第11章

Hive及Hadoop群

11

本章内容

- 什么是Hive
- Hive的安装
- 在数据仓库中使用Hive
- 其他与Hadoop相关的软件包

即使像Hadoop这样强大的工具，也不能满足每个人的需求。许多项目如雨后春笋般涌现出来，为特定目的扩展了Hadoop。那些突出并且得到很好维护的项目已经正式成为Apache Hadoop项目下的子项目。[①]这些子项目如下所示。

- Pig——一种高级数据流语言。
- Hive——一种类SQL数据仓库基础设施。
- HBase——一种模仿Google Bigtable的分布式的、面向列的数据库。
- ZooKeeper——一种用于管理分布式应用之间共享状态的可靠的协同系统。
- Chukwa——一种用于管理大型分布式系统的数据收集系统。

我们在第10章详细讨论了Pig，本章将学习Hive。此外，11.2节还会简单介绍其他与Hadoop相关的项目。其中一些（例如Cascading和CloudBase）与Apache没有关系。有些则仍在新生阶段（例如Hama和Mahout）。你会在第12章中看到这些工具在实战中的一些应用。

11.1 Hive

Hive[②]是建立在Hadoop基础上的数据仓库软件包。在开始阶段，它被Facebook用于处理大量的用户数据和日志数据。它现在是Hadoop的子项目并有许多的贡献者。其目标用户仍然是习惯了SQL的数据分析师，他们需要在Hadoop规模的数据上做即席查询、汇总和数据分析[③]。通过称为

① 我们至今在本书中所提到的“Hadoop”（HDFS和MapReduce）在技术上被称为Apache Hadoop的子项目“Hadoop Core”，不过人们倾向于把它通俗地称为Hadoop。

② http://hadoop.apache.org/hive/。

③ 请注意因为Hive构建在Hadoop的基础之上，它仍然设计用作处理低延迟与批量类型的作业。故而它并不会直接取代传统的SQL数据仓库，比如那些由Oracle提供的数据仓库。

HiveQL的类SQL语言，你可以发起一个查询来实现与Hive的交互。例如，一个从user表获取所有活跃用户的查询如下所示：

```
INSERT OVERWRITE TABLE user_active
SELECT user.*
FROM user
WHERE user.active = 1;
```

Hive的设计体现出它是一个管理和查询结构化数据的系统。通过专注结构化数据，Hive可以实现MapReduce一般所不具备的某些优化和可用性功能。受到SQL影响的Hive语言让用户可以脱离MapReduce编程的复杂性。它沿用了关系数据库中的常见概念，如表、行、列和schema，以便于学习。此外，虽然Hadoop天生支持平坦文件，但Hive可以使用目录结构来“划分”数据，以提高某些查询的性能。为支持这些额外的功能，Hive中有一个全新且重要的组件，它就是用于存储schema信息的metastore。这个metastore通常只有在关系数据库中才有。

你可以通过几种方法与Hive交互，包括Web GUI和JDBC（Java数据库连接）接口。但是大多数交互倾向于利用命令行界面（CLI），这正是我们所关注的焦点。你可以在图11-1上看到Hive的高层体系结构框图。

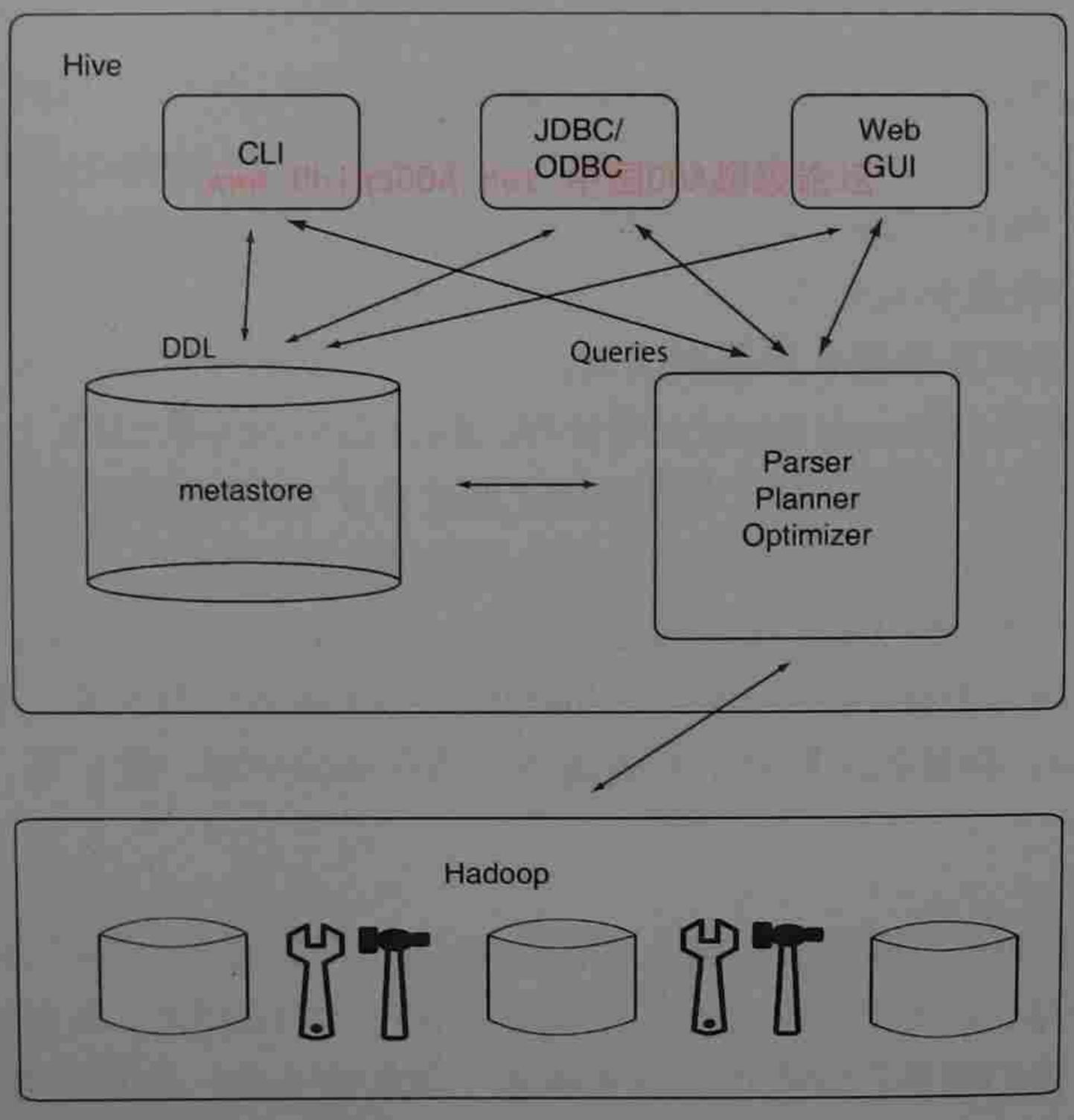

图11-1　Hive体系结构。查询在Hadoop上被解析与执行。metastore是一个重要组件，它用于确定查询如何执行

11.1.1　安装与配置Hive

Hive需要Java 1.6和Hadoop 0.17及以上版本的支持。你可以在http://hadoop.apache.org/hive

releases.html上找到Hive的最新版本。下载并将压缩包解压到HIVE_HOME目录。Hive需要Hadoop已经启动并运行。此外，还需要在HDFS中为Hive设置几个目录备用。

```
bin/hadoop fs -mkdir /tmp
bin/hadoop fs -mkdir /user/hive/warehouse
bin/hadoop fs -chmod g+w /tmp
bin/hadoop fs -chmod g+w /user/hive/warehouse
```

如果把数据完全交由Hive管理，Hive会把它们存储在/user/hive/warehouse目录下。它还会自动压缩数据并加入特殊的目录结构（例如分区）以提高查询性能。如果你计划使用Hive查询数据，最好让Hive来管理它。但如果数据已经在HDFS的其他目录中，而你还想让它们继续存在那里，Hive也可以直接对它们进行操作。此时，Hive会照原样读取数据，而不会为了查询处理而优化数据存储。有的用户不理解这种区别，他们以为Hive需要数据采用某些特殊的Hive格式。这绝对不是真的。

Hive将元数据存储在标准关系数据库中。Hive自带了Derby，它是一个开源、轻量、嵌入式的SQL数据库[①]，它与Hive一起安装并运行在客户端机器上。如果你是Hive的唯一用户，使用默认设置就足够了。但除非是在做初始测试和评价，最有可能的情况还是将Hive部署在多用户环境中，而你不会想让每个用户都创建自己的元数据。这就需要有一个集中的地方来存储元数据。通常会使用一个共享的SQL数据库，如MySQL，但任何符合JDBC的数据库都是可以的。你还需要一个数据库服务器，在其上创建一个数据库作为Hive的metastore。这个数据库通常命名为metastore_db。一旦创建了这个数据库，就把每个Hive的安装都配置成指向它，将它作为metastore。这个安装的配置是通过修改文件hive-site.xml和jpox.properties来完成的，它们都放在$HIVE_HOME/conf目录下。原始的安装中没有hive-site.xml文件，你必须创建它。此文件中的属性会重写hive-default.xml中的属性，类似于hadoop-site.xml重写hadoop-default.xml。文件hive-site.xml会重写3个属性，如下所示：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
 <name>hive.metastore.local</name>
 <value>false</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionURL</name>
 <value>jdbc:mysql://<hostname>/metastore_db</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionDriverName</name>
 <value>com.mysql.jdbc.Driver</value>
</property>

</configuration>
```

① http://db.apache.org/derby/。

表11-1解释了这些属性。再在文件jpox.properties中指定javax.jdo.option.ConnectionURL和javax.jdo.option.ConnectionDriverName属性。此外，在jpox.properties中还指定了登录到数据库的用户名和密码。jpox.properties文件应包含以下4行：

```
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionURL=jdbc:mysql://<hostname>/metastore_db
javax.jdo.option.ConnectionUserName=<username>
javax.jdo.option.ConnectionPassword=<password>
```

表11-1　在多用户模式下将MySQL数据库用作元数据存储的配置

属　性	描　述
hive.metastore.local	控制是否在客户端机器创建和使用一个本地metastore服务器。设为false，则使用远端的metastore服务器
javax.jdo.option.ConnectionURL	JDBC连接的URL，指定用于metastore的数据库[①]。例如jdbc:mysql://<hostname>/metastore_db
javax.jdo.option.ConnectionDriverName	JDBC驱动器的类名，例如com.mysql.jdbc.Driver
javax.jdo.option.ConnectionUserName	登录数据库的用户名
javax.jdo.option.ConnectionPassword	登录数据库的密码

一旦安装好了数据库，或者你仅为评估Hive而使用了它默认的单用户模式，都可以开始使用Hive的CLI。在$HIVE_HOME目录中键入/bin/hive，就会出现Hive的提示符，供你输入Hive命令。

```
bin/hive
Hive history file=/tmp/root/hive_job_log_root_200908240830_797162695.txt
hive>
```

11.1.2　查询的示例

在正式讲解HiveQL之前，先在命令行方式下运行几条命令是有好处的。可以感觉一下HiveQL是如何工作的，也可以自己随便探索一下。

假设你在本机上有专利引用数据cite75_99.txt，这是一个以逗号分隔的专利引用数据集。在Hive中，我们首先定义一个存储该数据的表：

```
hive> CREATE TABLE cite (citing INT, cited INT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.246 seconds
```

HiveQL语句以分号结尾。如上所示，一条语句可以跨多行，直到分号才结束。

这个4行命令的大部分动作在第一行。这里我们定义一个包含两个列的表，名为cite。第一列叫citing，类型为INT，而第二列叫cited，也是INT类型。命令的其他行告诉Hive该数据的存储方式（文本文件）和解析方式（以逗号分隔的字段）。

① MySQL JDBC驱动器的完整格式描述见http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html。

通过SHOW TABLES命令，我们可以看到Hive中当前的表：

```
hive> SHOW TABLES;
OK
cite
Time taken: 0.053 seconds
```

在Hive的“OK”和“Time taken”消息之间，可以看到有一个名为cite的表。还可以通过DESCRIBE命令检查它的schema：

```
hive> DESCRIBE cite;
OK
citing  int
cited   int
Time taken: 0.13 seconds
```

和预期的一样，表中包含我们定义的两个列。在HiveQL中对表的管理和定义类似于标准的关系数据库。下面将专利引用数据加载到这个表中。

```
hive> LOAD DATA LOCAL INPATH 'cite75_99.txt'
    > OVERWRITE INTO TABLE cite;
Copying data from file:/root/cite75_99.txt
Loading data to table cite
OK
Time taken: 9.51 seconds
```

这告诉Hive从本地文件系统上一个名为cite75_99.txt的文件中把数据加载到cite表中。在此过程中，本地计算机会把数据上传到HDFS，放在Hive管理的某些目录下。（除非你更改了配置，否则这些目录会位于/user/hive/warehouse下。）

当加载数据时，Hive不会让任何违反其schema的数据进入表中。Hive会将这些数据替换为空值。我们可以使用一个简单的SELECT语句来浏览cite表中的数据：

```
hive> SELECT * FROM cite LIMIT 10;
OK
NULL    NULL
3858241 956203
3858241 1324234
3858241 3398406
3858241 3557384
3858241 3634889
3858242 1515701
3858242 3319261
3858242 3668705
3858242 3707004
Time taken: 0.17 seconds
```

我们的schema将这两个列定义为整数。我们看到有一行为空值，表明有一条记录违反了schema。这是由于cite75_99.txt的第一行包含的是列名，而不是专利号码。总的来说，不是大问题。

既然我们已经确信Hive读到了数据并进行了管理，就可以对它进行各种查询。首先统计表的行数。SQL中有一个人们所熟知的实现，即SELECT COUNT(*)。这里HiveQL在语法上略有不同：

```
hive> SELECT COUNT(1) FROM cite;
Total MapReduce jobs = 1
Number of reduce tasks determined at compile time: 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 mapred.reduce.tasks=<number>
Starting Job = job_200908250716_0001, Tracking URL = http://ip-10-244-199-
143.ec2.internal:50030/jobdetails.jsp?jobid=job_200908250716_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=ip-10-
244-199-143.ec2.internal:9001 -kill job_200908250716_0001
 map = 0%,  reduce =0%
 map = 12%,  reduce =0%
 map = 25%,  reduce =0%
 map = 30%,  reduce =0%
 map = 34%,  reduce =0%
 map = 43%,  reduce =0%
 map = 53%,  reduce =0%
 map = 62%,  reduce =0%
 map = 71%,  reduce =0%
 map = 75%,  reduce =0%
 map = 79%,  reduce =0%
 map = 88%,  reduce =0%
 map = 97%,  reduce =0%
 map = 99%,  reduce =0%
 map = 100%,  reduce =0%
 map = 100%,  reduce =67%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0001
OK
16522439
Time taken: 85.153 seconds
```

通过这些消息，可以知道该查询生成了一个MapReduce作业。Hive之美在于用户根本不需要知道MapReduce的存在。用户所需关心的，仅仅是使用一种类似于SQL的语言来查询数据库。

上述查询的结果被直接输出在屏幕上。大多数情况下，查询结果应该被存盘，而且通常被放在其他的Hive表中。我们的下一个查询会查找每个专利的引用频率。首先生成一个表来存储它的结果：

```
hive> CREATE TABLE cite_count (cited INT, count INT);
OK
Time taken: 0.027 seconds
```

我们可以执行一个查询来得到引用频率。这个查询再次使用与SQL相类似的COUNT和GROUP BY功能。再加上INSERT OVERWRITE TABLE，就可以让Hive将结果写入表中：

```
hive> INSERT OVERWRITE TABLE cite_count
    > SELECT cited, COUNT(citing)
    > FROM cite
    > GROUP BY cited;
```

```
...
 map = 100%,  reduce =89%
 map = 100%,  reduce =90%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0002
Loading data to table cite_count
3258984 Rows loaded to cite_count
OK
Time taken: 103.331 seconds
```

查询执行告诉我们有3 258 984行被装载到引用频率表中。我们可以执行更多的HiveQL语句来浏览这个引用频率表：

```
hive> SELECT * FROM cite_count WHERE count > 10 LIMIT 10;
Total MapReduce jobs = 1
Number of reduce tasks is set to 0 as there's no reduce operator
...
 map = 80%,  reduce =0%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0003
OK
163404  13
164184  16
217584  13
246144  14
288134  11
347644  11
366494  11
443764  11
459844  13
490484  13
```

这个查询的一个有趣的地方是Hive足够智能，它知道“Number of reduce tasks is set to 0 as there's no reduce operator”（由于没有reduce运算符，reduce任务个数被置为0）。

当你已经使用完这个表时，可以使用DROP TABLE命令来删除它：

```
hive> DROP TABLE cite_count;
OK
Time taken: 0.024 seconds
```

使用这个命令时要小心。它不会让你确认是否真的想删除这个表。一旦你删除了这个表，就会很难恢复。

最终，你可以使用exit命令来退出Hive会话。

11.1.3 深入 HiveQL

在对Hive有了实际体验之后，我们现在就可以正式地深入观察HiveQL的不同方面及其用法了。

1. 数据模型

我们看到Hive将表作为基本的数据模型。物理上，Hive将表以目录形式放在/user/hive/warehouse下。例如，我们前面生成的cite表将数据放在/user/hive/warehouse/cite目录下。输出结果

的cite_count表则被放在/user/hive/warehouse/cite_count目录下。在大多数基本设置中，一个表下面的目录结构只有一层，表中的数据分散在一个目录下的多个文件中。

关系数据库在列上使用索引来加速对这些列的查询。Hive则使用了分区列（partition column）的概念，根据列的值将表划分为多个分区。例如，state列将表划分为50个分区，每个州有一个分区[①]。date列是用作日志数据的分区列，每天的数据归属于其自己的分区。Hive将分区列与常规的数据列区别看待，在分区列上执行查询要高效得多。这是因为Hive在物理上将不同的分区存在不同的目录中。例如，假设你有一个名为users的表，包含data和state两个分区列（加上常规的数据列）。Hive将为这个表生成如下的目录结构：

```
/user/hive/warehouse/users/date=20090901/state=CA
/user/hive/warehouse/users/date=20090901/state=NY
/user/hive/warehouse/users/date=20090901/state=TX
...
/user/hive/warehouse/users/date=20090902/state=CA
/user/hive/warehouse/users/date=20090902/state=NY
/user/hive/warehouse/users/date=20090902/state=TX
...
/user/hive/warehouse/users/date=20090903/state=CA
/user/hive/warehouse/users/date=20090903/state=NY
/user/hive/warehouse/users/date=20090903/state=TX
...
```

所有加利福尼亚州（state=CA）2009年9月1日（date=20090901）的用户数据放在一个目录中，而其他分区的数据放在其他目录中。如果进来一个查询，它请求加利福尼亚州在2009年9月1日的用户数据，Hive仅需处理那个目录下的数据，而不用管其他分区中存储的users表中的数据。对分区列的跨区查询涉及对多个目录的处理，但是Hive仍能避免去扫描users中的所有数据。某种程度上，分区为Hive带来的好处类似于索引对传统关系数据库的作用，但是分区在细粒度方面远远不及索引。你会想让每个分区的数据仍然足够大，使得对其运行MapReduce作业仍然有不错的效率。

除了分区，Hive数据模型还应用了桶（bucket）的概念，它可以提供对随机样本数据的高效查询。（例如，当计算一个列的平均值时，数据的随机样本可以提供一个很好的近似结果。）基于对桶列（bucket column）的散列，桶将分区的数据进一步划分为特定数目的文件。如果根据users表中的用户id，我们将桶的个数设定为32，那么Hive中的表将会有如下的完整文件结构：

```
/user/hive/warehouse/users/date=20090901/state=CA/part-00000
...
/user/hive/warehouse/users/date=20090901/state=CA/part-00031
/user/hive/warehouse/users/date=20090901/state=NY/part-00000
...
/user/hive/warehouse/users/date=20090901/state=NY/part-00031
/user/hive/warehouse/users/date=20090901/state=TX/part-00000
...
```

每个分区有32个桶。通过基于用户id的分桶，Hive会知道part-00000 ... part-00031中的

① 实际中你还必须处理哥伦比亚特区和各个准州。

每个文件都是一个用户的随机样本。许多汇总统计的计算在样本数据集上依然有相当好的精度。分桶对于加速一些查询特别有用。例如，Hive的查询可以用part-00000这段数据，它仅为一个分区上所有用户的1/32，而不用去读其他文件。不使用分桶，Hive也可以进行采样（基于列而不是桶列），但是这会涉及对所有数据的扫描，再随机地忽略大部分数据。这样，由采样所带来的大部分效率都因而丧失了。

2. 表的管理

我们已经看到如何为专利引用数据集生成一个样本表。现在让我们剖析一个更为复杂的表生成语句。这次生成一个名为page_view的表。

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
            page_url STRING, referrer_url STRING,
            ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY (dt STRING, country STRING)
CLUSTERED BY (userid) INTO 32 BUCKETS
ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      LINES TERMINATED BY '\n'
STORED AS SEQUENCEFILE;
```

第一部分看起来很像SQL中等价的语句：

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
            page_url STRING, referrer_url STRING,
            ip STRING COMMENT 'IP Address of the User')
```

它指定了表名（page_view）及其schema，包含列名和它们的类型。Hive支持如下的数据类型。

- TINYINT——单字节整数。
- SMALLINT——双字节整数。
- INT——四字节整数。
- BIGINT——八字节整数。
- DOUBLE——双精度浮点数。
- STRING——字符序列。

注意这里没有布尔类型，通常用TINYINT来替代。Hive还支持复杂的数据类型，如结构、映射和数组，它们可以嵌套。但是它们目前在语言中的支持还不够好，这涉及更高级的话题。

我们可以在每一列上附加一个描述性注释，就像这里对ip列所做的一样。而且，我们在表上也增加了一个描述性注释：

```
COMMENT 'This is the page view table'
```

CREATE TABLE语句的下一部分是指定分区列：

```
PARTITIONED BY (dt STRING, country STRING)
```

我们在前面讨论过，分区列面向查询进行了优化。它们不同于数据列viewTime、userid、page_url、referrer_url和ip。对于特定的行，分区列的值并没有显式地在行中存储，它隐含

在目录路径中。但是，分区列和数据列的查询在语法上并无差别。

```
CLUSTERED BY (userid) INTO 32 BUCKETS
```

CLUSTERED BY (...) INTO ... BUCKETS语句指定了分桶信息，其中包含参与随机采样的列，以及生成的桶数。桶数的选择依赖于以下标准：

(1) 每个分区下数据的大小；

(2) 打算使用的样本大小。

第一个标准很重要，因为在分区被进一步划分为指定个数的桶之后，你并不想让每个桶文件太小而导致Hadoop的效率很低。另一方面，桶应该和你所用样本有相同的大小或者比它更小。如果样本大小为用户基数的百分之三（1/32）左右，基于用户将分桶数设为32就是一个很好的选择。

注意　与分区不同，在数据被写入表时，Hive不会自动地强制分桶。指定分桶信息仅会告知Hive当数据写入一个表时，你会手动地强制分桶（取样）的标准，而Hive可以在处理查询时利用这一优势。为了强制设置分桶的标准，你需要在填充表时正确地设置reducer的个数。更多细节见http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL/BucketedTables。

ROW FORMAT子句告诉Hive表中数据按行存储的方式。若无此子句，Hive默认以换行符作为行分隔符，以001（Ctrl+A）的ASCII值作为字段分隔符。我们的子句告诉Hive改为使用制表符字符作为字段分隔符。我们还告诉Hive使用换行符作为行分隔符，但那已经是默认值，我们在这里包括它只是为了演示而已：

```
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '\t'
        LINES TERMINATED BY '\n'
```

最终，最后一个子句告诉Hive用于存储表中数据的文件格式：

```
STORED AS SEQUENCEFILE;
```

目前Hive支持两种格式，SEQUENCEFILE和TEXTFILE。序列文件是一种压缩的格式，通常可提供更高的性能。

我们可以在CREATE TABLE语句中添加一个EXTERNAL修饰符，这样表被创建为指向一个现有的数据目录。你需要指定此目录的位置。

```
CREATE EXTERNAL TABLE page_view(viewTime INT, userid BIGINT,
                        page_url STRING, referrer_url STRING, ip STRING)
LOCATION '/path/to/existing/table/in/HDFS';
```

在生成一个表之后，你可以用DESCRIBE命令在Hive中查询这个表的schema：

```
hive> DESCRIBE page_view;
```

你还可以用ALTER命令更改表的结构，包括更改表的名称：

```
hive> ALTER TABLE page_view RENAME TO pv;
```

或增加新的列：

```
hive> ALTER TABLE page_view ADD COLUMNS (newcol STRING);
```

或删除一个分区：

```
hive> ALTER TABLE page_view DROP PARTITION (dt='2009-09-01');
```

要删除整个表，使用DROP TABLE命令：

```
hive> DROP TABLE page_view;
```

要知道Hive正在管理着哪些表，你可以用SHOW命令把它们都显示出来：

```
hive> SHOW TABLES;
```

如果正在使用的表有很多，将难以把它们都列出来，你可以用Java正则表达式缩小其结果：

```
hive> SHOW TABLES 'page_.*';
```

3. 装载数据

将数据装载到一个Hive表中的方法有很多。主要是LOAD DATA命令：

```
hive> LOAD DATA LOCAL INPATH 'page_view.txt'
    > OVERWRITE INTO TABLE page_view;
```

它取一个名为page_view.txt的本地文件，并将其内容加载到page_view表。如果我们忽略了OVERWRITE修饰符，内容会被添加到表中，而不是替换已存在的所有内容。如果我们忽略LOCAL修饰符，文件会取自HDFS而不是本地文件系统。此外，还可以通过LOAD DATA命令在表中指定一个分区来加载数据：

```
hive> LOAD DATA LOCAL INPATH 'page_view.txt'
    > OVERWRITE INTO TABLE page_view
    > PARTITION (dt='2009-09-01', country='US');
```

当操作来自本地文件系统中的数据时，可以执行Hive CLI中的本地Unix命令，这很有用。需要在命令前加上感叹号（!），命令尾部加上分号（;）。例如，可以获取一个文件的列表

```
hive> ! ls ;
```

或检查文件的前几行

```
hive> ! head hive_result ;
```

请注意,! 和 ; 周围的空格不是必要的。我们添加它们是为了提高可读性。

4. 执行查询

大多数情况下，运行HiveQL查询与SQL查询惊人地相似。一个通常的差别为HiveQL查询的结果相对较大。几乎总是有一个INSERT子句，以便告诉Hive把查询结果存起来。往往存到其他的表中：

```
INSERT OVERWRITE TABLE query_result
```

也会是HDFS的一个目录：

```
INSERT OVERWRITE DIRECTORY '/hdfs_dir/query_result'
```

有时还会是一个本地目录：

```
INSERT OVERWRITE LOCAL DIRECTORY '/local_dir/query_result'
```

可以看到基本查询几乎与SQL完全相同：

```
INSERT OVERWRITE TABLE query_result
SELECT *
FROM page_view
WHERE country='US';
```

请注意查询作用于分区列（country）上，但如果是数据列，查询看起来也完全一样。只有一个语法的调整，即HiveQL使用了COUNT(1)以替代SQL通常在此处使用的COUNT(*)。例如，将使用如下的HiveQL查询查找来自美国的页面浏览数量：

```
SELECT COUNT(1)
FROM page_view
WHERE country='US';
```

像SQL一样，GROUP BY子句允许在组上执行聚合查询。此查询将列出每个国家的页面浏览数量：

```
SELECT country, COUNT(1)
FROM page_view
GROUP BY country;
```


而这个查询将列出每个国家的唯一用户数：

```
SELECT country, COUNT(DISTINCT userid)
FROM page_view
GROUP BY country;
```

表11-2显示在HiveQL中支持的所有运算符。这些运算符在SQL和编程语言中都是很标准的，我们不会详细解释它们。主要的例外是正则表达式的匹配。HiveQL提供了两个正则表达式匹配的命令——LIKE和REGEXP。（RLIKE等同于REGEXP。）LIKE仅执行简单的SQL正则表达式匹配，如B中的下划线（_）字符匹配A中任意单个字符，而百分号（%）字符匹配A中任意的数字符号。REGEXP将B视为一个完整的正则表达式[①]。表11-3和表11-4列出了主要的HiveQL函数。

表11-2 HiveQL中的标准运算符

<table>
<tr><th>运算符类型</th><th>运 算 符</th></tr>
<tr><td>比较运算符</td><td>A = B , A <> B , A < B , A <= B , A > B , A >= B , A IS NULL , A IS NOT NULL , A LIKE B , NOT A LIKE B , A RLIKE B , A REGEXP B</td></tr>
<tr><td>算术运算符</td><td>A + B , A - B , A * B , A / B , A % B</td></tr>
<tr><td>按位运算符</td><td>A & B , A | B , A ^ B, ~A</td></tr>
<tr><td>逻辑运算符</td><td>A AND B, A && B, A OR B, A || B, NOT A, !A</td></tr>
</table>

① …/regex/Pattern.html。

表11-3 内置函数

函 数	描 述
concat(string *a*, string *b*)	返回字符串*a*与字符串*b*的串接结果
substr(string *str*, int *start*) substr(string *str*, int start, int *length*)	返回从*start*开始的*str*的子字符串。结果直到字符串尾部，除非特别设定了*length*参数
round(double *num*)	返回最近的整数（BIGINT）
floor(double *num*)	返回等于或小于*num*的最大整数（BIGINT）
ceil(double *num*) ceiling(double *num*)	返回等于或大于*num*的最小整数（BIGINT）
sqrt(double *num*)	返回*num*的平方根
rand() rand(int *seed*)	返回一个随机数（每行不同）。指定选项*seed*的值形成确定的随机数序列
Ln(double *num*)	返回*num*的自然对数
log2(double *num*)	返回*num*以2为底的对数
log10(double *num*)	返回*num*以10为底的对数
log(double *num*) log(double *base*, double *num*)	返回*num*的自然对数，或者返回以*base*为底的对数
exp(double *a*)	*a*的*e*次幂（自然对数的底）
power(double *a*, double *b*) pow(double *a*, double *b*)	返回*a*的*b*次幂
upper(string *s*) ucase(string *s*)	返回大写的字符串*s*
lower(string *s*) lcase(string *s*)	返回小写的字符串*s*
trim(string *s*)	返回字符串*s*两端去掉空格的结果
ltrim(string *s*)	返回字符串*s*左端去掉空格的结果
rtrim(string *s*)	返回字符串*s*右端去掉空格的结果
regexp(string *s*, string *regex*)	返回字符串*s*是否与Java正则表达式*regex*相匹配
regexp_replace(string *s*, string *regex*, string *replacement*)	返回一个字符串，其中所有与Java的正则表达式*regex*匹配的都被替换为*replacement*
day(string *date*) dayofmonth(string *date*)	返回日期或时间戳字符串中天的部分
month(string *date*)	返回日期或时间戳字符串中月的部分
year(string *date*)	返回日期或时间戳字符串中年的部分
To_date(string *timestamp*)	返回一个时间戳字符串中日期的部分（年-月-日）
unix_timestamp(string *timestamp*)	将时间戳字符串转换为Unix时间
from_unixtime(int *unixtime*)	将整型的Unix时间转换为时间戳字符串
date_add(string *date*, int *days*)	在*date*字符串中增加天数
date_sub(string *date*, int *days*)	从*date*字符串中减去天数
datediff(string *date1*, string *date2*)	计算天数的不同，如果*date1*更早，则结果为负。

表11-4　内置聚合函数

函　数	描　述
count(1) count(DISTINCT *col*)	返回一个组中的成员个数，或者该列中不重复值的数量
sum(*col*) sum(DISTINCT *col*)	返回列上值的总和，或者该列中不重复值的和
avg(*col*) avg(DISTINCT *col*)	返回列的平均值，或者该列中不重复值的平均值
max(*col*)	返回该列中的最大值
min(*col*)	返回该列中的最小值

对于用户而言，寻求Pig Latin和HiveQL这种更高级的语言的一个主要动力在于支持联结操作。目前HiveQL只支持等联结。连接查询的示例如下：

```
INSERT OVERWRITE TABLE query_result
SELECT pv.*, u.gender, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.id);
```

在语法上，将关键字JOIN添加到FROM子句中两个表之间，然后在关键字ON后面指定联结的列。若要联结两个以上的表，我们重复这种模式：

```
INSERT OVERWRITE TABLE pv_friends
SELECT pv.*, u.gender, u.age, f.friends
FROM page_view pv JOIN user u ON (pv.userid = u.id)
                  JOIN friend_list f ON (u.id = f.uid);
```

我们可以通过修改FROM子句为任何查询添加采样。该查询会去计算平均浏览时间，除非这个平均值仅取自32个桶中第一桶的数据：

```
SELECT avg(viewTime)
FROM page_view TABLESAMPLE(BUCKET 1 OUT OF 32);
```

TABLESAMPLE的一般语法如下：

```
TABLESAMPLE(BUCKET x OUT OF y)
```

查询样本的大小为1/y左右。此外，y为表创建时所指定的桶数的倍数或因数。例如，如果我们将y改为16，查询为

```
SELECT avg(viewTime)
FROM page_view TABLESAMPLE(BUCKET 1 OUT OF 16);
```

那么样本的大小为大约每16个用户中取一个（桶列为userid）。表仍有32个桶，但Hive仅取1和17并一起处理以满足该查询。另一方面，如果设y为64，则Hive会对单个桶中的一半数据执行查询。x的值仅用于选择要使用的桶。在真正随机的抽样下，它的取值不会产生什么影响。

除了avg，Hive还有很多其他的内置函数。你可以在表11-3和表11-4中看到一些较为常见的函数。程序员还可以在Hive中添加UDF来定制处理函数。如何创建UDF的简介见http://wiki.apache.org/hadoop/Hive/AdminManual/Plugins。

11.1.4 Hive小结

Hive是建立在Hadoop大规模可扩展系统结构之上的数据仓库层。通过把重点放在结构化数据上，Hive添加了许多性能强化技术（如分区）以及可用性特征（如类SQL语言）。它使某些常见的任务（如联结）变得容易。Hive将Hadoop技术推广给更多的数据分析师和其他非编程人员。至2009年8月，Facebook统计其雇员中有29%是Hive的用户，其中一半以上为非工程人员[①]。

11.2 其他Hadoop相关的部分

Hadoop生态系统每天都在成长。下面为与Hadoop相关的项目或第三方软件，它们很实用或是有巨大的潜力。除了Aster Data和Greenplum之外，它们在某种程度上都是开源的。

11.2.1 HBase

HBase（http://hadoop.apache.org/hbase/）是一个可扩展的数据存储系统，旨在用于随机读写访问（相当于）结构化的数据。它的设计源自谷歌的Bigtable[②]，旨在支持大表，即包含数十亿计的行和数以百万计的列。它使用HDFS作为底层文件系统，以达到完全分布式和高可用性。版本0.20在性能上有了显著的提升。

11.2.2 ZooKeeper

ZooKeeper（http://hadoop.apache.org/zookeeper/）是用于构建大型分布式应用的一种协作式服务。你可以在Hadoop Core框架之外独立使用它。它实现了许多在大型分布式应用中常见的服务，如配置管理、命名、同步和组服务。从历史上看，开发人员不得不为每个分布式应用重塑这些服务，这样做不仅耗时，而且容易出错，因为这些服务难以正确实现。通过将底层的复杂性剥离出来，ZooKeeper使应用能够很方便地实现consensus（会商）、leader election（领导者选举）、presense protocol（存在协议）和其他原语。这解放了开发人员，使其可以专注于应用程序的语义。ZooKeeper常作为其他Hadoop相关项目的主要组件，如HBase和Katta。

11.2.3 Cascading

Cascading（http://www.cascading.org/）是Hadoop上用于组装和执行复杂数据处理工作流的一个API。它从MapReduce模型中抽象出包括元组（tuple）、管道（pipe）和出水/入水口（tap）的数据处理模型。管道对元组的流进行操作，其操作包括Each、Every、GroupBy和CoGroup。管道还可以组装和嵌套，以创建“程序集”（assembly）。当我们将管道组件接上（数据）入水口和（数

11

① 这件事引自http://www.facebook.com/note.php?note_id=114588058858。在http://www.facebook.com/note.php?note_id=16121578919中解释了Facebook决定如何构建其Hadoop基础设施。在http://www.slideshare.net/zshao/hive-data-warehousing-analytics-on-hadoop-presentation中详细介绍了Facebook如何围绕Hive来设计其数据仓库及分析系统。

② “Bigtable: A Distributed Storage System for Structured Data” by Chang et al., OSDI '06—Seventh Symposium on Operating System Design and Implementation, http://labs.google.com/papers/bigtable.html。

据）出水口时，就生成了一个可以执行的“流”（flow）。

Cascading与Pig之间有许多相似的设计与目标。不过，一个区别在于Pig的Grunt便于执行临时查询。还有一个区别是Pig程序都用Pig Latin编写，而Cascading更像一个Java框架，你要通过实例化各种Java类（Each和Every等）来创建数据的处理流程。使用Cascading不需要学习新的语言，而且创建的数据处理流程效率更高，因为它是直接由你自己写的。

11.2.4　Cloudera

Cloudera（http://www.cloudera.com/）试图在Hadoop中充当RedHat在Linux中的角色。它支持和封装Hadoop，以便对企业用户易用、友好。它在主要城市开设了实时培训课程，并在他们的网站上提供了教育视频。通过使用他们以RPM或Ubuntu/Debian包的形式提供的免费Hadoop发行版，可以简化Hadoop的部署。他们的Hadoop发行版基于Hadoop最新的稳定版本、未来发行版中有用（且已测试）的补丁，以及Pig和Hive等额外的工具。Cloudera还提供咨询和支持服务，帮助企业使用Hadoop。

11.2.5　Katta

其网址为http://katta.sourceforge.net/。因为Hadoop起源可以追溯到搜索引擎，它自然可以被应用到分布式索引和查询中。Nutch是以Hadoop为基础构建的Web搜索引擎[①]。但作为一个Web搜索引擎，Nutch有许多特别的限定。这使它往往无法成为与特定搜索应用相匹配的解决方案。

Katta是一个可扩展、容错、分布式的索引系统。它比Nutch更轻量、灵活。在某种意义上它为Lucene添加了一些额外的功能（如复制、冗余、容错和可扩展性），同时保留了基本应用程序的语义。

11.2.6　CloudBase

CloudBase（http://cloudbase.sourceforge.net/）是一个架构在Hadoop之上的ANSI SQL数据仓库层。与Hive不同，CloudBase直接工作在平面文件上，没有任何元数据存储区。它更严格地与ANSI SQL保持一致，而交互主要是通过JDBC驱动器，这使其更容易连接到商业智能报告工具。大多数情况下，CloudBase充当了编译器的角色，它获取SQL查询并将它们编译为MapReduce程序。写这篇文章时，相比于Pig和Hive而言，CloudBase开发者社区不够活跃，而其GPL许可证比Apache许可的限制性更强。

11.2.7　Aster Data和Greenplum

Aster Data Systems（http://www.asterdata.com/）和Greenplum（http://www.greenplum.com/）这两个商业公司提供了高性能、可扩展的数据仓库解决方案，将SQL与MapReduce紧密结合在一起。虽然它们支持MapReduce编程模型，但均独立于Hadoop之外，且采取了许多不同的底层设计

① 也许更准确的说法是Nutch激励了Hadoop的产生。更完整的故事见第1章。

方案。与Hadoop不同的是，它们的产品采用了特别针对企业用户的设计架构，以实现更高性能的SQL数据仓库。因为它们在MapReduce范式的实现上采用了与Hadoop不同的角度，学习它们有助于理解Hadoop在架构上所做的一些权衡。

11.2.8 Hama 和 Mahout

Hama（http://incubator.apache.org/hama/）和Mahout（http://lucene.apache.org/mahout/）都是使用Hadoop进行科学数据处理的项目。Hama是矩阵计算软件包，用于计算乘积、逆、特征值、特征向量和其他的矩阵运算。Mahout更专门针对基于Hadoop实现机器学习算法（更多的信息请参阅Manning出版社的*Mahout in Action*）。Mahout 0.1版于2009年4月发布，包含诸如Naïve Bayes分类、k-means聚类和协同过滤等算法的实现。

在写本书时，这两个项目都相对较新，位于Apache incubator（孵化器）中。感兴趣的读者可以考虑成为这些项目的参与者。

11.2.9 search-hadoop.com

作为一名Hadoop程序员，你经常需要找一些有关Hadoop或其子项目的文档。Sematext，一家专业从事搜索和分析的公司，运行了网站http://search-hadoop.com/，可以供你搜索所有Hadoop的子项目和数据源——邮寄列表档案、Wiki、问题跟踪系统、源代码等。基本的查询索引仍在不断更新。搜索结果允许按照项目、数据源和作者进行筛选，并可以按日期、关联或二者的组合进行排序。

11.3 小结

本章介绍了许多附加的Hadoop工具。我们特别关注了Hive，它是一个数据仓库软件包，允许你使用类SQL语言来处理Hadoop的数据。围绕着Hadoop的支撑性软件如雨后春笋般涌现，丰富了这个生态系统。在下一章中，你会在实际案例中看到它们的作用。