Assgnment 73

HiveQL: Data Definition:

HiveQL is the Hive query language. Like all SQL dialects in widespread use, it doesn’t fully conform to any particular revision of the ANSI SQL standard. It is perhaps closest to MySQL’s dialect, but with significant differences. Hive offers no support for row-level inserts, updates, and deletes. Hive doesn’t support transactions. Hive adds extensions to provide better performance in the context of Hadoop and to integrate with custom extensions and even external programs.

Still, much of HiveQL will be familiar. This chapter and the ones that follow discuss the features of HiveQL using representative examples. In some cases, we will briefly mention details for completeness, then explore them more fully in later chapters.

This chapter starts with the so-called data definition language parts of HiveQL, which are used for creating, altering, and dropping databases, tables, views, functions, and indexes. We’ll discuss databases and tables in this chapter, deferring the discussion of views until Chapter 7, indexes until Chapter 8, and functions until Chapter 13.

We’ll also discuss the SHOW and DESCRIBE commands for listing and describing items as we go.

Subsequent chapters explore the data manipulation language parts of HiveQL that are used to put data into Hive tables and to extract data to the filesystem, and how to explore and manipulate data with queries, grouping, filtering, joining, etc.

Databases in Hive

The Hive concept of a database is essentially just a catalog or namespace of tables. However, they are very useful for larger clusters with multiple teams and users, as a way of avoiding table name collisions. It’s also common to use databases to organize production tables into logical groups.

If you don’t specify a database, the default database is used.

The simplest syntax for creating a database is shown in the following example:

hive> CREATE DATABASE financials;

Hive will throw an error if financials already exists. You can suppress these warnings with this variation:

hive> CREATE DATABASE IF NOT EXISTS financials;

While normally you might like to be warned if a database of the same name already exists, the IF NOT EXISTS clause is useful for scripts that should create a database on-the-fly, if necessary, before proceeding.

You can also use the keyword SCHEMA instead of DATABASE in all the database-related commands.

At any time, you can see the databases that already exist as follows:

hive> SHOW DATABASES;

default

financials

hive> CREATE DATABASE human\_resources;

hive> SHOW DATABASES;

default

financials

human\_resources

If you have a lot of databases, you can restrict the ones listed using a regular expression, a concept we’ll explain in LIKE and RLIKE, if it is new to you. The following example lists only those databases that start with the letter h and end with any other characters (the .\* part):

hive> SHOW DATABASES LIKE 'h.\*';

human\_resources

hive> ...

Hive will create a directory for each database. Tables in that database will be stored in subdirectories of the database directory. The exception is tables in the default database, which doesn’t have its own directory.

The database directory is created under a top-level directory specified by the property hive.metastore.warehouse.dir, which we discussed in Local Mode Configuration and Distributed and Pseudodistributed Mode Configuration. Assuming you are using the default value for this property, /user/hive/warehouse, when the financials database is created, Hive will create the directory /user/hive/warehouse/financials.db. Note the .db extension.

You can override this default location for the new directory as shown in this example:

hive> CREATE DATABASE financials

> LOCATION '/my/preferred/directory';

You can add a descriptive comment to the database, which will be shown by the DESCRIBE DATABASE <database> command.

hive> CREATE DATABASE financials

> COMMENT 'Holds all financial tables';

hive> DESCRIBE DATABASE financials;

financials Holds all financial tables

hdfs://master-server/user/hive/warehouse/financials.db

Note that DESCRIBE DATABASE also shows the directory location for the database. In this example, the URI scheme is hdfs. For a MapR installation, it would be maprfs. For an Amazon Elastic MapReduce (EMR) cluster, it would also be hdfs, but you could set hive.metastore.warehouse.dir to use Amazon S3 explicitly (i.e., by specifying s3n://bucketname/… as the property value). You could use s3 as the scheme, but the newer s3n is preferred.

In the output of DESCRIBE DATABASE, we’re showing master-server to indicate the URI authority, in this case a DNS name and optional port number (i.e., server:port) for the “master node” of the filesystem (i.e., where the NameNode service is running for HDFS). If you are running in pseudo-distributed mode, then the master server will be localhost. For local mode, the path will be a local path, file:///user/hive/warehouse/financials.db.

If the authority is omitted, Hive uses the master-server name and port defined by the property fs.default.name in the Hadoop configuration files, found in the $HADOOP\_HOME/conf directory.

To be clear, hdfs:///user/hive/warehouse/financials.db is equivalent to hdfs://master-server/user/hive/warehouse/financials.db, where master-server is your master node’s DNS name and optional port.

For completeness, when you specify a relative path (e.g., some/relative/path), Hive will put this under your home directory in the distributed filesystem (e.g., hdfs:///user/<user-name>) for HDFS. However, if you are running in local mode, your current working directory is used as the parent of some/relative/path.

For script portability, it’s typical to omit the authority, only specifying it when referring to another distributed filesystem instance (including S3 buckets).

Lastly, you can associate key-value properties with the database, although their only function currently is to provide a way of adding information to the output of DESCRIBE DATABASE EXTENDED <database>:

hive> CREATE DATABASE financials

> WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date' = '2012-01-02');

hive> DESCRIBE DATABASE financials;

financials hdfs://master-server/user/hive/warehouse/financials.db

hive> DESCRIBE DATABASE EXTENDED financials;

financials hdfs://master-server/user/hive/warehouse/financials.db

{date=2012-01-02, creator=Mark Moneybags);

The USE command sets a database as your working database, analogous to changing working directories in a filesystem:

hive> USE financials;

Now, commands such as SHOW TABLES; will list the tables in this database.

Unfortunately, there is no command to show you which database is your current working database! Fortunately, it’s always safe to repeat the USE … command; there is no concept in Hive of nesting of databases.

Recall that we pointed out a useful trick in Variables and Properties for setting a property to print the current database as part of the prompt (Hive v0.8.0 and later):

hive> set hive.cli.print.current.db=true;

hive (financials)> USE default;

hive (default)> set hive.cli.print.current.db=false;

hive> ...

Finally, you can drop a database:

hive> DROP DATABASE IF EXISTS financials;

The IF EXISTS is optional and suppresses warnings if financials doesn’t exist.

By default, Hive won’t permit you to drop a database if it contains tables. You can either drop the tables first or append the CASCADE keyword to the command, which will cause the Hive to drop the tables in the database first:

hive> DROP DATABASE IF EXISTS financials CASCADE;

Using the RESTRICT keyword instead of CASCADE is equivalent to the default behavior, where existing tables must be dropped before dropping the database.

When a database is dropped, its directory is also deleted.

Alter Database

You can set key-value pairs in the DBPROPERTIES associated with a database using the ALTER DATABASE command. No other metadata about the database can be changed, including its name and directory location:

hive> ALTER DATABASE financials SET DBPROPERTIES ('edited-by' = 'Joe Dba');

There is no way to delete or “unset” a DBPROPERTY.

HiveQL: Data Manipulation:

Loading Data into Managed Tables

Since Hive has no row-level insert, update, and delete operations, the only way to put data into an table is to use one of the “bulk” load operations. Or you can just write files in the correct directories by other means.

We saw an example of how to load data into a managed table in Partitioned, Managed Tables, which we repeat here with an addition, the use of the OVERWRITE keyword:

LOAD DATA LOCAL INPATH '${env:HOME}/california-employees'

OVERWRITE INTO TABLE employees

PARTITION (country = 'US', state = 'CA');

This command will first create the directory for the partition, if it doesn’t already exist, then copy the data to it.