大数据技术之 Hive

第 1 章 Hive 基本概念

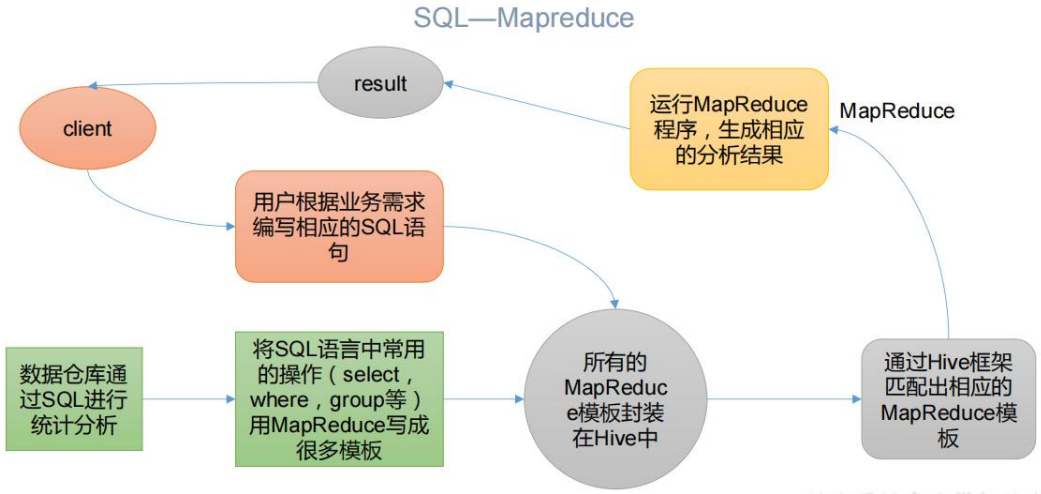
1.1 什么是 Hive

1）hive 简介

Hive：由 Facebook 开源用于解决海量结构化日志的数据统计工具。

Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张表，并提供类 SQL 查询功能。

2）Hive 本质：将 HQL 转化为 Mapreduce 程序



（1）Hive 处理的数据存储在 HDFS

（2）Hive 分析数据底层的实现是 MapReduce

（3）执行程序运行在 Yarn 上

1.2 Hive 的优缺点

1.2.1 优点

（1）操作接口采用类 SQL 语法，提供快速开发的能力（简单、容易上手）。

（2）避免了去写 MapReduce，减少开发人员的学习成本。

（3）Hive 的执行延迟比较高，因此 Hive 常用于数据分析，对实时性要求不高的场合。

（4）Hive 优势在于处理大数据，对于处理小数据没有优势，因为 Hive 的执行延迟比较高。

（5）Hive 支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。

1.2.2 缺点

1）Hive 的 HQL 表达能力有限

（1）迭代式算法无法表达

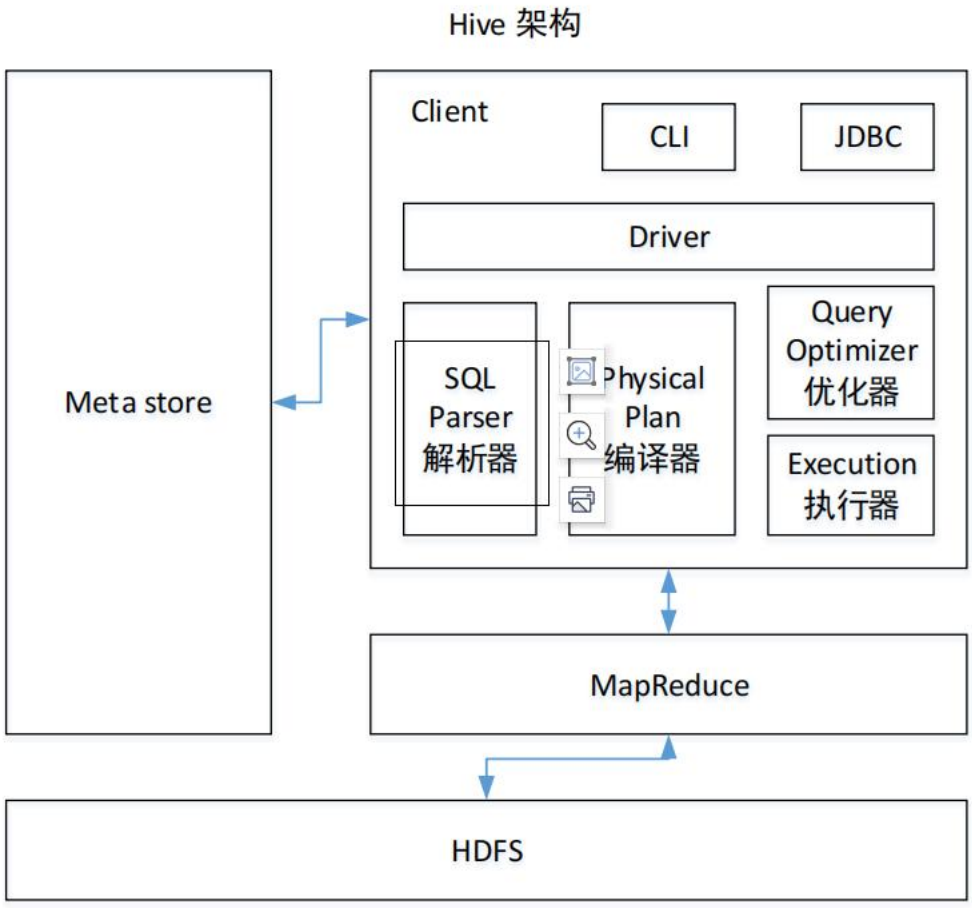
（2）数据挖掘方面不擅长，由于 MapReduce 数据处理流程的限制，效率更高的算法却无法实现。

2）Hive 的效率比较低

（1）Hive 自动生成的 MapReduce 作业，通常情况下不够智能化

（2）Hive 调优比较困难，粒度较粗

1.3 Hive 架构原理



1）用户接口：Client

CLI（command-line interface）、JDBC/ODBC(jdbc 访问 hive)、WEBUI（浏览器访问 hive）

2）元数据：Metastore

元数据包括：表名、表所属的数据库（默认是 default）、表的拥有者、列/分区字段、表的类型（是否是外部表）、表的数据所在目录等；

默认存储在自带的 derby 数据库中，推荐使用 MySQL 存储 Metastore

3）Hadoop

使用 HDFS 进行存储，使用 MapReduce 进行计算。

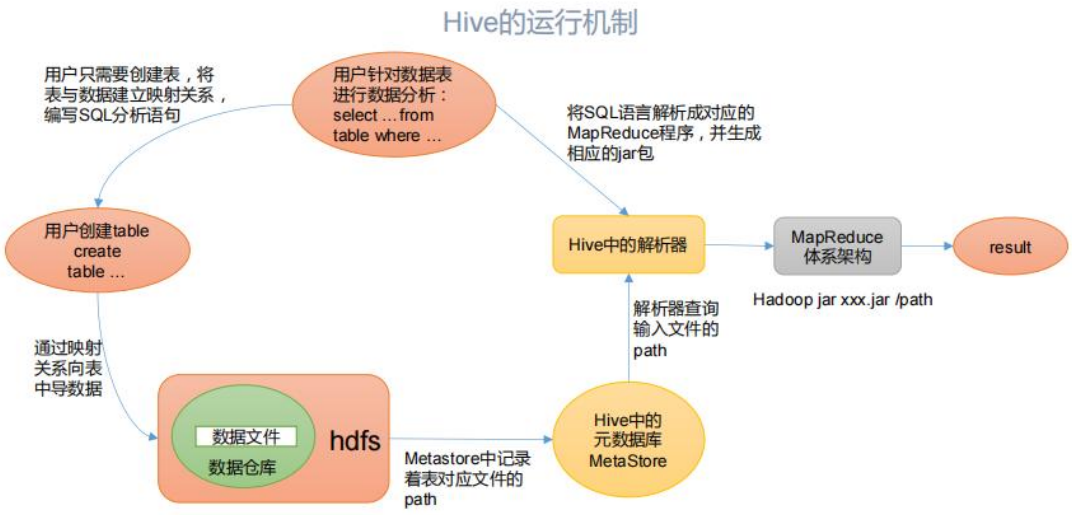
4）驱动器：Driver

（1）解析器（SQL Parser）：将 SQL 字符串转换成抽象语法树 AST，这一步一般都用第 三方工具库完成，比如 antlr；对 AST 进行语法分析，比如表是否存在、字段是否存在、SQL 语义是否有误。

（2）编译器（Physical Plan）：将 AST 编译生成逻辑执行计划。

（3）优化器（Query Optimizer）：对逻辑执行计划进行优化。

（4）执行器（Execution）：把逻辑执行计划转换成可以运行的物理计划。对于 Hive 来说，就是 MR/Spark。



Hive 通过给用户提供的一系列交互接口，接收到用户的指令(SQL)，使用自己的 Driver，结合元数据(MetaStore)，将这些指令翻译成 MapReduce，提交到 Hadoop 中执行，最后，将执行返回的结果输出到用户交互接口。

1.4 Hive 和数据库比较

由于 Hive 采用了类似 SQL 的查询语言 HQL(Hive Query Language)，因此很容易将 Hive 理解为数据库。其实从结构上来看，Hive 和数据库除了拥有类似的查询语言，再无类似之处。

本文将从多个方面来阐述 Hive 和数据库的差异。数据库可以用在 Online 的应用中，但是 Hive 是为数据仓库而设计的，清楚这一点，有助于从应用角度理解 Hive 的特性。

1.4.1 查询语言

由于 SQL 被广泛的应用在数据仓库中，因此，专门针对 Hive 的特性设计了类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。

1.4.2 数据更新

由于 Hive 是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive 中不建议对数据的改写，所有的数据都是在加载的时候确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用 INSERT INTO … VALUES 添加数据，使用 UPDATE … SET 修 改数据。

1.4.3 执行延迟

Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导 致 Hive 执行延迟高的因素是 MapReduce 框架。由于 MapReduce 本身具有较高的延迟，因此 在利用 MapReduce 执行 Hive 查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。 当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。

1.4.4 数据规模

由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

第 2 章 Hive 安装

2.1 Hive 的安装地址

1. Hive 官网地址：
   http://hive.apache.org/
2. 文档查看地址：
   https://cwiki.apache.org/confluence/display/Hive/GettingStarted
3. 下载地址：
   http://archive.apache.org/dist/hive/

4. github 地址

https://github.com/apache/hive

2.2 Hive 安装部署

1. Hive 安装以及配置

（1）把 apache-hive-1.2.1-bin.tar.gz 上传到 linux 的/usr/local/packages 目录下

（2）解压 apache-hive-1.2.1-bin.tar.gz 到/usr/local/soft/目录下面

```
[root@master packages]# tar -zxvf apache-hive-1.2.1-bin.tar.gz -C /usr/local/soft/
```

（3）修改 apache-hive-1.2.1-bin.tar.gz 的名称为 hive

```
[root@master soft]# mv apache-hive-1.2.1-bin/ hive
```

（4）修改/usr/local/soft/hive/conf 目录下的 hive-env.sh.template 名称为 hive-env.sh

```
[root@master conf]# mv hive-env.sh.template hive-env.sh
```

（5）配置 hive-env.sh 文件

（a）配置 HADOOP_HOME 路径

```
export HADOOP_HOME=/usr/local/soft/hadoop-2.7.6
```

（b）配置 HIVE_CONF_DIR 路径

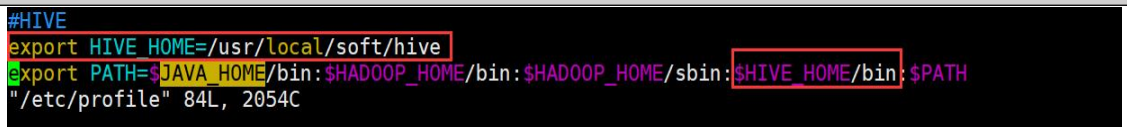
```
export HIVE_CONF_DIR=/usr/local/soft/hive/conf
```

（c）配置 JAVA_HOM 路径

```
export JAVA_HOME=/usr/local/soft/jdk1.8.0_171
```

（6）配置 hive 的环境变量

```
export HIVE_HOME=/usr/local/soft/hive
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$HIVE_HOME/bin:$PATH
```



2. Hive 基本启动

（1）启动 Hive

```
[root@master conf]# hive
```

```
[root@master conf]# hive

Logging initialized using configuration in jar:file:/usr/local/soft/hive/lib/hive-common-1.2.1.jar!/hive-log4j.properties
hive>
```

（2）查看数据库

```
hive> show databases;
```

```
hive> show databases;
OK
default
Time taken: 0.861 seconds, Fetched: 1 row(s)
hive>
```

（3）打开默认数据库

```
hive> use default;
```

（4）显示 default 数据库中的表

```
hive> show tables;
```

```
hive> show tables;
OK
Time taken: 0.027 seconds
hive>
```

（5）创建一张表

```
hive> create table student(id int,name string,gender string);
```

```
hive> create table student(id int,name string,gender string);
OK
Time taken: 0.416 seconds
hive> show tables;
OK
student
Time taken: 0.019 seconds, Fetched: 1 row(s)
hive>
```

（6）查看表的结构

```
hive> desc student;
```

```
hive> desc student;
OK
id                      int
name                    string
gender                  string
Time taken: 0.195 seconds, Fetched: 3 row(s)
hive>
```

（7）向表中插入数据

```
hive> insert into student values(2012003,"lyj","0");
```

```
hive> insert into student values(2012003,"lyj","0");
Query ID = root_20220328145448_cbac077e-4e85-4a4a-8db2-bc1a431553de
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1647919358644_0017, Tracking URL = http://master:8088/proxy/application_1647919358644_0017/
Kill Command = /usr/local/soft/hadoop-2.7.6/bin/hadoop job  -kill job_1647919358644_0017
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-03-28 14:54:55,049 Stage-1 map = 0%,  reduce = 0%
2022-03-28 14:55:02,366 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.47 sec
MapReduce Total cumulative CPU time: 2 seconds 470 msec
Ended Job = job_1647919358644_0017
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://master:9000/user/hive/warehouse/student/.hive-staging_hive_2022-03-28_14-54-48_097_3131135033411083022-1/-ext-10000
Loading data to table default.student
Table default.student stats: [numFiles=1, numRows=1, totalSize=14, rawDataSize=13]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 2.47 sec   HDFS Read: 3739 HDFS Write: 85 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 470 msec
OK
Time taken: 15.476 seconds
```

（8）查询表中数据

```
hive> select * from student;
```

```
hive> select * from student;
OK
2012003 lyj     0
Time taken: 0.035 seconds, Fetched: 1 row(s)
hive>
```

（9）退出 hive

```
hive> quit;
```

```
hive> quit;
[root@master conf]#
```

说明：（查看 hive 在 hdfs 中的结构）
数据库：在 hdfs 中表现为${hive.metastore.warehouse.dir}目录下一个文件夹
表：在 hdfs 中表现所属 db 目录下一个文件夹，文件夹中存放该表中的具体数据

2.3 将本地文件导入 Hive 案例

将本地/usr/local/soft/data/student.txt 这个目录下的数据导入到 hive 的 student(id int, name string,gender string)表中。

1. 数据准备

在/usr/local/soft/data/这个目录下准备数据

（1）在/usr/local/soft/目录下创建 data

```
[root@master soft]# mkdir data
```

（2）在/usr/local/soft/data 目录下创建 student.txt 文件并添加数据

```
[root@master data]# touch student.txt
[root@master data]# vi student.txt
2012003,lyj,0
2121001,lzh,1
2121002,wkh,1
2121003,jy ,0
2121004,syx,0
```

注意：以 ″,″ 键间隔。

2. Hive 实际操作

（1）启动 hive

```
[root@master data]# hive
```

（2）显示数据库

```
hive> show databases;
```

（3）使用 default 数据库

```
hive> use default;
```

（4）显示 default 数据库中的表

```
hive> show tables;
```

（5）删除已创建的 student 表

```
hive> drop table student;
```

注意：没有表就忽略。

（6）创建 student 表, 并声明文件分隔符’，’

```
create table student(
id int,
name string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

（7）加载/usr/local/soft/data/student.txt 文件到 student 数据库表中。

```
hive> load data local inpath '/usr/local/soft/data/student.txt' into
table student;
```

```
hive> load data local inpath '/usr/local/soft/data/student.txt' into table student;
Loading data to table default.student
Table default.student stats: [numFiles=2, numRows=0, totalSize=560, rawDataSize=0]
OK
Time taken: 1.046 seconds
```

（8）Hive 查询结果

```
hive> select * from student limit 10;
OK
2012003 lyj
2121001 lzh
2121002 wkh
2121003 jy
2121004 syx
2121005 ntt
2121006 gyl
2121007 hcr
2121008 xhy
```

3. 遇到的问题

再打开一个客户端窗口启动 hive，会产生 java.sql.SQLException 异常。

```
[root@master data]# hive
Logging initialized using configuration in
jar:file:/usr/local/soft/hive/lib/hive-common-1.2.1.jar!/hive-log4j.p
ropertie
Exception in thread "main" java.lang.RuntimeException:
java.lang.RuntimeException: Unable to instantiate
org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient
at
org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.jav
a:522)
at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:677)
at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:621)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.jav
```

```
a:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
Caused by: java.lang.RuntimeException: Unable to instantiate
org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient
at
org.apache.hadoop.hive.metastore.MetaStoreUtils.newInstance(MetaStore
Utils.java:
1523)
at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.<init>
(RetryingMetaStoreClient.java:86)
at
org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(Ret
ryingMetaSt
oreClient.java:132)
at
org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(Ret
ryingMetaSt
oreClient.java:104)
at
org.apache.hadoop.hive.ql.metadata.Hive.createMetaStoreClient(Hive.ja
va:3005)
at org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:3024)
at
org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.jav
a:503)
... 8 more
```

原因是，Metastore 默认存储在自带的 derby 数据库中，推荐使用 MySQL 存储 Metastore;

2.4 MySql 安装

1.安装 mysql5.7

（1）下载 yum Repository

```
[root@master packages]# wget -i -c
http://dev.mysql.com/get/mysql57-community-release-el7-10.noarch.rpm
```

（2）安装 yum Repository

```
[root@master packages]# yum -y install mysql57-community-release-el7-10.noarch.rpm
```

```
已安装:
  mysql57-community-release.noarch 0:el7-10

完毕!
[root@master packages]#
```

（3）安装 mysql5.7

```
[root@master packages]# yum -y install mysql-community-server --nogpgcheck
```

```
已安装:
  mysql-community-libs.x86_64 0:5.7.37-1.el7   mysql-community-libs-compat.x86_64 0:5.7.37-1.el7   mysql-community-server.x86_64 0:5.7.37-1.el7

作为依赖被安装:
  mysql-community-client.x86_64 0:5.7.37-1.el7                          mysql-community-common.x86_64 0:5.7.37-1.el7

替代:
  mariadb-libs.x86_64 1:5.5.68-1.el7

完毕!
[root@master packages]#
```

（4）开机自启动

```
[root@master packages]# systemctl enable mysqld.service
```

（5）启动 mysql

```
[root@master packages]# systemctl start mysqld.service
```

（6）查看状态

```
[root@master packages]# systemctl status mysqld.service
```

```
[root@master packages]# systemctl status mysqld.service
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since 一 2022-03-28 15:57:19 CST; 8s ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
  Process: 52733 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid $MYSQLD_OPTS (code=exited, status=0/SUCCESS)
  Process: 52679 ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
 Main PID: 52736 (mysqld)
    Tasks: 27
   CGroup: /system.slice/mysqld.service
           └─52736 /usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid

3月 28 15:57:12 master systemd[1]: Starting MySQL Server...
3月 28 15:57:19 master systemd[1]: Started MySQL Server.
```

（7）获取临时密码

```
[root@master packages]#grep "password" /var/log/mysqld.log
```

```
[root@master packages]# grep "password" /var/log/mysqld.log
2022-03-28T07:57:16.636998Z 1 [Note] A temporary password is generated for root@localhost: yp9eciEs:Jll
```

（8）登录 mysql

```
[root@master packages]#mysql -uroot -p
```

```
[root@master ~]# grep "password" /var/log/mysqld.log
2022-03-28T07:57:16.636998Z 1 [Note] A temporary password is generated for root@localhost: yp9eciEs:Jll
[root@master ~]# mysql -uroot -pyp9eciEs:Jll
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.37

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

（9）关闭密码复杂验证

```
set global validate_password_policy=0;
set global validate_password_length=1;
```

```
mysql> set global validate_password_policy=0;
Query OK, 0 rows affected (0.00 sec)

mysql> set global validate_password_length=1;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

（10）设置密码

```
alter user user() identified by "123456";
```

（11）修改权限

```
use mysql;

GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH
GRANT OPTION; --修改权限

flush privileges; --刷新权限

select host,user,authentication_string from user; --查看权限
```

```
mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> select host,user,authentication_string from user;
+-----------+---------------+-------------------------------------------+
| host      | user          | authentication_string                     |
+-----------+---------------+-------------------------------------------+
| localhost | root          | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
| localhost | mysql.session | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
| localhost | mysql.sys     | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
| %         | root          | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------+---------------+-------------------------------------------+
4 rows in set (0.00 sec)

mysql>
```

（12）卸载 yum Repository

```
#因为安装了 Yum Repository，以后每次 yum 操作都会自动更新，需要把这个卸载掉：
yum -y remove mysql57-community-release-el7-10.noarch
```

2. 卸载 mysql

```
删除依赖包
rpm -qa |grep -i mysql
yum remove mysql-community mysql-community-server mysql-community-libs
mysql-
community-common
清理文件
find / -name mysql
```

3.修改 MySQL 编码

(1) 修改 mysql 编码为 UTF-8

```
1、编辑配置文件
vim /etc/my.cnf
2、加入以下内容：
[client]
default-character-set = utf8mb4
[mysqld]
character-set-server = utf8mb4
collation-server = utf8mb4_general_ci
3、重启 mysql
systemctl restart mysqld
4、登录 mysql
mysql -uroot -p123456
5、查看 mysql 当前字符集
show variables like '%char%';
```

```
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html

[client]
default-character-set = utf8mb4

[mysqld]
character-set-server = utf8mb4
collation-server = utf8mb4_general_ci
```

```
mysql> show variables like '%char%';
+--------------------------------------+----------------------------+
| Variable_name                        | Value                      |
+--------------------------------------+----------------------------+
| character_set_client                 | utf8mb4                    |
| character_set_connection             | utf8mb4                    |
| character_set_database               | utf8mb4                    |
| character_set_filesystem             | binary                     |
| character_set_results                | utf8mb4                    |
| character_set_server                 | utf8mb4                    |
| character_set_system                 | utf8                       |
| character_sets_dir                   | /usr/share/mysql/charsets/ |
| validate_password_special_char_count | 1                          |
+--------------------------------------+----------------------------+
9 rows in set (0.00 sec)

mysql>
```

2.5 Hive 元数据配置到 MySql

2.5.1 驱动拷贝

1.上传 mysql-connector-java-5.1.27.tar.gz 驱动包至/usr/local/packages

```
[root@master packages]# ll
总用量 91696
-rw-r--r--. 1 root root 92834839 3月  28 11:33 apache-hive-1.2.1-bin.tar.gz
-rw-r--r--. 1 root root    21056 3月  28 16:29 derby.log
drwxr-xr-x. 5 root root      133 3月  28 16:29 metastore_db
-rw-r--r--. 1 root root    25548 4月   7 2017 mysql57-community-release-el7-10.noarch.rpm
-rw-r--r--. 1 root root  1006904 3月  28 22:59 mysql-connector-java-5.1.49.jar
[root@master packages]#
```

2.拷贝/usr/local/packages 目录下的 mysql-connector-java-5.1.49.jar 到 /usr/local/soft/hive/lib/

```
[root@master packages]# cp mysql-connector-java-5.1.49.jar
/usr/local/soft/hive/lib/
```

2.5.2 配置 Metastore 到 MySql

1. 在/usr/local/soft/hive/conf 目录下创建一个名为 hive-site.xml 的文件并配置

```
[root@master conf]# vim hive-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
```

```
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master:3306/hive?useSSL=false</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>123456</value>
<description>password to use against metastore database</description>
</property>
</configuration>
```

2. 将 hive 的 jline-2.12.jar 拷贝到 hadoop 对应目录下，hive 的 jline-2.12.jar 位置在 ：/usr/local/soft/hive/lib/jline-2.12.jar

```
[root@master conf]# cd /usr/local/soft/hive/lib/
[root@master lib]# cp jline-2.12.jar
/usr/local/soft/hadoop-2.7.6/share/hadoop/yarn/lib/
```

3. 配置完毕后，如果启动 hive 异常，可以重新启动虚拟机。（重启后，别忘了启 动 hadoop 集群）

2.5.3 多窗口启动 Hive 测试

1. 先启动 MySQL

```
[root@master conf]# mysql -uroot -p123456
```

查看有几个数据库

```
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
```

```
| mysql |
| performance_schema |
| sys |
+--------------------+
4 rows in set (0.00 sec)
mysql>
```

2. 再次打开多个窗口，分别启动 hive

```
[root@master conf]# hive
```

3. 启动 hive 后，回到 MySQL 窗口查看数据库，显示增加了 hive 数据库

```
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| hive |
| mysql |
| performance_schema |
| sys |
+--------------------+
5 rows in set (0.00 sec)
mysql>
```

4.修改 mysql 元数据库 hive，让其 hive 支持 utf-8 编码以支持中文

（1）切换到 hive 数据库

```
use hive;
```

（2）修改字段注释字符集

```
alter table COLUMNS_V2 modify column COMMENT varchar(256) character set
utf8;
```

（3）修改表注释字符集

```
alter table TABLE_PARAMS modify column PARAM_VALUE varchar(4000)
character set utf8;
```

（4）修改分区表参数，以支持分区键能够用中文表示

```
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;
```

（5）修改索引注解(可选)

```
alter table INDEX_PARAMS modify column PARAM_VALUE varchar(4000)
character set utf8;
```

2.6 HiveJDBC 访问

2.6.1 启动 hiveserver2 服务

```
[root@master conf]# hiveserver2
```

2.6.2 启动 beeline

```
[root@master conf]# beeline
Beeline version 1.2.1 by Apache Hive
beeline>
```

2.6.3 连接 hiveserver2

```
[root@master conf]# beeline
Beeline version 1.2.1 by Apache Hive
beeline> !connect jdbc:hive2://master:10000
Connecting to jdbc:hive2://master:10000
Enter username for jdbc:hive2://master:10000: root
Enter password for jdbc:hive2://master:10000:
Connected to: Apache Hive (version 1.2.1)
Driver: Hive JDBC (version 1.2.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://master:10000>
```

```
[root@master conf]# beeline
Beeline version 1.2.1 by Apache Hive
beeline> !connect jdbc:hive2://master:10000
Connecting to jdbc:hive2://master:10000
Enter username for jdbc:hive2://master:10000: root
Enter password for jdbc:hive2://master:10000:
Connected to: Apache Hive (version 1.2.1)
Driver: Hive JDBC (version 1.2.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://master:10000>
```

2.7 Hive 常用交互命令

```
[root@master conf]# hive -help
usage: hive
-d,--define <key=value> Variable subsitution to apply to hive
commands. e.g. -d A=B or --define A=B
    --database <databasename> Specify the database to use
-e <quoted-query-string> SQL from command line
-f <filename> SQL from files
-H,--help Print help information
    --hiveconf <property=value> Use value for given property
    --hivevar <key=value> Variable subsitution to apply to hive
-i <filename> Initialization SQL file
-S,--silent Silent mode in interactive shell
-v,--verbose Verbose mode (echo executed SQL to the
console)
[root@master conf]#
```

1. “-e”不进入 hive 的交互窗口执行 sql 语句

```
[root@master conf]# hive -e "select id from student;"
```

2. “-f”执行脚本中 sql 语句

（1）在/usr/local/soft/data 目录下创建 hivef.sql 文件

```
[root@master data]# touch hivef.sql
文件中写入正确的 sql 语句
[root@master data]# vi hivef.sql
select * from student;
```

（2）执行文件中的 sql 语句并将结果写入文件中

```
[root@master data]# hive -f
hivef.sql >/usr/local/soft/data/hive_result.txt
```

2.8 Hive 其他命令操作

1. 退出 hive 窗口：

```
hive> quit;
hive> exit;
```

在新版的 hive 中没区别了,在以前的版本是有的：exit:先隐性提交数据，再退出； quit:不提交数据，退出；

2. 在 hive cli 命令窗口中如何查看 hdfs 文件系统

```
hive> dfs -ls /;
```

3. 在 hive cli 命令窗口中如何查看本地文件系统

```
hive> ! ls /usr/local/soft/data;
```

4. 查看在 hive 中输入的所有历史命令

```
[root@master ~]# cat .hivehistory
```

2.9 Hive 常见属性配置

2.9.1 Hive 数据仓库位置配置

1)Default 数据仓库的最原始位置是在 hdfs 上的:/user/hive/warehouse 路径下。

2）在仓库目录下，没有对默认的数据库 default 创建文件夹。如果某张表属于 default 数 据库，直接

在数据仓库目录下创建一个文件夹。

3）修改 default 数据仓库原始位置（将 hive-default.xml.template 如下配置信息拷贝到 hive-site.xml 文件中）

```
<property>
<name>hive.metastore.warehouse.dir</name>
<value>/user/hive/warehouse</value>
<description>location of default database for the warehouse</description>
</property>
```

配置同组用户有执行权限

```
[root@master conf]# hadoop fs -chmod 777 /user/hive/warehouse
```

2.9.2 查询后信息显示配置

1）在 hive-site.xml 文件中添加如下配置信息，就可以实现显示当前数据库，以及查询表的头信息配置。

```
<property>
<name>hive.cli.print.header</name>
<value>true</value>
<description>Whether to print the names of the columns in query output.
</description>
</property>
<property>
<name>hive.cli.print.current.db</name>
<value>true</value>
<description>Whether to include the current database in the Hive prompt.
</description>
</property>
```

2）重新启动 hive，对比配置前后差异。

配置前

```
hive> select * from student limit 10;
OK
2012003 lyj
2121001 lzh
2121002 wkh
2121003 jy
2121004 syx
2121005 ntt
2121006 gyl
2121007 hcr
2121008 xhy
Time taken: 1.147 seconds, Fetched: 10 row(s)
```

配置后

```
hive (default)> select * from student limit 10;
OK
student.id student.name
2012003 lyj
2121001 lzh
2121002 wkh
2121003 jy
```

```
2121004 syx
2121005 ntt
2121006 gyl
2121007 hcr
2121008 xhy
Time taken: 0.371 seconds, Fetched: 10 row(s)
```

2.9.3 Hive 运行日志信息配置

1. Hive 的 log 默认存放在/tmp/root/hive.log 目录下（当前用户名下）
2. 修改 hive 的 log 存放日志到/usr/local/soft/hive/logs

（1）修改/usr/local/soft/hive/conf/hive-log4j.properties.template 文件名称为 hive-log4j.properties

```
[root@master conf]# mv hive-log4j.properties.template hive-log4j.properties
```

（2）在 hive-log4j.properties 文件中修改 log 存放位置

```
[root@master conf]# vim hive-log4j.properties
hive.log.dir=/usr/local/soft/hive/logs
```

2.9.4 参数配置方式

1. 查看当前所有的配置信息

```
hive (default)> set;
```

2. 参数的配置三种方式

（1）配置文件方式

默认配置文件：hive-default.xml

用户自定义配置文件：hive-site.xml

注意：用户自定义配置会覆盖默认配置。另外，Hive 也会读入 Hadoop 的配置，因为 Hive 是作为 Hadoop 的客户端启动的，Hive 的配置会覆盖 Hadoop 的配置。配置文件的设定对本机启动的所有 Hive 进程都有效。

（2）命令行参数方式

启动 Hive 时，可以在命令行添加-hiveconf param=value 来设定参数。

```
#例如
[root@master conf]# hive -hiveconf mapred.reduce.tasks=3;
#查看参数设置：
hive (default)> set mapred.reduce.tasks;
mapred.reduce.tasks=3
hive (default)>
```

注意：仅对本次 hive 启动有效

（3）参数声明方式

可以在 HQL 中使用 SET 关键字设定参数

```
hive (default)> set mapred.reduce.tasks=100;
#查看参数设置
hive (default)> set mapred.reduce.tasks;
mapred.reduce.tasks=100
hive (default)>
```

注意：仅对本次 hive 启动有效

上述三种设定方式的优先级依次递增。即配置文件<命令行参数<参数声明。注意某些系 统级的参数，例如 log4j 相关的设定，必须用前两种方式设定，因为那些参数的读取在会话 建立以前已经完成了。

第 3 章 Hive 数据类型

3.1 基本数据类型

Hive 数据类型	Java 数据类型	长度	例子
TINYINT	byte	1byte 有符号整数	20
SMALINT	short	2byte 有符号整数	20
INT	int	4byte 有符号整数	20
BIGINT	long	8byte 有符号整数	20
BOOLEAN	boolean	布尔类型，true 或者 false	TRUE FALSE
FLOAT	float	单精度浮点数	3.14159
DOUBLE	double	双精度浮点数	3.14159
STRING	string	字符系列。可以指定字符集。可以使用单引号或者双引号。	‘now is the time’ “for all good men”
TIMESTAMP		时间类型	
BINARY		字节数组	

对于 Hive 的 String 类型相当于数据库的 varchar 类型，该类型是一个可变的字符串，不过它不能声明其中最多能存储多少个字符，理论上它可以存储 2GB 的字符数。

3.2 集合数据类型

数据类型	描述	语法示例
STRUCT	和 c 语言中的 struct 类似，都可以通过“点”符号访问元素内容。例如，如果某个列的数据类型是 STRUCT{first STRING, last STRING},那么第 1 个元素可以通过字段.first 来引用。	struct() 例 如 struct<street:string, city:string>
MAP	MAP 是一组键-值对元组集合，使用数组表示法可以访问数据。例如，如果某个列的数据类型是 MAP，其中键->值对是’first’->’John’和’last’->’Doe’，那么可以通过字段名[‘last’]获取最后一个元素	map() 例如 map<string, int>
ARRAY	数组是一组具有相同类型和名称的变量的集合。这些变量称为数组的元素，每个数组元素都有一个编号，编号从零开始。例如，数组值为[‘John’，‘Doe’]，那么第 2 个元素可以通过数组名[1]进行引用。	Array() 例如 array<string>

Hive 有三种复杂数据类型 ARRAY、MAP 和 STRUCT。ARRAY 和 MAP 与 Java 中的 Array 和 Map 类似，而 STRUCT 与 C 语言中的 Struct 类似，它封装了一个命名字段集合，复杂数据类型允许任意层次的嵌套。

1）案例实操

（1）假设某表有如下一行，我们用 JSON 格式来表示其数据结构。在 Hive 下访问的格 式为

```
{
    "name": "songsong",
    "friends": ["bingbing", "lili"], //列表 Array,
    "children": { //键值 Map,
        "xiao song": 18,
        "xiaoxiao song": 19
    }
    "address": { //结构 Struct,
        "street": "hui long guan",
        "city": "beijing"
    }
}
```

（2）基于上述数据结构，我们在 Hive 里创建对应的表，并导入数据。

创建本地测试文件 test.txt

```
songsong,bingbing_lili,xiao song:18_xiaoxiao song:19,hui long guan_beijing
yangyang,caicai_susu,xiao yang:18_xiaoxiao yang:19,chao yang_beijing
```

注意：MAP，STRUCT 和 ARRAY 里的元素间关系都可以用同一个字符表示，这里用“_”。

（3）Hive 上创建测试表 test

```
create table test(
name string,
friends array < string > ,
children map < string, int > ,
address struct < street: string, city: string >
)
row format delimited fields terminated by ','
collection items terminated by '_'
map keys terminated by ':'
lines terminated by '\n';
```

字段解释：

row format delimited fields terminated by ',' -- 列分隔符

collection items terminated by '_' --MAP STRUCT 和 ARRAY 的分隔符(数 据分割符号)

map keys terminated by ':' -- MAP 中的 key 与 value 的分隔符

lines terminated by '\n'; -- 行分隔符

（4）导入文本数据到测试表

```
load data local inpath '/root/data/test.txt' into table test;
```

（5）访问三种集合列里的数据，以下分别是 ARRAY，MAP，STRUCT 的访问方式

```
hive (default)> select friends[1],children['xiao song'],address.city
from
test
where name="songsong";
OK
_c0 _c1 city
lili 18 beijing
Time taken: 0.076 seconds, Fetched: 1 row(s)
```

3.3 类型转化

Hive 的原子数据类型是可以进行隐式转换的，类似于 Java 的类型转换，例如某表达式使用 INT 类型，TINYINT 会自动转换为 INT 类型，但是 Hive 不会进行反向转化，例如，某表 达式使用 TINYINT 类型，INT 不会自动转换为 TINYINT 类型，它会返回错误，除非使用 CAST 操作。

1）隐式类型转换规则如下

（1）任何整数类型都可以隐式地转换为一个范围更广的类型，如 TINYINT 可以转换成 INT，INT 可以转换成 BIGINT。

（2）所有整数类型、FLOAT 和 STRING 类型都可以隐式地转换成 DOUBLE。

（3）TINYINT、SMALLINT、INT 都可以转换为 FLOAT。

（4）BOOLEAN 类型不可以转换为任何其它的类型。

2）可以使用 CAST 操作显示进行数据类型转换

例如 CAST('1' AS INT)将把字符串'1' 转换成整数 1；如果强制类型转换失败，如执行 CAST('X' AS INT)，表达式返回空值 NULL。

```
0: jdbc:hive2://hadoop102:10000> select '1'+2, cast('1'as int) + 2;
+------+------+--+
| _c0 | _c1 |
+------+------+--+
| 3.0 | 3 |
+------+------+--+
```

第 4 章 DDL 数据定义

4.1 创建数据库

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

1）创建一个数据库，数据库在 HDFS 上的默认存储路径是/user/hive/warehouse/*.db。

```
hive (default)> create database db_hive;
```

2）避免要创建的数据库已经存在错误，增加 if not exists 判断。（标准写法）

```
hive (default)> create database db_hive;
FAILED: Execution Error, return code 1 from
org.apache.hadoop.hive.ql.exec.DDLTask. Database db_hive already exists
hive (default)> create database if not exists db_hive;
```

3）创建一个数据库，指定数据库在 HDFS 上存放的位置

```
hive (default)> create database db_hive2 location '/db_hive2.db';
```

4.2 查询数据库

4.2.1 显示数据库

1）显示数据库

```
hive> show databases;
```

2）过滤显示查询的数据库

```
hive> show databases like 'db_hive*';
OK
db_hive
db_hive_1
```

4.2.2 查看数据库详情

1）显示数据库信息

```
hive> desc database db_hive;
OK
db_hive
hdfs://hadoop102:9820/user/hive/warehouse/db_hive.db
atguiguUSER
```

2）显示数据库详细信息，extended

```
hive> desc database extend
```

4.2.3 切换当前数据库

```
hive (default)> use db_hive;
```

4.3 修改数据库

用户可以使用 ALTER DATABASE 命令为某个数据库的 DBPROPERTIES 设置键-值对属性值，来描述这个数据库的属性信息。

```
hive (default)> alter database db_hive
set dbproperties('createtime'='20170830');
```

在 hive 中查看修改结果

```
hive> desc database extended db_hive;
db_name comment location owner_name owner_type parameters
db_hive hdfs://hadoop102:9820/user/hive/warehouse/db_hive.db
atguigu USER {createtime=20170830}
```

4.4 删除数据库

1）删除空数据库

```
hive>drop database db_hive2;
```

2）如果删除的数据库不存在，最好采用 if exists 判断数据库是否存在

```
hive> drop database db_hive;
FAILED: SemanticException [Error 10072]: Database does not exist: db_hive
hive> drop database if exists db_hive2;
```

3）如果数据库不为空，可以采用 cascade 命令，强制删除

```
hive> drop database db_hive;
FAILED: Execution Error, return code 1 from
org.apache.hadoop.hive.ql.exec.DDLTask.
InvalidOperationException(message:Database db_hive is not empty. One or
more tables exist.)
hive> drop database db_hive cascade;
```

4.5 创建表

1）建表语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name, col_name, ...)
[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement]
```

2）字段解释说明

（1）CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；

用户可以用 IF NOT EXISTS 选项来忽略这个异常。

（2）EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时可以指定一个指向实 际数据的路径（LOCATION），在删除表的时候，内部表的元数据和数据会被一起删除，而外 部表只删除元数据，不删除数据。

（3）COMMENT：为表和列添加注释。

（4）PARTITIONED BY 创建分区表

（5）CLUSTERED BY 创建分桶表

（6）SORTED BY 不常用，对桶中的一个或多个列另外排序

（7）ROW FORMAT

DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]
[MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
| SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value,
property_name=property_value, ...)]

用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表 的具体的列的数据。

SerDe 是 Serialize/Deserilize 的简称， hive 使用 Serde 进行行对象的序列与反序列化。

（8）STORED AS 指定存储文件类型

常用的存储文件类型：SEQUENCEFILE（二进制序列文件）、TEXTFILE（文本）、RCFILE（列 式存储格式文件） 如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCEFILE。

（9）LOCATION ：指定表在 HDFS 上的存储位置。

（10）AS：后跟查询语句，根据查询结果创建表。

（11）LIKE 允许用户复制现有的表结构，但是不复制数据。

4.5.1 管理表

1）理论

默认创建的表都是所谓的管理表，有时也被称为内部表。因为这种表，Hive 会（或多或 少地）控制着数据的生命周期。Hive 默认情况下会将这些表的数据存储在由配置项 hive.metastore.warehouse.dir(例如，/user/hive/warehouse)所定义的目录的子目录下。

当我们删除一个管理表时，Hive 也会删除这个表中数据。管理表不适合和其他工具共享数据。

2）案例实操

（0）原始数据

```
1001 ss1
1002 ss2
1003 ss3
1004 ss4
1005 ss5
1006 ss6
1007 ss7
1008 ss8
1009 ss9
1010 ss10
1011 ss11
1012 ss12
```

```
1013 ss13
1014 ss14
1015 ss15
1016 ss16
```

```
create table if not exists student(
id int, name string
)
row format delimited fields terminated by '\t'
stored as textfile
location '/user/hive/warehouse/student';
```

（1）普通创建表

（2）根据查询结果创建表（查询的结果会添加到新创建的表中）

```
create table if not exists student2 as select id, name from student;
```

（3）根据已经存在的表结构创建表

```
create table if not exists student3 like student;
```

4.5.2 外部表

1）理论

因为表是外部表，所以 Hive 并非认为其完全拥有这份数据。删除该表并不会删除掉这份数据，不过描述表的元数据信息会被删除掉。

2）管理表和外部表的使用场景

每天将收集到的网站日志定期流入 HDFS 文本文件。在外部表(原始日志表)的基础上 做大量的统计分析，用到的中间表、结果表使用内部表存储，数据通过 SELECT+INSERT 进入内部表。

3）案例实操

分别创建部门和员工外部表，并向表中导入数据。

（0）原始数据

dept:

```
10 ACCOUNTING 1700
20 RESEARCH 1800
30 SALES 1900
40 OPERATIONS 1700
```

emp：

```
7369 SMITH CLERK 7902 1980-12-17 800.00 20
7499 ALLEN SALESMAN 7698 1981-2-20 1600.00 300.00 30
7521 WARD SALESMAN 7698 1981-2-22 1250.00 500.00 30
7566 JONES MANAGER 7839 1981-4-2 2975.00 20
7654 MARTIN SALESMAN 7698 1981-9-28 1250.00 1400.00 30
7698 BLAKE MANAGER 7839 1981-5-1 2850.00 30
7782 CLARK MANAGER 7839 1981-6-9 2450.00 10
7788 SCOTT ANALYST 7566 1987-4-19 3000.00 20
7839 KING PRESIDENT 1981-11-17 5000.00 10
7844 TURNER SALESMAN 7698 1981-9-8 1500.00 0.00 30
7876 ADAMS CLERK 7788 1987-5-23 1100.00 20
7900 JAMES CLERK 7698 1981-12-3 950.00 30
7902 FORD ANALYST 7566 1981-12-3 3000.00 20
7934 MILLER CLERK 7782 1982-1-23 1300.00 10
```

（1）上传数据到 HDFS

```
hive (default)> dfs -mkdir /student;
hive (default)> dfs -put /usr/local/soft/datas/student.txt /student;
```

（2）建表语句，创建外部表

创建部门表

```
create external table if not exists dept(
deptno int,
dname string,
loc int
)
row format delimited fields terminated by '\t';
```

创建员工表

```
create external table if not exists emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
row format delimited fields terminated by '\t';
```

（3）查看创建的表

```
hive (default)>show tables;
```

（4）查看表格式化数据

```
hive (default)> desc formatted dept;
Table Type: EXTERNAL_TABLE
```

（5）删除外部表

```
hive (default)> drop table dept;
```

外部表删除后，hdfs 中的数据还在，但是 metadata 中 dept 的元数据已被删除

4.5.3 管理表与外部表的互相转换

（1）查询表的类型

```
hive (default)> desc formatted student2;
Table Type: MANAGED_TABLE
```

（2）修改内部表 student2 为外部表

```
alter table student2 set tblproperties('EXTERNAL'='TRUE');
```

（3）查询表的类型

```
hive (default)> desc formatted student2;
Table Type: EXTERNAL_TABLE
```

（4）修改外部表 student2 为内部表

```
alter table student2 set tblproperties('EXTERNAL'='FALSE');
```

（5）查询表的类型

```
hive (default)> desc formatted student2;
Table Type: MANAGED_TABLE
```

注意：('EXTERNAL'='TRUE')和('EXTERNAL'='FALSE')为固定写法，区分大小写！

4.6 修改表

4.6.1 重命名表

1）语法

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

2）实操案例

```
hive (default)> alter table dept_partition2 rename to dept_partition3;
```

4.6.2 增加、修改和删除表分区

见后面章节分区表基本操作

4.6.3 增加/修改/替换列信息

1）语法

（1）更新列

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

（2）增加和替换列

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

注：ADD 是代表新增一字段，字段位置在所有列后面(partition 列前)，REPLACE 则是表示替换表中所有字段。

2）实操案例

（1）查询表结构

```
hive> desc dept;
```

（2）添加列

```
hive (default)> alter table dept add columns(deptdesc string);
```

（3）查询表结构

```
hive> desc dept;
```

（4）更新列

```
hive (default)> alter table dept change column deptdesc desc string;
```

（5）查询表结构

```
hive> desc dept;
```

（6）替换列

```
hive (default)> alter table dept replace columns(deptno string, dname string, loc string);
```

（7）查询表结构

```
hive> desc dept;
```

4.7 删除表

```
hive (default)> drop table dept;
```

第 5 章 DML 数据操作

5.1 数据导入

5.1.1 向表中装载数据（Load）

1）语法

```
hive> load data [local] inpath '数据的 path' [overwrite] into table student [partition (partcol1=val1,…)];
```

（1）load data:表示加载数据

（2）local:表示从本地加载数据到 hive 表；否则从 HDFS 加载数据到 hive 表

（3）inpath:表示加载数据的路径

（4）overwrite:表示覆盖表中已有数据，否则表示追加

（5）into table:表示加载到哪张表

（6）student:表示具体的表

（7）partition:表示上传到指定分区

2）实操案例

（0）创建一张表

```
hive (default)> create table student(id string, name string) row format delimited fields terminated by '\t';
```

（1）加载本地文件到 hive

```
hive (default)> load data local inpath '/opt/module/hive/datas/student.txt' into table default.student;
```

（2）加载 HDFS 文件到 hive 中

上传文件到 HDFS

```
hive (default)> dfs -put /usr/local/soft/datastudent.txt /user/atguigu/hive;
```

加载 HDFS 上数据

```
hive (default)> load data inpath '/user/atguigu/hive/student.txt' into table default.student;
```

（3）加载数据覆盖表中已有的数据

上传文件到 HDFS

```
hive (default)> dfs -put /usr/local/soft/data/student.txt /user/atguigu/hive;
```

加载数据覆盖表中已有的数据

```
hive (default)> load data inpath '/user/atguigu/hive/student.txt' overwrite into table default.student;
```

5.1.2 通过查询语句向表中插入数据（Insert）

1）创建一张表

```
hive (default)> create table student_par(id int, name string) row format
delimited fields terminated by '\t';
```

2）基本插入数据

```
hive (default)> insert into table student_par
values(1,'wangwu'),(2,'zhaoliu');
```

3）基本模式插入（根据单张表查询结果）

```
hive (default)> insert overwrite table student_par
select id, name from student where month='201709';
```

insert into：以追加数据的方式插入到表或分区，原有数据不会删除
insert overwrite：会覆盖表中已存在的数据
注意：insert 不支持插入部分字段

4）多表（多分区）插入模式（根据多张表查询结果）

```
hive (default)> from student
insert overwrite table student partition(month='201707')
select id, name where month='201709'
insert overwrite table student partition(month='201706')
select id, name where month='201709';
```

5.1.3 查询语句中创建表并加载数据（As Select）

详见 4.5.1 章创建表。
根据查询结果创建表（查询的结果会添加到新创建的表中）

```
create table if not exists student3
as select id, name from student;
```

5.1.4 创建表时通过 Location 指定加载数据路径

1）上传数据到 hdfs 上

```
hive (default)> dfs -mkdir /student;
hive (default)> dfs -put /usr/local/soft/datas/student.txt /student;
```

2）创建表，并指定在 hdfs 上的位置

```
hive (default)> create external table if not exists student5(
id int, name string
)
row format delimited fields terminated by '\t'
location '/student;
```

3）查询数据

```
hive (default)> select * from student5;
```

5.1.5 Import 数据到指定 Hive 表中

注意：先用 export 导出后，再将数据导入。

```
hive (default)> import table student2
from '/user/hive/warehouse/export/student';
```

5.2 数据导出

5.2.1 Insert 导出

```
hive (default)> insert overwrite local directory
'/usr/local/soft/dataexport/student'
select * from student;
```

1）将查询的结果导出到本地

2）将查询的结果格式化导出到本地

```
hive(default)>insert overwrite local directory
'/usr/local/soft/dataexport/student1'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
select * from student;
```

3）将查询的结果导出到 HDFS 上(没有 local)

```
hive (default)> insert overwrite directory '/user/atguigu/student2'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
select * from student;
```

5.2.2 Hadoop 命令导出到本地

```
hive (default)> dfs -get /user/hive/warehouse/student/student.txt
/usr/local/soft/data/export/student3.txt;
```

5.2.3 Hive Shell 命令导出

基本语法：（hive -f/-e 执行语句或者脚本 > file）

```
[root@master hive]$ bin/hive -e 'select * from default.student;' >
/usr/local/soft/dataexport/student4.txt;
```

5.2.4 Export 导出到 HDFS 上

```
(defahiveult)> export table default.student
to '/user/hive/warehouse/export/student';
```

export 和 import 主要用于两个 Hadoop 平台集群之间 Hive 表迁移。

5.2.5 Sqoop 导出

后续课程专门讲。

5.2.6 清除表中数据（Truncate）

注意：Truncate 只能删除管理表，不能删除外部表中数据

```
hive (default)> truncate table student;
```

第 6 章 查询

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

查询语句语法：

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY col_list]
[CLUSTER BY col_list
| [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number]
```

6.1 基本查询（Select…From）

6.1.1 全表和特定列查询

0）数据准备

（0）原始数据

dept:

```
10 ACCOUNTING 1700
20 RESEARCH 1800
30 SALES 1900
40 OPERATIONS 1700
```

emp：

```
7369 SMITH CLERK 7902 1980-12-17 800.00 20
7499 ALLEN SALESMAN 7698 1981-2-20 1600.00 300.00 30
7521 WARD SALESMAN 7698 1981-2-22 1250.00 500.00 30
7566 JONES MANAGER 7839 1981-4-2 2975.00 20
7654 MARTIN SALESMAN 7698 1981-9-28 1250.00 1400.00 30
7698 BLAKE MANAGER 7839 1981-5-1 2850.00 30
7782 CLARK MANAGER 7839 1981-6-9 2450.00 10
7788 SCOTT ANALYST 7566 1987-4-19 3000.00 20
7839 KING PRESIDENT 1981-11-17 5000.00 10
7844 TURNER SALESMAN 7698 1981-9-8 1500.00 0.00 30
7876 ADAMS CLERK 7788 1987-5-23 1100.00 20
7900 JAMES CLERK 7698 1981-12-3 950.00 30
7902 FORD ANALYST 7566 1981-12-3 3000.00 20
```

```
7934 MILLER CLERK 7782 1982-1-23 1300.00 10
```

（1）创建部门表

```
create table if not exists dept(
deptno int,
dname string,
loc int
)
row format delimited fields terminated by '\t';
```

（2）创建员工表

```
create table if not exists emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
row format delimited fields terminated by '\t';
```

（3）导入数据

```
load data local inpath '/usr/local/soft/datas/dept.txt' into table
dept;
load data local inpath '/usr/local/soft/datas/emp.txt' into table emp;
```

1）全表查询

```
hive (default)> select * from emp;
hive (default)> select empno,ename,job,mgr,hiredate,sal,comm,deptno
from emp ;
```

2）选择特定列查询

```
hive (default)> select empno, ename from emp;
```

注意：

（1）SQL 语言大小写不敏感。

（2）SQL 可以写在一行或者多行

（3）关键字不能被缩写也不能分行

（4）各子句一般要分行写。
（5）使用缩进提高语句的可读性。

6.1.2 列别名

1）重命名一个列
2）便于计算
3）紧跟列名，也可以在列名和别名之间加入关键字‘AS’
4）案例实操

查询名称和部门

```
hive (default)> select ename AS name, deptno dn from emp;
```

6.1.3 算术运算符

运算符	描述
A+B	A 和 B 相加
A-B	A 减去 B
A*B	A 和 B 相乘
A/B	A 除以 B
A%B	A 对 B 取余
A&B	A 和 B 按位取与
A|B	A 和 B 按位取或
A^B	A 和 B 按位取异或
~A	A 按位取反

6.1.4 常用函数

1）求总行数（count）

```
hive (default)> select count(*) cnt from emp;
```

2）求工资的最大值（max）

```
hive (default)> select max(sal) max_sal from emp;
```

3）求工资的最小值（min）

```
hive (default)> select min(sal) min_sal from emp;
```

4）求工资的总和（sum）

```
hive (default)> select sum(sal) sum_sal from emp;
```

5）求工资的平均值（avg）

```
hive (default)> select avg(sal) avg_sal from emp;
```

6.1.5 Limit 语句

典型的查询会返回多行数据。LIMIT 子句用于限制返回的行数。

```
hive (default)> select * from emp limit 5;
hive (default)> select * from emp limit 2;
```

6.1.6 Where 语句

1）使用 WHERE 子句，将不满足条件的行过滤掉
2）WHERE 子句紧随 FROM 子句
3）案例实操

查询出薪水大于 1000 的所有员工

```
hive (default)> select * from emp where sal >1000;
```

注意：where 子句中不能使用字段别名

6.1.7 比较运算符（Between/In/ Is Null）

1）下面表中描述了谓词操作符，这些操作符同样可以用于 JOIN…ON 和 HAVING 语句中

操作符	支持的数据类型	描述
A=B	基本数据类型	如果 A 等于 B 则返回 TRUE，反之返回 FALSE
A<=>B	基本数据类型	如果 A 和 B 都为 NULL，则返回 TRUE，如果一边为 NULL，返回 False
A<>B, A!=B	基本数据类型	A 或者 B 为 NULL 则返回 NULL；如果 A 不等于 B，则返回 TRUE，反之返回 FALSE
A<B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 小于 B，则返回 TRUE，反之返回 FALSE
A<=B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 小于等于 B，则返回 TRUE，反之返回 FALSE
A>B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 大于 B，则返回 TRUE，反之返回 FALSE
A>=B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 大于等于 B，则返回 TRUE，反之返回 FALSE
A [NOT] BETWEEN B AND C	基本数据类型	如果 A，B 或者 C 任一为 NULL，则结果为 NULL。如果 A 的值大于等于 B 而且小于或等于 C，则结果为 TRUE，反之为 FALSE。如果使用 NOT 关键字则可达到相反的效果。
A IS NULL	所有数据类型	如果 A 等于 NULL，则返回 TRUE，反之返回 FALSE
A IS NOT NULL	所有数据类型	如果 A 不等于 NULL，则返回 TRUE，反之返回 FALSE
IN(数值 1, 数值 2)	所有数据类型	使用 IN 运算显示列表中的值
A [NOT] LIKE B	STRING 类型	B 是一个 SQL 下的简单正则表达式，也叫通配符模式，如果 A 与其匹配的话，则返回 TRUE；反之返回 FALSE。B 的表达式说明如下：‘x%’表示 A 必须以字母‘x’开头，‘%x’表示 A 必须以字母’x’结尾，而‘%x%’表示 A 包含有字母’x’,可以位于开头，结尾或者字符串中间。如果使用 NOT 关键字则可达到相反的效果。
A RLIKE B, A REGEXP B	STRING 类型	B 是基于 java 的正则表达式，如果 A 与其匹配，则返回 TRUE；反之返回 FALSE。匹配使用的是 JDK 中的正则表达式接口
		实现的，因为正则也依据其中的规则。例如，正则表达式必须和整个字符串 A 相匹配，而不是只需与其字符串匹配。

2）案例实操

（1）查询出薪水等于 5000 的所有员工

```
hive (default)> select * from emp where sal =5000;
```

（2）查询工资在 500 到 1000 的员工信息

```
hive (default)> select * from emp where sal between 500 and 1000;
```

（3）查询 comm 为空的所有员工信息

```
hive (default)> select * from emp where comm is null;
```

（4）查询工资是 1500 或 5000 的员工信息

```
hive (default)> select * from emp where sal IN (1500, 5000);
```

6.1.8 Like 和 RLike

1）使用 LIKE 运算选择类似的值

2）选择条件可以包含字符或数字:

% 代表零个或多个字符(任意个字符)。

_ 代表一个字符

3）RLIKE 子句

RLIKE 子句是 Hive 中这个功能的一个扩展，其可以通过 Java 的正则表达式这个更强大的语言来指定匹配条件

4）案例实操

（1）查找名字以 A 开头的员工信息

```
hive (default)> select * from emp where ename LIKE 'A%';
```

（2）查找名字中第二个字母为 A 的员工信息

```
hive (default)> select * from emp where ename LIKE '_A%';
```

（3）查找名字中带有 A 的员工信息

```
hive (default)> select * from emp where ename RLIKE '[A]';
```

6.1.9 逻辑运算符（And/Or/Not）

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

1）案例实操

（1）查询薪水大于 1000，部门是 30

```
hive (default)> select * from emp where sal>1000 and deptno=30;
```

（2）查询薪水大于 1000，或者部门是 30

```
hive (default)> select * from emp where sal>1000 or deptno=30;
```

（3）查询除了 20 部门和 30 部门以外的员工信息

```
hive (default)> select * from emp where deptno not IN(30, 20);
```

6.2 分组

6.2.1 Group By 语句

GROUP BY 语句通常会和聚合函数一起使用，按照一个或者多个列队结果进行分组，然后对每个组执行聚合操作。

1）案例实操：

（1）计算 emp 表每个部门的平均工资

```
hive (default)> select t.deptno, avg(t.sal) avg_sal from emp t group by t.deptno;
```

（2）计算 emp 每个部门中每个岗位的最高薪水

```
hive (default)> select t.deptno, t.job, max(t.sal) max_sal from emp t group by t.deptno, t.job;
```

6.2.2 Having 语句

1）having 与 where 不同点

（1）where 后面不能写分组函数，而 having 后面可以使用分组函数。

（2）having 只用于 group by 分组统计语句。

2）案例实操

（1）求每个部门的平均薪水大于 2000 的部门

求每个部门的平均工资

```
hive (default)> select deptno, avg(sal) from emp group by deptno;
```

求每个部门的平均薪水大于 2000 的部门

```
hive (default)> select deptno, avg(sal) avg_sal from emp group by deptno having avg_sal > 2000;
```

6.3 Join 语句

6.3.1 等值 Join

Hive 支持通常的 SQL JOIN 语句。

1）案例实操

（1）根据员工表和部门表中的部门编号相等，查询员工编号、员工名称和部门名称；

```
hive (default)> select e.empno, e.ename, d.deptno, d.dname from emp e join dept d on e.deptno = d.deptno;
```

6.3.2 表的别名

1）好处

（1）使用别名可以简化查询。

（2）使用表名前缀可以提高执行效率。

2）案例实操

合并员工表和部门表

```
hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno = d.deptno;
```

6.3.3 内连接

```
hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno = d.deptno;
```

内连接：只有进行连接的两个表中都存在与连接条件相匹配的数据才会被保留下来。

6.3.4 左外连接

左外连接：JOIN 操作符左边表中符合 WHERE 子句的所有记录将会被返回。

```
hive (default)> select e.empno, e.ename, d.deptno from emp e left join dept d on e.deptno = d.deptno;
```

6.3.5 右外连接

右外连接：JOIN 操作符右边表中符合 WHERE 子句的所有记录将会被返回。

```
hive (default)> select e.empno, e.ename, d.deptno from emp e right join
dept d on e.deptno = d.deptno;
```

6.3.6 满外连接

满外连接：将会返回所有表中符合 WHERE 语句条件的所有记录。如果任一表的指定字段没有符合条件的值的话，那么就使用 NULL 值替代。

```
hive (default)> select e.empno, e.ename, d.deptno from emp e full join
dept d on e.deptno = d.deptno;
```

6.3.7 多表连接

注意：连接 n 个表，至少需要 n-1 个连接条件。例如：连接三个表，至少需要两个连接条件。

数据准备

```
1700 Beijing
1800 London
1900 Tokyo
```

1）创建位置表

```
create table if not exists location(
loc int,
loc_name string
)
row format delimited fields terminated by '\t';
```

2）导入数据

```
hive (default)> load data local inpath
'/usr/local/soft/datas/location.txt'
into table location;
```

3）多表连接查询

```
hive (default)>SELECT e.ename, d.dname, l.loc_name
FROM emp e
JOIN dept d
ON d.deptno = e.deptno
JOIN location l
ON d.loc = l.loc;
```

大多数情况下，Hive 会对每对 JOIN 连接对象启动一个 MapReduce 任务。本例中会首先 启动一个 MapReduce job 对表 e 和表 d 进行连接操作，然后会再启动一个 MapReduce job 将第一个 MapReduce job 的输出和表 l;进行连接操作。

注意：为什么不是表 d 和表 l 先进行连接操作呢？这是因为 Hive 总是按照从左到右的顺序执行的。

优化：当对 3 个或者更多表进行 join 连接时，如果每个 on 子句都使用相同的连接键的话，那么只会产生一个 MapReduce job。

6.3.8 笛卡尔积

1）笛卡尔集会在下面条件下产生

（1）省略连接条件

（2）连接条件无效

（3）所有表中的所有行互相连接

2）案例实操

```
hive (default)> select empno, dname from emp, dept;
```

6.4 排序

6.4.1 全局排序（Order By）

Order By：全局排序，只有一个 Reducer

1）使用 ORDER BY 子句排序

ASC（ascend）：升序（默认）

DESC（descend）：降序

2）ORDER BY 子句在 SELECT 语句的结尾

3）案例实操

（1）查询员工信息按工资升序排列

```
hive (default)> select * from emp order by sal;
```

（2）查询员工信息按工资降序排列

```
hive (default)> select * from emp order by sal desc;
```

6.4.2 按照别名排序

按照员工薪水的 2 倍排序

```
hive (default)> select ename, sal*2 twosal from emp order by twosal;
```

6.4.3 多个列排序

按照部门和工资升序排序

```
hive (default)> select ename, deptno, sal from emp order by deptno, sal;
```

6.4.4 每个 Reduce 内部排序（Sort By）

Sort By：对于大规模的数据集 order by 的效率非常低。在很多情况下，并不需要全局排序，此时可以使用 sort by。

Sort by 为每个 reducer 产生一个排序文件。每个 Reducer 内部进行排序，对全局结果集 来说不是排序。

1）设置 reduce 个数

```
hive (default)> set mapreduce.job.reduces=3;
```

2）查看设置 reduce 个数

```
hive (default)> set mapreduce.job.reduces;
```

3）根据部门编号降序查看员工信息

```
hive (default)> select * from emp sort by deptno desc;
```

4）将查询结果导入到文件中（按照部门编号降序排序）

```
hive (default)> insert overwrite local directory
'/usr/local/soft/data/sortby-result'
select * from emp sort by deptno desc;
```

6.4.5 分区（Distribute By）

Distribute By：

在有些情况下，我们需要控制某个特定行应该到哪个 reducer，通常是为了进行后续的聚集操作。**distribute by** 子句可以做这件事。**distribute by** 类似 MR 中 partition（自定义分区），进行分区，结合 sort by 使用。 对于 distribute by 进行测试，一定要分配多 reduce 进行处理，否则无法看到 distribute by 的效果。

1）案例实操：

（1）先按照部门编号分区，再按照员工编号降序排序。

```
hive (default)> set mapreduce.job.reduces=3;
hive (default)> insert overwrite local directory
'/usr/local/soft/data/distribute-result' select * from emp distribute by
deptno sort by empno desc;
```

注意：

➢ distribute by 的分区规则是根据分区字段的 hash 码与 reduce 的个数进行模除后，

余数相同的分到一个区。

➢ Hive 要求 DISTRIBUTE BY 语句要写在 SORT BY 语句之前。

6.4.6 Cluster By

当 distribute by 和 sorts by 字段相同时，可以使用 cluster by 方式。

cluster by 除了具有 distribute by 的功能外还兼具 sort by 的功能。但是排序只能是升序排序，不能指定排序规则为 ASC 或者 DESC。

（1）以下两种写法等价

```
hive (default)> select * from emp cluster by deptno;
hive (default)> select * from emp distribute by deptno sort by deptno;
```

注意：按照部门编号分区，不一定就是固定死的数值，可以是 20 号和 30 号部门分到一 个分区里面去。

第 7 章 分区表和分桶表

7.1 分区表

分区表实际上就是对应一个 HDFS 文件系统上的独立的文件夹，该文件夹下是该分区所 有的数据文件。Hive 中的分区就是分目录，把一个大的数据集根据业务需要分割成小的数据集。在查询时通过 WHERE 子句中的表达式选择查询所需要的指定的分区，这样的查询效率会提高很多

7.1.1 分区表基本操作

1）引入分区表（需要根据日期对日志进行管理，通过部门信息模拟）

```
dept_20220810.log
dept_20220811.log
dept_20220812.log
```

2）创建分区表语法

```
hive (default)> create table dept_partition(
deptno int, dname string, loc string
)
partitioned by (day string)
row format delimited fields terminated by '\t';
```

注意：分区字段不能是表中已经存在的数据，可以将分区字段看作表的伪列。

3）加载数据到分区表中

（1） 数据准备

dept_20220810.log

```
10  ACCOUNTING  1700
20  RESEARCH    1800
```

dept_20220811.log

```
30  SALES   1900
40  OPERATIONS  1700
```

dept_20220812.log

```
50  TEST    2000
60  DEV 1900
```

（2） **加载数据**

```
hive (default)> load data local inpath
'/usr/local/soft/data/dept_20220810.log' into table dept_partition
partition(day='20220810');
hive (default)> load data local inpath
'/usr/local/soft/data/dept_20220811.log' into table dept_partition
partition(day='20220811');
hive (default)> load data local inpath
'/usr/local/soft/data/dept_20220812.log' into table dept_partition
partition(day='20220812');
```

注意：分区表加载数据时，必须指定分区

4）查询分区表中数据

单分区查询

```
hive (default)> select * from dept_partition where day='20220810';
```

多分区联合查询

```
union
select * from dept_partition where day='20220811'
union
select * from dept_partition where day='20220812';
hive (default)> select * from dept_partition where day='20220810' or
day='20220811' or day='20220812';
```

5）增加分区

创建单个分区

```
hive (default)> alter table dept_partition add
partition(day='20220813');
```

同时创建多个分区

```
hive (default)> alter table dept_partition add partition(day='20220814')
partition(day='20220815');
```

6）删除分区

删除单个分区

```
hive (default)> alter table dept_partition drop partition
(day='20220815');
```

同时删除多个分区

```
hive (default)> alter table dept_partition drop partition
(day='20220813'), partition(day='20220814');
```

7）查看分区表有多少分区

```
hive> show partitions dept_partition; //推荐这种方式（直接从元数据中获取分区信息）
hive>select distinct pt from students_pt; // 不推荐
```

8）查看分区表结构

```
hive> desc formatted dept_partition;
# Partition Information
# col_name data_type comment
month string
```

7.1.2 二级分区

思考: 如何一天的日志数据量也很大，如何再将数据拆分?

1）创建二级分区表

```
hive (default)> create table dept_partition2(
deptno int, dname string, loc string
)
partitioned by (day string, hour string)
row format delimited fields terminated by '\t';
```

2）正常的加载数据

（1）加载数据到二级分区表中

```
hive (default)> load data local inpath
'/usr/local/soft/data/dept_20220810.log' into table
dept_partition2 partition(day='20220810', hour='12');
```

（2）查询分区数据

```
hive (default)> select * from dept_partition2 where day='20220810' and
hour='12';
```

3）把数据直接上传到分区目录上，让分区表和数据产生关联的三种方式

（1）方式一：上传数据后修复

上传数据

```
hive (default)> dfs -mkdir -p
/user/hive/warehouse/mydb.db/dept_partition2/day=20220810/hour=13;
hive (default)> dfs -put /usr/local/soft/data/dept_20220810.log
/user/hive/warehouse/mydb.db/dept_partition2/day=20220810/hour=13;
```

查询数据（查询不到刚上传的数据）

```
hive (default)> select * from dept_partition2 where day='20220810' and
hour='13';
```

执行修复命令

```
hive> msck repair table dept_partition2;
```

再次查询数据

```
hive (default)> select * from dept_partition2 where day='20220810' and
hour='13';
```

（2）方式二：上传数据后添加分区

上传数据

```
hive (default)> dfs -mkdir -p
/user/hive/warehouse/mydb.db/dept_partition2/day=20220811/hour=14;
hive (default)> dfs -put /usr/local/soft/data/dept_20220811.log
/user/hive/warehouse/mydb.db/dept_partition2/day=220220811/hour=14;
```

执行添加分区

```
hive (default)> alter table dept_partition2 add
partition(day='220220811',hour='14');
```

查询数据

```
hive (default)> select * from dept_partition2 where day='220220811' and
hour='14';
```

（3）方式三：创建文件夹后 load 数据到分区

创建目录

```
hive (default)> dfs -mkdir -p
/user/hive/warehouse/mydb.db/dept_partition2/day=220220811/hour=15;
```

上传数据

```
hive (default)> load data local inpath
'/usr/local/soft/data/dept_220220811.log' into table
dept_partition2 partition(day='220220811',hour='15');
```

查询数据

```
hive (default)> select * from dept_partition2 where day='220220811' and
hour='15';
```

7.1.3 动态分区调整

关系型数据库中，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。

1）开启动态分区参数设置

（1）开启动态分区功能（默认 true，开启）

```
hive.exec.dynamic.partition=true
```

（2）设置为非严格模式（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。）

```
hive.exec.dynamic.partition.mode=nonstrict
```

（3）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。默认 1000

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

（4）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.pernode=100
```

（5）整个 MR Job 中，最大可以创建多少个 HDFS 文件。默认 100000

```
hive.exec.max.created.files=100000
```

（6）当有空分区生成时，是否抛出异常。一般不需要设置。默认 false

```
hive.error.on.empty.partition=false
```

2）案例实操

需求：将 dept 表中的数据按照地区（loc 字段），插入到目标表 dept_partition 的相应分区中。

（1）创建目标分区表

```
hive (default)> create table dept_partition_dy(id int, name string)
partitioned by (loc int) row format delimited fields terminated by '\t';
```

（2）设置动态分区

```
set hive.exec.dynamic.partition.mode = nonstrict;
hive (default)> insert into table dept_partition_dy partition(loc)
select deptno, dname, loc from dept;
```

（3）查看目标分区表的分区情况

```
hive (default)> show partitions dept_partition;
```

思考：目标分区表是如何匹配到分区字段的？

7.2 分桶表

分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理 的分区。对于一张表或者分区，Hive 可以进一步组织成桶，也就是更为细粒度的数据范围 划分。分桶是将数据集分解成更容易管理的若干部分的另一个技术。

分区针对的是数据的存储路径；分桶针对的是数据文件。

1）先创建分桶表

（1）数据准备

```
1001    ss1
1002    ss2
1003    ss3
1004    ss4
1005    ss5
1006    ss6
1007    ss7
1008    ss8
1009    ss9
```

```
1010    ss10
1011    ss11
1012    ss12
1013    ss13
1014    ss14
1015    ss15
1016    ss16
```

（2）创建分桶表

```
create table stu_buck(id int, name string)
clustered by(id)
into 4 buckets
row format delimited fields terminated by '\t';
```

（3）查看表结构

```
hive (default)> desc formatted stu_buck;
Num Buckets: 4
```

（4）导入数据到分桶表中，load 的方式

```
hive (default)> load data inpath '/student.txt' into table stu_buck;
```

（5）查看创建的分桶表中是否分成 4 个桶

（6）查询分桶的数据

```
hive(default)> select * from stu_buck;
```

（7）分桶规则：

根据结果可知：Hive 的分桶采用对分桶字段的值进行哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中

2）分桶表操作需要注意的事项:

（1）reduce 的个数设置为-1,让 Job 自行决定需要用多少个 reduce 或者将 reduce 的个数设置为大于等于分桶表的桶数

（2）从 hdfs 中 load 数据到分桶表中，避免本地文件找不到问题

（3）不要使用本地模式

3）insert 方式将数据导入分桶表

```
hive(default)>insert into table stu_buck select * from student_insert;
```

7.3 抽样查询

对于非常大的数据集，有时用户需要使用的是一个具有代表性的查询结果而不是全部结果。Hive 可以通过对表进行抽样来满足这个需求。

语法: TABLESAMPLE(BUCKET x OUT OF y)

查询表 stu_buck 中的数据。

```
hive (default)> select * from stu_buck tablesample(bucket 1 out of 4 on id);
```

注意：x 的值必须小于等于 y 的值，否则

```
FAILED: SemanticException [Error 10061]: Numerator should not be bigger than denominator in sample clause for table stu_buck
```

第 8 章 函数

8.1 系统内置函数

1）查看系统自带的函数

```
hive> show functions;
```

2）显示自带的函数的用法

```
hive> desc function upper;
```

3）详细显示自带的函数的用法

```
hive> desc function extended upper;
```

8.2 常用内置函数

8.2.1 空字段赋值

1）函数说明

NVL：给值为 NULL 的数据赋值，它的格式是 NVL(value，default_value)。它的功能是如果 value 为 NULL，则 NVL 函数返回 default_value 的值，否则返回 value 的值，如果两个参数 都为 NULL ，则返回 NULL。

3）数据准备：采用员工表

3）查询：如果员工的 comm 为 NULL，则用-1 代替

```
hive (default)> select comm,nvl(comm, -1) from emp;
```

4）查询：如果员工的 comm 为 NULL，则用领导 id 代替

```
hive (default)> select comm, nvl(comm,mgr) from emp;
```

8.2.2 CASE WHEN THEN ELSE END

1）数据准备

name	dept_id	sex
悟空	A	男
大海	A	男
宋宋	B	男
凤姐	A	女
婷姐	B	女
婷婷	B	女

2）需求

求出不同部门男女各多少人。结果如下：

```
dept_Id 男 女
A 2 1
B 1 2
```

3）创建本地 emp_sex.txt，导入数据

```
[root@master datas]# vim emp_sex.txt
悟空 A 男
大海 A 男
宋宋 B 男
凤姐 A 女
婷姐 B 女
婷婷 B 女
```

4）创建 hive 表并导入数据

```
create table emp_sex(
name string,
```

```
dept_id string,
sex string)
row format delimited fields terminated by "\t";
load data local inpath '/usr/local/soft/dataemp_sex.txt' into table
emp_sex;
```

5）按需求查询数据

```
dept_id,
sum(case sex when '男' then 1 else 0 end) male_count,
sum(case sex when '女' then 1 else 0 end) female_count
from emp_sex
group by dept_id;
```

8.2.3 行转列

1）相关函数说明

CONCAT(string A/col, string B/col…)：返回输入字符串连接后的结果，支持任意个输入字符串;

CONCAT_WS(separator, str1, str2,...)：它是一个特殊形式的 CONCAT()。第一个参数剩余参 数间的分隔符。分隔符可以是与剩余参数一样的字符串。如果分隔符是 NULL，返回值也将 为 NULL。这个函数会跳过分隔符参数后的任何 NULL 和空字符串。分隔符将被加到被连接的字符串之间;

注意: CONCAT_WS must be "string or array<string>

COLLECT_SET(col)：函数只接受基本数据类型，它的主要作用是将某字段的值进行去重汇总，产生 Array 类型字段。

2）数据准备

name	constellation	blood_type
孙悟空	白羊座	A
大海	射手座	A
宋宋	白羊座	B
猪八戒	白羊座	A
凤姐	射手座	A
苍老师	白羊座	B

3）需求

把星座和血型一样的人归类到一起。结果如下：

```
射手座,A 大海|凤姐
白羊座,A 孙悟空|猪八戒
```

```
白羊座,B 宋宋|苍老师
```

4）创建本地 constellation.txt，导入数据

```
[root@master datas]# vim constellation.txt
孙悟空 白羊座 A
大海 射手座 A
宋宋 白羊座 B
猪八戒 白羊座 A
凤姐 射手座 A
苍老师 白羊座 B
```

5）创建 hive 表并导入数据

```
create table person_info(
name string,
constellation string,
blood_type string)
row format delimited fields terminated by ",";
load data local inpath "/usr/local/soft/data/constellation.txt" into
table person_info;
```

6）按需求查询数据

```
SELECT
t1.c_b,
CONCAT_WS("|",collect_set(t1.name))
FROM (
SELECT
NAME,
CONCAT_WS(',',constellation,blood_type) c_b
FROM person_info
)t1
GROUP BY t1.c_b
```

使用 concat（）函数做字符串的拼接；
使用 concat_ws（）和 collect_set（）进行合并行
将上面列表中一个 user 可能会占用多行转换为每个 user 占一行的目标表格式，实际是“列转行”
collect_set 的作用：
（1）去重，对 group by 后面的 user 进行去重
（2）对 group by 以后属于同一 user 的形成一个集合，结合 concat_ws 对集合中元素使用，进行分隔形成字符串

8.2.4 列转行

1）函数说明

EXPLODE(col)：将 hive 一列中复杂的 Array 或者 Map 结构拆分成多行。

LATERAL VIEW

用法：LATERAL VIEW udtf(expression) tableAlias AS columnAlias

解释：用于和 split, explode 等 UDTF 一起使用，它能够将一列数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。

2）数据准备

movie	category
《疑犯追踪》	悬疑,动作,科幻,剧情
《Lie to me》	悬疑,警匪,动作,心理,剧情
《战狼 2》	战争,动作,灾难

3）需求

将电影分类中的数组数据展开。结果如下：

```
《疑犯追踪》      悬疑
《疑犯追踪》      动作
《疑犯追踪》      科幻
《疑犯追踪》      剧情
《Lie to me》    悬疑
《Lie to me》    警匪
《Lie to me》    动作
《Lie to me》    心理
《Lie to me》    剧情
《战狼 2》   战争
《战狼 2》   动作
《战狼 2》   灾难
```

4）创建本地 movie.txt，导入数据

```
[root@master datas]# vi movie_info.txt
《疑犯追踪》 悬疑,动作,科幻,剧情
《Lie to me》悬疑,警匪,动作,心理,剧情
《战狼 2》 战争,动作,灾难
```

5）创建 hive 表并导入数据

```
create table movie_info(
movie string,
category string)
row format delimited fields terminated by "\t";
```

```
load data local inpath "/usr/local/soft/data/movie.txt" into table
movie_info;
```

6）按需求查询数据

```
SELECT
movie,
category_name
FROM
movie_info
lateral VIEW
explode(split(category,",")) movie_info_tmp AS category_name;
```

8.2.5 窗口函数（开窗函数）

1）相关函数说明

OVER()：指定分析函数工作的数据窗口大小，这个数据窗口大小可能会随着行的变而变化。

CURRENT ROW：当前行

n PRECEDING：往前 n 行数据

n FOLLOWING：往后 n 行数据

UNBOUNDED：起点，

UNBOUNDED PRECEDING 表示从前面的起点，

UNBOUNDED FOLLOWING 表示到后面的终点

LAG(col,n,default_val)：往前第 n 行数据

LEAD(col,n, default_val)：往后第 n 行数据

NTILE(n)：把有序窗口的行分发到指定数据的组中，各个组有编号，编号从 1 开始，对于每一行，NTILE 返回此行所属的组的编号。注意：n 必须为 int 类型。

2）数据准备：name，orderdate，cost

```
jack,2017-01-01,10
tony,2017-01-02,15
jack,2017-02-03,23
tony,2017-01-04,29
jack,2017-01-05,46
jack,2017-04-06,42
tony,2017-01-07,50
jack,2017-01-08,55
mart,2017-04-08,62
mart,2017-04-09,68
neil,2017-05-10,12
```

```
mart,2017-04-11,75
neil,2017-06-12,80
mart,2017-04-13,94
```

3）需求

（1）查询在 2017 年 4 月份购买过的顾客及总人数

（2）查询顾客的购买明细及月购买总额

（3）上述的场景，将每个顾客的 cost 按照日期进行累加

（4）查询每个顾客上次的购买时间

（5）查询前 20%时间的订单信息

4）创建本地 business.txt，导入数据

```
[root@master datas]$ vi business.txt
```

5）创建 hive 表并导入数据

```
create table business(
name string,
orderdate string,
cost int
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
load data local inpath "/usr/local/soft/data/business.txt" into table
business;
```

6）按需求查询数据

（1） 查询在 2017 年 4 月份购买过的顾客及总人数

```
select name,count(*) over ()
from business
where substring(orderdate,1,7) = '2017-04'
group by name;
```

（2） 查询顾客的购买明细及月购买总额

```
select name,orderdate,cost,sum(cost) over(partition by
month(orderdate))
from business;
```

（3） 将每个顾客的 cost 按照日期进行累加

```
select name,orderdate,cost,
sum(cost) over() as sample1,--所有行相加
sum(cost) over(partition by name) as sample2,--按 name 分组，组内数据相加
sum(cost) over(partition by name order by orderdate) as sample3,--按 name 分组，组内数据累加
sum(cost) over(partition by name order by orderdate rows between UNBOUNDED PRECEDING and current row ) as sample4 ,--和 sample3 一样,由起点到
当前行的聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING and current row) as sample5, --当前行和前面一行做聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING AND 1 FOLLOWING ) as sample6,--当前行和前边一行及后面一行
sum(cost) over(partition by name order by orderdate rows between current row and UNBOUNDED FOLLOWING ) as sample7 --当前行及后面所有行
from business;
```

rows 必须跟在 order by 子句之后，对排序的结果进行限制，使用固定的行数来限制分区中的数据行数量

（4） 查看顾客上次的购买时间

```
select name,orderdate,cost,
lag(orderdate,1,'1900-01-01') over(partition by name order by orderdate )
as time1, lag(orderdate,2) over (partition by name order by orderdate) as
time2
from business;
```

（5） 查询前 20%时间的订单信息

```
select * from (
select name,orderdate,cost, ntile(5) over(order by orderdate) sorted
from business
) t
where sorted = 1;
```

8.2.6 Rank

1）函数说明

RANK() 排序相同时会重复，总数不会变

DENSE_RANK() 排序相同时会重复，总数会减少
ROW_NUMBER() 会根据顺序计算

2）数据准备

name	subject	score
孙悟空	语文	87
孙悟空	数学	95
孙悟空	英语	68
大海	语文	94
大海	数学	56
大海	英语	84
宋宋	语文	64
宋宋	数学	86
宋宋	英语	84
婷婷	语文	65
婷婷	数学	85
婷婷	英语	78

3）需求

计算每门学科成绩排名。

4）创建本地 score.txt，导入数据

```
[root@master datas]$ vi score.txt
```

5）创建 hive 表并导入数据

```
create table score(
name string,
subject string,
score int)
row format delimited fields terminated by "\t";
load data local inpath '/usr/local/soft/data/score.txt' into table
score;
```

6）按需求查询数据

```
select name,
```

```
subject,
score,
rank() over(partition by subject order by score desc) rp,
dense_rank() over(partition by subject order by score desc) drp,
row_number() over(partition by subject order by score desc) rmp
from score;
```

扩展：求出每门学科前三名的学生？

8.4 自定义函数

学习网址：https://www.cnblogs.com/sx66/p/12039552.html

1.UDF：一进一出

（1）创建 maven 项目，并加入依赖

```
 <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-exec</artifactId>
            <version>1.2.1</version>
        </dependency>
```

（2)编写代码，继承 org.apache.hadoop.hive.ql.exec.UDF，实现 evaluate 方法，在 evaluate 方法中实现自己的逻辑

```
import org.apache.hadoop.hive.ql.exec.UDF;

public class HiveUDF extends UDF {
    // hadoop => #hadoop$
    public String evaluate(String col1) {
    // 给传进来的数据 左边加上 # 号 右边加上 $
        String result = "#" + col1 + "$";
        return result;
    }
}
```

（3）打成 jar 包并上传至 Linux 虚拟机

（4） 在 hive shell 中，使用 add jar 路径将 jar 包作为资源添加到 hive 环境中

```
add jar /usr/local/soft/jars/HiveUDF2-1.0.jar;
```

（5）使用 jar 包资源注册一个临时函数，fxxx1 是你的函数名，'MyUDF'是主类名

```
create temporary function fxxx1 as 'MyUDF';
```

（6）使用函数名处理数据

```
select fxx1(name) as fxx_name from students limit 10;
#施笑槐$
#吕金鹏$
#单乐蕊$
#葛德曜$
#宣谷芹$
#边昂雄$
#尚孤风$
#符半双$
#沈德昌$
#羿彦昌$
```

2.UDTF：一进多出

```
hive(default)> select myudtf("hello,world,hadoop,hive", ",");
hello
world
hadoop
hive
```

方法二：自定 UDTF

(1) 代码

```
package com.shujia.hive_function;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import
org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObje
ctInspectorFactory;

import java.util.ArrayList;
import java.util.List;
```

```
public class HiveUDTF extends GenericUDTF {
    private ArrayList<String> outList = new ArrayList<>();
    @Override
    public StructObjectInspector initialize(StructObjectInspector
argOIs)
            throws UDFArgumentException {
        //1.定义输出数据的列名和类型
        List<String> fieldNames = new ArrayList<>();
        List<ObjectInspector> fieldOIs = new ArrayList<>();
        //2.添加输出数据的列名和类型
        fieldNames.add("lineToWord");


fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspecto
r);
        return

ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames,
                        fieldOIs);
    }
    @Override
    public void process(Object[] args) throws HiveException {

        //1.获取原始数据
        String arg = args[0].toString();
        //2.获取数据传入的第二个参数，此处为分隔符
        String splitKey = args[1].toString();
        //3.将原始数据按照传入的分隔符进行切分
        String[] fields = arg.split(splitKey);
        //4.遍历切分后的结果，并写出
        for (String field : fields) {
            //集合为复用的，首先清空集合
            outList.clear();
            //将每一个单词添加至集合
            outList.add(field);
            //将集合内容写出
            forward(outList);
        }
    }
    @Override
    public void close() throws HiveException {
    }
}
```

(2) SQL

```
select my_udtf("key1:value1,key2:value2,key3:value3");
```

```
字段：
id,col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12 共13列

数据：
a,1,2,3,4,5,6,7,8,9,10,11,12
b,11,12,13,14,15,16,17,18,19,20,21,22
c,21,22,23,24,25,26,27,28,29,30,31,32
转成3列：id,hours,value
例如：
a,1,2,3,4,5,6,7,8,9,10,11,12
a,0时,1
a,2时,2
a,4时,3
a,6时,4
```

```
create table udtfData(
    id string
    ,col1 string
    ,col2 string
    ,col3 string
    ,col4 string
    ,col5 string
    ,col6 string
    ,col7 string
    ,col8 string
    ,col9 string
    ,col10 string
    ,col11 string
    ,col12 string
)row format delimited fields terminated by ',';
```

代码：

```
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
```

```
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import
org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObje
ctInspectorFactory;

import java.util.ArrayList;

public class HiveUDTF2 extends GenericUDTF {
    @Override
    public StructObjectInspector initialize(StructObjectInspector
argOIs) throws UDFArgumentException {
        ArrayList<String> filedNames = new ArrayList<String>();
        ArrayList<ObjectInspector> fieldObj = new
ArrayList<ObjectInspector>();
        filedNames.add("col1");

fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspecto
r);
        filedNames.add("col2");

fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspecto
r);
        return
ObjectInspectorFactory.getStandardStructObjectInspector(filedNames,
fieldObj);
    }

    public void process(Object[] objects) throws HiveException {
        int hours = 0;
        for (Object obj : objects) {
            hours = hours + 1;
            String col = obj.toString();
            ArrayList<String> cols = new ArrayList<String>();
            cols.add(hours + "时");
            cols.add(col);
            forward(cols);
        }
    }

    public void close() throws HiveException {
```

```
    }
}
```

添加 jar 资源:

```
add jar /usr/local/soft/HiveUDF2-1.0.jar;
```

注册 udtf 函数：

```
create temporary function my_udtf as 'MyUDTF';
```

SQL:

```
select id,hours,value from udtfData lateral view
my_udtf(col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12) t as hours,value ;
```

连续登陆问题

在电商、物流和银行可能经常会遇到这样的需求：统计用户连续交易的总额、连续登陆天数、连续登陆开始和结束时间、间隔天数等

数据：

注意：每个用户每天可能会有多条记录

```
id  datestr     amount
1,2019-02-08,6214.23
1,2019-02-08,6247.32
1,2019-02-09,85.63
1,2019-02-09,967.36
1,2019-02-10,85.69
1,2019-02-12,769.85
1,2019-02-13,943.86
1,2019-02-14,538.42
1,2019-02-15,369.76
1,2019-02-16,369.76
1,2019-02-18,795.15
1,2019-02-19,715.65
1,2019-02-21,537.71
2,2019-02-08,6214.23
2,2019-02-08,6247.32
```

```
2,2019-02-09,85.63
2,2019-02-09,967.36
2,2019-02-10,85.69
2,2019-02-12,769.85
2,2019-02-13,943.86
2,2019-02-14,943.18
2,2019-02-15,369.76
2,2019-02-18,795.15
2,2019-02-19,715.65
2,2019-02-21,537.71
3,2019-02-08,6214.23
3,2019-02-08,6247.32
3,2019-02-09,85.63
3,2019-02-09,967.36
3,2019-02-10,85.69
3,2019-02-12,769.85
3,2019-02-13,943.86
3,2019-02-14,276.81
3,2019-02-15,369.76
3,2019-02-16,369.76
3,2019-02-18,795.15
3,2019-02-19,715.65
3,2019-02-21,537.71
```

建表语句

```
create table deal_tb(
    id string
    ,datestr string
    ,amount string
)row format delimited fields terminated by ',';
```

计算逻辑

（1）先按用户和日期分组求和，使每个用户每天只有一条数据

```
select  id
        ,datestr
        ,sum(amount) as sum_amount
 from deal_tb
 group by id,datestr
```

（2）根据用户 ID 分组按日期排序，将日期和分组序号相减得到连续登陆的开始日期，如果开始日期相同说明连续登陆

```
 select  tt1.id
         ,tt1.datestr
         ,tt1.sum_amount
         ,date_sub(tt1.datestr,rn) as grp
 from(
     select  t1.id
             ,t1.datestr
             ,t1.sum_amount
             ,row_number() over(partition by id order by datestr) as rn
     from(
       select  id
                ,datestr
                ,sum(amount) as sum_amount
        from deal_tb
        group by id,datestr
     ) t1
 ) tt1
```

（3）统计用户连续交易的总额、连续登陆天数、连续登陆开始和结束时间、间隔天数

```
select  ttt1.id
        ,ttt1.grp
        ,round(sum(ttt1.sum_amount),2) as sc_sum_amount
        ,count(1) as sc_days
        ,min(ttt1.datestr) as sc_start_date
        ,max(ttt1.datestr) as sc_end_date
        ,datediff(ttt1.grp,lag(ttt1.grp,1) over(partition by ttt1.id
order by ttt1.grp)) as iv_days
from(
    select  tt1.id
            ,tt1.datestr
            ,tt1.sum_amount
            ,date_sub(tt1.datestr,rn) as grp
    from(
        select  t1.id
                ,t1.datestr
                ,t1.sum_amount
                ,row_number() over(partition by id order by datestr) as
rn
        from(
```

```
            select  id
                    ,datestr
                    ,sum(amount) as sum_amount
            from deal_tb
            group by id,datestr
        ) t1
    ) tt1
) ttt1
group by ttt1.id,ttt1.grp;
```

结果

```
1   2019-02-07 13600.23   3   2019-02-08 2019-02-10 NULL
1   2019-02-08 2991.650   5   2019-02-12 2019-02-16 1
1   2019-02-09 1510.8     2   2019-02-18 2019-02-19 1
1   2019-02-10 537.71     1   2019-02-21 2019-02-21 1
2   2019-02-07 13600.23   3   2019-02-08 2019-02-10 NULL
2   2019-02-08 3026.649   4   2019-02-12 2019-02-15 1
2   2019-02-10 1510.8     2   2019-02-18 2019-02-19 2
2   2019-02-11 537.71     1   2019-02-21 2019-02-21 1
3   2019-02-07 13600.23   3   2019-02-08 2019-02-10 NULL
3   2019-02-08 2730.04        5   2019-02-12 2019-02-16 1
3   2019-02-09 1510.8     2   2019-02-18 2019-02-19 1
3   2019-02-10 537.71     1   2019-02-21 2019-02-21 1
```

第 9 章 压缩和存储

9.1 Hadoop 压缩配置

9.1.1 MR 支持的压缩编码

压缩格式	算法	文件扩展名	是否可切分
DEFLATE	DEFLATE	.deflate	否
Gzip	DEFLATE	.gz	否
bzip2	bzip2	.bz2	是
LZO	LZO	.lzo	是
Snappy	Snappy	.snappy	否

为了支持多种压缩/解压缩算法，Hadoop 引入了编码/解码器，如下表所示：

压缩格式	对应的编码/解码器
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

压缩性能的比较：

压缩算法	原始文件大小	压缩文件大小	压缩速度	解压速度
gzip	8.3GB	1.8GB	17.5MB/s	58MB/s
bzip2	8.3GB	1.1GB	2.4MB/s	9.5MB/s
LZO	8.3GB	2.9GB	49.3MB/s	74.6MB/s

http://google.github.io/snappy/

On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

9.1.2 压缩参数配置

要在 Hadoop 中启用压缩，可以配置如下参数（mapred-site.xml 文件中）：

参数	默认值	阶段	建议
io.compression.codecs（在 core-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec, org.apache.hadoop.io.compress.GzipCodec, org.apache.hadoop.io.compress.BZip2Codec, org.apache.hadoop.io.compress.Lz4Codec	输入压缩	Hadoop 使用文件扩展名判断是否支持某种编解码器
mapreduce.map.output.compress	false	mapper 输出	这个参数设为 true 启用压缩
mapreduce.map.output.compress.codec	org.apache.hadoop.io.compress.DefaultCodec	mapper 输出	使用 LZO、LZ4 或 snappy 编解码器在此阶段压缩数据
mapreduce.output.fileoutputformat.compress	false	reducer 输出	这个参数设为 true 启用压缩
mapreduce.output.fileoutputformat.compress.codec	org.apache.hadoop.io.compress. DefaultCodec	reducer 输出	使用标准工具或者编解码器，如 gzip 和 bzip2
mapreduce.output.fileoutputformat.compress.type	RECORD	reducer 输出	SequenceFile 输出使用的压缩类型：NONE 和 BLOCK

9.2 开启 Map 输出阶段压缩（MR 引擎）

开启 map 输出阶段压缩可以减少 job 中 map 和 Reduce task 间数据传输量。具体配置如下：

1）案例实操：

（1）开启 hive 中间传输数据压缩功能

```
hive (default)>set hive.exec.compress.intermediate=true;
```

（2）开启 mapreduce 中 map 输出压缩功能

```
hive (default)>set mapreduce.map.output.compress=true;
```

（3）设置 mapreduce 中 map 输出数据的压缩方式

```
hive (default)>set mapreduce.map.output.compress.codec=
org.apache.hadoop.io.compress.SnappyCodec;
```

（4）执行查询语句

```
hive (default)> select count(ename) name from emp;
```

9.3 开启 Reduce 输出阶段压缩

当 Hive 将输出写入到表中时，输出内容同样可以进行压缩。属性 hive.exec.compress.output 控制着这个功能。用户可能需要保持默认设置文件中的默认值 false，这样默认的输出就是非压缩的纯文本文件了。用户可以通过在查询语句或执行脚本中设置这个值为 true，来开启输出结果压缩功能。

1）案例实操：

（1）开启 hive 最终输出数据压缩功能

```
hive (default)>set hive.exec.compress.output=true;
```

（2）开启 mapreduce 最终输出数据压缩

```
hive (default)>set mapreduce.output.fileoutputformat.compress=true;
```

（3）设置 mapreduce 最终数据输出压缩方式

```
hive (default)> set mapreduce.output.fileoutputformat.compress.codec =
org.apache.hadoop.io.compress.SnappyCodec;
```

（4）设置 mapreduce 最终数据输出压缩为块压缩

```
hive (default)> set
mapreduce.output.fileoutputformat.compress.type=BLOCK;
```

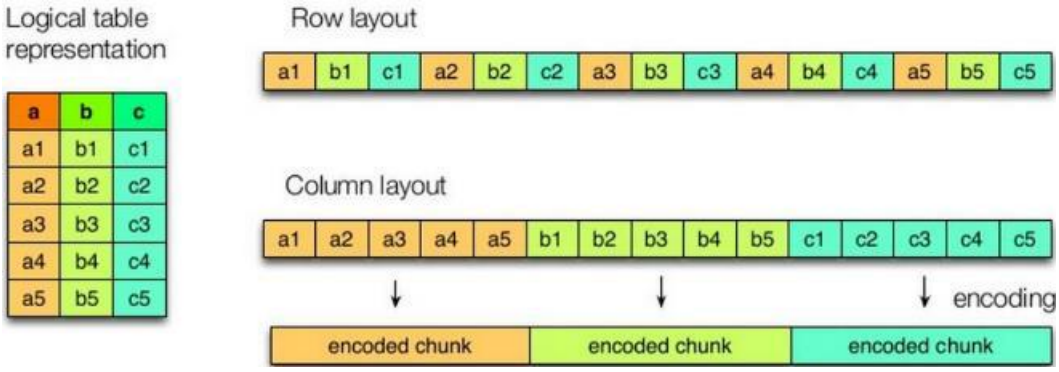
（5）测试一下输出结果是否是压缩文件

```
hive (default)> insert overwrite local directory
'/opt/module/data/distribute-result' select * from emp distribute by
deptno sort by empno desc;
```

9.4 文件存储格式

Hive 支持的存储数据的格式主要有：TEXTFILE 、SEQUENCEFILE、ORC、PARQUET

9.4.1 列式存储和行式存储



如图所示左边为逻辑表，右边第一个为行式存储，第二个为列式存储。

1）行存储的特点

查询满足条件的一整行数据的时候，列存储则需要去每个聚集的字段找到对应的每个列的值，行存储只需要找到其中一个值，其余的值都在相邻地方，所以此时行存储查询的速度更快。

2）列存储的特点

因为每个字段的数据聚集存储，在查询只需要少数几个字段的时候，能大大减少读取的
数据量；每个字段的数据类型一定是相同的，列式存储可以针对性的设计更好的设计压缩算法。

TEXTFILE 和 SEQUENCEFILE 的存储格式都是基于行存储的；

ORC 和 PARQUET 是基于列式存储的。

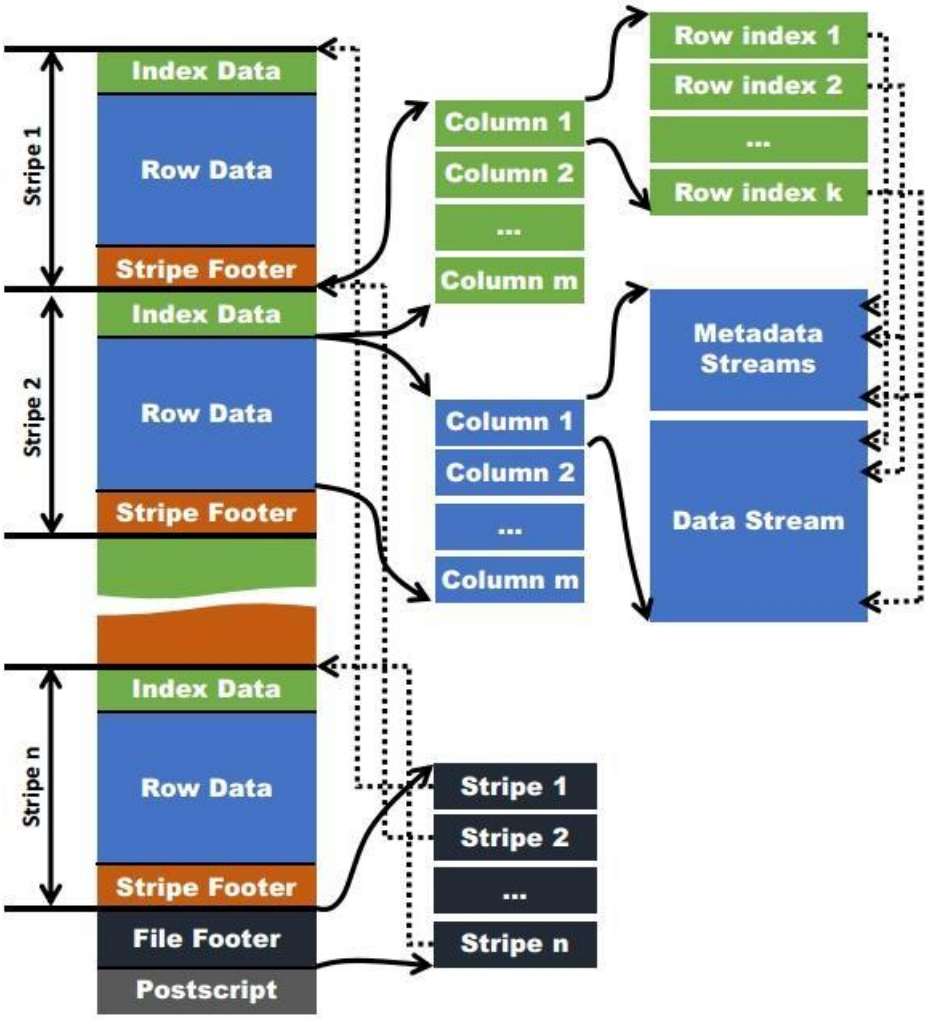
9.4.2 TextFile 格式

默认格式，数据不做压缩，磁盘开销大，数据解析开销大。可结合 Gzip、Bzip2 使用，但使用 Gzip 这种方式，hive 不会对数据进行切分，从而无法对数据进行并行操作。

9.4.3 Orc 格式

Orc (Optimized Row Columnar)是 Hive 0.11 版里引入的新的存储格式。

如下图所示可以看到每个 Orc 文件由 1 个或多个 stripe 组成，每个 stripe 一般为 HDFS 的块大小，每一个 stripe 包含多条记录，这些记录按照列进行独立存储，对应到 Parquet 中的 row group 的概念。每个 Stripe 里有三部分组成，分别是 Index Data，Row Data，Stripe Footer：



1）Index Data：一个轻量级的 index，默认是每隔 1W 行做一个索引。这里做的索引应该只是记录某行的各字段在 Row Data 中的 offset。

2）Row Data：存的是具体的数据，先取部分行，然后对这些行按列进行存储。对每个列进行了编码，分成多个 Stream 来存储。

3）Stripe Footer：存的是各个 Stream 的类型，长度等信息。每个文件有一个 File Footer，这里面存的是每个 Stripe 的行数，每个 Column 的数据类型信息等；每个文件的尾部是一个 PostScript，这里面记录了整个文件的压缩类型以及 FileFooter 的长度信息等。在读取文件时，会 seek 到文件尾部读 PostScript，从里面解析到 File Footer 长度，再读 FileFooter，从里面解析到各个 Stripe 信息，再读各个 Stripe，即从后往前读

9.4.4 Parquet 格式

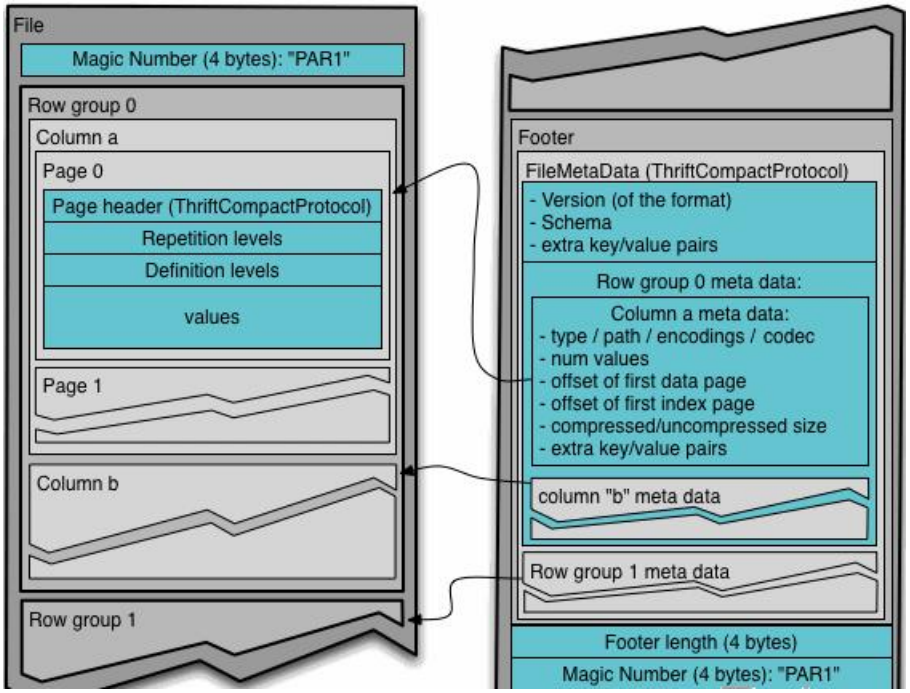
Parquet 文件是以二进制方式存储的，所以是不可以直接读取的，文件中包括该文件的
数据和元数据，因此 Parquet 格式文件是自解析的。

（1）行组(Row Group)：每一个行组包含一定的行数，在一个 HDFS 文件中至少存储一
个行组，类似于 orc 的 stripe 的概念。

（2）列块(Column Chunk)：在一个行组中每一列保存在一个列块中，行组中的所有列连续的存储在这个行组文件中。一个列块中的值都是相同类型的，不同的列块可能使用不同的算法进行压缩。

（3）页(Page)：每一个列块划分为多个页，一个页是最小的编码的单位，在同一个列块的不同页可能使用不同的编码方式。

通常情况下，在存储 Parquet 数据的时候会按照 Block 大小设置行组的大小，由于一般 情况下每一个 Mapper 任务处理数据的最小单位是一个 Block，这样可以把每一个行组由一个 Mapper 任务处理，增大任务执行并行度。Parquet 文件的格式。



上图展示了一个 Parquet 文件的内容，一个文件中可以存储多个行组，文件的首位都是 该文件的 Magic Code，用于校验它是否是一个 Parquet 文件，

Footer length 记录了文件元数据的大小，通过该值和文件长度可以计算出元数据的偏移量，文件的元数据中包括每一个行组的元数据信息和该文件存储数据的 Schema 信息。除了文件中每一个行组的元数据，每一 页的开始都会存储该页的元数据，在 Parquet 中，有三种类型的页：数据页、字典页和索引 页。数据页用于存储当前行组中该列的值，字典页存储该列值的编码字典，每一个列块中最多包含一个字典页，索引页用来存储当前行组下该列的索引，目前 Parquet 中还不支持索引页

9.4.5 主流文件存储格式对比实验

从存储文件的压缩比和查询速度两个角度对比。

存储文件的压缩比测试：

1）测试数据

Log.txt 见文档

2）TextFile

（1）创建表，存储数据格式为 TEXTFILE

```
create table log_text (
track_time string,
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as textfile;
```

（2）向表中加载数据

```
hive (default)> load data local inpath '/opt/module/hive/datas/log.data'
into table log_text ;
```

（3）查看表中数据大小

```
hive (default)> dfs -du -h /user/hive/warehouse/log_text;
```

18.13 M /user/hive/warehouse/log_text/log.data

3）ORC

（1）创建表，存储数据格式为 ORC

```
create table log_orc(
track_time string,
```

```
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as orc
tblproperties("orc.compress"="NONE"); -- 设置 orc 存储不使用压缩
```

（2）向表中加载数据

```
hive (default)> insert into table log_orc select * from log_text;
```

（3）查看表中数据大小

```
hive (default)> dfs -du -h /user/hive/warehouse/log_orc/ ;
```

7.7 M /user/hive/warehouse/log_orc/000000_0

4）Parquet

（1）创建表，存储数据格式为 parquet

```
create table log_parquet(
track_time string,
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as parquet;
```

（2）向表中加载数据

```
hive (default)> insert into table log_parquet select * from log_text;
```

（3）查看表中数据大小

```
hive (default)> dfs -du -h /user/hive/warehouse/log_parquet/;
```

13.1 M /user/hive/warehouse/log_parquet/000000_0

存储文件的对比总结：

ORC > Parquet > textFile

存储文件的查询速度测试：

（1）TextFile

```
hive (default)> insert overwrite local directory
'/opt/module/data/log_text' select substring(url,1,4) from log_text;
```

（2）ORC

```
hive (default)> insert overwrite local directory
'/opt/module/data/log_orc' select substring(url,1,4) from log_orc;
```

（3）Parquet

```
hive (default)> insert overwrite local directory
'/opt/module/data/log_parquet' select substring(url,1,4) from
log_parquet;
```

存储文件的查询速度总结：查询速度相近。

9.5 存储和压缩结合

9.5.1 测试存储和压缩

官网：

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

ORC 存储方式的压缩：

Key	Default	Notes
orc.compress	ZLIB	high level compression (one of NONE, ZLIB, SNAPPY)
orc.compress.size	262,144	number of bytes in each compression chunk
orc.stripe.size	268,435,456	number of bytes in each stripe
orc.row.index.stride	10,000	number of rows between index entries (must be >= 1000)
orc.create.index	true	whether to create row indexes
orc.bloom.filter.columns	""	comma separated list of column names for which bloom filter should be created
orc.bloom.filter.fpp	0.05	false positive probability for bloom filter (must >0.0 and <1.0)

注意：所有关于 ORCFile 的参数都是在 HQL 语句的 TBLPROPERTIES 字段里面出现

1）创建一个 ZLIB 压缩的 ORC 存储方式

（1）建表语句

```
create table log_orc_zlib(
track_time string,
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as orc
tblproperties("orc.compress"="ZLIB");
```

（2）插入数据

```
insert into log_orc_zlib select * from log_text;
```

（3）查看插入后数据

```
hive (default)> dfs -du -h /user/hive/warehouse/log_orc_zlib/ ;
```

2.78 M /user/hive/warehouse/log_orc_none/000000_0

2）创建一个 SNAPPY 压缩的 ORC 存储方式

（1）建表语句

```
create table log_orc_snappy(
track_time string,
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as orc
tblproperties("orc.compress"="SNAPPY");
```

（2）插入数据

```
insert into log_orc_snappy select * from log_text;
```

（3）查看插入后数据

```
hive (default)> dfs -du -h /user/hive/warehouse/log_orc_snappy/;
```

3.75 M /user/hive/warehouse/log_orc_snappy/000000_0

ZLIB 比 Snappy 压缩的还小。原因是 ZLIB 采用的是 deflate 压缩算法。比 snappy 压缩的 压缩率高。

3）创建一个 SNAPPY 压缩的 parquet 存储方式

（1）建表语句

```
create table log_parquet_snappy(
track_time string,
url string,
session_id string,
referer string,
ip string,
end_user_id string,
city_id string
)
row format delimited fields terminated by '\t'
stored as parquet
tblproperties("parquet.compression"="SNAPPY");
```

（2）插入数据

```
insert into log_parquet_snappy select * from log_text;
```

（3）查看插入后数据

```
hive (default)> dfs -du -h /user/hive/warehouse/log_parquet_snappy/;
```

6.39 MB /user/hive/warehouse/ log_parquet_snappy /000000_0

4）存储方式和压缩总结

在实际的项目开发当中,hive 表的数据存储格式一般选择:orc 或 parquet。压缩方式一 般选择 snappy，lzo。

第 10 章 企业级调优

10.1 执行计划（Explain）

1） 基本语法

```
EXPLAIN [EXTENDED | DEPENDENCY | AUTHORIZATION] query
```

2）案例操作

（1）查看下面这条语句的执行计划

没有生成 MR 任务的

```
hive (default)> explain select * from emp;
Explain
STAGE DEPENDENCIES:
Stage-0 is a root stage
STAGE PLANS:
Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
TableScan
alias: emp
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
Select Operator
expressions: empno (type: int), ename (type: string), job
(type: string), mgr (type: int), hiredate (type: string), sal (type:
double), comm (type: double), deptno (type: int)
outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6, _col7
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
ListSink
```

有生成 MR 任务的

```
hive (default)> explain select deptno, avg(sal) avg_sal from emp group
by
deptno;
Explain
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: emp
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
Select Operator
expressions: sal (type: double), deptno (type: int)
outputColumnNames: sal, deptno
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
Group By Operator
aggregations: sum(sal), count(sal)
keys: deptno (type: int)
mode: hash
outputColumnNames: _col0, _col1, _col2
Statistics: Num rows: 1 Data size: 7020 Basic stats:
COMPLETE Column stats: NONE
Reduce Output Operator
key expressions: _col0 (type: int)
sort order: +
Map-reduce partition columns: _col0 (type: int)
Statistics: Num rows: 1 Data size: 7020 Basic stats:
COMPLETE Column stats: NONE
value expressions: _col1 (type: double), _col2 (type:
bigint)
Execution mode: vectorized
Reduce Operator Tree:
Group By Operator
aggregations: sum(VALUE._col0), count(VALUE._col1)
keys: KEY._col0 (type: int)
mode: mergepartial
outputColumnNames: _col0, _col1, _col2
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
Select Operator
expressions: _col0 (type: int), (_col1 / _col2) (type: double)
outputColumnNames: _col0, _col1
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
File Output Operator
compressed: false
```

```
Statistics: Num rows: 1 Data size: 7020 Basic stats: COMPLETE
Column stats: NONE
table:
input format:
org.apache.hadoop.mapred.SequenceFileInputFormat
output format:
org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink
```

（2）查看详细执行计划

```
hive (default)> explain extended select * from emp;
hive (default)> explain extended select deptno, avg(sal) avg_sal from emp
group by deptno;
```

10.2 Fetch 抓取

Fetch 抓取是指，Hive 中对某些情况的查询可以不必使用 MapReduce 计算。例如：SELECT * FROM employees;在这种情况下，Hive 可以简单地读取 employee 对应的存储目录下的文件，然后输出查询结果到控制台。

在 hive-default.xml.template 文件中 hive.fetch.task.conversion 默认是 more，老版本 hive 默认是 minimal，该属性修改为 more 以后，在全局查找、字段查找、limit 查找等都不走 mapreduce。

```
<property>
<name>hive.fetch.task.conversion</name>
<value>more</value>
<description>
Expects one of [none, minimal, more].
Some select queries can be converted to single FETCH task minimizing
latency.
Currently the query should be single sourced not having any subquery
and should not have any aggregations or distincts (which incurs RS),
lateral views and joins.
0. none : disable hive.fetch.task.conversion
1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
2. more : SELECT, FILTER, LIMIT only (support TABLESAMPLE and
```

```
virtual columns)
</description>
</property
```

1）案例实操：

（1）把 hive.fetch.task.conversion 设置成 none，然后执行查询语句，都会执行 mapreduce 程序。

```
hive (default)> set hive.fetch.task.conversion=none;
hive (default)> select * from emp;
hive (default)> select ename from emp;
hive (default)> select ename from emp limit 3;
```

（2）把 hive.fetch.task.conversion 设置成 more，然后执行查询语句，如下查询方式都不会执行 mapreduce 程序。

```
hive (default)> set hive.fetch.task.conversion=more;
hive (default)> select * from emp;
hive (default)> select ename from emp;
hive (default)> select ename from emp limit 3;
```

10.3 本地模式

大多数的 Hadoop Job 是需要 Hadoop 提供的完整的可扩展性来处理大数据集的。不过，有时 Hive 的输入数据量是非常小的。在这种情况下，为查询触发执行任务消耗的时间可能会比实际 job 的执行时间要多的多。对于大多数这种情况，Hive 可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

用户可以通过设置 hive.exec.mode.local.auto 的值为 true，来让 Hive 在适当的时候自动启动这个优化。

```
set hive.exec.mode.local.auto=true; //开启本地 mr
//设置 local mr 的最大输入数据量，当输入数据量小于这个值时采用 local mr 的方式，默认
为 134217728，即 128M
set hive.exec.mode.local.auto.inputbytes.max=50000000;
//设置 local mr 的最大输入文件个数，当输入文件个数小于这个值时采用 local mr 的方式，默
认为 4
set hive.exec.mode.local.auto.input.files.max=10;
```

1）案例实操：

（2）关闭本地模式（默认是关闭的），并执行查询语句

```
hive (default)> select count(*) from emp group by deptno;
```

10.4 表的优化

10.4.1 小表大表 Join（MapJOIN）

将 key 相对分散，并且数据量小的表放在 join 的左边，可以使用 map join 让小的维度表先进内存。在 map 端完成 join。

实际测试发现：新版的 hive 已经对小表 JOIN 大表和大表 JOIN 小表进行了优化。小表放在左边和右边已经没有区别。

案例实操

1）需求介绍

测试大表 JOIN 小表和小表 JOIN 大表的效率

2）开启 MapJoin 参数设置

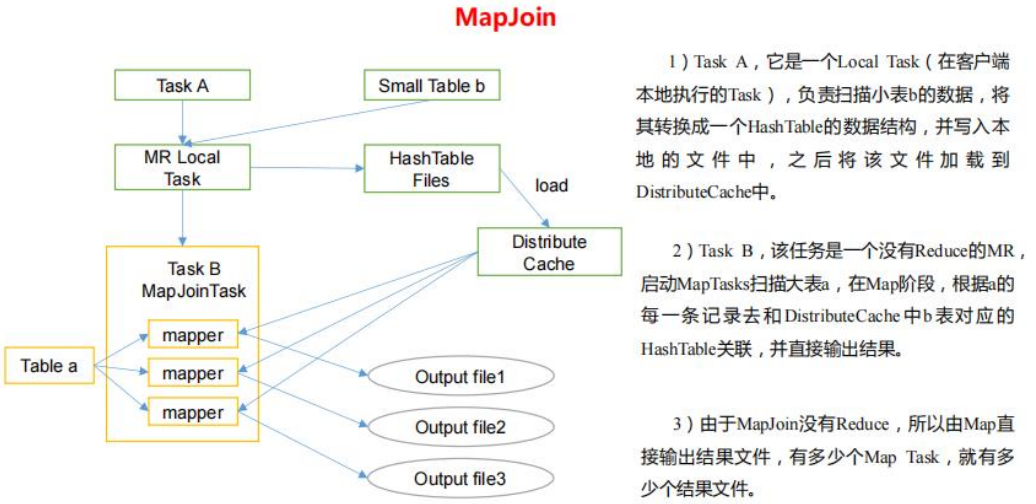
（1）设置自动选择 Mapjoin

```
set hive.auto.convert.join = true; 默认为 true
```

（2）大表小表的阈值设置（默认 25M 以下认为是小表）：

```
set hive.mapjoin.smalltable.filesize = 25000000;
```

3）MapJoin 工作机制



4）建大表、小表和 JOIN 后表的语句

```
// 创建大表
create table bigtable(id bigint, t bigint, uid string, keyword string,
url_rank int, click_num int, click_url string) row format delimited
fields terminated by '\t';
// 创建小表
```

```
create table smalltable(id bigint, t bigint, uid string, keyword string,
url_rank int, click_num int, click_url string) row format delimited
fields terminated by '\t';
// 创建 join 后表的语句
create table jointable(id bigint, t bigint, uid string, keyword string,
url_rank int, click_num int, click_url string) row format delimited
fields terminated by '\t';
```

5）分别向大表和小表中导入数据

```
hive (default)> load data local inpath '/opt/module/data/bigtable' into
table bigtable;
hive (default)>load data local inpath '/opt/module/data/smalltable' into
table smalltable;
```

6）小表 JOIN 大表语句

```
insert overwrite table jointable
select b.id, b.t, b.uid, b.keyword, b.url_rank, b.click_num, b.click_url
from smalltable s
join bigtable b
on b.id = s.id;
```

7）大表 JOIN 小表语句

```
insert overwrite table jointable
select b.id, b.t, b.uid, b.keyword, b.url_rank, b.click_num, b.click_url
from bigtable b
join smalltable s
on s.id = b.id;
```

10.4.2 大表 Join 大表

1）空 KEY 过滤

有时 join 超时是因为某些 key 对应的数据太多，而相同 key 对应的数据都会发送到相同的 reducer 上，从而导致内存不够。此时我们应该仔细分析这些异常的 key，很多情况下，这些 key 对应的数据是异常数据，我们需要在 SQL 语句中进行过滤。例如 key 对应的字段为空，操作如下：

案例实操

（1）配置历史服务器

配置 mapred-site.xml

```
<property>
<name>mapreduce.jobhistory.address</name>
```

```
<value>hadoop102:10020</value>
</property>
<property>
<name>mapreduce.jobhistory.webapp.address</name>
<value>hadoop102:19888</value>
</property>
```

启动历史服务器

```
sbin/mr-jobhistory-daemon.sh start historyserver
```

（2）创建原始数据空 id 表

```
// 创建空 id 表
create table nullidtable(id bigint, t bigint, uid string, keyword string, url_rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
```

（3）分别加载原始数据和空 id 数据到对应表中

```
hive (default)> load data local inpath '/opt/module/data/nullid' into table nullidtable;
```

（4）测试不过滤空 id

```
hive (default)> insert overwrite table jointable select n.* from nullidtable n left join bigtable o on n.id = o.id;
```

（5）测试过滤空 id

```
hive (default)> insert overwrite table jointable select n.* from (select * from nullidtable where id is not null) n left join bigtable o on n.id = o.id;
```

2）空 key 转换

有时虽然某个 key 为空对应的数据很多，但是相应的数据不是异常数据，必须要包含在 join 的结果中，此时我们可以表 a 中 key 为空的字段赋一个随机的值，使得数据随机均匀地分不到不同的 reducer 上。例如：

案例实操：

不随机分布空 null 值：

（1）设置 5 个 reduce 个数

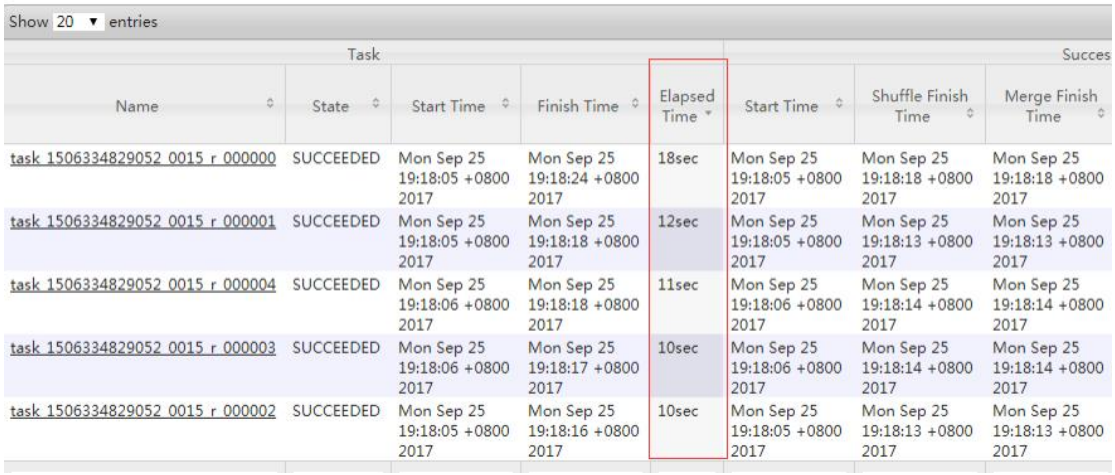
```
set mapreduce.job.reduces = 5;
```

（2）JOIN 两张表

```
insert overwrite table jointable
```

```
select n.* from nullidtable n left join bigtable b on n.id = b.id;
```

结果：如下图所示，可以看出来，出现了数据倾斜，某些 reducer 的资源消耗远大于其他 reducer。



Show 20 entries

Task					Succes		
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time
task 1506334829052 0015 r 000000	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:24 +0800 2017	18sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	Mon Sep 25 19:18:18 +0800 2017
task 1506334829052 0015 r 000001	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	12sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017
task 1506334829052 0015 r 000004	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	11sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task 1506334829052 0015 r 000003	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:17 +0800 2017	10sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task 1506334829052 0015 r 000002	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:16 +0800 2017	10sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017

随机分布空 null 值

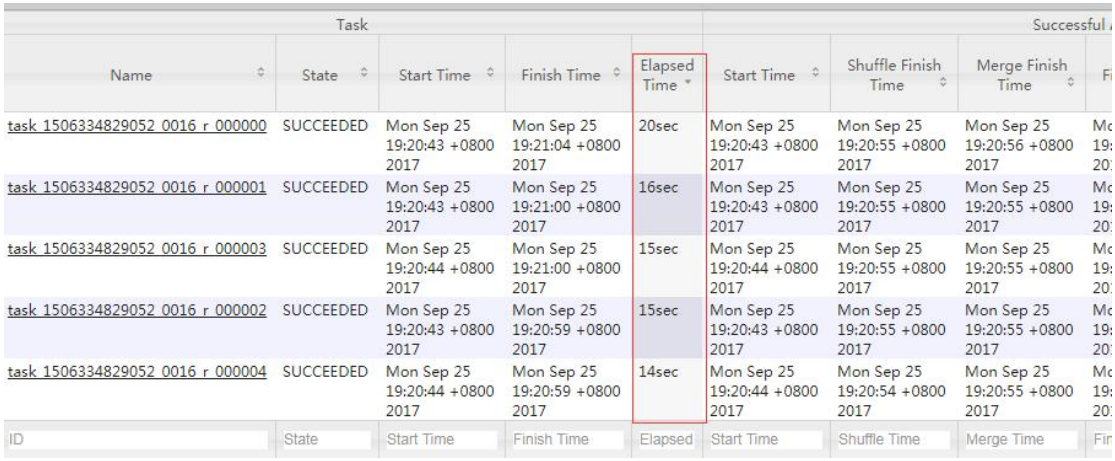
（1）设置 5 个 reduce 个数

```
set mapreduce.job.reduces = 5;
```

（2）JOIN 两张表

```
insert overwrite table jointable
select n.* from nullidtable n full join bigtable o on
nvl(n.id,rand()) = o.id;
```

结果：如下图所示，可以看出来，消除了数据倾斜，负载均衡 reducer 的资源消耗



Task					Successful			
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time	F
task 1506334829052 0016 r 000000	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:21:04 +0800 2017	20sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:56 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000001	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:21:00 +0800 2017	16sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000003	SUCCEEDED	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:21:00 +0800 2017	15sec	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000002	SUCCEEDED	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:59 +0800 2017	15sec	Mon Sep 25 19:20:43 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
task 1506334829052 0016 r 000004	SUCCEEDED	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:59 +0800 2017	14sec	Mon Sep 25 19:20:44 +0800 2017	Mon Sep 25 19:20:54 +0800 2017	Mon Sep 25 19:20:55 +0800 2017	Mo 19: 20:
ID	State	Start Time	Finish Time	Elapsed	Start Time	Shuffle Time	Merge Time	Fin

3）SMB(Sort Merge Bucket join)

（1）创建第二张大表

```
create table bigtable2(
id bigint,
```

```
t bigint,
uid string,
keyword string,
url_rank int,
click_num int,
click_url string)
row format delimited fields terminated by '\t';
load data local inpath '/opt/module/data/bigtable' into table bigtable2;
```

测试大表直接 JOIN

```
insert overwrite table jointable
select b.id, b.t, b.uid, b.keyword, b.url_rank, b.click_num, b.click_url
from bigtable s
join bigtable2 b
on b.id = s.id;
```

（2）创建分通表 1,桶的个数不要超过可用 CPU 的核数

```
create table bigtable_buck1(
id bigint,
t bigint,
uid string,
keyword string,
url_rank int,
click_num int,
click_url string)
clustered by(id)
sorted by(id)
into 6 buckets
row format delimited fields terminated by '\t';
load data local inpath '/opt/module/data/bigtable' into table
bigtable_buck1;
```

（3）创建分通表 2,桶的个数不要超过可用 CPU 的核数

```
create table bigtable_buck2(
id bigint,
t bigint,
uid string,
keyword string,
url_rank int,
click_num int,
click_url string)
```

```
clustered by(id)
sorted by(id)
into 6 buckets
row format delimited fields terminated by '\t';
load data local inpath '/opt/module/data/bigtable' into table
```

（4）设置参数

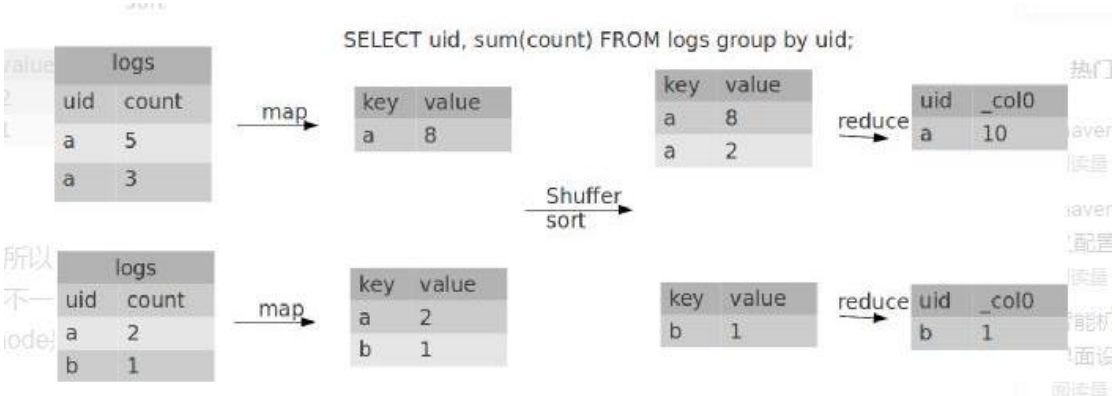
```
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set
hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFor
mat;
```

（5）测试

```
insert overwrite table jointable
select b.id, b.t, b.uid, b.keyword, b.url_rank, b.click_num, b.click_url
from bigtable_buck1 s
join bigtable_buck2 b
on b.id = s.id;
```

10.4.3 Group By

默认情况下，Map 阶段同一 Key 数据分发给一个 reduce，当一个 key 数据过大时就倾斜了。



并不是所有的聚合操作都需要在 Reduce 端完成，很多聚合操作都可以先在 Map 端进行 部分聚合，最后在 Reduce 端得出最终结果。

1）开启 Map 端聚合参数设置

（1）是否在 Map 端进行聚合，默认为 True

```
set hive.map.aggr = true
```

（2）在 Map 端进行聚合操作的条目数目

```
set hive.groupby.mapaggr.checkinterval = 100000
```

（3）有数据倾斜的时候进行负载均衡（默认是 false）

```
set hive.groupby.skewindata = true
```

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。

```
hive (default)> select deptno from emp group by deptno;
Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 23.68 sec HDFS Read:
19987 HDFS Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 23 seconds 680 msec
OK
deptno
10
20
30
```

优化以后

```
hive (default)> set hive.groupby.skewindata = true;
hive (default)> select deptno from emp group by deptno;
Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 28.53 sec HDFS Read:
18209 HDFS Write: 534 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 5 Cumulative CPU: 38.32 sec HDFS Read:
15014 HDFS Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 1 minutes 6 seconds 850 msec
OK
deptno
10
20
30
```

10.4.4 Count(Distinct) 去重统计

数据量小的时候无所谓，数据量大的情况下，由于 COUNT DISTINCT 操作需要用一个 Reduce Task 来完成，这一个 Reduce 需要处理的数据量太大，就会

导致整个 Job 很难完成，一般 COUNT DISTINCT 使用先 GROUP BY 再 COUNT 的方式替换,但是需要注意 group by 造成的数据倾斜问题.

1）**案例实操**

（1）创建一张大表

```
hive (default)> create table bigtable(id bigint, time bigint, uid string,
keyword
string, url_rank int, click_num int, click_url string) row format
delimited
fields terminated by '\t';
```

（2）加载数据

```
hive (default)> load data local inpath '/opt/module/data/bigtable' into
table bigtable;
```

（3）设置 5 个 reduce 个数

```
set mapreduce.job.reduces = 5;
```

（4）执行去重 id 查询

```
hive (default)> select count(distinct id) from bigtable;
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 7.12 sec HDFS Read:
120741990 HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 120 msec
OK
c0
100001
Time taken: 23.607 seconds, Fetched: 1 row(s)
```

（5）采用 GROUP by 去重 id

```
hive (default)> select count(id) from (select id from bigtable group by
id) a;
Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 17.53 sec HDFS Read:
120752703 HDFS Write: 580 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 4.29 sec2 HDFS Read:
9409 HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 21 seconds 820 msec
OK
_c0
100001
Time taken: 50.795 seconds, Fetched: 1 row(s)
```

虽然会多用一个 Job 来完成，但在数据量大的情况下，这个绝对是值得的。

10.4.5 笛卡尔积

尽量避免笛卡尔积，join 的时候不加 on 条件，或者无效的 on 条件，Hive 只能使用 1 个 reducer 来完成笛卡尔积。。

10.4.6 行列过滤

列处理：在 SELECT 中，只拿需要的列，如果有分区，尽量使用分区过滤，少用 SELECT
*。
行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在 Where 后面，那么就会先全表关联，之后再过滤，比如：
案例实操：
1）测试先关联两张表，再用 where 条件过滤

```
hive (default)> select o.id from bigtable b
join bigtable o on o.id = b.id
where o.id <= 10;
```

Time taken: 34.406 seconds, Fetched: 100 row(s)

2）通过子查询后，再关联表

```
hive (default)> select b.id from bigtable b
join (select id from bigtable where id <= 10) o on b.id = o.id;
```

Time taken: 30.058 seconds, Fetched: 100 row(s)

10.4.7 分区

详见 7.1 章。

10.4.8 分桶

详见 7.2 章。

10.5 合理设置 Map 及 Reduce 数

1）通常情况下，作业会通过 input 的目录产生一个或者多个 map 任务。
主要的决定因素有：input 的文件总个数，input 的文件大小，集群设置的文件块大小。
2）是不是 map 数越多越好？

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m），则每个小文件也会被当做一个块，用一个 map 任务来完成，而一个 map 任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的 map 数是受限的。

3）是不是保证每个 map 处理接近 128m 的文件块，就高枕无忧了？

答案也是不一定。比如有一个 127m 的文件，正常会用一个 map 去完成，但这个文件只针对上面的问题 2 和 3，我们需要采取两种方式来解决：即减少 map 数和增加 map 数；

10.5.1 复杂文件增加 Map 数

当 input 的文件都很大，任务逻辑复杂，map 执行非常慢的时候，可以考虑增加 Map 数，来使得每个 map 处理的数据量减少，从而提高任务的执行效率。

增加 map 的方法为：根据

computeSliteSize(Math.max(minSize,Math.min(maxSize,blocksize)))=blocksize=128M 公式，

调整 maxSize 最大值。让 maxSize 最大值低于 blocksize 就可以增加 map 的个数。

案例实操：

1）执行查询

```
hive (default)> select count(*) from emp;
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
```

2）设置最大切片值为 100 个字节

```
hive (default)> set mapreduce.input.fileinputformat.split.maxsize=100;
hive (default)> select count(*) from emp;
Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 1
```

10.5.2 小文件进行合并

1）在 map 执行前合并小文件，减少 map 数：CombineHiveInputFormat 具有对小文件进行合 并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。

```
set hive.input.format=
org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

2）在 Map-Reduce 的任务结束时合并小文件的设置：

在 map-only 任务结束时合并小文件，默认 true

```
SET hive.merge.mapfiles = true;
```

在 map-reduce 任务结束时合并小文件，默认 false

```
SET hive.merge.mapredfiles = true;
```

合并文件的大小，默认 256M

```
SET hive.merge.size.per.task = 268435456;
```

当输出文件的平均大小小于该值时，启动一个独立的 map-reduce 任务进行文件 merge

```
SET hive.merge.smallfiles.avgsize = 16777216;
```

10.5.3 合理设置 Reduce 数

1）调整 reduce 个数方法一

（1）每个 Reduce 处理的数据量默认是 256MB

```
hive.exec.reducers.bytes.per.reducer=256000000
```

（2）每个任务最大的 reduce 数，默认为 1009

```
hive.exec.reducers.max=1009
```

（3）计算 reducer 数的公式

```
N=min(参数 2，总输入数据量/参数 1)
```

2）调整 reduce 个数方法二

在 hadoop 的 mapred-default.xml 文件中修改

```
set mapreduce.job.reduces = 15;
```

设置每个 job 的 Reduce 个数

3）reduce 个数并不是越多越好

（1）过多的启动和初始化 reduce 也会消耗时间和资源；

（2）另外，有多少个 reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置 reduce 个数的时候也需要考虑这两个原则：处理大数据量利用合适的 reduce 数；使单个 reduce 任务处理数据量大小要合适；

10.6 并行执行

Hive 会将一个查询转化成一个或者多个阶段。这样的阶段可以是 MapReduce 阶段、抽样阶段、合并阶段、limit 阶段。或者 Hive 执行过程中可能需要的其他阶段。默认情况下，Hive 一次只会执行一个阶段。不过，某个特定的 job 可能包含众多的阶段，而这些阶段可能并非完全互相依赖的，也就是说有些阶段是可以并行执行的，这样可能使得整个 job 的执行时间缩短。不过，如果有更多的阶段可以并行执行，那么 job 可能就越快完成。

通过设置参数 hive.exec.parallel 值为 true，就可以开启并发执行。不过，在共享集群中，需要注意下，如果 job 中并行阶段增多，那么集群利用率就会增加。

```
set hive.exec.parallel=true; //打开任务并行执行
set hive.exec.parallel.thread.number=16; //同一个 sql 允许最大并行度，默认为
8。
```

当然，得是在系统资源比较空闲的时候才有优势，否则，没资源，并行也起不来。

10.7 严格模式

Hive 可以通过设置防止一些危险操作：

1）分区表不使用分区过滤

将 hive.strict.checks.no.partition.filter 设置为 true 时，对于分区表，除非 where 语句中含有分区字段过滤条件来限制范围，否则不允许执行。换句话说，就是用户不允许扫描所有分区。进行这个限制的原因是，通常分区表都拥有非常大的数据集，而且数据增加迅速。没有进行分区限制的查询可能会消耗令人不可接受的巨大资源来处理这个表。

2）使用 order by 没有 limit 过滤

将 hive.strict.checks.orderby.no.limit 设置为 true 时，对于使用了 order by 语句的查询，要 求必须使用 limit 语句。因为 order by 为了执行排序过程会将所有的结果数据分发到同一个 Reducer 中进行处理，强制要求用户增加这个 LIMIT 语句可以防止 Reducer 额外执行很长一段时间。

3）笛卡尔积

将 hive.strict.checks.cartesian.product 设置为 true 时，会限制笛卡尔积的查询。对关系型数据库非常了解的用户可能期望在 执行 JOIN 查询的时候不使用 ON 语句而是使用 where 语句，这样关系数据库的执行优化器就可以高效地将 WHERE 语句转化成那个 ON 语句。不幸的是，Hive 并不会执行这种优化，因此，如果表足够大，那么这个查询就会出现不可控的情况。

10.8 JVM 重用

10.9 压缩