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

1）Hive 处理的数据存储在 HDFS

2）Hive 分析数据底层的实现是 MapReduce

3）执行程序运行在 Yarn 上

1.2 Hive 的优缺点

1.2.1 优点

1）操作接口采用类 SQL 语法，提供快速开发的能力（简单、容易上手）

2）避免了去写 MapReduce，减少开发人员的学习成本。

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

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

5）Hive 支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。

1.2.2 缺点

1）Hive 的 HQL 表达能力有限

（1）迭代式算法无法表达

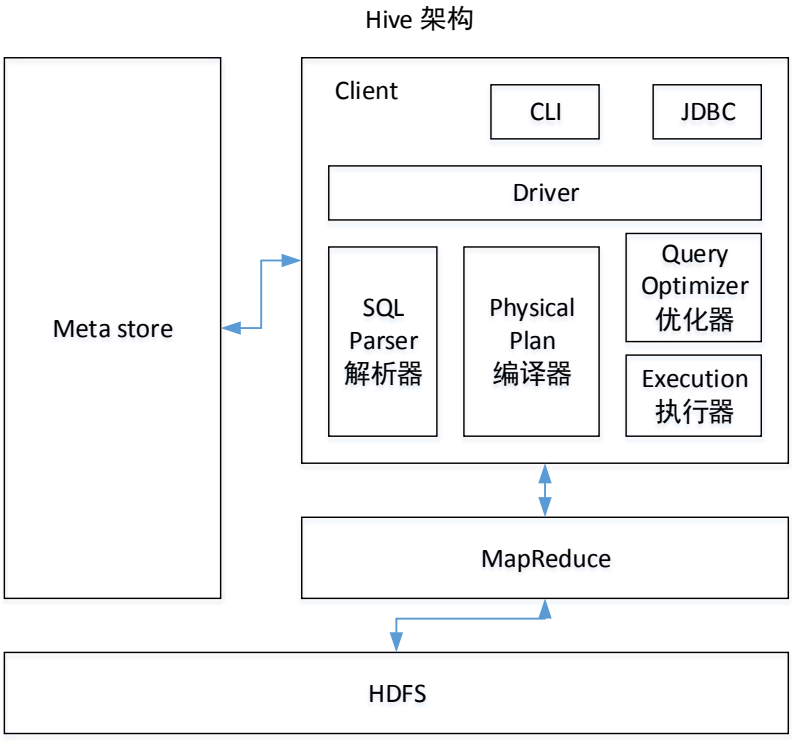
（2）数据挖掘方面不擅长

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



如图中所示，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）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。

1.4 Hive 和数据库比较

由于 Hive 采用了类似 SQL 的查询语言 HQL(Hive Query Language)，因此很容易将 Hive 理解为数据库。其实从结构上来看，Hive 和数据库除了拥有类似的查询语言，再无类似之处。本文将从多个方面来阐述 Hive 和数据库的差异。数据库可以用在 Online 的应用中，但是 Hive 是为数据仓库而设计的，清楚这一点，有助于从应用角度理解 Hive 的特性。

1.4.1 查询语言

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

1.4.2 数据存储位置

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/

4）github 地址：

https://github.com/apache/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

```
[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 777 /tmp
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -chmod 777 /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/datas/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/datas/student.txt 这个目录下准备数据

（1）在/opt/module/目录下创建 datas

[atguigu@hadoop102 module]$ mkdir datas

（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/datas/student.txt 文件到 student 数据库表中。

hive> load data local inpath '/opt/module/datas/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.SessionHiveMetaStoreClient
        at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.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(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.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.SessionHiveMetaStoreClient
        at org.apache.hadoop.hive.metastore.MetaStoreUtils.newInstance(MetaStoreUtils.java:1523)
        at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.<init>(RetryingMetaStoreClient.java:86)
        at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(RetryingMetaStoreClient.java:132)
        at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(RetryingMetaStoreClient.java:104)
        at org.apache.hadoop.hive.ql.metadata.Hive.createMetaStoreClient(Hive.java:3005)
        at org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:3024)
        at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.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，并设置当前用户执行权限

```
[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

[root@hadoop102 mysql-libs]# chmod u+x ./*
[root@hadoop102 mysql-libs]# ll
总用量 76048
-rwxr--r--. 1 root root 18509960 3 月  26 2015 MySQL-client-5.6.24-1.el6.x86_64.rpm
-rwxr--r--. 1 root root  3575135 12 月  1 2013 mysql-connector-java-5.1.27.tar.gz
-rwxr--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

```
mysql>delete from user where Host='hadoop102 ';
mysql>delete from user where Host='127.0.0.1';
mysql>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/opt/software/mysql-libs/mysql-connector-java-5.1.27 目录下的 mysql-connector-java-5.1.27-bin.jar 到/opt/module/hive/lib/

```
[root@hadoop102 mysql-connector-java-5.1.27]# cp 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 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

2.7 Hive 其他命令操作

1）退出 hive 窗口：

hive(default)>exit;

hive(default)>quit;

在新版的 oracle 中没区别了，在以前的版本是有的：

exit:先隐性提交数据，再退出；

quit:不提交数据，退出；

2）在 hive cli 命令窗口中如何查看 hdfs 文件系统

hive(default)>dfs -ls /;

3）在 hive cli 命令窗口中如何查看 hdfs 本地系统

hive(default)>! ls /opt/module/datas;

4）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

2.8 Hive 常见属性配置

2.8.1 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
```

2.8.2 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

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

（2）配置后

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

2.8.3 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

2.8.4 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

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

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

3.2 集合数据类型

数据类型	描述	语法示例
STRUCT	和 c 语言中的 struct 类似，都可以通过“点”符号访问元素内容。例如，如果某个列的数据类型是 STRUCT{first STRING, last STRING},那么第 1 个元素可以通过字段.first 来引用。	struct()
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	Array()

	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。	

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

3.3 类型转化

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

1）隐式类型转换规则如下。

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

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

（3）TINYINT、SMALLINT、INT 都可以转换为 FLOAT。

（4）BOOLEAN 类型不可以转换为任何其它的类型。

2）可以使用 CAST 操作显示进行数据类型转换，例如 CAST('1' AS INT)将把字符串'1' 转换成整数 1；如果强制类型转换失败，如执行 CAST('X' AS INT)，表达式返回空值 NULL。

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

3）创建一个数据库，指定数据库在 HDFS 上存放的位置

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

4.2 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 db_hive set dbproperties('createtime'='20170830');
```

在 mysql 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}

4.3 查询数据库

4.3.1 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
```

4.3.2 查看数据库详情

1）显示数据库信息

```
hive> desc database db_hive;
OK
db_hive         hdfs://hadoop102:8020/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://hadoop102:8020/user/hive/warehouse/db_hive.db      atguiguUSER
```

4.3.3 切换当前数据库

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

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_hive2;
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。

（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

4.6 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。

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

4.6.1 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

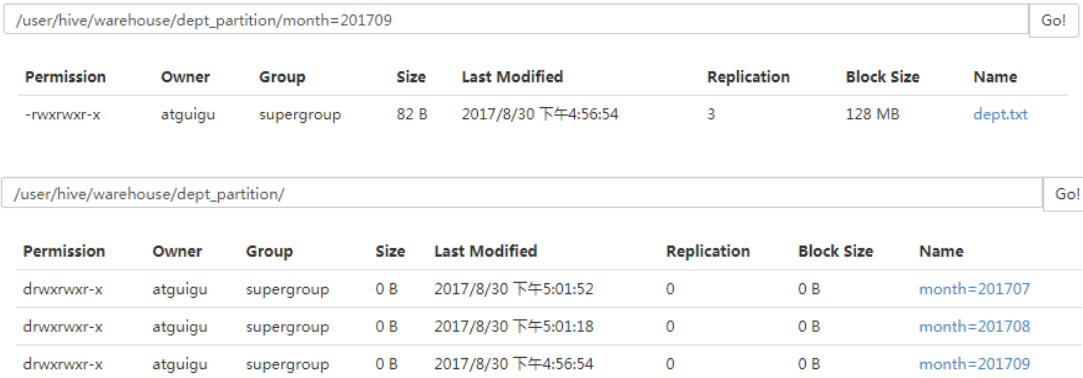
2）创建分区表语法

```
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');
```



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两种方式

（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：上传数据后 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

```
hive (default)> 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

```
[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

（1）查询名称和部门

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

6.1.3 算术运算符

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

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。

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 和 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）

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 后面不能写分组函数，而 having 后面可以使用分组函数。

（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、员工名称和部门编号；

```
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。

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。

0）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/datas/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, deptno from emp, dept;
```

FAILED: SemanticException Column deptno Found in more than One Tables/Subqueries

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.ename;
```

错误的

6.5 排序

6.5.1 全局排序（Order By）

Order By：全局排序，一个 MapReduce

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 MapReduce 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。但是排序只能是倒序排序，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。

1）先创建分桶表，通过直接导入数据文件的方式

（0）数据准备

student.txt

（1）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';
```

（2）查看表结构

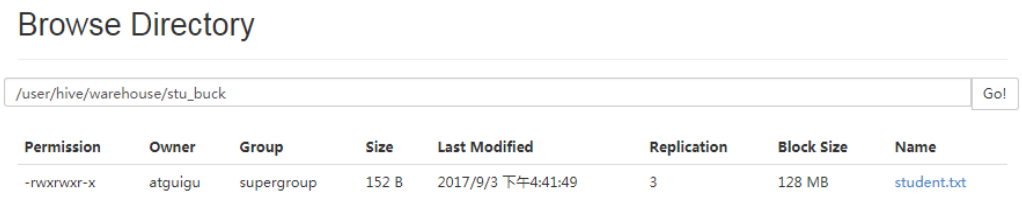
hive (default)> desc formatted stu_buck;

Num Buckets: 4

（3）导入数据到分桶表中

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

（4）查看创建的分桶表中是否分成 4 个桶



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;
```

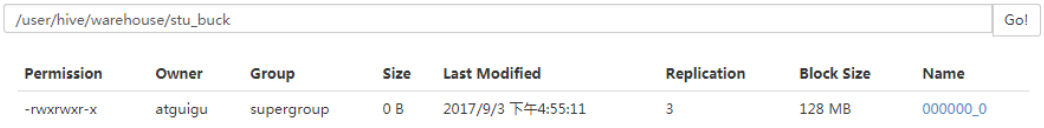
（3）清空 stu_buck 表中数据

```
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

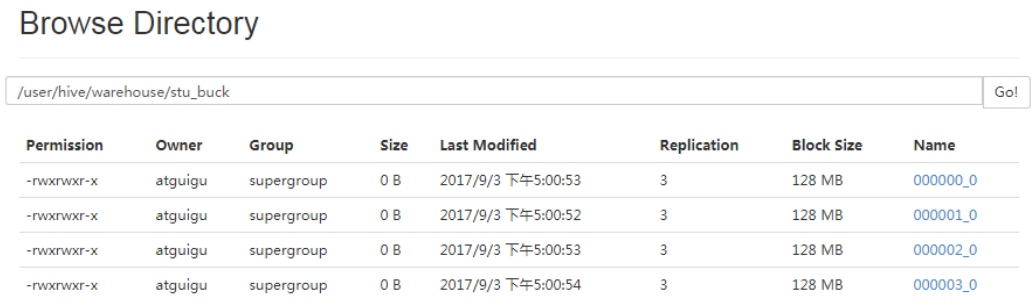
```
insert into table stu_buck
select id, name from stu;
```

（5）发现还是只有一个分桶



（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;
```



（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 2 out of 4)，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

6.6.3 数据块抽样

Hive 提供了另外一种按照百分比进行抽样的方式，这种是基于行数的，按照输入路径下的数据块百分比进行的抽样。

```
hive (default)> select * from stu tablesample(0.1 percent) ;
```

提示：这种抽样方式不一定适用于所有的文件格式。另外，这种抽样的最小抽样单元是一个 HDFS 数据块。因此，如果表的数据大小小于普通的块大小 128M 的话，那么将会返回所有行。

6.7 行转列查询

1）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

2）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
```

3）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
```

4）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/datas/person_info.txt” into table person_info;
```

5）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）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。

6.8 列转行查询

1）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

2）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
```

3）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
```

4）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;
```

5）按需求查询数据

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

6）相关函数说明

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。

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

6.9 开窗函数查询

1）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

2）需求:

（1）查询在 2017 年 4 月份购买过的顾客及总人数

（2）查询顾客的购买明细及月购买总额

（3）上述的场景,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

3）创建本地 business.txt，导入数据

[atguigu@hadoop102 datas]$ vi business.txt

4）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;
```

5）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) = '2015-04'
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）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

PRECEDING n:往前 n 行数据

FOLLOWING n: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。

第 7 章 函数

7.1 系统内置函数

1）查看系统内置函数

```
hive> show functions;
```

2）显示内置函数用法

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

3）详细显示内置函数用法

```
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；

7.3 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 (final String s) {

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

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

4）打成 jar 包上传到服务器/opt/module/jars/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 udf_lower as "com.atguigu.hive.Lower";
```

7）即可在 hql 中使用自定义的函数 strip

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

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

0）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
```

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

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

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

压缩性能的比较

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

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

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

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）：

参数	默认值	阶段	建议

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

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;
```

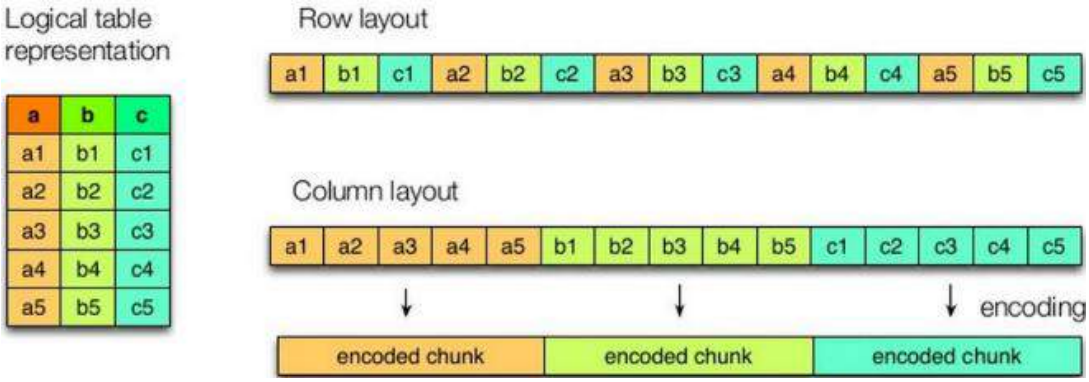
5）测试一下输出结果是否是压缩文件

```
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



上图左边为逻辑表，右边第一个为行式存储，第二个为列式存储。

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。

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。

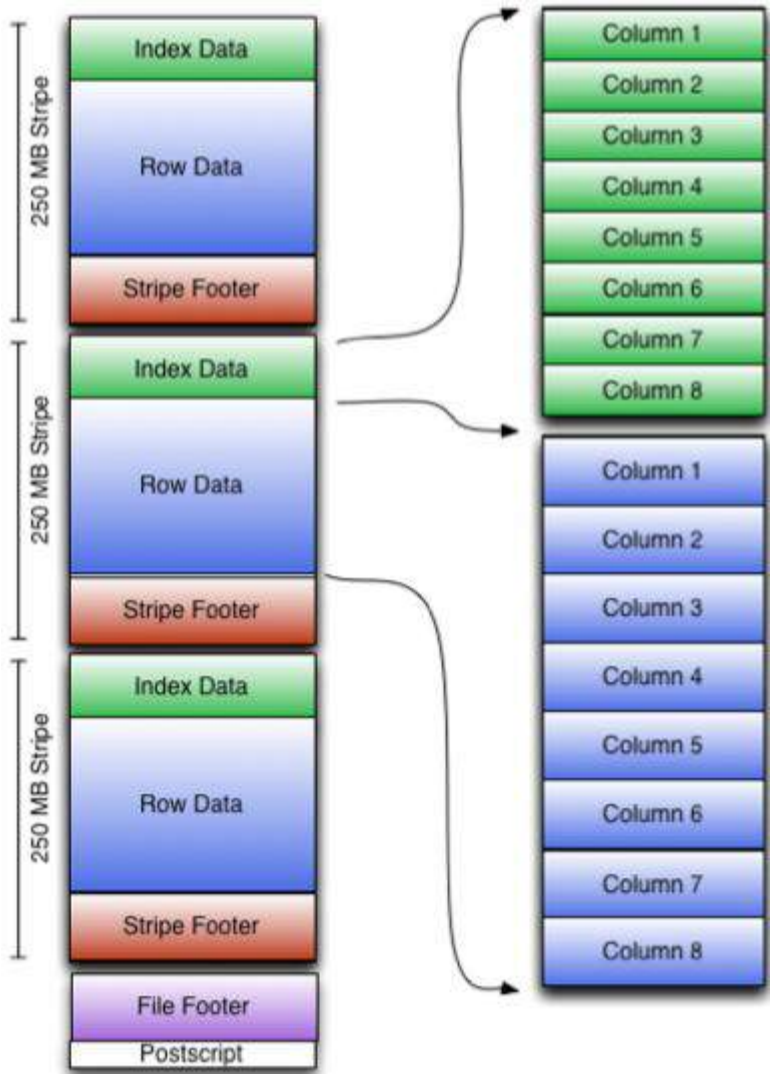
8.5.2 TextFile 格式

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。

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：



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。

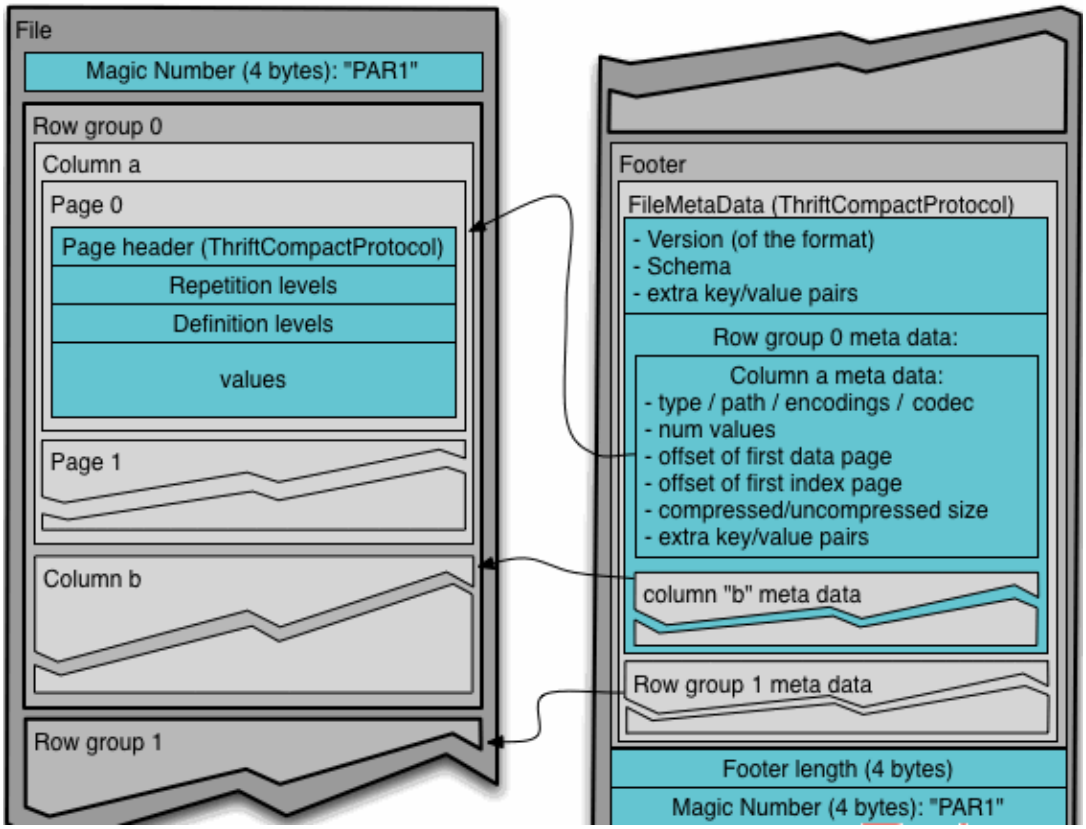
每个文件有一个 File Footer，这里面存的是每个 Stripe 的行数，每个 Column 的数据类型信息等；每个文件的尾部是一个 PostScript，这里面记录了整个文件的压缩类型以及 FileFooter 的长度信息等。在读取文件时，会 seek 到文件尾部读 PostScript，从里面解析到 File Footer 长度，再读 FileFooter，从里面解析到各个 Stripe 信息，再读各个 Stripe，即从后往前读。

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 文件的格式如下图所示。



上图展示了一个 Parquet 文件的内容，一个文件中可以存储多个行组，文件的首位都是该文件的 Magic Code，用于校验它是否是一个 Parquet 文件，Footer length 记录了文件元数据的大小，通过该值和文件长度可以计算出元数据的偏移量，文件的元数据中包括每一个行组的元数据信息和该文件存储数据的 Schema 信息。除了文件中每一个行组的元数据，每一页的开始都会存储该页的元数据，在 Parquet 中，有三种类型的页：数据页、字典页和索引页。数据页用于存储当前行组中该列的值，字典页存储该列值的编码字典，每一个列块中最多包含一个字典页，索引页用来存储当前行组下该列的索引，目前 Parquet 中还不支持索引页。

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：

0）测试数据

log.data

1）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

2）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

3）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)

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)

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)

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：

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 Group 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

（0）需求：测试大表 JOIN 小表和小表 JOIN 大表的效率

（1）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';
```

（2）分别向大表和小表中导入数据

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

（3）关闭 mapjoin 功能（默认是打开的）

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

（4）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

（5）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

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://192.168.1.102:19888/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/ori' into table ori;
hive (default)> load data local inpath '/opt/module/datas/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
```

（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
```

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

Show 20 entries

Task							Succes
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time
task_1506334829052_0015_r_000000	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:24 +0800 2017	18sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	Mon Sep 25 19:18:18 +0800 2017
task_1506334829052_0015_r_000001	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	12sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017
task_1506334829052_0015_r_000004	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:18 +0800 2017	11sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task_1506334829052_0015_r_000003	SUCCEEDED	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:17 +0800 2017	10sec	Mon Sep 25 19:18:06 +0800 2017	Mon Sep 25 19:18:14 +0800 2017	Mon Sep 25 19:18:14 +0800 2017
task_1506334829052_0015_r_000002	SUCCEEDED	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:16 +0800 2017	10sec	Mon Sep 25 19:18:05 +0800 2017	Mon Sep 25 19:18:13 +0800 2017	Mon Sep 25 19:18:13 +0800 2017

随机分布空 null 值

（1）设置 5 个 reduce 个数

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

（2）JOIN 两张表

```
insert overwrite table jointable
select n.* from nullidtable n full join ori o on
case when n.id is null then concat('hive', rand()) else n.id end = o.id;
```

结果：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

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：

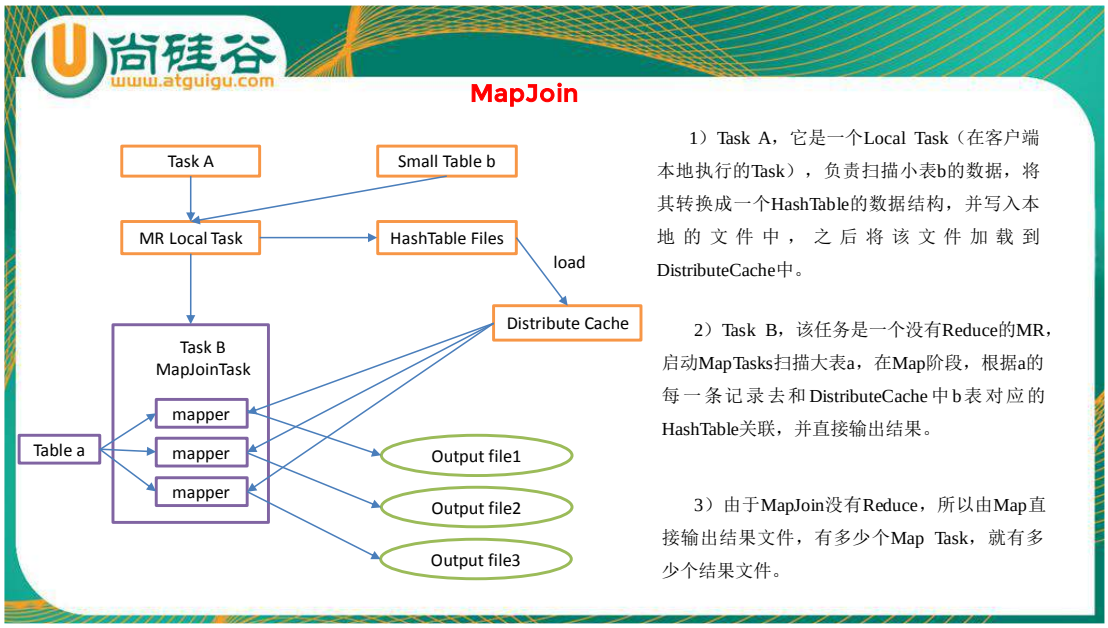
（1）设置自动选择 Mapjoin

```
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



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

（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

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。

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)
```

（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)

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 '/opt/module/datas/ds1' into table ori_partitioned
partition(p_time='20111230000010') ;
hive (default)> load data local inpath '/opt/module/datas/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;
```

（5）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 数据倾斜

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。

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向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 Sqoop

10.1 Sqoop 概述

Sqoop 是一款开源的工具，主要用于在 Hadoop(Hive)与传统的数据库(mysql、postgresql...)间进行数据的传递，可以将一个关系型数据库（例如 ： MySQL ,Oracle ,Postgres 等）中的数据导进到 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导进到关系型数据库中。

Sqoop 项目开始于 2009 年，最早是作为 Hadoop 的一个第三方模块存在，后来为了让使用者能够快速部署，也为了让开发人员能够更快速的迭代开发，Sqoop 独立成为一个 Apache 项目。

最新的稳定版本是 1.4.7。Sqoop2 的最新版本是 1.99.7。请注意，1.99.7 与 1.4.7 不兼容，且没有特征不完整，它并不打算用于生产部署。

10.2 Sqoop 下载与安装

10.2.1 Sqoop 安装地址

1）Sqoop 官网地址：

http://sqoop.apache.org/

2）文档查看地址：

http://sqoop.apache.org/docs/1.4.7/index.html

3）下载地址：

https://mirrors.tuna.tsinghua.edu.cn/apache/sqoop/1.4.7/

10.2.2 Sqoop 安装部署

1) 把 sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz 上传到 linux 的/opt/software 目录下
2) 解压 sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz 到/opt/module/目录下

```
[atguigu@hadoop102 software]$ tar -zxvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz -C /opt/module/
```

3) 修改 sqoop-1.4.7.bin__hadoop-2.6.0 的名称为 sqoop

```
[atguigu@hadoop102 software]$ mv sqoop-1.4.7.bin__hadoop-2.6.0/ sqoop
```

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

```
[atguigu@hadoop102 software]$ mv sqoop-env-template.sh sqoop-env.sh

[atguigu@hadoop102 software]$ mv sqoop-site-template.xml sqoop-site.xml
```

5）配置 sqoop-env.sh 文件

```
export HADOOP_COMMON_HOME=/opt/module/hadoop-2.7.2
export HADOOP_MAPRED_HOME=/opt/module/hadoop-2.7.2
export HIVE_HOME=/opt/module/hive
export ZOOKEEPER_HOME=/opt/module/zookeeper-3.4.10
export ZOOCFGDIR=/opt/module/zookeeper-3.4.10
```

10.2.3 添加 JDBC 驱动

拷贝/opt/software/mysql-libs/mysql-connector-java-5.1.27 目录下的

mysql-connector-java-5.1.27-bin.jar 到/opt/module/sqoop/lib/

```
[atguigu@hadoop102 mysql-connector-java-5.1.27]$ cp mysql-connector-java-5.1.27-bin.jar /opt/module/sqoop/lib/
```

10.2.4 验证 Sqoop

我们可以通过某一个 command 来验证 sqoop 配置是否正确：

```
$ bin/sqoop help
出现一些 Warning 警告（警告信息已省略），并伴随着帮助命令的输出：
Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  import-mainframe   Import datasets from a mainframe server to HDFS
  job                Work with saved jobs
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  merge              Merge results of incremental imports
  metastore          Run a standalone Sqoop metastore
  version            Display version information
```

10.2.5 测试 Sqoop 是否能够成功连接数据库

```
$ bin/sqoop list-databases --connect jdbc:mysql://hadoop102:3306/ --username root --password 000000
出现如下输出：
information_schema
metastore
mysql
performance_schema
```

10.3 导入数据

在 Sqoop 中，“导入”概念指：从非大数据集群（RDBMS）向大数据集群（HDFS，HIVE，

HBASE）中传输数据，叫做：导入，即使用 import 关键字。

10.3.1 RDBMS 到 HDFS

1) 确定 Mysql 服务开启正常

2) 在 Mysql 中新建一张表并插入一些数据

```
$ mysql -uroot -p123456
mysql> create database company;
mysql> create table company.staff(id int(4) primary key not null auto_increment, name varchar(255), sex varchar(255));
mysql> insert into company.staff(name, sex) values('Thomas', 'Male');
mysql> insert into company.staff(name, sex) values('Catalina', 'FeMale');
```

3) 导入数据

（1）全部导入

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t"
```

（2）查询导入

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
--query 'select name,sex from staff where id <=1 and $CONDITIONS;'
```

尖叫提示：must contain '$CONDITIONS' in WHERE clause.

尖叫提示：如果 query 后使用的是双引号，则$CONDITIONS 前必须加转移符，防止 shell 识别为自己的变量。

（3）导入指定列

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
```

```
--columns id,sex \
--table staff
```

尖叫提示：columns 中如果涉及到多列，用逗号分隔，分隔时不要添加空格

（4）使用 sqoop 关键字筛选查询导入数据

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
--table staff \
--where "id=1"
```

尖叫提示：在 Sqoop 中可以使用 sqoop import -D property.name=property.value 这样的方式加入执行任务的参数，多个参数用空格隔开。

10.3.2 RDBMS 到 Hive

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff \
--num-mappers 1 \
--hive-import \
--fields-terminated-by "\t" \
--hive-overwrite \
--hive-table staff_hive
```

提示：该过程分为两步，第一步将数据导入到 HDFS，第二步将导入到 HDFS 的数据迁移到 Hive 仓库，第一步默认的临时目录是/user/admin/表名

10.4 导出数据

在 Sqoop 中，“导出”概念指：从大数据集群（HDFS，HIVE，HBASE）向非大数据集群（RDBMS）中传输数据，叫做：导出，即使用 export 关键字。

4.2.1、HIVE/HDFS 到 RDBMS

```
$ bin/sqoop export \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 000000 \
--table staff \
--num-mappers 1 \
```

```
--export-dir /user/hive/warehouse/staff_hive \
--input-fields-terminated-by "\t"
```

注意：Mysql 中如果表不存在，不会自动创建

思考：数据是覆盖还是追加

10.5 脚本打包

使用 opt 格式的文件打包 sqoop 命令，然后执行

1) 创建一个.opt 文件

```
$ mkdir opt
$ touch opt/job_HDFS2RDBMS.opt
```

2) 编写 sqoop 脚本

```
$ vi opt/job_HDFS2RDBMS.opt

export
--connect
jdbc:mysql://hadoop102:3306/company
--username
root
--password
000000
--table
staff
--num-mappers
1
--export-dir
/user/hive/warehouse/staff_hive
--input-fields-terminated-by
"\t"
```

3) 执行该脚本

```
$ bin/sqoop --options-file opt/job_HDFS2RDBMS.opt
```

第 11 章 Hive 实战

11.1 需求描述

统计 Youtube 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
--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 以及他们上传的视频
--统计每个类别视频观看数 Top10

11.2 项目

11.2.1 数据结构

1）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	视频上传日期和 2007 年 2 月 15 日之间的整数天（Youtube 的独特设定）
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

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

11.2.2 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.z.youtube.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, "youtube-video-etl");

        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/youtube-0.0.1-SNAPSHOT.jar \
com.z.youtube.etl.ETLYoutubeVideosRunner \
/youtube/video/2008/0222 \
/youtube/output/video/2008/0222
```

11.3 准备工作

11.3.1 创建表

创建表：youtube_ori，youtube_user_ori，
创建表：youtube_orc，youtube_user_orc
youtube_ori：

```
create table youtube_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;
```

youtube_user_ori：

```
create table youtube_user_ori(
    uploader string,
    videos int,
    friends int)
clustered by (uploader) into 24 buckets
row format delimited
fields terminated by "\t"
stored as textfile;
```

然后把原始数据插入到 orc 表中

youtube_orc：

```
create table youtube_orc(
    videoId string,
    uploader string,
    age int,
    category array<string>,
    length int,
    views int,
    rate float,
    ratings int,
    comments int,
    relatedId array<string>)
clustered by (uploader) into 8 buckets
row format delimited fields terminated by "\t"
collection items terminated by "&"
stored as orc;
```

youtube_user_orc：

```
create table youtube_user_orc(
    uploader string,
    videos int,
    friends int)
clustered by (uploader) into 24 buckets
row format delimited
fields terminated by "\t"
stored as orc;
```

11.3.2 导入 ETL 后的数据

youtube_ori：

```
load data inpath "/youtube/output/video/2008/0222" into table youtube_ori;
```

youtube_user_ori：

```
load data inpath "/youtube/user/2008/0903" into table youtube_user_ori;
```

11.3.3 向 ORC 表插入数据

youtube_orc：

```
insert into table youtube_orc select * from youtube_ori;
```

youtube_user_orc：

```
insert into table youtube_user_orc select * from youtube_user_ori;
```

11.4 业务分析

11.4.1 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
    youtube_orc
order by
    views
desc limit
    10;
```

11.4.2 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
        youtube_orc lateral view explode(category) t_catetory as category_name) t1
group by
    t1.category_name
order by
    hot
desc limit
    10;
```

11.4.3 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
            youtube_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;
```

11.4.4 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
	youtube_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 youtube_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
	youtube_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
                        youtube_orc
                    order by
                        views
                    desc limit
                        50) t1) t2
            inner join
                youtube_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;
```

11.4.5 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 youtube_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 youtube_category
    select
        videoId,
        uploader,
```

```
        age,
        categoryId,
        length,
        views,
        rate,
        ratings,
        comments,
        relatedId
    from
        youtube_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
    youtube_category
where
    categoryId = "Music"
order by
    views
desc limit
    10;
```

11.4.6 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
    youtube_category
where
    categoryId = "Music"
order by
    ratings
desc limit
    10;
```

11.4.7 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
	youtube_user_orc
order by
	videos
desc limit
	10;
```

2) 通过 uploader 字段与 youtube_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
		youtube_user_orc
	order by
		videos desc
	limit
		10) t1
join
	youtube_orc t2
on
	t1.uploader = t2.uploader
order by
	views desc
limit
	20;
```

11.4.8 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
youtube_category) t1
where
	rank <= 10;
```

第 12 章 数据仓库

12.1 什么是数据仓库

数据仓库，英文名称为 Data Warehouse，可简写为 DW 或 DWH。数据仓库，是为企业所有级别的决策制定过程，提供所有类型数据支持的战略集合。它出于分析性报告和决策支持目的而创建。为需要业务智能的企业，提供指导业务流程改进、监视时间、成本、质量以及控制。

12.2 数据仓库能干什么？

1）年度销售目标的指定，需要根据以往的历史报表进行决策，不能拍脑袋。

2）如何优化业务流程

例如：一个电商网站订单的完成包括：浏览、下单、支付、物流，其中物流环节可能和中通、申通、韵达等快递公司合作。快递公司每派送一个订单，都会有订单派送的确认时间，可以根据订单派送时间来分析哪个快递公司比较快捷高效，从而选择与哪些快递公司合作，剔除哪些快递公司，增加用户友好型。

12.3 数据仓库的特点

1）数据仓库的数据是面向主题的

与传统数据库面向应用进行数据组织的特点相对应，数据仓库中的数据是面向主题进行组织的。什么是主题呢？首先，主题是一个抽象的概念，是较高层次上企业信息系统中的数据综合、归类并进行分析利用的抽象。在逻辑意义上，它是对应企业中某一宏观分析领域所涉及的分析对象。面向主题的数据组织方式，就是在较高层次上对分析对象的数据的一个完整、一致的描述，能完整、统一地刻划各个分析对象所涉及的企业的各项数据，以及数据之间的联系。所谓较高层次是相对面向应用的数据组织方式而言的，是指按照主题进行数据组织的方式具有更高的数据抽象级别。

2）数据仓库的数据是集成的

数据仓库的数据是从原有的分散的数据库数据抽取来的。操作型数据与 DSS 分析型数据之间差别甚大。第一，数据仓库的每一个主题所对应的源数据在原有的各分散数据库中有许多重复和不一致的地方，且来源于不同的联机系统的数据都和不同的应用逻辑捆绑在一起；第二，数据仓库中的综合数据不能从原有的数据库系统直接得到。因此在数据进入数据仓库之前，必然要经过统一与综合，这一步是数据仓库建设中最关键、最复杂的一

步，所要完成的工作有：

（1）要统一源数据中所有矛盾之处，如字段的同名异义、异名同义、单位不统一、字长不一致等。

（2)进行数据综合和计算。数据仓库中的数据综合工作可以在从原有数据库抽取 数据时生成，但许多是在数据仓库内部生成的，即进入数据仓库以后进行综合生成的。

3）数据仓库的数据是不可更新的

数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一般情况下并不进行修改操作。数据仓库的数据反映的是一段相当长的时间内历史数据的内容，是不同时点的数据库快照的集合，以及基于这些快照进行统计、综合和重组的导出数据，而不是联机处理的数据。数据库中进行联机处理的数据经过集成输入到数据仓库中，一旦数据仓库存放的数据已经超过数据仓库的数据存储期限，这些数据将从当前的数据仓库中删去。因为数据仓库只进行数据查询操作，所以数据仓库管理系统相比数据库管理系统而言要简单得多。数据库管理系统中许多技术难点，如完整性保护、并发控制等等，在数据仓库的管理中几乎可以省去。但是由于数据仓库的查询数据量往往很大，所以就对数据查询提出了更高的要求，它要求采用各种复杂的索引技术；同时由于数据仓库面向的是商业企业的高层管理者，他们会对数据查询的界面友好性和数据表示提出更高的要求。

4）数据仓库的数据是随时间不断变化的

数据仓库中的数据不可更新是针对应用来说的，也就是说，数据仓库的用户进行分析处理时是不进行数据更新操作的。但并不是说，在从数据集成输入数据仓库开始到最终被删除的整个数据生存周期中，所有的数据仓库数据都是永远不变的。

数据仓库的数据是随时间的变化而不断变化的，这是数据仓库数据的第四个特征。这一特征表现在以下 3 方面：

（1）数据仓库随时间变化不断增加新的数据内容。数据仓库系统必须不断捕捉 OLTP 数据库中变化的数据，追加到数据仓库中去，也就是要不断地生成 OLTP 数据库的快照，经统一集成后增加到数据仓库中去；但对于确实不再变化的数据库快照，如果捕捉到新的变化数据，则只生成一个新的数据库快照增加进去，而不会对原有的数据库快照进行修改。

（2）数据仓库随时间变化不断删去旧的数据内容。数据仓库的数据也有存储期限，一旦超过了这一期限，过期数据就要被删除。只是数据仓库内的数据时限要远远长于操作型

环境中的数据时限。在操作型环境中一般只保存有 60~90 天的数据，而在数据仓库中则需要保存较长时限的数据（如 5~10 年），以适应 DSS 进行趋势分析的要求。

（3）数据仓库中包含有大量的综合数据，这些综合数据中很多跟时间有关，如数据经常按照时间段进行综合，或隔一定的时间片进行抽样等等。这些数据要随着时间的变化不断地进行重新综合。因此，数据仓库的数据特征都包含时间项，以标明数据的历史时期。

11.4 数据仓库发展历程

数据仓库的发展大致经历了这样的三个过程：

1）简单报表阶段：这个阶段，系统的主要目标是解决一些日常的工作中业务人员需要的报表，以及生成一些简单的能够帮助领导进行决策所需要的汇总数据。这个阶段的大部分表现形式为数据库和前端报表工具。

2）数据集市阶段：这个阶段，主要是根据某个业务部门的需要，进行一定的数据的采集，整理，按照业务人员的需要，进行多维报表的展现，能够提供对特定业务指导的数据，并且能够提供特定的领导决策数据。

3）数据仓库阶段：这个阶段，主要是按照一定的数据模型，对整个企业的数据进行采集，整理，并且能够按照各个业务部门的需要，提供跨部门的，完全一致的业务报表数据，能够通过数据仓库生成对对业务具有指导性的数据，同时，为领导决策提供全面的数据支持。

通过数据仓库建设的发展阶段，我们能够看出，数据仓库的建设和数据集市的建设的重要区别就在于数据模型的支持。因此，数据模型的建设，对于我们数据仓库的建设，有着决定性的意义。

11.5 数据库与数据仓库的区别

了解数据库与数据仓库的区别之前，首先掌握三个概念。数据库软件、数据库、数据仓库。

数据库软件：是一种软件，可以看得见，可以操作。用来实现数据库逻辑功能。属于物理层。

数据库：是一种逻辑概念，用来存放数据的仓库。通过数据库软件来实现。数据库由很多表组成，表是二维的，一张表里可以有很多字段。字段一字排开，对应的数据就一行一行写入表中。数据库的表，在于能够用二维表现多维关系。目前市面上流行的数据库都是二维数据库。如：Oracle、DB2、MySQL、Sybase、MS SQL Server 等。

数据仓库：是数据库概念的升级。从逻辑上理解，数据库和数据仓库没有区别，都是通过数据库软件实现的存放数据的地方，只不过从数据量来说，数据仓库要比数据库更庞大得多。数据仓库主要用于数据挖掘和数据分析，辅助领导做决策。

在 IT 的架构体系中，数据库是必须存在的。必须要有地方存放数据。比如现在的网购，淘宝，京东等等。物品的存货数量，货品的价格，用户的账户余额之类的。这些数据都是存放在后台数据库中。或者最简单理解，我们现在微博，QQ 等账户的用户名和密码。在后台数据库必然有一张 user 表，字段起码有两个，即用户名和密码，然后我们的数据就一行一行的存在表上面。当我们登录的时候，我们填写了用户名和密码，这些数据就会被传回到后台去，去跟表上面的数据匹配，匹配成功了，你就能登录了。匹配不成功就会报错说密码错误或者没有此用户名等。这个就是数据库，数据库在生产环境就是用来干活的。凡是跟业务应用挂钩的，我们都使用数据库。

数据仓库则是 BI 下的其中一种技术。由于数据库是跟业务应用挂钩的，所以一个数据库不可能装下一家公司的所有数据。数据库的表设计往往是针对某一个应用进行设计的。比如刚才那个登录的功能，这张 user 表上就只有这两个字段，没有别的字段了。但是这张表符合应用，没有问题。但是这张表不符合分析。比如我想知道在哪个时间段，用户登录的量最多？哪个用户一年购物最多？诸如此类的指标。那就要重新设计数据库的表结构了。对于数据分析和数据挖掘，我们引入数据仓库概念。数据仓库的表结构是依照分析需求，分析维度，分析指标进行设计的。

数据库与数据仓库的区别实际讲的是 OLTP 与 OLAP 的区别。

操作型处理，叫联机事务处理 OLTP（On-Line Transaction Processing），也可以称面向交易的处理系统，它是针对具体业务在数据库联机的日常操作，通常对少数记录进行查询、修改。用户较为关心操作的响应时间、数据的安全性、完整性和并发支持的用户数等问题。传统的数据库系统作为数据管理的主要手段，主要用于操作型处理。

分析型处理，叫联机分析处理 OLAP（On-Line Analytical Processing）一般针对某些主题的历史数据进行分析，支持管理决策。

表 操作型处理与分析型处理的比较

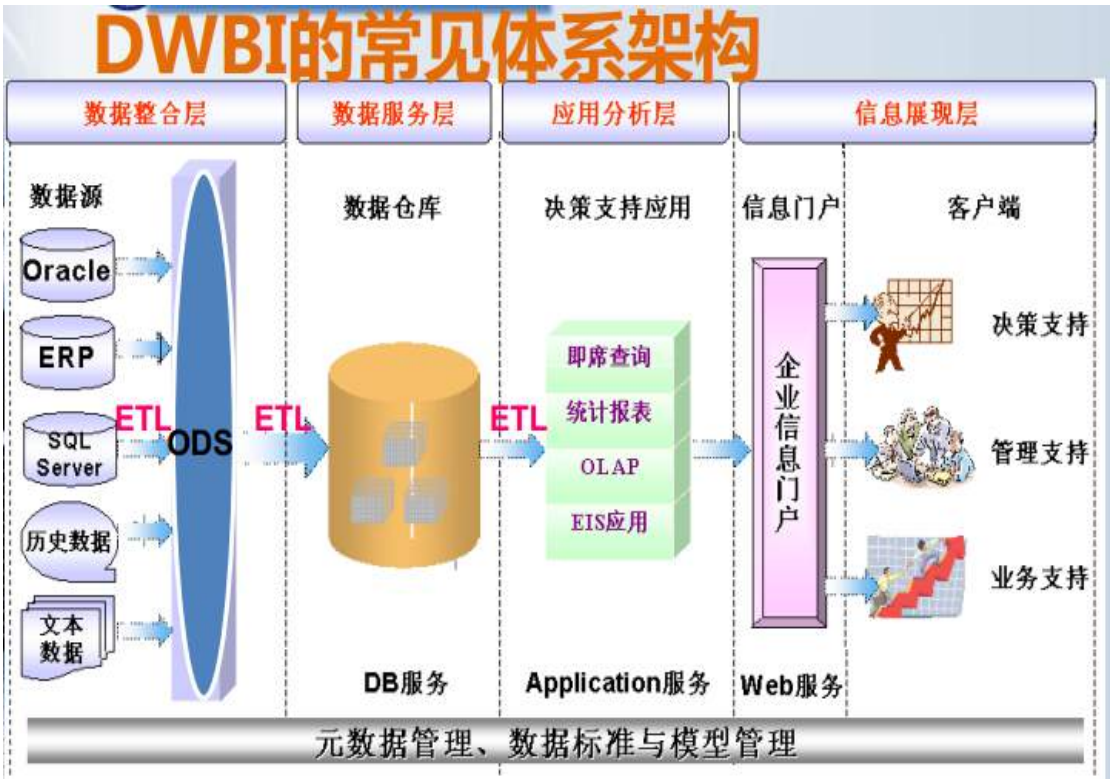
操作型处理	分析型处理
细节的	综合的或提炼的
实体——关系（E-R）模型	星型模型或雪花模型

存取瞬间数据	存储历史数据，不包含最近的数据
可更新的	只读、只追加
一次操作一个单元	一次操作一个集合
性能要求高，响应时间短	性能要求宽松
面向事务	面向分析
一次操作数据量小	一次操作数据量大
支持日常操作	支持决策需求
数据量小	数据量大
客户订单、库存水平和银行账户查询等	客户收益分析、市场细分等

11.6 数据仓库架构分层

11.6.1 数据仓库架构

数据仓库标准上可以分为四层：ODS（临时存储层）、PDW（数据仓库层）、DM（数据集市层）、APP（应用层）。



1）ODS 层：

为临时存储层，是接口数据的临时存储区域，为后一步的数据处理做准备。一般来说 ODS 层的数据和源系统的数据是同构的，主要目的是简化后续数据加工处理的工作。从数

据粒度上来说 ODS 层的数据粒度是最细的。ODS 层的表通常包括两类，一个用于存储当前需要加载的数据，一个用于存储处理完后的历史数据。历史数据一般保存 3-6 个月后需要清除，以节省空间。但不同的项目要区别对待，如果源系统的数据量不大，可以保留更长的时间，甚至全量保存；

2）PDW 层：

为数据仓库层，PDW 层的数据应该是一致的、准确的、干净的数据，即对源系统数据进行了清洗（去除了杂质）后的数据。这一层的数据一般是遵循数据库第三范式的，其数据粒度通常和 ODS 的粒度相同。在 PDW 层会保存 BI 系统中所有的历史数据，例如保存 10 年的数据。

3）DM 层：

为数据集市层，这层数据是面向主题来组织数据的，通常是星形或雪花结构的数据。从数据粒度来说，这层的数据是轻度汇总级的数据，已经不存在明细数据了。从数据的时间跨度来说，通常是 PDW 层的一部分，主要的目的是为了满足用户分析的需求，而从分析的角度来说，用户通常只需要分析近几年（如近三年的数据）的即可。从数据的广度来说，仍然覆盖了所有业务数据。

4）APP 层：

为应用层，这层数据是完全为了满足具体的分析需求而构建的数据，也是星形或雪花结构的数据。从数据粒度来说是高度汇总的数据。从数据的广度来说，则并不一定会覆盖所有业务数据，而是 DM 层数据的一个真子集，从某种意义上来说是 DM 层数据的一个重复。从极端情况来说，可以为每一张报表在 APP 层构建一个模型来支持，达到以空间换时间的目的数据仓库的标准分层只是一个建议性质的标准，实际实施时需要根据实际情况确定数据仓库的分层，不同类型的数据也可能采取不同的分层方法。

11.6.2 为什么要对数据仓库分层？

1）用空间换时间，通过大量的预处理来提升应用系统的用户体验（效率），因此数据仓库会存在大量冗余的数据。

2）如果不分层的话，如果源业务系统的业务规则发生变化将会影响整个数据清洗过程，工作量巨大。

3）通过数据分层管理可以简化数据清洗的过程，因为把原来一步的工作分到了多个步骤去完成，相当于把一个复杂的工作拆成了多个简单的工作，把一个大的黑盒变成了一个白

盒，每一层的处理逻辑都相对简单和容易理解，这样我们比较容易保证每一个步骤的正确性，当数据发生错误的时候，往往我们只需要局部调整某个步骤即可。

11.7 元数据介绍

当需要了解某地企业及其提供的服务时，电话黄页的重要性就体现出来了。元数据（Metadata）类似于这样的电话黄页。

1）元数据的定义

数据仓库的元数据是关于数据仓库中数据的数据。它的作用类似于数据库管理系统的数据字典，保存了逻辑数据结构、文件、地址和索引等信息。广义上讲，在数据仓库中，元数据描述了数据仓库内数据的结构和建立方法的数据。

元数据是数据仓库管理系统的重要组成部分，元数据管理器是企业级数据仓库中的关键组件，贯穿数据仓库构建的整个过程，直接影响着数据仓库的构建、使用和维护。

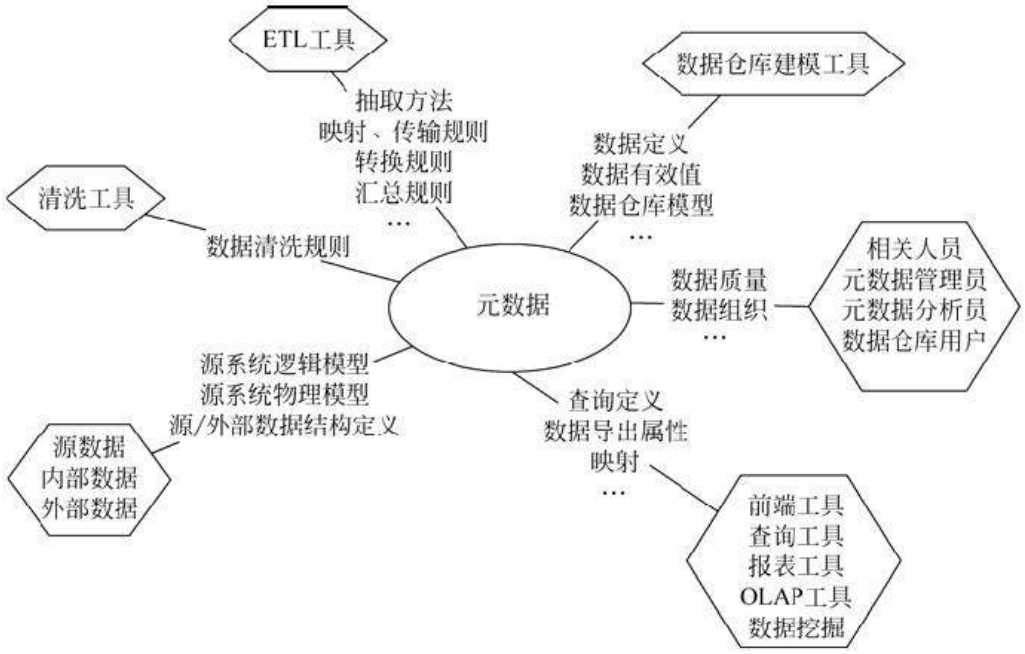
（1）构建数据仓库的主要步骤之一是 ETL。这时元数据将发挥重要的作用，它定义了源数据系统到数据仓库的映射、数据转换的规则、数据仓库的逻辑结构、数据更新的规则、数据导入历史记录以及装载周期等相关内容。数据抽取和转换的专家以及数据仓库管理员正是通过元数据高效地构建数据仓库。

（2）用户在使用数据仓库时，通过元数据访问数据，明确数据项的含义以及定制报表。

（3）数据仓库的规模及其复杂性离不开正确的元数据管理，包括增加或移除外部数据源，改变数据清洗方法，控制出错的查询以及安排备份等。

元数据可分为技术元数据和业务元数据。技术元数据为开发和管理数据仓库的 IT 人员使用，它描述了与数据仓库开发、管理和维护相关的数据，包括数据源信息、数据转换描述、数据仓库模型、数据清洗与更新规则、数据映射和访问权限等。而业务元数据为管理层和业务分析人员服务，从业务角度描述数据，包括商务术语、数据仓库中有什么数据、数据的位置和数据的可用性等，帮助业务人员更好地理解数据仓库中哪些数据是可用的以及如何使用。

由上可见，元数据不仅定义了数据仓库中数据的模式、来源、抽取和转换规则等，而且是整个数据仓库系统运行的基础，元数据把数据仓库系统中各个松散的组件联系起来，组成了一个有机的整体，如图所示



2）元数据的存储方式

元数据有两种常见存储方式：一种是以数据集为基础，每一个数据集有对应的元数据文件，每一个元数据文件包含对应数据集的元数据内容；另一种存储方式是以数据库为基础，即元数据库。其中元数据文件由若干项组成，每一项表示元数据的一个要素，每条记录为数据集的元数据内容。上述存储方式各有优缺点，第一种存储方式的优点是调用数据时相应的元数据也作为一个独立的文件被传输，相对数据库有较强的独立性，在对元数据进行检索时可以利用数据库的功能实现，也可以把元数据文件调到其他数据库系统中操作；不足是如果每一数据集都对应一个元数据文档，在规模巨大的数据库中则会有大量的元数据文件，管理不方便。第二种存储方式下，元数据库中只有一个元数据文件，管理比较方便，添加或删除数据集，只要在该文件中添加或删除相应的记录项即可。在获取某数据集的元数据时，因为实际得到的只是关系表格数据的一条记录，所以要求用户系统可以接受这种特定形式的数据。因此推荐使用元数据库的方式。

元数据库用于存储元数据，因此元数据库最好选用主流的关系数据库管理系统。元数据库还包含用于操作和查询元数据的机制。建立元数据库的主要好处是提供统一的数据结构和业务规则，易于把企业内部的多个数据集市有机地集成起来。目前，一些企业倾向建立多个数据集市，而不是一个集中的数据仓库，这时可以考虑在建立数据仓库（或数据集市）之前，先建立一个用于描述数据、服务应用集成的元数据库，做好数据仓库实施的初期支持工作，对后续开发和维护有很大的帮助。元数据库保证了数据仓库数据的一致性和准确性，为企业进行数据质量管理提供基础。

3）元数据的作用

在数据仓库中，元数据的主要作用如下。

（1）描述哪些数据在数据仓库中，帮助决策分析者对数据仓库的内容定位。

（2）定义数据进入数据仓库的方式，作为数据汇总、映射和清洗的指南。

（3）记录业务事件发生而随之进行的数据抽取工作时间安排。

（4）记录并检测系统数据一致性的要求和执行情况。

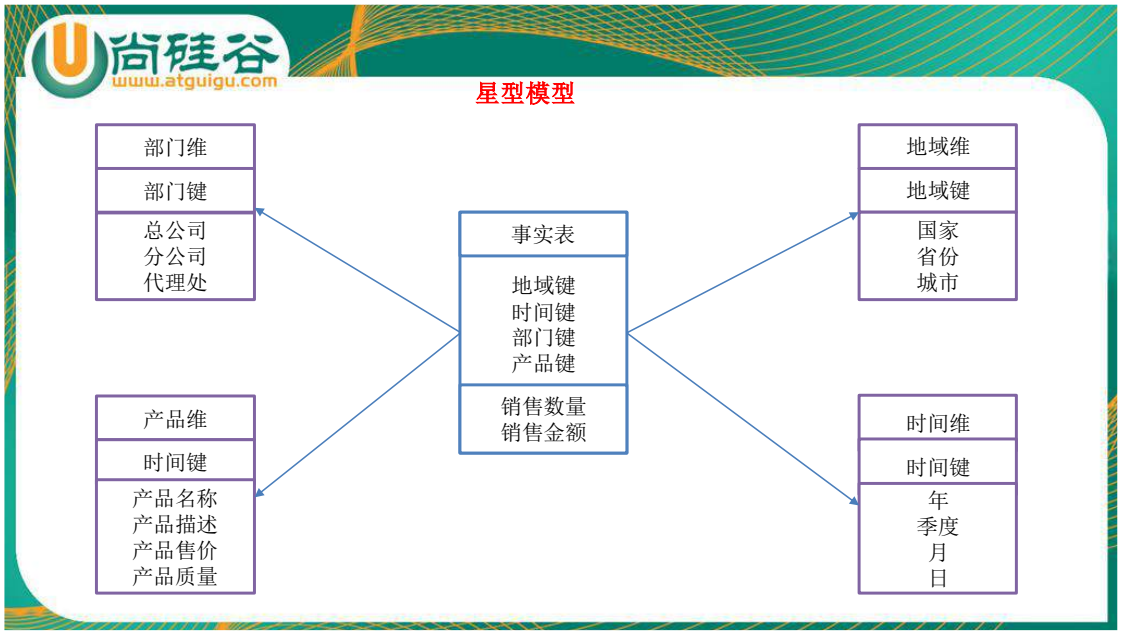
（5）评估数据质量。

11.8 星型模型和雪花模型

在多维分析的商业智能解决方案中，根据事实表和维度表的关系，又可将常见的模型分为星型模型和雪花型模型。在设计逻辑型数据的模型的时候，就应考虑数据是按照星型模型还是雪花型模型进行组织。

11.8.1 星型模型

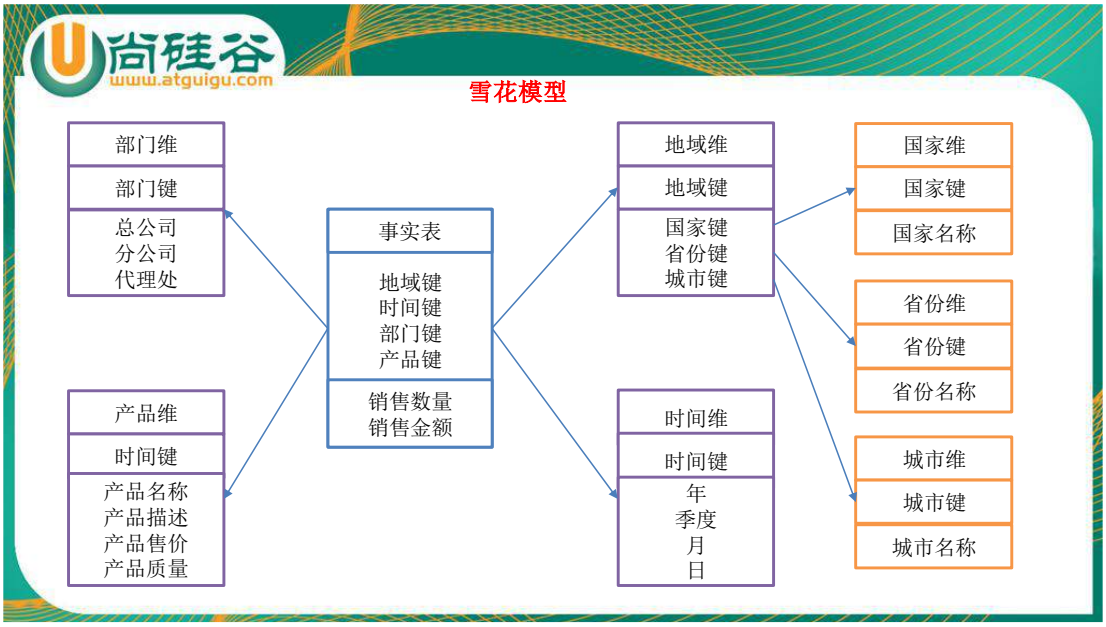
当所有维表都直接连接到“ 事实表”上时，整个图解就像星星一样，故将该模型称为星型模型。



星型架构是一种非正规化的结构，多维数据集的每一个维度都直接与事实表相连接，不存在渐变维度，所以数据有一定的冗余，如在地域维度表中，存在国家 A 省 B 的城市 C 以及国家 A 省 B 的城市 D 两条记录，那么国家 A 和省 B 的信息分别存储了两次，即存在冗余。

11.8.2 雪花模型

当有一个或多个维表没有直接连接到事实表上，而是通过其他维表连接到事实表上时，其图解就像多个雪花连接在一起，故称雪花模型。雪花模型是对星型模型的扩展。它对星型模型的维表进一步层次化，原有的各维表可能被扩展为小的事实表，形成一些局部的" 层次 " 区域，这些被分解的表都连接到主维度表而不是事实表。如图所示，将地域维表又分解为国家，省份，城市等维表。它的优点是：**通过最大限度地减少数据存储量以及联合较小的维表来改善查询性能。**雪花型结构去除了数据冗余。



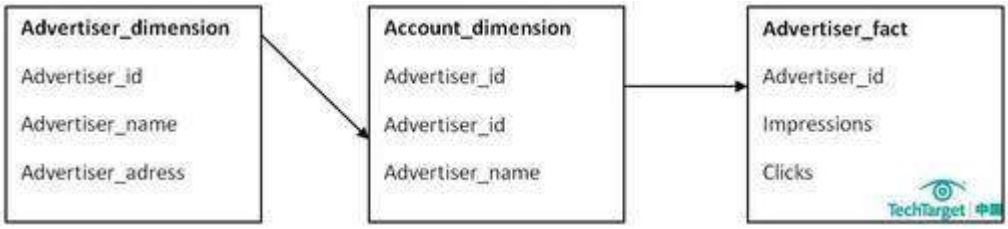
星型模型因为数据的冗余所以很多统计查询不需要做外部的连接，因此一般情况下效率比雪花型模型要高。星型结构不用考虑很多正规化的因素，设计与实现都比较简单。雪花型模型由于去除了冗余，有些统计就需要通过表的联接才能产生，所以效率不一定有星型模型高。正规化也是一种比较复杂的过程，相应的数据库结构设计、数据的 ETL、以及后期的维护都要复杂一些。**因此在冗余可以接受的前提下，实际运用中星型模型使用更多，也更有效率。**

11.8.3 星型模型和雪花模型对比

星形模型和雪花模型是数据仓库中常用到的两种方式，而它们之间的对比要从四个角度来进行讨论。

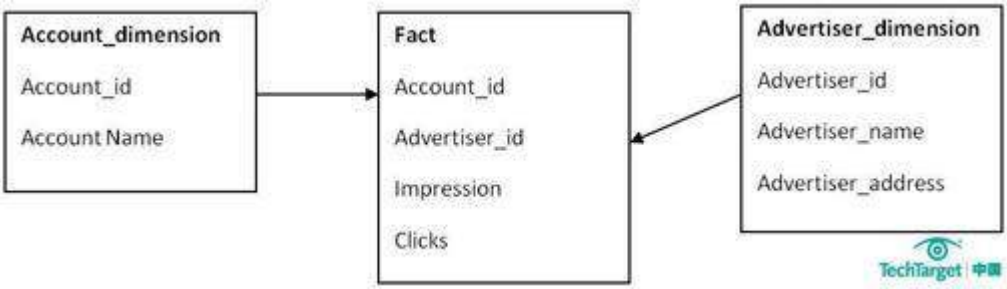
1）数据优化

雪花模型使用的是规范化数据，也就是说数据在数据库内部是组织好的，以便消除冗余，因此它能够有效地减少数据量。通过引用完整性，其业务层级和维度都将存储在数据模型之中。



雪花模型

相比较而言，星形模型使用的是反规范化数据。在星形模型中，维度直接指的是事实表，业务层级不会通过维度之间的参照完整性来部署。



星形模型

2）业务模型

主键是一个单独的唯一键(数据属性)，为特殊数据所选择。在上面的例子中，Advertiser_ID 就将是一个主键。外键(参考属性)仅仅是一个表中的字段，用来匹配其他维度表中的主键。在我们所引用的例子中，Advertiser_ID 将是 Account_dimension 的一个外键。在雪花模型中，数据模型的业务层级是由一个不同维度表主键-外键的关系来代表的。而在星形模型中，所有必要的维度表在事实表中都只拥有外键。

3）性能

第三个区别在于性能的不同。**雪花模型在维度表、事实表之间的连接很多，因此性能方面会比较低。**举个例子，如果你想要知道 Advertiser 的详细信息，雪花模型就会请求许多信息，比如 Advertiser Name、ID 以及那些广告主和客户表的地址需要连接起来，然后再与事实表连接。

而星形模型的连接就少的多，在这个模型中，如果你需要上述信息，你只要将 Advertiser 的维度表和事实表连接即可。

4）ETL

雪花模型加载数据集市，因此 ETL 操作在设计上更加复杂，而且由于附属模型的限制，不能并行化。

星形模型加载维度表，不需要再维度之间添加附属模型，因此 ETL 就相对简单，而且可以实现高度的并行化。

总结

雪花模型使得维度分析更加容易，比如“针对特定的广告主，有哪些客户或者公司是在线的?”星形模型用来做指标分析更适合，比如“给定的一个客户他们的收入是多少?”

第 13 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.CombineHiveInputFormat

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。

附录：Sqoop 常用命令及参数手册

这里给大家列出来了一部分 Sqoop 操作时的常用参数，以供参考，需要深入学习的可以参看对应类的源代码。

序号	命令	类	说明
1	import	ImportTool	将数据导入到集群
2	export	ExportTool	将集群数据导出
3	codegen	CodeGenTool	获取数据库中某张表数据生成 Java 并打包 Jar
4	create-hive-table	CreateHiveTableTool	创建 Hive 表
5	eval	EvalSqlTool	查看 SQL 执行结果

6	import-all-tables	ImportAllTablesTool	导入某个数据库下所有表到 HDFS 中
7	job	JobTool	用来生成一个 sqoop 的任务，生成后，该任务并不执行，除非使用命令执行该任务。
8	list-databases	ListDatabasesTool	列出所有数据库名
9	list-tables	ListTablesTool	列出某个数据库下所有表
10	merge	MergeTool	将 HDFS 中不同目录下面的数据合在一起，并存放在指定的目录中
11	metastore	MetastoreTool	记录 sqoop job 的元数据信息，如果不启动 metastore 实例，则默认的元数据存储目录为：~/.sqoop，如果要更改存储目录，可以在配置文件 sqoop-site.xml 中进行更改。
12	help	HelpTool	打印 sqoop 帮助信息
13	version	VersionTool	打印 sqoop 版本信息

命令&参数详解

刚才列举了一些 Sqoop 的常用命令，对于不同的命令，有不同的参数，让我们来一一列举说明。

首先来我们来介绍一下公用的参数，所谓公用参数，就是大多数命令都支持的参数。

公用参数：数据库连接

序号	参数	说明
1	--connect	连接关系型数据库的 URL
2	--connection-manager	指定要使用的连接管理类
3	--driver	Hadoop 根目录
4	--help	打印帮助信息
5	--password	连接数据库的密码
6	--username	连接数据库的用户名
7	--verbose	在控制台打印出详细信息

公用参数：import

序号	参数	说明

1	--enclosed-by <char>	给字段值前加上指定的字符
2	--escaped-by <char>	对字段中的双引号加转义符
3	--fields-terminated-by <char>	设定每个字段是以什么符号作为结束，默认为逗号
4	--lines-terminated-by <char>	设定每行记录之间的分隔符，默认是\n
5	--mysql-delimiters	Mysql 默认的分隔符设置，字段之间以逗号分隔，行之间以\n 分隔，默认转义符是\，字段值以单引号包裹。
6	--optionally-enclosed-by <char>	给带有双引号或单引号的字段值前后加上指定字符。

公用参数：export

序号	参数	说明
1	--input-enclosed-by <char>	对字段值前后加上指定字符
2	--input-escaped-by <char>	对含有转移符的字段做转义处理
3	--input-fields-terminated-by <char>	字段之间的分隔符
4	--input-lines-terminated-by <char>	行之间的分隔符
5	--input-optionally-enclosed-by <char>	给带有双引号或单引号的字段前后加上指定字符

公用参数：hive

序号	参数	说明
1	--hive-delims-replacement <arg>	用自定义的字符串替换掉数据中的\r\n 和\013 \010 等字符
2	--hive-drop-import-delims	在导入数据到 hive 时，去掉数据中的\r\n\013\010 这样的字符
3	--map-column-hive <arg>	生成 hive 表时，可以更改生成字段的数据类型
4	--hive-partition-key	创建分区，后面直接跟分区名，分区字段的默认类型为 string
5	--hive-partition-value <v>	导入数据时，指定某个分区的值
6	--hive-home <dir>	hive 的安装目录，可以通过该参数覆盖之前默认配置的目录
7	--hive-import	将数据从关系数据库中导入

		到 hive 表中
8	--hive-overwrite	覆盖掉在 hive 表中已经存在的数据
9	--create-hive-table	默认是 false，即，如果目标表已经存在了，那么创建任务失败。
10	--hive-table	后面接要创建的 hive 表,默认使用 MySQL 的表名
11	--table	指定关系数据库的表名

公用参数介绍完之后，我们来按照命令介绍命令对应的特有参数。

命令&参数：import

将关系型数据库中的数据导入到 HDFS（包括 Hive，HBase）中，如果导入的是 Hive，那么当 Hive 中没有对应表时，则自动创建。

1) 命令：

如：导入数据到 hive 中

```
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff \
--hive-import
```

如：增量导入数据到 hive 中，mode=append

```
append 导入：
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff \
--num-mappers 1 \
--fields-terminated-by "\t" \
--target-dir /user/hive/warehouse/staff_hive \
--check-column id \
--incremental append \
--last-value 3
```

尖叫提示：append 不能与--hive-等参数同时使用（Append mode for hive imports is not yet supported. Please remove the parameter --append-mode）

如：增量导入数据到 hdfs 中，mode=lastmodified

```
先在 mysql 中建表并插入几条数据：
mysql> create table company.staff_timestamp(id int(4), name varchar(255), sex varchar(255),
```

```
last_modified timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP);
mysql> insert into company.staff_timestamp (id, name, sex) values(1, 'AAA', 'female');
mysql> insert into company.staff_timestamp (id, name, sex) values(2, 'BBB', 'female');
先导入一部分数据：
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff_timestamp \
--delete-target-dir \
--m 1
再增量导入一部分数据：
mysql> insert into company.staff_timestamp (id, name, sex) values(3, 'CCC', 'female');
$ bin/sqoop import \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff_timestamp \
--check-column last_modified \
--incremental lastmodified \
--last-value "2017-09-28 22:20:38" \
--m 1 \
--append
```

尖叫提示：使用 lastmodified 方式导入数据要指定增量数据是要--append（追加）还是要--merge-key（合并）

尖叫提示：last-value 指定的值是会包含于增量导入的数据中

2) 参数：

序号	参数	说明
1	--append	将数据追加到 HDFS 中已经存在的 DataSet 中，如果使用该参数，sqoop 会把数据先导入到临时文件目录，再合并。
2	--as-avrodatafile	将数据导入到一个 Avro 数据文件中
3	--as-sequencefile	将数据导入到一个 sequence 文件中
4	--as-textfile	将数据导入到一个普通文本文件中
5	--boundary-query <statement>	边界查询，导入的数据为该参数的值（一条 sql 语句）所执行的结果区间内的数据。
6	--columns <col1, col2, col3>	指定要导入的字段

7	--direct	直接导入模式，使用的是关系数据库自带的导入导出工具，以便加快导入导出过程。
8	--direct-split-size	在使用上面 direct 直接导入的基础上，对导入的流按字节分块，即达到该阈值就产生一个新的文件
9	--inline-lob-limit	设定大对象数据类型的最大值
10	--m 或–num-mappers	启动 N 个 map 来并行导入数据，默认 4 个。
11	--query 或--e <statement>	将查询结果的数据导入，使用时必须伴随参--target-dir，--hive-table，如果查询中有 where 条件，则条件后必须加上$CONDITIONS 关键字
12	--split-by <column-name>	按照某一列来切分表的工作单元，不能与 --autoreset-to-one-mapper 连用（请参考官方文档）
13	--table <table-name>	关系数据库的表名
14	--target-dir <dir>	指定 HDFS 路径
15	--warehouse-dir <dir>	与 14 参数不能同时使用，导入数据到 HDFS 时指定的目录
16	--where	从关系数据库导入数据时的查询条件
17	--z 或--compress	允许压缩
18	--compression-codec	指定 hadoop 压缩编码类，默认为 gzip(Use Hadoop codec default gzip)
19	--null-string <null-string>	string 类型的列如果 null，替换为指定字符串
20	--null-non-string <null-string>	非 string 类型的列如果 null，替换为指定字符串
21	--check-column <col>	作为增量导入判断的列名
22	--incremental <mode>	mode：append 或 lastmodified
23	--last-value <value>	指定某一个值，用于标记增量导入的位置

命令&参数：export

从 HDFS（包括 Hive 和 HBase）中奖数据导出到关系型数据库中。

1) 命令：

如：

```
$ bin/sqoop export \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--table staff \
--export-dir /user/company \
--input-fields-terminated-by "\t" \
--num-mappers 1
```

2) 参数：

序号	参数	说明
1	--direct	利用数据库自带的导入导出工具，以便于提高效率
2	--export-dir <dir>	存放数据的 HDFS 的源目录
3	-m 或--num-mappers <n>	启动 N 个 map 来并行导入数据，默认 4 个
4	--table <table-name>	指定导出到哪个 RDBMS 中的表
5	--update-key <col-name>	对某一列的字段进行更新操作
6	--update-mode <mode>	updateonly allowinsert(默认)
7	--input-null-string <null-string>	请参考 import 该类似参数说明
8	--input-null-non-string <null-string>	请参考 import 该类似参数说明
9	--staging-table <staging-table-name>	创建一张临时表，用于存放所有事务的结果，然后将所有事务结果一次性导入到目标表中，防止错误。
10	--clear-staging-table	如果第 9 个参数非空，则可以在导出操作执行前，清空临时事务结果表

命令&参数：codegen

将关系型数据库中的表映射为一个 Java 类，在该类中有各列对应的各个字段。

如：

```
$ bin/sqoop codegen \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
```

```
--table staff \
--bindir /home/admin/Desktop/staff \
--class-name Staff \
--fields-terminated-by "\t"
```

序号	参数	说明
1	--bindir <dir>	指定生成的 Java 文件、编译成的 class 文件及将生成文件打包为 jar 的文件输出路径
2	--class-name <name>	设定生成的 Java 文件指定的名称
3	--outdir <dir>	生成 Java 文件存放的路径
4	--package-name <name>	包名，如 com.z，就会生成 com 和 z 两级目录
5	--input-null-non-string <null-str>	在生成的 Java 文件中，可以将 null 字符串或者不存在的字符串设置为想要设定的值（例如空字符串）
6	--input-null-string <null-str>	将 null 字符串替换成想要替换的值（一般与 5 同时使用）
7	--map-column-java <arg>	数据库字段在生成的 Java 文件中会映射成各种属性，且默认的数据类型与数据库类型保持对应关系。该参数可以改变默认类型，例如：--map-column-java id=long, name=String
8	--null-non-string <null-str>	在生成 Java 文件时，可以将不存在或者 null 的字符串设置为其他值
9	--null-string <null-str>	在生成 Java 文件时，将 null 字符串设置为其他值（一般与 8 同时使用）
10	--table <table-name>	对应关系数据库中的表名，生成的 Java 文件中的各个属性与该表的各个字段一一对应

命令&参数：create-hive-table

生成与关系数据库表结构对应的 hive 表结构。

命令：

如：

```
$ bin/sqoop create-hive-table \
--connect jdbc:mysql://linux01:3306/company \
```

```
--username root \
--password 123456 \
--table staff \
--hive-table hive_staff
```

参数：

序号	参数	说明
1	--hive-home <dir>	Hive 的安装目录，可以通过该参数覆盖掉默认的 Hive 目录
2	--hive-overwrite	覆盖掉在 Hive 表中已经存在的数据
3	--create-hive-table	默认是 false，如果目标表已经存在了，那么创建任务会失败
4	--hive-table	后面接要创建的 hive 表
5	--table	指定关系数据库的表名

命令&参数：eval

可以快速的使用 SQL 语句对关系型数据库进行操作，经常用于在 import 数据之前，了解一下 SQL 语句是否正确，数据是否正常，并可以将结果显示在控制台。

命令：

如：

```
$ bin/sqoop eval \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--query "SELECT * FROM staff"
```

参数：

序号	参数	说明
1	--query 或--e	后跟查询的 SQL 语句

命令&参数：import-all-tables

可以将 RDBMS 中的所有表导入到 HDFS 中，每一个表都对应一个 HDFS 目录

命令：

如：

```
$ bin/sqoop import-all-tables \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456 \
--warehouse-dir /all_tables
```

参数：

序号	参数	说明
1	--as-avrodatafile	这些参数的含义均和 import 对应的含义一致
2	--as-sequencefile	
3	--as-textfile	
4	--direct	
5	--direct-split-size <n>	
6	--inline-lob-limit <n>	
7	--m 或—num-mappers <n>	
8	--warehouse-dir <dir>	
9	-z 或--compress	
10	--compression-codec	

命令&参数：job

用来生成一个 sqoop 任务，生成后不会立即执行，需要手动执行。

命令：

如：

```
$ bin/sqoop job \
  --create myjob -- import-all-tables \
  --connect jdbc:mysql://linux01:3306/company \
  --username root \
  --password 123456
$ bin/sqoop job \
--list
$ bin/sqoop job \
--exec myjob
```

尖叫提示：注意 import-all-tables 和它左边的--之间有一个空格

尖叫提示：如果需要连接 metastore，则--meta-connect jdbc:hsqldb:hsql://linux01:16000/sqoop

参数：

序号	参数	说明
1	--create <job-id>	创建 job 参数
2	--delete <job-id>	删除一个 job
3	--exec <job-id>	执行一个 job
4	--help	显示 job 帮助
5	--list	显示 job 列表
6	--meta-connect <jdbc-uri>	用来连接 metastore 服务
7	--show <job-id>	显示一个 job 的信息
8	--verbose	打印命令运行时的详细信息

尖叫提示：在执行一个 job 时，如果需要手动输入数据库密码，可以做如下优化

```
<property>
    <name>sqoop.metastore.client.record.password</name>
    <value>true</value>
    <description>If true, allow saved passwords in the metastore.</description>
```

```
</property>
```

命令&参数：list-databases

命令：

如：

```
$ bin/sqoop list-databases \
--connect jdbc:mysql://linux01:3306/ \
--username root \
--password 123456
```

参数：与公用参数一样

命令&参数：list-tables

命令：

如：

```
$ bin/sqoop list-tables \
--connect jdbc:mysql://linux01:3306/company \
--username root \
--password 123456
```

参数：与公用参数一样

命令&参数：merge

将 HDFS 中不同目录下面的数据合并在一起并放入指定目录中

数据环境：

```
new_staff
1	AAA	male
2	BBB	male
3	CCC	male
4	DDD	male
old_staff
1	AAA	female
2	CCC	female
3	BBB	female
6	DDD	female
```

尖叫提示：上边数据的列之间的分隔符应该为\t，行与行之间的分割符为\n，如果直接复制，请检查之。

命令：

如：

```
创建 JavaBean：
$ bin/sqoop codegen \
--connect jdbc:mysql://linux01:3306/company \
```

```
--username root \
--password 123456 \
--table staff \
--bindir /home/admin/Desktop/staff \
--class-name Staff \
--fields-terminated-by "\t"

开始合并：
$ bin/sqoop merge \
--new-data /test/new/ \
--onto /test/old/ \
--target-dir /test/merged \
--jar-file /home/admin/Desktop/staff/Staff.jar \
--class-name Staff \
--merge-key id
结果：
1    AAA    MALE
2    BBB    MALE
3    CCC    MALE
4    DDD    MALE
6    DDD    FEMALE
```

参数：

序号	参数	说明
1	--new-data <path>	HDFS 待合并的数据目录，合并后在新的数据集中保留
2	--onto <path>	HDFS 合并后，重复的部分在新的数据集中被覆盖
3	--merge-key <col>	合并键，一般是主键 ID
4	--jar-file <file>	合并时引入的 jar 包，该 jar 包是通过 Codegen 工具生成的 jar 包
5	--class-name <class>	对应的表名或对象名，该 class 类是包含在 jar 包中的
6	--target-dir <path>	合并后的数据在 HDFS 里存放的目录

命令&参数：metastore

记录了 Sqoop job 的元数据信息，如果不启动该服务，那么默认 job 元数据的存储目录为~/.sqoop，可在 sqoop-site.xml 中修改。

命令：

如：启动 sqoop 的 metastore 服务

```
$ bin/sqoop metastore
```

参数：

序号	参数	说明
1	--shutdown	关闭 metastore