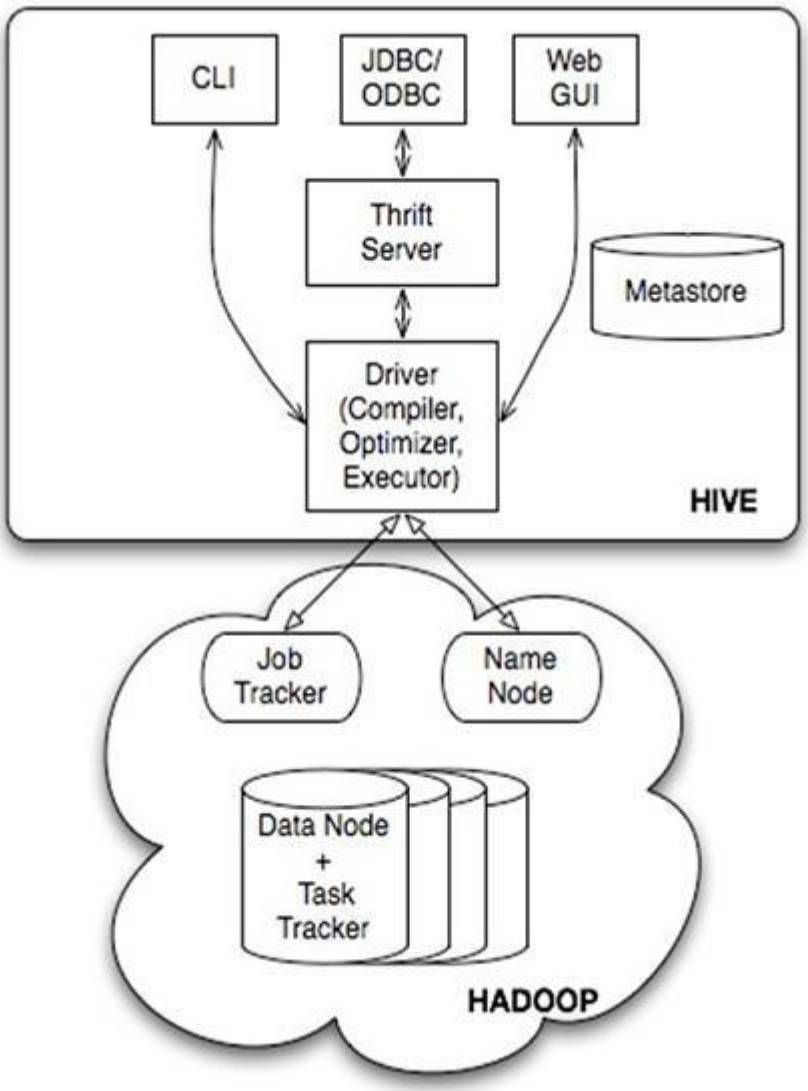
Hive 操作笔记




# Hive 安装（1.2.1）

1、解压一个 hive 安装包到集群的任意一台机器上
2、配置 hive 的目录到环境变量中
3、将 hive 的 lib 中的 jline.2.12.jar 替换掉 hadoop2.6.4/share/hadoop/yarn/lib/jline.0.94.jar
4、修改配置文件
在 hive 的 conf 目录中
vi hive-site.xml

```
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
```

```
<description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
<description>Driver class name for a JDBC metastore</description>
</property>

<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
<description>username to use against metastore database</description>
</property>

<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>root</value>
<description>password to use against metastore database</description>
</property>
</configuration>
```

数据库前提

1、远程连接 mysql 权限被拒绝时，先在 mysql 服务器上用客户端连上，然后敲如下命令：

grant all privileges on *.* to 'root'@'%' identified by 'root 的密码' with grant option;

flush privileges;

2、数据库的排序规则必须为 latin1

3、在 hive 的 lib 中放置一个 mysql 的 jdbc 驱动 jar 包

5、启动 hive

*注意：先保证你的 hdfs 和 yarn 正常运行，hadoop 已配置在环境变量中*

*否则：Cannot find hadoop installation: $HADOOP_HOME or $HADOOP_PREFIX must be set or hadoop must be in the path*

启动命令： bin/hive

测试：

show databases;

show tables;

新版本(0.12.0 以上)中需要增加以下配置

<property>

<name>hive.querylog.location</name>

<value>/home/edu360/deployed-soft/hive-0.14.0/tmp</value>

</property>

```
<property>
<name>hive.exec.local.scratchdir</name>
<value>/home/edu360/deployed-soft/hive-0.14.0/tmp</value>
</property>
<property>
<name>hive.downloaded.resources.dir</name>
<value>/home/edu360/deployed-soft/hive-0.14.0/tmp</value>
</property>
```

# Linux 下 Mysql 数据库

## 2.1 安装

1 删除原本依赖
rpm -e --nodeps `rpm -qa |grep MySQL`
2 然后 yum 在线安装
  yum install -y mysql-server
3 启动 mysql 服务
sudo service mysqld start
4 初始化配置
mysql_secure_installation
5 加入到开机启动项
chkconfig mysqld on
6 权限授予
grant all privileges on *.* to 'root'@'%' identified by 'root';
flush privileges;

## 2.2hive 乱码问题

修改 hive mysql 元数据库的以下编码即可：

-- 修改表字段和表 COMMENT
ALTER TABLE COLUMNS_V2 MODIFY COLUMN COMMENT VARCHAR(256) CHARACTER SET utf8;
ALTER TABLE TABLE_PARAMS MODIFY COLUMN PARAM_VALUE VARCHAR(4000) CHARACTER SET utf8;
-- 修改分区字段 COMMENT
ALTER TABLE PARTITION_PARAMS MODIFY COLUMN PARAM_VALUE VARCHAR(4000) CHARACTER SET utf8;
ALTER TABLE PARTITION_KEYS MODIFY COLUMN PKEY_COMMENT VARCHAR(4000) CHARACTER SET utf8;

-- 修改索引 COMMENT
ALTER TABLE INDEX_PARAMS MODIFY COLUMN PARAM_VALUE VARCHAR(4000) CHARACTER SET utf8;

## 2.7linux 下 mysql 编码修改

rpm 离线包安装：
cp /usr/share/mysql/my-small.cnf /etc/my.cnf
vim /etc/my.cnf
在[client]下面增加
default-character-set=utf8
在[mysqld]下面增加
default-character-set=utf8
init_connect='SET NAMES utf8'

yum 在线方式：
vim /etc/my.cnf
在[mysqld]下面增加
init_connect='SET collation_connection = utf8_unicode_ci'
init_connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_unicode_ci
skip-character-set-client-handshake

重启 mysql 服务
service mysql restart

查看编码
show variables like 'char%';
临时设置编码
SET NAMES utf8;

## 2.8Hive 在 mysql 的体现

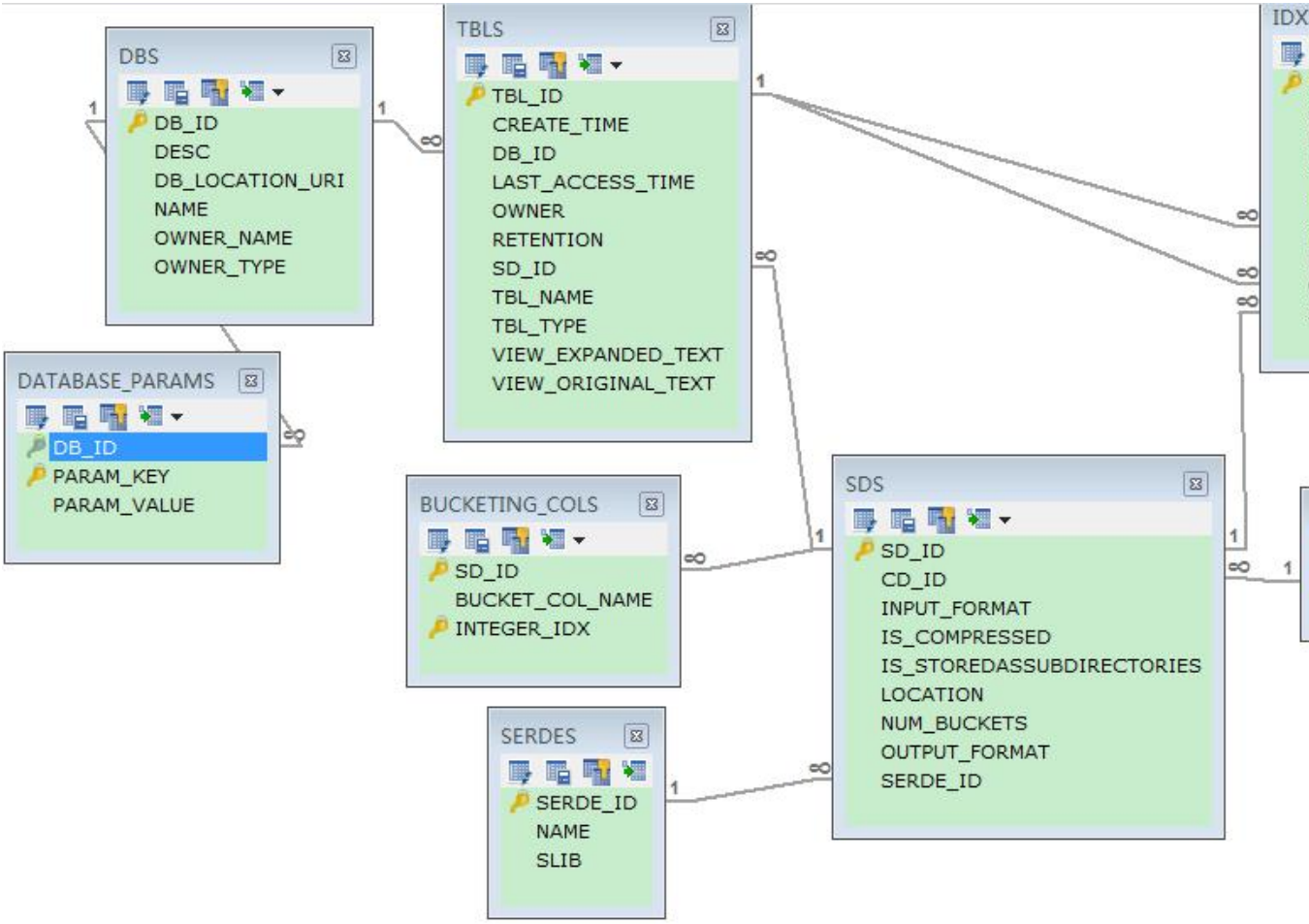



表 TBLS 维护表与数据库的关系


表 DBS 描述了有哪些数据库

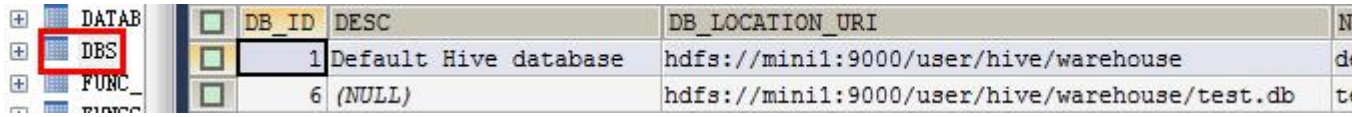
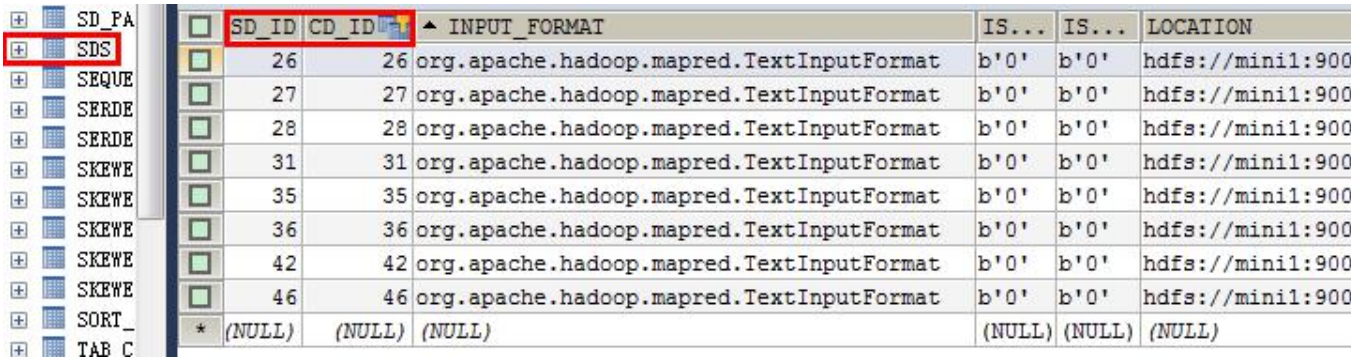



表 SDS 描述表在 hdfs 的位置，并 维护表与字段的关联关系



| SD_ID | CD_ID | INPUT_FORMAT | IS... | IS... | LOCATION |
|---|---|---|---|---|---|
| 26 | 26 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 27 | 27 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 28 | 28 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 31 | 31 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 35 | 35 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 36 | 36 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 42 | 42 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| 46 | 46 | org.apache.hadoop.mapred.TextInputFormat | b'0' | b'0' | hdfs://mini1:900 |
| (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) |

表 COLUMNS_V2 描述表与字段的关系

| CD_ID | COMMENT | COLUMN_NAME | TYPE_NAME | INTEGER_IDX |
|---|---|---|---|---|
| 26 | ??ID | id | int | 0 |
| 26 | ???? | name | string | 1 |
| 27 | ???? | age | int | 3 |
| 27 | ?? | deptid | int | 1 |
| 27 | ?? | hometown | string | 6 |
| 27 | ??ID | id | int | 0 |
| 27 | ???? | name | string | 2 |
| 27 | ???? | phone | bigint | 5 |
| 27 | ????,1????0??? | sex | int | 4 |

# hive 的运行及访问方式

## 3.1 执行 Hive 脚本

linux 终端下
hive -f hive.hql
dt=1;hive -S -hiveconf hive.exec.mode.local.auto=true -e "use default;select * from t1 where id%2==$dt"

hive 终端下
source hive.hql

Hive 命令参数
-S 开启静默模式，输出结果去除 OK 等行
-i 指定一个文件，cli 启动时会先运行这个文件，
（hive 会自动运行 currentUser.home 目录的下的.hiverc 文件）

## 3.2Thrift 服务端 JDBC 的方式

hive 启动 hive Thrift 服务端 (默认端口 10000,可通过 hive.server2.thrift.port 参数调整)
hive --service hiveserver2
启动时指定端口
hive --service hiveserver2 --hiveconf hive.server2.thrift.port=10002
后台启动
nohup ./hive --service hiveserver2 --hiveconf hive.server2.thrift.port=10002 &

org.apache.hive.jdbc.HiveDriver
在 java 代码中调用 hive 的 JDBC 建立连接
url：
jdbc:hive2://mini3:10002/test

beeline 里连接
!connect jdbc:hive2://mini3:10002

## 3.3 WebGUI 的方式

这里简单的说一下，WebGUI 的搭建和访问过程
1 、解压 hive 源码包 并进入 hwi 子目录
tar -zxvf apache-hive-0.14.0-src.tar.gz -C ...(具体目录)
3 、制作 war 包
jar cvfM hive-hwi-1.2.1.war -C web/ .
4 、 拷 贝 hive-hwi-0.14.0.war 至 $HIVE_HOME/lib 目 录
cp hive-hwi-1.2.1.war $HIVE_HOME/lib
cp $JAVA_HOME/lib/tools.jar $HIVE_HOME/lib
5 、修改 hive-site.xml 配置文件

```
<property>
<name>hive.hwi.listen.host</name>
<value>0.0.0.0</value>
</property>
<property>
<name>hive.hwi.listen.port</name>
<value>9999</value>
</property>
<property>
<name>hive.hwi.war.file</name>
<value>lib/hive-hwi-1.2.1.war</value>
```

</property>
5 、启动 hive，及其访问
nohup bin/hive --service hwi &
访问地址：
mini3:9999/hwi
执行查询
进入会话管理页面：
在 Result File 中填入结果保存文件；注意：这个文件必须存在。
在 Query 中填入要执行的 HQL 语句；
Start Query 选择 YES；
点击 Submit 开始执行 HQL 语句。

# Hive 参数获取

## 4.1 启动 hive cli 时，增加参数

hive -hiveconf dt=2016-05-05
获取参数
Set dt;
select * from t1 where id%2==${hiveconf:dt}

## 4.2hive cli or Beeline command

重置所有配置，（不适用 hiveconf: 为前缀的变量）
reset
打印所有用户变量
set
打印所有 hivehe hadoop 的配置
set -v

ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]*
LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]
DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]
例：
add file /root/t1;
list files;

## 4.3 常见参数

显示列名称
set hive.cli.print.header=true;
显示数据库名称
set hive.cli.print.current.db=true;
本地模式
set hive.exec.mode.local.auto=true;

开启分桶
set hive.enforce.bucketing=true;

开启动态分区支持
set hive.exec.dynamic.partition=true;
动态分区的模式，默认 strict，表示必须指定至少一个分区为静态分区，nonstrict 模式表示允许所有的分区字段都可以使用动态分区。
set hive.exec.dynamic.partition.mode=nonstrict;
允许所有的分区列都是动态分区列
set hive.exec.dynamic.partition.mode=nostrict;
在每个执行 MR 的节点上，最大可以创建多少个动态分区，如果超过了这个数量就会报错
hive.exec.max.dynamic.partitions.pernode （缺省值 100）：
在所有执行 MR 的节点上，最大一共可以创建多少个动态分区
hive.exec.max.dynamic.partitions （缺省值 1000）：
整个 MR Job 中，最大可以创建多少个 HDFS 文件
hive.exec.max.created.files （缺省值 100000）：

对分区表查询必须加分区过滤,不允许笛卡尔积查询，order by 后面必须有 limit 限制
set hive.mapred.mode=strict;
不严格模式
set hive.mapred.mode=nostrict;

开启任务并行执行
set hive.exec.parallel=true;

开启 map join
set hive.auto.convert.join=true;
set hive.mapjoin.smalltable.filesize=25000000;

忽略 MAPJOIN 标记
set hive.ignore.mapjoin.hint=true;

## 4.4Hive cli 更多功能

hive 会自动运行$HOME/.hiverc 文件
例如当前用户是 root
可以在/root/目录下，创建一个.hiverc 文件，内容如下：

```
set hive.cli.print.header=true;
set hive.cli.print.current.db=true;
set hive.exec.mode.local.auto=true;
```

Hive 会将最近 100 行命令记录到$HOME/.hivehistory

在命令前加!，可执行常见的 shell 命令
例：

```
!echo "dog";
!pwd;
!cat /tmp/payroll/00000_0
```

可直接查看 hdfs 文件

```
dfs ls /;
```

## hive 开启分布式锁

在 hive-site.xml 中配置

```
<property>
   <name>hive.support.concurrency</name>
   <value>true</value>
</property>
<property>
   <name>hive.zookeeper.quorum</name>
   <value>mini1:2181,mini2:2181,mini3:2181</value>
</property>
```

# Hive 语法

## 5.0 database

DATABASE 和 SCHEMA 是一样的意思

创建数据库

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

例：

```
create database financials
comment 'test'
location '/hdfs/financials'
with dbproperties('date'='2017-03-31');
```

查看数据库描述

```
desc database financials;
desc database extended financials;
```

查看数据库列表

```
show databases;
show databases like 't.*';
```

删除数据库

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

例：

```
drop database financials;
drop database if exists financials;
```

先删除数据库中的表再删数据库

```
drop database test cascade;
drop database if exists financials cascade;
```

修改数据库属性

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES
(property_name=property_value, ...);   -- (Note: SCHEMA added in Hive 0.14.0)
```

例：

```
alter database financials set dbproperties('date'='today');
```

```
ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;
```

查看当前数据库所有的表

```
show tables;
```

查看指定数据库所有的表

```
show tables in test;
```

查看表结构

```
desc t1.id;
desc t1;
desc extended t1;
desc formatted    t1;
```

查看表分区

```
show partitions tblName;
```

查看表的创建信息

```
show create table test;
```

```
CREATE TABLE `test`(
   `line` string)
ROW FORMAT SERDE
   'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   'hdfs://mini1:9000/hdfs/clean'
TBLPROPERTIES (
   'COLUMN_STATS_ACCURATE'='false',
   'column_stats_accurate'='true',
   'numFiles'='0',
   'numRows'='-1',
   'numfiles'='1',
   'numrows'='0',
   'rawDataSize'='-1',
   'rawdatasize'='0',
   'totalSize'='0',
   'totalsize'='246',
   'transient_lastDdlTime'='1490851050',
   'transient_lastddltime'='1490798365')
```

# 5.1DDL

重命名表名
alter table tblname rename to new_tblname
例：
alter table t1 rename to t2;

给表增加一个字段：
alter table tblName add columns (columnName type comment 'info');
例：
alter table t1 add columns (col1 string);

修改某一个字段：
alter table table_name change
[cloumn] col_old_name col_new_name column_type
[conmment col_conmment]
[first|after column_name];
eg.
alter table tblname change colname new_colname new_type;
例：
alter table t1 change name stu_name string;
将 age 放到第一列
alter table t1 change age age int first;
将 age 放在某一列的后面
alter table t1 change age age int after id;

替换所有的字段类型为指定的类型列表：
alter table tblname replace columns (col_spec[, col_spec ...])
例：
```
alter table t1 replace columns (
   t_id int comment 't_id'
);
```

修改表位置,(只能是 hdfs 上的全路径)
alter table t6_external set location 'hdfs://mini1:9000/data/ttt';

## 5.2 创建表

创建表

create table if not exists t1(id int);

根据其他表模式创建表

create table if not exists t1 like test.t_dept;

查看当前数据库所有 test 开头的表

show tables 'test.*';

注：in db 和表名正则匹配不能同时使用

---

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
-- (Note: TEMPORARY 在 0.14.0 以后版本可用)
  [(col_name data_type [COMMENT col_comment], ...
[constraint_specification])]
  [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, ...)]                  -- (Note: 在 0.10.0
以后版本可用)
     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 (...)]
-- (Note: 在 0.6.0 以后版本可用)
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]   -- (Note:
Available in Hive 0.6.0 and later)
  [AS select_statement];   -- (Note: Available in Hive 0.5.0 and later; not
supported for external tables)

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  LIKE existing_table_or_view_name
  [LOCATION hdfs_path];

data_type
  : primitive_type
  | array_type
  | map_type
  | struct_type
```

```
  | union_type  -- (Note: Available in Hive 0.7.0 and later)

primitive_type
  : TINYINT
  | SMALLINT
  | INT
  | BIGINT
  | BOOLEAN
  | FLOAT
  | DOUBLE
  | DOUBLE PRECISION -- (Note: Available in Hive 2.2.0 and later)
  | STRING
  | BINARY      -- (Note: Available in Hive 0.8.0 and later)
  | TIMESTAMP   -- (Note: Available in Hive 0.8.0 and later)
  | DECIMAL     -- (Note: Available in Hive 0.11.0 and later)
  | DECIMAL(precision, scale)  -- (Note: Available in Hive 0.13.0 and later)
  | DATE        -- (Note: Available in Hive 0.12.0 and later)
  | VARCHAR     -- (Note: Available in Hive 0.12.0 and later)
  | CHAR        -- (Note: Available in Hive 0.13.0 and later)
array_type
  : ARRAY < data_type >
map_type
  : MAP < primitive_type, data_type >
struct_type
  : STRUCT < col_name : data_type [COMMENT col_comment], ...>
union_type
   : UNIONTYPE < data_type, data_type, ... >  -- (Note: Available in Hive
0.7.0 and later)

row_format
  : DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS
TERMINATED BY char]
      [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
      [NULL DEFINED AS char]   -- (Note: Available in Hive 0.13 and later)
  | SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value,
property_name=property_value, ...)]

file_format:
  : SEQUENCEFILE
  | TEXTFILE    -- (Default, depending on hive.default.fileformat
configuration)
  | RCFILE      -- (Note: Available in Hive 0.6.0 and later)
  | ORC         -- (Note: Available in Hive 0.11.0 and later)
  | PARQUET     -- (Note: Available in Hive 0.13.0 and later)
```

```
  | AVRO       -- (Note: Available in Hive 0.14.0 and later)
  | INPUTFORMAT input_format_classname OUTPUTFORMAT
output_format_classname

constraint_specification:
  : [, PRIMARY KEY (col_name, ...) DISABLE NOVALIDATE ]
    [, CONSTRAINT constraint_name FOREIGN KEY (col_name, ...) REFERENCES
table_name(col_name, ...) DISABLE NOVALIDATE
```

例：

```
create table t2(
   id int comment 't2\'s ID',
   stu_name string comment 'name',
   stu_birthday date comment 'birthday',
   online boolean comment 'is online'
) row format delimited
fields terminated by '\t'
lines terminated by '\n';

create table t3_stu_hobby(
   id int comment 'ID',
   stu_name string comment 'stu name',
   stu_hobby array<string> comment "stu's hobby"
) row format delimited
fields terminated by '\t'
collection items terminated by ',';

create table t4_stu_scores(
   id int comment 'ID',
   stu_name string comment 'stu name',
   stu_scores map<string, int> comment "stu's scores"
) row format delimited
fields terminated by '\t'
collection items terminated by ','
map keys terminated by ':';

create table t5_staff_address (
   id int comment 'ID',
   staff_name string comment 'staff name',
   staff_address struct<home_addr:string, office_addr:string> comment "staff's address"
) row format delimited
fields terminated by '\t'
collection items terminated by ',';
```

```
create table employees (
   id int,
   salary float,
   subordinates array<string>,
   deductions map<string, float>,
   address struct<city:string,province:string,zip:int>
) row format delimited
fields terminated by '\001'
collection items terminated by '\002'
map keys terminated by '\003'
lines terminated by '\n';
```

```
create <external> table tblName (字段 类型...) location hdfs_uri;
```

例：

```
create external table t6_external (
   id int,
   name string,
   birthday date,
   online boolean )
row format delimited fields terminated by '\t' location '/hdfs/input/';
```

**RegEx**

```
CREATE TABLE logs(
   host STRING,
   identity STRING,
   username STRING,
   time STRING,
   request STRING,
   status STRING,
   size STRING,
   referer STRING,
   agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
"input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (\\[.*\\]) \"(.*?)\" (-|[0-9]*) (-|[0-9]*) \"(.*?)\" \"(.*?)\"",
"output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"
)
STORED AS TEXTFILE;
```

例：

```
CREATE TABLE logs(
   host STRING,
   identity STRING,
   username STRING,
```

```
   time STRING,
   request STRING,
   status STRING,
   size STRING,
   referer STRING,
   agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
"input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) \\[(.*)\\] \"(.*?)\" (-|[0-9]*) (-|[0-9]*) \"(.*?)\" \"(.*?)\""
)
STORED AS TEXTFILE;

load data local inpath 'hive-demo/access.log' into table logs;
select * from logs limit 10;

select substring(time,0,14) datetime ,host, count(*) as count
from logs
group by substring(time,0,14), host
having count > 100;
```

### Create Table As Select (CTAS)

CTAS has these restrictions:
The target table cannot be a partitioned table.
The target table cannot be an external table.
The target table cannot be a list bucketing table.
目标表不能是分区表，外部表或桶表
Example:

```
CREATE TABLE new_key_value_store
    ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
    STORED AS RCFile
    AS
SELECT (key % 1024) new_key, concat(key, value) key_value_pair
FROM key_value_store
SORT BY new_key, key_value_pair;
```

## RCFile orc parquet

RCFILE 是一种行列存储相结合的存储方式。首先，其将数据按行分块，保证同一个 record 在一个块上，避免读一个记录需要读取多个 block。其次，块数据列式存储，有利于数据压缩和快速的列存取。

Orc 文件格式是对 RCFile 的一个扩展和升级，拥有更高效的压缩比和执行效率。

Parquet 是新起的一种列式存储文件类型

创建方式

```
create table t1
    stored as orc
    as
    select * from t3;
```

# Hive ACID(仅 orc 格式的分桶表支持)

```
create table test(id int, name string)
clustered by (id) into 3 buckets
stored as orc
tblproperties('transactional'='true');
```

插入数据：

```
insert into table test values(1, 'zhangsan'),(2, 'lisi'),(3, wangwu');

update test set name='zhaoliu' where id=1;
```

删除操作：delete from test where id=1;

ACID 概念

- Atomicity（原子性）
  - 要么全部完成，要么全部不完成
  - 事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态
- Consistency（一致性）
  - 不论事务并发多少个，系统也必须如同串行事务一样操作
  - 例如：A 与 B 账户之间转账 5 元，B 与 E 之间转账 15 元，三个账户总额不变
- Isolation（隔离性）
  - 在同一时间仅有一个请求修改或获取同一数据，每一事务会被串行化处理
- Durability（持久性）
  - 在事务完成以后，所作的更改便持久的保存在数据库中

支持事务（Transaction）的数据库

在事务过程（Transaction processing）能够保证数据的正确性

ACID 实现的两种方式

- Write ahead logging
- Shadow paging

在 cli 配置不起作用，必须在 hive-site.xml 配置

```
<property>
    <name>hive.support.concurrency</name>
    <value>true</value>
```

```
        <description>
          控制是否开发并发控制，当使用 zookeeper Hive lock manager 时，zookeeper 必须运行
        </description>
    </property>
    <property>
        <name>hive.txn.manager</name>
        <value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>
        <description>
          设置 DbTxnManager 开启事物，同时需设置
          hive.compactor.initiator.on,
          hive.compactor.worker.threads,
          hive.support.concurrency (true),
          hive.enforce.bucketing(true),
          hive.exec.dynamic.partition.mode (nonstrict).
          默认的 DummyTxnManager 不提供事务
        </description>
    </property>
    <property>
        <name>hive.enforce.bucketing</name>
        <value>true</value>
    </property>
    <property>
        <name>hive.compactor.initiator.on</name>
        <value>true</value>
    </property>
    <property>
        <name>hive.compactor.worker.threads</name>
        <value>2</value>
    </property>
    <property>
        <name>hive.exec.dynamic.partition.mode</name>
        <value>nonstrict</value>
    </property>
    <property>
      <name>hive.txn.max.open.batch</name>
      <value>1000</value>
      <description>
        事务的最大数量，可以用 open_txns()获取
      </description>
    </property>
```

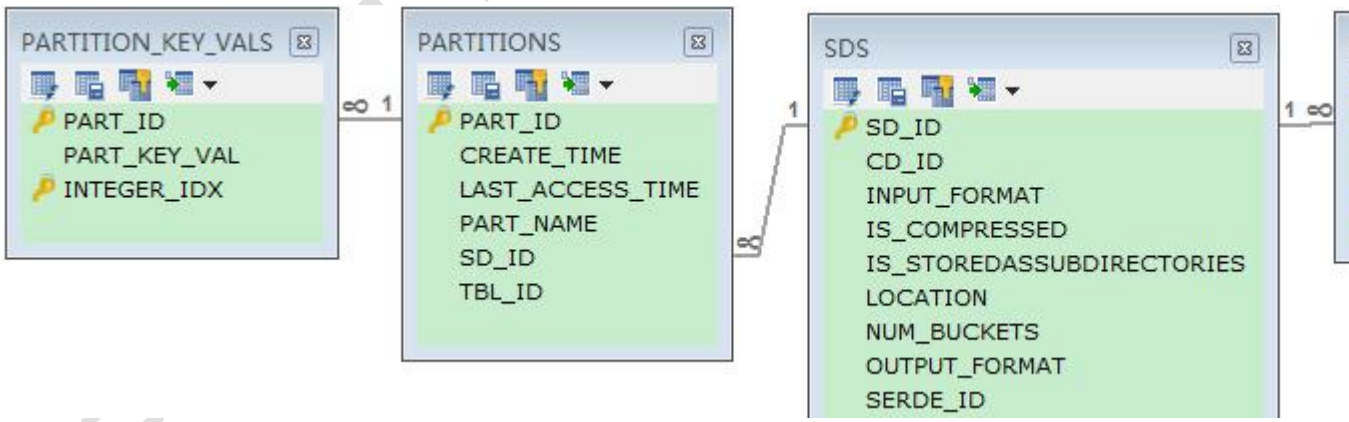
如果在这段时间内，客户端没有发出心跳信号，事务就会被宣告终止。

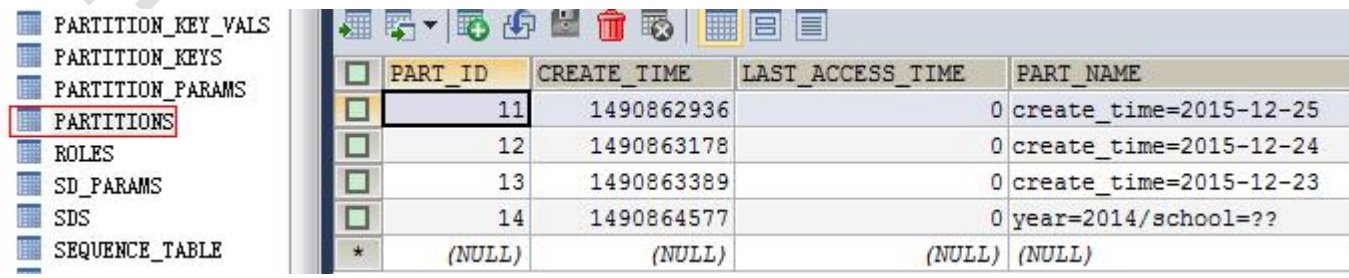
hive.txn.timeout

300

压缩作业在该时间内未完成将被宣布失败，压缩操作被重新排入队列。
hive.compactor.worker.timeout
86400
检查是否有分区需要被压缩。
hive.compactor.check.interval
300
delta 目录的数目，达到该数后将触发次要压缩。
hive.compactor.delta.num.threshold
10
与基础文件中 delta 文件的百分比，达到该值会触发主要压缩。（1 = 100%）
hive.compactor.delta.pct.threshold
0.1
给定分区中已中止事务的数目，达到该数会也会触发主要压缩。
hive.compactor.abortedtxn.threshold
1000

## 5.2.1 分区表




例：
create table t7_partition_1 (
id int comment 'ID',

```
    name string comment 'name'
) partitioned by (create_time date comment 'create time')
row format delimited
fields terminated by '\t';

create table t7_partition_2 (
    id int comment 'ID',
    name string comment 'name'
) partitioned by (year int comment 'admission year',
school string comment 'school name')
row format delimited
fields terminated by '\t';
```

查看表分区：

```
show partitions    t7_partition_1;
```

删除分区：

```
alter table tblName drop partition(partition_spec);
alter table t7_partition_1 drop partition(create_time='2015-12-25');
```

添加分区

```
alter table t7_partition_1 add partition(create_time='2015-12-25');
alter table t7_partition_2 add partition(year=2014,school='呵呵');
```

将指定分区的数据打成 har 包

```
alter table t7_partition_1 archive partition(create_time='2015-12-25');
```

解开指定分区的 har 包

```
alter table t7_partition_1 unarchive partition(create_time='2015-12-25');
```

防止分区被删除或被查询

```
alter table t7_partition_1 partition(create_time='2015-12-25') enable no_drop;
alter table t7_partition_1 partition(create_time='2015-12-25') enable offline;
```

恢复

```
alter table t7_partition_1 partition(create_time='2015-12-25') disable no_drop;
alter table t7_partition_1 partition(create_time='2015-12-25') disable offline;
```

修改已存在分区的位置

```
alter table t7_partition_1 partition(create_time='2015-12-25') set location 'hdfs://mini1:9000/hdfs/sheets';
```

添加分区同时指定位置

```
alter table t7_partition_1 add partition(create_time='2015-12-24') location 'hdfs://mini1:9000/hdfs/input';
```

加载数据到分区

```
load data local inpath '/root/t1' into table t7_partition_1 partition(create_time='2015-12-23');
```

对分区表查询必须加分区过滤
set hive.mapred.mode=strict;
不严格模式
set hive.mapred.mode=nostrict;

如何用脚本自动给分区表 test(分区字段 statDate string)每天创建一个分区？
yesterday=`date -d '1 days ago' +'%Y%m%d'`
hive -e "use default;alter table test add partition
(statdate='$yesterday') location '/data/workcopy/$yesterday';"

## 5.2.2 桶表

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
        page_url STRING, referrer_url STRING,
        ip STRING COMMENT 'IP Address of the User')
  COMMENT 'This is the page view table'
  PARTITIONED BY(dt STRING, country STRING)
  CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
  ROW FORMAT DELIMITED
     FIELDS TERMINATED BY '\001'
     COLLECTION ITEMS TERMINATED BY '\002'
     MAP KEYS TERMINATED BY '\003'
  STORED AS SEQUENCEFILE;
```

例：
create table t8_bucket(id int) clustered by (id) into 3 buckets;

```
create table sc_buck(Sno int,Cno int,Grade int)
clustered by(Sno)
sorted by(Grade DESC)
into 4 buckets
row format delimited
fields terminated by ',';
```

开启分桶，使 reduce 的数量自动适配 bucket 的个数
set hive.enforce.bucketing=true;
向桶表插入数据
insert into table t8_bucket select * from t1 where id is not null;
insert overwrite table sc_buck
select * from sc cluster by(Sno);

ALTER TABLE table_name CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name, ...)]

INTO num_buckets BUCKETS;

### 5.2.3 分区表分区归档 har

需走 mapreduce，速度较慢

```
mkdir $HIVE_HOME/auxlib/
cp $HADOOP_HOME/share/hadoop/tools/lib/hadoop-archives-2.6.0.jar $HIVE_HOME/auxlib/


set hive.archive.enabled=true;
alter table t7_partition_1 archive partition(create_time='2012-04-07');
```

解归档

```
alter table t7_partition_1 unarchive partition(create_time='2012-04-07');
```

## 5.3 修改表属性（描述&分隔符）

修改表描述：

```
alter table t1 set tblproperties ('comment' = 'new_comment');
```

内部表与外部表互转

外-->内：

```
alter table external_tbl set tblproperties("EXTERNAL"="FALSE");
```

然后把元数据中 TBLS 表对应行的 TBL_TYPE 修改为 MANAGED_TABLE

内-->外：

```
alter table managed_tbl set tblproperties("EXTERNAL"="TRUE");
```

然后把元数据中 TBLS 表对应行的 TBL_TYPE 修改为 EXTERNAL_TABLE

修改表的分隔符

```
alter table t1 set serdeproperties('field.delim'='\t');
alter table t1 set serdeproperties('colelction.delim'='\t');
alter table t1 set serdeproperties('mapkey.delim'=':');
alter table t1 set serdeproperties('line.delim'='\n');

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

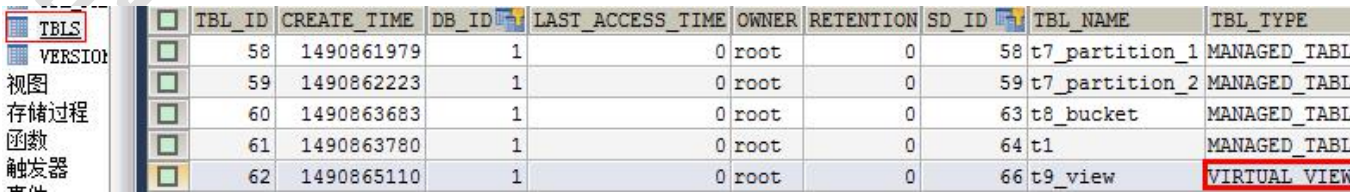
partition_col_value, ...)
例：
truncate table t1;

## 主键外键

ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (column, ...) DISABLE NOVALIDATE;
ALTER TABLE table_name ADD CONSTRAINT constraint_name FOREIGN KEY (column, ...) REFERENCES table_name(column, ...) DISABLE NOVALIDATE RELY;
ALTER TABLE table_name DROP CONSTRAINT constraint_name;

## 5.4 视图

CREATE VIEW [IF NOT EXISTS] [db_name.]view_name [(column_name [COMMENT column_comment], ...) ]
  [COMMENT view_comment]
  [TBLPROPERTIES (property_name = property_value, ...)]
  AS SELECT ...;
Example:
CREATE VIEW onion_referrers(url COMMENT 'URL of Referring page')
  COMMENT 'Referrers to The Onion website'
  AS
  SELECT DISTINCT referrer_url
  FROM page_view
  WHERE page_url='http://www.theonion.com';
例：
create view t9_view as select * from t2;




| TBL_ID | CREATE_TIME | DB_ID | LAST_ACCESS_TIME | OWNER | RETENTION | SD_ID | TBL_NAME | TBL_TYPE |
|---|---|---|---|---|---|---|---|---|
| 58 | 1490861979 | 1 | 0 | root | 0 | 58 | t7_partition_1 | MANAGED_TABL |
| 59 | 1490862223 | 1 | 0 | root | 0 | 59 | t7_partition_2 | MANAGED_TABL |
| 60 | 1490863683 | 1 | 0 | root | 0 | 63 | t8_bucket | MANAGED_TABL |
| 61 | 1490863780 | 1 | 0 | root | 0 | 64 | t1 | MANAGED_TABL |
| 62 | 1490865110 | 1 | 0 | root | 0 | 66 | t9_view | VIRTUAL_VIEW |

DROP VIEW [IF EXISTS] [db_name.]view_name;
例：
DROP VIEW onion_referrers;

ALTER VIEW [db_name.]view_name SET TBLPROPERTIES table_properties;

table_properties:
  : (property_name = property_value, property_name = property_value, ...)

修改视图：
ALTER VIEW [db_name.]view_name AS select_statement;

## 5.5 索引

索引可加快含有 group by 语句的查询速度
hive.optimize.index.filter：自动使用索引
hive.optimize.index.groupby：使用聚合索引优化 GROUP BY 操作

```
CREATE INDEX index_name
  ON TABLE base_table_name (col_name, ...)
  AS index_type
  [WITH DEFERRED REBUILD]
  [IDXPROPERTIES (property_name=property_value, ...)]
  [IN TABLE index_table_name]
  [
      [ ROW FORMAT ...] STORED AS ...
      | STORED BY ...
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (...)]
  [COMMENT "index comment"];
```

创建索引
```
create index t1_index on table t1(id) as
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
with deferred rebuild in table t1_index_table;
```
as 指定索引器
重建索引
```
ALTER INDEX index_name ON table_name [PARTITION partition_spec] REBUILD;
```
例：
```
alter index t1_index on t1 rebuild;
```
显示索引
```
show formatted index on t1;
```
删除索引
```
DROP INDEX [IF EXISTS] index_name ON table_name;
```
例：
```
drop index if exists t1_index on t1;
```

## 5.6 show

SHOW (DATABASES|SCHEMAS) [LIKE 'identifier_with_wildcards'];

SHOW TABLES [IN database_name] ['identifier_with_wildcards'];

```
SHOW VIEWS [IN/FROM database_name] [LIKE 'pattern_with_wildcards'];
```

例：

```
SHOW VIEWS;                                    -- show all views in the current database
SHOW VIEWS 'test_*';                           -- show all views that start with "test_"
SHOW VIEWS '*view2';                           -- show all views that end in "view2"
SHOW VIEWS LIKE 'test_view1|test_view2';       -- show views named either "test_view1" or
"test_view2"
SHOW VIEWS FROM test1;                         -- show views from database test1
SHOW VIEWS IN test1;                           -- show views from database test1 (FROM and
IN are same)
SHOW VIEWS IN test1 "test_*";                  -- show views from database test2 that start
with "test_"
```

```
SHOW PARTITIONS table_name;
SHOW PARTITIONS table_name PARTITION(ds='2010-03-03');          -- (Note:
Hive 0.6 and later)

SHOW PARTITIONS [db_name.]table_name [PARTITION(partition_spec)];    --
(Note: Hive 0.13.0 and later)
```

例：

```
SHOW PARTITIONS databaseFoo.tableBar PARTITION(ds='2010-03-03', hr='12');
```

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

例：

```
show table extended like part_table;

SHOW TBLPROPERTIES tblname;
SHOW TBLPROPERTIES tblname("foo");

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

## 5.7 数据装载&导出

load data [local] inpath 'filepath' [overwrite] into table tablename [partition (partcol1=val1, partcol2=val2 ...)]
例：
load data local inpath '/root/t1' into table t7_partition_1 partition(create_time='2015-12-23');

insert [into/overwrite] table t1 select columns... from t2;
例:
insert overwrite table t8_bucket select * from t1 where id is not null;

通过其他表的查询结果创建表
create table new_tblname as select [cloumns...] from other_tblname;
例：
create table t2  as select * from t1;

同一份数据多种处理
写入到指定表
```
from t1
   insert overwrite table t2 select * where id%3==1
   insert overwrite table t3 select * where id%3==2;
```
写入到指定文件
```
FROM TABLE1
   INSERT OVERWRITE LOCAL DIRECTORY '/data/data_table/data_table1.txt' SELECT 20140303, col1, col2, 2160701, COUNT(DISTINCT col) WHERE col3 <= 20140303 AND col3 >= 20140201 GROUP BY col1, col2
   INSERT OVERWRITE LOCAL DIRECTORY '/data/data_table/data_table2.txt' SELECT 20140302, col1, col2, 2160701, COUNT(DISTINCT col) WHERE col3 <= 20140302 AND col3 >= 20140131 GROUP BY col1, col2
   INSERT OVERWRITE LOCAL DIRECTORY '/data/data_table/data_table3.txt' SELECT 20140301, col1, col2, 2160701, COUNT(DISTINCT col) WHERE col3 <= 20140301 AND col3 >= 20140130 GROUP BY col1, col2;
```

导出 t1 表到 hdfs 目录
export table t1 to '/data';
将 hdfs 的数据导入到 t3 表
import table t3 from '/data';

使用 DIRECTORY
INSERT OVERWRITE [LOCAL] DIRECTORY '...' SELECT ...FROM...WHERE ...;

例：
insert overwrite local directory '/root/reuslt' select * from t1;

## 5.10Hive 的分隔符

行分隔符'\n'
列分隔符\001
在文本中可以通过 ctrl+v 然后 ctrl+a 来输入\001
元素分隔符\002
在键盘上通过 ctrl+v 和 ctrl+b 来输入
键值对分隔符为\003
通过 ctrl+v 和 和 ctrl+c 输入

## 5.11 读模式与写模式区别

读模式是指读取数据的时候对数据合法性进行校验：
写数据的时候不校验，适合于大量数据的一次性加载

写模式是指读取数据的时候对数据合法性进行校验：
这样存储在表中的数据都是是合乎规范的 ，有利于快速查询和快速处理

Hive 是读模式。当上传文件到表中，hive 不会对表中的数据进行校验，对于错误的数据格式查询的时候也不会报错，只是显示为 NULL。

# hive 查询

## like 和 rlike

like　　_表示一个任意字符，%表示 0 个或多个任意字符
rlike　　完全兼容 java 正则表达式

## 浮点数比较

用 cast(0.2 as float)转换
尽量使用 round，floor 或 ceil 转换为整数

## hive 数据类型

### Number Types

| Type | size | Postfix | Example |
| --- | --- | --- | --- |
| TINYINT | 1byte | Y | 100Y |
| SMALLINT | 2byte | S | 100S |
| INT | 4byte | | 100 |
| BIGINT | 8byte | L | 100L |

FLOAT　　4byte 单精度
DOUBLE　8byte 双精度

DECIMAL type　基于 Java's BigDecimal

```
CREATE TABLE foo (
   a DECIMAL, -- Defaults to decimal(10,0)
   b DECIMAL(9, 7)
)
```

DECIMAL(9, 7)表示最大精度是 9 小数位是 7,9 代表十进制数值总长度
hive 0.12.0 和 0.13.0 的 Decimal 类型不兼容
需要使用如下语句修正

```
ALTER TABLE foo CHANGE COLUMN dec_column_name dec_column_name DECIMAL(38,18);
```

字符串 string 使用单引号或双引号表示
varchar 是指定最大长度的字符串
Char 是指定长度的字符串，超过长度的字符串会被截断，少于长度的会在尾部补充空白

cast(string as date) 如果字符串是'YYYY-MM-DD'格式，将返回 year/month/day 的 date 类型.
否则返回 NULL.
cast(date as string) 返回 'YYYY-MM-DD'格式的字符串.

## Allowed Implicit Conversions

| | void | boolean | tinyint | smallint | int | bigint | float | double | decima |
|---|---|---|---|---|---|---|---|---|---|
| **void to** | true | true | true | true | true | true | true | true | true |
| **boolean to** | false | true | false | false | false | false | false | false | false |
| **tinyint to** | false | false | true | true | true | true | true | true | true |
| **smallint to** | false | false | false | true | true | true | true | true | true |
| **int to** | false | false | false | false | true | true | true | true | true |
| **bigint to** | false | false | false | false | false | true | true | true | true |
| **float to** | false | false | false | false | false | false | true | true | true |
| **double to** | false | false | false | false | false | false | false | true | true |
| **decimal to** | false | false | false | false | false | false | false | false | true |
| **string to** | false | false | false | false | false | false | false | true | true |
| **varchar to** | false | false | false | false | false | false | false | true | true |
| **timestamp to** | false | false | false | false | false | false | false | false | false |
| **date to** | false | false | false | false | false | false | false | false | false |
| **binary to** | false | false | false | false | false | false | false | false | false |



# Hive 的排序

order by 排序使用全局排序只有一个 reducer task 运行。

sort by 排序是在每个 reducer task 里面进行局部排序，不做全局排序。
cluster by ...是 distribute by ... sort by ...的简写。

```
distribute by s.symbol
 sort by s.symbol asc,s.ymd asc;
```

distribute by 表示分区字段，指定的字段相同字段内容会进入同一个 reducer
sort by 指定排序列。
asc 是默认值，可以省略

# having

对于聚合函数的结果只能通过 having 过滤，否则需要通过嵌套 select 子查询进行 where 过滤

# 表连接

连接共有：
join(inner join),left join(left outer join),right join(right outer join),full join(full outer join)

注意：
hive 的连接条件一定要在 on 中，否则会笛卡尔积
不允许笛卡尔积查询
set hive.mapred.mode=strict;
允许
set hive.mapred.mode=nonstrict;

还有 LEFT SEMI-JOIN
left semi join 是 in/exists 子查询的一种更高效的实现。
hive 当前没有实现 in/exists 子查询，所以你可以用 left semi join 重写你的子查询语句。
left semi join 的限制是，join 子句中右边的表只能在 on 子句中设置过滤条件，在 where 子句、select 子句或其他地方过滤都不行。

```
select a.key, a.value from a
where a.key in (select b.key from b);
```

可以被重写为：

```
select a.key, a.value from a
left semi join b on (a.key = b.key)
```

但是 hive 不支持

```
select s.ymd, a.symbol,s.price_close from stocks s
where s.ymd,s.symbol in
(select d.ymd,d.symbol from dividends d);
```

只能使用 left semi join

```
select s.ymd, a.symbol,s.price_close from stocks s
left semi join dividends d on s.ymd=d.ymd and s.symbol=d.symbol;
```

hive 只支持等值连接

```
select
   e.id, e.name, e.age, if(e.sex==1, '男','女') as gender,
   s.salary, d.name from t_dept d
   join t_employee e on d.id = e.deptid
   join t_salary s on e.id = s.empid
where e.name in ('张三','韩梅梅') and s.salary > 5000;
```

Hive 会对每一对 JOIN 连接对象启动一个 MapReduce 任务。

本例中，

会首先启动一个 MR Job

对表 d 和表 e 进行连接操作，

然后会在启动一个 MR Job

将第一个 MRJob 的输出和表 c 进行连接操作。

最后对结果进行过滤

可改写成先过滤，再连接

```
select
   e.id, e.name, e.age,
   if(e.sex==1, '男','女') as gender,
   s.salary, d.name
from t_dept d
join
   (select * from t_employee where name in ('张三','韩梅梅')) e
on d.id = e.deptid
join
   (select * from t_salary where salary > 5000) s
on e.id = s.empid;
```



# join 优化

使连续查询的表的大小从左到右依次增加，hive 在对每行操作时，会尝试把其他表缓存起来

也可以通过 /*+STREAMTABLE(d)*/ 显式指定大表

**mapjoin**

Hive v0.12.0 版本，设置如下参数

set hive.auto.convert.join=true;

在 hive0.11 在表的大小符合设置时

hive.mapjoin.smalltable.filesize=25000000，

默认会自动把 join 转换为 map join

是否将多个 mapjoin 合并成一个

set hive.auto.convert.join.noconditionaltask=true;

多个 mapjoin 合并成一个 mapjoin 时，其表的总的大小须小于该值

set hive.auto.convert.join.noconditionaltask.size=10000;

/*+ MAPJOIN(d)*/显式指定小表

使用默认启动该优化的方式如果出现默名奇妙的 BUG(比如 MAPJOIN 并不起作用),

就设置不忽略 mapjoin 标记，不自动进行 mapjoin.

set hive.ignore.mapjoin.hint=fase;

set hive.auto.convert.join=fase;

然后在查询语句中显式指定：

```
select /*+ MAPJOIN(d)*/ e.id, e.name, e.age,
   if(e.sex==1, '男','女') as gender,
   s.salary, d.name
from t_dept d
inner join t_employee e on d.id = e.deptid
inner join t_salary s on e.id = s.empid;
```

```
select /*+MAPJOIN(smallTableTwo)*/ idOne, idTwo, value FROM
   ( select /*+MAPJOIN(smallTableOne)*/ idOne, idTwo, value FROM
      bigTable JOIN smallTableOne
      on bigTable.idOne = smallTableOne.idOne) firstjoin
JOIN smallTableTwo
ON firstjoin.idTwo = smallTableTwo.idTwo;
```

**bucketmapjoin**

对两个表在 join 的 key 上都做 hash bucket，之后可以用 bucket mapjoin，因为 bucket 划分数据后,相同 hash 值的 key 都放在了同一个 bucket 中，在 join 时，小表将会被复制到所有节点, map 加载小表的 bucket 到内存 hashtable，与大表对应的 bucket 文件进行 join.

开启桶表的 mapjoin

set hive.optimize.bucketmapjoin=true;

操作例子：

```
 创建表，并以 id 字段划分桶，桶个数为 20 个
（也就是在插入数据时会生成 20 个数据文件对应 20 个桶）
create table test_mapjoin(id int) clustered by (id) into 20 buckets;
插入 300 万行数据
set hive.enforce.bucketing=true;
```

```
insert overwrite table test_mapjoin select * from ids limit 3000000;
使用 bucket mapjoin
set hive.optimize.bucketmapjoin=true;
select /*+mapjoin(a)*/ count(*) from test_mapjoin a join test_mapjoin b on a.id = b.id;
```

**sort-merge-bucket (smb) join**

表 a 为：

```
create table a(key int, othera string)
clustered by(key) into 4 buckets
row format delimited
fields terminated by '\001'
collection items terminated by '\002'
map keys terminated by '\003'
stored as sequencefile;
```

表 b 为：

```
create table b(key int, otherb string)
clustered by(key) into 32 buckets
row format delimited
fields terminated by '\001'
collection items terminated by '\002'
map keys terminated by '\003'
stored as sequencefile;
```

因为 join 列都是分桶列,则可开启 **bucketmapjoin**：

```
set hive.optimize.bucketmapjoin = true;
select /*+ mapjoin(b) */ a.key, a.value from a join b on a.key = b.key;
```

这样，join 的过程是，表 a 的 bucket 1 只会与表 b 中的 bucket 1 进行 join，而不再考虑表 b 中的其他 bucket 2~32。

如果上述表具有相同的 bucket，如都是 32 个，而且还是排序的，亦即，

```
clustered by(key) sorted by(key) into 32 buckets
```

设置如下参数

```
set hive.input.format=org.apache.hadoop.hive.ql.io.bucketizedhiveinputformat;
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
```

则上述 join 语句会执行一个 sort-merge-bucket (smb) join

# 数据倾斜

## null 值过多导致数据倾斜

解决方法 1： userid 为空的不参与关联，然后与为空的结果合并

```
select * from log l
 left join user u
    on l.userid is not null
    and l.userid = u.userid
union all
select * from log where userid is null;
```

解决方法 2 ：赋与空值分新的 key 值

```
select * from log l
left join user u on
   case when l.userid is null
          then concat('hive',rand())
          else l.userid
 and l.userid = u.userid;
```

## 小表不小不大，map join 解决倾斜示例

使用 map join 解决小表(记录数少)关联大表的数据倾斜问题，
这个方法使用的频率非常高，但如果小表很大，大到 map join 会出现 bug 或异常，
这时就需要特别的处理。
users 表有 600w+ 的记录，把 users 分发到所有的 map 上也是个不小的开销，
而且 map join 不支持这么大的小表。如果用普通的 join，又会碰到数据倾斜的问题。
以下例子:

```
select * from log a
  left join users b
  on a.user_id = b.user_id;
```

解决方法：

```
select /*+mapjoin(x)*/ * from log a
left join (
  select   /*+mapjoin(c)*/ d.*
     from (select distinct user_id from log) c
     join users d
     on c.user_id = d.user_id
  ) x
on a.user_id = x.user_id;
```

假如，log 里 user_id 有上百万个，这就又回到原来 map join 问题。所幸，每日的会员 uv 不会太多，有交易的会员不会太多，有点击的会员不会太多，有佣金的会员不会太多等等。所以这个方法能解决很多场景下的数据倾斜问题。

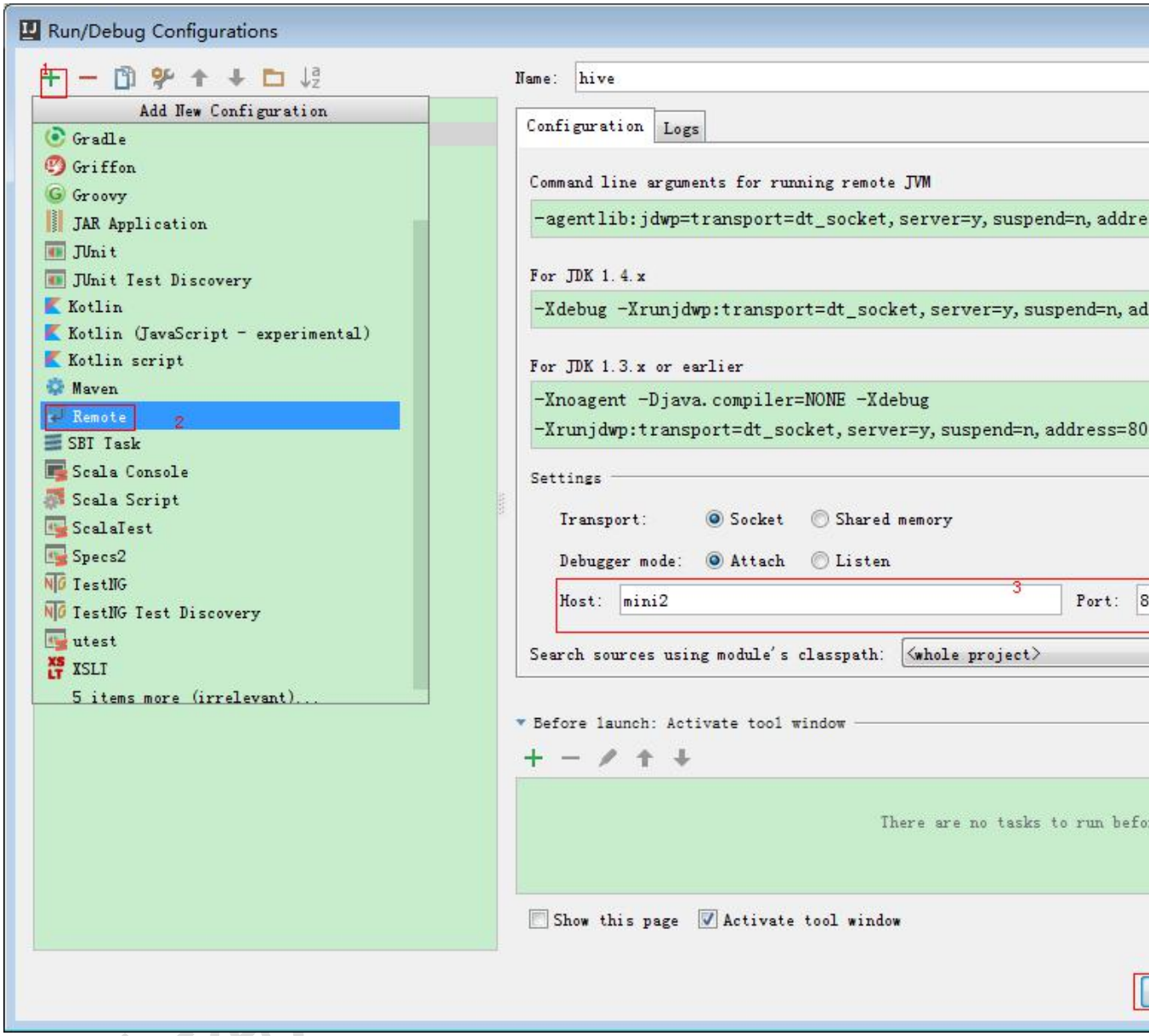
# Hive 开发

## debug 模式

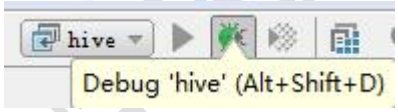
hive -hiveconf hive.root.logger=DEBUG,console

## debug 远程调试 hive

先启动 hive 远程调试
hive --debug
出现如下提示
Listening for transport dt_socket at address: 8000
然后在 idea 新建一个远程连接应用




然后以 debug 形式运行即可




# 自定义 hook

Hive 中没有超级管理员一说，任何用户都可以进行 grant/revoke 等相关操作可以通过前置 hook 实现指定用户为超级管理员

写一个类继承 Hive 中的钩子 AbstractSemanticAnalyzerHook，并打成 jar 包，上传至 $HIVE_HOME/lib/目录下
然后在 hive-site.xml 中配置

```
<property>
  <name>hive.semantic.analyzer.hook</name>
  <value>hive.AuthHook</value>
</property>
```

代码如下：

```
package hive;

import org.apache.hadoop.hive.ql.parse.*;
import org.apache.hadoop.hive.ql.session.SessionState;

public class AuthHook extends AbstractSemanticAnalyzerHook {
    // 默认管理员账号
    private static String admin = "root";
    /**
     * 作为超级管理员，其实质是当拥有删除、创建等等操作之前，对以上行为进行拦截
     * 所以需要再 hook 切面里面来做
     */
    @Override
    public ASTNode preAnalyze(HiveSemanticAnalyzerHookContext context, ASTNode ast) throws SemanticException {
        switch (ast.getToken().getType()) {
            case HiveParser.TOK_CREATEDATABASE:
            case HiveParser.TOK_DROPDATABASE:
            case HiveParser.TOK_CREATEROLE:
            case HiveParser.TOK_DROPROLE:
            case HiveParser.TOK_GRANT:
            case HiveParser.TOK_REVOKE:

            case HiveParser.TOK_GRANT_ROLE:
            case HiveParser.TOK_REVOKE_ROLE:
                String userName = null;
                if (SessionState.get() != null && SessionState.get().getAuthenticator() != null)
                    userName = SessionState.get().getAuthenticator().getUserName();
                if (!admin.equalsIgnoreCase(userName))
                    throw new SemanticException(userName + " can't use ADMIN options, except " + admin + ".");
                break;
            default:
                break;
        }
        return ast;
    }
}
```

# 自定义 UDF 函数

1、自定义 UDF extends org.apache.hadoop.hive.ql.exec.UDF
2、需要实现 evaluate 函数，evaluate 函数支持重载
3、把程序打包放到目标机器上去
4、进入 hive 客户端，添加 jar 包：hive>add jar jar 路径
可添加 hdfs 路径
add jar hdfs://mini1:9000/jars/hive.jar;
5、创建临时函数:
CREATE TEMPORARY FUNCTION 自定义名称 AS '自定义 UDF 的全类名'
6、执行 HQL 语句；
7、销毁临时函数:
DROP TEMPORARY FUNCTION 自定义名称
注： ：UDF 只能实现一进一出的操作.
可通在类上增加 Description 注解增加函数描述

@Description(
name = "MyUpper",
value = " _FUNC_(str) - Returns str with all characters changed to uppercase",
extended = "Example:\n > SELECT _FUNC_(name) FROM src;")

操作案例：

编写代码

```
@Description(name = "MyUpper",
      value = " _FUNC_(str) - Returns str with all characters changed to uppercase",
      extended = "Example:\n > SELECT _FUNC_(name) FROM src;")
public class MyUpper extends UDF {
   public String evaluate(String text) {
      return text==null?null:text.toUpperCase();
   }
}
```

打 jar 包并上传

```
add jar /root/hive.jar;
create temporary function myUp as 'lidacui.MyUpper';
select myup(line) from dula;
```

```
@Description(name = "ZodiacOrConste",
       value = " _FUNC_(data,int) - The specified date into the zodiac or constellation," +
```

```
                " 0 on behalf of the zodiac, the constellation of the 1",
        extended = "Example:\n"
                + " > SELECT _FUNC_(name,0) FROM src;\n"
                + " > SELECT _FUNC_(name,1) FROM src;")
public class ZodiacOrConstellationUDF extends UDF {

    public final String[] zodiacArr = {"猴", "鸡", "狗", "猪", "鼠", "牛", "虎", "兔", "龙",
"蛇", "马", "羊"};
    public final String[] constellationArr = {"水瓶座", "双鱼座", "白羊座", "金牛座", "双子
座", "巨蟹座",
            "狮子座", "处女座", "天秤座", "天蝎座", "射手座", "魔羯座"};
    public final int[] constellationEdgeDay = {20, 19, 21, 21, 21, 22, 23, 23, 23, 23, 22, 22};

    /**
     * @param birthday
     * @param type     0 表示要获取生肖 1 表示要获取星座
     * @return
     */
    public String evaluate(Date birthday, Integer type) {
        if (birthday == null)
            return null;
        if (type == 0)
            return getZodica(new java.util.Date(birthday.getTime()));
        else if (type == 1)
            return getConstellation(new java.util.Date(birthday.getTime()));

        return null;
    }
    /**
     * 根据日期获取生肖
     * @return
     */
    public String getZodica(java.util.Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        return zodiacArr[cal.get(Calendar.YEAR) % 12];
    }

    /**
     * 根据日期获取星座
     * @return
     */
    public String getConstellation(java.util.Date date) {
        Calendar cal = Calendar.getInstance();
```

```
        cal.setTime(date);
        int month = cal.get(Calendar.MONTH);
        int day = cal.get(Calendar.DAY_OF_MONTH);
        if (day < constellationEdgeDay[month])
            month = month - 1;
        if(month==-1) month=11;
        return constellationArr[month];
    }
}
```

```
add jar /root/hive.jar;
create temporary function zoc as 'lidacui.ZodiacOrConstellationUDF';
select name,time,zoc(time,0) zodiac,zoc(time,1) constellation from t1;
```

# 自定义 GenericUDF 函数

操作案例：

```
@Description(name = "nvl", value = "_FUNC_(expr1, expr2) - Returns expr2 if expr1 is null",
        extended = "Example:\n"
      + " > SELECT _FUNC_(dep, 'Not Applicable') FROM src;\n 'Not Applicable' if dep is null")
public class GenericUDFNVL extends GenericUDF {
   private ObjectInspector[] argumentOIs;
   private GenericUDFUtils.ReturnObjectInspectorResolver returnOIResolver;
   @Override
   public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {
      if (arguments.length != 2)
         throw new UDFArgumentLengthException("The function nvl(expr1, expr2) needs at least two arguments.");
      for (int i = 0; i < arguments.length; i++)
         if (arguments[i].getCategory() != ObjectInspector.Category.PRIMITIVE)
            throw new UDFArgumentTypeException(i, "Only primitive type arguments are accepted but "
                  + arguments[i].getTypeName() + " is passed.");
        //保存参数类型列表
        argumentOIs = arguments;
        //用于获取参数值的对象，需调用 update 更新要获取的参数
      returnOIResolver = new GenericUDFUtils.ReturnObjectInspectorResolver(true);
      if (!(returnOIResolver.update(arguments[0]) && returnOIResolver.update(arguments[1])))
         throw new UDFArgumentTypeException(1,
               "The first and the second arguments of function NVL should have the same type, "
                     + "but they are different: \"" + arguments[0].getTypeName() + "\" and \""
                     + arguments[1].getTypeName() + "\"");
      return returnOIResolver.get();
```

```
    }
    @Override
    public Object evaluate(DeferredObject[] arguments) throws HiveException {
        // fieldValue is null, return defaultValue
        Object retVal = returnOIResolver.convertIfNecessary(arguments[0].get(), argumentOIs[0]);
        if(retVal==null)
            retVal=returnOIResolver.convertIfNecessary(arguments[1].get(), argumentOIs[1]);
       return retVal;
    }
    @Override
    public String getDisplayString(String[] children) {
       StringBuilder sb = new StringBuilder();
       sb.append("if ").append(children[0]).append(" is null returns").append(children[1]);
       return sb.toString();
    }
}
```

```
add jar /root/hive.jar;
create temporary function nvl as 'com.wangzhe.hivefunc.GenericUDFNVL';
select nvl(1,2),nvl(null,5),nvl(null,"hello") from t1;
```

## UDFType 三大属性

1、deterministic

如果传给一个 UDF 的参数值相同，无论调用多少次 UDF，它的返回值都是相同的，我们就认为这个 UDF 是确定性 UDF。上例中就属于不确定的。

2、ststeful

有状态的 UDF，UDF 实例内部会维护一个状态，每次调用该 UDF 的 evaluate 方法，该状态可能会改变，如上例行号属性。

3、distinctLike

这个标记通常出现在聚合函数中，在做聚合时，如果碰到两个相同的值，我们只取其中一个不影响最后的聚合结果，那么 distinctLike 为真，否则为假。

# hive 函数

所有的内置函数都注册在
org.apache.hadoop.hive.ql.exec.FunctionRegistry 类中
以静态代码快的形式进行注册

show functions;
desc function functionName;
desc function extended functionName;

例：
desc function year;
year(param) - Returns the year component of the date/timestamp/interval

desc function extended when;
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END - When a = true, returns b; when c = true, return d; else return e
Example:

```
 SELECT
 CASE
    WHEN deptno=1 THEN Engineering
    WHEN deptno=2 THEN Finance
    ELSE admin
 END,
 CASE
    WHEN zone=7 THEN Americas
    ELSE Asia-Pac
 END
 FROM emp_details
```

desc function extended concat;
concat(str1, str2, ... strN) - returns the concatenation of str1, str2, ... strN or concat(bin1, bin2, ... binN) - returns the concatenation of bytes in binary data  bin1, bin2, ... binN
Returns NULL if any argument is NULL.
Example:

```
   > SELECT concat('abc', 'def') FROM src LIMIT 1;
   'abcdef'
```

desc function extended concat_ws;
concat_ws(separator, [string | array(string)]+) - returns the concatenation of the strings separated by the separator.
Example:

```
   > SELECT concat_ws('.', 'www', array('facebook', 'com')) FROM src LIMIT 1;
```

'www.facebook.com'

desc function extended collect_list;
collect_list(x) - Returns a list of objects with duplicates

desc function extended collect_set;
collect_set(x) - Returns a set of objects with duplicate elements eliminated

desc function extended split;
split(str, regex) - Splits str around occurances that match regex
Example:
> SELECT split('oneAtwoBthreeC', '[ABC]') FROM src LIMIT 1;
["one", "two", "three"]

desc function extended explode;
explode(a) - separates the elements of array a into multiple rows, or the elements of a map into multiple rows and columns

desc function extended case;
CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END - When a = b, returns c; when a = d, return e; else return f
Example:

```
SELECT
CASE deptno
   WHEN 1 THEN Engineering
   WHEN 2 THEN Finance
   ELSE admin
END,
CASE zone
   WHEN 7 THEN Americas
   ELSE Asia-Pac
END
FROM emp_details
```

desc function extended get_json_object;
get_json_object(json_txt, path) - Extract a json object from path
Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid.
A limited version of JSONPath supported:
$ : Root object
. : Child operator
[] : Subscript operator for array
* : Wildcard for []
Syntax not supported that's worth noticing:

'' : Zero length string as key
.. : Recursive descent
@ : Current object/element
() : Script expression
?() : Filter (script) expression.
[,] : Union operator
[start:end:step] : array slice operator

desc function extended json_tuple;
json_tuple(jsonStr, p1, p2, ..., pn) - like get_json_object, but it takes multiple names and return a tuple. All the input parameters and output column types are string.

# 时间函数

timestamp 为 yyyy-MM-dd HH:mm:ss.SSS 格式的字符串
datetime 为 yyyy-MM-dd HH:mm:ss 格式的字符串
date 为 yyyy-MM-dd 格式的字符串

select current_date from dual;
2017-04-08
select current_timestamp from dual;
2017-04-08 10:10:25.783

from_unixtime(bigint unixtime[, string format]) return: string
例：
select from_unixtime(1323308943,'yyyyMMdd') from dual;
20111208
select from_unixtime(1323308943) from dual;
2011-12-08 09:49:03

unix_timestamp([string date[,string pattern]]) return:bigint
说明:
不带任何参数，则返回当前时间戳
不带参数 pattern，则"yyyy-MM-dd HH:mm:ss"格式的字符串到 UNIX 时间戳。
如果转化失败，则返回 0。
例:
select unix_timestamp() from dual;
1491615665
select unix_timestamp('2011-12-07 13:01:03') from dual;
1323234063
select unix_timestamp('20111207 13:01:03','yyyyMMdd HH:mm:ss') from dual;
1323234063

to_date(string/date/datetime timestamp) return:string
说明: 返回日期时间字段中的日期部分。
例：
select to_date('2011-12-08 10:03:01') from dual;
2011-12-08

year(string date):int
例：
select year('2011-12-08 10:03:01') from dual;
2011
select year('2012-12-08') from dual;
2012
select month('2011-12-08 10:03:01') from dual;
12
select month('2011-08-08') from dual;
8
select day('2011-12-08 10:03:01') from dual;
8
select day('2011-12-24') from dual;
24
select hour('2011-12-08 10:03:01') from dual;
10
select minute('2011-12-08 10:03:01') from dual;
3
select second('2011-12-08 10:03:01') from dual;
1

说明: 返回日期在当年的周数。
select weekofyear('2011-12-08 10:03:01') from dual;
49

datediff(string enddate, string startdate) : int 天数
例：
select datediff('2012-12-08','2012-05-09') from dual;

date_add(string startdate, int days) : string
例：
select date_add('2012-12-08',10) from dual;
2012-12-18

date_sub (string startdate, int days) : string
例：
select date_sub('2012-12-08',10) from dual;
2012-11-28

## 8.1 流程控制

```
select id,
   case id
      when 1 then 'a'
      when 2 then 'b'
      when 3 then 'c'
      when 4 then 'd'
      else 'other'
   end
from t1;
```

## 8.2 单词计数

```
select wc.word,count(1) count from (
   select explode(split(line,' ')) word from dula
) wc
group by wc.word
order by count desc;
```

## 8.3 行转列

```
select
   id,
   name,
   concat_ws(",",collect_set(address)) addr
from address group by id,name;
```

## 8.4 列转行与 lateral view

```
select id, name, tmpTable.address addr
from addr2
lateral view
   explode(split(addr,",")) tmpTable
as address;
```

lateral view 会产生一个支持别名表的虚拟表。

## 8.5 分组 topN

```
create table emp
as
select e.id empid,e.deptid,s.salary from t_employee e
join t_salary s on e.id=s.empid;
```

```
select e.empid id,e.deptid deptid,e.salary salary from(
   select *,
      row_number() over(partition by deptid order by salary) rank
   from emp) e
where e.rank<=3;
```

## 8.6json 函数

```
select * from json;
```

| json.line |
| --- |
| {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"} |
| {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"} |
| {"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"} |
| {"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"} |

```
select a.* from json lateral view json_tuple(line,'movie','rate','timeStamp','uid') a as movie,rate,time,uid;
```

| a.movie | a.rate | a.time | a.uid |
| --- | --- | --- | --- |
| 1193 | 5 | 978300760 | 1 |
| 661 | 3 | 978302109 | 1 |
| 914 | 3 | 978301968 | 1 |
| 3408 | 4 | 978300275 | 1 |

```
select
    movie,
    rate,
    from_unixtime(cast(time as bigint), 'yyyy-MM-dd HH:mm:ss') time,
    uid
from json
lateral view json_tuple(line,'movie','rate','timeStamp','uid') a
as movie,rate,time,uid;
```

```
select
    json_tuple(line,'movie','rate','timeStamp','uid') as (movie,rate,time,uid)
from dula;
```

| movie | rate | time | uid |
|---|---|---|---|
| 1193 | 5 | 2001-01-01 06:12:40 | 1 |
| 661 | 3 | 2001-01-01 06:35:09 | 1 |
| 914 | 3 | 2001-01-01 06:32:48 | 1 |
| 3408 | 4 | 2001-01-01 06:04:35 | 1 |

```
select
   get_json_object(line,'$.movie') movie,
   get_json_object(line,'$.rate') rate,
   from_unixtime(cast(get_json_object(line,'$.timeStamp') as bigint), 'yyyy-MM-dd HH:mm:ss')
time,
   get_json_object(line,'$.uid') uid
from json;
```

| movie | rate | time | uid |
|---|---|---|---|
| 1193 | 5 | 2001-01-01 06:12:40 | 1 |
| 661 | 3 | 2001-01-01 06:35:09 | 1 |
| 914 | 3 | 2001-01-01 06:32:48 | 1 |
| 3408 | 4 | 2001-01-01 06:04:35 | 1 |

## 8.7 窗口函数 over partition by

例一：sum 窗口

需求：

```
create table t_salary(username string,month string,salary int)
row format delimited fields terminated by ',';
load data local inpath '/root/t_salary.dat' into table t_salary;

A,2015-01,5
A,2015-01,15
B,2015-01,5
A,2015-01,8
B,2015-01,25
A,2015-01,5
A,2015-02,4
```

A,2015-02,6
B,2015-02,10
B,2015-02,5
A,2015-03,14
A,2015-03,16
B,2015-03,20
B,2015-03,5

需要得到如下统计结果：

| 用户 | 月份 | 月总额 | 累计总额 |
| --- | --- | --- | --- |
| A | 2015-01 | 33 | 33 |
| A | 2015-02 | 10 | 43 |
| A | 2015-03 | 30 | 73 |
| B | 2015-01 | 30 | 30 |
| B | 2015-02 | 15 | 45 |

.....

```
create view sumsal as
select
   username,
   month,
   sum(salary) salary
from t_salary
group by username,month;

create table salt as
select
   username,
   month,
   sum(salary) salary
from t_salary
group by username,month;

select
   username,
   month,
   salary,
   sum(salary) over(partition by username order by month) accumulativeSalary
from salt;

select
   username,
   month,
   salary,
```

```
   sum(salary) over(partition by username order by month) accumulativeSalary
from (
   select
      username,
      month,
      sum(salary) salary
   from t_salary
   group by username,month;
);
```

over（order by salary range between 5 preceding and 5 following）：窗口范围为当前行数据幅度减 5 加 5 后的范围内的。

```
create table t2(name string,class string,s int)
row format delimited fields terminated by ',';

adf,3,45
asdf,3,55
cfe,2,74
3dd,3,78
fda,1,80
gds,2,92
ffd,1,95
dss,1,95
ddd,3,99
gf,3,99

load data local inpath '/root/hive-demo/t2.dat' into table t2;
```

select * from t2

```
t2.name t2.class        t2.s
adf     3       45
asdf    3       55
cfe     2       74
3dd     3       78
fda     1       80
gds     2       92
ffd     1       95
dss     1       95
ddd     3       99
gf      3       99
```

select name,class,s, sum(s)over(order by s range between 2 preceding and 2 following) mm from t2;

| name | class | s | mm |
|---|---|---|---|
| adf | 3 | 45 | 45 |
| asdf | 3 | 55 | 55 |
| cfe | 2 | 74 | 74 |
| 3dd | 3 | 78 | 158 |
| fda | 1 | 80 | 158 |
| gds | 2 | 92 | 92 |
| dss | 1 | 95 | 190 |
| ffd | 1 | 95 | 190 |
| gf | 3 | 99 | 198 |
| ddd | 3 | 99 | 198 |

select name,class,s, sum(s)over(order by s rows between 2 preceding and 2 following) mm from t2;

| name | class | s | mm |
|---|---|---|---|
| adf | 3 | 45 | 174 |
| asdf | 3 | 55 | 252 |
| cfe | 2 | 74 | 332 |
| 3dd | 3 | 78 | 379 |
| fda | 1 | 80 | 419 |
| gds | 2 | 92 | 440 |
| dss | 1 | 95 | 461 |
| ffd | 1 | 95 | 480 |
| gf | 3 | 99 | 388 |
| ddd | 3 | 99 | 293 |

select name,class,s,rank()over(partition by class order by s desc) mm from t2;

| name | class | s | mm |
|---|---|---|---|
| dss | 1 | 95 | 1 |
| ffd | 1 | 95 | 1 |
| fda | 1 | 80 | 3 |
| gds | 2 | 92 | 1 |
| cfe | 2 | 74 | 2 |
| gf | 3 | 99 | 1 |
| ddd | 3 | 99 | 1 |
| 3dd | 3 | 78 | 3 |
| asdf | 3 | 55 | 4 |
| adf | 3 | 45 | 5 |

select name,class,s,dense_rank()over(partition by class order by s desc) mm from t2;

| name | class | s | mm |
|---|---|---|---|
| dss | 1 | 95 | 1 |
| ffd | 1 | 95 | 1 |
| fda | 1 | 80 | 2 |
| gds | 2 | 92 | 1 |
| cfe | 2 | 74 | 2 |
| gf | 3 | 99 | 1 |

```
ddd     3       99      1
3dd     3       78      2
asdf    3       55      3
adf     3       45      4
```

between 语法：

```
-- 默认为从起点到当前行
sum(pv) over(partition by cookieid order by createtime ASC)
--从起点到当前行
sum(pv) over(partition by cookieid order by createtime rows between unbounded preceding and current row)
--分组内所有行
sum(pv) over(partition by cookieid)
--当前行+往前 3 行
sum(pv) over(partition by cookieid order by createtime rows between 3 preceding and current row)
--当前行+往前 3 行+往后 1 行
sum(pv) over(partition by cookieid count
order by createtime rows between 3 preceding and 1 following)
---当前行+往后所有行
sum(pv) over(partition by cookieid order by createtime rows between current row and unbounded following)

rows between A and B
current row 当前行
unbounded preceding 起始行
3 preceding 前三行
1 following 后一行
unbounded following 末尾行
```



# hive 调优

## explain

explain select sum(salary) from t_salary;
首先，查询计划会打印出抽象的语法书。

它表明 Hive 是如何将查询解析成 token(符号)和 literal(字面值)的。

```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: t_salary
            Statistics: Num rows: 32 Data size: 129 Basic stats: COMPLETE Column stats: NONE
            Select Operator
              expressions: salary (type: float)
              outputColumnNames: salary
              Statistics: Num rows: 32 Data size: 129 Basic stats: COMPLETE Column stats: NONE
              Group By Operator
                aggregations: sum(salary)
                mode: hash
                outputColumnNames: _col0
                Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
                Reduce Output Operator
                  sort order:
                  Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
                  value expressions: _col0 (type: double)
      Reduce Operator Tree:
        Group By Operator
          aggregations: sum(VALUE._col0)
          mode: mergepartial
          outputColumnNames: _col0
          Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
          File Output Operator
            compressed: false
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
            table:
                input format: org.apache.hadoop.mapred.TextInputFormat
                output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        ListSink
```

explain extended select sum(salary) from t_salary;

```
Explain
ABSTRACT SYNTAX TREE:
```

```
TOK_QUERY
   TOK_FROM
      TOK_TABREF
         TOK_TABNAME
            t_salary
   TOK_INSERT
      TOK_DESTINATION
         TOK_DIR
            TOK_TMP_FILE
      TOK_SELECT
         TOK_SELEXPR
            TOK_FUNCTION
               sum
               TOK_TABLE_OR_COL
                  salary


STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: t_salary
            Statistics: Num rows: 32 Data size: 129 Basic stats: COMPLETE Column stats: NONE
            GatherStats: false
            Select Operator
              expressions: salary (type: float)
              outputColumnNames: salary
              Statistics: Num rows: 32 Data size: 129 Basic stats: COMPLETE Column stats: NONE
              Group By Operator
                aggregations: sum(salary)
                mode: hash
                outputColumnNames: _col0
                Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
                Reduce Output Operator
                  sort order:
                  Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
                  tag: -1
                  value expressions: _col0 (type: double)
```

```
                auto parallelism: false
        Path -> Alias:
          hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary [t_salary]
        Path -> Partition:
          hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary
            Partition
              base file name: t_salary
              input format: org.apache.hadoop.mapred.TextInputFormat
              output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
              properties:
                COLUMN_STATS_ACCURATE true
                bucket_count -1
                columns id,empid,salary
                columns.comments '??ID','??ID','????'
                columns.types int:int:float
                comment ?????
                field.delim
                file.inputformat org.apache.hadoop.mapred.TextInputFormat
                file.outputformat org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                location hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary
                name test.t_salary
                numFiles 1
                numRows 0
                rawDataSize 0
                serialization.ddl struct t_salary { i32 id, i32 empid, float salary}
                serialization.format
                serialization.lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                totalSize 129
                transient_lastDdlTime 1490706218
              serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

              input format: org.apache.hadoop.mapred.TextInputFormat
              output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
              properties:
                COLUMN_STATS_ACCURATE true
                bucket_count -1
                columns id,empid,salary
                columns.comments '??ID','??ID','????'
                columns.types int:int:float
                comment ?????
                field.delim
                file.inputformat org.apache.hadoop.mapred.TextInputFormat
                file.outputformat org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                location hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary
```

```
                    name test.t_salary
                    numFiles 1
                    numRows 0
                    rawDataSize 0
                    serialization.ddl struct t_salary { i32 id, i32 empid, float salary}
                    serialization.format
                    serialization.lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                    totalSize 129
                    transient_lastDdlTime 1490706218
                  serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                  name: test.t_salary
                name: test.t_salary
        Truncated Path -> Alias:
          /test.db/t_salary [t_salary]
        Needs Tagging: false
        Reduce Operator Tree:
          Group By Operator
            aggregations: sum(VALUE._col0)
            mode: mergepartial
            outputColumnNames: _col0
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
            File Output Operator
              compressed: false
              GlobalTableId: 0
              directory:
hdfs://mini1:9000/tmp/hive/root/a3799134-d5d0-49fc-a840-9a0c4dbd90c8
/hive_2017-04-03_17-29-49_558_6616053246494828033-1/-mr-10000
/.hive-staging_hive_2017-04-03_17-29-49_558_6616053246494828033-1/-ext-10001
              NumFilesPerFileSink: 1
              Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: NONE
              Stats Publishing Key Prefix:
hdfs://mini1:9000/tmp/hive/root/a3799134-d5d0-49fc-a840-9a0c4dbd90c8
/hive_2017-04-03_17-29-49_558_6616053246494828033-1/-mr-10000
/.hive-staging_hive_2017-04-03_17-29-49_558_6616053246494828033-1/-ext-10001/
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                  properties:
                    columns _col0
                    columns.types double
                    escape.delim \
                    hive.serialization.extend.additional.nesting.levels true
                    serialization.format 1
                    serialization.lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```

```
                    serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                TotalFiles: 1
                GatherStats: false
                MultiFileSpray: false

  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        ListSink
```

# Map 端聚合

Map 端部分聚合，相当于 Combiner
set hive.map.aggr=true;

hive.groupby.mapaggr.checkinterval：在 Map 端进行聚合操作的条目数目

hive.groupby.skewindata=true;
数据倾斜时负载均衡，当选项设定为 true，生成的查询计划会有两个 MR Job
第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果
这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；
第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

# 开启压缩

任务中间压缩
set hive.exec.compress.intermediate=true;
set hive.intermediate.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;（常用）
set hive.intermediate.compression.type=BLOCK;

```
<property>
    <name>hive.exec.compress.intermediate</name>
    <value>true</value>
</property>
<property>
```

<name>hive.intermediate.compression.codec</name>
<value>org.apache.hadoop.io.compress.GzipCodec<value/>
</property>
<property>
<name>hive.intermediate.compression.type</name>
<value>BLOCK<value/>
</property>
最终结果压缩（compress.type--针对 SequenceFile 而言）
set hive.exec.compress.output=true;
<property>
<name>hive.exec.compress.output</name>
<value>false</value>
</property>
老 api：
set mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
set mapred.output.compression.type=BLOCK;
hadoop 新 api：
set mapreduce.output.fileoutputformat.compress=true;
set mapreduce.output.fileoutputformat.compress.type=BLOCK;
set
mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.GzipCodec
;
必须设置如下参数，决定压缩是否生效
set hive.exec.compress.output=true;

操作例子：

```
set hive.exec.compress.output=true;
set mapreduce.output.fileoutputformat.compress.type=BLOCK;
set mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.GzipCodec;

create table seqt
stored as sequencefile
as select * from access;

insert overwrite table seqt select * from access;

dfs -ls /user/hive/warehouse/seqt ;
dfs -cat /user/hive/warehouse/seqt/000000_0;
dfs -text /user/hive/warehouse/codectest/000000_0.gz;
```

```
create table t1
    stored as orc
    as
```

select * from t3;

# 开启 jvm 重用

set mapreduce.job.jvm.numtasks=10;

```
<property>
  <name>mapreduce.job.jvm.numtasks</name>
  <value>1</value>
  <description>How many tasks to run per jvm. If set to -1, there is no limit. </description>
</property>
```

# 开启并行执行 hive job

set hive.exec.parallel=true;
set hive.exec.parallel.thread.number=8

# 关闭推测执行

set mapreduce.map.speculati=false;
set mapreduce.reduce.speculati=false;

**mapreduce.map.speculative**：如果为 true 则 Map Task 可以推测执行，即一个 Map Task 可以启动 Speculative Task 运行并行执行，该 Speculative Task 与原始 Task 同时处理同一份数据，谁先处理完，则将谁的结果作为最终结果。默认为 true。
**mapreduce.reduce.speculative**：同上，默认值为 true。
**mapreduce.job.speculative.speculative-cap-running-tasks**：能够推测重跑正在运行任务（单个 JOB）的百分之几，默认是：0.1
**mapreduce.job.speculative.speculative-cap-total-tasks**：能够推测重跑全部任务（单个 JOB）的百分之几，默认是：0.01
**mapreduce.job.speculative.minimum-allowed-tasks**：可以推测重新执行允许的最小任务数。默认是：10
首先，mapreduce.job.speculative.minimum-allowed-tasks 和 mapreduce.job.speculative.speculative-cap-total-tasks * 总任务数，取最大值。
然后，拿到上一步的值和 mapreduce.job.speculative.speculative-cap-running-tasks * 正在运行的任务数，取最大值，该值就是猜测执行的运行的任务数
**mapreduce.job.speculative.retry-after-no-speculate**：等待时间（毫秒）做下一轮的猜测，如

果没有任务，推测在这一轮。默认：1000（ms）
**mapreduce.job.speculative.retry-after-speculate**：等待时间（毫秒）做下一轮的猜测，如果有任务推测在这一轮。默认：15000（ms）
**mapreduce.job.speculative.slowtaskthreshold**：标准差，任务的平均进展率必须低于所有正在运行任务的平均值才会被认为是太慢的任务，默认值：1.0

当多个 GROUP BY 语句有相同的分组列，则会优化为一个 MR 任务
set hive.multigroupby.singlemar=true;

# 虚拟列

INPUT__FILE__NAME map 任务读入 File 的全路径
BLOCK__OFFSET__INSIDE__FILE
RCFile 或 SequenceFile 显示 Block file Offset(当前块在文件中的偏移量)
TextFile，显示当前行在文件中的偏移量
ROW__OFFSET__INSIDE__BLOCK
RCFile 和 SequenceFile 显示 row number
textfile 显示为 0
注：若要显示 ROW__OFFSET__INSIDE__BLOCK ，必须进行如下设置
set hive.exec.rowoffset=true;

例：
select INPUT__FILE__NAME, BLOCK__OFFSET__INSIDE__FILE,ROW__OFFSET__INSIDE__BLOCK,salary from t_salary;
显示如下结果：

| input__file__name | block__offset__inside__file | row__offset__inside__block | salary |
|---|---|---|---|
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 0 | 0 | 5500.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 11 | 0 | 4500.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 22 | 0 | 1900.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 33 | 0 | 4800.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 44 | 0 | 6500.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 55 | 0 | 14500.0 |
| hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt | 67 | 0 | |

44500.0
hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt 79 0
5000.0
hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt 91 0
2900.0
hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt 103 0
6500.0
hdfs://mini1:9000/user/hive/warehouse/test.db/t_salary/salary.txt 116 0
6000.0

# 本地模式

set hive.exec.mode.local.auto=true;

1.job 的输入数据大小必须小于参数：hive.exec.mode.local.auto.inputbytes.max(默认 128MB)
2.job 的 map 数必须小于参数：hive.exec.mode.local.auto.tasks.max(默认 4)
3.job 的 reduce 数必须为 0 或者 1
hive.exec.mode.local.auto.inputbytes.max=134217728
hive.exec.mode.local.auto.tasks.max=4
hive.exec.mode.local.auto=true;
hive.mapred.local.mem：本地模式启动的 JVM 内存大小

# 动态分区

开启动态分区支持
set hive.exec.dynamic.partition=true;
动态分区的模式，默认 strict，表示必须指定至少一个分区为静态分区，nonstrict 模式表示允许所有的分区字段都可以使用动态分区。
set hive.exec.dynamic.partition.mode=nonstrict;
允许所有的分区列都是动态分区列
set hive.exec.dynamic.partition.mode=nostrict;
在每个执行 MR 的节点上，最大可以创建多少个动态分区，如果超过了这个数量就会报错
hive.exec.max.dynamic.partitions.pernode （缺省值 100）：
在所有执行 MR 的节点上，最大一共可以创建多少个动态分区
hive.exec.max.dynamic.partitions （缺省值 1000）：
整个 MR Job 中，最大可以创建多少个 HDFS 文件
hive.exec.max.created.files （缺省值 100000）：

DATANODE：dfs.datanode.max.xceivers=8192：允许 DATANODE 打开多少个文件

关系型数据库（如 Oracle）中，对分区表 Insert 数据时候，数据库自动会根据分区字段的值，将数据插入到相应的分区中，Hive 中也提供了类似的机制，即动态分区(Dynamic Partition)，只不过，使用 Hive 的动态分区，需要进行相应的配置。

先看一个应用场景，源表 t_lxw1234 的数据如下：

```
SELECT day,url FROM t_lxw1234;
2015-05-10      url1
2015-05-10      url2
2015-06-14      url1
2015-06-14      url2
2015-06-15      url1
2015-06-15      url2
.....
```

目标表为：

```
CREATE TABLE t_lxw1234_partitioned (
url STRING
) PARTITIONED BY (month STRING,day STRING)
stored AS textfile;
```

需求：将 t_lxw1234 中的数据按照时间(day)，插入到目标表 t_lxw1234_partitioned 的相应分区中。

如果按照之前介绍的往指定一个分区中 Insert 数据，那么这个需求很不容易实现。

这时候就需要使用动态分区来实现，使用动态分区需要注意设定以下参数：

- **hive.exec.dynamic.partition**

默认值：false

是否开启动态分区功能，默认 false 关闭。

使用动态分区时候，该参数必须设置成 true;

- **hive.exec.dynamic.partition.mode**

默认值：strict

动态分区的模式，默认 strict，表示必须指定至少一个分区为静态分区，nonstrict 模式表示允许所有的分区字段都可以使用动态分区。

一般需要设置为 nonstrict

- **hive.exec.max.dynamic.partitions.pernode**

默认值：100

在每个执行 MR 的节点上，最大可以创建多少个动态分区。

该参数需要根据实际的数据来设定。

比如：源数据中包含了一年的数据，即 day 字段有 365 个值，那么该参数就需要设置成大于 365，如果使用默认值 100，则会报错。

- **hive.exec.max.dynamic.partitions**

默认值：1000

在所有执行 MR 的节点上，最大一共可以创建多少个动态分区。

同上参数解释。

- **hive.exec.max.created.files**

默认值：100000

整个 MR Job 中，最大可以创建多少个 HDFS 文件。

一般默认值足够了，除非你的数据量非常大，需要创建的文件数大于 100000，可根据实际情况加以调整。

- **hive.error.on.empty.partition**

默认值：false

当有空分区生成时，是否抛出异常。

一般不需要设置。

那么，上面的需求可以使用如下的语句来完成：

```
1.
2. SET hive.exec.dynamic.partition=true;
3. SET hive.exec.dynamic.partition.mode=nonstrict;
4. SET hive.exec.max.dynamic.partitions.pernode = 1000;
```

```
SET hive.exec.max.dynamic.partitions=1000;

INSERT overwrite TABLE t_lxw1234_partitioned PARTITION (month,day)

SELECT url,substr(day,1,7) AS month,day
FROM t_lxw1234;
```

注意：在 **PARTITION (month,day)**中指定分区字段名即可；

在 SELECT 子句的最后两个字段，必须对应前面 **PARTITION (month,day)**中指定的分区字段，包括顺序。

执行结果如下：

Loading data to table liuxiaowen.t_lxw1234_partitioned partition (month=null, day=null)
Loading partition {month=2015-05, day=2015-05-10}
Loading partition {month=2015-06, day=2015-06-14}
Loading partition {month=2015-06, day=2015-06-15}
Partition liuxiaowen.t_lxw1234_partitioned{month=2015-05, day=2015-05-10} stats: [numFiles=1, numRows=2, totalSize=10, rawDataSize=8]
Partition liuxiaowen.t_lxw1234_partitioned{month=2015-06, day=2015-06-14} stats: [numFiles=1, numRows=2, totalSize=10, rawDataSize=8]
Partition liuxiaowen.t_lxw1234_partitioned{month=2015-06, day=2015-06-15} stats: [numFiles=1, numRows=2, totalSize=10, rawDataSize=8]
查看目标表有哪些分区：
hive> show partitions t_lxw1234_partitioned;
OK
month=2015-05/day=2015-05-10
month=2015-06/day=2015-06-14
month=2015-06/day=2015-06-15

# 合并小文件

hive.merg.mapfiles=true：合并 map 输出
hive.merge.mapredfiles=false：合并 reduce 输出
hive.merge.size.per.task=256*1000*1000：合并文件的大小
hive.mergejob.maponly=true：如果支持 CombineHiveInputFormat 则生成只有 Map 的任务执行 merge
hive.merge.smallfiles.avgsize=16000000：文件的平均大小小于该值时，会启动一个 MR 任务执行 merge。

# 控制 Reduce Task 的数量

参数 mapred.reduce.tasks（默认为-1）

如果是负数，那么推测 reducer 任务数量。

参数 1 hive.exec.reducers.bytes.per.reducer

决定每个 reducer task 可以处理的最大数据量，默认是 256MB。

参数 2 hive.exec.reducers.max

决定了最大 reducers 任务数量，默认是 1009.

reducer 数量计算公式

N=min(reducers.max，总输入数据量/bytes.per.reducer)

依据 Hadoop 的经验，可以将

reducers.max 设定为 0.95*nodemanager 数量或 1.75*nodemanager 数量

0.95 倍 表示 所有的 reduce 会在 map 任务的输出传输结束后同时开始运行。

1.75 倍 表示 高速 node 会在完成第一批 reduce 任务计算之后开始计算第二批 reduce 任务。

什么情况下会有 reduce 数量只有一个呢？

1° 进行汇总操作的时候，没有使用 group by 分组

2° 使用了子句 order by 排序（全局排序，只能在单节点进行排序）

3° 笛卡尔积 M * N （表连接，但是没有 on 条件）