# APPLYING STRUCTURE TO HADOOP DATA WITH HIVE 5

## IMPORTANT QUESTIONS

### PART-A

**Q.1** ***What is Hive?***

**Ans. Hive :** Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Initially Hive was developed by Facebook. later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

**Q.2** ***What is the difference between an external table and a managed table in Hive?***

**Ans. Difference between an external table and a managed table**

| External Table | Managed Table |
|---|---|
| External tables in Hive refer to the data that is at an existing location outside the warehouse directory | Also known as the internal table, these types of tables manage the data and move it into its warehouse directory by default |
| Hive deletes the metadata information of a table and does not change the table data present in HDFS | If one drops a managed table, the metadata information along with the table data is deleted from the Hive warehouse directory |

**Q.3** ***What are the features of hive?***

**Ans. Features of Hive**

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable. and extensible.

**Q.4** ***What is a partition in Hive?***

**Ans. Partition in Hive :** Partition is a process for grouping similar types of data together based on columns or partition keys. Each table can have one or more partition keys to identify a particular partition.

**Q.5** ***Why is partioning required in Hive?***

**Ans.** Partitioning provides granularity in a Hive table. It reduces the query latency by scanning only relevant partitioned data instead of the entire data set. We can partition the transaction data for a bank based on month – January, February, etc. Any operation regarding a particular month, say February, will only have to scan the February partition, rather than the entire table data.

**Q.6** ***Why does Hive not store metadata information in HDFS?***

**Ans.** We know that the Hive's data is stored in HDFS. However, the metadata is either stored locally or it is stored

in RDBMS. The metadata is not stored in HDFS, because HDFS read/write operations are time-consuming. As such, Hive stores metadata information in the metastore using RDBMS instead of HDFS. This allows us to achieve low latency and is faster.

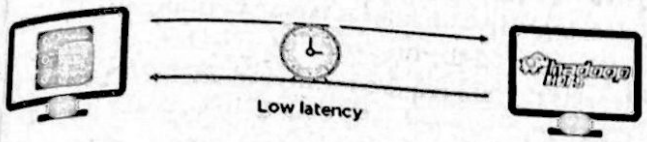


*Fig.*

**Q.7** ***What are the components used in Hive query processors?***

**Ans.** The components used in Hive query processors are:

- Parser
- Semantic Analyzer
- Execution Engine
- User-Defined Functions
- Logical Plan Generation
- Physical Plan Generation
- Optimizer
- Operators
- Type checking

## PART-B

**Q.8** ***Give a introduction to hive (hello to Hive).***

**Ans.** Hive provides Hadoop with a bridge to the RDBMS world and provides an SQL dialect known as Hive Query Language (HiveQL), which can be used to perform SQL-like tasks. That's the big news, but there's more to Hive than meets the eye, as they say, or more applications of this new technology than you can present in a standard elevator pitch. For example, Hive also makes possible the concept known as enterprise data warehouse (EDW) augmentation, a leading use case for Apache Hadoop, where data warehouses are set up as RDBMSs built specifically for data analysis and reporting. Now, some experts will argue that Hadoop (with Hive, HBase, Sqoop, and its assorted buddies) can replace the EDW, but we disagree. We believe that Apache Hadoop is a great addition to the enterprise and that it can augment and complement existing EDWs.

Closely associated with RDBMS/EDW technology is extract, transform, and load (ETL) technology. To grasp what ETL does, it helps to know that, in many use cases, data cannot be immediately loaded into the relational database – it must first be extracted from its native source, transformed into an appropriate format, and then loaded into the RDBMS or EDW. For example, a company or an organization might extract unstructured text data from an Internet forum, transform the data into a structured format that's both valuable and useful, and then load the structured data into its EDW.

Hive is a powerful ETL tool in its own right, along with the major player in this realm: Apache Pig. Again, users may try to set up Hive and Pig as the new ETL tools for the data center. (Let them try.) As with the debate over Last but not least, Apache Hive gives you powerful analytical tools, all within the framework of HiveQL. These tools should look and feel quite familiar to IT professionals who understand how to use SQL.

**Q.9** ***What are the key differences between Hive and Pig?***

**Ans.**

| Hive | Pig |
|---|---|
| It uses a declarative language, called HiveQL, which is similar to SQL for reporting. | Uses a high-level procedural language called Pig Latin for programming |
| Operates on the server-side of the cluster and allows structured data. | Operates on the Client side of the cluster and allows both structured and unstructured data |
| It does not support the Avro file format by default. This can be done using "Org.Apache.Hadoop. Hive.serde2.Avro" | Supports Avro file format by default. |
| Facebook developed it and it supports partition | Yahoo developed it and it does not support partition |

BDA.72

**Q.10** ***Explain the architecture of HIVE with neat sketch.***

**Ans.** The following component diagram depicts the architecture of Hive:

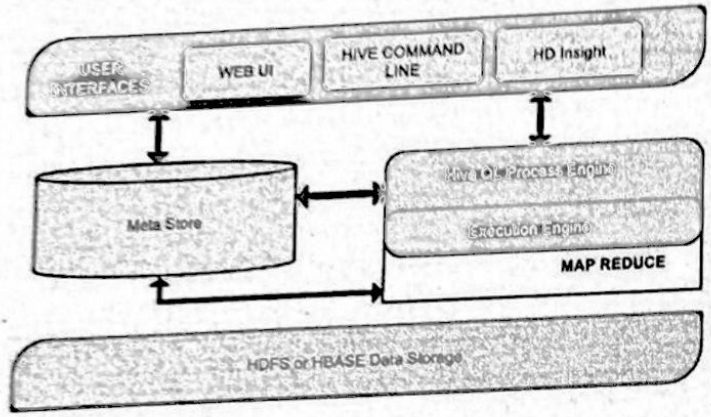


*Fig.*

This component diagram contains different units. The following table describes each unit:

| Unit Name | Operation |
|---|---|
| User Interface | Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server). |
| Meta Store | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. |
| HiveQL Process Engine | HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. |
| HDFS or HBASE | Hadoop distributed file system or HBASE are the data storage techniques to store data into file system. |

**Q.11** ***Explain HIVE CLI client.***

**Ans. The Hive CLI client :** To master the finer points of the Hive CLI client, it might help to revisit the (somewhat busy-looking) Hive architecture diagram shown in Figure, we've streamlined the original figure to focus only on the components that are required when running the CLI.

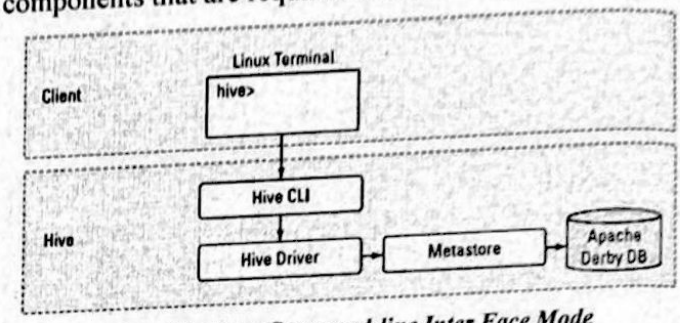


***Fig. : The Hive Command-line Inter Face Mode***

Illustrates the components of Hive that are needed when running the CLI on a Hadoop cluster. In this you run Hive in local mode, which uses local storage, rather than the HDFS, for your data.

To run the Hive CLI, you execute the Hive command and specify the CLI as the service you want to run. In Listing 1, you can see the command that's required as well as some of our first HiveQL statements. (We have included a steps annotation using the A-B-C model in the listing to direct your attention to the key commands.)

**Listing 1: Using the Hive CLI to Create a Table**

```
(A) $ $HIVE_HOME/bin hive --service cli
(B) hive > set hive.c1i.print.current.db=true;
(C) hive (default) > CREATE DATABASE ourfirstdatabase;
    OK
    Time taken: 3.756 seconds
(D) hive (default) > USE ourfirstdatabase;
    OK
    Time taken: 0.039 seconds
(E) hive (ourfirstdatabase) > CREATE TABLE our_first_cable
    (
      > FirstName       STRING,
      > Lasttiame       STRING,
      > Employeeld      INT) ;
    OK
```

```
Time taken: 0.043 seconds
hive (ourfirstdatabase) > quit;
(F) $ ls/home/biadmin/Hive/warehouse/ourfirstdatabase .db
our_first_table
```

The first command in Listing 1 (see Step A) starts the Hive CLI using the $HIVE_HOME environment variable.

**Listing 2:** Setting Up Apache Hive Environment Variables in .bashrc

```
export HADOOP_HOME=/home/user/Hive/hadoop/hadoop-1.2.1
export JAVA_HOME=/opt /jdk
export HIVE_HOME=/home/user/Hive/hive-0.11.0
export PATH=$HADOOP_ HOME/bin:$HIVE_HOME/bin:
$JAVA_HOME/bin: $ PATH
```

The service cli command-line option directs the Hive system to start the command-line interface, though you could have chosen other servers. ('In fact, you can try a few later in this section.) Next, in Step B, you tell the Hive CLI to print your current working database so that you know where you are in the namespace. (This statement will make sense after we explain how to use the next command, so hold tight.) Continuing in Listing 1, in Step C you use HiveQL's data definition language (DDL) to create your first database. (Remember that databases in Hive are simply namespaces where particular tables reside; because a set of tables can be thought of as a database or schema you could have used the term SCHEMA in place of DATABASE to accomplish the same result.) More specifically, you're using DDL to tell the system to create a database called ourfirstdatabase and then to make this database the default for subsequent HiveQL DDL commands using the USE command in Step D. In Step E, you create your first table and give it the (quite appropriate) name our_first_table. (Until now, you may have believed that it looks a lot like SQL, with perhaps a few minor differences in syntax depending on which RDBMS you're accustomed to – and you would have been right.)

The last command, in Step F, carries out a directory listing of your chosen Hive warehouse directory so that you can see that our_first_table has infact been stored on disk.

You set the hive.metastore .warehouse .dir variable to point to the local directory /home/biadmin/Hive/warehouse in your Linux virtual machine rather than use the HDFS as you would on a proper Hadoop cluster.

After you've created a table, it's interesting to view the table's metadata. In production environments, you might have dozens of tables or more, so it's helpful to be able to review the table structure from time to time. You can use a HiveQL command to do this using the Hive CLI, but the Hive Web Interface (HWI) Server provides a helpful interface for this type of operation. (More on HWI in the next section.)

Using the HWI Server instead of the CLI can also be more secure. Careful consideration must be made when using the CLI in production environments because the machine running the CLI must have access to the entire Hadoop cluster. Therefore, system administrators typically put in place tools like the secure shell (ssh) in order to provide controlled and secure access to the machine running the CLI as well as to provide network encryption. However, when the HWI Server is employed, a user can only access Hive data allowed by the HWI Server via his or her web browser.

**Q.12** ***Write a note on web browser as HIVE client.***

**Ans. The Web Browser as Hive client :** Using the Hive CLI requires only one command to start the Hive shell, but when you want to access Hive using a web browser, you first need to start the HWI Server and then point your browser to the port on which the server is listening. Figure 1 illustrates how this type of Hive client configuration might work. (Note that even though you might not be using the Hive CLI, it's not an optional component and is still present.)

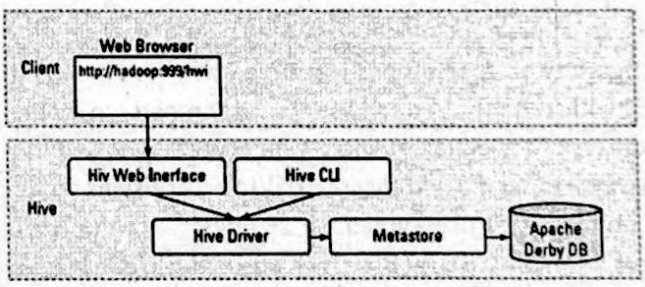


*Fig. 1 : The Hive Web Interface Client Configuration*

The following steps show you what you need to do before you can start the HWI Server:

1. Using the commands in Listing (following this list), configure the $HIVE_HOME/conf/hive-site.xml file to ensure that Hive can find and load the HWI's Java server pages.
2. The HWI Server requires Apache Ant libraries to run, so you need to download more files. Download Ant from the Apache site at http : / / ant.apache.org/ bindownload.cgi.

3. Install Ant using the following commands:

   mkdir ant

   cp apache-ant-1.9.2-bin.tar.gz ant; cd ant

   gunzip apache-ant-1.9.2-bin.tar.gz

   tar xvf apache-ant-1.9.2-bin.tar

4. Set the $ANT_LIB environment variable and start the HWI Server by using the following commands:

   $ export ANT_LIB=/home/user/ant/apache-ant-1.9.2/lib

   $ bin/hive --service hwi

   13/09/24 16:54:37 INFO hwi.HWIServer: HWI is starting up

   ...

   13/09/24 16:54:38 INFO mortbay.log: Started SocketConnector@0. 0 . 0 . 0 : 9999

   Listing : Configuring the $HIVE_HOME/conf/hive-site.xml file

   <property>

   <name>hive.hwi.war.file</name>

   <value>$ {HIVE_HOME)/lib/Give_hwi . war</value>

   <description> This is the WAR file with the jsp content for Hive Web Interface</description>

   </property>

In a production environment, you'd probably configure two other properties: hive.hwi.listen.host:and hive.hwi.listen.port. You can use the first property to set the IP address of the system running your HWI Server, and use the second to set the port that the HWI Server listens on. In this exercise, you use the default settings: With the HWI Server now running, you simply enter the URL http:// 1ocalhost:9999 /hwi/ into your web browser and view the metadata for our_first_table.

**Listing :** *Refer to Q.11, Listing 1.*

Figure 2 shows what the screen looks like after selecting the Browse Schema link followed by ourfirstdatabase and our_first_table.

In production environments, working with the HWI Server can save you the time of loading the Hive distribution on every client – instead, you just point your browser to the server running the HWI. Additionally, you can use the HWI Server to view Hive Thrift Server diagnostics and query tables. The HWI Server allows you to set up batch sessions for long-running queries. To set up a session, you simply click the Create Session link (Figure 2).

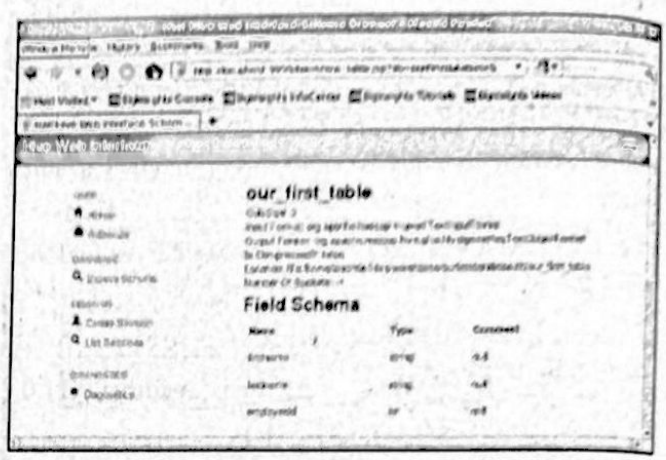


*Fig. 2 : Using the Hive Web Interface to browse the metadata*

**Q.13** ***Explain SQuirreL as hive client.***

**Ans. SQuirreL as Hive client :** You can download this universal SQL client from the SourceForge website: http :// sourceforge.net. It provides a user interface to Hive and simplifies the tasks of querying large tables and analyzing data with Apache Hive.

Figure 1 illustrates how the Hive architecture would work when using tools such as SQuirreL.

In the figure 1, you can see that the SQuirreL client uses the JDBC APIs to pass commands to the Hive Driver by way of the Server.

For a helpful example of a Hive Java client connecting to the system via the JDBC interface, see

https : / /cwiki . apache . org/confluence/display/Hive/HiveClient#Hiveclient-JDBC

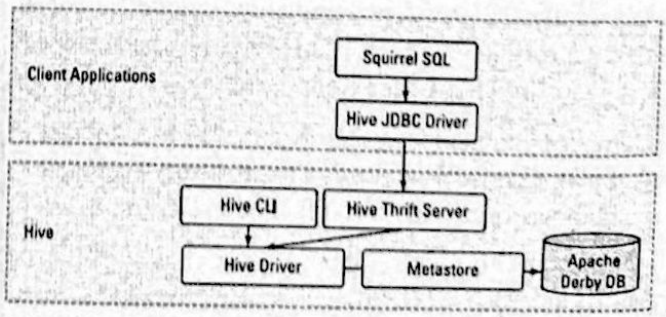


*Fig. 1 : Using the client with SQuirreL Apache Hive*

**Follow these steps to get SQuirreL running:**

1. Start the Hive Thrift Server using the command in Listing (following list).
2. Download the latest SQuirreL distribution from the SourceForge site into a directory of your choice.

   For this example, we downloaded squirrel–sql–3.5.0-standard.tar.gz from http : //sourceforge .net/projects/squirrel-sql/files/1-stable/3.5.0-plainzip.
3. Uncompress the SQuirreL package using the gunzip command and expand the archive using the tar command.

   gunzip squirrel-sql- 3 . 5 . 0 -st:andard . tar. gz; tar xvf squirrel-sql-3 . 5 . 0-standard. tar .gz
4. Change to the new SQuirreL. release directory and start the tool using the following command.

   $ cd squirrel - sql- 3 . 5 . 0 -standard; . /squirrel-sql.sh

   $ cd squirrel - sql- 3 . 5 . 0 -standard; . /squirrel- sql .sh
5. Follow the directions for running SQuirreL with Apache Hive at

   https : / /cwiki . apache . org/confluence/disp1ay/Hive/HiveJDBCInterface - HiveJDBCInterface-IntegrationwithSQuirrelSQLC1ient

Note that the instructions for including the Hadoop core . jar file may differ depending on the Hadoop release. In this case, the Hadoop . jar file was named hadoop-core- 1.2.1.jar, so including $HADOOP_ HOME/hadoop - * - core . jar per the online instructions was incorrect. We had to use $HADOOP_HOME/hadoop-core*.jar.

**Listing : Starting the Hive Thrift Server**

```
$ $Hive_HOME/bin/hive --service hiveserver -p. 10000 -v
Starting Hive Thrift Server
Starting Hive Thrift Server on port 10000 with 100 min
worker threads and 2147483647 max worker
threads
```

This is all that's required to begin using the SQuirreL graphical user interface. Figure 2 shows some HiveQL commands running against the Hive Driver.

Listing : *Refer to Q.11, Listing 1.*

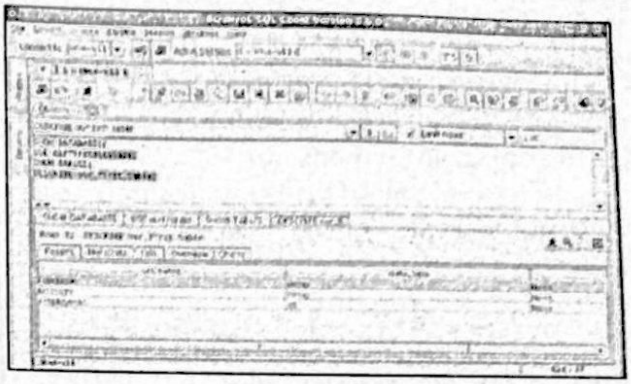

*Fig. 2*

The Apache Hive 0.11 release also includes a new Hive Thrift Server called HiveServer2. When configured correctly, HiveServer2 can support multiple clients (a CLI client and a SQuirreL client at the same time, for example) and it provides better security.

**Q.14** ***Write the features of query language.***

**Ans. Security:** Apache Hive provides a security subsystem that can be quite helpful in preventing accidental data corruption or compromise among trusted members of workgroups. However, as of this writing, the Hive Language Manual clearly states that the Hive Security subsystem isn't designed to prevent nefarious users from compromising a Hive system. Hive security can be established for individual users, groups, and administrative roles. Hive provides privileges that can be granted or revoked to users, groups, or administrative roles. The Hive 0.10 release improved security in multi-user environments by providing authorization to the metastore, and future Hive releases will provide increasing integration with the Hadoop security framework. Kerberos is emerging as the technology of choice for securing Apache Hadoop.

**Multi-User Locking:** Hive supports multi-user warehouse access when configured with Apache Zookeeper. Without this support, one user may read a table at the same time another user is deleting that table which is, obviously, unacceptable. Multi-user access is enabled via configuration variables in the hive-site.xml file. Once configured, Hive implicitly acquires locks through Zookeeper for certain table operations. Users can also explicitly manage locks in the Hive CLI. Locks and associated configuration properties/variables are described in the Hive Language Manual.

**Compression:** Data compression cannot only save space on the HDFS but also improve performance by reducing the overall size of input/output operations. Additionally, compression between the Hadoop mappers and reducers can improve performance, because less data is passed between nodes in the cluster. Hive supports intermediate compression between the mappers and reducers as well as table output compression. Hive also understands how to ingest compressed data into the warehouse. Files compressed with Gzip or Bzip2 can be read by Hive's LOAD DATA command.

**Functions:** HiveQL provides a rich set of built-in operators, built-in functions, built-in aggregate functions, and built-in table-generating functions. To list all built-in functions for any particular Hive release, use the SHOW FUNCTIONS HiveQL command. You can also retrieve information about a built-in function by using the HiveQL commands DESCRIBE FUNCTION function_name and DESCRIBE FUNCTION EXTENDED function_name. Using the EXTENDED keyword sometimes returns usage examples for the specified built-in function. Additionally, Hive allows users to create their own functions, called user-defined functions, or UDFs. Using Hive's Java-based UDF framework, you can create additional functions, including aggregates and table-generating functions. This feature is one of the reasons that Hive can function as an ETL tool.

**Q.15** ***Explain various types of joins in HIVE QL.***

**Ans. JOIN :** There are different types of joins given as follows:

- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

The following query executes JOIN on the CUSTOMER and ORDER tables and retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN
ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response

| ID | NAME | AGE | AMOUNT |
|---|---|---|---|
| 3 | kaushik | 23 | 3000 |
| 3 | kaushik | 23 | 1500 |
| 2 | Khilan | 25 | 1560 |
| 4 | Chaitali | 25 | 2060 |

**LEFT OUTER JOIN :** The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table. A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

The following query demonstrates LEFT OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c LEFT
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 1 | Ramesh | NULL | NULL |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
| 5 | Hardik | NULL | NULL |
| 6 | Komal | NULL | NULL |
| 7 | Muffy | NULL | NULL |

**RIGHT OUTER JOIN :** The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c RIGHT
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 3 | kaushik | 3000 | 2009-10-08 |
| 3 | kaushik | 1500 | 2009-10-08 |
| 2 | Khilan | 1560 | 2009-11-20 |
| 4 | Chaitali | 2060 | 2008-05-20 |

**FULL OUTER JOIN**

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c FULL
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 1 | Ramesh | NULL | NULL |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
| 5 | Hardik | NULL | NULL |
| 6 | Komal | NULL | NULL |
| 7 | Muffy | NULL | NULL |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

## PART-C

**Q.16** ***Explain the working of HIVE with neat sketch.***

**Ans.** The following diagram depicts the workflow between Hive and Hadoop

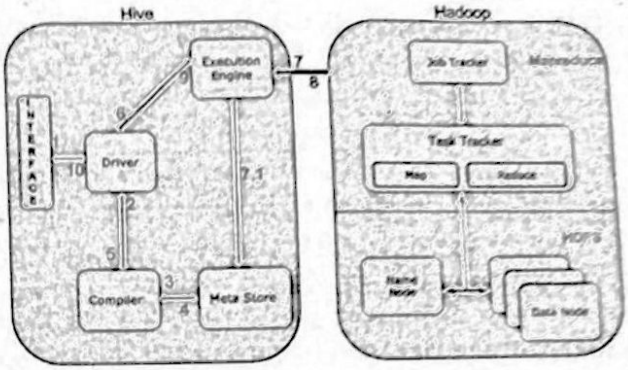


*Fig.*

The followings define how Hive interacts with Hadoop framework:

1. **Execute Query :** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2. **Get Plan :** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3. **Get Metadata :** The compiler sends metadata request to Metastore (any database).
4. **Send Metadata :** Metastore sends metadata as a response to the compiler.
5. **Send Plan :** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6. **Execute Plan :** The driver sends the execute plan to the execution engine.
7. **Execute Job :** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
   - **Metadata Ops :** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8. **Fetch Result :** The execution engine receives the results from Data nodes.
9. **Send Results to the Driver :** The execution engine sends those resultant values to the driver.
10. **Send Results to Hive Interface :** The driver sends the results to Hive Interfaces.

BDA.78

**Q.17** ***Explain the installation of hive or getting started with HIVE.***

**Ans.** As with most technological matters, there's no better way to see what's what than to install the software and give it a test run – Hive is no exception. And, as with other technologies in the Hadoop ecosystem, it doesn't take long to get started.

If you have the time and the network bandwidth, it's always best to download an entire Apache Hadoop distribution with all the technologies integrated and ready to run. You can find a list of Apache Hadoop bundles at

http://wiki.apache .org/hadoop/Distributions% 20and%20 Commercial%20Support

If you take the full-distribution route, a popular approach for learning the ins and outs of Hive is to run your Hadoop distribution in a Linux virtual machine (VM) on a 64-bit-capable laptop with sufficient RAM. (Eight gigabytes or more of RAM tends to work well if Windows 7 is hosting your VM, although we've met engineers who live dangerously with less.) You also need Java 6 or later and – of course – a supported operating system: Linux, Mac OS X, or Cygwin, to provide a Linux shell for Windows users. (We use Red Hat Linux on Windows 7 in a VMware virtual machine for the sample environment.)

The setup steps run something like this:

**1. Download the latest Hive release from this site:**

http://hive.apache.org/releases.html

By this, we downloaded Hive version 11.0. You also need the Hadoop and MapReduce subsystems, so be sure to complete Step 2.

**2. Download Hadoop version 1.2.1 from this site:**

http: //hadoop.apache.org/releases .html

**3. Using the commands in Listing 1 :** (the listing following this step list), place the releases in separate directories, and then uncompress and untar them. (Untar is one of those pesky Unix terms which simply means to expand an archived software package.)

**4. Using the commands in Listing 2 :** (again, following this step list), set up your Apache Hive environment variables, including HADOOP_HOME, JAVA_HOME, HIVE_HOME and PATH, in your shell profile script.

**5.** Create the Hive configuration file that you'll use to defin specific Hive configuration settings.

The Apache Hive distribution includes a templat configuration file that provides all default settings for Hiv To customize Hive for your environment, all you need to d is copy the template file to the file named hive-site.xml an edit it. Listing 3 shows the steps to accomplish this task.

Because you're running Hive in stand-alone mode o a virtual machine rather than in a real-life Apache Hadoo cluster, configure the system to use local storage rather tha the HDFS: Simply set the hive. metastore.warehouse.di parameter. As we demonstrate in the next section, when yo start a Hive client, the $HIVE_HOME environment variabl tells the client that it should look for your configuration fil (hive-site.xml) in the conf directory.

**Listing 1:** Installing Apache Hadoop and Hive

```
$ mkdir hadoop; cp hadoop-1.2.1.tar.gz hadoop; cd
hadoop
$ gunzip hadoop-1.2.1.tar.gz
$ tar xvf *.tar
$ mkdir hive; cp hive-0.11.0.tar.gz hive; cd hive
$ gunzip hive-0.11.0.tar.gz
$ tar xvf *.tar
```

**Listing 2:** Setting Up Apache Hive Environment Variable in .bashrc

```
export HADOOP_HOME=/home/user/Hive/hadoop
hadoop-1.2.1
export JAVA_HOME=/opt /jdk
export HIVE _HOME=/home/user/Hive/hive-0.11.0
export PATH=$HADOOP_ HOME/
bin:$HIVE_HOME/bin:
$JAVA_HOME/bin: $ PATH
```

**Listing 3:** Setting Up the hive-site.xml File

```
$ cd $HIVE_HOME/conf
$ cp hive-default .xml.template hive-site.xml
```

(Using your favorite editor, modify the hive-site.xml file so that it only includes the "hive.metastore. warehouse .dir" property for now. When finished it will look like the XML file below. Note that we removed the comments to shorten the listing) :

```
<?xml version="1.0"?>
<?xml-stylesheet type="text /xsl"
    href="configuration.xsl"?>
<configuration>
<!-- Hive Execution Parameters -->
<property>
<name>hive.metastore. warehouse .dir</name>
<value>/home /biadmin/Hive/warehouse</value>
<description>location of default database for the
warehouse</description>
</property>
</configuration>
```

**Hive - Installation**

All Hadoop sub-projects such as Hive, Pig, and HBase support Linux operating system. Therefore. you need to install any Linux flavored OS. The following simple steps are executed for Hive installation:

**Step 1: Verifying JAVA Installation**

Java must be installed on your system before installing Hive. Let us verify java installation using the following command:

```
$ java -version
```

If Java is already installed on your system, you get to see the following response:

```
java version "1.7.0 71"
Java(TM) SE Runtime Environment (build 1.7.0 71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If java is not installed in your system, then follow the steps given below for installing java.

**Installing Java**

**Step I :** Download java (JDK <latest version> - X64.tar.gz) by visiting the following link http://www.oracle.com/technetwork/java/javase/downloads/jdk7/-downloads-1880260.html.

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

**Step II :** Generally you will find the downloaded java file in the Downloads folder. Verify it and extract the jdk-/u/1-linux-x64.gz file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz
$ tar zxf jdk-7u71-linux-x64.gz
$ ls
jdk1.7.071 jdk-7u71-linux-x64.gz
```

**Step III :** To make java available to all the users, you have to move it to the location "/usr/local/". Open root, and type the following commands.

```
$ su
password:
# mv.jdk1.7.0-71 /usr/local/
# exit
```

**Step IV :** For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/jdk01.7.0–71
export PATH=PATH: $JAVA_HOME/bin
```

Now verify the installation using the command java - version from the terminal as explained above.

**Step 2 : Verifying Hadoop Installation**

Hadoop must be installed on your system before installing Hive. Let us verify the Hadoop installation using the following command:

```
$ hadoop version
```

If Hadoop is already installed on your system, then you will get the following response:

```
Hadoop 2.4.1 Subversion https://svn.apache.org/repos/asf/hadoop/common -r 152976
Compiled by hortonmu on 2013-10-07T06: 28Z
Compiled with protoc 2.5.0
From source with checksum
79e53ce7994d1628b240f09af9lelaf4
```

If Hadoop is not installed on your system, then proceed with the following steps:

**Downloading Hadoop**

Download and extract Hadoop 2.4.1 from Apache Software Foundation using the following commands.

```
$ su
password:
```

# cd /usr/local

# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/hadoop-2.4.1.tar.gz

# tar xzf hadoop-2.4.1.tar.gz

# mv hadoop-2.4.1/* to hadoop/

# exit

**Installing Hadoop in Pseudo Distributed Mode**

The following steps are used to install Hadoop 2.4.1 in pseudo distributed mode.

**Step I : Setting up Hadoop**

You can set Hadoop environment variables by appending the following commands to ~/.bashrc file.

export HADOOP_HOME=/usr/local/hadoop

export HADOOP_MAPRED_HOME = $HADOOP_HOME

export HADOOP_COMMON_HOME=$HADOOP_HOME

export HADOOP_HDFS_HOME=$HADOOP_HOME

export YARN_HOME=$HADOOP_HOME

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native export

PATH=$PATH: $HADOOP_HOME/sbin: $HADOOP_HOME/bin

Now apply all the changes into the current running system.

$ source ~/.bashrc

**Step II : Hadoop Configuration**

You can find all the Hadoop- configuration files in the – location "$HADOOP_HOME/etc/hadoop". You need to make suitable changes in those configuration files according to your Hadoop infrastructure.

$ cd $HADOOP_HOME/etc/hadoop

In order to develop Hadoop programs using java, you have to reset the java environment

variables in hadoop-env.sh file by replacing JAVA_HOME value with the location of java in your system.

export JAVA_HOME=/usr/local/jdk1.7.0_71

Given below are the list of files that you have to edit to configure Hadoop.

**core-site.xml**

The core-site.xml file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and the size of Read/Write buffers.

Open the core-site.xml and add the following properties in between the <configuration> and </configuration> tags.

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost : 900@</value>

</property>

</configuration>

**hdfs-site.xml**

The hdfs-site.xml file contains information such as the value of replication data, the namenode path, and the datanode path of your local file systems. It means the place where you want to store the Hadoop infra.

Let us assume the following data.

dfs.replication (data replication value) = 1

(In the following path /hadoop/ is the user name.

hadoopinfra/hdfs/namenode is the directory created by hdfs file system. )

namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system. )

datanode path = //home/hadoop/hadoopinfra/hdfs/datanode

Open this file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>dfs.replication</name>

<value>i1</value>

</property>

<property>

<name>dfs .name.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>

</property>

<property>

<name>dfs .data.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>

</property>

</configuration>

[Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.]

**yarn-site.xml**

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>yarn. nodemanager . aux- services</name>

<value>mapreduce_shuffle</value>

</property>

</configuration>

**mapred-site.xml**

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, you need to copy the file from mapred-site, xml.template to mapred-site.xml file using the following command.

$ cp mapred-site.xml.template mapred-site.xml

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>mapreduce. framework .name</name>

<value>yarn</value>

</property>

</configuration>

**Verifying Hadoop Installation**

The following steps are used to verify the Hadoop installation.

**Step I : Name Node Setup**

Set up the namenode using the command "hdfs namenode -format" as follows.

$ cd ~

$ hdfs namenode -format

The expected result is as follows.

10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:

/**********

STARTUP_MSG: Startir.g NameNode

STARTUP_MSG: host = localhost/192.168.1.11

STARTUP_MSG: args = [-format]

STARTUP_MSG: version = 2.4.1

...

...

10/24/14 21:30:56 INFO common.Storage: Storage directory /home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.

10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0

10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0

10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:

/**********

**Step II : Verifying Hadoop dfs**

The following command is used to start dfs. Executing this command will start your Hadoop file system.

$ start–dfs.sh

The expected output is as follows:

10/24/14 21:37:56

Starting namenodes on [localhost]

localhost: starting namenode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-h

localhost: starting datanode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-h

Starting secondary namenodes [0.0.0.0]

**Step III : Verifying Yarn Script**

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

$ start-yarn.sh

The expected output is as follows:

starting yarn daemons

starting resourcemanager, logging to /home/hadoop/hadoop-2.4.1/1ogs/yarn-hadoop-localhost: startingnode manager, logging to /home/hadoop/hadoop-2.4.l/1ogs/yarn-

**Step IV : Accessing Hadoop on Browser**

The default port number to access Hadoop is 50070. Use the following url to get Hadoopservices on your browser.

http://1ocalhost:50070/

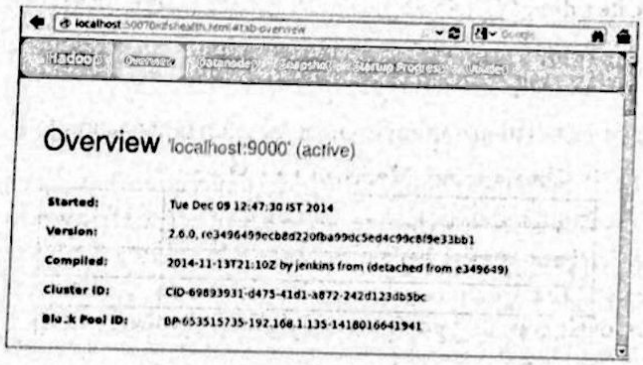


*Fig.*

**Step V: Verify all applications for cluster**

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

http://1ocalhost:8088/

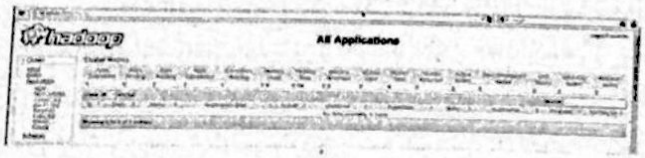


*Fig.*

**Step 3: Downloading Hive**

We use hive-0.14.0 in this tutorial. You can download it by visiting the following link http://apache.petsads.us/hive/hive-0.14.0/. Let us assume it gets downloaded onto the /Downloads directory. Here, we download Hive archive named "apache-hive-0.14.0- bin.tar.gz". The following command is used to verify the download:

$ cd Downloads

$ ls

On successful download, you get to see the following response:

apache-hive-0.14.0-bin.tar.gz

**Step 4: Installing Hive**

The following steps are required for installing Hive on your system. Let us assume the Hive archive is downloaded onto the /Downloads directory.

**Extracting and verifying Hive Archive**

The following command is used to verify the download and extract the hive archive:

$ tar zxvf apache-hive-0.14.0-bin.tar.gz

$ ls

On successful download, you get to see the following response:

apache-hive–0.14.0-bin apache–hive–0.14.0-bin.tar.gz

**Copying files to /usr/local/hive directory**

We need to copy the files from the super user "su-". The following commands are used to copy the files from the extracted directory to the /usr/local/hive" directory.

$ su »

passwd:

# cd /home/user/Download

# mv apache-hive-0.14.0-bin /usr/local/hive

# exit

**Setting up environment for Hive**

You can set up the Hive environment by appending the following lines to ~/.bashrc file:

export HIVE_HOME=/usr/local/hive

export PATH=$PATH:$HIVE_HOME/bin

export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:

export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:

The following command is used to execute ~/.bashrc file.

$ source ~/.bashrc

**Step 5: Configuring Hive**

To configure Hive with Hadoop, you need to edit the hive-env.sh file, which is placed in the $HIVE_HOME/conf directory. The following commands redirect to Hive config folder and copy the template file:

$ cd $HIVE_HOME/conf

$ cp hive-env.sh.template hive-env.sh

Edit the hive-env.sh file by appending the following line:

export HADOOP_HOME=/usr/local/hadoop

Hive installation is completed successfully. Now you require an external database server to configure Metastore. We use Apache Derby database.

**Step 6: Downloading and Installing Apache Derby**

Follow the steps given below to download and install Apache Derby:

**Downloading Apache Derby**

The following command is used to download Apache Derby. It takes some time to download.

$ cd ~

$ wget http://archive.apache.org/dist/db/derby/db-derby–10.4.2.0/db-derby-10.4.2

The following command is used to verify the download:

$ ls

On successful download, you get to see the following response:

db-derby-18.4:2.0-bin.tar.gz

**Extracting and verifying Derby archive**

The following commands are used for extracting and verifying the Derby archive:

$ tar zxvf db-derby-10.4.2.0-bin.tar.gz

$ ls

On successful download, you get to see the following response :

db-derby-10.4.2.0-bin db-derby-10.4.2.0-bin.tar.gz

**Copying files to /usr/local/derby directory**

We need to copy from the super user "su -". The following commands are used to copy the files from the extracted directory to the /usr/local/derby directory:

$ su -

passwd:

# cd /home/user

# mv db-derby-10.4.2.0-bin /usr/local/derby

# exit

**Setting up environment for Derby**

You can set up the Derby environment by appending the following lines to ~/.bashrc file:

export DERBY_HOME=/usr/local/derby

export PATH=$PATH:$DERBY_HOME/bin

Apache Hive

18

export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools

The following command is used to execute -/.bashrc file:

$ source ~/.bashrc

**Create a directory to store Metastore**

Create a directory named data in $DERBY_HOME directory to store Metastore data

$ mkdir $DERBY_HOME/data

Derby installation and environmental setup is now complete.

**Step 7: Configuring Metastore of Hive**

Configuring Metastore means specifying to Hive where the database is stored. You can do this by editing the hive-site.xml file, which is in the $HIVE_HOME/conf directory. First of all, copy the template file using the following command:

$ cd $HIVE_HOME/conf

$ cp hive-default.xml.template hive-site.xml

Edit hive-site.xml and append the following lines between the <configuration> and </configuration> tags:

<property>

<name>javax.jdo.option.ConnectionURL</name)

<value>jdbc:derby://localhost:1527/metastore_db;create =true </value>

<description>JDBC connect string for a JDBC metastore </description>

</property>

Create a file named jpox.properties and add the following lines into it:

javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl

org.jpox.autoCreateSchema = false

org.jpox.validateTables = false

org.jpox.validateColumns = false

org.jpox.validateConstraints = false

org.jpox.storeManagerType = rdbms

org.jpox.autoCreateSchema = true

org.jpox.autoStartMechanismMode = checked

org.jpox.transactionIsolation = read_committed

javax.jdo.option.DetachA110nCommit = true

javax.jdo.option.NontransactionalRead= true

javax.jdo.option.ConnectionDriverName= org.apache.derby.jdbc.C1ientDriver

javax.jdo.option.ConnectionURL = jdbc: derby:// hadoopl:1527/metastore_db;create

javax.jdo.option.ConnectionUserName = APP

javax.jdo.option.ConnectionPassword = mine

**Step 8: Verifying Hive installation**

Before running Hive, you need to create the /tmp folder and a separate Hive folder in HDFS. Here, we use the /user/hive/warehouse folder. You need to set write permission for these newly created folders as shown below:

chmod g+w

Now set them in HDFS before verifying Hive. Use the following commands :

$ $HADOOP_HOME/bin/hadoop fs-mkdir /tmp

$ $HADOOP_HOME/bin/hadoop fs-mkdir /user/hive/warehouse

$ $HADOOP_HOME/bin/hadoop fs-chmod g+w /tmp

$ $HADO0P_HOME/bin/hadoop fs-chmod g+w /user/hive/warehouse

The following commands are used to verify Hive installation:

$ cd $H IVE_HOME

$ bin/hive

On successful installation of Hive, you get to see the following response

Logging initialized using configuration in jar:file:/home/hadoop/hive-0.9.0/1ib/

Hive history file=/tmp/hadoop/hive_job_log_hadoop_201312121621_1494929084 txt

********.

hive>

The following sample command is executed to display all t tables :

hive> show tables;

OK

Time taken: 2.798 seconds

hive>

The following sample command is executed to displa all the tables :

hive> show tables;

OK

Time taken: 2.798 seconds

hive>

**Q.18** ***Explain various datatypes in hive.***

**Ans. Hive - Data Types**

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

**Column Types :** Column type are used as column data types of Hive. They are as follows:

**Integral Types :** Integer type data can be specified using integral data types, INT. \when the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

| Type | Postfix | Example |
|---|---|---|
| TINYINT | Y | 10Y |
| SMALLINT | S | 10S |
| INT | - | 10 |
| BIGINT | L | 10L |

**String Types :** String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR Hive follows C-types escape characters.

The following table depicts various CHAR data types:

| Data Type | Length |
|---|---|
| VARCHAR | 1 to 65355 |
| CHAR | 255 |

**Timestamp :** It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format "YYYY-MM-DD HH:MM:SS.fffffffff" and format "yyyy-mm-dd hh:mm:ss.fffffffff.

**Dates :** DATE values are described in year/'month'day format in the form {{YYYY– MM-DD}}.

**Decimals :** The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

**Union Types :** Union is a collection of heterogeneous data types. You can create an instance using create union. The syntax and example is as follows:

UNIONTYPE<int , double , array<string> , struct<a: "int, b:string>>

{0:1}

{1:2 . 0}

{2 : ["three" , "four"]}

{3:{"a" : 5 , "b" : "five"}}

{2 : ["six" , "seven"]}

{3 : {"a" :8, "b" : "eight"}}

{0 : 9}

{1:10.0}

**Literals :** The following literals are used in Hive:

**Floating Point Types :** Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

**Decimal Type :** Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10 -308 to 10 308 .

**Null Value :** Missing values are represented by the special value NULL.

**Complex Types :** The Hive complex data types are as follows:

**Arrays :** Arrays in Hive are used the same way they are used in Java.

**Syntax:** ARRAY<data_type>

**Maps :** Maps in Hive are similar to Java Maps.

**Syntax :** MAP<primitive_type: data_type>

**Structs :** Structs in Hive is similar to using complex data with comment.

**Syntax:** STRUCT<col_name: data_type [COMMENT col_comment], ...>

---

**Q.19** ***Write detailed note on creating and managing database and tables.***

---

**Ans. Hive - Create Database :** Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. Hive contains a default database named default.

**Create Database Statement**

- Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows: CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
- Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named userdb:

  hive > CREATE DATABASE [IF NOT EXISTS] userdb;

  or

  hive > CREATE SCHEMA userdb;
- The following query is used to verify a databases list:

  hive> SHOW DATABASES;

  default

  userdb

**Drop Database Statement**

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS]

database_name [RESTRICT | CASCADE];

The following queries are used to drop a database. Let us assume that the database name is userdb.

hive> DROP DATABASE IF EXISTS userdb;

The following query drops the database using CASCADE. It means dropping respective tables before dropping the database.

hive> DROP DATABASE IF EXISTS userdb CASCADE;

The following query drops the database using SCHEMA.

hive> DROP SCHEMA userdb;

**Create Table Statement**

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col__comment], ...)]

[COMMENT table_comment]

[ROW FORMAT row_format] [STORED AS file_format]

**Example**

Let us assume you need to create a table named employee using CREATE TABLE statement. The following table lists the fields and their data types in employee table:

| S.No. | Field Name | Data Type |
|---|---|---|
| 1. | Eid | int |
| 2. | Name | String |
| 3. | Salary | Float |
| 4. | Designation | string |

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

COMMENT Employee details FIELDS TERMINATED BY\tLINES TERMINATED

BY\nSTORED IN TEXT FILE

The following query creates a table named employee using the above data.

hive> CREATE TABLE IF NOT. EXISTS employee(eid int, name String, salary String, destination String)

COMMENT Employee details ROW FORMAT DELIMITED FIELDS TERMINATED BY\tLINES TERMINATED BY\nStored AS TEXTFILE;

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists

On successful creation of table, you get to see the following response:

OK

Time taken; 5.905 seconds

hive>

**Load Data Statement**

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system

While inserting data into Hive, it is better to use LOAD DATA to store bulk records There are two ways to load data: one is from local file system and second is from Hadoop file system

**Syntax**

The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION [partcol1=vall, partcol2=val2 ...)]

- LOCAL is identifier to specify the local path. It is optional
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

**Example**

We will insert the following data into the table. It is a text file named sample.txt in /home/user directory.

| | | | |
|---|---|---|---|
| 1201 | Gopal | 45000 | Technical manager |
| 1202 | Manisha | 45000 | Proof reader |
| 1203 | Masthanvali | 40000 | Technical writer |
| 1204 | Kiran | 40000 | Hr Admin |
| 1205 | Kranthi | 30000 | Op Admin |

The following query loads the given text into the table

hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt' OVERWRITE INTO TABLE employee;

On successful download, you get to see the following response:

OK

Time taken : 15.905 seconds

hive>

**Alter Table Statement**

It is used to alter a table in Hive.

**Syntax**

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new_name

ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])

ALTER TABLE name DROP [COLUMN] column_name

ALTER TABLE name CHANGE column_name new_name new_type

ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec...])

**Rename To Statement**

The following query renames the table from employee to emp.

hive> ALTER TABLE employee RENAME TO emp;

**Change Statement**

The following table contains the fields of employee table and it shows the fields to be changed (in bold).

| Field name | Convert from data type | Change field name | Convert to data type |
|---|---|---|---|
| eid | int | eid | int |
| name | String | ename | String |
| salary | Float | salary | Double |
| designation | String | designation | String |

The following queries rename the column name and column data type using the above data:

hive> ALTER TABLE employee CHANGE name ename String;

hive> ALTER TABLE employee CHANGE salary salary Double;

**Add Columns Statement**

The following query adds a column named dept to the employee table.

hive> ALTER TABLE employee ADD COLUMNS (dept STRING COMMENT 'Department name');

**Replace Statement**

The following query deletes all the columns from the employee table and replaces it with emp and name colunms:

hive> ALTER TABLE employee REPLACE COLUMNS (eid INT empid Int, ename STRING name String);

**Drop Table Statement**

Hive Metastore, it removes the table/column data and their metadata. It can be a normal table (stored in Metastore) or an external table (stored in local file system); Hive treats both in the same manner, irrespective of their types.

The syntax is as follows:

DROP TABLE [IF EXISTS] table_name;

The following query drops a table named employee:

hive> DROP TABLE IF EXISTS employee;

On successful execution of the query, you get to response:

OK

hive>

The following query is used to verify the list of tables:

Hive> SHOW TABLES;

emp ok

Time taken; 2.1 seconds

hive>

**Creating, Dropping and Altering Databases in Apache Hive is as below –**

(1) $ $HIVE_HOME/bin hive --service cli

(2) hive> set hive.cli.print.current.db = true;

(3) hive (default)> USE ourfirstdatabase;

(4) hive (ourfirstdatabase)> ALTER DATABASE ourfirstdatabase SET DBPROPERTIES ('creator'='Bruce Brown', 'created_for'='Learning Hive DDL');

OK

Time taken: 0.138 seconds

(5) hive (ourfirstdatabase)> DESCRIBE DATABASE EXTENDED ourfirstdatabase;

OK

ourfirstdatabase

file:/home/biadmin/Hive/warehouse/

ourfirstdatabase.db {created_for=Learning

Hive DDL, creator=Bruce Brown}

Time taken: 0.084 seconds, Fetched: 1 row(s)CREATE

(DATABASE|SCHEMA) [IF NOT EXISTS] database_name

(6) hive (ourfirstdatabase)> DROP DATABASE

ourfirstdatabase CASCADE;

OK

Time taken: 0.132 seconds

In Line 4 of above instructions, we're now altering the database which have already created with the name ourfirstdatabase to include two new metadata items: creator and created_for. These two can be quite useful for documentation purposes and coordination within your working group. The command in Line 5 is used to view the metadata. With the help of command in Line 6 we're dropping the entire database – removing it from the server. We can use the DROP TABLE command to delete individual tables.

**Creating and managing tables with Hive**

Apache Hive lets you define the record format separately from the file format. Hive tables default to the configuration in below Listing unless you override the default settings.

CREATE TABLE

...

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE

The following listing specifies on how fields will be separated or delimited whenever you insert or load data into the table.

(l) Hive> CREATE TABLE data_types_table(

...

(33) > ROW FORMAT DELIMITED

(34) > FIELDS TERMINATED BY '^'

(35) > COLLECTION ITEMS TERMINATED BY '|'

(36) > MAP KEYS TERMINATED BY '^'

(37) > LINES TERMINATED BY '\n'

(38) > STORED AS TEXTFILE

...

(39) > TBLPROPERTIES ('creator'='Bruce Brown', 'created_at'='Sat Sep 21 20:46:32 EDT 2013');

In the above listing Lines 33-37 define the Hive row format for our data_types_table. Line 38 defines the Hive file format – a text file – when the data is stored in the HDFS.

So far, we have been using the default TEXTFILE format for your Hive table records. However, as you know, text files are slower to process, and they consume a lot of disk space unless you compress them. For these reasons and more, the Apache Hive community came up with several choices for storing our tables on the HDFS.

**File Formats of Hive**

The following list describes the file formats you can choose from as of Hive version 0.1 l.

**TEXTFILE**: The default file format for Hive records. Alphanumeric characters from the Unicode standard are used to store your data.

**SEQUENCEFILE**: The format for binary files composed of key/value pairs. Sequence files, which are used heavily by Hadoop, are often good choices for Hive table storage, especially if you want to integrate Hive with other technologies in the Hadoop ecosystem.

**RCFILE:** RCFILE stands for record columnar file. Stores records in a column-oriented fashion rather than a row-oriented fashion – like the TEXTFILE format approach

**ORC:** ORC stands for optimized row columnar. A format (new as of Hive 0.11) that has significant optimizations to improve Hive reads and writes and the processing of tables. For example, ORC files include optimizations for Hive complex types and new types such as DECIMAL. Also lightweight indexes are included with ORC files to improve performance.

**INPUTFORMAT, OUTPUTFORMAT:** INPUTFORMAT will read data from the Hive table, OUTPUTFORMAT does the same thing for writing data to the Hive table. To see the default settings for the table, simply execute a DESCRIBE

EXTENDED tablename HiveQL statement and we'll see the INPUTFORMAT and OUTPUTFORMAT classes for your table.

**Defining table record formats**

The Java technology that Hive uses to process records and map them to column data types in Hive tables is called SerDe, which is short for SerializerDeserializer. Figure will help us to understand how Hive keeps file formats separate from record formats.

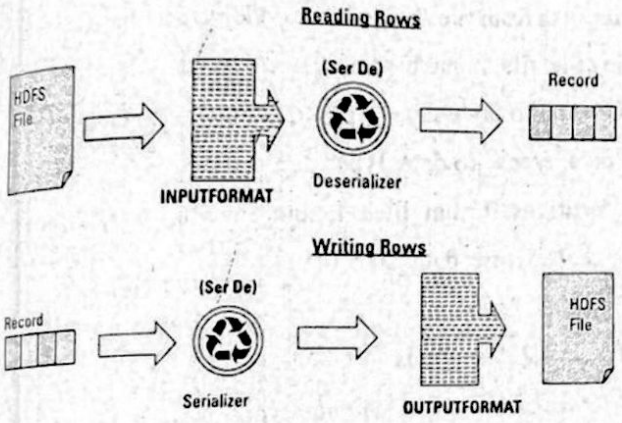


**Fig. : How Hive Reads and Writes Records**

When Hive is reading data from the HDFS (or local file system), a Java Deserializer formats the data into a record that maps to table column data types. It is used at the time of HiveQL SELECT statement. When Hive is writing data, a Java Serializer accepts the record Hive uses and translates it such that the OUTPUTFORMAT class can write it to the HDFS (or local file system). It is used at the time of HiveQL CREATE-TABLE-AS-SELECT statement. So the INPUTFORMAT, OUTPUTFORMAT and SerDe objects allow Hive to separate the table record format from the table file format.

Hive bundles a number of SerDes for us. We can also develop your own SerDes if you have a more unusual data type that you want to manage with a Hive table. Some of those are specified as below.

**LazySimpleSerDe:** The default SerDe that's used with the TEXTFILE format;

**ColumnarSerDe:** Used with the RCFILE format,

**RegexSerDe:** RegexSerDe can form a powerful approach for building structured data in Hive tables from unstructured blogs, semi-structured log files, e-mails, tweets, and other data from social media. Regular expressions allow us to extract meaningful information.

**HBaseSerDe:** Included with Hive to enables it to integrate with HBase.

**JSONSerDe:** A third-party SerDe for reading and writing JSON data records with Hive.

**AvroSerDe:** Included with Hive so that you can read and write Avro data in Hive tables.

The following example shows us all of the options we've been discussing in this section.

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS]
    [db_name.]table_name
... (Skipping some lines for brevity)
[ROW FORMAT row_format] [STORED AS file_format]
| STORED BY 'storage.handler.class.name' [WITH
SERDEPROPERTIES (...)] ]
... (Skipping some lines for brevity)
row_format
: DELIMITED [FIELDS TERMINATED BY char
[ESCAPED BY
char]] [COLLECTION ITEMS TERMINATED BY char]
[MAP KEYS TERMINATED BY char] [LINES
TERMINATED BY-char] [NULL DEFINED AS char]
| SERDE serde_name [WITH SERDEPROPERTIES
(property_name=property_value, property_
name=property_value, ...)]
file_format:
: SEQUENCEFILE | TEXTFILE | RCFILE | ORC
| INPUTFORMAT input_format_classname
OUTPUTFORMAT
output_format_classname
```

**Tying it all together with an example**

We want to tie things together in this section with two examples. In this first example, we revisit data_types_table from Listing. Here we leverage the DESCRIBE EXTENDED data_types_table HiveQL command to illustrate what Hive does with our CREATE TABLE statement under the hood.

```
hive> DESCRIBE EXTENDED data_types_table;
OK
our_tinyint tinyint 1 byte
signed integer
our_smallint smallint 2 byte
```

signed integer

...

(A) inputFormat :
org.apache.hadoop.mapred.TextInputFormat outputFormat:

(B) org.apache.hadoop.hive.q1.io.

HiveIgnoreKeyTextOutputFormat

...

serializationLib:

org.apache.hadoop.hive.serde2.1azy.LazySimpleSerDe,

(C) parameters={co1lection.delim=|, mapkey.delim=^, line delim= }

(D), serialization.format= „ field.delim=,}),

...

**Q.20** ***Explain how hive data manipulation language works?***

**Ans.** Hive's data manipulation language (DML) allows us to load and insert data into tables and create tables from other tables.

**LOAD DATA examples**

Now we have to place data into the data_types_table with LOAD DATA command. The syntax for the LOAD DATA command is given below.

"LOAD DATA [LOCAL] INPATH 'path to file' [OVERWRITE] INTO

TABLE 'table name' [PARTITION partition column = value1, partition column2 = value2, . . .]

In the above syntax optional LOCAL keyword tells Hive to copy data from the input file on the local file system into the Hive data warehouse directory. Without the LOCAL keyword, the data is simply moved (not copied) into the warehouse directory. The optional OVERWRITE keyword, causes the system to overwrite data in the specified table if it already has data stored in it. Finally, the optional PARTITION list tells Hive to partition the storage of the table into different directories in the data warehouse directory structure. This powerful concept improves query performance in Hive, by Rather than run a MapReduce job over the entire table to find the data you want to view or analyze, you can isolate a segment of the table and save a lot of system time with partitions. The following Listing shows the commands to use to load the data_types_table with data.

(A) $ cat data.txt

100, 32000, 2000000, 9200000000000000000, 0.15625,
4.9406564584, 124654,

1.23E+3,2013–09–21 20:19:52.025,true, test string,
\0xFFFFDDDDEEEEAAAA,1|2|3|4,key^1024,

1|3.1459|test struct,2|test union

(B) hive (data_types_db)> LOAD DATA LOCAL INPATH '/home/biadmin/Hive/data.txt' INTO TABLE

data_types_table;

Copying data from file:/home/biadmin/Hive/data.txt

Copying file: file:/home/biadmin/Hive/data.txt

Loading data to table data_types_db.data_types_table

Table data_types_db.data_types_table stats:

[num_partitions: 0, num_files: 1, num_rows: 0,

total_size: 185, raw_data_size: 0]

OK

Time taken: 0.287 seconds

(C) hive> SELECT * FROM data_types_table;

OK

100 32000 2000000 9200000000000000000 0.15625

4.940656458412465

1230 2013-09-21 20:19:52.025 true test string

\0xFFFFDDDDEEEEAAAA [1,2, 3,4] {"key" :1024}

{"first":1,"second":3.1459,"third":"test struct"}

(D) {2:"test union"}

Time taken: 0.201 seconds, Fetched: 1 row(s)

**Listing : Loading our_first_table with Data**

In the above listing Step (A) shows listing of data you intend to load. This data file has only one record in it, but there's a value for each field in the table. As we specified at table creation time, fields are separated by a comma; collections (such as STRUCT and UNIONTYPE) are separated by the vertical bar or pipe character (|); and the MAP keys and values are separated by the caret character (^).Step (B) has the LOAD DATA command, and in Step (C) we're retrieving the record we just loaded in Step (B) so that we can view the data.

**Example:**

In the below listing we created two identical tables, named FlightInfo2007 and FlightInfo2008, as you can see in steps (A) and (F) in below Listing

(A) CREATE TABLE IF NOT EXISTS FlightInfo2007 (Year SMALLINT, Month TINYINT, DayofMonth TINYINT,

DayofWeek TINYINT,

DepTime SMALLINT, CRSDepTime SMALLINT, ArrTime SMALLINT,

CRSArrTime SMALLINT,

UniqueCarrier STRING, FlightNum STRING, TailNum STRING,

ActualElapsedTime SMALLINT, CRSElapsedTime SMALLINT,

AirTime SMALLINT, ArrDelay SMALLINT, DepDelay SMALLINT,

Origin STRING, Dest STRING,Distance INT,

TaxiIn SMALLINT, TaxiOut SMALLINT, Cancelled SMALLINT,

CancellationCode STRING, Diverted SMALLINT,

CarrierDelay SMALLINT, WeatherDelay SMALLINT,

NASDelay SMALLINT, SecurityDelay SMALLINT,

LateAircraftDelay SMALLINT)

COMMENT 'Flight InfoTable'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE

TBLPROPERTIES ('creator'='Bruce Brown', 'created_at' = 'Thu Sep 19 10:58:00 EDT 2013');

(B) hive (flightdata)> LOAD DATA INPATH '/home/biadmin/ Hive/Data/2007.csv' INTO TABLE FlightInfo2007;

Loading data to table flightdata.flightinfo2007

Table flightdata.flightinfo2007 stats: [num_partitions: 0, num_files: 2, num_rows : 0, total_size: 1405756086, raw_data_size: 0]

OK

Time taken: 0.284 seconds;

(C) hive (flightdata)> SELECT * FROM FlightInfo2007 LIMIT 2;

OK

NULL NULL NULL NULL NULL NULL NULL

NULL UniqueCarrier FlightNum TailNum

NULL NULLNULL NULL NULL Origin

Dest NULL NULL NULL NULL

CancellationCode NULL NULL

NULLNULL NULL NULL

2007 1111232 1225 1341

1340 WN 2891 N351 69 75

54 1 7SMF ONT 389 4 ll

0 0 0 0 0

0 0

Time taken: 0.087 seconds, Fetched: 2 row(s)

(D) LOAD DATA INPATH '/home/biadmin/Hive/Data/2007.csv'

OVERWRITE INTO TABLE FlightInfo2007;

(E) hive (flightdata)> SELECT * FROM FlightInfo2007 LIMIT 2;

OK

2007 1111232 1225 1341

1340 WN 2891 N351 69 75

54 1 7SMF ONT 389 4 11

0 0 0 0 0

0 0

2007 1 1 1 1918 1905 2043

2035 WN 462 N370 85 90

SMF PDX 479 5

6 0 0 0 0

0 0 0

Time taken: 0.089 seconds, Fetched: 2 row(s)

(F) CREATE TABLE IF NOT EXISTS FlightInfo2008 LIKE FlightInfo2007;

(G) LOAD DATA INPATH '/home/biadmin/Hive/Data/2008.csv'INTO TABLE FlightInfo2008;

BDA.92

**Listing : Flight Information Tables from 2007 and 2008**

In Step (B) of above listing we didn't use the LOCAL keyword. That's because these files are large; you'll move the data into your Hive warehouse, not make another copy on your small and tired laptop disk. You'd likely want to do the same thing on a real cluster and not waste the storage. In Step (B) of above listing we use the LIMIT keyword because this table is huge. In Step (F), the LIKE keyword instructs Hive to copy the existing FlightInfo2007 table definition when creating the FlightInfo2008 table. In Step (G) you're using the same technique as in Step (B).

In the above Listing, Hive could not (at first) match the first record with the data types you specified in your CREATE TABLE statement. So the system showed NULL values in place of the real data, and the command completed successfully. This behavior illustrates that Hive uses a Schema on Read verification approach as opposed to the Schema on Write verification approach, which you find in RDBMS technologies. This is one reason why Hive is so powerful for big data analytics.

**INSERT examples**

Another Hive DML command to explore is the INSERT command. To demonstrate this new DML command, we have you create a new table that will hold a subset of the data in the FlightInfo2008 table. We basically have three INSERT variants. Two of those are specified in below listing.

```
(A) CREATE TABLE IF NOT EXISTS myFlightInfo
(Year SMALLINT, DontQueryMonth TINYINT,
DayofMonth
    TINYINT, DayofWeek TINYINT,
    DepTime SMALLINT, ArrTime SMALLINT,
    UniqueCarrier STRING, FlightNum STRING,
    AirTime SMALLINT, ArrDelay SMALLINT,
DepDelay SMALLINT,
    Origin STRING, Dest STRING, Cancelled
SMALLINT,
    CancellationCode STRING)
    COMMENT 'FlightInfoTable'
    PARTITIONED BY(Month TINYINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
(B) STORED AS RCFILE
    TBLPROPERTIES ('creator'='Bruce Brown',
'created_at'='MonSep 2 14:24:19 EDT 2013');
(C) INSERT OVERWRITE TABLE myflightinfo
    PARTITION (Month=1)
    SELECT Year, Month, DayofMonth, DayofWeek,
DepTime,
    ArrTime, UniqueCarrier,
    FlightNum, AirTime, ArrDelay, DepDelay, Origin,
    Dest, Cancelled,
    CancellationCode
    FROM FlightInfo2008 WHERE Month=1;
(D) FROM FlightInfo2008
    INSERT INTO TABLE myflightinfo
    PARTITION (Month=2)
    SELECT Year, Month, DayofMonth, DayofWeek,
DepTime,
    ArrTime, UniqueCarrier, FlightNum,
    AirTime, ArrDelay, DepDelay, Origin, Dest, Cancelled,
    CancellationCode WHERE Month=2
    ... (Months 3 through 11 skipped for brevity)
INSERT INTO TABLE myflightinfo
PARTITION (Month=12)
SELECT Year, Month, DayofMonth, DayofWeek, DepTime,
    ArrTime, UniqueCarrier, FlightNum,
    AirTime, ArrDelay, DepDelay, Origin, Dest, Cancelled,
    CancellationCode WHERE Month=12;
(E) hive (flightdata)> SHOW PARTITIONS myflightinfo;
    OK
    month=1
    month=10
    month=11
    month=12
(F) $ ls
    /home/biadmin/Hive/warehouse/flightdata.db/
myflightinfo
    month=1 month=11 month=2 month=4 month=6
month=8 month=10 month=12 month=3 month=5 month=7
month=9
```

(G) $HIVE_HOME/bin/hive --service rcfilecat /home/biadmin/Hive/warehouse/flightdata.db/myflightinfo/month=12/000000__0

...

2008 12 13 6 655 856 DL
1638 85 0 -5 PBI ATL
0

2008 12 13 6 1251 1446 DL
1639 89 9 11 IAD ATL
0

2008 12 13 6 1110 1413 DL
1641 104 -5 7 SAT ATL
0

**Listing : Partitioned Version of 2008 Flight Information Table**

In the above listing in Step (A), we create this new table and in Step(B), we specify that the file format will be row columnar instead of text. This format is more compact than text and often performs better, depending on our access patterns. (If you're accessing a small subset of columns instead of entire rows, try the RCFILE format.) . In step(C) we use the INSERT OVERWRITE command to insert data via a SELECT statement from the FlightInfo2008 table.

Note that we're partitioning our data using the PARTITION keyword based on the Month field. After we're finished, we'll have 12 table partitions, or actual directories, under the warehouse directory in the file system on our virtual machine, corresponding to the 12 months of the year. As we explain earlier, partitioning can dramatically improve our query performance if we want to query data in the myFlightInfo table for only a certain month. We can see the results of the PARTITION approach with the SHOW PARTITIONS command in Steps (E) and (F). Notice in Step (D) that we're using a variant of the INSERT command to insert data into multiple partitions at one time. We have only shown month 2 and 12 for brevity but months 3 through 11 would have the same syntax.

You can also use this FROM table 1 INSERT INTO table2 SELECT(-----) format to insert into multiple tables at a time. We have you use INSERT instead of OVERWRITE here to show the option of inserting instead of overwriting. Hive allows only appends, not inserts, into tables, so the INSERT keyword simply instructs Hive to append the data to the table. Finally, note in Step (G) that we have to use a special Hive command service (rcfilecat) to view this table in your warehouse, because the RCFILE format is a binary format, unlike the previous TEXTFILE format examples. Third one is the Dynamic Partition Inserts variant. In below Listing, you partition the myFlightInfo table into 12 segments, 1 per month. If you had hundreds of partitions, this task would have become quite difficult, and it would have required scripting to get the job done. Instead, Hive supports a technique for dynamically creating partitions with the INSERT OVERWRITE statement.

**Create Table As Select (CTAS) examples :** The powerful technique in Hive known as Create Table As Select. or CTAS. Its constructs allow us to quickly derive Hive tables from other tables as we build powerful schemas for big data analysis. The following Listing shows you how CTAS works.

**(A)** hive> CREATE TABLE myflightinfo2007 AS

> SELECT Year, Month, DepTime, ArrTime, FlightNum,

Origin, Dest FROM FlightInfo2007

> WHERE (Month = 7 AND DayofMonth = 3) AND

(Origin='JFK' AND Dest='ORD');

**(B)** hive> SELECT * FROM myFlightInfo2007;

OK

| | | | | | | |
|---|---|---|---|---|---|---|
| 2007 | 7 | 700 | 834 | 5447 | JFK | ORD |
| 2007 | 7 | 1633 | 1812 | 5469 | JFK | ORD |
| 2007 | 7 | 1905 | 2100 | 5492 | JFK | ORD |
| 2007 | 7 | 1453 | 1624 | 4133 | JFK | ORD |
| 2007 | 7 | 1810 | 1956 | 4392 | JFK | ORD |
| 2007 | 7 | 643 | 759 | 903 | JFK | ORD |
| 2007 | 7 | 939 | 1108 | 907 | JFK | ORD |
| 2007 | 7 | 1313 | 1436 | 915 | JFK | ORD |
| 2007 | 7 | 1617 | 1755 | 917 | JFK | ORD |
| 2007 | 7 | 2002 | 2139 | 919 | JFK | ORD |

Time taken: 0.089 seconds, Fetched: 10 row(s)

hive> CREATE TABLE myFlightInfo2008 AS

> SELECT Year, Month, DepTime, ArrTime, FlightNum,

Origin, Dest FROM FlightInfo2008

> WHERE (Month = 7 AND DayofMonth = 3) AND

(Origin='JFK' AND Dest='ORD')

hive> SELECT * FROM myFlightInfo2008;

OK

| | | | | | | |
|---|---|---|---|---|---|---|
| 2008 | 7 | 930 | 1103 | 5199 | JFK | ORD |
| 2008 | 7 | 705 | 849 | 5687 | JFK | ORD |
| 2008 | 7 | 1645 | 1914 | 5469 | JFK | ORD |
| 2008 | 7 | 1345 | 1514 | 4392 | JFK | ORD |
| 2008 | 7 | 1718 | 1907 | 1217 | JFK | ORD |
| 2008 | 7 | 757 | 929 | 1323 | JFK | ORD |
| 2008 | 7 | 928 | 1057 | 907 | JFK | ORD |
| 2008 | 7 | 1358 | 1532 | 915 | JFK | ORD |
| 2008 | 7 | 1646 | 1846 | 917 | JFK | ORD |
| 2008 | 7 | 2129 | 2341 | 919 | JFK | ORD |

Time taken: 0.186 seconds, Fetched: 10 row(s)

**Listing: An Example of Using CREATE TABLE . . . AS SELECT**

In Step A, we build two smaller tables derived from the FlightInfo2007 and FlightInfo2008 by selecting a subset of fields from the larger tables for a particular day (in this case, July 3), where the origin of the flight is New York's JFK airport (JFK) and the destination is Chicago's O'Hare airport (ORD). Then in Step B we simply dump the contents of these small tables so that you can view the data.

**Q.21** ***Explain querying and analyzing the data.***

**Ans. Joining tables with Hive :** Well, remember that the underlying operating system for Hive is (surprise!) Apache Hadoop: MapReduce is the engine for joining tables, and the Hadoop File System (HDFS) is the underlying storage. Disk and network access is a lot slower than memory access, so minimize HDFS reads and writes as much as possible. Hive table reads and writes via HDFS usually involve very large blocks of data, the more data you can manage altogether in one table, the better the overall performance.

Now we show you a Hive join example using our flight data tables. The above Listing shows you how to create and display a myflightinfo2007 table and a myflightinfo2008 table from the larger FlightInfo2007 and FlightInfo2008 tables. The plan all along was to use the CTAS created myflightinfo2007 and myllightinfo2008 tables to illustrate how you can perform joins in Hive. The plan all along was to use the CTAS created myflightinfo2007 and myflightinfo2008 tables to illustrate how you can perform joins in Hive. Figure 1 shows the result of an inner join with the myflightinfo2007 and myflightinfo2008 tables using the SQuirreL SQL client.

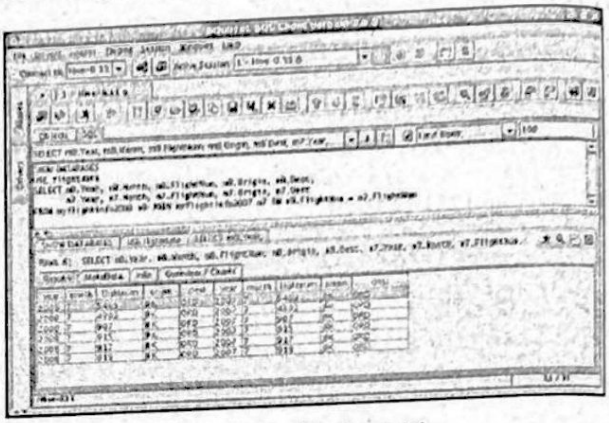

*Fig. 1 : The Hive inner join*

Hive supports equi-joins, a specific type of join that only uses equality comparisons in the join predicate. Other comparators such as Less Than (<) are not supported. This restriction is only because of limitations on the underlying MapReduce engine. Also, you cannot use OR in the ON clause.

Figure 2 illustrates how an inner join works using a Venn diagram technique. The basic idea here is that an inner join returns the records that match between two tables. So an inner join is a perfect analysis tool to determine which flights are the same from JFK (New York) to ORD (Chicago) in July of 2007 and July of 2008.

**Hive Join Examples**

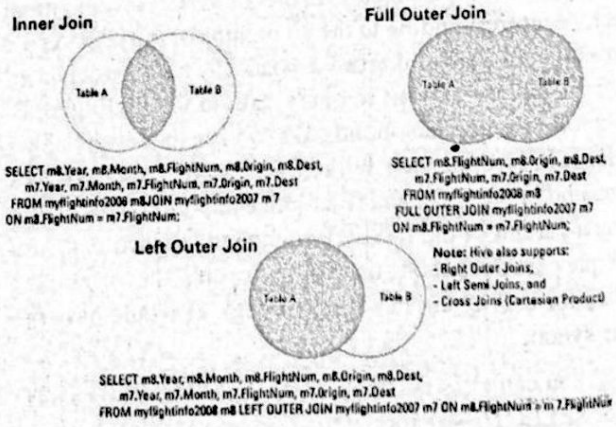


*Fig. 2 : Hive inner join, full outer join, and left outer join*

**Improving your Hive queries with indexes :** Creating an index is common practice with relational databases when

we want to speed access to a column or set of columns in your database. Without an index, the database system has to read all rows in the table to find the data we have selected. Indexes become even more essential when the tables grow extremely large. Hive supports index creation on tables. In below Listing, we list the steps necessary to index the FlightInfo2008 table.

(A) CREATE INDEX f08_index ON TABLE flightinfo2008

(Origin) AS 'COMPACT' WITH DEFERRED REBUILD;

(B) ALTER INDEX f08_INDEX ON flightinfo2008 REBUILD;

(C) hive (flightdata)> SHOW INDEXES ON FlightInfo2008;

OK

f08index flightinfo2008 origin

flightdata__flightinfo2008_f08index__ compact

Time taken: 0.079 seconds, Fetched: 1 row(s)

(D) hive (flightdata)> DESCRIBE

flightdata__flightinfo2008_f08index__;

OK

origin string None

_bucketname string

_offsets array<bigint>

Time taken: 0.112 seconds, Fetched: 3 row(s)

(E) hive (flightdata)> SELECT Origin, COUNT(l) FROM flightinfo2008 WHERE Origin = 'SYR' GROUP BY Origin;

SYR 12032

Time taken: 17.34 seconds, Fetched: 1 row(s)

(F) hive (flightdata)> SELECT Origin, SIZE('_offsets') FROM flightdata__flightinfo2008_f08index__ WHERE origin = 'SYR';

SYR 12032

Time taken: 8.347 seconds, Fetched: 1 row(s)

(G) hive (flightdata)> DESCRIBE

flightdata__flightinfo2008_f08index__;

OK

origin string None

_bucketname string

_offsets array<bigint>

Time taken: 0.12 seconds, Fetched: 3 row(s)

**Listing : Creating an Index on the Flightinfo2008 Table**

Step (A) creates the index using the 'COMPACT index handler on the Origin column. Hive also offers a bitmap index handler as of the 0.8 release, which is intended for creating indexes on columns with a few unique values. The keywords WITH DEFERRED REBUILD instructs Hive to first create an empty index; Step (B) is where we actually build the index with the ALTER INDEX REBUILD command. Deferred index builds can be very useful in workflows where one process creates the tables and indexes, another loads the data and builds the indexes and a final process performs data analysis. Hive doesn't provide automatic index maintenance, so you need to rebuild the index if you overwrite or append data to the table. Also, Hive indexes support table partitions, so a rebuild can be limited to a partition. Step (C) illustrates how we can list or show the indexes created against a particular table. Step (D) illustrates an important point regarding

**Hive Indexes :** Hive indexes are implemented as tables. This is why we need to first create the index table and then build it to populate the table. Therefore, we can use indexes in at least two ways:

- Count on the system to automatically use indexes that you create.
- Rewrite some queries to leverage the new index table

In Step (E) we write a query that seeks to determine how many flights left the Syracuse airport during 2008.

To get this information, we leverage the COUNT aggregate function. In Step (F), you leverage the new index table and use the SIZE function instead.

**Windowing in HiveQL :** The concept of windowing, introduced in the SQL:2003 standard, allows the SQL programmer to create a frame from the data against which aggregate and other window functions can operate. HiveQL now supports windowing per the SQL standard. One question we had when we first discovered this data set was, "What exactly is the average flight delay per day?" So we created a query in below Listing that produces the average departure delay per day in 2008.

**(A)**
```
hive (flightdata)> CREATE VIEW avgdepdelay AS
> SELECT DayOfWeek, AVG(DepDelay) FROM
FlightInfo2008 GROUP BY DayOfWeek;
OK
Time taken: 0.121 seconds
```

**(B)**
```
hive (flightdata)> SELECT * FROM avgdepdelay;
...
OK
1.  10.269990244459473
2.  8.97689712068735
3.  8.289761053658728
4.  9.772897177836702
5.  12.158036387869656
6.  8.645680904903614
7.  11.568973392595312
Time taken: 18.6 seconds, Fetched: 7 row(s)
```

**Listing : Finding the Average Departure Delay per Day in 2008**

As shown in step(A) of above Listing Hive's Data Definition Language (DDL) also includes the CREATE VIEW statement, which can be quite useful. In Hive, views allow a query to be saved but data is not stored as with the Create Table as Select (CTAS) statement.

Suppose if you want to know "What is the first flight between Airport X and Y?" Suppose that in addition to this information, you want to know about subsequent flights. just in case you're not a "morning person." Write the query as below.

**(A)**
```
hive (flightdata)> SELECT f08.Month,
f08.DayOfMonth
    cr.description, f08.0rigin, f08.Dest,
    f08.FlightNum, f08.DepTime, MIN(f08.DepTime)
    OVER (PARTITION BY f08.DayOfMonth ORDER
BY f08.DepTime)
    FROM flightinfo2008 f08 JOIN Carriers cr ON
    f08.UniqueCarrier = cr.code
    WHERE f08.origin = 'JFK' AND f08.Dest = 'ORD'
AND
    f08.Month = 1 AND f08.DepTime != 0
    ...
    OK
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | JetBlue Airways | JFK | ORD | 903 | 641 | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1323 | 833 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 907 | 929 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5083 | 945 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5634 | 1215 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 915 | 1352 | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1323 | 833 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 907 | 929 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5083 | 945 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5634 | 1215 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 915 | 1352 | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1815 | 1610 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 917 | 1735 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5469 | 1749 | 641 |
| 1 | 1 | Comair Inc. | JFK | ORD | 5492 | 2000 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 919 | 2102 | 641 |
| 1 | 1 | JetBlue Airways | JFK | ORD | 919 | 48 | 48 |

**Listing : Using Aggregate Window Functions on the Flight Data**

In Step (A), we've replaced the GROUP BY clause with the OVER clause where we specify the PARTITION or window over which we want the MIN aggregate function to operate. We've also included the ORDER BY clause so that we can see those subsequent flights after the first one.

❑❑❑