TRANSWARP
星 环 科 技

Inceptor的使用

陈晓勇
Xiaoyong.chen@transwarp.io
13801795827
星环科技
www.transwarp.io

Inceptor DDL

TRANSWARP 星环科技

- DDL一般由CREATE, DROP和 ALTER开头，作用于DATABASE，TABLE，VIEW等图表对象，对它们进行添加，删除和修改。
- Inceptor SQL可以完成的DDL包括：
  - 创建/删除/修改数据库：CREATE/DROP/ALTER DATABASE
  - 创建/清空/删除表：CREATE/TRUNCATE/DROP TABLE
  - 表的分区：PARTITIONED BY子句
  - 表的分桶：CLUSTERED BY子句
  - 修改表/分区/列：ALTER TABLE/PARTITION/COLUMN
  - 创建/修改/删除视图：CREATE/DROP/ALTER VIEW
  - 创建/删除函数：CREATE/DROP FUNCTION
  - 查看已有数据库/表/函数：SHOW
  - 描述表和数据库：DESCRIBE和DESCRIBE EXTENDED

创建数据库

- 创建数据库
  - CREATE DATABASE语句创建一个给定名称的数据库。在创建时，我们可以选择加上注解，指定目录和附加属性
- 语法：

```
CREATE DATABASE [IF NOT EXISTS] database_name [COMMENT database_comment][LOCATION
hdfs_path] [WITH DBPROPERTIES (property_name=property_value, ...)]
```

- 示例：

```
[$host]transwarp> CREATE DATABASE exchange_platform;
[$host]transwarp> CREATE DATABASE my_db LOCATION
hdfs://ns/inceptor_id/user/hive/warehouse/my/directory';
[$host]transwarp> CREATE DATABASE my_db LOCATION 'inceptor_id/user/hive/warehouse/my/directory';
```

删除数据库

TRANSWARP 星环科技

- 删除数据库
  - DROP DATABASE该语句删除一个给定名字的数据库。
- 语法：

```
DROP DATABASE database_name
```

- 示例：

```
[$host]transwarp> DROP DATABASE exchange_platform;
```

修改数据库

TRANSWARP 星环科技

- 修改数据库
  - ALTER DATABASE
  - 该语句对给定名字的数据库进行修改。用ALTER TABLE可以对数据库做出的修改只有DBPROPERTIES：
- 语法

```
ALTER DATABASE database_name SET DBPROPERTIES (property_name=property_value, ...)
```

- 举例

```
[$host]transwarp> ALTER DATABASE exchange_platform SET DBPROPERTIES ('creater'='Zhang San');
```

设置当前数据库

TRANSWARP 星环科技

- 设置当前数据库
  - USE DATABASE
  - USE指令可以设置当前使用的数据库，进而调用指定数据库里的表和视图，不专门指定，Inceptor默认使用default数据库。
- 语法

```
USE database_name
```

- 举例

```
[$host]transwarp> USE exchange_platform;
transwarp> SELECT * FROM user_info;
[$host]transwarp> USE default: //设置当前数据库为default
transwarp> SELECT * FROM exchange_platform.user_info; //查询exchange_platform下的user_info表
```

基础建表语句：CREATE TABLE

TRANSWARP 星环科技

- 语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)] INTO
num_buckets BUCKETS]
[SKEWED BY (col_name, col_name, ...) ON ([(col_value, col_value, ...), ...|
col_value, col_value, ...])
[STORED AS DIRECTORIES]]
[
[ROW FORMAT row_format]
[STORED AS file_format]
| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
]
[LOCATION hdfs_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement];
```

简单的CREATE TABLE语句

TRANSWARP 星环科技

- CREATE TABLE创建一个给定名称的表。
- 建表有三种方式：
  - 直接定义列
  - CREATE TABLE…LIKE
  - CREATE TABLE…AS SELECT (CTAS)

通过直接定义列创建表

- 创建表时要声明列名和列中数据的数据类型，列中数据类型可以是原生或者复杂类别
- Inceptor会将其创建在默认数据库default里。
  - 加上<code>[db.name.]</code>选项，Inceptor会将表创建在指定的数据库里
  - 也可以使用USE DATABASE，将当前使用数据库设置为指定数据库，然后再创建表:
- 语法

```
CREATE TABLE table_name (col1 col_type1, col2 col_type2, ...)
```

- 示例

```
[$host]transwarp>use  database dbname;
[$host]transwarp> CREATE TABLE dbname.user_info (
name STRING, //用户名字
acc_num STRING, //账户号码
password STRING, //账户密码
citizen_id STRING, //身份证号
bank_acc STRING, //绑定银行账户号码
reg_date STRING, //开户日期
acc_level STRING //账户级别
);
```

创建外表

TRANSWARP 星环科技

- 语法
  - ROW FORMAT和STORED AS子句指明表的存储格式

```
CREATE EXTERNAL TABLE table_name (col1 data_type1, col2 data_type2, ...)
ROW FORMAT row_format
STORED AS file_format
LOCATION 'hdfs_path'
```

- 举例

```
[$host] transwarp> CREATE EXTERNAL TABLE user_info_ext (name STRING, acc_num STRING,
password STRING, citizen_id STRING, bank_acc STRING, reg_date STRING,
acc_level STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/user/root/test';
```

CREATE TABLE...LIKE

TRANSWARP 星环科技

- 我们可以用CREATE TABLE...LIKE来拷贝一个已存在的表的描述(schema)，但不拷贝表中的数据。
- CREATE TABLE...LIKE将一张表的描述完全拷贝，表的描述包括的信息有列名、列值类型、列的注解、表的注解、使用的SerDe等等。
- 语法

```
CREATE TABLE table_name LIKE existing_table_or_view_name
```

- 举例

```
[$host]transwarp> CREATE TABLE empty_user_info LIKE user_info;
transwarp> DESCRIBE FORMATTED user_info;
transwarp> DESCRIBE FORMATTED empty_user_info;
////两张表的描述除了表名和创建时间不同外，其他都相同。
```

CREATE TABLE...AS SELECT (CTAS)

TRANSWARP 星环科技

- 用CTAS我们可创建一个表来存放一次查询的结果。
  - 如果需要经常使用某个查询的结果，可以用CTAS将查询结果存储起来，调用查询结果时就不需要重复计算了。
  - 一个CTAS语句包含两个部分，CREATE部分和SELECT部分。SELECT部分可以是任何SELECT语句，而CREATE部分则将SELECT部分得到的查询结果填入新创建的表中。
- 语法

```
[$host]transwarp> CREATE TABLE table_name AS SELECT select_statement;
```

- 示例

```
[$host]transwarp> CREATE TABLE nonsecure_user_info as SELECT name, acc_num, reg_date,
acc_level from user_info;
[$host]transwarp> select * from user_info;
[$host]transwarp> select * from nonsecure_user_info;
```

创建内存表

TRANSWARP 星环科技

- Inceptor支持建立内存表，内存表中的数据会在机器运行时一直存储在内存中，所以将一些常用查询结果存储在内存表内可以大大提高计算速度。
- Inceptor提供checkpoint机制，将计算数据同步写入HDFS中，可以保证在存储了内存表的机器当机时，内存表中的数据可以从HDFS中直接读取恢复而不需要重新进行查询计算。
- 内存表可以用两种方式创建：
  - 通过CTAS在建表，建表时数据即填入。这种情况下，内存表不能分区或者分桶。
  - 建空内存表，此时内存表可以分区分桶。之后可以通过INSERT INTO SELECT来插入数据。

创建内存表

TRANSWARP 星环科技

- 语法：通过CTAS建表

```
CREATE TABLE table_name TBLPROPERTIES ("cache" = "cache_medium","cache.checkpoint"="true|false",
["filters"="filter_type"]) AS SELECTselect_statment
```

- 语法：建空表，然后插入数据

```
CREATE TABLE table_name (column_name data_type, column_name data_type...) PARTITIONED
BY (partition_key data_type, partition_key data_type) CLUSTERED BY (cluster_key,
cluster_key, ...) INTO n BUCKETS TBLPROPERTIES ("cache" = "cache_medium",
"cache.checkpoint"="true|false", ["filters"="filter_type"])
```

- 说明
  - "cache" = "cache_medium"指定计算缓存的介质。可以选择ram，SSD和memeory。只有当服务器上配置有SSD时，才可以选择SSD作为缓存，Inceptor会自动利用SSD为计算加速。
  - "cache.checkpoint" = "true|false"指定是否设置checkpoint。如果设置checkpoint，查询结果会被同步放入HDFS中，在存储了内存表的机器当机时，内存表中的数据可以从HDFS中直接读取恢复而不需要重新进行查询计算。
  - "filters"="filter_type"为可选项，它指定一个过滤器。利用过滤器可以为某些查询进行优化。

表的分区

- PARTITIONED BY 子句
- Inceptor支持对表的单值分区和范围分区。
- 为一张表分区:
  - 在物理上，将表中的数据按分区放在表目录下的对应子目录中，一个分区对应一个子目录；
  - 在逻辑上，分区表和未分区表没有区别。
- 分区在创建表时完成，也可以通过ALTER TABLE来添加或者删除。
- 语法：创建单值分区表

```
CREATE TABLE table_name (col1 data_type1, col2, data_type2, ... )
PARTITIONED BY (partition_key1 data_type1, partition_key2 data_type2...)
```

表的分区

- 示例

```
[$host]transwarp> CREATE TABLE exchange_platform.partition_user_info (
name STRING,
acc_num STRING,
password STRING,
citizen_id STRING,
bank_acc STRING,
reg_date STRING
) PARTITIONED BY (acc_level STRING);
```

- Inceptor会在表的目录下加上分区对应的子目录：

```
…/part_user_info/acc_level=A
…/part_user_info/acc_level=B
…/part_user_info/acc_level=C
…/part_user_info/acc_level=D
…/part_user_info/acc_level=E
```

- 查询

```
[$host] transwarp> SELECT * FROM partition_user_info WHERE acc_level ='A';
```

  - 执行这个语句时，Inceptor只读取了 /part_user_info/acc_level=A 下的信息，而忽略了其他的子目录。如果part_user_info没有被分区，Inceptor会进行全表扫描来完成这次查询。

表的分区：多层分区

TRANSWARP 星环科技

- 我们还可以对一个表进行多层分区。同样是用户信息表，现在我们用帐户级别和开户日期来分区：

```
[$host]transwarp> CREATE TABLE exchange_platform.doub_partition_user_info (
name STRING,
acc_num STRING,
password STRING,
citizen_id STRING,
bank_acc STRING
) PARTITIONED BY (acc_level STRING, reg_date DATE);
```

- user_info2的子目录将是这样的：

```
……
…/doub_partition_user_info/acc_level=A/reg_date=20121024
…/doub_partition_user_info/acc_level=A/reg_date=20110101
……
…/doub_partition_user_info/acc_level=B/reg_date=20080214
```

- 示例：查询所有2012年1月24日开户的A级用户

```
SELECT * FROM user_info WHERE acc_level ='A' AND reg_date='20121024';
```

表的分区：范围分区

TRANSWARP 星环科技

- 范围分区
  - 每个分区对应分区键的一个区间。
  - 凡是落在指定区间内的记录都会被放入对应的分区下。
  - 各个分区之间按顺序排列，前一个分区的最大值即为后一个分区的最小值，第一
  - 个分区的最小值为该字段类型所允许的最小值，关键词MAXVALUE代表该字段类型所允许的最大值，一般最后一个分区会用MAXVALUE包住所有其他可能的值
- 语法

```
CREATE TABLE table_name (column_name column_type, ...)
PARTITIONED BY RANGE (partition_key_name, partition_key_type, ...)
(PARTITION [partition_name] VALUES LESS THAN (partition_value1),
PARTITION [partition_name] VALUES LESS THAN (partition_value2),
...
PARTITION [partition_name] VALUES LESS THAN (MAXVALUE))
```

表的分区：范围分区

- Inceptor中的范围分区
  - 所有range partition分区均需要手工指定，目前Inceptor只支持建表时一次性建好所有的range partition分区。
  - 每个分区可以给定个一个唯一的分区名partition_name，分区名的意义是会用该分区名作为hdfs上的分区目录名，没有分区名的分区会用（分区字段名 less_than 分区最大值）作为分区目录名。
  - 分区的范围为* [最小值, 最大值)* 前闭后开区间，即value less than的字面意义
  - 不支持 从文件导入范围分区表。
  - 支持INSERT INTO/OVERWRITE…SELECT…FROM…；形式向范围分区表中插入数据，插入时不需要像单值分区一样指定分区字段的值，形式上类似于动态分区插入。
  - 不支持 和单值分区混用来进行多层分区。
  - 为了渐少对某些客户SQL修改的工作量，我们的语法解析会识别很多Oracle中的关键字，如：

```
... partition A0200701 values less than ('200701')
tablespace USR_PMS_TBS  pctfree 0  initrans 1 maxtrans 255 storage
( initial 8M next 1M minextents 1
maxextents unlimited
) ...
```

表的分区：范围分区

TRANSWARP 星环科技

- 示例

```
[$host] transwarp> CREATE TABLE range_partition_user_info (
name STRING,
acc_num STRING,
password STRING,
citizen_id STRING,
bank_acc STRING,
acc_level string)
PARTITIONED BY RANGE (reg_date string) (
PARTITION VALUES LESS THAN (20090000),
PARTITION VALUES LESS THAN (20100000),
PARTITION VALUES LESS THAN (20110000),
PARTITION VALUES LESS THAN (20120000),
PARTITION VALUES LESS THAN (20130000),
PARTITION VALUES LESS THAN (20140000),
PARTITION VALUES LESS THAN (MAXVALUE));
[$host] transwarp> INSERT INTO TABLE range_partition_user_info partition (reg_date)
SELECT name, acc_num, password, citizen_id, bank_acc, acc_level, reg_date FROM
user_info;
[$host] transwarp> SELECT * FROM range_partition_user_info;
```

将user_info表用reg_date来范围分区，将用户按注册年份放入不同的分区里。比如在2008至2009年之间注册的用户将落入第一个分区。

多层分区

TRANSWARP 星环科技

- Inceptor支持多层分区字段，具体用法和普通分区相同。
- 示例

```
CREATE TABLE t1 (value INT)
partitioned by range(id1 INT, id2 INT, id3 INT)
(
partition values less than (5, 105, 205),
partition values less than (5, 105, 215),
partition values less than (5, 115, 205),
partition values less than (5, 115, 215),
partition values less than (5, 115, MAXVALUE),
partition p10_105_205 values less than (10, 105, 205),
partition p10_105_215 values less than (10, 105, 215),
partition p10_115_205 values less than (10, 115, 205),
partition p10_115_215 values less than (10, 115, 215),
partition pall_max values less than (MAXVALUE, MAXVALUE, MAXVALUE)
);
```

关于分区的建议

TRANSWARP 星环科技

- 分区的目的是减少扫描成本。所以单个分区的大小和总分区数目都应该控制在合理范围内。
- 使用多层分区带来的直接问题是总分区个数过多，因为总分区个数是所有分区键对应分区个数的乘积。所以我们建议尽量减少使用多层分区
- 对于时间、日期一类的值，使用单值分区会导致分区过多。推荐使用*范围分区(RANGE PARTITION)*。

表的分桶

- CLUSTERED BY子句
- 为表分桶可以让某些任务（比如取样，map join等）运行得更快。
- 语法

```
CREATE TABLE table_name (col_name data_type, col_name data_type, ...)
CLUSTERED BY (col_name, col_name,...)
//和分区键不同，分桶键必须是表中的列,和分区类似，Inceptor支持单层和多层分桶
[SORTED BY (col_name, col_name,...)[ASC|DESC]]
//根据指定的键排序。ASC表示升序，DESC表示降序。默认顺序是升序。
INTO n BUCKETS//指定桶的数量
```

- 示例：单层分桶与多层分桶

```
[$host]transwarp> CREATE TABLE bucket_user_info(
> name STRING,
> acc_num STRING,
> password STRING,
> citizen_id STRING,
> bank_acc STRING,
> reg_date STRING,
> acc_level STRING)
> CLUSTERED BY (acc_num)
> INTO 3 BUCKETS;
```

```
[$host]transwarp> CREATE TABLE bucket_user_info(
> name STRING,
> acc_num STRING,
> password STRING,
> citizen_id STRING,
> bank_acc STRING,
> reg_date STRING,
> acc_level STRING)
> CLUSTERED BY (acc_num, citizen_id)
> INTO 6 BUCKETS;
```

分桶表的数据写入

TRANSWARP 星环科技

- 向分桶表写入数据只能通过INSERT，而不能LOAD。
- 为表分桶不会影响Inceptor如何向表写入数据，只会改变Inceptor从表读出数据的方式。所以在向分桶表中写入数据时，Inceptor不会自动按照建表时指定的分桶方式将数据写入不同桶中。
- 相较未分桶表，用户在分桶表中可以进行更高效的取样和一些其他的省时操作，比如map或join。但是，Inceptor要等到对表进行写入时才会执行创建表时规定的分桶方式，所以表的元数据显示的表实行可能和表真实的形式不一样。我们应该避免这样的情况。

向分桶表写入数据

TRANSWARP 星环科技

- 建表：

```
[$host]transwarp> CREATE TABLE user_info_bucketed (
> name STRING,
> acc_num STRING,
> password STRING,
> citizen_id STRING,
> bank_acc STRING,
> reg_date STRING)
> PARTITIONED BY (acc_level STRING)
> CLUSTERED BY (acc_num) INTO 25 BUCKETS;
```

- 根据user_id分桶，写入数据

```
[$host]transwarp> set hive.enforce.bucketing = true;  //让Inceptor自动根据表的分桶填入数据，写入时，Inceptor会尽量均匀地将数据哈希进各个桶中。
[$host]transwarp> FROM user_id
> INSERT OVERWRITE TABLE user_info_bucketed
> PARTITION (acc_level='A')
> SELECT name, acc_num, password, citizen_id, bank_acc, reg_date
>WHERE acc_level = 'A';
```

修改表

TRANSWARP 星环科技

- ALTER TABLE
  - 可以对表的大多数属性进行修改。
  - 大多数时候，ALTER TABLE修改的是表的元数据而不改变表中的数据。
- 重命名：ALTER TABLE...RENAME TO

```
ALTER TABLE table_name RENAME TO new_table_name;
```

- 改变表属性：ALTER TABLE...SET TBLPROPERTIES
  - 用这个语句给表添加自定义的元数据
  - 语法：

```
ALTER TABLE table_name SET TBLPROPERTIES table_properties;
table_properties:
: (property_name = property_value, property_name = property_value, ... )
```

  - 示例：
    - 为表table_1改变（或者添加）属性zhuren的值为 yangyao

```
alter table table_1 set tblproperties ("zhuren"="yangyao");
```

修改表

TRANSWARP 星环科技

- 改动表的注解

```
ALTER TABLE table_name SET TBLPROPERTIES ('comment' = new_comment);
```

- 改变SerDe属性

```
ALTER TABLE table_name SET SERDE serde_class_name [WITH SERDEPROPERTIES serde_properties];
```

- 给表的SerDe添加用户自定义的元数据

```
ALTER TABLE table_name SET SERDEPROPERTIES serde_properties;
serde_properties:
: (property_name = property_value, property_name = property_value, ... )
```

- 改变表的分桶

```
ALTER TABLE table_name CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name, ...)]
INTO num_buckets BUCKETS;
```

修改分区

- 语法

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec
[LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...;
partition_spec:
: (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

  - [LOCATION 'location']选项允许我们将已存在的目录添加为表的分区。注意，该目录必须是HDFS目录，不可以是文件。ALATER TABLE ... ADD PARTITION时，Inceptor会将该目录下所有的文件中的数据都写入表中。

- 示例
  - HDFS目录/user/root/test2下有一个放有所有acc_level = A的文件

```
[$host] transwarp> ALTER TABLE partition_user_info2 ADD PARTITION (acc_level = 'A')
LOCATION '/user/root/test2';
[$host] transwarp> SELECT * FROM partition_user_info2;
马从筠 6513065 115591 1404001987110123 07 96080357291141 20110101 A
魏向卉 3912384 841242 5226321993010294 04 68537153578048 20091202 A
邱坤 0700735 737297 3408111992112522 78 14388242322818 20121024 A
```

重命名分区

TRANSWARP 星环科技

- 语法

```
ALTER TABLE table_name PARTITION partition_spec1 RENAME TO PARTITION partition_spec2;
partition_spec:
: (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

- 示例
  - 将表partition_user_info2的partition由A改为B

```
ALTER TABLE partition_user_info2 PARTITION (acc_level='A') RENAME TO PARTITION
(acc_level='B');
SELECT * FROM partition_user_info2;
马从筠 6513065 115591 1404001987110123 07 960803572911 41 20110101 B
魏向卉 3912384 841242 5226321993010294 04 685371535780 48 20091202 B
邱坤 0700735 737297 3408111992112522 78 143882423228 18 20121024 B
```

交换分区

TRANSWARP 星环科技

- 语法

```
ALTER TABLE table_name_1 EXCHANGE PARTITION (partition_spec) WITH TABLE table_name_2;
partition_spec:
: (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

删除分区

TRANSWARP 星环科技

- 语法

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION
partition_spec,...;
partition_spec:
: (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

- 举例
  - 将分区的用户信息表中的acc_level = 'A’ 的分区删除：

```
[$host] transwarp> ALTER TABLE partition_user_info DROP PARTITION (acc_level = 'A');
[$host] transwarp> SELECT * FROM partition_user_info;
华微 5224133 531547 420529198911075631 326382810959O7 20080214 B
祝韩恒 6670192 205239 230801197908126178 737903699909 71 20100101 C
潘营泽 6600641 990590 511521198705077435 484711355936O8 20110430 C
管淑艳 2394923 783438 330683198005210864 999138634451 74 20141003 C
宁新瑶 4580952 986634 420822199001119507 977110088565 76 20081031 D
李韩瑶 2755506 015859 310230197912126559 424123962422 37 20110916 D
李平 8725869 600709 460100198902070313 430813070469 84 20130702 E
```

修改表/分区的位置

TRANSWARP 星环科技

- 语法

```
ALTER TABLE table_name [PARTITION partition_spec] SET LOCATION "new location";
```

修改表/分区的保护

TRANSWARP 星环科技

- 用户可以给表和分区加上保护。
- ENABLE NO_DROP指令可以给表/分区加上保护，阻止表/分区被删除。DISABLE NO_DROP可以将这层保护除去。
- ENABLE OFFLINE指令可以为表/分区加上保护，阻止对表/分区的查询，但是被阻止查询的表/分区的元数据还是可见的。DISABLE OFFLINE可以将这层保护除去。
- 如果一张表中有分区被加上保护阻止删除，那么这张表便也不能被删除。
- 但是，如果一张表被加上保护阻止删除，表中的分区还是可以被删除的，除非加上CASCADE关键词，也就是说如果对一张表ENABLE NO_DROPP CASCADE，那么表和表中的分区都不能被删除。
- 语法

```
ALTER TABLE table_name [PARTITION partition_spec] ENABLE|DISABLE NO_DROP [CASCADE]
ALTER TABLE table_name [PARTITION partition_spec] ENABLE|DISABLE OFFLINE
```

修改列

TRANSWARP 星环科技

- 修改列名、列的数值类型、列在表中的位置和列的注解
- 语法

```
ALTER TABLE table_name CHANGE [COLUMN] col_old_name col_new_name column_type
[COMMENT col_comment] [FIRST|AFTER column_name] [CASCADE|RESTRICT];
```

  - 对一列的列名、列的数值类型、列在表中的位置、列的注解或者这些的任意组合做出修改。这个指令仅仅修改表的元数据。
- 示例：

```
CREATE TABLE test_change (a INT, b INT, c INT);
//将列a重命名为列a1
ALTER TABLE test_change CHANGE a a1 INT;
//将列a1重命名为列a2，它的数据类型改为STRING，并将它放在列b之后
ALTER TABLE test_change CHANGE a1 a2 STRING AFTER b;
//表test_change的新结构为：(b INT, a2 STRING, c INT)
//将列c重命名为c1, 并将它定位第一列
ALTER TABLE test_change CHANGE c c1 INT FIRST;
//表test_change的新结构为：(c1 INT, b INT, a2 STRING)
```

添加/替换列

- ADD COLUMNS可以将新的列加入表中，位置在所有列之后，分区之前。
- REPLACE COLUMNS将所有已有的列删除然后加入新指定的列。只能对使用Inceptor自带SerDe (DynamicSerDe,MetadataTypedColumnsetSerDe, LazySimpleSerDe and ColumnarSerDe) 的表使用REPLACECOLUMNS。REPLACE COLUMNS也可以被用于删除列。
- 语法

```
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name data_type [COMMENT
col_comment], ...) [CASCADE|RESTRICT]
```

- 示例

```
CREATE TABLE test_change (a INT, b INT, c INT);
ALTER TABLE test_change REPLACE COLUMNS (a int, b int);
//将列c删除。
```

TRUNCATE TABLE语句

TRANSWARP 星环科技

- TRUNCATE TABLE删除表或者分区中的数据，但不删除表或分区的元数据。这个操作只能用于托管表。
- 可以将一张指定表下的数据全部删除(对分区表和非分区表都适用）
- 语法
  - 将一张指定表下的数据全部删除

```
TRUNCATE TABLE table_name
```

  - 将一张指定表下的指定分区中的数据全部删除

```
TRUNCATE TABLE table_name [PARTITION partition_spec]
partition_spec:
: (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

TRUNCATE TABLE语句

TRANSWARP 星环科技

- 示例
  - 删除分区表partition_user_info中acc_level为A的记录：

```
[$host] transwarp> SELECT * FROM partition_user_info;
[$host] transwarp> TRUNCATE TABLE partition_user_info PARTITION (acc_level='A');
[$host] transwarp> SELECT * FROM partition_user_info;
```

  - 删除所有partition_user_info中的数据(partition_user_info将成为空表）：

```
[$host]transwarp> TRUNCATE TABLE partition_user_info;
```

DROP TABLE语句

TRANSWARP 星环科技

- DROP TABLE删除一个指定名称的表。
- 当被删除的表是托管表时，表的元数据和表中数据都会被删除。如果被删除的表是外部表，则只有它的元数据会被删除。
- 语法

```
[$host]transwarp> DROP TABLE table_name;
```

创建视图

TRANSWARP 星环科技

- CREATE VIEW创建一个视图
- 语法

```
CREATE VIEW view_name [(column_name，column_name, ...)] AS SELECT select_statement;
```

  - 在创建视图时，指定列名和列值类型不是必须的。
  - CREATE VIEW的语法和CTAS非常相像。区别在于VIEW是非实体化的，CREATE VIEW给查询创建一个快捷方式，而CTAS将查询结果写入磁盘中。
- 示例
  - 下面代码用CREATE VIEW创建了一个视图来显示user_info表中的name, reg_date和acc_level三列：

```
[$host]transwarp> CREATE VIEW non_secure_info
AS SELECT name, reg_date, acc_level
FROM user_info;
```

删除视图

TRANSWARP 星环科技

- DROP VIEW
  - 删除一个指定的视图
  - DROP VIEW将指定视图的元数据删除。虽然视图和表有很多共同之处，但是DROP TABLE不能用来删除VIEW。
  - 如果删除的视图被其他视图引用，Inceptor不会给出任何警告。依靠于被删除视图的视图会变成无效视图，但是用户需要自己处理这些变成无效的视图。
- 语法

```
DROP VIEW [IF EXISTS] view_name;
```

修改视图属性

TRANSWARP 星环科技

- 用户自定义的键值对属性也用TBLPROPERTIES来定义：
- 语法

```
ALTER VIEW view_name SET TBLPROPERTIES table_properties;
table_properties:
: (property_name = property_value, property_name = property_value, ...)
```

修改视图定义

TRANSWARP 星环科技

- ALTER VIEW AS SELECT改变指定视图的定义。
- 在改变一个视图的定义前，该视图的定义必须已经被创建过。如果一个视图有分区，那么它不能通过ALTER VIEW AS SELECT替换。
- 语法

```
ALTER VIEW view_name AS select_statement;
```

视图的分区

- 表的分区使用PARTITIONED BY子句来指定分区键，而视图的分区使用PARTITIONED ON子句来指定分区键。这是为了区分这两种不同的分区方式。
- 对于表分区，需要在表定义 之外 定义分区键，表的分区键不能和列重名；而视图分区，是使用视图定义中 已存在 的列作为分区键，所以不需要指定分区键的数值类型。
- 和表的分区相似，PARTITIONED ON子句指定的分区键必须在视图定义（也就是SELECT语句的查询结果）的末尾，而且它们在PARTITIONED ON子句中出现的顺序必须和在视图定义中出现的顺序相同。

视图的分区

TRANSWARP 星环科技

- 语法

```
CREATE VIEW view_name [(column_name [COMMENT column_comment], ...) ]
PARTITIONED ON (col1, col2, ...)
AS SELECT ...
```

- 举例

```
[$host] transwarp> CREATE VIEW partition_user_info_view PARTITIONED ON (acc_level) AS
SELECT * FROM user_info;
[$host] transwarp> DESCRIBE partition_user_info_view;
name string None
acc_num string None
password string None
citizen_id string None
bank_acc string None
reg_date string None
acc_level string None
# Partition Information
# col_name data_type comment
acc_level string None
```

临时函数

- 创建临时函数
  - 语法

```
CREATE TEMPORARY FUNCTION function_name AS class_name
```

    - 创建一个由class_name实施的临时函数。这个新创建的函数只能在当前session使用。用户可以使用任意一个在class path中的class。用户也可以通过ADD JAR向class path加jar包。
- 删除临时函数
  - 语法

```
DROP TEMPORARY FUNCTION [IF EXISTS] function_name
```

永久函数

TRANSWARP 星环科技

- 创建永久函数
  - 语法

```
CREATE FUNCTION [db_name.]function_name AS class_name
[USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ]
```

    - 创建一个由class_name实施的函数。这个函数将在metastore登记，不需要在每个session重新建临时函数。需要加入环境的jar包，文件或者档案可以通过USING子句指定。
    - 第一次使用该函数时，这些资源会像被ADD JAR/FILE一样加到环境中。如果Inceptor不再本地模式，那么资源的地址也必须是非本地URI，比如HDFS地址。
    - 该函数会被加进当前使用的数据库，或者是由db_name指定的数据可。该函数可以通过db_name.function_name调用，如果函数属于当前数据库，那么拽可以直接通过function_name调用
- 删除永久函数
  - 语法

```
DROP FUNCTION [IF EXISTS] function_name
```

SHOW

TRANSWARP 星环科技

- 用SHOW可以查询metastore中的信息。
- 通常，我们用SHOW来查看数据库，表和视图，列等图表对象。

SHOW DATABASES

TRANSWARP 星环科技

- SHOW DATABASES
  - 该语句可以查看所有数据库。
  - 使用 [LIKE identifier_with_wildcards] 选项可以通过正则表达式过滤查看结果。正则表达式中的外卡只有*（表示任意个字母）和|（表示选择）。
    - 比如，'employees', 'emp*', 'emp*|*ees' 都可以和’ employees' 匹配。
- 语法

```
SHOW DATABASES [LIKE identifier_with_wildcards]
```

- 示例

```
[$host] transwarp> SHOW DATABASES LIKE 'test';
test
[$host] transwarp> SHOW DATABASES LIKE 'test*';
test
test_db
[$host] transwarp> SHOW DATABASES LIKE 'test*|*db';
my_db
test
test_db
```

SHOW TABLES

TRANSWARP 星环科技

- SHOW TABLES
  - SHOW TABLES会显示当前使用的数据库中所有的表和视图。
  - 如果想要查看非当前数据库中的表和视图，可以使用 [IN database_name] 来指定数据库。
  - 使用 [LIKE identifier_with_wildcards] 选项可以通过正则表达式过滤查看结果。
- 语法

```
SHOW TABLES [IN database_name] [LIKE identifier_with_wildcards];
```

SHOW PARTITIONS

TRANSWARP 星环科技

- SHOW PARTITIONS
  - 显示指定表中的所有分区。
  - 使用 [PARTITION (partition_key = partition_value, partition_key = partition_value, ...)] 选项可以过滤查看的结果。
- 语法

```
SHOW PARTITIONS table_name [PARTITION (partition_key = partition_value, partition_key = partition_value, ...)]
```

- 示例

```
[$host] transwarp> SHOW PARTITIONS doub_partition_user_info;
[$host] transwarp> SHOW PARTITIONS doub_partition_user_info PARTITION (reg_date ='20080214');
[$host] transwarp> SHOW PARTITIONS doub_partition_user_info PARTITION (acc_level ='A');
[$host] transwarp> SHOW PARTITIONS doub_partition_user_info PARTITION (reg_date ='20091202',acc_level = 'A');
```

SHOW TABLE EXTENDED

TRANSWARP 星环科技

- SHOW TABLE EXTENDED
  - 列出所有和给出的正则表达式(注意：此处的正则表达式两端不要加引号)匹配的表信息。如果指定了分区，则不能使用正则表达式。
  - 该指令的输出包括基础的表信息和文件系统信息，包括totalNumberFiles, totalFileSize, maxFileSize,minFileSize,lastAccessTime, 和lastUpdateTime。
  - 如果指定了分区，该指令则不会输出表的文件系统信息，而是输出指定的分区的文件系统信息
- 语法

```
SHOW TABLE EXTENDED [IN|FROM database_name] LIKE identifier_with_wildcards
[PARTITION(partition_spec)]
```

SHOW TBLPROPERTIES

TRANSWARP 星环科技

- SHOW TBLPROPERTIES
  - 可以查看表的所有表属性。
  - 如果只要查看某个属性，可以在表名后面的括号中指明。
- 语法

```
SHOW TBLPROPERTIES table_name;
SHOW TBLPROPERTIES table_name("property_name");
```

- 举例

```
[$host] transwarp> SHOW TBLPROPERTIES user_info;
numPartitions 0
numFiles 1
last_modified_by yarn
last_modified_time 1417711896
transient_lastDdlTime 1417864612
totalSize 707
numRows 0
rawDataSize 0
[$host] transwarp> SHOW TBLPROPERTIES user_info('numFiles');
1
```

SHOW CREATE TABLE

TRANSWARP 星环科技

- SHOW CREATE TABLE
  - 查看指定表和视图的建表语句。
- 语法

```
SHOW CREATE TABLE ([db_name.]table_name|view_name)
```

- 举例

```
[$host] transwarp> SHOW CREATE TABLE user_info;
CREATE TABLE user_info(
name string,
acc_num string,
password string,
citizen_id string,
bank_acc string,
reg_date string,
.....
```

SHOW COLUMNS

TRANSWARP 星环科技

- SHOW COLUMNS
  - 查看所有指定表中的列。FROM和IN没有区别，可替换使用。
- 语法

```
SHOW COLUMNS (FROM|IN) table_name [(FROM|IN) db_name]
```

- 举例

```
[$host] transwarp> SHOW COLUMNS IN user_info;

name
acc_num
password
citizen_id
bank_acc
reg_date
acc_level
```

SHOW FUNCTIONS

TRANSWARP 星环科技

- SHOW FUNCTIONS
  - 显示所有匹配指定正则表达式的函数。
- 语法

```
SHOW FUNCTIONS indentifier_with_wildcards;
```

- 示例

```
[$host] transwarp> SHOW FUNCTIONS 'x.*';
xpath
xpath_boolean
xpath_double
xpath_float
xpath_int
xpath_long
xpath_number
xpath_short
xpath_string
```

  - 如果想要查看所有函数，使用SHOW FUNCTIONS '.*'。

DESCRIBE和DESCRIBE EXTENDED

TRANSWARP 星环科技

- DESCRIBE和DESCRIBE EXTENDED用于查看图表对象的schema。
- DESCRIBE查看基础的schema
- DESCRIBE EXTENDED也可查看图表对象属性。

DESCRIBE DATABASE

TRANSWARP 星环科技

- DESCRIBE DATABASE查看数据库名称，注解和根目录。
- DESCRIBE DATABASE EXTENDED 同时也查看数据库属性。
- 语法

```
DESCRIBE DATABASE [EXTENDED] db_name;
```

DESCRIBE TABLE

TRANSWARP 星环科技

- DESCRIBE 显示包括分区键的所有列的列名和列数据类型。
- DESCRIBE EXTENDED显示所有表的元数据，通常只有会在debug时用到。
- DESCRIBE FORMATTED会用表格形式显示所有表的元数据。
- 语法

```
DESCRIBE [EXTENDED|FORMATTED] [db_name.]table_name;
```

DESCRIBE COLUMN

TRANSWARP 星环科技

- 该语句查看指定表中指定列的信息。
- 语法

```
DESCRIBE FORMATTED [db_name.]table_name.column_name;
```

DESCRIBE PARTITION

TRANSWARP 星环科技

- 该语句显示指定表中指定分区的元数据
- 语法

```
DESCRIBE [EXTENDED|FORMATTED] [db_name.]table_name PARTITION partition_spec;
```

DESCRIBE FUNCTION

TRANSWARP 星环科技

- 该语句描述函数的用法和给出示例。
- 语法

```
DESCRIBE FUNCTION [EXTENDED] function_name;
```