尚硅谷大数据技术之 Hive

(作者：尚硅谷大数据研发部)

版本：V1.5

第 1 章 Hive 基本概念

1.1 什么是 Hive

Hive：由 Facebook 开源用于解决海量结构化日志的数据统计。

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。

本质是：将 HQL 转化成 MapReduce 程序

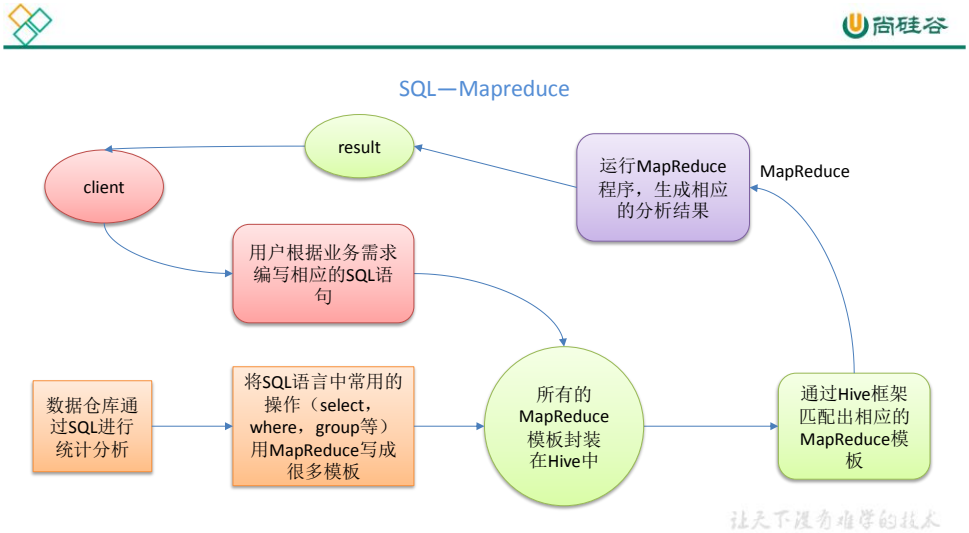

图 1-1 HQL 转换 MR 流程

1）Hive 处理的数据存储在 HDFS

2）Hive 分析数据底层的默认实现是 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

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

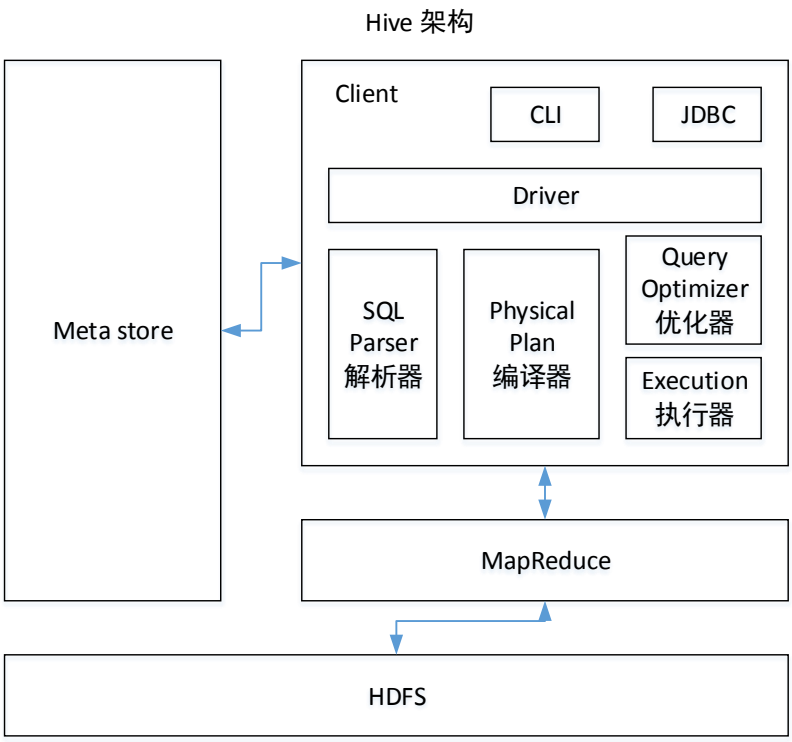

图 1-2 Hive 架构原理

1. 用户接口：Client

CLI（hive shell）、JDBC/ODBC(java 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。

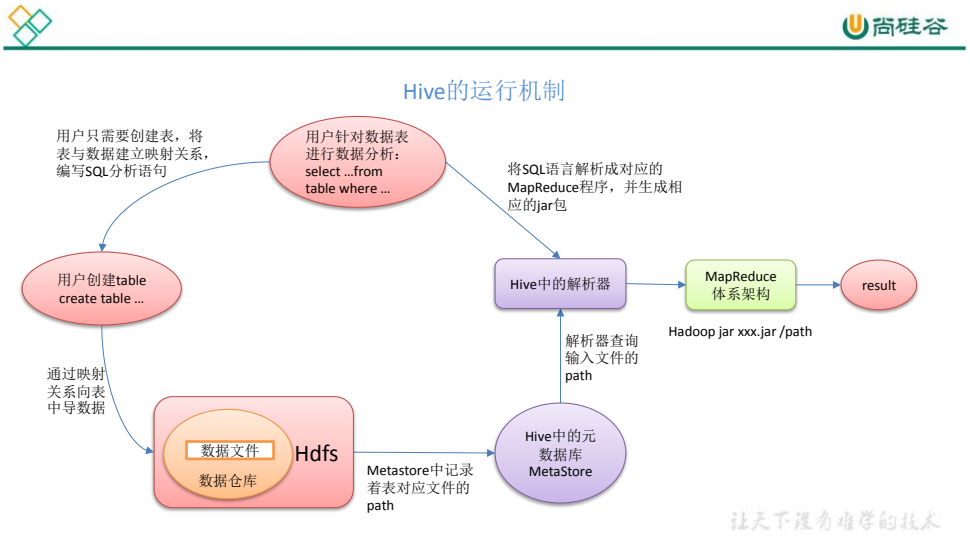

图 1-3 Hive 运行机制

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。本节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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

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 数据存储位置

Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

1.4.3 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。

1.4.4 索引

Hive 在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些 Key 建立索引。Hive 要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。数据库中，通常会针对一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

1.4.5 执行

Hive 中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的。而数据库通常有自己的执行引擎。

1.4.6 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。

1.4.7 可扩展性

由于 Hive 是建立在 Hadoop 之上的，因此 Hive 的可扩展性是和 Hadoop 的可扩展性是一致的（世界上最大的 Hadoop 集群在 Yahoo!，2009 年的规模在 4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 Oracle 在理论上的扩展能力也只有 100 台左右。

1.4.8 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 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/

2.2 Hive 安装部署

1. Hive 安装及配置

（1）把 apache-hive-1.2.1-bin.tar.gz 上传到 linux 的/opt/software 目录下

（2）解压 apache-hive-1.2.1-bin.tar.gz 到/opt/module/目录下面

```
[atguigu@hadoop102 software]$ tar -zxvf apache-hive-1.2.1-bin.tar.gz -C /opt/module/
```

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

```
[atguigu@hadoop102 module]$ mv apache-hive-1.2.1-bin/ hive
```

（4）修改/opt/module/hive/conf 目录下的 hive-env.sh.template 名称为 hive-env.sh

```
[atguigu@hadoop102 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=/opt/module/hadoop-2.7.2
```

（b）配置 HIVE_CONF_DIR 路径

```
export HIVE_CONF_DIR=/opt/module/hive/conf
```

2．Hadoop 集群配置

（1）必须启动 hdfs 和 yarn

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh
[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh
```

（2）在 HDFS 上创建/tmp 和/user/hive/warehouse 两个目录并修改他们的同组权限可写(可不操作，系统会自动创建)

```
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -mkdir /tmp
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -mkdir -p /user/hive/warehouse
```

```
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -chmod g+w /tmp
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -chmod g+w /user/hive/warehouse
```

3．Hive 基本操作

（1）启动 hive

```
[atguigu@hadoop102 hive]$ bin/hive
```

（2）查看数据库

```
hive> show databases;
```

（3）打开默认数据库

```
hive> use default;
```

（4）显示 default 数据库中的表

```
hive> show tables;
```

（5）创建一张表

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

（6）显示数据库中有几张表

```
hive> show tables;
```

（7）查看表的结构

```
hive> desc student;
```

（8）向表中插入数据

```
hive> insert into student values(1000,"ss");
```

（9）查询表中数据

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

（10）退出 hive

```
hive> quit;
```

2.3 将本地文件导入 Hive 案例

需求

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

1．数据准备

在/opt/module/data 这个目录下准备数据

（1）在/opt/module/目录下创建 data

```
[atguigu@hadoop102 module]$ mkdir data
```

（2）在/opt/module/datas/目录下创建 student.txt 文件并添加数据

```
[atguigu@hadoop102 datas]$ touch student.txt
[atguigu@hadoop102 datas]$ vi student.txt
1001 zhangshan
1002 lishi
1003 zhaoliu
```

注意以 tab 键间隔。

2. Hive 实际操作

（1）启动 hive

```
[atguigu@hadoop102 hive]$ bin/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;
```

（6）创建 student 表，并声明文件分隔符'\t'

```
hive> create table student(id int, name string) ROW FORMAT
DELIMITED FIELDS TERMINATED
 BY '\t';
```

（7）加载/opt/module/data/student.txt 文件到 student 数据库表中。

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

（8）Hive 查询结果

```
hive> select * from student;
OK
1001 zhangshan
1002 lishi
1003 zhaoliu
Time taken: 0.266 seconds, Fetched: 3 row(s)
```

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。

```
Exception in thread "main" java.lang.RuntimeException:
java.lang.RuntimeException:
 Unable to instantiate

org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClien
t
       at
org.apache.hadoop.hive.ql.session.SessionState.start(Session
State.java: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(NativeMethodAcce
```

```
ssorImpl.java:57)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMe
thodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        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.SessionHiveMetaStoreClien
t
        at
org.apache.hadoop.hive.metastore.MetaStoreUtils.newInstance(
MetaStoreUtils.java:1523)
        at
org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.<in
it>(RetryingMetaStoreClient.java:86)
        at
org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.get
Proxy(RetryingMetaStoreClient.java:132)
        at
org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.get
Proxy(RetryingMetaStoreClient.java:104)
        at
org.apache.hadoop.hive.ql.metadata.Hive.createMetaStoreClien
t(Hive.java:3005)
        at
org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:302
4)
        at
org.apache.hadoop.hive.ql.session.SessionState.start(Session
State.java: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

2.4.1 安装包准备

1. 查看 mysql 是否安装，如果安装了，卸载 mysql

（1）查看

```
[root@hadoop102 桌面]# rpm -qa|grep mysql
mysql-libs-5.1.73-7.el6.x86_64
```

（2）卸载

```
[root@hadoop102 桌面]# rpm -e --nodeps
mysql-libs-5.1.73-7.el6.x86_64
```

2. 解压 mysql-libs.zip 文件到当前目录

```
[root@hadoop102 software]# unzip mysql-libs.zip
[root@hadoop102 software]# ls
mysql-libs.zip
mysql-libs
```

3. 进入到 mysql-libs 文件夹下

```
[root@hadoop102 mysql-libs]# ll
```

```
总用量 76048
-rw-r--r--.  1  root  root  18509960  3  月  26  2015
MySQL-client-5.6.24-1.el6.x86_64.rpm
-rw-r--r--.  1  root  root  3575135  12  月  1  2013
mysql-connector-java-5.1.27.tar.gz
-rw-r--r--.  1  root  root  55782196  3  月  26  2015
MySQL-server-5.6.24-1.el6.x86_64.rpm
```

2.4.2 安装 MySql 服务器

1. 安装 mysql 服务端

```
[root@hadoop102 mysql-libs]# rpm -ivh
MySQL-server-5.6.24-1.el6.x86_64.rpm
```

2. 查看产生的随机密码

```
[root@hadoop102 mysql-libs]# cat /root/.mysql_secret
OEXaQuS8IWkG19Xs
```

3. 查看 mysql 状态

```
[root@hadoop102 mysql-libs]# service mysql status
```

4. 启动 mysql

```
[root@hadoop102 mysql-libs]# service mysql start
```

2.4.3 安装 MySql 客户端

1. 安装 mysql 客户端

```
[root@hadoop102 mysql-libs]# rpm -ivh
MySQL-client-5.6.24-1.el6.x86_64.rpm
```

2. 链接 mysql

```
[root@hadoop102 mysql-libs]# mysql -uroot -pOEXaQuS8IWkG19Xs
```

3. 修改密码

```
mysql>SET PASSWORD=PASSWORD('000000');
```

4. 退出 mysql

```
mysql>exit
```

2.4.4 MySql 中 user 表中主机配置

配置只要是 root 用户+密码，在任何主机上都能登录 MySQL 数据库。

1. 进入 mysql

```
[root@hadoop102 mysql-libs]# mysql -uroot -p000000
```

2. 显示数据库

```
mysql>show databases;
```

3. 使用 mysql 数据库

```
mysql>use mysql;
```

4. 展示 mysql 数据库中的所有表

```
mysql>show tables;
```

5. 展示 user 表的结构

```
mysql>desc user;
```

6. 查询 user 表

```
mysql>select User, Host, Password from user;
```

7. 修改 user 表，把 Host 表内容修改为%

```
mysql>update user set host='%' where host='localhost';
```

8. 删除 root 用户的其他 host

```
delete from user where Host='hadoop102';
delete from user where Host='127.0.0.1';
delete from user where Host='::1';
```

9. 刷新

```
mysql>flush privileges;
```

10. 退出

```
mysql>quit;
```

2.5 Hive 元数据配置到 MySql

2.5.1 驱动拷贝

1. 在/opt/software/mysql-libs 目录下解压 mysql-connector-java-5.1.27.tar.gz 驱动包

```
[root@hadoop102 mysql-libs]# tar -zxvf mysql-connector-java-5.1.27.tar.gz
```

2. 拷贝 mysql-connector-java-5.1.27-bin.jar 到/opt/module/hive/lib/

```
[root@hadoop102 mysql-connector-java-5.1.27]# cp /opt/software/mysql-libs/mysql-connector-java-5.1.27/mysql-connector-java-5.1.27-bin.jar/opt/module/hive/lib/
```

2.5.2 配置 Metastore 到 MySql

1. 在/opt/module/hive/conf 目录下创建一个 hive-site.xml

```
[atguigu@hadoop102 conf]$ touch hive-site.xml
[atguigu@hadoop102 conf]$ vi hive-site.xml
```

2. 根据官方文档配置参数，拷贝数据到 hive-site.xml 文件中

https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://hadoop102:3306/metastore?createDatabaseIfNotExist=true</value>
    <description>JDBC connect string for a JDBC metastore</description>
  </property>

  <property>
```

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

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

  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>000000</value>
    <description>password to use against metastore database</description>
  </property>
</configuration>
```

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

```
[atguigu@hadoop102 mysql-libs]$ mysql -uroot -p000000
```

查看有几个数据库

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

2．再次打开多个窗口，分别启动 hive

```
[atguigu@hadoop102 hive]$ bin/hive
```

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

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| metastore          |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

2.6 HiveJDBC 访问

2.6.1 启动 hiveserver2 服务

```
[atguigu@hadoop102 hive]$ bin/hiveserver2
```

2.6.2 启动 beeline

```
[atguigu@hadoop102 hive]$ bin/beeline
Beeline version 1.2.1 by Apache Hive
beeline>
```

2.6.3 连接 hiveserver2

```
beeline> !connect jdbc:hive2://hadoop102:10000（回车）
Connecting to jdbc:hive2://hadoop102:10000
Enter username for jdbc:hive2://hadoop102:10000: atguigu（回车）
Enter password for jdbc:hive2://hadoop102:10000: （直接回车）
Connected to: Apache Hive (version 1.2.1)
Driver: Hive JDBC (version 1.2.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hadoop102:10000> show databases;
+----------------+--+
| database_name  |
+----------------+--+
| default        |
| hive_db2       |
+----------------+--+
```

2.7 Hive 常用交互命令

```
[atguigu@hadoop102 hive]$ bin/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
                              commands. e.g. --hivevar A=B
 -i <filename>                   Initialization SQL file
 -S,--silent                      Silent mode in interactive
shell
 -v,--verbose                    Verbose mode (echo executed SQL
to the console)
```

1. “-e”不进入 hive 的交互窗口执行 sql 语句

```
[atguigu@hadoop102 hive]$ bin/hive -e "select id from student;"
```

2. “-f”执行脚本中 sql 语句

（1）在/opt/module/datas 目录下创建 hivef.sql 文件

```
[atguigu@hadoop102 datas]$ touch hivef.sql
```

文件中写入正确的 sql 语句

select * from student;

（2）执行文件中的 sql 语句

```
[atguigu@hadoop102 hive]$ bin/hive -f /opt/module/datas/hivef.sql
```

（3）执行文件中的 sql 语句并将结果写入文件中

```
[atguigu@hadoop102 hive]$ bin/hive -f /opt/module/datas/hivef.sql > /opt/module/datas/hive_result.txt
```

2.8 Hive 其他命令操作

1. 在 hive cli 命令窗口中如何查看 hdfs 文件系统

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

2. 在 hive cli 命令窗口中如何查看本地文件系统

```
hive> ! ls /opt/module/datas;
```

3. 查看在 hive 中输入的所有历史命令

（1）进入到当前用户的根目录/root 或/home/atguigu

（2）查看. hivehistory 文件

```
[atguigu@hadoop102 ~]$ 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

```
bin/hdfs dfs -chmod g+w /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>
```

```
</property>

<property>
  <name>hive.cli.print.current.db</name>
  <value>true</value>
</property>
```

2）重新启动 hive，对比配置前后差异。

（1）配置前，如图 6-2 所示

```
hive> select * from student;
OK
1001    xiaoli
1002    libingbing
1003    fanbingbing
Time taken: 0.318 seconds, Fetched: 3 row(s)
```

图 2-1 配置前

（2）配置后，如图 6-3 所示

```
hive (db_hive)> select * from db_hive.student;
OK
student.id      student.name
1001    xiaoli
1002    libingbing
1003    fanbingbing
```

图 2-2 配置后

2.9.3 Hive 运行日志信息配置

1. Hive 的 log 默认存放在/tmp/atguigu/hive.log 目录下（当前用户名下）
2. 修改 hive 的 log 存放日志到/opt/module/hive/logs

（1）修改/opt/module/hive/conf/hive-log4j.properties.template 文件名称为 hive-log4j.properties

```
[atguigu@hadoop102 conf]$ pwd
/opt/module/hive/conf
[atguigu@hadoop102 conf]$ mv hive-log4j.properties.template
hive-log4j.properties
```

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

hive.log.dir=/opt/module/hive/logs

2.9.4 参数配置方式

1. 查看当前所有的配置信息

```
hive>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：

```
[atguigu@hadoop103 hive]$ bin/hive -hiveconf mapred.reduce.tasks=10;
```

注意：仅对本次 hive 启动有效

查看参数设置：

```
hive (default)> set mapred.reduce.tasks;
```

（3）参数声明方式

可以在 HQL 中使用 SET 关键字设定参数

例如：

```
hive (default)> set mapred.reduce.tasks=100;
```

注意：仅对本次 hive 启动有效。

查看参数设置

```
hive (default)> set mapred.reduce.tasks;
```

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

表 3-1

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

表 3-2

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()
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()
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()

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。

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

```
hive (default)> load data local inpath "/opt/module/datas/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['xiaosong'],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。

第 4 章 DDL 数据定义

4.1 创建数据库

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';
```

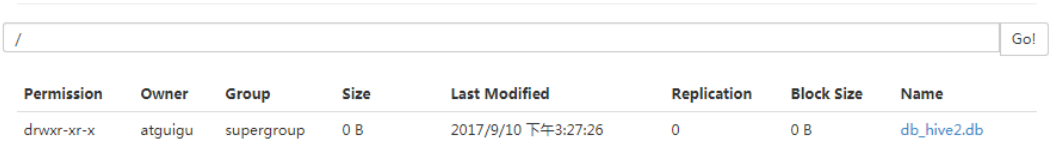

图 3-1 数据库存放位置

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://hadoop102:9000/user/hive/warehouse/db_hive.db
   atguiguUSER
```

2．显示数据库详细信息，extended

```
hive> desc database extended db_hive;
OK
```

```
db_hive
  hdfs://hadoop102:9000/user/hive/warehouse/db_hive.db
  atguiguUSER
```

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。数据库的其他元数据信息都是不可更改的，包括数据库名和数据库所在的目录位置。

```
hive (default)> alter database 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://hadoop102:8020/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]
```

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，在建表的同时指定一个指向实际数据的路径（LOCATION），Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。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

（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，目的是用于序列化和反序列化。

（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）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

（1）普通创建表

```
create table if not exists student2(
id int, name string
)
row format delimited fields terminated by '\t'
stored as textfile
location '/user/hive/warehouse/student2';
```

（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 student3 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 student4 like student;
```

（4）查询表的类型

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

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。

（1）原始数据

dept.txt　　emp.txt

（2）建表语句

创建部门表

```
create external table if not exists default.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 default.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;
OK
tab_name
dept
emp
```

（4）向外部表中导入数据

导入数据

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

查询结果

```
hive (default)> select * from emp;
hive (default)> select * from dept;
```

（5）查看表格式化数据

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

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 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。

4.6.1 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）

```
/user/hive/warehouse/log_partition/20170702/20170702.log
/user/hive/warehouse/log_partition/20170703/20170703.log
/user/hive/warehouse/log_partition/20170704/20170704.log
```

2. 创建分区表语法

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

3. 加载数据到分区表中

```
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept_partition partition(month='201709');
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept_partition partition(month='201708');
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept_partition partition(month='201707');
```

/user/hive/warehouse/dept_partition/month=201709 Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	82 B	2017/8/30 下午4:56:54	3	128 MB	dept.txt

图 4-1 加载数据到分区表

/user/hive/warehouse/dept_partition/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午5:01:52	0	0 B	month=201707
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午5:01:18	0	0 B	month=201708
drwxrwxr-x	atguigu	supergroup	0 B	2017/8/30 下午4:56:54	0	0 B	month=201709

图 4-2 分区表

4. 查询分区表中数据

单分区查询

```
hive (default)> select * from dept_partition where
```

```
month='201709';
```

多分区联合查询

```
hive (default)> select * from dept_partition where month='201709'
            union
            select * from dept_partition where month='201708'
            union
            select * from dept_partition where month='201707';

_u3.deptno      _u3.dname       _u3.loc _u3.month
10      ACCOUNTING      NEW YORK        201707
10      ACCOUNTING      NEW YORK        201708
10      ACCOUNTING      NEW YORK        201709
20      RESEARCH        DALLAS  201707
20      RESEARCH        DALLAS  201708
20      RESEARCH        DALLAS  201709
30      SALES   CHICAGO 201707
30      SALES   CHICAGO 201708
30      SALES   CHICAGO 201709
40      OPERATIONS      BOSTON  201707
40      OPERATIONS      BOSTON  201708
40      OPERATIONS      BOSTON  201709
```

5. 增加分区

创建单个分区

```
hive (default)> alter table dept_partition add partition(month='201706') ;
```

同时创建多个分区

```
hive (default)> alter table dept_partition add partition(month='201705') partition(month='201704');
```

6. 删除分区

删除单个分区

```
hive (default)> alter table dept_partition drop partition (month='201704');
```

同时删除多个分区

```
hive (default)> alter table dept_partition drop partition (month='201705'), partition (month='201706');
```

7. 查看分区表有多少分区

```
hive> show partitions dept_partition;
```

8. 查看分区表结构

```
hive> desc formatted dept_partition;

# Partition Information
# col_name              data_type               comment
month                   string
```

4.6.2 分区表注意事项

1. 创建二级分区表

```
hive (default)> create table dept_partition2(
            deptno int, dname string, loc string
            )
```

```
partitioned by (month string, day 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 '/opt/module/datas/dept.txt' into table default.dept_partition2 partition(month='201709', day='13');
```

（2）查询分区数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='13';
```

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/dept_partition2/month=201709/day=12;
hive (default)> dfs -put /opt/module/datas/dept.txt /user/hive/warehouse/dept_partition2/month=201709/day=12;
```

查询数据（查询不到刚上传的数据）

```
hive (default)> select * from dept_partition2 where month='201709' and day='12';
```

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 month='201709' and day='12';
```

（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/dept_partition2/month=201709/day=11;
hive (default)> dfs -put /opt/module/datas/dept.txt /user/hive/warehouse/dept_partition2/month=201709/day=11;
```

执行添加分区

```
hive (default)> alter table dept_partition2 add partition(month='201709', day='11');
```

查询数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='11';
```

（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/dept_partition2/month=201709/day=10;
```

上传数据

```
hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table dept_partition2 partition(month='201709',day='10');
```

查询数据

```
hive (default)> select * from dept_partition2 where month='201709' and day='10';
```

4.7 修改表

4.7.1 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;
```

4.7.2 增加、修改和删除表分区

详见 4.6.1 分区表基本操作。

4.7.3 增加/修改/替换列信息

1. 语法

更新列

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

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_partition;
```

（2）添加列

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

（3）查询表结构

```
hive> desc dept_partition;
```

（4）更新列

```
hive (default)> alter table dept_partition change column deptdesc desc int;
```

（5）查询表结构

```
hive> desc dept_partition;
```

（6）替换列

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

（7）查询表结构

```
hive> desc dept_partition;
```

4.8 删除表

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

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 '/opt/module/datas/student.txt' [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/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 /opt/module/datas/student.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 /opt/module/datas/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. 创建一张分区表

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

2. 基本插入数据

```
hive (default)> insert into table student partition(month='201709') values(1,'wangwu');
```

3. 基本模式插入（根据单张表查询结果）

```
hive (default)> insert overwrite table student partition(month='201708')
            select id, name from student where month='201709';
```

4. 多插入模式（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. 创建表，并指定在 hdfs 上的位置

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

2. 上传数据到 hdfs 上

```
hive (default)> dfs -put /opt/module/datas/student.txt /user/hive/warehouse/student5;
```

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 partition(month='201709') from
 '/user/hive/warehouse/export/student';
```

5.2 数据导出

5.2.1 Insert 导出

1. 将查询的结果导出到本地

```
hive (default)> insert overwrite local directory
'/opt/module/datas/export/student'
select * from student;
```

2. 将查询的结果格式化导出到本地

```
hive(default)>insert overwrite local directory
'/opt/module/datas/export/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/month=201709/000000_0
/opt/module/datas/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）

```
[atguigu@hadoop102 hive]$ bin/hive -e 'select * from
default.student;' >
/opt/module/datas/export/student4.txt;
```

5.2.4 Export 导出到 HDFS 上

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

5.2.5 Sqoop 导出

后续课程专门讲。

5.3 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：

```
[WITH CommonTableExpression (, CommonTableExpression)*] (Note:
Only available
starting with Hive 0.13.0)
```

```
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

1. 全表查询

```
hive (default)> select * 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

表 6-1

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

案例实操

查询出所有员工的薪水后加 1 显示。

```
hive (default)> select sal +1 from emp;
```

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;
```

6.2 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;
```

6.2.1 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。

表 6-2

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，其他的和等号（=）操作符的结果一致，如果任一为 NULL 则结果为 NULL
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，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 是一个正则表达式，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);
```

6.2.2 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）查找以 2 开头薪水的员工信息

```
hive (default)> select * from emp where sal LIKE '2%';
```

（2）查找第二个数值为 2 的薪水的员工信息

```
hive (default)> select * from emp where sal LIKE '_2%';
```

（3）查找薪水中含有 2 的员工信息

```
hive (default)> select * from emp where sal RLIKE '[2]';
```

6.2.3 逻辑运算符（And/Or/Not）

表 6-3

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

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);
```

6.3 分组

6.3.1 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。

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;
```

6.3.2 Having 语句

1．having 与 where 不同点

（1）where 针对表中的列发挥作用，查询数据；having 针对查询结果中的列发挥作用，筛选数据。

（2）where 后面不能写聚合函数，而 having 后面可以使用聚合函数。

（3）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;
```

6.4 Join 语句

6.4.1 等值 Join

Hive 支持通常的 SQL JOIN 语句，但是只支持等值连接，不支持非等值连接。

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;
```

6.4.2 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;
```

6.4.3 内连接

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。

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

6.4.4 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;
```

6.4.5 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;
```

6.4.6 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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

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

6.4.7 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

location.txt

1. 创建位置表

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

2. 导入数据

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

3. 多表连接查询

```
hive (default)>SELECT e.ename, d.deptno, 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。

6.4.8 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;
```

6.4.9 连接谓词中不支持 or

```
hive (default)> select
              >    e.empno,
              >    e.ename,
              >    d.deptno
              > from
              >    emp e
              > join
              >    dept d
              > on
              >    e.deptno=d.deptno or e.ename=d.dname;
FAILED: SemanticException [Error 10019]: Line 10:3 OR not
supported in JOIN currently 'dname'
```

6.5 排序

6.5.1 全局排序（Order By）

Order By：全局排序，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;
```

6.5.2 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;
```

6.5.3 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 ;
```

6.5.4 每个 MapReduce 内部排序（Sort By）

Sort By：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 empno 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
'/opt/module/datas/sortby-result'
 select * from emp sort by deptno desc;
```

6.5.5 分区排序（Distribute By）

Distribute By：类似 MR 中 partition，进行分区，结合 sort by 使用。

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

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。

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
'/opt/module/datas/distribute-result' select * from emp
distribute by deptno sort by empno desc;
```

6.5.6 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。

6.6 分桶及抽样查询

6.6.1 分桶表数据存储

分区针对的是数据的存储路径；分桶针对的是数据文件。

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，特别是之前所提到过的要确定合适的划分大小这个疑虑。

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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

1．先创建分桶表，通过直接导入数据文件的方式

（1）数据准备

student.txt

（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

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

（5）查看创建的分桶表中是否分成 4 个桶，如图 6-7 所示

Browse Directory

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	152 B	2017/9/3 下午4:41:49	3	128 MB	student.txt

图 6-1 未分桶

发现并没有分成 4 个桶。是什么原因呢？

2．创建分桶表时，数据通过子查询的方式导入

（1）先建一个普通的 stu 表

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

（2）向普通的 stu 表中导入数据

```
load data local inpath '/opt/module/datas/student.txt' into
table stu;
```

（3）清空 stu_buck 表中数据

```
truncate table stu_buck;
select * from stu_buck;
```

（4）导入数据到分桶表，通过子查询的方式

```
insert into table stu_buck
select id, name from stu;
```

（5）发现还是只有一个分桶，如图 6-2 所示

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午4:55:11	3	128 MB	000000_0

图 6-2 未分桶

（6）需要设置一个属性

```
hive (default)> set hive.enforce.bucketing=true;
hive (default)> set mapreduce.job.reduces=-1;
hive (default)> insert into table stu_buck
select id, name from stu;
```

Browse Directory

/user/hive/warehouse/stu_buck Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:53	3	128 MB	000000_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:52	3	128 MB	000001_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:53	3	128 MB	000002_0
-rwxrwxr-x	atguigu	supergroup	0 B	2017/9/3 下午5:00:54	3	128 MB	000003_0

图 6-3 分桶

（7）查询分桶的数据

```
hive (default)> select * from stu_buck;
OK
stu_buck.id     stu_buck.name
1004    ss4
1008    ss8
1012    ss12
1016    ss16
1001    ss1
1005    ss5
1009    ss9
1013    ss13
1002    ss2
1006    ss6
1010    ss10
1014    ss14
1003    ss3
1007    ss7
1011    ss11
1015    ss15
```

6.6.2 分桶抽样查询

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。

查询表 stu_buck 中的数据。

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

注：tablesample 是抽样语句，语法：TABLESAMPLE(BUCKET x OUT OF y) 。

y 必须是 table 总 bucket 数的倍数或者因子。hive 根据 y 的大小，决定抽样的比例。例如，table 总共分了 4 份，当 y=2 时，抽取(4/2=)2 个 bucket 的数据，当 y=8 时，抽取(4/8=)1/2 个 bucket 的数据。

x 表示从哪个 bucket 开始抽取，如果需要取多个分区，以后的分区号为当前分区号加上

y。例如，table 总 bucket 数为 4，tablesample(bucket 1 out of 2)，表示总共抽取（4/2=）2 个 bucket 的数据，抽取第 1(x)个和第 3(x+y)个 bucket 的数据。

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

6.7 其他常用查询函数

6.7.1 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(string1, replace_with)。它的功能是如果 string1 为 NULL，则 NVL 函数返回 replace_with 的值，否则返回 string1 的值，如果两个参数都为 NULL ，则返回 NULL。

2）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 nvl(comm,-1) from emp;
OK
_c0
20.0
300.0
500.0
-1.0
1400.0
-1.0
-1.0
-1.0
-1.0
0.0
-1.0
-1.0
-1.0
-1.0
```

4）查询：如果员工的 comm 为 NULL，则用领导 id 代替

```
hive (default)> select nvl(comm,mgr) from emp;
OK
_c0
20.0
300.0
500.0
7839.0
1400.0
7839.0
7839.0
```

```
7566.0
NULL
0.0
7788.0
7698.0
7566.0
```

6.7.2 时间类

1）date_format:格式化时间

```
hive (default)> select date_format('2019-06-29','yyyy-MM-dd');
OK
_c0
2019-06-29
```

2）date_add:时间跟天数相加

```
hive (default)> select date_add('2019-06-29',5);
OK
_c0
2019-07-04

hive (default)> select date_add('2019-06-29',-5);
OK
_c0
2019-06-24
```

3）date_sub:时间跟天数相减

```
hive (default)> select date_sub('2019-06-29',5);
OK
_c0
2019-06-24

hive (default)> select date_sub('2019-06-29 12:12:12',5);
OK
_c0
2019-06-24

hive (default)> select date_sub('2019-06-29',-5);
OK
_c0
2019-07-04
```

4）datediff:两个时间相减

```
hive (default)> select datediff('2019-06-29','2019-06-24');
OK
_c0
5

hive (default)> select datediff('2019-06-24','2019-06-29');
OK
_c0
```

```
-5

hive (default)> select datediff('2019-06-24 12:12:12','2019-06-29');
OK
_c0
-5

hive (default)> select datediff('2019-06-24 12:12:12','2019-06-29
13:13:13');
OK
_c0
-5
```

6.7.3 CASE WHEN

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：

```
A     2       1
B     1       2
```

3. 创建本地 emp_sex.txt，添加数据

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 '/opt/module/data/emp_sex.txt' into table
emp_sex;
```

5. 按需求查询数据

```
select
  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;
```

6.7.4 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;

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

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
```

4. 创建本地 constellation.txt，导入数据

```
[atguigu@hadoop102 datas]$ vi 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
```

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 "\t";

load data local inpath "/opt/module/data/person_info.txt" into
```

```
table person_info;
```

6. 按需求查询数据

```
select
    t1.base,
    concat_ws('|', collect_set(t1.name)) name
from
    (select
        name,
        concat(constellation, ",", blood_type) base
    from
        person_info) t1
group by
    t1.base;
```

6.7.5 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

```
[atguigu@hadoop102 datas]$ vi movie.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 array<string>)
row format delimited fields terminated by "\t"
collection items terminated by ",";

load data local inpath "/opt/module/datas/movie.txt" into table
movie_info;
```

6. 按需求查询数据

```
select
    movie,
    category_name
from
    movie_info  lateral  view  explode(category)  table_tmp  as
category_name;
```

6.7.6 窗口函数

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化而变化；

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)：往前第 n 行数据；

LEAD(col,n)：往后第 n 行数据；

NTILE(n)：把有序分区中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购买总额

（3）上述的场景,要将 cost 按照日期进行累加

（4）查询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

```
[atguigu@hadoop102 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 "/opt/module/datas/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）上述的场景,要将 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;
```

（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;
```

6.7.7 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 movie.txt，导入数据

```
[atguigu@hadoop102 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 '/opt/module/datas/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;

name    subject score   rp      drp     rmp
孙悟空  数学    95      1       1       1
宋宋    数学    86      2       2       2
婷婷    数学    85      3       3       3
大海    数学    56      4       4       4
宋宋    英语    84      1       1       1
大海    英语    84      1       1       2
婷婷    英语    78      3       2       3
孙悟空  英语    68      4       3       4
大海    语文    94      1       1       1
孙悟空  语文    87      2       2       2
婷婷    语文    65      3       3       3
宋宋    语文    64      4       4       4
```

第 7 章 函数

7.1 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;
```

7.2 自定义函数

1）Hive 自带了一些函数，比如：max/min 等，但是数量有限，自己可以通过自定义 UDF

来方便的扩展。

2）当 Hive 提供的内置函数无法满足你的业务处理需要时，此时就可以考虑使用用户自定义函数（UDF：user-defined function）。

3）根据用户自定义函数类别分为以下三种：

（1）UDF（User-Defined-Function）

一进一出

（2）UDAF（User-Defined Aggregation Function）

聚集函数，多进一出

类似于：count/max/min

（3）UDTF（User-Defined Table-Generating Functions）

一进多出

如 lateral view explore()

4）官方文档地址

https://cwiki.apache.org/confluence/display/Hive/HivePlugins

5）编程步骤

（1）继承 org.apache.hadoop.hive.ql.UDF

（2）需要实现 evaluate 函数；evaluate 函数支持重载；

（3）在 hive 的命令行窗口创建函数

a）添加 jar

```
add jar linux_jar_path
```

b）创建 function

```
create [temporary] function [dbname.]function_name AS class_name;
```

（4）在 hive 的命令行窗口删除函数

```
Drop [temporary] function [if exists] [dbname.]function_name;
```

6）注意事项

（1）UDF 必须要有返回类型，可以返回 null，但是返回类型不能为 void；

7.2.1 自定义 UDF 函数

1）创建一个 Maven 工程 Hive

2）导入依赖

```
<dependencies>
        <!--
```

```
https://mvnrepository.com/artifact/org.apache.hive/hive-exec -->
       <dependency>
           <groupId>org.apache.hive</groupId>
           <artifactId>hive-exec</artifactId>
           <version>1.2.1</version>
       </dependency>
</dependencies>
```

3）创建一个类

```
package com.atguigu.hive;
import org.apache.hadoop.hive.ql.exec.UDF;

public class Lower extends UDF {

    public String evaluate (String s) {

        if (s == null) {
            return null;
        }

        return s.toLowerCase();
    }
}
```

4）打成 jar 包上传到服务器/opt/module/datas/udf.jar

5）将 jar 包添加到 hive 的 classpath

```
hive (default)> add jar /opt/module/datas/udf.jar;
```

6）创建临时函数与开发好的 java class 关联

```
hive (default)> create temporary function mylower as "com.atguigu.hive.Lower";
```

7. 即可在 hql 中使用自定义的函数

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

7.2.2 自定义 UDTF 函数

1）需求说明

自定义一个 UDTF 实现将一个任意分割符的字符串切割成独立的单词，例如：

```
Line:"hello,world,hadoop,hive"

Myudtf(line, ",")

hello
world
hadoop
hive
```

2）代码实现

```
package com.atguigu.udtf;

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.ObjectInspectorFact
ory;
import
org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspect
or;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.Primitive
ObjectInspectorFactory;

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

public class MyUDTF 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.javaStringObjectInsp
ector);

        return
ObjectInspectorFactory.getStandardStructObjectInspector(fieldName
s, 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 {

    }
}
```

3）打成 jar 包上传到服务器/opt/module/data/udtf.jar

4）将 jar 包添加到 hive 的 classpath 下

```
hive (default)> add jar /opt/module/data/udtf.jar;
```

5）创建临时函数与开发好的 java class 关联

```
hive (default)> create temporary function myudtf as "com.atguigu.hive.MyUDTF";
```

6）即可在 hql 中使用自定义的函数

```
hive (default)> select myudtf(line, ",") word from words;
```

第 8 章 压缩和存储

8.1 Hadoop 源码编译支持 Snappy 压缩

8.1.1 资源准备

1. CentOS 联网

配置 CentOS 能连接外网。Linux 虚拟机 ping www.baidu.com 是畅通的

注意：采用 root 角色编译，减少文件夹权限出现问题

2. jar 包准备(hadoop 源码、JDK8 、maven、protobuf)

（1）hadoop-2.7.2-src.tar.gz

（2）jdk-8u144-linux-x64.tar.gz

（3）snappy-1.1.3.tar.gz

（4）apache-maven-3.0.5-bin.tar.gz

（5）protobuf-2.5.0.tar.gz

8.1.2 jar 包安装

注意：所有操作必须在 root 用户下完成

1. JDK 解压、配置环境变量 JAVA_HOME 和 PATH，验证 java-version(如下都需要验证是否配置成功)

```
[root@hadoop101 software] # tar -zxf jdk-8u144-linux-x64.tar.gz -C /opt/module/
[root@hadoop101 software]# vi /etc/profile
#JAVA_HOME
export JAVA_HOME=/opt/module/jdk1.8.0_144
export PATH=$PATH:$JAVA_HOME/bin
[root@hadoop101 software]#source /etc/profile
```

验证命令：java -version

2. Maven 解压、配置 MAVEN_HOME 和 PATH

```
[root@hadoop101 software]# tar -zxvf apache-maven-3.0.5-bin.tar.gz -C /opt/module/
[root@hadoop101 apache-maven-3.0.5]# vi /etc/profile
#MAVEN_HOME
export MAVEN_HOME=/opt/module/apache-maven-3.0.5
export PATH=$PATH:$MAVEN_HOME/bin
[root@hadoop101 software]#source /etc/profile
```

验证命令：mvn -version

8.1.3 编译源码

1. 准备编译环境

```
[root@hadoop101 software]# yum install svn
[root@hadoop101 software]# yum install autoconf automake libtool cmake
[root@hadoop101 software]# yum install ncurses-devel
[root@hadoop101 software]# yum install openssl-devel
[root@hadoop101 software]# yum install gcc*
```

2. 编译安装 snappy

```
[root@hadoop101 software]# tar -zxvf snappy-1.1.3.tar.gz -C /opt/module/
[root@hadoop101 module]# cd snappy-1.1.3/
[root@hadoop101 snappy-1.1.3]# ./configure
[root@hadoop101 snappy-1.1.3]# make
[root@hadoop101 snappy-1.1.3]# make install
# 查看 snappy 库文件
[root@hadoop101 snappy-1.1.3]# ls -lh /usr/local/lib |grep snappy
```

3. 编译安装 protobuf

```
[root@hadoop101 software]# tar -zxvf protobuf-2.5.0.tar.gz -C /opt/module/
[root@hadoop101 module]# cd protobuf-2.5.0/
[root@hadoop101 protobuf-2.5.0]# ./configure
[root@hadoop101 protobuf-2.5.0]# make
[root@hadoop101 protobuf-2.5.0]# make install
# 查看 protobuf 版本以测试是否安装成功
[root@hadoop101 protobuf-2.5.0]# protoc --version
```

4. 编译 hadoop native

```
[root@hadoop101 software]# tar -zxvf hadoop-2.7.2-src.tar.gz
[root@hadoop101 software]# cd hadoop-2.7.2-src/
[root@hadoop101 software]# mvn clean package -DskipTests -Pdist,native -Dtar -Dsnappy.lib=/usr/local/lib -Dbundle.snappy
```

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

执行成功后，/opt/software/hadoop-2.7.2-src/hadoop-dist/target/hadoop-2.7.2.tar.gz 即为新生成的支持 snappy 压缩的二进制安装包。

8.2 Hadoop 压缩配置

8.2.1 MR 支持的压缩编码

表 8-1

压缩格式	工具	算法	文件扩展名	是否可切分
DEFAULT	无	DEFAULT	.deflate	否
Gzip	gzip	DEFAULT	.gz	否
bzip2	bzip2	bzip2	.bz2	是
LZO	lzop	LZO	.lzo	是
Snappy	无	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：

表 8-2

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：

表 8-3

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.

8.2.2 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）：

表 8-4

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	Sequence File 输出使用的压缩类型：NONE 和 BLOCK

8.3 开启 Map 输出阶段压缩

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：

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;
```

8.4 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。

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/datas/distribute-result' select * from emp distribute by deptno sort by empno desc;
```

8.5 文件存储格式

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

8.5.1 列式存储和行式存储

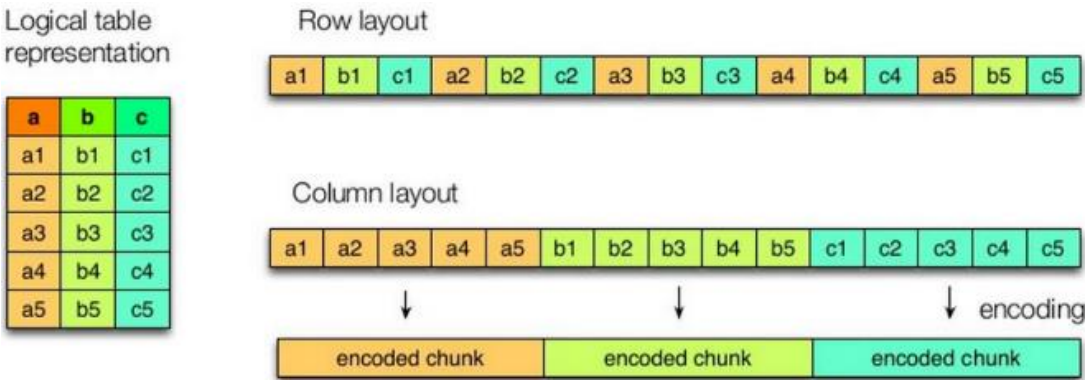

图 8-1 列式存储和行式存储

如图 8-1 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。

8.5.2 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。

8.5.3 Orc 格式

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

如图 8-2 所示可以看到每个 Orc 文件由 1 个或多个 stripe 组成，每个 stripe250MB 大小，这个 Stripe 实际相当于 RowGroup 概念，不过大小由 4MB->250MB，这样应该能提升顺序读的吞吐率。每个 Stripe 里有三部分组成，分别是 Index Data，Row Data，Stripe Footer：

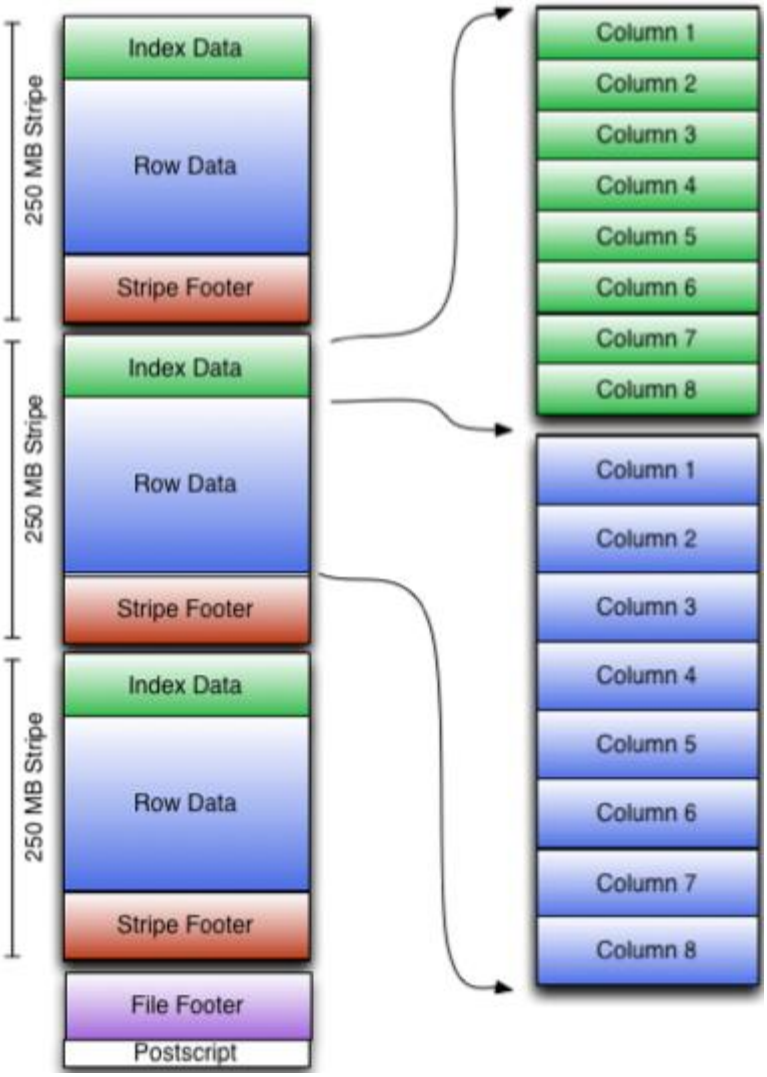

图 8-2 Orc 格式

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。

8.5.4 Parquet 格式

Parquet 是面向分析型业务的列式存储格式，由 Twitter 和 Cloudera 合作开发，2015 年 5 月从 Apache 的孵化器里毕业成为 Apache 顶级项目。

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。

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 文件的格式如图 8-3 所示。

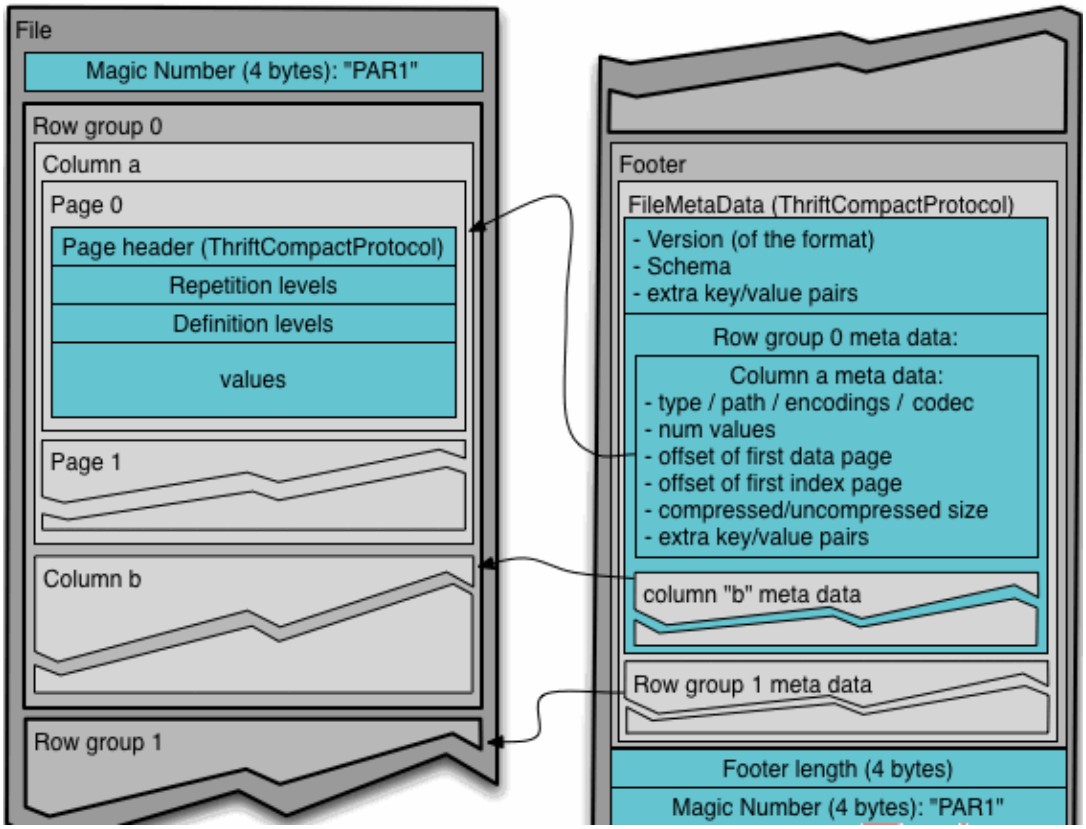

图 8-3 Parquet 格式

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

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

页。

8.5.5 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.data

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/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;
```

18.1 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 ;
```

（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/ ;
```

2.8 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

存储文件的压缩比总结：

ORC > Parquet > textFile

存储文件的查询速度测试：

1. TextFile

```
hive (default)> select count(*) from log_text;
_c0
100000
Time taken: 21.54 seconds, Fetched: 1 row(s)
Time taken: 21.08 seconds, Fetched: 1 row(s)
Time taken: 19.298 seconds, Fetched: 1 row(s)
Time taken: 25.544 seconds, Fetched: 1 row(s)
```

2. ORC

```
hive (default)> select count(*) from log_orc;
_c0
100000
Time taken: 20.867 seconds, Fetched: 1 row(s)
Time taken: 22.667 seconds, Fetched: 1 row(s)
Time taken: 18.36 seconds, Fetched: 1 row(s)
Time taken: 20.768 seconds, Fetched: 1 row(s)
```

3. Parquet

```
hive (default)> select count(*) from log_parquet;
_c0
100000
Time taken: 22.922 seconds, Fetched: 1 row(s)
Time taken: 21.074 seconds, Fetched: 1 row(s)
Time taken: 18.384 seconds, Fetched: 1 row(s)
Time taken: 21.53 seconds, Fetched: 1 row(s)
```

存储文件的查询速度总结：查询速度相近。

8.6 存储和压缩结合

8.6.1 修改 Hadoop 集群具有 Snappy 压缩方式

1. 查看 hadoop checknative 命令使用

```
[atguigu@hadoop104 hadoop-2.7.2]$ hadoop checknative [-a|-h] check native hadoop and compression libraries availability
```

2. 查看 hadoop 支持的压缩方式

```
[atguigu@hadoop104 hadoop-2.7.2]$ hadoop checknative
17/12/24 20:32:52 WARN bzip2.Bzip2Factory: Failed to load/initialize native-bzip2 library system-native, will use pure-Java version
17/12/24 20:32:52 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
Native library checking:
hadoop: true /opt/module/hadoop-2.7.2/lib/native/libhadoop.so
zlib: true /lib64/libz.so.1
snappy: false
lz4: true revision:99
bzip2: false
```

3. 将编译好的支持 Snappy 压缩的 hadoop-2.7.2.tar.gz 包导入到 hadoop102 的 /opt/software 中

4. 解压 hadoop-2.7.2.tar.gz 到当前路径

```
[atguigu@hadoop102 software]$ tar -zxvf hadoop-2.7.2.tar.gz
```

5. 进入到/opt/software/hadoop-2.7.2/lib/native 路径可以看到支持 Snappy 压缩的动态链接库

```
[atguigu@hadoop102 native]$ pwd
/opt/software/hadoop-2.7.2/lib/native
[atguigu@hadoop102 native]$ ll
-rw-r--r--. 1 atguigu atguigu 472950 9 月 1 10:19 libsnappy.a
-rwxr-xr-x. 1 atguigu atguigu 955 9 月 1 10:19 libsnappy.la
lrwxrwxrwx. 1 atguigu atguigu 18 12 月 24 20:39 libsnappy.so -> libsnappy.so.1.3.0
lrwxrwxrwx. 1 atguigu atguigu 18 12 月 24 20:39 libsnappy.so.1 -> libsnappy.so.1.3.0
-rwxr-xr-x. 1 atguigu atguigu 228177 9 月 1 10:19 libsnappy.so.1.3.0
```

6. 拷贝/opt/software/hadoop-2.7.2/lib/native 里面的所有内容到开发集群的 /opt/module/hadoop-2.7.2/lib/native 路径上

```
[atguigu@hadoop102 native]$ cp ../native/* /opt/module/hadoop-2.7.2/lib/native/
```

7. 分发集群

```
[atguigu@hadoop102 lib]$ xsync native/
```

8. 再次查看 hadoop 支持的压缩类型

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop checknative
17/12/24 20:45:02 WARN bzip2.Bzip2Factory: Failed to load/initialize native-bzip2 library system-native, will use
```

```
pure-Java version
17/12/24 20:45:02 INFO zlib.ZlibFactory: Successfully loaded &
initialized native-zlib library
Native library checking:
hadoop:                                                    true
/opt/module/hadoop-2.7.2/lib/native/libhadoop.so
zlib:    true /lib64/libz.so.1
snappy:                                                    true
/opt/module/hadoop-2.7.2/lib/native/libsnappy.so.1
lz4:     true revision:99
bzip2:   false
```

9. 重新启动 hadoop 集群和 hive

8.6.2 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：

表 8-5

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	67,108,864	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)

1. 创建一个非压缩的的 ORC 存储方式

（1）建表语句

```
create table log_orc_none(
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");
```

（2）插入数据

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

（3）查看插入后数据

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

7.7 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

```
hive (default)> insert into table 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/ ;
```

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

3．上一节中默认创建的 ORC 存储方式，导入数据后的大小为

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

比 Snappy 压缩的还小。原因是 orc 存储文件默认采用 ZLIB 压缩。比 snappy 压缩的小。

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 9 章 企业级调优

9.1 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>
```

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;
```

9.2 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;
```

案例实操：

1）开启本地模式，并执行查询语句

```
hive (default)> set hive.exec.mode.local.auto=true;
hive (default)> select * from emp cluster by deptno;
Time taken: 1.328 seconds, Fetched: 14 row(s)
```

2）关闭本地模式，并执行查询语句

```
hive (default)> set hive.exec.mode.local.auto=false;
hive (default)> select * from emp cluster by deptno;
Time taken: 20.09 seconds, Fetched: 14 row(s)
```

9.3 表的优化

9.3.1 小表、大表 Join

将 key 相对分散，并且数据量小的表放在 join 的左边，这样可以有效减少内存溢出错误发生的几率；再进一步，可以使用 map join 让小的维度表（1000 条以下的记录条数）先进内存。在 map 端完成 reduce。

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明显区别。

案例实操

1. 需求

测试大表 JOIN 小表和小表 JOIN 大表的效率

2. 建大表、小表和 JOIN 后表的语句

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

3. 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;
```

4. 关闭 mapjoin 功能（默认是打开的）

```
set hive.auto.convert.join = false;
```

5. 执行小表 JOIN 大表语句

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

Time taken: 35.921 seconds

Time taken: 43.152 seconds

6. 执行大表 JOIN 小表语句

```
insert overwrite table jointable
select b.id, b.time, b.uid, b.keyword, b.url_rank, b.click_num,
```

```
b.click_url
from bigtable  b
left join smalltable  s
on s.id = b.id;
```

Time taken: 34.196 seconds

Time taken: 40.807 seconds

9.3.2 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
```

查看 jobhistory

http://hadoop102:19888/jobhistory

（2）创建原始数据表、空 id 表、合并后数据表

```
// 创建原始表
create table ori(id bigint, time bigint, uid string, keyword string, url_rank int, click_num int, click_url string) row format delimited fields terminated by '\t';
// 创建空 id 表
create table nullidtable(id bigint, time 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, time 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/datas/SogouQ1.txt' into table ori;
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 ori o on n.id = o.id;
```

Time taken: 42.038 seconds

Time taken: 37.284 seconds

Time taken: 41.743 seconds

（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 ori o on n.id = o.id;
```

Time taken: 31.725 seconds

Time taken: 28.876 seconds

Time taken: 33.29 seconds

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 ori b on n.id = b.id;
```

结果：如图 6-13 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

图 9-1 空 key 转换

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 ori o on
case when n.id is null then concat('hive', rand()) else n.id end
= o.id;
```

结果：如图 6-14 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	Fi
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

图 9-2 随机分布空值

9.3.3 MapJoin

如果不指定 MapJoin 或者不符合 MapJoin 的条件，那么 Hive 解析器会将 Join 操作转换成 Common Join，即：在 Reduce 阶段完成 join。容易发生数据倾斜。可以用 MapJoin 把小表全部加载到内存在 map 端进行 join，避免 reducer 处理。

1．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 一下认为是小表）：

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

2．MapJoin 工作机制，如图 6-15 所示

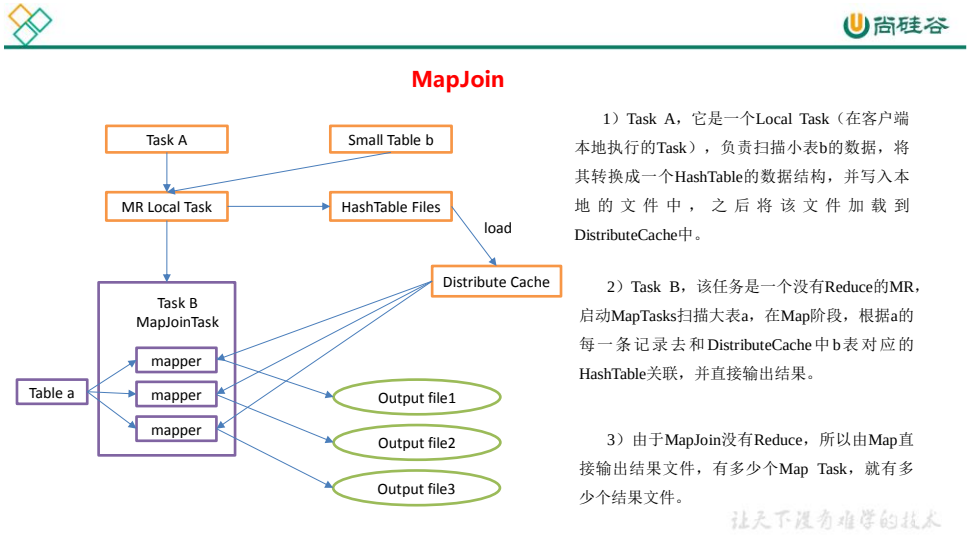

图 9-3　MapJoin 工作机制

案例实操：

（1）开启 Mapjoin 功能

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

（2）执行小表 JOIN 大表语句

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

Time taken: 24.594 seconds

Time taken: 26.212 seconds

（3）执行大表 JOIN 小表语句

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

Time taken: 24.315 seconds

Time taken: 24.498 seconds

9.3.4 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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

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

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

（2）在 Map 端进行聚合操作的条目数目

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

（3）有数据倾斜的时候进行负载均衡（默认是 false）

```
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。

9.3.5 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：

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/datas/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)
Time taken: 39.753 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 sec
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)
Time taken: 63.108 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。

9.3.6 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。

9.3.7 行列过滤

列处理：在 SELECT 中，只拿需要的列，如果有，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 ori 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 ori where id <= 10 ) o on b.id = o.id;
```

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

9.3.8 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。

```
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。

```
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。

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

2．案例实操

需求：将 ori 中的数据按照时间(如：20111230000008)，插入到目标表 ori_partitioned_target 的相应分区中。

（1）创建分区表

```
create table ori_partitioned(id bigint, time bigint, uid
string, keyword string,
 url_rank int, click_num int, click_url string)
partitioned by (p_time bigint)
row format delimited fields terminated by '\t';
```

（2）加载数据到分区表中

```
hive (default)> load data local inpath '/home/atguigu/ds1' into
table
 ori_partitioned partition(p_time='20111230000010') ;
hive (default)> load data local inpath '/home/atguigu/ds2' into
table ori_partitioned partition(p_time='20111230000011') ;
```

（3）创建目标分区表

```
create table ori_partitioned_target(id bigint, time bigint,
uid string,
 keyword string, url_rank int, click_num int, click_url string)
PARTITIONED BY (p_time STRING) row format delimited fields
terminated by '\t';
```

（4）设置动态分区

```
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions = 1000;
set hive.exec.max.dynamic.partitions.pernode = 100;
set hive.exec.max.created.files = 100000;
set hive.error.on.empty.partition = false;

hive (default)> insert overwrite table ori_partitioned_target
partition (p_time)
select id, time, uid, keyword, url_rank, click_num, click_url,
p_time from ori_partitioned;
```

（6）查看目标分区表的分区情况

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

9.3.9 分桶

详见 6.6 章。

9.3.10 分区

详见 4.6 章。

9.4 MR 优化

9.4.1 合理设置 Map 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，但这个文件只有一个或者两个小字段，却有几千万的记录，如果 map 处理的逻辑比较复杂，用一个 map 任务去做，肯定也比较耗时。

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；

9.4.2 小文件进行合并

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;
```

9.4.3 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
```

9.4.4 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

设置每个 job 的 Reduce 个数

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

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；

9.5 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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

9.6 严格模式

Hive 提供了一个严格模式，可以防止用户执行那些可能意想不到的不好的影响的查询。

通过设置属性 hive.mapred.mode 值为默认是非严格模式 nonstrict 。开启严格模式需要修改 hive.mapred.mode 值为 strict，开启严格模式可以禁止 3 种类型的查询。

```
<property>
    <name>hive.mapred.mode</name>
    <value>strict</value>
    <description>
      The mode in which the Hive operations are being performed.
      In strict mode, some risky queries are not allowed to run. They
include:
        Cartesian Product.
        No partition being picked up for a query.
        Comparing bigints and strings.
        Comparing bigints and doubles.
        Orderby without limit.
</description>
</property>
```

1) 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) 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) 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。

9.7 JVM 重用

JVM 重用是 Hadoop 调优参数的内容，其对 Hive 的性能具有非常大的影响，特别是对于很难避免小文件的场景或 task 特别多的场景，这类场景大多数执行时间都很短。

Hadoop 的默认配置通常是使用派生 JVM 来执行 map 和 Reduce 任务的。这时 JVM 的启动过程可能会造成相当大的开销，尤其是执行的 job 包含有成百上千 task 任务的情况。JVM 重用可以使得 JVM 实例在同一个 job 中重新使用 N 次。N 的值可以在 Hadoop 的

mapred-site.xml 文件中进行配置。通常在 10-20 之间，具体多少需要根据具体业务场景测试得出。

```
<property>
   <name>mapreduce.job.jvm.numtasks</name>
   <value>10</value>
   <description>How many tasks to run per jvm. If set to -1, there is
  no limit.
   </description>
</property>
```

这个功能的缺点是，开启 JVM 重用将一直占用使用到的 task 插槽，以便进行重用，直到任务完成后才能释放。如果某个“不平衡的”job 中有某几个 reduce task 执行的时间要比其他 Reduce task 消耗的时间多的多的话，那么保留的插槽就会一直空闲着却无法被其他的 job 使用，直到所有的 task 都结束了才会释放。

9.8 推测执行

在分布式集群环境下，因为程序 Bug（包括 Hadoop 本身的 bug），负载不均衡或者资源分布不均等原因，会造成同一个作业的多个任务之间运行速度不一致，有些任务的运行速度可能明显慢于其他任务（比如一个作业的某个任务进度只有 50%，而其他所有任务已经运行完毕），则这些任务会拖慢作业的整体执行进度。为了避免这种情况发生，Hadoop 采用了推测执行（Speculative Execution）机制，它根据一定的法则推测出“拖后腿”的任务，并为这样的任务启动一个备份任务，让该任务与原始任务同时处理同一份数据，并最终选用最先成功运行完成任务的计算结果作为最终结果。

设置开启推测执行参数：Hadoop 的 mapred-site.xml 文件中进行配置

```
<property>
  <name>mapreduce.map.speculative</name>
  <value>true</value>
  <description>If true, then multiple instances of some map tasks
             may be executed in parallel.</description>
</property>

<property>
  <name>mapreduce.reduce.speculative</name>
  <value>true</value>
  <description>If true, then multiple instances of some reduce tasks
             may be executed in parallel.</description>
</property>
```

不过 hive 本身也提供了配置项来控制 reduce-side 的推测执行：

```
  <property>
    <name>hive.mapred.reduce.tasks.speculative.execution</name>
    <value>true</value>
    <description>Whether speculative execution for reducers should
be turned on. </description>
  </property>
```

关于调优这些推测执行变量，还很难给一个具体的建议。如果用户对于运行时的偏差非常敏感的话，那么可以将这些功能关闭掉。如果用户因为输入数据量很大而需要执行长时间的 map 或者 Reduce task 的话，那么启动推测执行造成的浪费是非常巨大大。

9.9 压缩

详见第 8 章。

9.10 执行计划（Explain）

1. 基本语法

EXPLAIN [EXTENDED | DEPENDENCY | AUTHORIZATION] query

2. 案例实操

（1）查看下面这条语句的执行计划

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

（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;
```

第 10 章 Hive 实战之谷粒影音

10.1 需求描述

统计硅谷影音视频网站的常规指标，各种 TopN 指标：
--统计视频观看数 Top10
--统计视频类别热度 Top10
--统计视频观看数 Top20 所属类别以及类别包含的 Top20 的视频个数
--统计视频观看数 Top50 所关联视频的所属类别 Rank
--统计每个类别中的视频热度 Top10
--统计每个类别中视频流量 Top10
--统计上传视频最多的用户 Top10 以及他们上传的观看次数在前 20 视频
--统计每个类别视频观看数 Top10

10.2 项目

10.2.1 数据结构

1. 视频表

表 6-13 视频表

字段	备注	详细描述
video id	视频唯一 id	11 位字符串
uploader	视频上传者	上传视频的用户名 String

age	视频年龄	视频在平台上的整数天
category	视频类别	上传视频指定的视频分类
length	视频长度	整形数字标识的视频长度
views	观看次数	视频被浏览的次数
rate	视频评分	满分 5 分
ratings	流量	视频的流量，整型数字
conments	评论数	一个视频的整数评论数
related ids	相关视频 id	相关视频的 id，最多 20 个

2．用户表

表 6-14 用户表

字段	备注	字段类型
uploader	上传者用户名	string
videos	上传视频数	int
friends	朋友数量	int

10.2.2 ETL 原始数据

通过观察原始数据形式，可以发现，视频可以有多个所属分类，每个所属分类用&符号分割，且分割的两边有空格字符，同时相关视频也是可以有多个元素，多个相关视频又用“\t”进行分割。为了分析数据时方便对存在多个子元素的数据进行操作，我们首先进行数据重组清洗操作。即：将所有的类别用“&”分割，同时去掉两边空格，多个相关视频 id 也使用“&”进行分割。

1．ETL 之 ETLUtil

```
public class ETLUtil {
    public static String oriString2ETLString(String ori){
        StringBuilder etlString = new StringBuilder();
        String[] splits = ori.split("\t");
        if(splits.length < 9) return null;
        splits[3] = splits[3].replace(" ", "");
        for(int i = 0; i < splits.length; i++){
            if(i < 9){
                if(i == splits.length - 1){
                    etlString.append(splits[i]);
                }else{
                    etlString.append(splits[i] + "\t");
                }
            }else{
                if(i == splits.length - 1){
                    etlString.append(splits[i]);
                }else{
                    etlString.append(splits[i] + "&");
                }
            }
        }

        return etlString.toString();
```

```
    }
}
```

2. ETL 之 Mapper

```
import java.io.IOException;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import com.atguigu.util.ETLUtil;

public class VideoETLMapper extends Mapper<Object, Text,
NullWritable, Text>{
    Text text = new Text();

    @Override
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String etlString =
ETLUtil.oriString2ETLString(value.toString());

        if(StringUtils.isBlank(etlString)) return;

        text.set(etlString);
        context.write(NullWritable.get(), text);
    }
}
```

3. ETL 之 Runner

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class VideoETLRunner implements Tool {
    private Configuration conf = null;

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
    }

    @Override
    public Configuration getConf() {
        return this.conf;
    }

    @Override
```

```
    public int run(String[] args) throws Exception {
        conf = this.getConf();
        conf.set("inpath", args[0]);
        conf.set("outpath", args[1]);

        Job job = Job.getInstance(conf);

        job.setJarByClass(VideoETLRunner.class);

        job.setMapperClass(VideoETLMapper.class);
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);
        job.setNumReduceTasks(0);

        this.initJobInputPath(job);
        this.initJobOutputPath(job);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    private void initJobOutputPath(Job job) throws IOException {
        Configuration conf = job.getConfiguration();
        String outPathString = conf.get("outpath");

        FileSystem fs = FileSystem.get(conf);

        Path outPath = new Path(outPathString);
        if(fs.exists(outPath)){
            fs.delete(outPath, true);
        }

        FileOutputFormat.setOutputPath(job, outPath);

    }

    private void initJobInputPath(Job job) throws IOException {
        Configuration conf = job.getConfiguration();
        String inPathString = conf.get("inpath");

        FileSystem fs = FileSystem.get(conf);

        Path inPath = new Path(inPathString);
        if(fs.exists(inPath)){
            FileInputFormat.addInputPath(job, inPath);
        }else{
            throw new RuntimeException("HDFS 中该文件目录不存在： " +
inPathString);
        }
    }

    public static void main(String[] args) {
        try {
            int resultCode = ToolRunner.run(new VideoETLRunner(),
args);
            if(resultCode == 0){
                System.out.println("Success!");
            }else{
```

```
                System.out.println("Fail!");
            }
            System.exit(resultCode);
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

4. 执行 ETL

```
$ bin/yarn jar ~/softwares/jars/gulivideo-0.0.1-SNAPSHOT.jar \
com.atguigu.etl.ETLVideosRunner \
/gulivideo/video/2008/0222 \
/gulivideo/output/video/2008/0222
```

10.3 准备工作

10.3.1 创建表

创建表：gulivideo_ori，gulivideo_user_ori,

创建表：gulivideo_orc，gulivideo_user_orc

gulivideo_ori：

```
create table gulivideo_ori(
    videoId string,
    uploader string,
    age int,
    category array<string>,
    length int,
    views int,
    rate float,
    ratings int,
    comments int,
    relatedId array<string>)
row format delimited
fields terminated by "\t"
collection items terminated by "&"
stored as textfile;
```

gulivideo_user_ori：

```
create table gulivideo_user_ori(
    uploader string,
    videos int,
    friends int)
row format delimited
fields terminated by "\t"
stored as textfile;
```

然后把原始数据插入到 orc 表中

gulivideo_orc：

```
create table gulivideo_orc(
    videoId string,
    uploader string,
    age int,
    category array<string>,
```

```
    length int,
    views int,
    rate float,
    ratings int,
    comments int,
    relatedId array<string>)
row format delimited fields terminated by "\t"
collection items terminated by "&"
stored as orc;
```

gulivideo_user_orc：

```
create table gulivideo_user_orc(
    uploader string,
    videos int,
    friends int)
row format delimited
fields terminated by "\t"
stored as orc;
```

10.3.2 导入 ETL 后的数据

gulivideo_ori：

```
load data inpath "/gulivideo/output/video/2008/0222" into table gulivideo_ori;
```

gulivideo_user_ori：

```
load data inpath "/gulivideo/user/2008/0903" into table gulivideo_user_ori;
```

10.3.3 向 ORC 表插入数据

gulivideo_orc：

```
insert into table gulivideo_orc select * from gulivideo_ori;
```

gulivideo_user_orc：

```
insert into table gulivideo_user_orc select * from gulivideo_user_ori;
```

10.4 业务分析

10.4.1 统计视频观看数 Top10

思路：使用 order by 按照 views 字段做一个全局排序即可，同时我们设置只显示前 10 条。

最终代码：

```
select
    videoId,
    uploader,
    age,
    category,
    length,
    views,
    rate,
    ratings,
    comments
from
    gulivideo_orc
order by
    views
```

```
desc limit
    10;
```

10.4.2 统计视频类别热度 Top10

思路：

1) 即统计每个类别有多少个视频，显示出包含视频最多的前 10 个类别。
2) 我们需要按照类别 group by 聚合，然后 count 组内的 videoId 个数即可。
3) 因为当前表结构为：一个视频对应一个或多个类别。所以如果要 group by 类别，需要先将类别进行列转行(展开)，然后再进行 count 即可。
4) 最后按照热度排序，显示前 10 条。

最终代码：

```
select
    category_name as category,
    count(t1.videoId) as hot
from (
    select
        videoId,
        category_name
    from
        gulivideo_orc lateral view explode(category) t_catetory as
category_name) t1
group by
    t1.category_name
order by
    hot
desc limit
    10;
```

10.4.3 统计出视频观看数最高的 20 个视频的所属类别以及类别包含 Top20 视频的个数

思路：

1) 先找到观看数最高的 20 个视频所属条目的所有信息，降序排列
2) 把这 20 条信息中的 category 分裂出来(列转行)
3) 最后查询视频分类名称和该分类下有多少个 Top20 的视频

最终代码：

```
select
    category_name as category,
    count(t2.videoId) as hot_with_views
from (
    select
        videoId,
        category_name
    from (
        select
            *
        from
            gulivideo_orc
        order by
            views
        desc limit
            20) t1 lateral view explode(category) t_catetory as
```

```
category_name) t2
group by
    category_name
order by
    hot_with_views
desc;
```

10.4.4 统计视频观看数 Top50 所关联视频的所属类别 Rank

思路：

1) 查询出观看数最多的前 50 个视频的所有信息(当然包含了每个视频对应的关联视频)，记为临时表 t1

t1：观看数前 50 的视频

```
select
    *
from
    gulivideo_orc
order by
    views
desc limit
    50;
```

2) 将找到的 50 条视频信息的相关视频 relatedId 列转行，记为临时表 t2

t2：将相关视频的 id 进行列转行操作

```
select
    explode(relatedId) as videoId
from
    t1;
```

3) 将相关视频的 id 和 gulivideo_orc 表进行 inner join 操作

t5：得到两列数据，一列是 category，一列是之前查询出来的相关视频 id

```
 (select
    distinct(t2.videoId),
    t3.category
from
    t2
inner join
    gulivideo_orc t3 on t2.videoId = t3.videoId) t4 lateral view
explode(category) t_catetory as category_name;
```

4) 按照视频类别进行分组，统计每组视频个数，然后排行

最终代码：

```
select
    category_name as category,
    count(t5.videoId) as hot
from (
    select
        videoId,
        category_name
    from (
        select
            distinct(t2.videoId),
            t3.category
        from (
            select
                explode(relatedId) as videoId
            from (
                select
```

```
                *
            from
                gulivideo_orc
            order by
                views
            desc limit
                50) t1) t2
        inner join
            gulivideo_orc t3 on t2.videoId = t3.videoId) t4 lateral view
explode(category) t_catetory as category_name) t5
group by
    category_name
order by
    hot
desc;
```

10.4.5 统计每个类别中的视频热度 Top10，以 Music 为例

思路：

1) 要想统计 Music 类别中的视频热度 Top10，需要先找到 Music 类别，那么就需要将 category 展开，所以可以创建一张表用于存放 categoryId 展开的数据。

2) 向 category 展开的表中插入数据。

3) 统计对应类别（Music）中的视频热度。

最终代码：

创建表类别表：

```
create table gulivideo_category(
    videoId string,
    uploader string,
    age int,
    categoryId string,
    length int,
    views int,
    rate float,
    ratings int,
    comments int,
    relatedId array<string>)
row format delimited
fields terminated by "\t"
collection items terminated by "&"
stored as orc;
```

向类别表中插入数据：

```
insert into table gulivideo_category
    select
        videoId,
        uploader,
        age,
        categoryId,
        length,
        views,
        rate,
        ratings,
        comments,
        relatedId
    from
```

```
gulivideo_orc lateral view explode(category) catetory as categoryId;
```

统计 Music 类别的 Top10（也可以统计其他）

```
select
    videoId,
    views
from
    gulivideo_category
where
    categoryId = "Music"
order by
    views
desc limit
    10;
```

10.4.6 统计每个类别中视频流量 Top10，以 Music 为例

思路：

1) 创建视频类别展开表（categoryId 列转行后的表）

2) 按照 ratings 排序即可

最终代码：

```
select
    videoId,
    views,
    ratings
from
    gulivideo_category
where
    categoryId = "Music"
order by
    ratings
desc limit
    10;
```

10.4.7 统计上传视频最多的用户 Top10 以及他们上传的观看次数在前 20 的视频

思路：

1) 先找到上传视频最多的 10 个用户的用户信息

```
select
    *
from
    gulivideo_user_orc
order by
    videos
desc limit
    10;
```

2) 通过 uploader 字段与 gulivideo_orc 表进行 join，得到的信息按照 views 观看次数进行排序即可。

最终代码：

```
select
    t2.videoId,
    t2.views,
    t2.ratings,
    t1.videos,
    t1.friends
from (
    select
        *
    from
        gulivideo_user_orc
    order by
        videos desc
    limit
        10) t1
join
    gulivideo_orc t2
on
    t1.uploader = t2.uploader
order by
    views desc
limit
    20;
```

10.4.8 统计每个类别视频观看数 Top10

思路：

1) 先得到 categoryId 展开的表数据

2) 子查询按照 categoryId 进行分区，然后分区内排序，并生成递增数字，该递增数字这一列起名为 rank 列

3) 通过子查询产生的临时表，查询 rank 值小于等于 10 的数据行即可。

最终代码：

```
select
    t1.*
from (
    select
        videoId,
        categoryId,
        views,
        row_number() over(partition by categoryId order by views desc)
rank from gulivideo_category) t1
where
    rank <= 10;
```

第 11 章 常见错误及解决方案

1）SecureCRT 7.3 出现乱码或者删除不掉数据，免安装版的 SecureCRT 卸载或者用虚拟机直接操作或者换安装版的 SecureCRT

2）连接不上 mysql 数据库

（1）导错驱动包，应该把 mysql-connector-java-5.1.27-bin.jar 导入/opt/module/hive/lib 的不是这个包。错把 mysql-connector-java-5.1.27.tar.gz 导入 hive/lib 包下。

（2）修改 user 表中的主机名称没有都修改为%，而是修改为 localhost

3）hive 默认的输入格式处理是 CombineHiveInputFormat，会对小文件进行合并。

```
hive (default)> set hive.input.format;
hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFo
rmat
```

可以采用 HiveInputFormat 就会根据分区数输出相应的文件。

```
hive (default)> set
hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
```

4）不能执行 mapreduce 程序

可能是 hadoop 的 yarn 没开启。

5）启动 mysql 服务时，报 MySQL server PID file could not be found! 异常。

在/var/lock/subsys/mysql 路径下创建 hadoop102.pid，并在文件中添加内容：4396

6）报 service mysql status MySQL is not running, but lock file (/var/lock/subsys/mysql[失败])异常。

解决方案：在/var/lib/mysql 目录下创建： -rw-rw----. 1 mysql mysql 5 12 月 22 16:41 hadoop102.pid 文件，并修改权限为 777。

7）JVM 堆内存溢出

描述：java.lang.OutOfMemoryError: Java heap space

解决：在 yarn-site.xml 中加入如下代码

```
<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>2048</value>
</property>
<property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>2048</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-pmem-ratio</name>
    <value>2.1</value>
</property>
<property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx1024m</value>
</property>
```