## Transcript name: Working with Hive

| English |
|---|
| Let us examine Hive. As mentioned in the overview, Hive is a technology for turning Hadoop into a data warehouse, complete with an SQL dialect for querying it. We will begin our look at Hive with its configuration. You can configure Hive using any one of three methods. You can edit a file called hive-site.xml. You can use this file to specify the location of your HDFS NameNode and your MapReduce JobTracker. You can also use it for specifying configuration settings for the metastore, a topic we will come to later. These same options can be specified when starting the Hive command shell by specifying a -hiveconf option. Finally, within the Hive shell, you can change any of these settings using the set command.<br><br>There are three ways to run Hive. You can run it interactively by launching the hive shell using the hive command with no arguments. You can run a Hive script by passing the -f option to the hive command along with the path to your script file. Finally, you can execute a Hive program as one command by passing the -e option to the hive command followed by your Hive program in quotes.<br><br>Hive also supports launching services from the hive command. You can launch a service that lets you access Hive through Thrift, ODBC, or JDBC by passing --service to the hive command followed by the word hiveserver. There is also a web interface to hive whose service is launched by following the --service option with hwi. You can also use a Hive service to run the hadoop command with the jar option the same as you could do directly, but with Hive jars on the classpath. Lastly, there is a service for an out of process metastore.<br><br>The metastore stores the Hive metadata. There are three configurations you can choose for your metastore. First is embedded, which runs the metastore code in the same process with your Hive program and the database that backs the metastore is in the same process as well. The second option is to run it as local, which keeps the metastore code running in process, but moves the database into a separate process that the metastore code communicates with. The third option is to move the metastore itself out of process as well. This can be useful if you wish to share a metastore with other users.<br><br>One of the fundamental aspects of Hive's design is Schema-On-Read. In most SQL databases, the schema is applied when data is loaded into the database. In other words, most SQL databases are schema-on-write. Data coming in to the database must adhere to the schema. Hive's approach is to defer the application of a schema until you attempt to read the data. This has both benefits and drawbacks relative to schema-on-write. The benefits are that loads are faster and you have the flexibility of using multiple schemas for the same data. The downside is slower queries.<br><br>Hive Query Language or HiveQL is an SQL dialect. It does not support the fully |

SQL92 specification modeling itself more on MySQL than SQL92. You can INSERT OVERWRITE TABLE but cannot UPDATE or DELETE. There are no transactions and no indexes. There is no support for the HAVING clause in a SELECT statement, nor for correlated subqueries, subqueries outside FROM clauses, updateable or materialized views or stored procedures.

HiveQL has many of the same extensions over SQL92 as MySQL does. It also has extensions designed to address the MapReduce features available in Hadoop such as a MAP clause and a REDUCE clause. Like Pig, Hive has both simple types like INT and STRING and complex types ARRAY, MAP and STRUCT.

Hive has many built-in functions though not as many as SQL. You can view them by typing SHOW FUNCTIONS from the Hive shell. You can find information on an individual function by typing DESCRIBE FUNCTION followed by the function's name in the Hive shell.

Like other SQL databases, Hive works in terms of tables. There are two kinds of tables you can create: managed tables whose data is managed by Hive and external tables whose data is managed outside of Hive. When you load a file into a managed table, Hive moves the file into its data warehouse. When you drop such a table, both the data and metadata are deleted. When you load a file into an external table, no files are moved. Dropping the table only deletes the metadata. The data is left alone. External tables are useful for sharing data between Hive and other Hadoop applications or when you wish to use more than one schema on the same data.

Hive offers a way to speed up queries of subsets of your data. You can partition your data based on the value of a column. When creating a table, you can specify a PARTITION BY clause to specify the column used to partition the data. Then, when loading the data, you specify a PARTITION clause to say what partition you are loading. You can then query individual partitions more efficiently than you could unpartitioned data. The SHOW PARTITIONS command will let you see a table's partitions.

Another option Hive provides for speeding up queries is bucketing. Like partitioning, bucketing splits up the data by a particular column, but in bucketing you do not specify the individual values for that column that correspond to buckets, you simply say how many buckets to split the table into and let Hive figure out how to do it. The benefit of bucketing is that imposes extra structure on your table that can be used to speed up certain queries, such as joins on bucketed columns. It also improves performance when sampling your data. You specify the column to bucket on and the number of buckets using the CLUSTERED BY clause. If you wanted the bucket to be sorted as well, you use the SORTED BY clause. If you wish to query a sample of your data rather than the whole data set, you can use the TABLESAMPLE command and it will take advantage of bucketing.

Hive has multiple ways of reading and storing your data on disk. There is a delimited text format and two binary formats. A serializer/deserializer or SerDe is used for translating to and from the storage format. Among the binary SerDes are a SerDe for dealing with row-oriented binary data in a SequenceFile and one for dealing with column oriented binary data in an RCFile.

Like PigLatin, HiveQL lets you extend it by writing user-defined functions in Java, registering them into HiveQL, and optionally aliasing them to shorter names. There are three types of HiveQL UDFs: an ordinary UDF that whose input and output are both a single row, an aggregate UDF or UDAF that transforms multiple rows into a single row, and a table UDF or UDTF that transforms a single row into multiple rows. You register the jar file containing your UDF with the ADD JAR command and alias it with the CREATE TEMPORARY FUNCTION command.

Now let us take a look at Hive in action.

BigInsights is configured to use a metastore separate from the hive shell so we start the metastore service using the start.sh script. The web service gets started as well.

As in the Pig demonstration, we will use the U.S. foreign aid dataset.

Let us start the hive shell with the hive command.

First, we create a table to store the foreign aid data.

We give it a schema by specifying country for the first column and sum for the second.

The file is comma delimited, so we specify the row format as delimited and we state that fields are terminated by a comma.

The data is stored on disk as a textfile.

We now see our table when we run SHOW TABLES

We can see its schema using DESCRIBE.

Let us load data from HDFS into our newly created table

Now if we write a SELECT statement with a LIMIT we quickly get back a picture of some of our data.

Now let us compute the sum for each country as we did in Pig. We simply use the standard SQL GROUP BY clause.

Maps and reduces are run and here is the result.

This lesson is continued in the next video.