，不推荐修改任何表组默认配置。

executeTimeout表示该表组的全局Query超时时间，默认为30000，单位毫秒。

当我们需要删除一个表组时，我们可以提交如下SQL：

```
drop tablegroup db_name.tablegroup_name;
```

注意，仅允许删除没有任何表的空表组，维度表组不允许删除。

当我们需要修改表组的两个属性中的任何一个时，我们可以提交如下SQL：

```
alter tablegroup db_name.tablegroup_name key=value;
```

其中key为属性名，value为新的属性值。需要注意的是，minRedundancy修改后需要下次装载数据时才会生效。

## 3.3 创建和修改表

在快速开始中，我们已经大概了解了在如何通过iDB Cloud创建和修改表。在这一节中，我们将详细描述如何通过DDL创建和修改表。

由于分析型数据库有一些传统关系型数据库没有的特性，所以分析型数据库的DDL在遵循类MySQL规范的基础上，有不少的独有的属性和语法。分析型数据库中创建事实表DDL语法结构如下：

```
CREATE TABLE db_name.table_name (  
  col1 bigint COMMENT 'col1',  
  col2 varchar COMMENT 'col2',  
  col3 int COMMENT 'col3',  
  col4 bigint COMMENT 'col4',  
  col5 multivalue COMMENT 'col5 多值列',  
  [primary key (col1, col3)]  
)  
PARTITION BY HASH KEY(col1)  
PARTITION NUM 50  
[SUBPARTITION BY LIST (part_col2 long)]  
[SUBPARTITION OPTIONS (available_partition_num = 30)]  
[CLUSTERED BY (col3,col4)]  
TABLEGROUP ads_test ;  
[options (updateType='{realtime | batch}')]
```

其中‘[]’中的内容为视情况填写的可选项。创建事实表（也就是非维度表）时，必须指定的是一个数据库名（db\_name）和表名（table\_name），以及至少一列的列信息，至少一个分区的信息以及一个表组。

首先，根据表的数据更新方式不同，分析型数据库的表根据updateType分为批量更新表（仅能够离线批量更新数据）和实时更新表（能够通过insert/delete实时更新数据），用updateType以区分，如果updateType选项不填则默认为批量更新表。需要注意的是，updateType=realtime暨为实时更新表时，必须指定合法的主键并且不能有二级分区。

列信息的基本格式为“列名 类型 COMMENT ‘注释’ ”，关于列的更多属性，请参照后文3.4节。” [primary key (col1, col3)]”一段指定了表的主键，如果表为批量更新的，则主键没有意义，而表为实时更新的，则必须指定主键并且主键中必须含有一级Hash分区列（可以是联合主键）。

关于分区方面，目前分析型数据库支持最多两级分区，并且一级分区仅支持HASH分区，二级分区仅支持LIST分区。HASH分区是一种动态分区值类型，暨根据实际数据中的某一列的内容进行分区。所以在语法上，一级HASH分区的用法是：

```
PARTITION BY HASH KEY(col1)
PARTITION NUM 50
```

其中col1为需要进行分区的列名，必须是表中实际存在的列。‘PARTITION NUM’为分区数量，一般根据该表的数据量确定，每个分区一般不超过1500万条记录为宜（亦可通过划分二级分区实现无限扩展）。另外HASH分区列的数据分布要尽可能均匀，不能有非常明显的倾斜（暨分区列值），否则会较严重的影响查询性能。

一定需要注意的一点：目前分析型数据库要求一个表组下所有表的一级分区数目一致，所以建立第一张表时指定一级分区数量时请谨慎。一级分区数量默认不能超过256个。

另外需要注意的是，若一张表仅有一级HASH分区并且是批量导入的表，则每次导入数据时会对已有进行全量覆盖。若需要每次导入数据时增量导入，则需要再指定二级List分区信息：

```
SUBPARTITION BY LIST (part_col2 bigint)
SUBPARTITION OPTIONS (available_partition_num = 30)
```

二级List分区为非动态分区，暨分区值不是由数据本身决定的，而是由每次导入/写入数据时用户指定的。所以在进行分区信息定义时需要指定一个和现有数据中的列不同的新列名，以及这个列的类型（目前仅支持long）。二级分区有一个可选属性，available\_partition\_num，即为最大保留的二级分区数，当新的数据装载进来后，若线上存在的二级分区数大于这个值，那么会根据二级分区的值进行排序，下线最小的若干分区的数据。

0.8版本的实时更新表暂不支持二级分区，0.9版本支持。但是实时更新表的二级分区仅用于极大的扩展单表容量，以及进行生命周期管理，实时更新表的增量更新不依赖于二级分区。

惯常的用法是，将经常需要进行Join的列（例如买家ID）作为一级Hash分区列，而将日期列作为二级分区列。这样的表既可以进行大表Join的加速，又可以每天进行增量数据导入，并且指定保留若干天的数据在线上来进行生命周期管理。

但是，一级分区数量和二级分区数量的设置，并不是越多越好的。而是要看表的数据量，以及数据库拥有的资源数。若一级二级分区过多而数据库的资源数过少，则很容易让分区的数据Meta将内存耗尽。

CLUSTERED BY子句用于指定聚集列，用户可以把一列或者多列指定为聚集列，注意如果指定多列，那么该表的数据聚集顺序按照DDL中这个子句中指定的列组合顺序进行排序。通常，我们将一个表的查询中肯定会涉及到的并且数据区分度很大的列设置为聚集列，有时候能较显著的提升查询性能。

事实表的创建上，默认有如下限制：（1）一张事实表至少有一级Hash分区并且分区数不能小于8个；（2）一个事实表组最多可以创建256个事实表；（3）一个事实表最多不能超过1024个列。

与创建事实表相比，创建维度表要简单的多：

```
CREATE DIMENSION TABLE db_name.dim_table_name (
  col1 int comment 'col1',
  col2 varchar comment 'col2'
  [primary key (col1)]
)
[options (updateType='{realtime | batch}')]
```

```
;
```

创建维度表时只需要指定数据库名和表名即可，维度表会创建到统一的维度表组，并且无需指定分区信息。

在表已经创建好之后，目前支持有限的修改：主要是支持增加列。

```
ALTER TABLE db_name.table_name ADD column col_new varchar;
```

注意，批量导入表新增加的列在数据重新装载后才会生效，实时写入表新增列后，一般需要几分钟后可以读写。

最大二级分区数目前可以在建表后进行修改，但是修改后的下一次数据导入发起后才会生效：

```
ALTER TABLE db_name.table_name subpartition_available_partition_num = N
```

N为新的二级分区数。

## 3.4 列的类型和属性

在DDL中，建表时一列的定义是：

```
col_name type [NOT NULL | NULL] [DEFAULT default_value] [PRIMARY KEY] [COMMENT 'string']  
[column_options(precision, scale, disableIndex...)]
```

其中col\_name为列名，type为列的数据类型，详见2.3节。[NOT NULL | NULL]是否可为空，以及[DEFAULT default\_value]定义的列的默认值和标准的MySQL DDL中无甚不同。

关于[PRIMARY KEY] 主键部分，对于批量更新表，分析型数据库中主键的概念是弱化的，分析型数据库不要求一个表有主键，有主键的表的性能和用法上和没有主键的表之间没有任何区别。若一个表进行数据导入时该次导入的数据中存在主键冲突，则该次导入会失败并且报错。对于实时更新表，请使用在所有列尾部的"primary key (col1, col3)"语法指定主键。

列属性上，一个列可以设置列属性disableIndex = true，用于屏蔽分析型数据库的默认列索引，不过需要注意的是，要如此做，则这个列应该不在实际查询中所筛选和计算的。precision和scale属性则是针对decimal数据类型（目前暂未上线，未来会上线该功能）特有的属性，precision为数字整体有效数字个数，scale为小数点后的数字个数。

## 3.5 索引

同主键一样，分析型数据库中索引的概念也是弱化的。在前文中介绍分析型数据库拥有高度智能的自动化索引机制，所以通常用户无需亲自为自己的数据表配置索引。但是有一种情况例外：暨0.8版本下用户需要对某列进行Hash Join时，无论是事实表之间的Join还是事实表和维度表的Join，都需要为事实表的该列建立索引（公共

云当前版本无需 )。

建立索引的语句如下：

```
ALTER TABLE tbl_name ADD INDEX [index_name] [index_type] (index_col_name) [comment=' '];
```

其中，[index\_type] 为索引类型，需要指定为HashMap，index\_col\_name 为被索引列的列名。例如：

```
ALTER TABLE db_name.table_name  
ADD INDEX user_id_index HashMap (user_id)
```

批量导入表索引修改后，需要重新导入数据后生效。实时写入表一般在24小时内自动生效，或如果要加速这个过程，可以执行optimize table 命令：

```
optimize table <tablename>;
```

执行成功一段时间后，新的索引会生效。

0.9及以后版本中分析型数据库会自动处理Hash Join时的索引构建，故无需用户自行创建HashMap索引也可以进行Hash Join，因此HashMap索引被废弃。

## 3.6 ECU管理

在DMS for 分析型数据库中，用户可以在图形界面上进行扩容/缩容，以及查看扩容/缩容执行状态等操作(实例管理->DB容量管理->容量变更 按钮)。同时，用户可以通过DDL来进行ECU变更，也就是扩容/缩容（需要Alter Database权限或DB owner角色）。

```
ALTER DATABASE SET ecu_count = N;
```

N为要设置的目标ECU数量，若目标ECU数量大于目前当前的ECU数量，则为扩容行为，若目标ECU数量小于当前的ECU数量，则为缩容行为。若用户的数据量大于目标ECU的总存储，则缩容会失败。

缩容和扩容都不是瞬时的同步操作，可以使用元数据查询状态：

```
select * from information_schema.resource_request;
```

查看目前已有的ECU状态：

```
select * from information_schema.current_instances;
```

## 第四章 DML

### 4.1 SELECT语句基本语法

通过DML进行计算是用户在分析型数据库中最常用的操作。目前分析型数据库对DML支持使用SELECT进行查询，以及支持INSERT、DELETE等数据修改操作（INSERT/DELETE见4.3节）。另外本节的内容需要配合附录一：函数的使用与4.2中特殊语法，以及4.4中的双计算引擎和Hint的说明来阅读。

分析型数据库中SELECT语句的基本结构如下：

```
SELECT
[DISTINCT]
select_expr [, select_expr ...]
[FROM table_references
[WHERE filter_condition]
[GROUP BY {col_name | expr | position}
[HAVING having_condition]
[ORDER BY {col_name | expr | position}
[ASC | DESC], ...]
[UNION ALL (SELECT select_expr..)]
[UNION|INTERSECT|MINUS] (SELECT select_expr..)]
[LIMIT {row_count}]
```

下面，我们将一个完整的SELECT分为几个部分加以介绍：列投射部分（列和列表表达式）、FROM/JOIN子句（表扫描、表连接与子查询）、WHERE子句（过滤表达式）、GROUP BY/HAVING子句（分组和分组后过滤）、ORDER BY子句（排序）、两个查询间的UNION [ALL]/INTERSECT/MINUS、LIMIT子句（返回行数限制）。

#### 列投射（列和列表表达式）

分析型数据库中，SELECT语句中的列投射的基本结构为：

```
SELECT [DISTINCT] select_expr [, select_expr ...]
```

其中 select\_expr 可以是基本列（直接从表或子查询中选出的），也可以是列表表达式（包括算数计算、聚合函数、系统UDF等）。

当 select\_expr 为基本列时：

基本列是表中的列，这里的表可以是物理表也可以是虚表（物理表的别名引用或子查询产生的表）。

例如：

- SELECT A.a FROM A：指定表名和列名，表来自FROM子句，列属于FROM子句指定的表
- SELECT a FROM A：仅指定列名，列属于FROM子句指定的表

- SELECT A.a AS x FROM A：指定表名和列名（带别名），表来自FROM子句，列属于FROM子句
- SELECT a AS x FROM A：仅指定列名（带别名），表来自FROM子句，列属于FROM子句
- SELECT b, a FROM A JOIN B ...，仅指定列名，列属于FROM子句指定的第一个包含该列的表
- SELECT A.a FROM (SELECT A.a, b FROM A JOIN B ...)，指定表名和列名，列属于子查询返回的列（<table>.<column>完全匹配）
- SELECT a FROM (SELECT a, b FROM A JOIN B ...)，仅指定列名，列属于子查询返回的列（<column>完全匹配）
- SELECT X.a FROM (SELECT a FROM A) AS X：指定表名和列名，表来自子查询且是别名引用，列属于子查询返回的列（<column>完全匹配）
- SELECT X.x FROM (SELECT A.a AS x, b FROM A JOIN B ...) AS X：指定表名和列名，表来自子查询且是别名引用，列属于子查询返回的列且带别名
- SELECT count(distinct a) from A where b = 123, a不是分区列，且经过where条件筛选后数据量不超过100万条
- SELECT NULL ...：空值列
- SELECT 1, 2 ...：常量值列
- SELECT true ...：常量值（boolean类型的取值使用1或0）列
- SELECT \* ...：通配符代表所有列

暂不支持：

- SELECT A.\* ...：通配符代表特定表的所有列
- SELECT a, b, a ...：重复列

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- SELECT distinct a | count(distinct a) from A group by b：其中a不是分区列
- SELECT distinct a | count(distinct a) from A where b = 123, a不是分区列，且经过where条件筛选后数据量超过100万条

当 select\_expr 中使用列表表达式时：

例如：

- SELECT a + b ...：常用算术操作符（+，-，\*，/，%），其中表达式（a，b）是合法的值、基本列或列表表达式，但必须是数值类型
- SELECT min(a), UDF\_EXAMPLE(a, s) ...：系统内置或用户自定义函数，可以是值、基本列或列表表达式，但数据类型要符合函数参数要求
- SELECT CASE WHEN a > 0 THEN a + b ELSE 0 END ...：CASE-WHEN表达式，其中表达式（a > 0）是合法的 **行过滤表达式**（参看后面），THEN和ELSE取值是合法的 **列表表达式** 而且数据类型相同（int）
- SELECT a IS NULL, a IS NOT TRUE, a >= 0, s LIKE '%foo' ...：合法的 **关系表达式**（参看后面）
- SELECT ! a, NOT s ...：合法的 **逻辑表达式**（参看后面）

暂不支持：

- SELECT a IS [NOT] TRUE, a BETWEEN 1 AND 2, a IN (1, 2), a CONTAINS (1, 2, 3)：不支持部分关系表达

- SELECT a XOR b：不支持逻辑表达式（参看4.3节）
- SELECT a & b FROM A, B：不支持位操作

当 select\_expr 为列聚集函数表达式时：

包含任何聚集函数的列表表达式，即聚集表达式：

例如：

- SELECT COUNT(\*)：常用聚合函数，详见附件中函数表
- SELECT UDF\_SYS\_COUNT\_COLUMN(a) FROM A：系统内置聚合函数（UDF\_SYS\_...）
- SELECT SUM(CASE WHEN ... THEN ... ELSE ... END)：聚合表达式（SUM(...)）套列普通表达式（CASE WHEN ...）做为其子表达式
- SELECT SUM(a) / COUNT(a) FROM A：列普通表达式（/）套聚合表达式（做为其子表达式）

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- SELECT a + COUNT(\*) FROM A：普通表达式（+）不能同时套普通表达式（a）和聚合表达式（COUNT(...)）；
- SELECT SUM(COUNT(\*))：聚合表达式不能套聚合表达式做为其子表达式；

## FROM/JOIN子句（表扫描、连接与子查询）

用法：

```
FROM table_references
```

或

```
FROM table_referencesA as table_aliasA JOIN table_referencesB as table_aliasB on join_conditions
```

或

```
FROM (SELECT ...) as aliasA JOIN table_referencesB as table_aliasB on join_conditions
```

或

```
FROM (SELECT ...) as aliasA JOIN (SELECT ...)as aliasB on join_conditions
```

例如：

- FROM A：指定表
- FROM (SELECT ...)：子查询
- FROM A JOIN B ON A.a = B.b JOIN C ON A.a = C.c：单表或多表连接，默认是符合条件记录的笛卡尔集。需要注意的是，目前分析型数据库中使用JOIN时若一方是物理表，那么物理表参与JOIN的列必须是分区列并且已建立HASH索引

- FROM (SELECT ...) AS X JOIN A on X.x = A.a : 表、子查询和JOIN结合使用
- SELECT ... FROM A JOIN (SELECT ...) : 在V0.6.x及后续版本支持
- SELECT ... FROM (SELECT ...) JOIN (SELECT ...) : 两个子查询之间的Join, 在V0.6.x及后续版本支持

不支持或语义错误：

- SELECT ... FROM A FULL JOIN B : 不支持全连接；
- SELECT ... FROM A RIGHT JOIN B : 不支持右连接，需要转换为左连接；
- SELECT ... FROM A SEMI JOIN B : 不支持半连接；
- SELECT ... FROM A, B : 单表或多表连接，但没有ON条件
- SELECT ... FROM A, B WHERE A.a = B.b : 单表或多表连接在WHERE子句中有隐含ON条件，但是没有on子句的，暂不支持

0.8版本中两个表关联可运行的充要条件（四原则）：

- （1）两个表均为事实表且在同一个表组，或两个表中有一个是维度表。
- （2）两个表均为事实表且拥有相同的一级分区列，或两个表中有一个是维度表。
- （3）两个表均为事实表且关联条件(on)中至少含有一个条件是两个表各自的分区列的等值关联条件，或两个表中有一个是维度表。
- （4）关联条件（on）中的条件两端有有效的HashMap索引

0.9版本中两个表关联可加速运行的条件：

- （1）两个表均为事实表且在同一个表组，或两个表中有一个是维度表。
- （2）两个表均为事实表且拥有相同的一级分区列，或两个表中有一个是维度表。
- （3）两个表均为事实表且关联条件(on)中至少含有一个条件是两个表各自的分区列的等值关联条件，或两个表中有一个是维度表。

0.9版本若使用Full MPP Mode支持任意形态的表关联（详见4.4节），该功能目前邀请少数客户测试，后续全面开放。

## WHERE子句（过滤表达式）

在过滤表达式中，单个条件之间可以用AND和OR进行连接，支持括号决定条件表达式的优先级，亦支持表达式嵌套。

像普通的SQL一样，单个条件的组成是一个布尔判断。暨无论如何组合，单个条件最终是一个返回True/False的表达式。在最简单的情况下，单个条件由左侧的主体、中间的比较操作符和右侧的断言值组成。

例如：column\_name IS NOT NULL 中，column\_name 为布尔主体，IS NOT为比较操作符，NULL为断言值。

分析型数据库中比较操作符支持 = 、 >= 、 > 、 <= 、 < 、 <> 、 != 等基本操作符，也支持IS、IN、CONTAINS、BETWEEN...AND...、LIKE这样的关系操作符，并且关系操作符均支持其反义的版本（IS NOT、NOT IN、NOT LIKE等）。

布尔主体可以是一个列，也可以是表达式的嵌套。

例如：

- WHERE a > 0：关系表达式（>，>=，<，<=，=，<>，!=）
- WHERE a IS [NOT] NULL：关系表达式，判断是否为空值
- WHERE a BETWEEN 1 AND 10：对于表达式（a）是数字类型（long，int，short，double，boolean）和时间类型（date，time，timestamp）支持
- WHERE s LIKE '%sample'：对于表达式（s）是字符串（string）类型支持
- WHERE a IN (1,2,3)：对于表达式（a）是多值列类型支持
- WHERE a IN (select a from dim\_table where c=1)：对于对于表dim\_table是维度表，支持IN后带子查询（V0.6.2之后的版本支持）
- WHERE a CONTAINS (1,2,3)：对于表达式（a）是多值列类型支持
- WHERE a > 0 AND b > 0：逻辑表达式
- WHERE (a > 0 OR b > 0) AND c > 0：支持括号决定条件表达式的优先级
- WHERE (CASE WHEN (a + b) > 0 THEN 0 ELSE 1 END) != 0 AND UDF\_EXAMPLE(a, b) > 0：表达式嵌套。

不支持或语义错误：

- WHERE EXISTS (SELECT...)：不支持
- WHERE a = ANY (SELECT ...)：不支持
- WHERE a = ALL (SELECT ...)：不支持

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- WHERE a IN (select a from fact\_table where c=1): fact\_table不是维度表的情况下0.8版本不支持

## GROUP BY/HAVING子句（分组和分组后过滤）

GROUP BY子句的用法：

```
GROUP BY {col_expr}
```

col\_expr为列（表中的列或子查询、别名产生的列）。

例如：

- `... FROM A GROUP BY a：仅指定列
- `... FROM A GROUP BY A.a：同时指定表名和列
- GROUP BY s, a IS NULL, b + c, UDF\_EXAMPLE(d)：一个或多个列表表达式
- SELECT a, SUM(a) FROM A GROUP BY a, s：可以包含不属于列透视的列表表达式（s`）
- `... FROM (SELECT A.a AS x, b FROM A JOIN B ...) AS X JOIN C ... GROUP BY b, c, X.x：包含在子查询或多表连接的一个或多个列表表达式
- `SELECT CASE WHEN a > 1 THEN 'Y' ELSE 'N' END AS x ... GROUP BY x：别名引用产生的列表表达式
- SELECT a, b, c ... GROUP BY 3, 1：按位置引用列投射的列表表达式

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- ... FROM A GROUP BY s ORDER BY COUNT(a) LIMIT 100：分区表达式（s）不是分区列（a）同时又有TOP-N（ORDER BY COUNT(a) LIMIT 100）0.8版本下计算的结果可能是不精确的（详见4.3节）

HAVING子句的用法：

HAVING子句用于在GROUP BY子句执行后对分组后的结果进行过滤。支持使用列投射和聚合表达式后的结果进行过滤。

```
HAVING having_condition
```

例如：

- SELECT a, sum(length(s)) AS x FROM A JOIN B ON A.a = B.b GROUP BY a HAVING (CASE WHEN (x / a > 1) THEN a ELSE 0 END) != 0：分组表达式（a）是分区列，HAVING表达式是基于列表表达式（a，sum(length(s))，s）的行过滤表达式

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- SELECT s, count(a) AS x FROM A GROUP BY s HAVING x > 1：分组表达式（s）不是分区列（a）

## ORDER BY子句（排序）

例如：

- SELECT a FROM A ORDER BY a DESC LIMIT 100：按照分区列（a）排序并返回TOP-N（100）
- SELECT a FROM A ORDER BY (a + b) DESC LIMIT 100：按照非分区列（a）排序并返回TOP-N（100）
- SELECT a, COUNT(a) FROM A GROUP BY a ORDER BY x：按照分区列（a）分组聚合再排序

0.8版本中不支持但是0.9版本的Full MPP Mode支持（详见4.4节）：

- SELECT a + b, COUNT(1) AS x FROM A GROUP BY a + b ORDER BY x：0.8版本下按照非分区列表达式（a+b）分组聚合再取TOP-N是非精确的（详见4.3）

## UNION [ALL]/INTERSECT/MINUS子句

UNION/INTERSECT/MINUS 0.8版本下仅支持操作字段为分区列，但是0.9版本的Full MPP Mode无限制（详见4.4节）。

UNION ALL无任何限制。

## LIMIT子句（返回行数限制）

示例：

- SELECT ... LIMIT 100 : 返回行限制为100行

不支持 :

- SELECT ... LIMIT 100, 50 : 不支持带偏移量的返回行限制

需要注意的是, 分析型数据库会对通过SELECT语句进行查询的返回行数做全局的最大限制, 公共云上目前为10000行, 若不加LIMIT或LIMIT行数超过10000, 则只会返回10000行。

## 4.2 逻辑表达式和特殊语法

### 表Join的条件

为了更高效的进行表关联, 分析型数据库对表关联操作在分析型数据库中, 两个事实表进行Join的充要条件是 : ( 1 ) 这两个表在一个表组 ; ( 2 ) 这两个表的Join Key是Hash分区列 ; ( 3 ) 两张表的Hash分区数必须一致 , 否则Join结果不准确 ; ( 4 ) 两张表的Join Key至少有一列建立了HashMap索引, 推荐建立在数据量较小的一侧。

维度表参与的关联, 只需要符合上述的 ( 4 ) 一点即可。

### 逻辑表达式

在分析型数据库中对逻辑表达式的支持如下 :

- NOT a : 如果表达式 ( a ) 是0则返回1, 非0则返回0, NULL则返回NULL
- !a : 同上
- a && b : 如果表达式 ( a ) 和 ( b ) 都不是NULL且都非0则返回1, 如果其中有一个是0则返回0, 否则返回NULL
- a || b : 如果表达式 ( a ) 和 ( b ) 都是0是返回0否则如果都不是NULL则返回1, 如果其中有一个为NULL则另外一个为0则返回0, 非0则返回1, 如果都是NULL则返回NULL

例如 :

- NOT 0 : 1
- NOT 5 : 0
- NOT NULL : NULL
- NOT a : 根据表达式 ( a ) 的取值决定
- ! 0 : 1
- ! 5 : 0
- ! NULL : NULL
- ! a : 根据表达式 ( a ) 的取值决定
- 1 && 5 : 1
- 1 && 0 : 0
- 5 && 0 : 0
- NULL && 0 : 0
- NULL && NULL : NULL

- NULL && 5 : NULL
- a && b : 根据表达式 ( a ) 和表达式 ( b ) 的取值决定
- a AND b : 同上
- 0 || 0 : 0
- 1 || 5 : 1
- NULL || 1 : 1
- NULL || 5 : 1
- NULL || 0 : 0
- NULL || NULL : NULL
- a || b : 根据表达式 ( a ) 和表达式 ( b ) 的取值决定
- a OR b : 同上

## 多值列的用法和限制

在分析型数据库中，多值列这个数据类型有其特殊的用法和限制。对于多值列，不允许在不经由WHERE子句中用IN/CONTAINS进行枚举筛选的情况下直接在SELECT/GROUP BY中使用。

例如：

- SELECT m, COUNT(a) FROM A WHERE m IN('foo', 'bar') GROUP BY m, m为多值列，这种情况下可支持。
- SELECT m FROM A limit 10, m为多值列时不支持
- SELECT m, COUNT(a) FROM A GROUP BY m, m为多值列时不支持

另外，由于多值列的数据结构的特殊性，在对含有多值列的行按照多值列进行分组时，多值列的每一个单值会等价于产生一个新的行，计算count/sum等聚合函数的结果可能会被多次计算。

例如原始数据如下：

| a | m                    |
|---|----------------------|
| 0 | 'bar'                |
| 1 | 'foo' , 'bar'        |
| 2 | 'foo' , 'bar' , 'oh' |

在此之上执行 SELECT m, COUNT(a) FROM A WHERE m IN('foo', 'bar') GROUP BY m 的结果：

| m     | COUNT(a) |
|-------|----------|
| 'foo' | 2        |
| 'bar' | 3        |
| 'oh'  | 1        |

## 非分区列的使用限制

分析型数据库作为分布式实时OLAP系统，其数据根据分区列分布在不同的物理服务器中，所以在一些特殊的情

况下，分区列和非分区列的使用上有所不同。

- count(distinct cor\_expr)中cor\_expr必须是分区列或分区列衍生的列，否则不能和group by连用
- SELECT distinct cor\_expr...中若cor\_expr不是分区列或分区列衍生的列，则结果可能不精确
- TOP-N计算（order by expr limit 100），若是同时和非分区列的GROUP BY子句连用，结果可能不精确

0.9版本的Full MPP Mode支持上述SQL的正确运行，详见4.4节

## 子查询使用限制

分析型数据库中，0.8版本及先前版本，子查询使用有所限制，例如：

```
select count(*) from (
select a from tbl group by a
) tmp;

-- a不是一级分区列时结果不准确，当group by a 返回结果不超过10000条时，可以临时使用：

select udf_sys_rowcount(*) from (
select a from tbl group by a
) tmp;

select a, cnt from (
select a,count(b) as cnt from tbl group by a
) tmp;

-- a,b均不是一级分区列时结果不准确，可以临时使用：

select a, sum(cnt) from (
select a,count(b) as cnt from tbl group by a
) tmp group by a;
```

0.9版本的Full MPP Mode支持上述SQL的正确运行，详见4.4节

## 4.3 Insert/Delete命令

如果一个表是实时写入的表，则分析型数据库会支持对该表的单条数据进行Insert/Delete操作，不过由于分析型数据库不支持事务，对于Insert/Delete命令，有一些限制。

在分析型数据库中Insert语句的语法是：

```
INSERT [IGNORE]
INTO tbl_name (col1,col2...)
VALUES (value1, value2...), (value1, value2...)....
```

在分析型数据库中，能够实时插入的表一定要定义主键或组合主键，和MySQL有一个非常大的不同是，分析型数据库在进行数据插入时，若发现同主键的数据时，默认执行覆盖行为。若使用insert ignore语法，则在发现有同主键的数据时，丢弃新插入的数据，保留原有数据。但是无论如何，在主键冲突时分析型数据库无法直接返回错误给insert的执行方。

在分析型数据库中的Delete语句的语法是：

```
DELETE FROM tbl_name WHERE where_definition
```

where\_definition中暂不支持函数定义。

对分析型数据库是数据插入和删除操作，原则上在分析型数据库操作返回成功后一分钟内可查（但不保证一定是一分钟的数据可见延迟），不过在表刚刚创建时的十分钟内，数据可能无法查询。

Delete语句执行后，数据会不可查询到，但不会在物理上立刻删除，原则上24小时内会自动清除掉，如果删除了大量数据想立刻生效，可以执行：

```
optimize table <tablename>;
```

执行成功一段时间后，物理上数据会被删除。

关于最终一致性：对于主键相同的数据的多次变更，分析型数据库会遵循分析型数据库返回语句执行成功的顺序进行；对于主键不同的数据的两次变更，分析型数据库不保证先执行的变更会比后执行的变更更优先的查询到。但是当业务端暂停数据写入的若干时间后，分析型数据库会保证数据的最终一致。

若希望达成数据写入速率的最大化，建议：

（1）每条insert插入多条数据，具体条数视数据列数决定。若需要极限性能，每条insert插入的数据需拥有相同的一级分区号（后续分析型数据库提供SDK帮助客户达成此要求）；（2）每购买一个c1/c8 ECU，客户端可增加20-30个并发连接插入数据。

## 4.4 多计算引擎和Hint

本节内容适用于分析型数据库的0.9.5或以上版本。

文中提及的Full MPP Mode，在公共云中当前处于内测状态，需要使用的用户请提交工单申请开通。在专有云中，若您的集群或数据库没有开通该功能，请联系运维管理员或DBA开通。

### 多计算引擎

在0.9.15或更新的分析型数据库版本中，拥有两套计算引擎：

- COMPUTENODE Local/Merge(简称LM)：先前版本的旧计算引擎，优点是计算性能很好，并发能力强，缺点是对部分跨一级分区列的计算支持较差。
- Full MPP Mode（简称MPP）：0.9.5版本新增的计算引擎，优点是计算功能全面，对跨一级分区列的计算有良好的支持，可以通过全部TPC-H查询测试用例（22个），和60%以上的TPC-DS查询测试

用例。缺点是计算性能相对LM引擎较差，并且计算并发能力相对很差。

Full MPP Mode计算引擎目前面向定向邀请的客户内测，未来会全面开放。

在已经开启Full MPP Mode引擎功能的数据库中，分析型数据库可以自动对查询Query进行路由，将LM引擎不支持的查询路由给MPP引擎，尽可能兼顾性能和通用性，目前以下几种情况会自动路由到Full MPP Mode：

(1)特定函数，如row\_number over等，LM模式不识别，捕获异常；

(2)Join类：

- 事实表 join 事实表，join key全部在非分区列上；
- 不同表组的事实表 join 事实表；
- 维度表在前，left join 事实表；
- 事实表 right join 维度表（同上）；
- 事实表 join 事实表，一级分区数不同。

(3)Group By、Order By、Having类：

- Group By仅含非分区列，外层套子查询；
- Group By仅含非分区列，带Order By；
- Group By仅含非分区列，带Having。

(4)UNION/INTERSECT/MINUS不含分区列

(5)SELECT 复杂表达式，如SUM/SUM, 任何带聚合函数的计算表达式等

(6)COUNT DISTINCT或DISTINCT非分区列

但是，在一些情况下，自动路由并非最适于用户使用，用户可以自行判断自己的Query合适哪一种计算引擎，使用Hint强制将查询路由到某一个计算引擎中：

```
--强制使用COMPUTENODE LM计算引擎
/*+engine=COMPUTENODE*/ select * from ....

--强制使用Full MPP Mode计算引擎
/*+engine=MPP*/ select * from ....
```

## Join的小表广播

在0.8.35版本后，分析型数据库支持小表广播模式进行一种特殊的高性能关联：虽然不符合关联四原则中分区列等值或分区数一致的原则，但是关联的左表（或子查询）/右表（或子查询）中有一个表（或子查询）很小（一般不超过3万行，5列，512KB），则可以通过小表广播模式快速关联，例如：

```
/*+engine=COMPUTENODE*/ select A.a, B.b from (select/*+broadcast=true*/ a from B where c=100) A join B on
A.a = B.a;
```

其中A子查询数据量较小，是被广播的表，B表数据量无限制。这种Join性能也是较好的。

## 第五章 Data Pipeline

### 5.1 数据的导入

分析型数据库目前支持从阿里云外售的ODPS ( <http://www.aliyun.com/product/odps/> ) 中导入数据。OSS、RDS等数据源的导入可以通过CDP进行。

将数据导入分析型数据库有两种方法：直接使用SQL命令进行导入，或通过DMS界面进行导入。通过DMS界面导入数据已在快速开始这一章中进行了介绍。下面主要详细介绍如果通过SQL命令导入数据。

#### (1) 准备ODPS表或分区

首先我们需要创建好数据源头的 ODPS 表（可以是分区表或非分区表，目前要求分析型数据库中的字段名称和 ODPS表中的字段名称一致），并在表中准备好要导入的数据。首次导入一个新的ODPS表时，公共云用户需要在ODPS中授权该表给分析型数据库的导入用账号 ALIYUN\$garuda\_build@aliyun.com 与 ALIYUN\$garuda\_data@aliyun.com，需要表的Describe和Select权限。另外为了保护用户的数据安全，分析型数据库目前仅允许导入操作者为Proejct Owner的ODPS Project的数据，或者操作者为ODPS Table的创建者。

#### (2) 通过SQL命令导入数据

#### SQL语法

```
LOAD DATAFROM 'sourcepath' [IN VERSION dataVersion][OVERWRITE] INTO TABLE  
tablename [PARTITION (partition_name,...)]
```

如果使用ODPS数据源，则sourcepath为：

```
odps://<project>/<table>/[<partition-1=v1>/.../<partition-n=vn>]
```

- ODPS项目名称 project
- ODPS表名 table
- ODPS分区 partition-n=vn，可以是多级任意类型分区

数据版本dataVersion：

数据版本（dataVersion）是分析型数据库管理一次导入数据用的批次号，可用于未来可能开放的数据回滚等feature，通常情况下可以不需要填写。

- 数据版本必须是long型，且取值在表上单调递增
- 可选，如果不指定则取当前时间（秒），格式是 yyyyMMddHHmss，如 20140812080000

分析型数据库表（分区）：

分析型数据库表名（tablename）

- 格式 `<table_schema>.<table_name>`，其中 `table_name` 是分析型数据库表名，`table_schema` 是表所属DB名
- 不区分大小写
- 分区格式 `partition_column=partition_value`，其中 `partition_column` 是分区列名，`partition_value` 是分区值
- 分区值必须是 long 型
- 分区值不存在时表示动态分区，例如第一级hash分区
- 不区分大小写
- 目前最多支持二级分区

覆盖导入选项（OVERWRITE）：

- 导入时，如果指定数据日期的表在分析型数据库线上已存在，则返回异常，除非显示指定覆盖

返回值：

`'\<jobId\>'`

任务ID（jobId），用于后续查询导入状态。

- 任务ID，唯一标识该导入任务
- 字符串，最长256字节

## 5.2 数据导入状态查询

数据导入命令发送后，数据并不会立刻导入到分析型数据库中，而是会在后台进行数据的导入工作。用户可以通过使用SQL命令或在iDB Cloud中进行查询数据导入状态。高级用户也可以直接在`information_schema`中查询全部数据导入的信息（具体见附录）。

查询数据的导入状态有多种方法：

（1）通过SQL语句查询：

通过SQL命令查询适合提交导入没有完成或完成不久的情况。

SQL语法：

```
select state from information_schema.current_job where job_id = '\<jobid\>'
```

返回值：

`'\<jobState\>'`

返回值表示该任务状态（ jobState ）：

- NEW 初始状态
- INITED 排队中
- RUNNING 正在导入
- SUCCEEDED 导入成功
- FAILED 导入失败
- ERROR 导入出错（系统内部错误）

### 查询导入状态

- 任务不存在，则返回空
- 已完成的任务会被定时清理转为历史任务

（2）通过iDB Cloud 查询数据导入状态 在iDB Cloud中，点击菜单上的导入导出->导入状态，即可查询每天的导入任务情况，并且通过多个维度进行筛选浏览。

（3）通过Meta DB进行查询

## 5.3 海量数据导出

分析型数据库的查询模式适合在海量数据中进行分析计算后输出适量数据，若需要输出的数据量达到一定规模，分析型数据库提供数据导出（DUMP）的方式。

同数据导入，进行数据导出时分析型数据库通过一个云账号进行数据导出，因此对于导出到ODPS的情况，公有云用户需要对garuda\_data@aliyun.com账号授予目标ODPS项目的createInstance权限，以及对导出目标表的describe、select、alter、update、drop权限。

导出到OSS等更多目标支持，会在陆续上线。

### 通过iDB Cloud页面导出

进入iDB Cloud页面，选择菜单栏上的导出按钮，弹出导出对话框。选择或填写相应的值：

- **目标库**：导出数据文件的存放目标库，目前支持阿里云ODPS（<http://www.aliyun.com/product/odps>）
- **是否覆盖**：若指定的数据存放路径已经存在，是否进行覆盖，选择否时，导出返回错误，无法成功
- **数据存放路径**：导出数据文件在目标库OSS中路劲（例如：  
odps://project\_name/table\_name/partition\_spec）
- **SQL**：进行数据导出计算的查询语句

填写完毕后，点击“确定”按钮。

### 通过类DML语句进行导出

类似于普通的SQL查询语句，用户也可通过类似于DML语句进行数据导出。

```
DUMP DATA
[OVERWRITE] INTO 'odps://project_name/table_name'
SELECT C1, C2 FROM DB1.TABLE1 WHERE C1 = 'xxx' LIMIT 100000
```

## 关于返回数据行数

导出方式对海量数据的计算输出具有良好的性能（百万行数据导出在数百毫秒数据级），但是，对于数据精确度有一定牺牲，即实际返回的数据行数，可能是不完全精确。以限制导出行数为1000为例（LIMIT 1000）：

- 实际数据行数**可能稍大于1000**，例如此时有120个数据分片，则等同于每个分区明确指定"LIMIT 9"，最多可能返回1080
- 实际数据行数**可能稍小于1000**，如果符合条件的行数的总数小于1000
- 实际数据行数**可能稍小于1000**，如果数据分片很均匀，例如此时有120个数据分片，如果某些分片返回数据行小于9的话，则等同于每个分区明确指定"LIMIT 9"

# 第六章 用户与权限

## 6.1 用户账号类型与用户管理

分析型数据库用户基于阿里云帐号进行认证，用户建立的数据库属于该用户，用户也可以授权给其他用户访问其数据库下的表。

## 用户帐号与认证

### 阿里云帐号

- 帐号格式：ALIYUN\$user\_account@aliyun.com，其中ALIYUN\$为帐号前缀，标识该帐号类型为阿里云帐号，未来分析型数据库会推出更多的帐号类型支持。
- 通过阿里云官方网站注册

### 认证

- Access Key：通过阿里云官网的阿里云控制台生成和管理，使用分析型数据库的用户必须在[https://i.aliyun.com/access\\_key/](https://i.aliyun.com/access_key/) 拥有至少一个有效的access key。
- 访问分析型数据库服务时使用 Access Key。

## 用户类型

- 数据库所有者：必须开通了分析型数据库服务，同时是数据库的创建者
- 用户：被授权的数据库用户，由数据库所有者授权时添加，无须开通分析型数据库服务

## 添加用户

### SQL语法

```
ADD USER user ON db_name.*
```

### 返回值

空

### 添加用户

- 创建数据库时，数据库的创建者即做为拥有者添加
- 用户在第一次被授权一个数据库中的任何权限时，分析型数据库会自动添加为该数据库的用户
- 主动添加到数据库的用户初始时没有任何权限

## 查看用户列表

### SQL语法

```
LIST USERS ON db_name.*
```

### 返回值

```
\<user_id\> \<user_id\> ...
```

### 查看用户列表

- 只有数据库的拥有者可以查看该数据库的所有用户

## 6.2 分析型数据库的权限模型

分析型数据库支持基于数据库表的层级权限管理模型，提供类似MySQL的ACL授权模式。一个ACL授权由被授权的用户、授权对象和授予的对象权限组成。

### 分析型数据库中的用户

如6.1所述，任何分析型数据库支持的账号类型均可视为一个用户。和MySQL略有不同的是，分析型数据库目

前不支持针对用户在host上授权。

分析型数据库的权限对象和各对象权限

- Database，即 db\_name.\* 或 \*（默认数据库），指定数据库或数据库上所有表/表组
- Table，即 db\_name.table\_name 或 table\_name，指特定表
- TableGroup，即 db\_name.table\_group\_name 或 table\_group\_name，指特定表组
- Column，语法上由 column\_list 和 Table 组成，指定表的特定列
- 聚合：Database -> Table[Group] -> Column，即每个权限级别能聚合其下面级别的所有权限
- 在 Database 级别上，某个权限实际可能包含多个权限，例如 GRANT CREATE ON \*.\* 同时包含创建数据库和创建表的权限

| -         | Database | Table 或 TableGroup | Column | 说明                                                     |
|-----------|----------|--------------------|--------|--------------------------------------------------------|
| SELECT    | +        | +                  | +      | 查询数据                                                   |
| LOAD DATA | +        | +                  | -      | 导入表（分区）数据                                              |
| DUMP DATA | +        | +                  | -      | 导出表（分区）数据                                              |
| DESCRIBE  | +        | +                  | -      | 查看数据库、表/表组信息（Global、Database）、查看表/表组信息（Table[Group]）   |
| SHOW      | +        | +                  | -      | 列出数据库、表/表组内部对象（Global、Database）、列出表内部对象（Table[Group]）  |
| ALTER     | +        | +                  | -      | 修改表/表组定义                                               |
| DROP      | +        | +                  | -      | 删除数据库、表/表组或分区（Global、Database）、删除表/表组或分区（Table[Group]） |
| CREATE    | +        | -                  | -      | 创建数据库、表/表组或分区（Global）、创建表/表组（Database）                 |
| INSERT    | +        | +                  | -      | 执行Insert的权限                                            |

|                  |   |   |   |             |
|------------------|---|---|---|-------------|
| DELETE           | + | + | - | 执行Delete的权限 |
| ALL [PRIVILEGES] | + | + | + | 以上所有权限      |

## 查询数据的权限

- 查询表数据需要 SELECT 权限，最小级别是列
- 并非所有查询都需要该权限，例如 SELECT now()

## 导出数据的权限

- 导出数据同时需要 DUMP DATA 和 SELECT 权限
- 同时需要数据导出目的地的数据写入相关权限

## 6.3 使用ACL进行权限管理

前文提到，类似MySQL，分析型数据库中的数据库创建者可以使用GRANT/REVOKE语句进行授权和权限回收。

### 进行授权

#### SQL语法

```
GRANT privilege_type [(column_list)] [, privilege_type [(column_list)]] ... ON [object_type]
privilege_level TO user [, user] ...
```

#### 返回值

空

#### 被授权用户

```
user: 'user_name'[@'host_name']
```

- user\_name 和 host\_name 必须使用单引号或双引号，如 'ALIYUN\$test-user@aliyun.com'@'%'
- 目前，host\_name 只支持 %，即不支持指定Host
- 可以写成 user\_name，等价于 'user\_name'@'%'

#### 权限

privilege\_type 为具体的权限类型，每一级对象拥有的权限类型参见6.2节

[(column\_list)] 为可选，当对象级别为表时，这里可以填写列的列表，进行针对具体列的授权

[object\_type] 为可选，标明权限对象的类型，如Database、Table、TableGroup等，建议填写

privilege\_level 为被授权对象的表达式，填写方法见6.2节

### 例子

```
GRANT describe, select ON tablegroup db_name.table_group_name TO user 'ALIYUN$test_user@aliyun.com'@'%';
```

```
GRANT describe, select (col1, col2) ON table db_name.table__name TO user 'ALIYUN$test_user@aliyun.com';
```

## 权限回收

### SQL语法

```
REVOKE privilege_type [(column_list)] [, privilege_type [(column_list)]] ... ON [object_type]  
privilege_level FROM user [, user] ...
```

### 返回值

空

- 在语法上与授权语句基本一致

## 查看用户权限

### SQL语法

```
SHOW GRANTS [FOR user] ON [object_type] privilege_level
```

### 返回值

```
'GRANT ALL ON db_name.* TO user' 'GRANT SELECT(column_name) ON db_name.table_name  
TO user'
```

### 查看用户权限

- 不指定用户或是指定用户即当前用户，则列出当前用户在指定数据库上被授予的权限
- 否则，列出当前用户在指定数据库上授予指定用户的权限

## 其它相关内容

### 权限的默认约定

- 数据库创建者拥有在该数据库上的所有权限
- 表（组）创建者拥有在该表（组）上的所有权限
- 其他数据库用户拥有被各数据库所有者授予的权限
- SHOW DATABASES 语句不进行权限检查，可在不指定数据库的前提下之星，会返回用户所有拥有权限的数据库列表
- 创建数据库时不检查权限，但用户必须通过阿里云官网开通了分析型数据库服务并处于服务正常运行状态

### 授权/回收权限的权限

- 数据库所有者（暨创建者）在该数据库上执行 Database 及其级别以下的授权/回收权限，如 SELECT ON db\_name.\*
- 其他数据库用户目前无法执行授权/回收权限操作

### 设置ip白名单

目前分析型数据库支持设置DB粒度的ip访问白名单，该功能目前处于内测，有需要的客户可以提交工单索取设置方法和注意事项。

## 第七章 性能优化和诊断

### 7.1 使用执行计划

在进行一个业务SQL的性能调优，或是查看一个业务SQL的计算成本消耗时，使用explain命令来进行查看是一个很好的选择。iDB Cloud for 分析型数据库中提供图形化的执行计划命令，同时，用户也可以手动使用explain指令进行查看。

#### explain 语句

分析型数据库支持通过explain语句来查看逻辑计划和物理执行计划，当用户发起一个explain查询到分析型数据库系统后，分析型数据库会抽样一个数据分区来分析执行计划，并以图形方式展现给用户。

explain语句的格式为， explain + select语句, 例如:

简单sql的explain

```
explain select count(*) from test4dmp.test where id > 0
```

## 复杂sql的explain

```
explain
select student.id, count(*)
from test4dmp.student
inner join test4dmp.grade on student.id =grade.sid
inner join test4dmp.elective on elective.id=student.id
group by student.id
having count(*) > 2
order by student.id
limit 10
```

## 返回文本格式

当用户通过查询的方式，想要获取文本格式的explain语句后，将会得到如下的json串：

- 返回的json格式

```
| EXPLAIN      |
| logical json |
| physical json|
```

返回格式说明 一共会返回2行1列并且列名为EXPLAIN的ResultSet记录。其中第一行为逻辑计划，第二行为物理计划。

### JSON格式说明

- Node 代表着唯一的子节点
- LeftNode 代表左子树
- RightNode 代表右子树
- MiddleNode 代表所有的中间子树（多叉执行树），可以有多个MiddleNode
- 其余key-value对统一在同层级的Node和\*Node节点显示

## Explain 逻辑计划详细语义

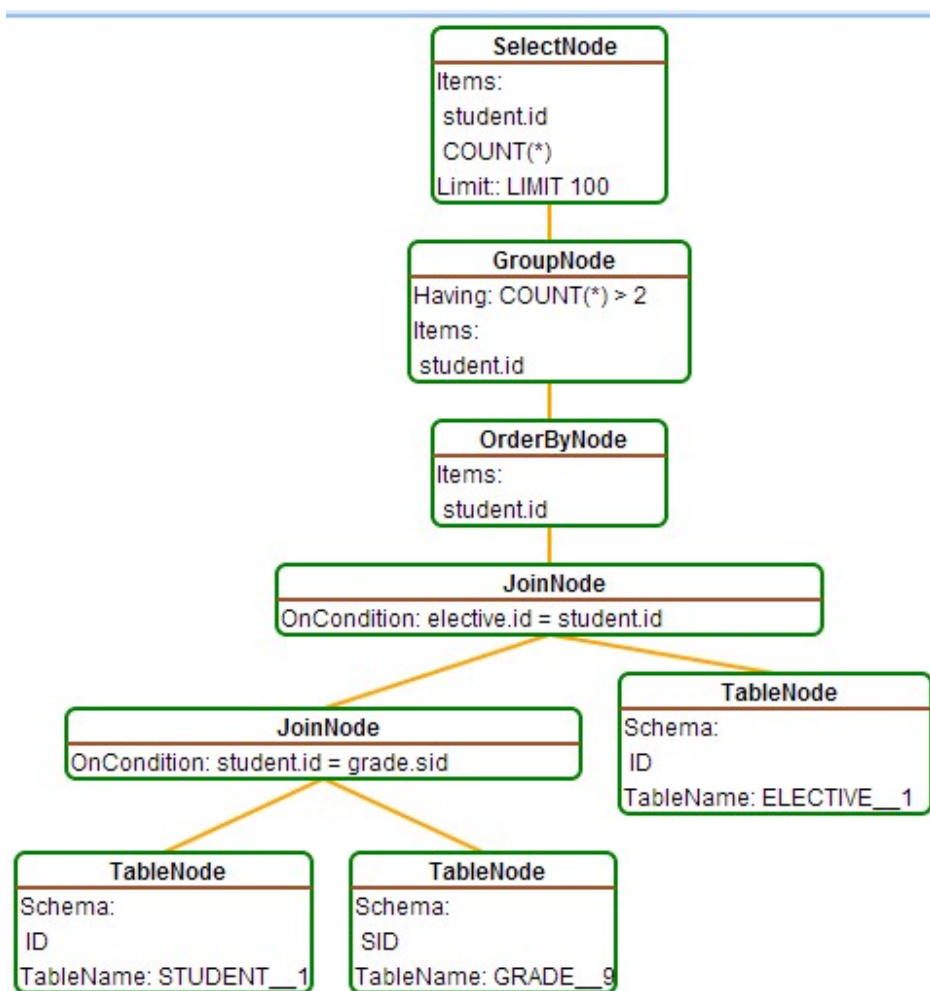
### 样例sql

```
explain
select student.id, count(*)
from test4dmp.student
inner join test4dmp.grade on student.id =grade.sid
inner join test4dmp.elective on elective.id=student.id
group by student.id
having count(*) > 2
order by student.id
limit 10
```

## 逻辑计划的explain string

```
{
  "Items": ["student.id", "COUNT(*)"],
  "Name": "SelectNode",
  "Node": {
    "Having": "COUNT(*) > 2",
    "Items": ["student.id"],
    "Name": "GroupNode",
    "Node": {
      "Items": ["student.id"],
      "Name": "OrderByNode",
      "Node": {
        "Name": "JoinNode",
        "LeftNode": {
          "Name": "JoinNode",
          "LeftNode": {
            "Name": "TableNode",
            "Schema": ["ID"],
            "TableName": "STUDENT__1"
          },
          "OnCondition": "student.id = grade.sid",
          "RightNode": {
            "Name": "TableNode",
            "Schema": ["SID"],
            "TableName": "GRADE__9"
          }
        },
        "OnCondition": "elective.id = student.id",
        "RightNode": {
          "Name": "TableNode",
          "Schema": ["ID"],
          "TableName": "ELECTIVE__1"
        }
      }
    }
  },
  "Limit": "LIMIT 100"
}
```

逻辑计划explain string的图形化展示效果：



- 逻辑计划各个节点说明：

- SelectNode表示这select中最终输出表达式的相关信息，例如select要输出的表达式集合
- GroupNode表示GroupBy语句的相关信息，例如groupby的列，having的表达式等
- OrderByNode表示OrderBy的列信息，例如列名，顺序等。
- JoinNode表示逻辑Join树的信息，例如join的on条件
- TableNode表示分区表的信息，例如参与计算的列，表名等。

## Explain 物理执行计划详细语义

样例sql

```

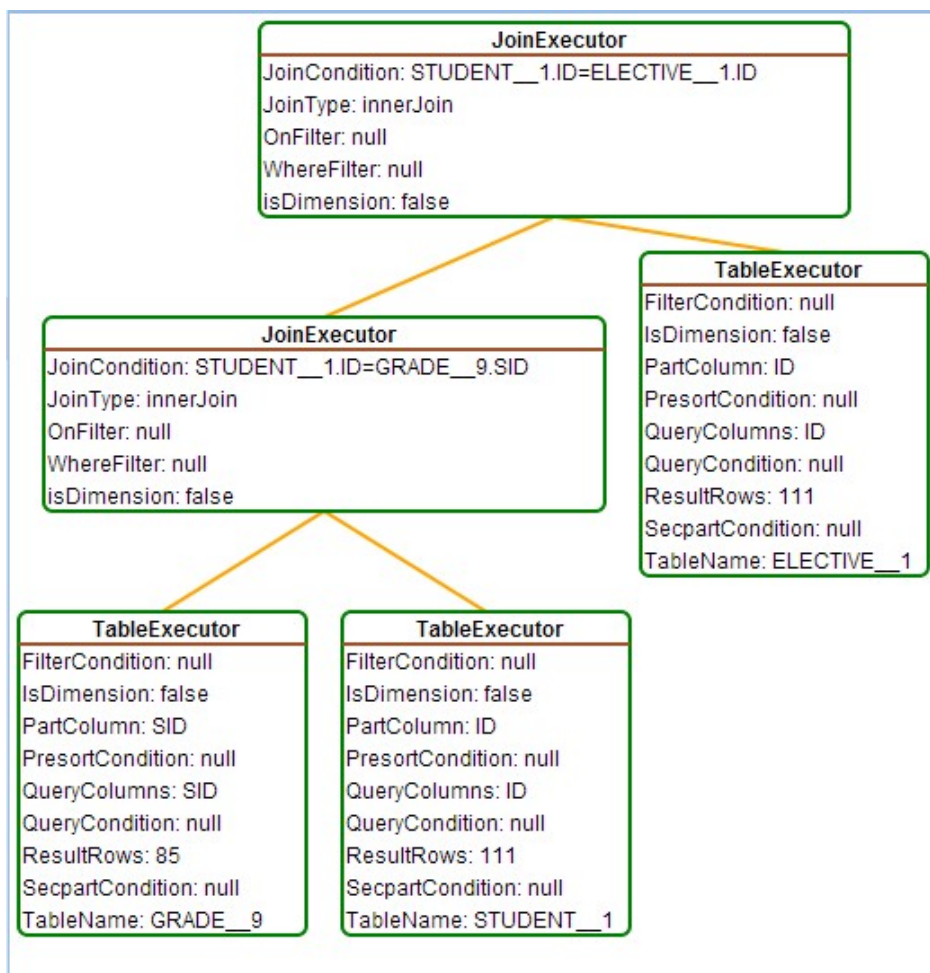
explain
select student.id, count(*)
from test4dmp.student
inner join test4dmp.grade on student.id =grade.sid
inner join test4dmp.elective on elective.id=student.id
group by student.id
having count(*) > 2
order by student.id
  
```

```
limit 10
```

物理计划的explain string

```
{
  "Name": "JoinExecutor",
  "isDimension": "false",
  "LeftNode": {
    "Name": "TableExecutor",
    "PresortCondition": "null",
    "SecpartCondition": "null",
    "QueryColumns": "ID, BOOLEAN_TEST, LONG_TEST",
    "FilterCondition": "null",
    "ResultRows": 0,
    "PartColumn": "ID",
    "IsDimension": "false",
    "TableName": "TEST_2",
    "QueryCondition": "(ID = 0)"
  },
  "JoinCondition": "TEST_2.ID=TEST_2.ID",
  "RightNode": {
    "Name": "TableExecutor",
    "PresortCondition": "null",
    "SecpartCondition": "null",
    "QueryColumns": "ID, BOOLEAN_TEST",
    "FilterCondition": "null",
    "ResultRows": 0,
    "PartColumn": "ID",
    "IsDimension": "false",
    "TableName": "TEST_2",
    "QueryCondition": "((INT_TEST < 0) AND (ID = 0))"
  },
  "JoinType": "innerJoin",
  "WhereFilter": "(BOOLEAN_TEST = BOOLEAN_TEST)",
  "OnFilter": "null"
}
```

物理计划explain string的图形化展示效果：



- 物理计划各个节点说明：

- JoinExecutor表示Join的节点
  - JoinCondition：join的条件
  - JoinType：join的类型，innerJoin, leftJoin, rightJoin, simiJoin等
  - OnFilter: on上面的过滤条件
  - WhereFilter：where里面的过滤条件（只有leftjoin中on和where的条件才有差异）
  - IsDimension：该Join子树中是否含有维度表
- TableExecutor表示参与计算的表信息
  - FilterCondition: 该表参与过滤计算的表达上（不能下沉索引的表达式）
  - IsDimension：该表是否是维度表
  - PartColumn：一级分区列
  - SecPartColumn：二级分区列
  - QueryColumns: 参与计算的列数（下沉索引计算的列不包含在此之内）
  - QueryCondition：索引条件
  - ResultRows：该节点预估cost
  - PresortCondition：预排序条件，可以用做优先索引条件下沉
  - SecpartCondition：二级分区筛选条件，可以用做二级分区筛选。
  - TableName: 分区表名

## 7.2 查看DB性能指标和报告

在日常的管理中，了解数据库的各项性能指标是非常重要的一个环节。分析型数据库提供实时的性能指标以及定时产生的性能报告，可以通过查询Meta DB的方式获取，也可以在iDB Cloud界面中查看图形化的报告。

在Meta DB中，DB的性能指标目前主要是存储在performance\_schema中。实时性能指标存储在performance\_schema.minute\_db\_profile 表中。

由于分析型数据库是一个分布式系统，所以minute\_db\_profile中存储了每一个前端服务节点记录的信息。要查询准确的DB层级汇总信息，需要先在performance\_schema.servers中查询一个数据库当前在线的前端服务节点ID。可使用如下SQL查询：

```
select distinct online_server_id from performance_schema.servers where table_schema = 'db_name';
```

其中db\_name为数据库名。之后再用上述SQL返回的online\_server\_id列表查询minute\_db\_profile表获取数据库30s粒度的实时信息：

```
select update_time, qps, pv, avg_rt, data_size from performance_schema.minute_db_profile where server_id in ('4d8b2019a04df41acaf83b5101d64e5e', 'a1408c344ebc134587f684d586703ede') and table_schema = 'db_name';
```

其中db\_name为数据库名，server\_id的内容为servers表中返回的所有online\_server\_id。这条SQL会返回多条记录，原则上每个server\_id一条，每条记录的update\_time是一个四个数字组成的字符串，标识着这条记录的更新时间，格式为'MMSS'，原则上30秒产生一条记录。返回的具体指标的含义详见附录六。

而查询定时产生的（目前是每小时一次，后续可能更改）性能报告则要简单的多。直接查询performance\_schema.hour\_db\_profile表即可。查询这个表时亦必须传入table\_schema参数暨数据库名。该表的thedata用'YYYYMMDD'的格式标明报告产生的日期，hour字段用'hh'的格式标明报告产生的时间。目前最多保留最近七天的报告。报告记录的具体指标亦详见附录六。

另外，performance\_schema.hour\_slow\_query表中提供了和性能报告同样周期内的最常见的最慢查询，查询方式和hour\_db\_profile一致。

若不想通过SQL查询数据库的性能信息，分析型数据库在iDB Cloud中提供了图形化的性能报告。30s内更新的实时性能指标可在iDB Cloud的数据库首页中直接看到。而定时的性能报告可在iDB Cloud的数据库首页中点击性能诊断报告按钮，进入一个列表页，然后选择一个报告产生的时间即可查看图形化的性能报告和最慢查询列表。

## 第八章 在生产中使用分析型数据库

## 8.1 业务系统连接分析型数据库并进行查询

使用jdbc odbc php python R-Mysql等连接分析型数据库的例子和注意事项介绍，以及流控、retry链接、异常处理等方法。

分析型数据库集群系统部署在阿里云环境中，用户在部署业务应用系统时，尽可能保证业务系统与阿里云环境的网络联通性，如购买阿里云ECS主机（<https://www.aliyun.com/product/ecs/>）作为业务系统的服务器。

### JDBC直接连接分析型数据库

最简单直接的连接并访问分析型数据库的方式是通过JDBC，分析型数据库支持MySQL自带的客户端以及大部分版本的mysql-jdbc驱动。

#### 支持的mysql jdbc驱动版本号

- **5.0系列**：5.0.2，5.0.3，5.0.4，5.0.5，5.0.7，5.0.8
- **5.1系列**：  
5.1.1，5.1.2，5.1.3，5.1.4，5.1.5，5.1.6，5.1.7，5.1.8，5.1.11，5.1.12，5.1.13，5.1.14，5.1.15，5.1.16，5.1.17，5.1.18，5.1.19，5.1.20，5.1.21，5.1.22，5.1.23，5.1.24，5.1.25，5.1.26，5.1.27，5.1.28，5.1.29，5.1.31
- **5.4系列**
- **5.5系列**

Java程序中，将合适的mysql-jdbc驱动包（mysql-connector-java-x.x.x.jar）加入CLASSPATH中，通过以下示例程序就能连接并访问分析型数据库。通过该JDBC方式直连分析型数据库时，和直连MySQL类似，需注意在使用完连接并不准备进行复用的情况下，需要释放连接资源。用户根据具体情况，设置\${url}，\${my\_access\_key\_id}，\${my\_access\_key\_secret}和\${query}值。

```
Connection connection = null;
Statement statement = null;
ResultSet rs = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    String url = "jdbc:mysql://mydbname-xxxx.ads-hz.aliyuncs.com:5544/my_ads_db?useUnicode=true&characterEncoding=UTF-8";

    Properties connectionProps = new Properties();
    connectionProps.put("user", "my_access_key_id");
    connectionProps.put("password", "my_access_key_secret");

    connection = DriverManager.getConnection(url, connectionProps);
    statement = connection.createStatement();

    String query = "select count(*) from information_schema.tables";
    rs = statement.executeQuery(query);

    while (rs.next()) {
        System.out.println(rs.getObject(1));
    }
}
```

```
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

## 通过JDBC连接池连接分析型数据库

通过JDBC连接池来连接分析型数据库是不错的选择，在1.5章节中，已经演示了通过连接池Druid连接分析型数据库的方式，请参考。

## 通过PHP连接分析型数据库

在1.5章节中，已经演示了在PHP应用中连接分析型数据库的方式，请参考。

## 关于负载均衡

用户在创建数据库（章节3.1）时，分析型数据库会根据用户指定的服务参数，为用户在分析型数据库集群中分配特定数量的前端服务机，这些机器对用户透明，用户得到的是访问该数据库的URL（例如：mydbname-xxxx.ads-hz.aliyuncs.com:5544，在控制台中的连接信息中获取），分析型数据库集群自己使用阿里云负载均衡产品SLB（<http://www.aliyun.com/product/slb>）来对访问请求进行负载均衡，用户无需关心访问分析型数据库时的负载均衡策略。若用户的业务应用自己想采用一定的负载均衡方案，可以参考<http://www.aliyun.com/product/slb>。

## 关于重试以及异常处理

网络环境下连接和使用分析型数据库，从业务应用的角度来看，使用适当的重试和异常处理机制是保证业务应用高可用性的手段之一。无论是通过JDBC直连、JDBC连接池、PHP，还是Python等方式连接分析型数据库，出现连接异常时，应用均可采用适当的重试机制来保证连接可用性。在海量数据并发处理任务量大的情况下，偶尔在查询过程中出现异常时，也可采用适当的重试机制来重发查询。

## 8.2.数据导入任务生产指南

在使用分析型数据库时，稳定的数据导入是非常重要的生产要素。一般新用户经常在进行首次的数据导入时因为操作不当无法成功，或成功后无法稳定运行。这里我们来看一下建立一个生产化的数据导入任务的注意事项。

### 数据的准备方面

想要稳定的导入数据，首先要在数据的源头稳定的产出数据。一份对于分析型数据库来说稳定的数据至少要满足：

数据所在的项目名（对应源头为ODPS）/文件访问路径（对应源头为OSS，暂不支持）/服务器连接串（对应源头为RDS，暂不支持）和表名与LOAD DATA命令中的源头一致并保持稳定。

数据表的字段名，在源头上与在分析型数据库上的配置一致，源头表可以比分析型数据库有更多的字段，但是不能比分析型数据库表缺少字段。

源头表进行导入的分区的数据不能为空，进行导入的数据主键不能有NULL值，HASH分区键不能存在大量NULL值或同样的HASH分区键的数据条数过多，例如超过了每个分区的平均数据量的三倍。否则不仅会对查询性能造成影响，在极端情况下也会导致数据导入时间过长或者失败。

### 调用导入命令

在数据产出后，可以通过MySQL连接的方式或者HTTP Rest-API的方式调用数据导入命令，这时应该注意：

调用命令时，命令所引用的源头表/分区的数据已经完整的产出完毕，并且若源头是ODPS/OSS，应该不在有任何在源头的写入操作。所以通常需要一个良好的离线任务调度系统（例如阿里云DPC中的数据开发平台）来进行相关的任务运行和调度。

调用命令时，要确保命令所引用的源头表/分区已经对ALIYUN\$garuda\_build@aliyun.com授予足够的权限并未开启保护模式等阻止数据流出的安全策略。

调用命令时分析型数据库中该表没有正在运行的导入任务，否则会返回失败。

### 查询数据导入状态和解决导入中的问题

在生产系统中查询数据导入状态，通常更多的是通过HTTP API进行的查询的，这里如果有一个较好的离线任务调度系统，那么实现难度并不大。

在数据导入的过程中，经常会因为出现各种错误而导致导入中断，具体的错误处理可以见附录一：错误码中。

## 8.3 BI/Adhoc业务的数据模型设计

在一个BI/Adhoc类OLAP业务中，数据的模型设计是非常重要的一点。下面，我们就基于一个企业商品库存分析的业务，来看一看在分析型数据库中如何进行数据模型的设计。

### 事实表

#### 事实表与大表分区

事实表一般都比较大会（行数>1亿或数据量>20G），需要分表，即按行水平切分为多个分区，即实现为分区表

- 目前，分区数一般是100
- 选定一个字段（后续将支持多个）做为一个分区列
- 每行数据的分区列的hash值对分区数（100）取模，决定该行被切分到哪个分区
- 分区列推荐是bigint型，也支持varchar类型（varchar用于join性能较差）
- 分区列首先决定于如何查询数据，因为与SQL查询能力紧密相关
- 同时，分区列的取值应该是离散的

#### 分区表与SQL查询

以下是关于分区表的SQL查询能力。

分区表与任意维度条件过滤（WHERE）

- 可以支持任意字段条件过滤
- 显然，包含OR操作的WHERE过滤表达式的性能会随条件的个数增加而较快降低

分区表与JOIN

- 分区表之间的JOIN只能基于分区列（事实表与维度表之间，或维度表相互之间JOIN不受此限制）
- 分区表之间JOIN需要分区数完全相同
- 分区列如果是varchar类型，则join性能相对较差
- 综合上述概念，分析型数据库引入表组（即多个事实表按照相同切分规则的组合）

分区表与Group-By、Order-By查询

- Group-By条件包含分区列，则应该放到第一列，此时结果是精确的且查询性能很好
- Group-By表达式如果不包含分区列（或第一列不是分区列），则分组数量在范围内（<5000），结果是精确的，但性能随分组数量增加而降低
- Group-By表达式如果不包含分区列（或第一列不是分区列），且分组数量不在范围内（>=5000），此时结果是不精确的且性能随分组数量增加而降低

- 如果Group-By表达式第一列不是按分区列，则Having语句不支持
- 全局分组TOP(N)：Group-By表达式包含分区列，则应该放到第一列，同时包含Order-By表达式，如果分组数量在范围内（<1000），结果是精确的，但性能随分组数量增加而较大降低
- 全局模糊分组TOP(N)：Group-By表达式第一列如果不是分区列且包含Order-By表达式，而且分组数量不在范围内（>=1000），此时结果是不精确的且性能随分组数量增加而较大降低
- 全数据排序：Order-By表达式第一列如果不是分区列，性能会随排序列或表达式取值增加而较大降低

### 分区表与子查询

- 子查询实现为UNION(ALL)各个分区内的查询结果，因此对于嵌套子查询或子查询加JOIN的最终结果与RDBMS是不一致的
- 子查询之间JOIN性能较差

### 事实表与小表

事实表如果较小（行数<1亿或数据量<10G），可以不建成分区表：

- 分区数设置为1，即不进行任何切分
- 单表的查询能力（如子查询、Group-By、Order-By等）不受限制，比分区表查询能力更强
- 与维度表可以按任意字段JOIN（但不支持string类型列做为JOIN列）
- 数据量增长时可以转换为分区表

## 维度表

### 维度表与小表

大部分情况下，维度表是小表（<1000w且<1G），Garuda针对维度表的支持：

- 标识为维度表，且分区数设置为1，即不进行任何切分
- 维度表单表的查询能力（如子查询、Group-By、Order-By等）不受限制
- 多个维度表之间可以按任意字段JOIN（但不支持string类型列做为JOIN列）
- 维度表与事实表（分区表）也可以按任意字段JOIN

### 维度表与分区表

- 维度表如果是大表，则必须建成分区表，将会有一些限制（符合上述分区表相关规则）

## 宽表

- 维度表存在多个层次时，通常建议打平为单张宽表，通过减少JOIN次数会提高查询性能
- 维度退化：维度表列数很少时可以直接打平到事实表，这样能避免事实表与维度表之间的JOIN，提高性能
- 如果维度表字段很多，而且字段都会参与多维查询的条件过滤或分组排序，则不建议打平，因为可能会造成事实表极度膨胀，反而影响性能
- 如果维度表因为自身很大必须建成分区表，则不仅不能打平到事实表，更会有一些限制（符合上述分区表相关规则）

## 聚合表

- 包含事实数据表的汇总信息的表
- 按照某个维度通过聚合操作（max、min、sum、count、avg）把事实列聚合，比如按照时间（月度等）
- 聚合表的数据量会大量减少，可以不建成分区表

## 实例分析

### 事实表

企业商品库存表（ent\_items\_stock\_detail）

- 数据量20亿
- 包含企业的商品库存值
- 事实列仅一列即库存数（code\_stock\_count）

### 维度表（直接）

以下维度字段 **直接关联** -> 企业信息表（ent\_info）

- ent\_type\_id string COMMENT '企业类型ID'，
- ent\_type\_name string COMMENT '企业类型名称'，
- ent\_info\_id string COMMENT '企业ID'，
- ent\_info\_name string COMMENT '企业名称'
- region\_id string COMMENT '地域ID'，
- region\_name string COMMENT '地域名称'

以下维度字段 **直接关联** -> 商品信息表（ent\_net\_detail）

- item\_info\_id string COMMENT '商品信息ID'，
- sale\_batch\_id string COMMENT '商品批次ID'，
- produce\_batch\_no string COMMENT '生产批次号'，
- produce\_date string COMMENT '生产日期'，
- exprie\_date string COMMENT '有效期至'
- item\_base\_info\_id string COMMENT '商品通用信息ID'，
- item\_base\_info\_name string COMMENT '商品基本信息名称'，
- produce\_ent\_info\_id string COMMENT '生产企业ID'，
- produce\_ent\_info\_name string COMMENT '生产企业名称'

以下维度字段 **直接关联** -> 地域信息表（region\_info）

- region\_id string COMMENT '地域ID'，
- region\_name string COMMENT '地域名称'，
- prov\_id string COMMENT '省ID'，

- prov\_name string COMMENT '省名称' ,
- city\_id string COMMENT '市ID' ,
- city\_name string COMMENT '市名称' ,
- area\_id string COMMENT '县ID' ,
- area\_name string COMMENT '县名称' ,

### 维度表（层级）

以下维度字段通过 商品信息表（ent\_net\_detail）**间接关联** -> 企业信息表（ent\_info）

- produce\_ent\_info\_id string COMMENT '生产企业ID' ,
- produce\_ent\_info\_name string COMMENT '生产企业名称' ,
- region\_detail string COMMENT '生产企业属性：region\_detail' ,
- produce\_region\_id string COMMENT '生产企业所在地域ID' ,
- produce\_region\_name string COMMENT '生产企业所在地域名称'

以下维度字段通过 商品信息表（无）**间接关联** -> 商品基本信息表（ent\_base）

- item\_base\_info\_id string COMMENT '商品通用信息ID' ,
- item\_base\_info\_name string COMMENT '商品基本信息名称' ,
- item\_spec string COMMENT '商品通用信息属性：item\_spec' ,
- approval\_no\_id string COMMENT '批准文号ID'

### 推荐方案

企业商品库存表（ent\_items\_stock\_detail）

- 事实表，建成分区表
- 分区数100
- 分区列选企业ID（ent\_info\_id），**需要先单独生成一个对应的long型唯一ID**
- 查询或子查询按照企业ID进行Group-By，Order-By时（做为第一列），查询能力没有限制且性能很好
- 查询或子查询按照商品ID进行Group-By，Order-By时，先在WHERE过滤条件里指定商品ID或取值范围，否则不精确且性能较差
- 查询或子查询按照商品其他层级维度（如省市县）时，维度取值较少，或先在WHERE过滤条件里该维度条件，否则不精确且性能较差

商品信息表（新建表 items\_info）

- 维度表，数据量小，不需要建分区表
- 存在多个层级维度（商品基本信息），打平为单张宽维度表
- 因为打平后存在列数很多，不建议进一步打平到企业商品库存表，因为会导致数据量膨胀很大
- 商品信息表本身SQL查询能力不受限制
- 商品信息表与企业商品库存表按 商品信息ID（item\_info\_id）JOIN
- 商品信息表支持任意字段条件过滤

企业信息表（ent\_info）

- 维度表，数据量不大（300w），不需要建分区表
- 如果以后扩展太大（列数），可转换为分区表，分区列是企业ID（ent\_info\_id），可以与企业商品库存表JOIN
- 目前列数较多，不建议打平到企业商品库存表
- 企业信息表本身SQL查询能力不受限制（若转换为分区表后有相应限制）
- 企业信息表与企业商品库存表按 企业ID（ent\_info\_id）JOIN
- 企业信息表支持任意字段条件过滤

#### 地域信息表（region\_info）

- 维度表，数据量不大（300w），不需要建分区表
- 仅单个层级维度
- 目前列数较多，不建议打平到企业商品库存表
- 地域信息表与企业商品库存表按 地域ID（region\_id）JOIN
- 地域信息表与商品信息表按 生产企业ID（produce\_ent\_info\_id）JOIN
- 地域信息表支持任意字段条件过滤

#### 查询示例

```
-- 某种商品库存TOP(100)的企业
select ent_info_id, count(*) as stock
from ent.ent_items_stock_detail
join ent.items_info on ent_items_stock_detail.item_info_id = items_info.item_info_id
where items_info.item_base_info_name='xxx商品'
group by ent_info_id
order by stock desc limit 100
```

## 8.4 在BI工具中连接和使用分析型数据库

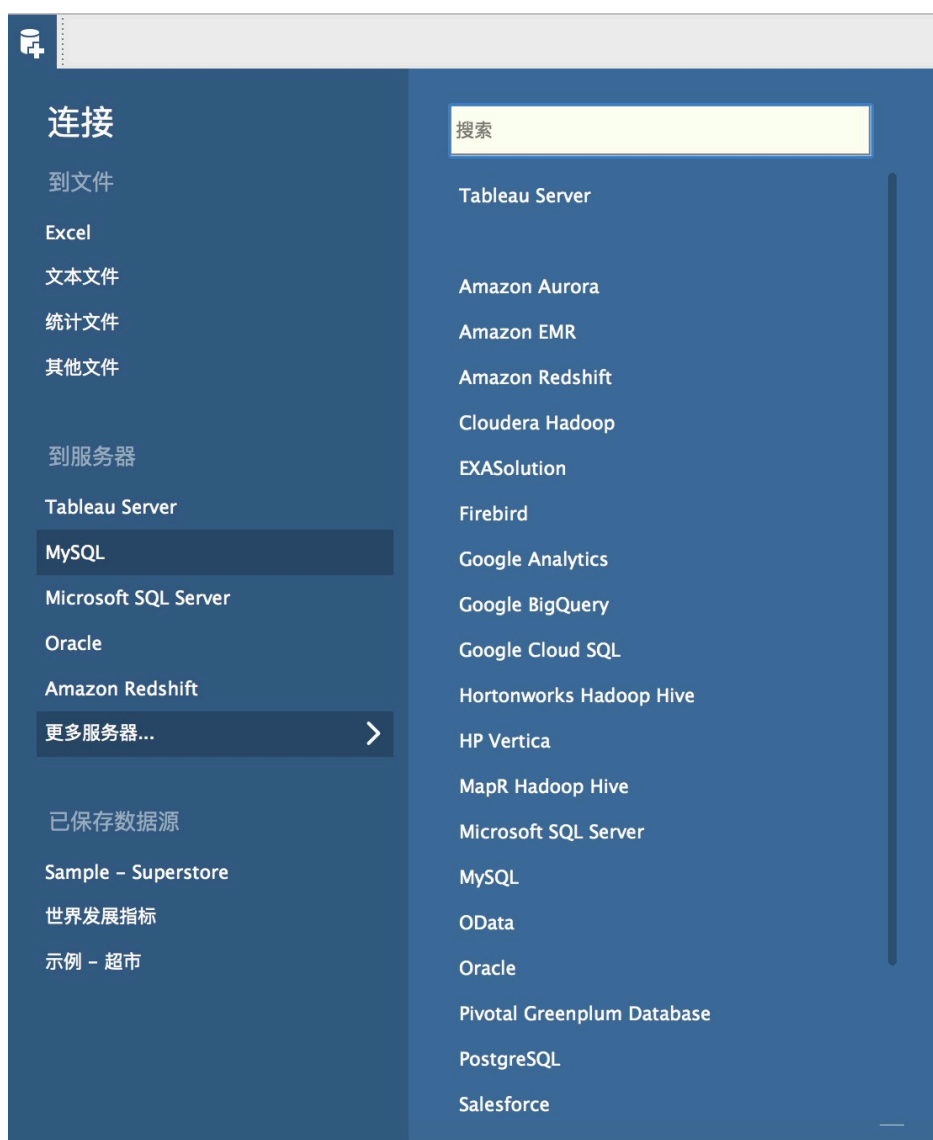
目前，阿里云分析型数据库能够支持在部分BI工具中以MySQL连接的方式进行连接和使用。

在使用这些BI工具连接分析型数据库时，通常需要先安装好ODBC MySQL Driver，这里推荐使用MySQL Connector/ODBC 3.51 版本（下载地址 <http://dev.mysql.com/downloads/connector/odbc/3.51.html>）。

### 在Tableau Desktop中使用分析型数据库

现在可以在Tableau Desktop 9.0及以上版本，通过MySQL连接的方式连接到阿里云分析型数据库读取和分析数据，产生可视化报表，大部分Tableau Desktop 9.x的功能可以在阿里云分析型数据库的连接中使用。

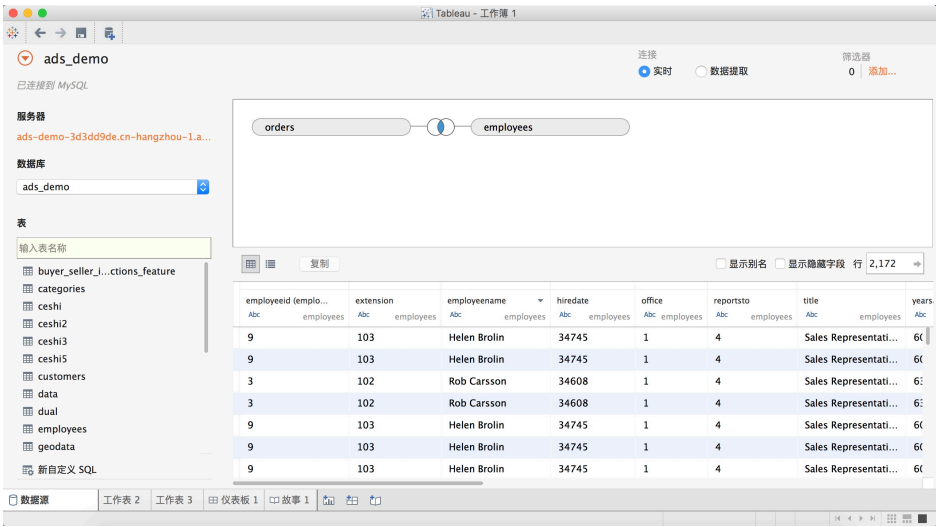
打开Tableau Desktop，点击“添加新的数据源”按钮 并选择数据源类型为MySQL。



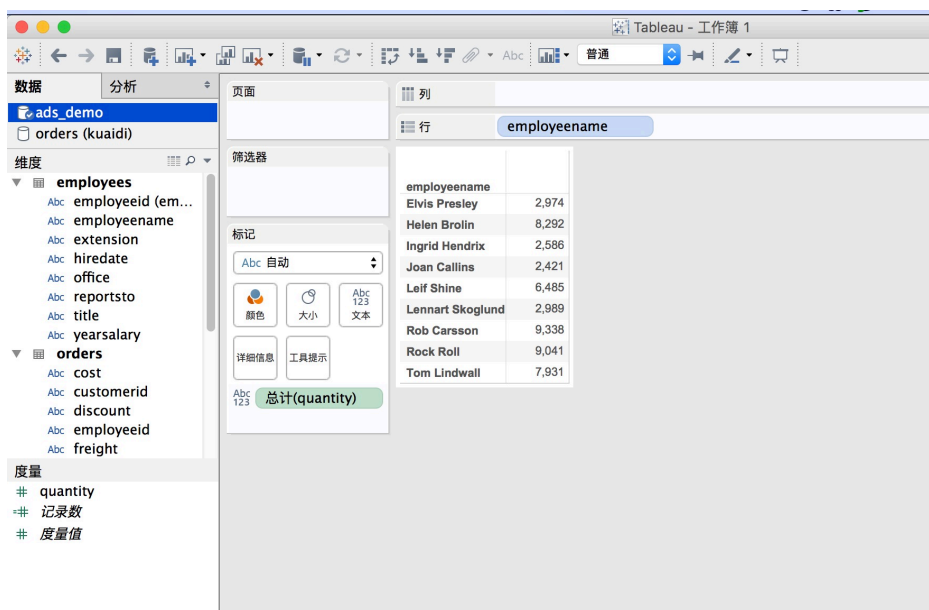
之后在弹出的对话框中，输入您要访问的数据库的域名和端口号（在DMS的数据库选择页面获取），以及将您的账号在阿里云的Access Key ID / Access Key Secret输入到用户名/密码框中，点击确定。



之后在Tableau的数据源页面上，可以看到您有权限的数据库，注意，您这里只能选择您连接到的域名端口所对应的数据库。选择后可以进行数据表选取，以及数据预览。

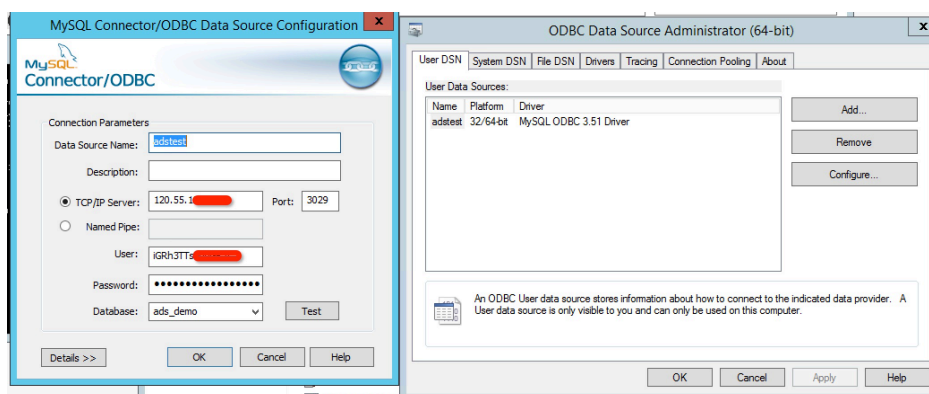


之后便可以进行工作表的建立以及更多其他功能。



## 在QlikView中使用分析型数据库

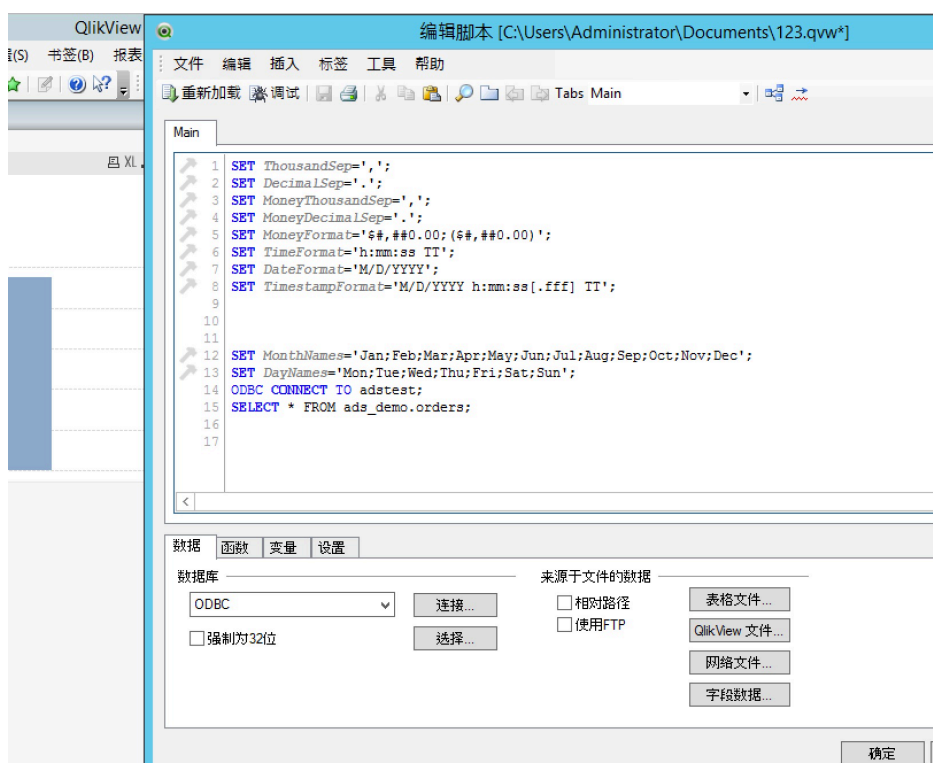
要使用QlikView连接分析型数据库，首先需要配置操作系统的ODBC数据源。安装ODBC MySQL Driver后，进入 控制面板->管理工具->ODBC数据源（不同操作系统此步骤可能不同），新建一个用户DSN，选择“MySQL ODBC 5.xx Driver”一项。



在ODBC TCP/IP服务器处填入连接分析型数据库的域名和端口号（在DMS的数据库选择页面获取），部分情况下，可能需要您将域名解析为IP地址填入；以及将您的账号在阿里云的Access Key ID / Access Key Secret输入到用户名/密码框中，在数据库一项中选择您要连接的数据库，点击测试按钮测试连接通过后，确定。

打开QlikView（建议版本：11.20.x），打开“编辑脚本”，在末端键入：

```
ODBC CONNECT TO adstest;
select * from ads_demo.example_table limit 100;
```



其中adstest为之前在ODBC中建立的数据源名称。之后运行脚本，即可在QlikView中读取和使用分析型数据库中的数据。

## 在Microsoft Excel中读取分析型数据库中的数据

若是要在Excel中读取分析型数据库中的数据，建议安装微软的免费插件：Microsoft Power Query for Excel。需要按照Qlikview一节中相同的方式配置好ODBC数据源之后，参照MySQL相关教程进行。

## 8.5 使用阿里云数据传输实时同步RDS的数据

通过阿里云数据传输（<https://www.aliyun.com/product/dts/>），并使用 dts-ads-writer 插件，可以将您在阿里云RDS中的数据表的变更实时同步到分析型数据库中对应的实时写入表中（RDS端目前暂时仅支持MySQL引擎）。

您需要在您RDS所在的云账号下开通阿里云数据传输服务。并 [点击此处](#) 下载dts-ads-writer插件(8月11日已更新为0.17版本，使用老版本的用户请尽快更新，MD5：0d023dd7d882bc74096e21d0c107af1a)到您的一台服务器上并解压（需要该服务器可以访问互联网，建议使用阿里云ECS以最大限度保障可用性）。服务器上需要有Java 6或以上的运行环境（JRE/JDK）。

### 操作步骤

1. 在分析型数据库上创建目标表，数据更新类型为实时写入，字段名称和MySQL中的建议均相同，字段类型的映射见本文档 2.2节；
2. 在阿里云数据传输的控制台上创建数据订阅通道（见：

[https://help.aliyun.com/document\\_detail/dts/Getting-Started/data-subscription.html](https://help.aliyun.com/document_detail/dts/Getting-Started/data-subscription.html) ) , 并记录这个通道的ID ;

3. 配置dts-ads-writer/app.conf文件 , 配置方式详见下文 ;
4. 运行dts-ads-writer/bin/startup.sh(sh bin/startup.sh) ;
5. 配置监控程序监控进程存活和日志中的常见错误码 ( 见下文 ) 。

## 插件目录结构

```
dts-ads-writer\  
|--bin\  
|--lib\  
|--log\  
|--tmp\  
|--dts-ads-writer.jar  
|--app.conf  
|--dts-ads-writer.pid
```

各目录和文件的含义 :

- bin: 保存了启动和停止进程的脚本(startup.sh和stop.sh), 停止进程请用stop.sh执行 : sh stop.sh
- lib: 相关java依赖包
- log: 运行时产生的日志
- tmp: 临时目录
- dts-ads-writer.jar: 插件的主程序包
- app.conf: 配置文件
- dts-ads-writer.pid: 插件启动后的进程id

## 配置项

所有配置均保存在app.conf中 , 运行前请保证配置正确 ; 修改配置后 , 请重启writer

基本配置 :

```
{  
  "dtsAccessId": "", // 拥有数据订阅通道的云账号的accessId, 必须配置  
  "dtsAccessKey": "", // 拥有数据订阅通道的云账号的accessKey, 必须配置  
  "dtsTunnelId": "", // 数据订阅通道的id, 必须配置; 注意是id, 不是通道名称  
  "adsUserName": "", // 访问您的分析型数据库的用户名 ( accessId ), 必须配置  
  "adsPassword": "", // 访问您的分析型数据库的密码(accessKey), 必须配置  
  "adsJdbcUrl": "", // 访问分析型数据库的jdbc连接串, 必须配置 ( 格式jdbc:mysql://ip:port/dbname )  
  "tables": [  
    {  
      "source": {  
        "primaryKeys": [] // 主键定义, 必须配置; 注意RDS和分析型数据库中的主键定义必须一致  
        "db": "", // 源头RDS的db名称, 必须配置  
        "table": "", // 源头RDS的table名称, 必须配置  
        "skipColumns": ["col1"] // 可选, 若在此配置了RDS表某列名, 则该列不会同步  
      },  
      "target": {  
        "table": "" // 目标分析型数据库的table名称, 必须配置  
      }  
    }  
  ]  
}
```

```
"target": {
  "table": "" // 分析型数据库表的table名称, 必须配置
},
"columnMapping": {
  "": "" // rds表和ads表的列对应关系: key为rds的列名, value为分析型数据库的列名, 选填, 不填则按照列名——对应
}
}
]
```

tables节点的配置示例, 表示rds\_db库下的rds\_table表对应ads\_table表, 并且rds\_table表的col1列对应ads\_table表的col1\_ads列, rds\_table表的col2列对应ads\_table表的col2\_ads列

```
"tables": [
{
  "source": {
    "primaryKeys": [
      "col1",
      "col2"
    ],
    "db": "rds_db",
    "table": "rds_table"
  },
  "target": {
    "table": "ads_table"
  },
  "columnMapping": {
    "col1": "col1_ads",
    "col2": "col2_ads"
  }
}
]
```

## 注意事项

1. RDS表和分析型数据库中表的主键定义必须完全一致; 如果不一致会出现数据不一致问题。如果需要调整RDS/分析型数据库表的主键, 建议先停止writer进程;
2. 一个插件进程中分析型数据库db只能是一个, 由adsJdbcUrl指定;
3. 一个插件进程只能对应一个数据订阅通道; 如果更新通道中的订阅对象时, 需要重启进程
4. RDS中DDL操作不做同步处理;
5. 更新app.conf需要重启插件进程才能生效;
6. 如果工具出现bug或某种其它原因需要重新同步历史数据, 只能回溯最近24小时的数据(在阿里云数据传输的控制台中修改消费位点);
7. 插件的最大同步性能与运行插件的服务器的互联网带宽和磁盘IOPS成正比。

## 常见错误码

logs目录下的日志中的异常信息均以ErrorCode=XXXX ErrorMessage=XXXX形式给出, 可以进行监控, 具体如下:

| 错误码(ErrorCode)         | 错误信息(ErrorMessage)                                                               | 含义                          | 解决                                      |
|------------------------|----------------------------------------------------------------------------------|-----------------------------|-----------------------------------------|
| ConfigFileNotFound     | missing config file: app.conf                                                    | app.conf文件没有找到              | 确认app.conf和writer.jar文件在同一路径            |
| MissingConfig          | missing required config: XXXX(注: XXXX为具体的配置名)                                    | XXXX对应的配置项不存在               | 补充相应的配置                                 |
| InvalidTableDefinition | missing RDS table definition at position: XXXX                                   | 第XXXX项tables定义缺少rds相关定义     | 检查第XXXX项tables定义, 补充rds相应的配置            |
| InvalidTableDefinition | missing ADS table definition at position: XXXX                                   | 第XXXX项tables定义缺少目标相关定义      | 检查第XXXX项tables定义, 补充目标相应的配置             |
| InvalidTableDefinition | missing required config[db] for RDS table definition at position: XXXX           | 第XXXX项rds定义缺少db配置项          | 检查第XXXX项tables定义, 补充rds表的db名称           |
| InvalidTableDefinition | missing required config[table] for RDS table definition at position: XXXX        | 第XXXX项rds定义缺少table配置项       | 检查第XXXX项tables定义, 补充rds表的table名称        |
| InvalidTableDefinition | missing required config[table] for ADS table definition at position: XXXX        | 第XXXX项ads定义缺少table配置项       | 检查第XXXX项tables定义, 补充ads表的table名称        |
| DtsClientError         | SignatureDoesNotMatch : Specified signature is not matched with our calculation. | dtsAccessId和dtsAccessKey不正确 | 检查dtsAccessId和dtsAccessKey是否为创建dts通道的账号 |
| DtsClientError         | get guid info failed                                                             | dtsTunnelId配置不正确            | 从阿里云控制台确认dtsTunnelId是否为正确的dts通道id       |
| SqlExecutionError      | execute sql exception: XXXX(注: XXXX为具体的错误)                                       | 分析型数据库执行sql报错               | 联系技术支持查看失败原因                            |
| InternalError          | XXXX                                                                             | writer内部错误                  | 联系技术支持查看失败原因                            |

在0.8.43或0.9.7以后的版本的阿里云分析型数据库中, 支持通过阿里云访问控制(<https://ram.console.aliyun.com/>) 创建的子账号登录分析型数据库, 并管理子账号在不同条件下是否有使用分析型数据库的权限。

主账号在阿里云访问控制的控制台中, 可以新建多个子账号, 通过授予对应的授权策略, 使子账号在一定条件下可以访问分析型数据库。子账号访问分析型数据库的MySQL协议端时需要使用其的Access Key ID/Secret作

为用户名和密码。若在访问控制中允许子账号登录阿里云控制台，子账号亦可登录分析型数据库的控制台DMS。

在阿里云访问控制中，授权一个子账号可以访问分析型数据库，可以使用系统内置的授权策略AliyunAnalyticDBFullAccess 授予子账号权限。在需要添加更多条件，或所在环境中找不到系统内置的授权策略的，可以按照如下示例的方式制定访问策略(限制访问来源IP必须是)：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "ads:*",
      "Resource": "*",
      "Effect": "Allow"
    }
  ],
  "Condition": {
    "IpAddress": {
      "acs:SourceIp": ["42.120.66.0/24"]
    }
  }
}
```

此策略限制该用户只能在42.120.66.0/24网段访问分析型数据库。详细的策略编写方法详见[https://help.aliyun.com/document\\_detail/28663.html](https://help.aliyun.com/document_detail/28663.html)。

注意一旦授予子账号相关访问策略，子账号将继承主账号在分析型数据库的全部权限（除ACL授权相关权限），包括主账号作为owner的数据库的权限，以及主账号被授权的其他数据库的相关被授予的权限。另外需要注意，目前暂不支持STS Token访问分析型数据库。

## 附录

### 附录一 元数据库数据字典

分析型数据库的元数据库分为记载性能相关信息的performance\_schema和记载元数据的information\_schema，并和MySQL的元数据库有一定的兼容性，但是不是100%一致。

查询元数据库可以直接在JDBC连接中使用SQL语句进行查询，如：

```
SELECT state
FROM information_schema.current_job
WHERE table_schema='db_name'
AND table_name='table_name'
ORDER BY start_time DESC
```

LIMIT 10

即可查询分析型数据库某表近期的10次数据导入的状态。

## information\_schema 库

### SCHEMATA

- SCHEMATA表提供了关于数据库的信息

| FIELD                      | TYPE         | 是否为分析型数据库扩展字段 | COMMENT                |
|----------------------------|--------------|---------------|------------------------|
| CATALOG_NAME               | varchar(512) | N             | 分析型数据库中返回NULL          |
| SCHEMA_NAME                | varchar(64)  | N             | 数据库名称                  |
| DEFAULT_CHARACTER_SET_NAME | varchar(32)  | N             | 字符集名称.分析型数据库中返回Unicode |
| DEFAULT_COLLATION_NAME     | varchar(32)  | N             | 字符校验规则名称.分析型数据库中返回OFF  |
| SQL_PATH                   | varchar(512) | N             | 分析型数据库中返回null          |
| CREATOR_ID                 | varchar(512) | Y             | 创建数据库的云账号数字id          |
| CREATOR_NAME               | varchar(512) | Y             | 创建数据库的云账号名称            |
| CREATE_TIME                | timestamp    | Y             | 数据库创建时间                |
| DOMAIN_URL                 | varchar(512) | Y             | 数据库的连接地址（域名:端口）        |
| VIP                        | varchar(512) | Y             | 数据库的连接ip地址             |
| PORT                       | varchar(512) | Y             | 数据库的连接端口号              |
| DISABLED                   | tinyint      | Y             | 若为1则数据库被禁用（通常是已欠费）     |

### TABLES

- TABLES表提供数据库表信息。该部分数据包括表的元数据与部分表对应数据的元数据，如分区信息等。

| FIELD        | TYPE        | 是否为分析型数据库扩展字段 | COMMENT |
|--------------|-------------|---------------|---------|
| TABLE_SCHEMA | varchar(64) | N             | 数据库名称   |
| TABLE_NAME   | varchar(64) | N             | 表名称     |

|                          |              |   |                                               |
|--------------------------|--------------|---|-----------------------------------------------|
| TABLE_GROUP              | varchar(64)  | Y | 表组名称                                          |
| TABLE_TYPE               | varchar(64)  | N | 分析型数据库中返回:PARTITION_TABLE/<br>DIMENSION_TABLE |
| ENGINE                   | varchar(64)  | N | 数据库引擎名称,分析<br>型数据库中目前返回<br>ANALYSISTABLE      |
| CREATE_TIME              | timestamp    | N | 创建时间                                          |
| UPDATE_TIME              | timestamp    | N | 更新时间                                          |
| CREATOR_ID               | varchar(512) | Y | 创建数据库的云账号数<br>字id                             |
| CREATOR_NAME             | varchar(512) | Y | 创建数据库的云账号名<br>称                               |
| CLUSTER_BY_COLU<br>MNS   | varchar(512) | Y | 该表的聚集列                                        |
| PRIMARY_KEY_COLU<br>MNS  | varchar(512) | Y | 主键列                                           |
| CREATOR_NAME             | varchar(512) | Y | 创建数据库的云账号名<br>称                               |
| FROM_CTAS                | tinyint      | Y | 是否使用create<br>table as select创建的<br>, 0-否 1-是 |
| PARTITION_TYPE           | varchar(512) | Y | 分区类型<br>, DIM/HASH                            |
| PARTITION_COLUM<br>N     | varchar(64)  | Y | 一级分区列名                                        |
| PARTITION_COUNT          | int          | Y | 一级分区数                                         |
| IS_SUB_PARTITION         | tinyint      | Y | 有无二级分区, 0无<br>1有                              |
| SUB_PARTITION_TYP<br>E   | varchar(512) | Y | 二级分区列类型, 目前<br>如不为空都是LIST                     |
| SUB_PARTITION_CO<br>LUMN | varchar(64)  | Y | 二级分区列名                                        |
| SUB_PARTITION_CO<br>UNT  | int          | Y | 二级分区个数                                        |
| UPDATE_TYPE              | varchar(512) | Y | 数据更新类型<br>, batch/realtime                    |

## COLUMNS

- 提供所有表的列信息

| FIELD | TYPE | 是否为分析型数据库扩 | COMMENT |
|-------|------|------------|---------|
|-------|------|------------|---------|

|                          |                     | 展字段 |                                                                                                                      |
|--------------------------|---------------------|-----|----------------------------------------------------------------------------------------------------------------------|
| TABLE_CATALOG            | varchar(512)        | N   | NULL                                                                                                                 |
| TABLE_SCHEMA             | varchar(64)         | N   | 数据库名称                                                                                                                |
| TABLE_NAME               | varchar(64)         | N   | 表名称                                                                                                                  |
| COLUMN_NAME              | varchar(64)         | N   | 列名称                                                                                                                  |
| ORDINAL_POSITION         | bigint(21) unsigned | N   | 列编号, 从1开始                                                                                                            |
| COLUMN_DEFAULT           | varchar(1024)       | N   | 默认值                                                                                                                  |
| IS_NULLABLE              | varchar(3)          | N   | 是否可以为空. 默认为<br>, ,                                                                                                   |
| IS_PRIMARYKEY            | varchar(3)          | N   | 是否为主键. 默认为<br>, ,                                                                                                    |
| IS_AUTOINCREMENT         | varchar(3)          | N   | 是否为自增. 默认为<br>, ,                                                                                                    |
| DATA_TYPE                | varchar(64)         | N   | 数据类型, 比如<br>varchar. 默认为' '                                                                                          |
| CHARACTER_MAXIMUM_LENGTH | bigint(21) unsigned | N   | 字符最大长度, 以字符<br>为单位. 默认为<br>NULL. 如果是大对象<br>类型, 返回-1                                                                  |
| CHARACTER_OCTET_LENGTH   | bigint(21) unsigned | N   | 最大长度, 以字节为单<br>位. 默认为NULL. 大对<br>象列类型, 返回-1                                                                          |
| NUMERIC_PRECISION        | bigint(21) unsigned | N   | 数据最大精度. 分析型<br>数据库中返回NULL                                                                                            |
| NUMERIC_SCALE            | bigint(21) unsigned | N   | 小数位数. 分析型数据<br>库中返回NULL                                                                                              |
| CHARACTER_SET_NAME       | varchar(32)         | N   | 字符集名称. 分析型数<br>据库中返回NULL                                                                                             |
| COLLATION_NAME           | varchar(32)         | N   | 校验规则名称. 默认为<br>NULL                                                                                                  |
| COLUMN_TYPE              | varchar(1024)       | N   | 列类型, 目前支持类型<br>:boolean/short/int/b<br>igint/float/double/d<br>ate/time/timestamp/<br>string(varchar)/multi<br>value |
| COLUMN_KEY               | varchar(3)          | N   | 字段的键类型. 分析型<br>数据库中返回' '                                                                                             |
| EXTRA                    | varchar(27)         | N   | 附加信息, 目前返回为<br>, ,                                                                                                   |
| PRIVILEGES               | varchar(80)         | N   | 该列对应权限, 分析型<br>数据库中返回<br>' SELECT' (大写)                                                                              |

|                |              |   |            |
|----------------|--------------|---|------------|
| COLUMN_COMMENT | varchar(255) | N | 列注释.默认为' ' |
|----------------|--------------|---|------------|

## PARTITIONS

- 提供表分区信息。注意该表数据包含所有历史版本数据，所以不能用于计算表的存储占用、数据条数等。

| FIELD                         | TYPE                | 是否为分析型数据库扩展字段 | COMMENT                                            |
|-------------------------------|---------------------|---------------|----------------------------------------------------|
| TABLE_CATALOG                 | varchar(512)        | N             | NULL                                               |
| TABLE_SCHEMA                  | varchar(64)         | N             | 数据库名称                                              |
| TABLE_NAME                    | varchar(64)         | N             | 表名                                                 |
| PARTITION_NAME                | varchar(64)         | N             | 分区名称                                               |
| SUBPARTITION_NAME             | varchar(64)         | N             | 二级分区名称                                             |
| PARTITION_ORDINAL_POSITION    | bigint(21) unsigned | N             | 分区编号，从1开始                                          |
| SUBPARTITION_ORDINAL_POSITION | bigint(21) unsigned | N             | 子分区编号，从1开始                                         |
| PARTITION_METHOD              | varchar(12)         | N             | RANGE, LIST, HASH, LINEAR HASH, KEY, or LINEAR KEY |
| SUBPARTITION_METHOD           | varchar(12)         | N             | 子分区分区方法                                            |
| PARTITION_EXPRESSION          | varchar(1024)       | N             | 分区表达式                                              |
| SUBPARTITION_EXPRESSION       | varchar(1024)       | N             | 子分区表达式                                             |
| PARTITION_DESCRIPTION         | varchar(1024)       | N             | 分区描述                                               |
| TABLE_ROWS                    | bigint(21) unsigned | N             | 该分区表中的记录数。<br>innodb中是预估的                          |
| AVG_ROW_LENGTH                | bigint(21) unsigned | N             | 平均行记录长度，in bytes                                   |
| DATA_LENGTH                   | bigint(21) unsigned | N             | 数据长度，in bytes                                      |
| MAX_DATA_LENGTH               | bigint(21) unsigned | N             | 最大数据长度                                             |
| INDEX_LENGTH                  | bigint(21) unsigned | N             | 索引长度                                               |
| DATA_FREE                     | bigint(21) unsigned | N             | 空闲的长度,in bytes                                     |
| CREATE_TIME                   | datetime            | N             |                                                    |
| UPDATE_TIME                   | datetime            | N             |                                                    |

|                   |                     |   |                           |
|-------------------|---------------------|---|---------------------------|
| CHECK_TIME        | datetime            | N | 分区最后一次检查时间                |
| CHECKSUM          | bigint(21) unsigned | N | 分区的checksum               |
| PARTITION_COMMENT | varchar(80)         | N | 注释                        |
| NODEGROUP         | varchar(12)         | N | 所属的节点组。分析型数据库中置为“ ”       |
| TABLESPACE_NAME   | varchar(64)         | N | 现在都是Default. 分析型数据库中置为“ ” |
| PARTITION_NO      | bigint(21) unsigned | Y | 分区号，从0开始                  |
| DATA_DISK_SIZE    | bigint(21) unsigned | Y | 该分区占用的磁盘空间                |
| COMPRESS_RATIO    | float               | Y | 数据压缩率                     |
| MEM_SIZE          | bigint(21) unsigned | Y | 装载该分区需要的内存空间              |
| ZIP_SIZE          | bigint(21) unsigned | Y | 该分区压缩包的大小                 |

## INDEXES

- 提供关于表索引的信息，目前已废弃

| FIELD        | TYPE        | 是否为分析型数据库扩展字段 | COMMENT                        |
|--------------|-------------|---------------|--------------------------------|
| TABLE_SCHEMA | varchar(64) | N             | 数据库名称。不分区大小写                   |
| TABLE_NAME   | varchar(64) | N             | 表名称。不区分带瞎写                     |
| INDEX_NAME   | varchar(64) | N             | key的名称                         |
| COLUMN_NAME  | varchar(64) | N             | 列名称                            |
| Asc_Or_Desc  | varchar(1)  | N             | 是否为倒序                          |
| INDEX_TYPE   | varchar(16) | N             | 用过的索引方法。分析型数据库中返回 BITMAP, HASH |
| COMMENT      | varchar(16) | N             | 注释                             |

## CURRENT\_JOB

- 数据批量导入任务或实时表的optimize table的任务信息，通常只保留最近24小时的任务

| FIELD        | TYPE        | 是否为分析型数据库扩展字段 | COMMENT |
|--------------|-------------|---------------|---------|
| TABLE_SCHEMA | varchar(64) | Y             | 数据库名称   |
| TABLE_NAME   | varchar(64) | Y             | 表名称     |

|                |              |   |                                      |
|----------------|--------------|---|--------------------------------------|
| TABLE_GROUP    | varchar(64)  | Y | 表组名称                                 |
| DATA_VERSION   | bigint       | Y | 数据版本号                                |
| JOB_ID         | varchar(64)  | Y | 任务ID，任务的唯一标识                         |
| STATE          | varchar(64)  | Y | 任务状态，INITED/RUNNING/SUCCEEDED/FAILED |
| USER           | varchar(64)  | Y | 发起任务的用户名                             |
| START_TIME     | timestamp    | Y | 任务启动时间                               |
| FINISH_TIME    | timestamp    | Y | 任务完成时间（若任务没有完成则是最后状态更新的时间）           |
| SOURCE_PATH    | varchar(512) | Y | 数据来源                                 |
| PARTITION_PATH | varchar(64)  | Y | 在分析型数据库的分区列的信息                       |
| ERROR_CODE     | varchar(512) | Y | 错误代码，目前恒为空                           |
| ERROR_MSG      | varchar(512) | Y | 错误信息，目前恒为空                           |

## CURRENT\_INSTANCES

- 展示该数据库的进程信息

| FIELD          | TYPE        | 是否为分析型数据库扩展字段 | COMMENT                                    |
|----------------|-------------|---------------|--------------------------------------------|
| TABLE_SCHEMA   | varchar(64) | Y             | 数据库名称                                      |
| WORKER_ID      | int         | Y             | 进程ID                                       |
| WORKER_TYPE    | varchar(64) | Y             | 进程类型<br>(FRONTNODE/COMPUTENODE/BUFFERNODE) |
| RESOURCE_TYPE  | varchar(64) | Y             | ECU类型                                      |
| INSTANCE_STATE | varchar(64) | Y             | 状态，目前为NEW或RUNNING均为正常                      |
| MEM_SIZE       | BIGINT      | Y             | 内存配额大小                                     |
| DISK_SIZE      | BIGINT      | Y             | 磁盘配额大小                                     |
| DISK_USED      | BIGINT      | Y             | 磁盘当前使用大小，COMPUTENODE的磁盘使用为实际的数据存储          |

## RESOURCE\_REQUEST

- 展示该数据库的扩容、缩容记录| FIELD | TYPE | 是否为分析型数据库扩展字段 | COMMENT||:-----|:-----|:-----|:-----:| TABLE\_SCHEMA | varchar(64) |Y|数据库名称|| REQUEST\_ID | varchar(64) |Y|请求ID|| REQUEST\_TYPE | varchar(64) |Y|请求类型，扩容ALLOCATE，缩容DELOCATE|| RESOURCE\_TYPE | varchar(64) |Y|ECU类型|| RESOURCE\_COUNT | int |Y|目标ECU数量|| CREATE\_TIME | timestamp |Y| 创建时间|| UPDATE\_TIME | timestamp |Y| 更新时间|| STATE | varchar(64) |Y| 任务状态，ERROR/RUNNING/SUCCEEDED|| ERROR\_CODE | varchar(64) |Y|错误码|| ERROR\_MSG | varchar(64) |Y|错误详细信息|| USER\_ID | varchar(64) |Y|发起者账号ID|| USER\_NAME | varchar(64) |Y|发起者账号名|

## performance\_schema 库

### query\_profile

- 该表存储了最近30天内一个数据库的每一条查询的性能信息。该表不可用select \* 查询。

| FIELD        | TYPE         | 是否为分析型数据库扩展字段 | COMMENT         |
|--------------|--------------|---------------|-----------------|
| TABLE_SCHEMA | varchar(512) | Y             | 数据库名称           |
| TABLE_NAME   | varchar(64)  | Y             | 数据库名称           |
| PID          | varchar(32)  | Y             | 查询id            |
| CREATE_TIME  | timestamp    | Y             | 查询时间            |
| SQL          | varchar(512) | Y             | sql语句，目前为空      |
| SUCCESS      | tinyint      | Y             | 查询是否成功，0否<br>1是 |
| RT           | float        | Y             | 查询响应时间，单位毫秒     |
| USER_NAME    | varchar(512) | Y             | 发起查询的用户名称       |
| ROW_COUNT    | int          | Y             | 查询返回记录的条数       |

## 附录二：SQL函数表

### 特色函数

#### UDF\_SYS\_COUNT\_COLUMN

作用：用于做多group by的聚合，可以将多个group by的语句合并成多个UDF，写到一条sql中

格式：UDF\_SYS\_COUNT\_COLUMN(columnName, columnName2...), 参数必须是列名

返回：一个json的字符串列，类似：{ "0" :3331656," 2" :3338142," 1" :3330202}

实例：

a. select UDF\_SYS\_COUNT\_COLUMN(c1) from table 等价于select count(\*) from table group by c1

b. select UDF\_SYS\_COUNT\_COLUMN(c1,c2) from table 等价于select count(\*) from table group by c1,c2

c. select UDF\_SYS\_COUNT\_COLUMN(c1),  
UDF\_SYS\_COUNT\_COLUMN(c2),UDF\_SYS\_COUNT\_COLUMN(c3), UDF\_SYS\_COUNT\_COLUMN(c4)

等价于如下四条sql语句：select count() from table group by c1;select count() from table group by c2;select count() from table group by c3;select count() from table group by c4;

d. 一个真实的实例：select UDF\_SYS\_COUNT\_COLUMN(user\_gender),  
UDF\_SYS\_COUNT\_COLUMN(user\_level) from db\_name.userbase返回1行，2列  
：{ "0" :3331656," 2" :3338142," 1" :3330202} , { "0" :4668150," 2" :1891176," 1" :1984606 , " 6" :5818}

## UDF\_SYS\_RANGECOUNT\_COLUMN

作用：用于做静态样本分段(老的函数UDF\_SYS\_SEGCOUNT\_COLUMN已经弃用，请使用该函数)

格式：UDF\_SYS\_RANGECOUNT\_COLUMN(columnName, count, min, max)

- 第一个参数是列名
- 第二个参数分段数目
- 第三个参数是参与分段的该列在全表的最小值
- 第四个参数是参与分段的该列在全表的最大值

返回：一个json的字符串列，类似：{ "ranges" :[{ "start" :0," end" :599},  
{ "start" :600," end" :1899}, { "start" :1900," end" :65326003}]}

使用说明：使用该函数进行动态分段统计，需要三个步骤：

- 第一步，通过min, max求出符合条件的最小值和最大值
- 第二步，通过UDF\_SYS\_RANGECOUNT\_COLUMN获取各个分段。
- 第三步，通过case when+group by获取每个分段中真实聚合数据。

特别说明UDF\_SYS\_RAGNECOUNT\_COLUMN和通过  
UDF\_SYS\_RANGECOUNT\_SAMPLING\_COLUMN的区别在于，前者是静态分段，也就是根据  
(max-min+1)/segcount进行分段，而后是在动态分段，可以保证每个分段区间内的数目是大致均

衡的。

## UDF\_SYS\_GEO\_IN\_CYCLE

作用：用于做基于地理位置的经纬度画圈

格式：UDF\_SYS\_GEO\_IN\_CYCLE(longitude, latitude, point, radius)

- 第一个参数为经度列名称，类型float
- 第二个参数为纬度列名称，类型float
- 第三个参数为圆圈中心点的位置，格式=>“ 经度，纬度” ，=>“ 120.85979,30.011984”
- 第四个参数为圆圈的半径，单位米

返回：返回一个boolean值

使用说明：

```
select count(*) db_name.usertag where udf_sys_geo_in_cycle(longitude,latitude,
    "120.85979,30.011984" , 5000)=true求以“ 120.85979,30.011984” 为中心点，半径为5km的
    圆圈内的人数
```

```
select longitude,latitude from db_name.usertag where
    udf_sys_geo_in_cycle(longitude,latitude, "120.85979,30.011984" , 5000)=true order by
    longitude
```

## UDF\_SYS\_GEO\_IN\_RECTANGLE

作用：用于做基于地理位置的经纬度画矩形

格式：UDF\_SYS\_GEO\_IN\_RECTANGLE(longitude, latitude, pointA, pointB)

- 第一个参数为经度列名称, 类型float
- 第二个参数为纬度列名称, 类型float
- 第三个参数为矩形的左下角坐标，格式=>“ 经度，纬度” ，=>“ 120.85979,30.011984”
- 第四个参数为矩形的右上角坐标，格式=>“ 经度，纬度” ，=>“ 120.88450,31.21011”

返回：返回一个boolean值

使用说明：

```
select count(*) db_name.usertag where udf_sys_geo_in_rectangle(longitude,latitude,
    "120.85979,30.011984" , "120.88450,31.21011" )=true求以“ 120.85979,30.011984” 和
    “ ” 120.88450,31.21011” ” 为2个斜角构成的矩形圈内的人数
```

## UDF\_SYS\_GEO\_DISTANCE

作用：用作一个经纬度列和一个固定的坐标点的距离计算

格式：UDF\_SYS\_GEO\_DISTANCE(longitude, latitude, pointA)

- 第一个参数为经度列名称, 类型float
- 第二个参数为纬度列名称, 类型float
- 第三个参数为固定坐标点的经纬度, 格式=> " 经度, 维度" ,  
=> " 120.85979,30.011984"

返回：返回一个int值, 单位为米(M)

使用说明：

```
select count(*) db_name.usertag where udf_sys_geo_in_rectangle(longitude,latitude,  
"120.85979,30.011984" , "120.88450,31.21011" )=true求以" 120.85979,30.011984" 和  
" " 120.88450,31.21011" " 为2个斜角构成的矩形圈内的人数
```

## 字符串函数

### Concat

连接2个字符串, 格式类似concat(str1, str2,...)

### Lcase

返回字符串的全小写

### Ucase

返回字符串的全大写

### Length

返回字符串的长度

### Substring

返回字符串的字串SUBSTRING (str, pos , [len])这里的pos为下标从1开始, 比如substring( 'abc' , 2,2)返回的是' bc'

### Trim

类似java string的trim, 用于除去字符串前后的space

## MID

MID 函数用于从文本字段中提取字符。 MID(column\_name,start[,length])

- column\_name 必需。要提取字符的字段。
- start 必需。规定开始位置（起始值是 1）
- length 可选, 要返回的字符数。如果省略，则MID函数返回剩余文本。

## Left/Right

Left/Right(str,len) 返回从字符串str 开始的len 最左/右字符

## Reverse

reverse(str) 返回字符串str的倒叙内容，如 reverse( 'a,b,c,d' )返回' d,c,b,a'

## instr

instr(str\_col, str) 返回字符串str在某一个字段str\_col的内容中的位置, 没有找到字符串返回0，否则返回位置（从1开始）

## 日期函数

### Year

查询指定列的年份，例：year(date\_test);

### Month

查询指定列的月份，例：month(date\_test);

### Day

查询指定列的日，例：day(date\_test);

### Week

查询指定列的是所属年的第几周，支持mode参数（与MySQL定义相同），例：week(date\_test, 1);

### WeekDay

返回日期d是星期几的索引（位置），0表示星期一，1表示星期二，...，6表示星期日。

### WeekofYear

相当于 WEEK(d,3)

## Hour/Minute/Second

返回一个Timestamp的小时/分钟/秒，例：hour(time\_stamp\_test)

## to\_days

给定一个日期date,返回一个天数 (从年份0开始的天数)，用法 to\_days(date)

## Yearmonth

查询指定列的日和月，例如YEARMONTH( '20140602' )=201406;

## Curdate

查询当前日期，例：curdate();

## DateDiff

datediff(date1, date2) 返回date1 date2两个日期之间相差的天数，也可以传入timestamp类型

## from\_unixtime

from\_unixtime(time\_int[, format\_str]) time\_int为Unix时间戳，该函数将Unix时间戳转换为字符串格式的日期时间。若不传入format\_str参数，则返回“ yyyy-MM-dd hh:MM:ss” 格式的字符串。如果传入format\_str，则format\_str中的参数格式中：yyyy代表年，MM代表月，dd代表日，hh代表小时，MM代表分钟，ss代表秒。

## unix\_timestamp

unix\_timestamp(date) 返回一个date或timestamp类型数据的Unix时间戳（整形）。

## 聚合函数

### sum

求分组中一列所有数据的和。必须输入数值类型。

### count([distinct] )

求分组中的记录数。

若使用count(distinct col\_name)去重，group by中的列或col\_name建议为分区列；如果不是分区列，则不能和其他聚合函数在同一个sql语句中使用。

### avg

求分组中一列所有数据的平均数。必须输入数值类型

group\_concat

字符串聚合函数。

目前仅支持在聚合时Group By中包括所有参与计算的表分分区列（维度表）时可以使用。

语法：GROUP\_CONCAT([DISTINCT] expr [,expr ...] [ORDER BY col\_name [ASC | DESC] [,col\_name ...]] [SEPARATOR str\_val])

distinct用于将组内多个相同的字符串仅输出一个。若含有ORDER BY结构，则组内字符串会根据col\_name列表排序输出。使用SEPARATOR可以指定聚合后的字符串分隔符，默认是逗号。

min/ max

求一个分组中的列的最小、最大值。支持传入字符串或数值类型。

其它函数

CAST

转换当前列的数据类型，例: cast(string\_test as bigint)

Coalesce

返回第一个非null的表达式, 使用方式：coalesce(expression [,...n])例：coalesce(string\_test, "" );

Bit\_Count

Bit\_Count(value)：返回 value 的二进制转换中“ 1 ”字节的个数

附录三 常见错误码表

| 类别    | 错误码       | SQL状态码 | 错误信息                                                 | 处理方法                            |
|-------|-----------|--------|------------------------------------------------------|---------------------------------|
| DML异常 | -1(65535) | HY000  | Reject execution of sql with #key#, statement: #sql# | SQL包含被过滤的关键字，联系阿里云团队确认          |
| DML异常 | 1047      | 08S01  | No database selected                                 | 修改语句，指定 database schema，修改后重新执行 |
| DDL异常 | 1064      | HY000  | ShowSyntaxException: You have an error in            | 检查SHOW语句语法                      |

|       |      |       |                                                                                                                       |                                     |
|-------|------|-------|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------|
|       |      |       | your SQL syntax                                                                                                       |                                     |
| 权限异常  | 1142 | 42000 | DenyAccessExc<br>eption:User[#us<br>er#] does not<br>have[#action#]<br>access to<br>resource[#reso<br>urce#]          | 确认用户的执行<br>权限                       |
| 权限异常  | 1142 | 42000 | NotAllowedAccess<br>Exception:User[<br>#user#] does<br>not<br>have[#action#]<br>access to<br>resource[#reso<br>urce#] | 确认用户的执行<br>权限                       |
| 权限异常  | 1142 | 42000 | AclException:Us<br>er[#user#] does<br>not<br>have[#action#]<br>access to<br>resource[#reso<br>urce#]                  | 确认用户的执行<br>权限                       |
| DML异常 | 1149 | 42000 | SQL parsing<br>failed(SQL<br>syntax error):<br>#Parse error<br>message#                                               | 检查SQL语法<br>, 修改后重新执<br>行            |
| DML异常 | 1149 | 42000 | SQL parsing<br>unknown error:<br>#Parse error<br>message#                                                             | 检查SQL语法<br>, 修改后重新执<br>行            |
| DML异常 | 1235 | 42000 | SQL dump data<br>selecting<br>columns count<br>exceed limit:<br>#dump_column<br>_count_limit#                         | DUMP列超过系<br>统允许的最大值                 |
| DML异常 | 1235 | 42000 | SQL selecting<br>columns count<br>exceed limit:<br>#select_column<br>_count_limit#                                    | 确认需要<br>select的列数超<br>过系统允许的最<br>大值 |
| DML异常 | 1235 | 42000 | SQL feature<br>NOT supported<br>yet: SELECT or<br>SELECT #tb#.                                                        | 修改SQL, 重新<br>执行                     |
| DML异常 | 1236 | 42000 | SQL feature<br>NOT supported<br>yet: SELECT                                                                           | 修改SQL, 采用<br>子查询方式, 重<br>新执行        |

|       |      |       |                                                                                                                               |                                                                   |
|-------|------|-------|-------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
|       |      |       | agg_func() ...<br>UNION/UNION<br>ALL/INTERSECT<br>SELECT<br>agg_func() ...                                                    |                                                                   |
| DML异常 | 1236 | 42000 | SQL feature<br>NOT supported<br>yet: SELECT ...<br>ORDER BY ...<br>UNION/UNION<br>ALL/INTERSECT<br>SELECT ...<br>ORDER BY ... | 修改SQL，目前<br>分布式查询模式<br>下，UNION分支<br>中ORDER BY无<br>意义，考虑修改<br>查询逻辑 |
| DML异常 | 1236 | 42000 | SQL feature<br>NOT supported<br>yet: SELECT ...<br>LIMIT X<br>UNION/UNION<br>ALL/INTERSECT<br>SELECT ... LIMIT<br>X           | 修改SQL，目前<br>分布式查询模式<br>下，UNION分支<br>中LIMIT无意义<br>，考虑修改查询<br>逻辑    |
| DDL异常 | 3080 | HY000 | AcquireLockExc<br>eption:lockNam<br>e =<br>#schema_table<br>#                                                                 | 提交重新执行                                                            |
| DDL异常 | 3081 | HY000 | IllegalParamete<br>rException:<br>paramaterNam<br>e=#parameter<br>name#                                                       | 检查DDL语句参<br>数                                                     |
| DDL异常 | 3081 | HY000 | IllegalParamete<br>rException:<br>paramaterNam<br>e=Column type<br>is invalid:<br>#data_type#                                 | 检查DDL列的数<br>据类型                                                   |
| DDL异常 | 3082 | HY000 | IllegalParamete<br>rValueExceptio<br>n: [DDL-<br>N3000]:<br>groupName<br>should be<br>provided for<br>DIMENSION tb.           | 检查DDL语句参<br>数,为维度表指定<br>groupName                                 |
| DDL异常 | 3082 | HY000 | IllegalParamete<br>rValueExceptio<br>n: [DDL-<br>N3001]:<br>PARTITION<br>NUM should =<br>1 for<br>DIMENSION tb.               | 检查DDL语句参<br>数,维度表分区数<br>必须为1                                      |

|       |      |       |                                                                                                                                                |                                                  |
|-------|------|-------|------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N3002]: hashPartitionNum should be provided and set value to 1 for DIMENSION tb.                          | 检查DDL语句参数,为维度表指定hashPartitionNum, 值必须为1          |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N3003]: target tb group is not for DIMENSION tb: #tb group#                                               | 检查DDL语句参数,目标表组为分区表表组, 不能创建维度表                    |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N3004]: target tb group is only for DIMENSION tb: #tb group#                                              | 检查DDL语句参数,目标表组为维度表组, 不能创建分区表                     |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N3005]: can not use DIMENSION_GROUP as target tb group, which is reserved for virtual dimension tb group. | 检查DDL语句参数,不能使用DIMENSION_GROUP作为目标表组, 为内部虚拟维度表组预留 |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N4000]: PARTITION NUM should # 0 for HASH partition.                                                      | 检查DDL语句参数,分区表的分区数必须大于0                           |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-N5000]: originalTableName should not be null.                                                             | 检查DDL语句参数,指定originalTableName                    |
| DDL异常 | 3082 | HY000 | IllegalParameterValueException: [DDL-NFFFF]                                                                                                    | 检查DDL语句参数                                        |

|       |      |       |                                                                                                                   |                                                      |
|-------|------|-------|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
|       |      |       | Unexpected exception: #exception message#                                                                         |                                                      |
| DDL异常 | 3082 | HY000 | IllegalParameterException: parameterName=#parameterName#, parameterValue=#parameterValue#, correct type is #type# | 检查DDL语句参数                                            |
| DDL异常 | 3082 | HY000 | IllegalParameterException: #name# is exceed the valid range, reasonable range:[#min#, #max#]                      | 检查并修改DDL语句参数值范围                                      |
| DDL异常 | 3082 | HY000 | IllegalParameterException:loader.originalTableName is not null                                                    | 检查DDL语句参数,指定originalTableName                        |
| DDL异常 | 3082 | HY000 | IllegalParameterException:loader.originalTableName conflict                                                       | 检查DDL语句参数,目标originalTableName已存在,修改originalTableName |
| DDL异常 | 3082 | HY000 | IllegalParameterException:valueType must is Illegal                                                               | 检查DDL语句参数valueType                                   |
| DDL异常 | 3082 | HY000 | IllegalParameterException:valueType is not null                                                                   | 检查DDL语句参数valueType                                   |
| DDL异常 | 3084 | HY000 | DropTableOperationFailedException:#error message#                                                                 | 联系阿里云技术支持                                            |
| DDL异常 | 3086 | HY000 | CreateTableOperationFailed:Table '#schema#.#tableName#' already exists                                            | 目标表已经存在,无需执行                                         |

|       |       |       |                                                                                           |                                                                |
|-------|-------|-------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| DDL异常 | 3086  | HY000 | CreateTableOperationFailed:put tableInfo failed, sql:#DDL statement#                      | 联系阿里云技术支持                                                      |
| DDL异常 | 3086  | HY000 | CreateTableOperationFailed:put onlineGroupInfo failed, sql:#DDL statement#                | 联系阿里云技术支持                                                      |
| DDL异常 | 3087  | HY000 | CreateTableRollBackException: #error message#                                             | 联系阿里云技术支持                                                      |
| DDL异常 | 3088  | HY000 | AlterTableIsNotExist:alter tb is not exist! tablePath:#tb path#                           | 目标表不存在，确认表的正确性                                                 |
| DDL异常 | 3000  | HY000 | AlterRepeatException:indexName:#indexName# repeat                                         | 目标index已存在                                                     |
| DDL异常 | 3092  | HY000 | DescTableSqlFormatExcpetion: check sql                                                    | 检查并修正 SHOW语法，重新执行                                              |
| DDL异常 | 3094  | HY000 | DropSyntaxException:#DDL statement#                                                       | 检查并修正 DROP DDL语法，重新执行                                          |
| DDL异常 | 3096  | HY000 | AlterSyntaxException:#DDL statement#                                                      | 检查并修正 ALTER DDL语法，重新执行                                         |
| DDL异常 | 3099  | HY000 | ExceedMaxTableNumException : maxTableNumInDB=#maximum tb number allowed in this database# | 目标数据库已到达允许的表数上限，请删除无用表                                         |
| DML异常 | 31000 | HY000 | SQL planning partition routing error: #error message#                                     | 系统错误，联系阿里云技术支持，若消息中含有 DATA_NOT_FOUND，亦有可能是离线导入的表还没第一次导入数据，无法查询 |
| DML异常 | 32000 | HY000 | SQL planning                                                                              | 系统错误，联系                                                        |

|       |       |       |                                                              |                                     |
|-------|-------|-------|--------------------------------------------------------------|-------------------------------------|
|       |       |       | partition routing error: ZERO_ROUTE                          | 阿里云技术支持                             |
| DML异常 | 51000 | HY000 | Partitions query error: all failed; #error message#          | 系统错误，联系阿里云技术支持                      |
| DML异常 | 55000 | HY000 | Partition merging error: #error message#                     | 系统错误，联系阿里云技术支持                      |
| DML异常 | 55001 | HY000 | Partitions query error: partitions miss: #missed part count# | 系统错误，联系阿里云技术支持                      |
| 导入异常  | 1066  | HY000 | unable to extract param#sourcePath#                          | 语法检查失败，检查语句中参数，重新load data          |
| 导入异常  | 1066  | HY000 | job does not exist:                                          | 确认job是否成功提交                         |
| 导入异常  | 1066  | HY000 | check signature exception:                                   | 系统错误，联系阿里云技术支持                      |
| 导入异常  | 1066  | HY000 | target table not exist:                                      | 确认load data中的分析型数据库表存在后再发送load data |
| 导入异常  | 1066  | HY000 | target database not found:                                   | 确认load data中的分析型数据库存在后再发送load data  |
| 导入异常  | 1066  | HY000 | table DB.TABLE hash partition num:XXX more than limit XXX    | 分析型数据库离线导入表表HASH分区数限制为不超过256，需调整分区数 |
| 导入异常  | 1066  | HY000 | table size exceed limit                                      | 暂未使用                                |
| 导入异常  | 1066  | HY000 | source table has no records                                  | 确认odps源表中有数据后再发送load data           |
| 导入异常  | 1066  | HY000 | source table column mapping with dest table error            | 确定分析型数据库表中的列全部在源表中找得到对应的列           |
| 导入异常  | 1066  | HY000 | get odps table error                                         | 确认是否授权garuda_build@                 |

|      |      |       |                                      |                                              |
|------|------|-------|--------------------------------------|----------------------------------------------|
|      |      |       |                                      | aliyun.com对源表/视图的DESC权限                      |
| 导入异常 | 1066 | HY000 | no permission to select source table | 确认是否授权garuda_build@aliyun.com对原表/视图的SELECT权限 |

## 分析型数据库与MySQL 5.x 语法偏离表

[点击下载 \( PDF \)](#)