第5章

Hive数据仓库

5.1 Hive概述

5.1.1 Hive简介

Hive起源于Facebook公司，Facebook公司有大量的日志数据，而Hadoop是实现了MapReduce模式开源的分布式并行计算的框架，可轻松处理大规模数据。而对于Java语言熟悉的工程师来说开发MapReduce程序很容易，但对于其他语言使用者则难度较大。因此，Facebook开发团队想设计一种使用SQL对日志数据查询分析的工具，Hive就诞生于此。只要懂SQL，就能够利用Hive胜任大数据分析方面的工作，节省了开发人员的学习成本。

Hive是建立在Hadoop文件系统上的数据仓库分析系统，它提供了一系列工具，能够对存储在HDFS中的数据进行数据提取、转换和加载，可以存储、查询和分析存储在Hadoop中的大规模数据。Hive定义简单的类SQL(即HQL)，可以将结构化的数据文件映射为一张数据表，允许熟悉SQL的用户查询数据。

Hive采用了SQL的查询语言HQL，因此很容易将Hive理解为数据库。从结构上来看，Hive和数据库除了拥有类似的查询语言外，再无类似之处，Hive与MySQL对比如表5-1所示。

表5-1 Hive与MySQL对比

对 比 项	Hive	MySQL
查询语言	HQL	SQL
数据存储位置	HDFS	块设备、本地文件系统
数据格式	用户定义	系统决定

续表

对 比 项	Hive	MySQL
数据更新	不支持	支持
事务	不支持	支持
执行延迟	高	低
可扩展性	高	低
数据规模	大	小

5.1.2 Hive的架构

Hive的架构如图5-1所示。

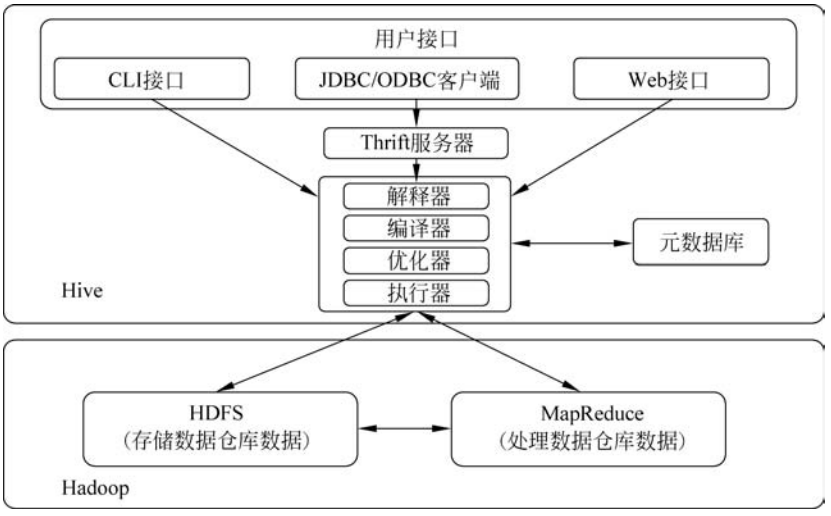


图5-1 Hive的架构

(1) 用户接口：主要包括CLI、JDBC/ODBC客户端和Web接口。其中，CLI为Shell命令行；JDBC/ODBC是Hive的Java接口实现，与传统数据库JDBC类似；Web接口通过浏览器访问Hive。

(2) 元数据库：Hive将元数据存储在数据库中(MySQL或者Derby)。Hive中的元数据包括表的名字、表的列和分区及其属性、表的属性(是否为外部表等)、表的数据所在目录等。

(3) Thrift服务器：允许客户端使用包括Java或其他很多种语言，通过编程的方式远程访问Hive。

(4) 解释器、编译器、优化器、执行器：完成HQL查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在HDFS中，并在随后调用执行MapReduce。

5.1.3 Hive的优缺点

1. Hive的优点

(1) 适合大数据的批量处理,解决了传统关系数据库在大数据处理上的瓶颈。

(2) Hive构建在Hadoop之上,充分利用了集群的存储资源、计算资源,最终实现并行计算。

(3) Hive学习使用成本低。Hive支持标准的SQL语法,免去了编写MapReduce程序的过程,减少了开发成本。

(4) 具有良好的扩展性,且能够实现和其他组件的结合使用。

2. Hive的缺点

(1) HQL的表达能力依然有限。由于本身SQL的不足,不支持迭代计算,有些复杂的运算用HQL不易表达,还需要单独编写MapReduce来实现。

(2) Hive的运行效率低、延迟高。Hive是转换成MapReduce任务来进行数据分析,MapReduce是离线计算,所以Hive的运行效率也很低,而且是高延迟。

(3) Hive调优比较困难。由于Hive是构建在Hadoop之上的,Hive的调优还要考虑MapReduce层面,因此Hive的整体调优比较困难。

5.2 Hive的安装

5.2.1 安装MySQL

(1) 下载MySQL的yum源,在CentOS命令行中输入如下命令:

```
wget http://dev.mysql.com/get/mysql57-community-release-el7-7.noarch.rpm
```

(2) 查看下载源中包含的rpm包,在CentOS命令行中输入如下命令:

```
rpm -qpl mysql57-community-release-el7-7.noarch.rpm
```

(3) 安装MySQL,在CentOS命令行中输入如下命令:

```
yum install -y mysql-community-server
```

(4) 启动mysqld服务,在CentOS命令行中输入如下命令:

```
systemctl start mysqld.service
```

(5) 查找MySQL初始密码,在CentOS命令行中输入如下命令:

```
grep "password"  /var/log/mysqld.log
```

运行结果如下所示:

```
[root@bigdata02 bin]#  grep "password" /var/log/mysqld.log
2020-06-06T01:40:43.399513Z 1 [Note] A temporary password is generated for root@localhost: Kel0d6tmrT/b
```

(6) 登录 MySQL,在 CentOS 命令行中输入 mysql -uroot -p,输入第(5)步查找的初始密码。

```
[root@bigdata02 bin]# mysql -uroot -p
Enter password:
```

(7) 设置安全级别,MySQL 命令如下:

```
set global validate_password_policy = 0;
```

(8) 设置密码长度,MySQL 命令如下:

```
set global validate_password_length = 4;
```

(9) 设置密码,MySQL 命令如下:

```
set password = password('1234');
```

(10) 创建数据库,MySQL 命令如下:

```
create database hive;
```

运行结果如下所示:

```
mysql> set global validate_password_policy=0;
Query OK, 0 rows affected (0.01 sec)

mysql> set global validate_password_length=4;
Query OK, 0 rows affected (0.00 sec)

mysql> set password=password('1234');
Query OK, 0 rows affected, 1 warning (0.00 sec)

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

(11) 创建 MySQL 用户,名称为 hadoop,设置用户密码,MySQL 命令如下:

```
grant all on *.* to hadoop@'%' identified by "hadoop";
grant all on *.* to hadoop@'localhost' identified by 'hadoop';
grant all on *.* to hadoop@'bigdata02' identified by 'hadoop';
flush privileges;
```

运行结果如下所示:

```
mysql> grant all on *.* to hadoop@'%' identified by "hadoop";
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> grant all on *.* to hadoop@'localhost' identified by 'hadoop';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> grant all on *.* to hadoop@'bigdata02' identified by 'hadoop';
Query OK, 0 rows affected, 1 warning (0.00 sec)

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

5.2.2 安装 Hive

(1) 下载安装包,下载镜像为 http://mirror.bit.edu.cn/apache/hive/hive-2.3.7/apache-hive-2.3.7-bin.tar.gz。

将安装包上传到服务器,并进行解压,其中 hadoop 是 Hive 的安装目录,命令如下:

```
tar -zxvf  apache-hive-2.3.7-bin.tar.gz  -C /hadoop/
```

(2) 配置 Hive 环境变量,在 etc/profile 文件中加入环境变量,内容如下:

```
export HIVE_HOME=/hadoop/apache-hive-2.3.7-bin/
export PATH=$HIVE_HOME/bin:$PATH
```

配置完成并保存后,刷新配置文件,执行命令 source /etc/profile。

(3) 在 Hadoop 下创建 hive 文件夹,在 CentOS 命令行上执行以下 Hadoop 命令:

```
hadoop fs -mkdir -p /hive/tmp
hadoop fs -mkdir -p /hive/warehouse
hadoop fs -chmod g+w /hive/tmp
hadoop fs -chmod g+w /hive/warehouse
hadoop fs -chmod -R 777 /user/hive/tmp
```

(4) 修改 Hive 配置文件,在 CentOS 命令行上执行以下命令:

```
cd /hadoop/apache-hive-2.3.7-bin/etc
cp hive-env.sh.template hive-env.sh
cp hive-default.xml.template hive-site.xml
cp hive-log4j2.properties.template hive-log4j2.properties
cp hive-exec-log4j2.properties.template hive-exec-log4j2.properties
```

在 hive-env.sh 文件中加入 JAVA_HOME、HADOOP_HOME 配置,内容如下:

```
export JAVA_HOME=/usr/java/jdk1.8.0_161
export HADOOP_HOME=/hadoop/hadoop-2.7.7
```

在 hive-site.xml 文件中加入以下配置,注意 XML 文件中的 ConnectionUserName 和 ConnectionPassword 配置的是 MySQL 的用户名和密码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
    <property>
        <name>hive.exec.scratchdir</name>
        <value>/hive/tmp</value>
    </property>
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/hive/warehouse</value>
    </property>
    <property>
        <name>hive.querylog.location</name>
```

```
            <value>/hive/log</value>
        </property>
        <!-- 配置 MySQL 数据库连接信息 -->
        <property>
            <name>javax.jdo.option.ConnectionURL</name>
<value> jdbc:mysql://localhost:3306/metastore?createDatabaseIfNotExist=true&amp;
characterEncoding=UTF-8&amp;useSSL=false</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionDriverName</name>
            <value>com.mysql.jdbc.Driver</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionUserName</name>
            <value>hadoop</value>
        </property>
        <property>
            <name>javax.jdo.option.ConnectionPassword</name>
            <value>hadoop</value>
        </property>
</configuration>
```

(5) 初始化 Hive,在 CentOS 命令行上执行以下命令:

```
./schematool -dbType mysql -initSchema hive hive
```

(6) 下载 mysql-connector-java-5.1.46.jar,并复制到 Hive 安装目录的 lib 目录下。

(7) 启动 Hive,在 CentOS 命令行上执行命令 hive,运行结果如下所示。

```
./hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/hadoop/apache-hive-2.3.7-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLogger
Binder.class]
SLF4J: Found binding in [jar:file:/hadoop/hadoop-2.7.7/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Sta
ticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in file:/hadoop/apache-hive-2.3.7-bin/conf/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
hive>
```

5.3 Hive 数据库相关操作

Hive 是基于 Hadoop 构建的一套数据仓库分析系统,它提供了丰富的 SQL 查询方式来分析存储在 Hadoop 分布式文件系统中的数据,可以将结构化的数据文件映射为一张数据库表,并提供完整的 SQL 查询功能,可以将 SQL 语句转换为 MapReduce 任务进行运行,通过自己的 SQL 去查询分析需要的内容。这套 SQL 简称 HQL。HQL 可以使不熟悉 MapReduce 的用户很方便地利用 SQL 查询、汇总和分析数据。而 MapReduce 开发人员可以将自己编写的 Mapper 和 Reducer 作为插件来支持 Hive 做更复杂的数据分析。Hive 中的数据库和常见的关系数据库中的数据库的作用几乎是一样的,在生产环境

中，如果表非常多，一般都会用数据库把表组织起来，形成逻辑组，这样可以有效防止大规模集群中表名冲突的问题。Hive 数据库也是用来组织数据表的，它的本质就是数据仓库下的一个目录。

5.3.1 Hive 的数据类型

Hive 的内置数据类型可以分为两大类，分别是基础数据类型和复杂数据类型，Hive 的基础数据类型如表 5-2 所示，Hive 的复杂数据类型如表 5-3 所示。

表 5-2 Hive 的基础数据类型

数 据 类 型	描　　述
tinyint	1 字节的有符号整数
smallint	2 字节有符号整数
int	4 字节有符号整数
bigint	8 字节有符号整数
float	4 字节单精度浮点数
double	8 字节双精度浮点数
double precision	double 的别名，从 Hive 2.2.0 开始提供
decimal	任意精度的带符号小数
numeric	同样是 decimal，从 Hive 3.0 开始
timestamp	精度到纳秒的时间戳
date	以年/月/日形式描述的日期
interval	表示时间间隔
string	字符串
varchar	同 string，字符串长度不固定
char	固定长度的字符串
boolean	布尔型

表 5-3 Hive 的复杂数据类型

数 据 类 型	描　　述
array	一组有序字段，字段类型必须相同
map	一组无序键值对。键的类型必须是原子类型，值可以是任意类型，同一个映射的键的类型必须相同，值的类型也必须相同
struct	一组命名的字段，字段的类型可以不同

5.3.2 Hive 基础 SQL 语法

1. DDL 操作

DDL(Data Definition Language)是数据定义语言，与关系数据库操作相似。

1) 创建数据库

例如，创建一个名为 bigdata 的数据库，DDL 语句为“create database bigdata;”，运行

结果如下：

```
hive> create database  bigdata;
OK
Time taken: 0.122 seconds
```

显示数据库命令，DDL 语句为“show database;”，运行结果如下：

```
hive> show databases;
OK
default
hhao
Time taken: 3.343 seconds, Fetched: 2 row(s)
```

使用数据库，例如使用一个名为 bigdata 的数据库，DDL 语句为“use bigdata;”，运行结果如下：

```
hive> use bigdata;
OK
Time taken: 0.013 seconds
```

2）创建表

Hive 在创建表时默认创建内部表，将数据移动到数据仓库指向的路径，而创建外部表（需要加关键字 external），仅记录数据所在的路径，不对数据的位置做任何改变；Hive 删除表时，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。创建表的语法如下：

```
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 exist 选项来忽略这个异常
```

- external：关键字，可以让用户创建一个外部表。
- comment：可以为表与字段增加描述。
- partitioned by：分区。
- clustered by：数据汇总。
- sorted by：按某列排序。
- buckets：分桶，设置词句分桶才有效，如“set hive.enforce.bucketing=true;”。
- row format delimited：按分割格式读取文件。
  [fields terminated by char]：每个列字段通过什么分割。
  [collection items terminated by char]：每个键值之间分割符。
  [map keys terminated by char]：每个键值对分割符。
  [lines terminated by char]：每行之间通过什么分割。
- stored as：存储为不同文件格式。
  [textfile]：文本文件。
  [sequencefile]：序列化文件。
  [rcfile]：面向列的数据格式文件。
  [inputformat input_format_classname outputformat output_format_classname]：自定义的输入输出流。
- location：在建表的同时指定一个指向实际数据的路径。

(1) 创建内部表。例如,创建一个名为 student 的内部表,DDL 语句为:

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

运行结果如下:

```
hive> create table if not exists student(
    > id int,
    > name string,
    > birthday timestamp)
    > row format delimited
    > fields terminated by '\t'
    > lines terminated by '\n'
    > stored as textfile
    > location '/hive/warehouse/ student';
OK
Time taken: 0.267 seconds
```

查看当前数据库中表的命令为 show tables,运行结果如下:

```
hive> show tables;
OK
student
Time taken: 0.023 seconds, Fetched: 1 row(s)
```

查看数据库中的表,使用命令 show tables in bigdata。

查看数据表的结构信息使用命令 desc ,比如查看 student 表结构信息的命令为 desc student,运行结果如下:

```
hive> desc student;
OK
id                      int
name                    string
birthday                timestamp
Time taken: 0.051 seconds, Fetched: 3 row(s)
```

通过复制另一张表的表结构来创建表。要注意使用这种复制方式创建的表只复制表结构,不复制表中的数据,比如参照 bigdata 数据库中的 student 表的结构创建表名为 student_copy 的数据表,DDL 语句为“create table if not exists student_copy like student;”,运行结果如下:

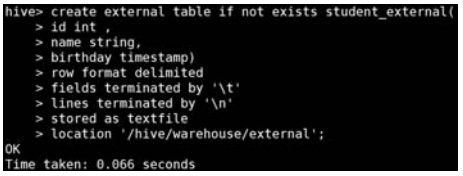
```
hive> create table if not exists student_copy like student;
OK
Time taken: 0.315 seconds
```

(2) 创建外部表,例如创建一个名为 student_external 的外部表,DDL 语句为:

```
create external table if not exists student_external(
id int,
name string,
birthday timestamp)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
```

```
stored as textfile
location '/hive/warehouse/external';
```

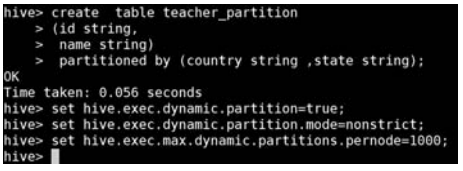
运行结果如下：



(3) 创建分区表。例如创建一个名为 teacher_partition 的分区表，并设置成动态建立分区，DDL 语句为：

```
create  table teacher_partition
 (id string,
 name string)
 partitioned by (country string ,state string);
set hive.exec.dynamic.partition = true;                    #开启动态分区,默认是 false
      set hive.exec.dynamic.partition.mode = nonstrict;    #开启允许所有分区都是动态的,否
                                                           #则必须要有静态分区才能使用
set hive.exec.max.dynamic.partitions.pernode = 1000;       #动态分区最大数量
```

运行结果如下：



(4) 创建桶表。在 Hive 分区表中，分区中的数据量过于庞大时，建议使用桶表。桶表是对某一列数据进行 Hash 取值以将数据打散，然后放到不同文件中存储。在分桶时，对指定字段的值进行 Hash 运算得到 Hash 值，并使用 Hash 值除以桶的个数再取余运算得到的值进行分桶，保证每个桶中有数据但每个桶中的数据不一定相等。做 Hash 运算时，Hash()函数的选择取决于分桶字段的数据类型。桶表按某一列的值将记录进行分桶存放，即分文件存放，即将大表分解成一系列小表，涉及 Join 操作时，可以在桶与桶间关联即可，这样便减小了 Join 的数据量，提高了执行效率。

创建一个名为 teacher_bucket 的桶表，按照 id 分为 4 个桶，DDL 语句为：

```
create table teacher_bucket(
 id string,
 name string,
 country string,
 state string)
 clustered by(id) into 4 buckets;
```

运行结果如下：

```
hive> use bigdata;
OK
Time taken: 3.3 seconds
hive> create table teacher_bucket(
    >  id string,
    >  name string,
    >  country string,
    >  state string)
    >  clustered by(id) into 4 buckets;
OK
Time taken: 0.488 seconds
hive> show tables;
OK
sp_trans
student
student_external
teacher
teacher_bucket
teacher_partition
Time taken: 0.08 seconds, Fetched: 6 row(s)
```

3）修改表

在 Hive 中，使用 alter table 子句来修改表的属性，实际上是修改表的元数据，而不会修改表中实际的数据。

（1）对表重命名。例如，通过 rename to 命令将表 student_copy 表重命名为 student01，DDL 语句及运行结果如下：

```
hive> alter table student_copy rename to student01;
OK
Time taken: 0.081 seconds
```

（2）修改列信息。例如，通过 change column 命令对表 student01 的 id 字段重命名，将 id 字段重命名为 uid，DDL 语句及运行结果如下：

```
hive> alter table student01
    > change column id uid int;
OK
Time taken: 0.074 seconds
hive> desc student01;
OK
uid                     int
name                    string
birthday                timestamp
Time taken: 0.029 seconds, Fetched: 3 row(s)
```

（3）修改字段的类型。例如，通过 change column 命令将 uid 的类型由 int 改为 string，DDL 语句及运行结果如下：

```
hive> desc student01;
OK
uid                     int
name                    string
birthday                timestamp
Time taken: 0.029 seconds, Fetched: 3 row(s)
hive> alter table student01
    > change column  uid  uid string;
OK
Time taken: 0.066 seconds
hive> desc student01;
OK
uid                     string
name                    string
birthday                timestamp
Time taken: 0.028 seconds, Fetched: 3 row(s)
```

（4）增加列。例如，通过 add columns 命令将 student01 表添加 grade 和 class 两个列，DDL 语句为“alter table student01 add columns(grade string,class string);”，DDL 语句及运行结果如下：

```
hive> desc student01;
OK
id                      int
name                    string
birthday                timestamp
Time taken: 0.022 seconds, Fetched: 3 row(s)
hive> alter table student01 add columns(grade string,class string);
OK
Time taken: 0.074 seconds
hive> desc student01;
OK
id                      int
name                    string
birthday                timestamp
grade                   string
class                   string
Time taken: 0.022 seconds, Fetched: 5 row(s)
```

(5) 删除或替换列。例如,通过 replace columns 命令删除或替换 student01 表中的 grade 和 class 两个列,DDL 语句及运行结果如下:

```
hive> alter table student01 replace columns(grade int,class string);
OK
Time taken: 0.059 seconds
```

4) 删除表。

例如,使用 drop 命令删除 student01 表,DDL 语句及运行结果如下:

```
hive> show tables;
OK
student
student01
Time taken: 0.013 seconds, Fetched: 2 row(s)
hive> drop table student01;
OK
Time taken: 0.113 seconds
hive> show tables;
OK
student
Time taken: 0.011 seconds, Fetched: 1 row(s)
```

2. DML 操作

DML(Data Manipulation Language)即数据操作语言,是用来对 Hive 数据库中的数据进行操作的语言。数据操作主要是如何向表中装载数据和如何将表中的数据导出,主要操作命令有 load、insert 等,insert 语句的语法基本与标准 SQL 相同。

1) 装载数据

(1) 从本地路径下一次性装载大量的数据的方式。以上面建立的内部表 teacher 为例,DML 语句为"load data local inpath '/hadoop/data' into table teacher;"。该语句会把本地的/ hadoop /data 文件夹下的所有文件都追加到 teacher 表,其实际上就是一个文件的移动。如果加上 local 关键字,Hive 会将本地文件复制一份然后再上传到指定目录,如果不加 local 关键字,Hive 只会将 HDFS 上的数据移动到指定目录。DML 语句及运行结果如下:

```
hive> load data local inpath '/data.txt'  into table teacher;
Loading data to table bigdata.teacher
OK
Time taken: 0.3 seconds
hive> select * from teacher;
OK
1       tom     US      CA
2       jack    US      CB
3       mike    CA      BB
4       ariana  CA      BC
Time taken: 0.075 seconds, Fetched: 4 row(s)
```

(2) 覆盖表中已有的记录。需要加上 overwrite 关键字。以 teacher 为例,DML 语句为"load data local inpath '/data. txt' overwrite into table teacher;"。

(3) 装载分区表数据。分区表在 DML 命令中需要指定分区,以分区表 teacher_partition 为例,DML 语句为"load data local inpath '/data.txt' overwrite into table teacher_partition partition (country='US',state='CA');",运行结果如下:

```
hive> load data local inpath '/data.txt' overwrite into table teacher_partition  partition (country='US',state=
'CA');
Loading data to table bigdata.teacher_partition partition (country=US, state=CA)
OK
Time taken: 0.778 seconds
```

2) 插入数据

(1) 通过标准 SQL 插入数据。例如,向 teacher 表插入一条记录,DML 语句为"insert into teacher(id,name,city,state) values(5,'aliy','CA','BD');",运行结果如下:

```
hive> insert into teacher(id,name,city,state) values(5,'aliy','CA','BD');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider u
sing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20200614222457_27db22fc-05e7-48d7-be95-4664f696c393
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1591063491031_0025, Tracking URL = http://bigdata01:8088/proxy/application_159106349
1031_0025/
Kill Command = /hadoop/hadoop-2.7.7/bin/hadoop job  -kill job_1591063491031_0025
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-06-14 22:25:05,434 Stage-1 map = 0%,  reduce = 0%
2020-06-14 22:25:10,636 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.9 sec
MapReduce Total cumulative CPU time: 1 seconds 900 msec
Ended Job = job_1591063491031_0025
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://ns1/hive/warehouse/bigdata.db/teacher/.hive-staging_hive_2020-06-14_22-
24-57_330_5921734926122211004-1/-ext-10000
Loading data to table bigdata.teacher
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 1.9 sec   HDFS Read: 4265 HDFS Write: 84 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 900 msec
OK
Time taken: 15.682 seconds
```

(2) 通过查询语句向表中插入数据。例如,将 teacher 表的数据插入 teacher01 表中,DML 语句为"insert into teacher01 select * from teacher;",运行结果如下:

```
hive> insert into teacher01 select * from teacher;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider u
sing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20200614223104_bbf17a19-d19a-48d0-97df-b2a4b0ab3024
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1591063491031_0027, Tracking URL = http://bigdata01:8088/proxy/application_159106349
1031_0027/
Kill Command = /hadoop/hadoop-2.7.7/bin/hadoop job  -kill job_1591063491031_0027
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-06-14 22:31:09,342 Stage-1 map = 0%,  reduce = 0%
2020-06-14 22:31:15,531 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.97 sec
MapReduce Total cumulative CPU time: 1 seconds 970 msec
Ended Job = job_1591063491031_0027
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://ns1/hive/warehouse/bigdata.db/teacher01/.hive-staging_hive_2020-06-14_2
2-31-04_376_5329050658129452336-1/-ext-10000
Loading data to table bigdata.teacher01
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 1.97 sec   HDFS Read: 4167 HDFS Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 970 msec
OK
Time taken: 12.455 seconds
```

注意,通过查询语句向表中插入数据时要保证两个表的格式是一致的,这里的一致指的是要保证查询结果的格式和插入表的格式一致。加 overwrite 表示对源表数据进行覆盖。如果是分区表,则必须用 partition 指定分区。

(3) 导出数据。使用 insert 子句可以将数据导出到本地(加 local)或 HDFS,例如将 teacher 表中的数据导出到本地的'/hadoop/data'目录下,DML 语句为"insert overwrite

local directory '/hadoop/data' select * from teacher;",运行结果如下:

```
hive> insert overwrite local directory '/hadoop/data' select * from teacher;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider u
sing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20200614224828_5c795021-370f-4d0b-af20-856870fe56b0
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1591063491031_0030, Tracking URL = http://bigdata01:8088/proxy/application_159106349
1031_0030/
Kill Command = /hadoop/hadoop-2.7.7/bin/hadoop job  -kill job_1591063491031_0030
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-06-14 22:48:36,178 Stage-1 map = 0%,  reduce = 0%
2020-06-14 22:48:40,407 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.53 sec
MapReduce Total cumulative CPU time: 1 seconds 530 msec
Ended Job = job_1591063491031_0030
Moving data to local directory /hadoop/data
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 1.53 sec   HDFS Read: 3706 HDFS Write: 67 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 530 msec
OK
Time taken: 14.229 seconds
```

(4) 删除、更新数据。Hive 从 0.14 版本开始支持事务和行级更新,但默认是不支持的,需要一些附加配置。要想支持行级 update、delete,需要配置 Hive 支持事务。在此介绍表全删除命令 truncate,DML 语句为"truncate table teacher;",运行结果如下:

```
hive> select * from teacher;
OK
5       aliy    CA      BD
1       tom     US      CA
2       jack    US      CB
3       mike    CA      BB
4       ariana  CA      BC
Time taken: 0.091 seconds, Fetched: 5 row(s)
hive> truncate table teacher;
OK
Time taken: 0.072 seconds
hive> select * from teacher;
OK
Time taken: 0.085 seconds
hive>
```

3. DQL 操作

DQL(Data Query Language)即数据查询语言,实现数据的简单查询,主要操作命令有 select、where 等。可以在查询时对数据进行排序、分组等操作,如 sort by、order by、group by 等,同时支持嵌套查询,例如执行 DQL 语句对 teacher 表按城市名称进行分组,同时查询出教师在同一城市数量大于 1 的城市。DML 语句为"select count(*) from teacher group by city having count(*)>1;"。

```
hive> select city,count(*) from teacher group by city having count(*)>1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider u
sing a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20200615101744_1520af7e-0223-4938-849a-b4e24c72f723
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1591063491031_0032, Tracking URL = http://bigdata01:8088/proxy/application_159106349
1031_0032/
Kill Command = /hadoop/hadoop-2.7.7/bin/hadoop job  -kill job_1591063491031_0032
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-06-15 10:17:51,003 Stage-1 map = 0%,  reduce = 0%
2020-06-15 10:17:55,196 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.54 sec
2020-06-15 10:18:00,326 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.26 sec
MapReduce Total cumulative CPU time: 4 seconds 260 msec
Ended Job = job_1591063491031_0032
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.26 sec   HDFS Read: 8705 HDFS Write: 121 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 260 msec
OK
CA      3
US      2
Time taken: 18.241 seconds, Fetched: 2 row(s)
```

5.4 本章小结

本章主要介绍了 Hive 的架构、安装和基本操作，因为 Hive 的安装需要 MySQL 数据库，因此也相应介绍了 MySQL 的安装。本章的重点是 Hive 的基本操作，主要是 DDL、DML、DQL 基本操作。