Unit 6

Applying Structure to Hadoop Data with Hive:

Saying Hello to Hive, Seeing How the Hive is Put Together, Getting Started with Apache Hive, Examining the Hive Clients, Working with Hive Data Types, Creating and Managing Databases and Tables, Seeing How the Hive Data Manipulation Language Works, Querying and Analysing Data

- Java MapReduce programs and the Hadoop Distributed File System (HDFS) provide you with a powerful distributed computing framework, but they relying on them limits the use of Hadoop to Java programmers who can think in Map and Reduce terms when writing programs.

- Apache Hive was created at Facebook by a team of engineers who were led by Jeff Hammerbacher. Hive, a top-level Apache project and a vital component within the Apache Hadoop ecosystem, drives several leading big-data use cases and has brought Hadoop into data centres across the globe.

Saying Hello to Hive (Introduction to hive)

- 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.
- 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.

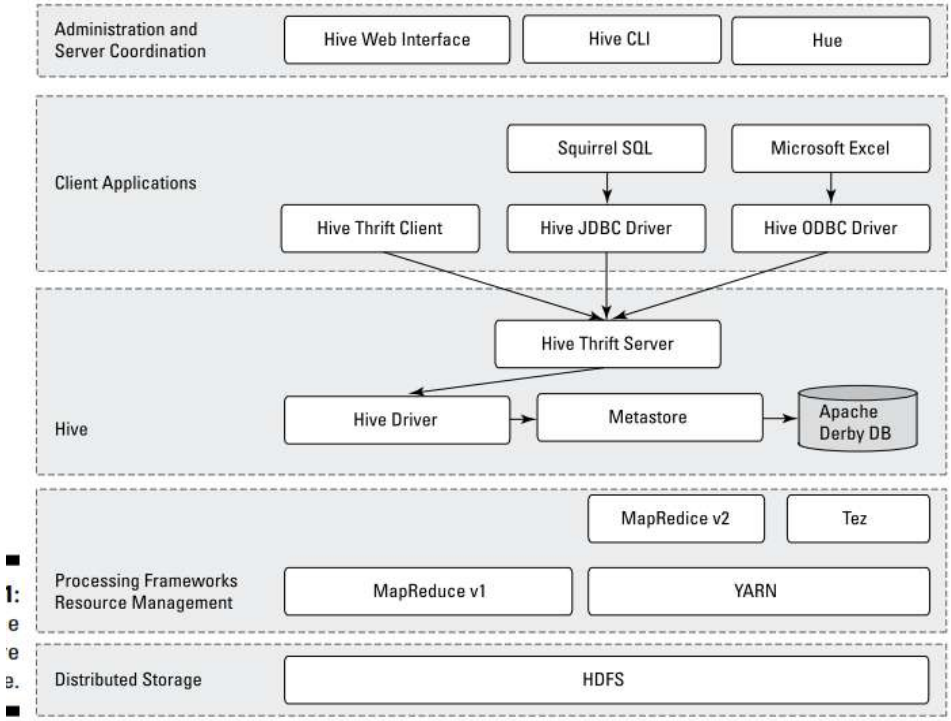
- Closely associated with RDBMS/EDW technology is extract, transform, and

load (ETL) technology.

- ETL works as follows — it must first be extracted from its native source, transformed into an appropriate format, and then loaded into the RDBMS or EDW
- Hive is a powerful ETL tool in its own right, along with the major player in this realm: Apache Pig.
- 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.

Seeing How the Hive is Put Together(Architecture):

- The architecture of Apache Hive and explain its various components, as shown in the illustration in Figure 13-1.



- As shown in Figure 13-1, you can see at the bottom that Hive sits on top of

the Hadoop Distributed File System (HDFS) and MapReduce systems.

- In the case of MapReduce Hive queries are converted to MapReduce code and executed using the MapReduce v1 (MRv1) infrastructure, like the JobTracker and TaskTracker. With Hadoop 2, YARN has decoupled resource management and scheduling from the MapReduce framework.
- There is a new framework under development called Apache Tez, which is designed to improve Hive performance for batch-style queries and support smaller interactive (also known as real-time) queries
- Moving up the diagram, you find the Hive Driver, which compiles, optimizes, and executes the HiveQL. The Hive Driver may choose to execute HiveQL statements and commands locally or spawn a MapReduce job, depending on the task at hand.
- By default, Hive includes the Apache Derby RDBMS configured with the metastore in what's called embedded mode.
- Embedded mode means that the Hive Driver, the metastore, and Apache Derby are all running in one Java Virtual Machine (JVM).
- Two other modes exist — local and remote — which can better support multiple Hive sessions in production environments. The key to application support is the Hive Thrift Server (see Figure 13-1), which enables a rich set of clients to access the Hive subsystem, Apache Thrift clients connect to Hive via the Hive Thrift Server, just as the JDBC and ODBC clients do.
- In the Hive architecture drawing in Figure 13-1, note that Hive includes a Command Line Interface (CLI), where you can use a Linux terminal window to issue queries and administrative commands directly to the Hive Driver. There is another web browser technology known as Hue that provides a graphical user interface (GUI) to Apache Hive.

Getting Started with Apache Hive(installation)

The setup steps run something like this:

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

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

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 13-1 (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 13-2 (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 define specific Hive configuration settings.

The Apache Hive distribution includes a template configuration file that provides all default settings for Hive. To customize Hive for your environment, all you need to do is copy the template file to the file named `hive-site.xml` and edit it. Listing 13-3 shows the steps to accomplish this task.

Listing 13-1: 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 13-2: 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
```

Listing 13-3: Setting Up the hive-site.xml File

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


<?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>
```

Examining the Hive Clients

In general, we have the following hive clients:

- Hive command-line interface (CLI),
- Hive Web Interface (HWI) Server
- SQuirreL

The Hive CLI client

In following figure components that are required when running the CLI.

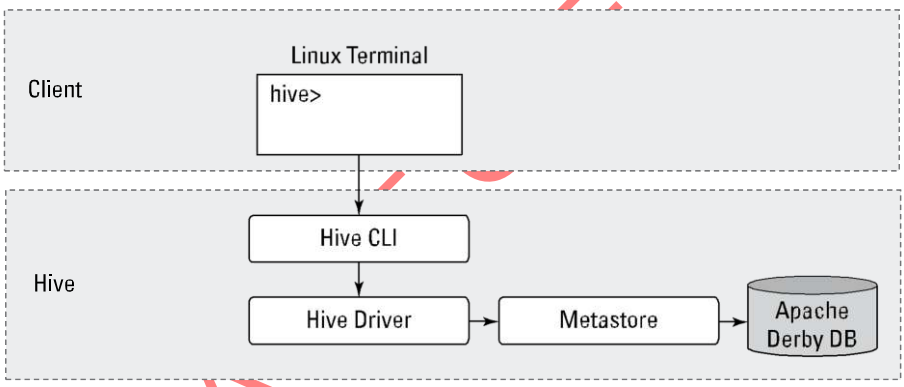


Figure 13-2: The Hive command-line interface mode.

- To run the Hive CLI, you execute the `hive` command and specify the CLI as the service you want to run.
- In Listing 13-4, you can see the command that's required as well as some of our first HiveQL statements.

Listing 13-4: Using the Hive CLI to Create a Table

```
(A) $ $HIVE_HOME/bin hive --service cli
(B) hive> set hive.cli.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_table
          (
                        > FirstName       STRING,
                        > LastName        STRING,
                        > EmployeeId      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 13-4 (see Step A) starts the Hive CLI using the `$HIVE_HOME` environment variable .
- The `–service cli` command-line option directs the Hive system to start the command-line interface, though you could have chosen other servers.
- Next, in Step B, you tell the Hive CLI to print your current working database so that you know where you are in the namespaceContinuing in Listing 13-4, in Step C you use HiveQL's data definition language (DDL) to create your first database.

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 13-3 illustrates how this type of Hive client configuration might work.

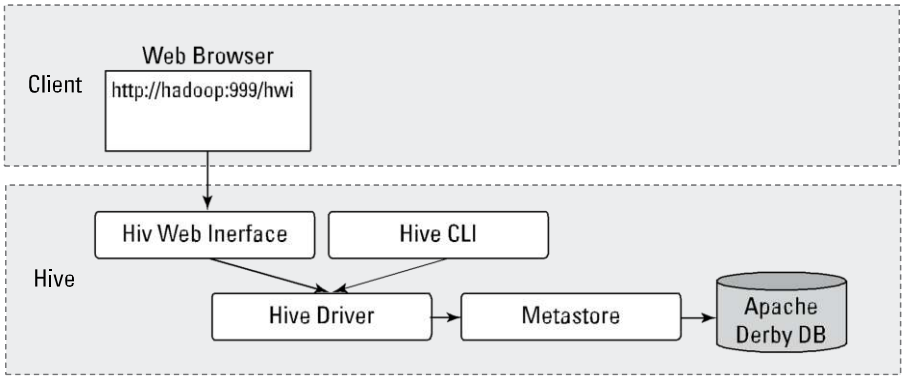


Figure 13-3: The Hive Web Interface client configuration.

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

1. **Using the commands in Listing 13-5 (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
```

Listing 13-5: Configuring the $HIVE_HOME/conf/hive-site.xml file

```
<property>
   <name>hive.hwi.war.file</name>
   <value>${HIVE_HOME}/lib/hive_hwi.war</value>
   <description>This is the WAR file with the
 jsp
   content for Hive Web Interface</description>
</property>
```

- With the HWI Server now running, you simply enter the URL `http://localhost:9999/hwi/` into your web browser and view the metadata for `our_first_table`

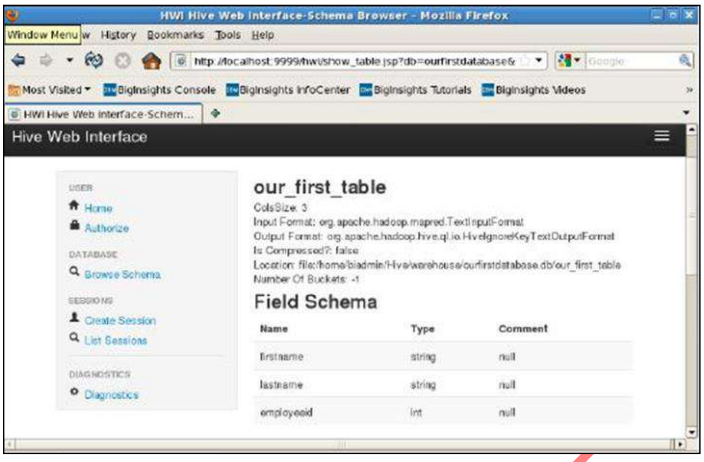


Figure 13-4: Using the Hive Web Interface to browse the metadata.

SQuirreL as Hive client with the JDBC Driver

- SQuirreL SQL is the open source tool. we 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 analysing data with Apache Hive.
-
- Figure 13-5 illustrates how the Hive architecture would work when using tools such as SQuirreL.

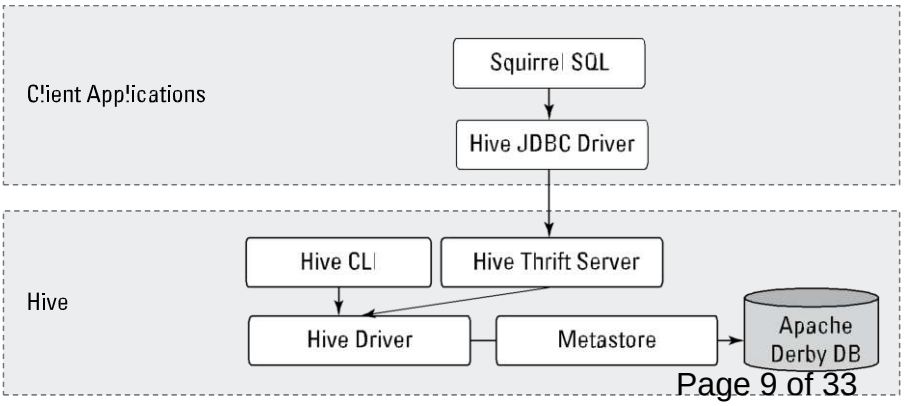


Figure 13-5: Using the SQuirreL client with Apache Hive.

Follow these steps to get SQuirreL running:

1. **Start the Hive Thrift Server using the command in Listing 13-6 (following this 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-standard.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
```

5. **Follow the directions for running SQuirreL with Apache Hive at**

```
https://cwiki.apache.org/confluence/display/Hive/
HiveJDBCInterface - HiveJDBCInterface-
IntegrationwithSQuirrelSQLClient
```

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 13-6: 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
```

- Figure 13-6 shows some HiveQL commands running against the Hive Driver.

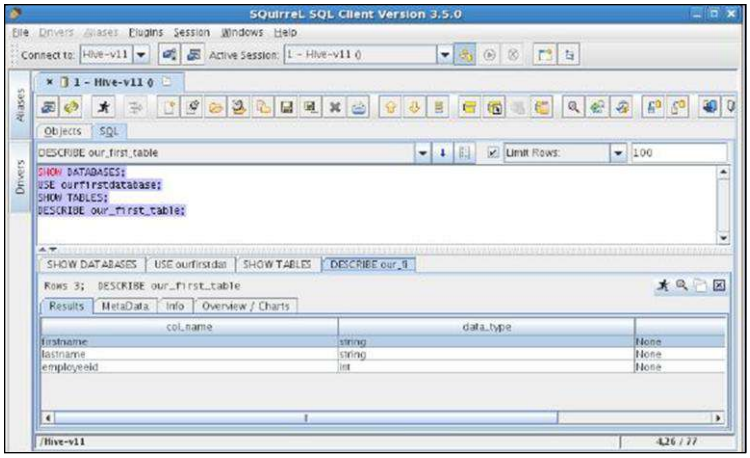


Figure 13-6: Using the SQuirreL SQL client to run HiveQL commands.

The Apache Hive 0.11 release also includes a new Hive Thrift Server called HiveServer2. When configured correctly, HiveServer2 can support multiple clients

Working with Hive Data Types

Listing 13-7 goes to the trouble of creating a table that uses all Hive-supported data types and the amount of memory required.

Listing 13-7: HiveQL-Supported Data Types

```
$ ./hive --service cli
hive> CREATE DATABASE data_types_db;
OK
Time taken: 0.119 seconds
hive> USE data_types_db;
OK
Time taken: 0.018 seconds
(1)Hive>  CREATE TABLE data_types_table (
(2) > our_tinyint   TINYINT    COMMENT '1 byte signed integer',
(3) > our_smallint  SMALLINT   COMMENT '2 byte signed integer',
(4) > our_int       INT        COMMENT '4 byte signed integer',
(5) > our_bigint    BIGINT     COMMENT '8 byte signed integer',
(6) > our_float     FLOAT      COMMENT 'Single precision floating point',
(7) > our_double    DOUBLE     COMMENT 'Double precision floating point',
(8) > our_decimal   DECIMAL    COMMENT 'Precise decimal type based
(9) >                                  on Java BigDecimal Object',
(10) > our_timestamp TIMESTAMP  COMMENT 'YYYY-MM-DD HH:MM:SS.fffffffff"
(11) >                                  (9 decimal place precision)',
(12) > our_boolean     BOOLEAN    COMMENT 'TRUE or FALSE boolean data type',
(13) > our_string    STRING     COMMENT 'Character String data type',
(14) > our_binary    BINARY     COMMENT 'Data Type for Storing arbitrary
(15) >                                   number of bytes',
(16) > our_array     ARRAY<TINYINT>  COMMENT 'A collection of fields all of
(17) >                                   the same data type indexed BY
(18) >                                   an integer',
(19) > our_map     MAP<STRING,INT> COMMENT 'A Collection of Key,Value Pairs
(20) >                                   where the Key is a Primitive
(21) >                                   Type and the Value can be
(22) >                                   anything.  The chosen data
(23) >                                   types for the keys and values
(24) >                                   must remain the same per map',
(25) > our_struct  STRUCT<first : SMALLINT, second : FLOAT, third : STRING>
(26) >                               COMMENT 'A nested complex data
(27) >                                   structure',
(28) > our_union    UNIONTYPE<INT,FLOAT,STRING>
(29) >                               COMMENT 'A Complex Data Type that can
(30) >                                   hold One of its Possible Data
(31) >                                   Types at Once')
(32) > COMMENT 'Table illustrating all Apache Hive data types'
(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');
OK
Time taken: 0.886 seconds
```

Creating and Managing Databases and Tables

Managing Hive databases

The following listing shows more features used to create a table in HIVE; Listing 13-8 illustrates a few of them.The general operations are create,use,alter,describe and Drop a database.

Listing 13-8: Creating, Dropping, and Altering Databases in Apache Hive

```
(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

(6) hive (ourfirstdatabase)> DROP DATABASE ourfirstdatabase CASCADE;
           OK
           Time taken: 0.132 seconds
```

Creating and managing tables with Hive

Defining table file formats

The following listing shows an example to create a table .

Listing 13-9: Defining the Hive Row Format for the `TEXTFILE` File Format

```
(1)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');
```

- Lines 33–37 define the Hive *row format* for your `data_types_table` and provide specifics on how fields will be separated or delimited whenever you insert or load data into the table.
- Line 38 defines the Hive *file format* — a text file — when the data is stored in the HDFS . You may be wondering why `our_first_table` lacks these extra keywords and delimiters.

Listing 13-10: Hive Table Default Row and File Format

```
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 list describes the file formats you can choose from as of Hive version 0.11.

- ✓ **TEXTFILE:** The default file format for Hive records. Alphanumeric characters from the Unicode standard (see `www.unicode.org`) 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:** Stores records in a column-oriented fashion rather than a row-oriented fashion — like the `TEXTFILE` format approach. Using the `RCFILE` format makes sense when tables have a large number of columns, but only a few columns are typically accessed. (`RCFILE` stands for *r*ecord *c*olumnar *file*.)
- ✓ **ORC:** A format (new as of Hive 0.11) that has significant optimizations to improve Hive reads and writes and the processing of tablesFor example, `ORC` files include optimizations for Hive complex types and new types such as `DECIMAL`.

✓ **INPUTFORMAT, OUTPUTFORMAT**`OUTPUTFORMAT` does the same thing for writing data to the Hive table. The keywords in the earlier table entries (`TEXTFILE`, for example) provide shortened syntax so that you don't have to specify both `INPUTFORMAT` and `OUTPUTFORMAT` for every `CREATE TABLE` statement

Defining table record formats

The Java technology that Hive uses to process records and map them to column data types in Hive tables (like you defined in Listing 13-7) is called *SerDe*, which is short for *Ser*ializer*De*serializer. Figure 13-7 illustrates how SerDes are leveraged and it will help you understand how Hive keeps file for- mats separate from record formats.

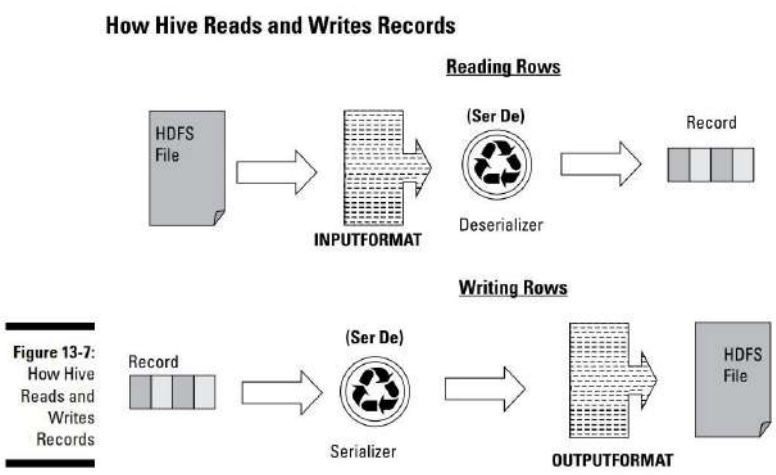


Figure 13-7: How Hive Reads and Writes Records

- Hive allows you to separate your record format from your file format .This can be established either replace `STORED AS TEXTFILE` with something like `STORED AS RCFILE`.
- 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**.
- This would characterize the data flow for a 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).
- This would characterize the data flow for a HiveQL `CREATE-TABLE-AS-SELECT` statement.
- 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 you to choose from, and you'll find a larger number available from third parties if you search online. You can also develop your own SerDes if you have a more unusual data type that you want to manage with a Hive table.
- In the list below, we describe some of the SerDes provided with Hive as well as one third-party option that you may find useful.

✓ **LazySimpleSerDe:** The default SerDe that's used with the `TEXTFILE` format; it would be used with `our_first_table` from Listing 13-4 and with `data_types_table` from Listing 13-7.

✓ **ColumnarSerDe:** Used with the `RCFILE` format.

✓ **RegexSerDe:** The regular expression SerDe, which ships with Hive to enable the parsing of text files, 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 you to extract meaningful information (an e-mail address, for example) with HiveQL from an unstructured or semi-structured text document incompatible with traditional SQL and RDBMSs.

✓ **HBaseSerDe:** Included with Hive to enables it to integrate with HBase. You can store Hive tables in HBase by leveraging this SerDe.

✓ **JSONSerDe:** A third-party SerDe for reading and writing JSON data records with Hive. We quickly found (via Google and GitHub) two JSON SerDes by searching online for the phrase *json serde for hive*.

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

- Reviewing the Language Manual DDL can be very helpful before you start creating your tables. The following listing shows (in bold print) all of the options to create a table.

```
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

- 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.ql.io.
          HiveIgnoreKeyTextOutputFormat
, ...
serializationLib:
  org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe,
(C)parameters:{collection.delim=|, mapkey.delim=^, line.
          delim=
(D), serialization.format= ,, field.delim=,}),
...
```

An example of how to use the HBase SerDe

- In this example, we want to show you how to specify a SerDe instead of letting Hive pick a default SerDe .
- Hive includes an HBase SerDe, whichis great news if you want to put a HiveQL front end on your HBase table.
- In the example in Listing 13-11, you create an `EXTERNAL` Hive table that connects with an HBase table
- Listing 13-11 shows the schema and contents of the HBase table that you connect to with Hive using the HBase SerDe.

Listing 13-11: Customer Information HBase Table

```
ROW    COLUMN+CELL
00001  column=ContactInfo:EA,  value=John.Smith@xyz.com
00001  column=ContactInfo:SA,  value=1 Hadoop Lane, NY
           11111
00001  column=CustomerName:FN, value=John
00001  column=CustomerName:LN, value=Smith
00001  column=CustomerName:MN, value=Timothy
00002  column=ContactInfo:EA,  value=Jane.Doe@xyz.com
00002  column=ContactInfo:SA,  value=7 HBase Ave, CA 22222
00002  column=CustomerName:FN, value=Jane
00002  column=CustomerName:LN, value=Doe
00002  column=CustomerName:MN, value=A
```

- The Customer Information HBase table consists of two rows and two column families: `ContactInfo` and `CustomerName`. The `ContactInfo` column family has two columns storing the customer's e-mail address (`EA`) and street address (`SA`).
- The `CustomerName` column family has three rows storing the first name (`FN`), middle name (`MN`) and last name (`LN`) of the customer.
- In Listing 13-12, you see the HiveQL statements you need in order to create a table that connects to your HBase table (refer to Listing 13-11) using map data types.

Listing 13-12: Creating an External Hive Table to Connect to the HBase Customer Information Table

```
(A) CREATE EXTERNAL TABLE hive_hbase_table (
key     INT,
name    map<STRING,STRING>,
info    map<STRING,STRING>)
STORED BY 'org.apache.hadoop.hive.hbase.
          HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
          ":key,CustomerName:,ContactInfo:")
TBLPROPERTIES ("hbase.table.name" = "customerinfo");

(B) hive> SELECT * FROM hive_hbase_table;
OK
1       {"FN":"John","LN":"Smith","MN":"Timothy"}
        {"EA":"John.Smith@xyz.com","SA":"1 Hadoop Lane, NY
          11111"}
2       {"FN":"Jane","LN":"Doe","MN":"A"}
        {"EA":"Jane.Doe@xyz.com","SA":"6 Novice HBase Ave,
          CA 22222"}
Time taken: 1.422 seconds
(C) hive> SELECT info["EA"] FROM hive_hbase_table WHERE
          name["FN"] = "Jane" AND name["LN"] = "Doe";
Total MapReduce jobs = 1
...
OK
Jane.Doe@xyz.com
```

In Step (A), you create an external table with a Key field to link up with the HBase row keys (`00001` and `00002` from Listing 13-11), and two map data types (`name` and `info`) to link up with the two column families (`ContactInfo` and `CustomerName`). Note the syntax for providing this linkage via the `WITH SERDEPROPERTIES` keywords. This SerDe configuration technique is quite common in Hive DDL. Note as well that the `TBLPROPERTIES` keyword is crucial for connecting the new external `hive_hbase_table` with the actual `customerinfo` HBase table name.

Step (B) shows how the key value pairs in HBase ({“FN”,”John”}, for example) are now available for querying with the help of the HiveQL. Note the syntax for accessing the Hive map data type in Step (C). You can select the value of the `info` map type using the notation `info ["EA"]` where `"EA"` is the key.

Seeing How the Hive Data Manipulation Language Works

- Hive's data manipulation language (DML) — it lets you load and insert data into tables and create tables from other tables.

LOAD DATA examples

The syntax for the LOAD `DATA` command is shown in Listing 13-13.

Listing 13-13: LOAD DATA Command Syntax

```
"LOAD DATA [LOCAL] INPATH 'path to file' [OVERWRITE] INTO
          TABLE 'table name' [PARTITION partition column1
          = value1, partition column2 = value2,...]
```

- First, the optional `LOCAL` keyword tells Hive to copy data from the input file on the local file system into the Hive data warehouse directory.
- When running in distributed mode, if you omit the `LOCAL` keyword Hive assumes your data is already in the HDFS,
- Second, the optional `OVERWRITE` key- word, as you might imagine, 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
- When you think about the magnitude of data that can be managed by Hivein the HDFS, partitioning makes a lot of sense. 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.
- Apache Hive uses the MapReduce technology within Hadoop to query and analyze tables .we can set the configuration variable `hive.exec.mode.local.auto` in the `hive-site.xml file`.
- When the variable is set to `true`, Hive tries to execute queries on small data sets locally without MapReduce whenever possible, to speed execution.

Listing 13-14 shows the commands to use to load the `data_types_table` with data. Again, we've annotated the listing so that we can discuss each step.

Listing 13-14: Loading `our_first_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)
```

- Step (A) is a listing (using the Unix `cat` command) 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.
- Note that type delimiters can be a comma or 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) you're retrieving the record you just loaded in Step (B) so that you can view the data. The data retrieved using the `SELECT` command is as expected

Example

- We have downloaded some historical airline flight data for the years 2007 and 2008 from the website `http://stat-computing.org/dataexpo/2009/the-data.html`.
- In this example We show you how to load this airline data into a Hive table, and then you get a chance to perform some analysis with HiveQL!
- To put this airline data in perspective, the data for the year 2007 is approximately 671MB and the data for the year 2008 is 659MB.
- If you were to download all 22 years' worth of data from `http://stat-computing.org/dataexpo/2009/the-data.html`, it would amount to well over 1 terabyte (TB) of information
- we created two identical tables, named `FlightInfo2007` and `FlightInfo2008`, as you can see in steps (A) and (F) in Listing 13-15.
- Step (B) use the `LOCAL` keyword to load the data. That's because these files are large;.

Listing 13-15: Flight Information Tables from 2007 and 2008

```
(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    1       1       1       1232    1225    1341
          1340    WN      2891    N351    69      75
          54      1   7SMF     ONT     389     4       11
          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    1       1       1       1232    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
          74      8   13       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;

INSERT examples

- Basically, we have three `INSERT` variants;
- To demonstrate this new DML command, we have you create a new table that will hold a subset of the data in the `FlightInfo2008`
- In Step (A), you create this new table and specify that the file format will be row columnar (Step (B)) instead of text.
- The default SerDe for `RCFILE` format is the ColumnarSerDe. You can verify this fact by running the `DESCRIBE EXTENDED myFlightInfo` HiveQL com- mand from the command line interface.

Listing 13-16: Partitioned Version of 2008 Flight Information Table

```
(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 'Flight InfoTable'
                PARTITIONED BY(Month TINYINT)
                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LINES TERMINATED BY '\n'
        (B) STORED AS RCFILE TBLPROPERTIES ('creator'='Bruce Brown',
        'created_at'='Mon Sep  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
                  ...
                  month=9
```

```
(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
```

- Partitions are quite useful to the Hive programmer. However, it's not uncom- mon to encounter a data set where partitioning could become unwieldy, espe- cially if multiple partitions are specified.
- You can also use this `FROM table1 INSERT INTO table2 SELECT ...` format to insert into multiple tables at a time.
- Hive allows only appends, not inserts, into tables, so the `INSERT` keyword simply instructs Hive to append the data to the table.
- Step (G) that you have to use a special Hive command service (`rcfilecat`) to view this table in your warehouse, because the `RCFILE` format is a binary format.

- Third `INSERT` DML is the Dynamic Partition Inserts variant. In Listing 13-16, you partition the `myFlightInfo` table into 12 segments, 1 per month. If you had hundreds of partitions, this task would have become quite diffi- cult, 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

In the Hive DML example in this section, we illustrate the powerful technique in Hive known as *Create Table As Select*, or *CTAS*. Its constructs allow you to quickly derive Hive tables from other tables as you build powerful schemas for big data analysis.

Listing 13-17 shows you how CTAS works, and it sets the stage for other HiveQL examples later in this chapter.

Listing 13-17: An Example of Using CREATE TABLE . . . AS SELECT

```
(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
```

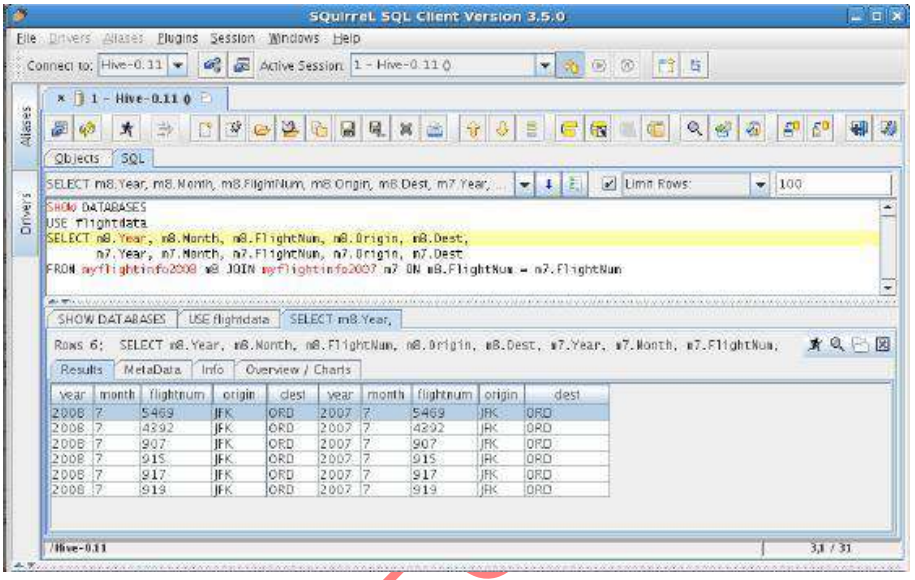
Querying and Analyzing Data

- Hive data types, Hive's DDL, and Hive's DML, helps creating and managing tables but now we help you explore some HiveQL features for querying and analysing data.We begin by exploring table joins in Hive.

Joining tables with Hive:

- in relational database modelling, we split the tables for normalization purpose and use join operation t get the data when it is required.
- Database *normalization* is a technique that guards against data loss, redundancy, and other anomalies as data is updated and retrieved.
- When you begin to query and analyse the data, tables are joined based on the defined relationships between them using SQL — which means that the disks are ultimately busy on your server when you start joining tables, and busy disks usually result in slower user response times. The solutions is that RDBMSs and EDWs are tuned to make joins as fast as possible using HIVE.
- MapReduce is the engine for joining tables, and the Hadoop File System (HDFS) is the underlying storage.
- The potential to unlock information that's hidden in massive data structures is exciting. However, joins with Hive usually don't perform as well as they do in the RDBMS/EDW world.
- Hive table reads and writes via HDFS usually involve very large blocks of data, more data you can manage altogether in one table, the better the overall performance.
- With this background information in mind, we can tackle making joins with Hive. Fortunately, the Hive development community was realistic and understood that users would want and need to join tables with HiveQL.

Figure 13-8:The Hive Inner join



- HIVE supports *equijoins*, a specific type of join that only uses equality comparisons in the join predicate. (`ON m8.FlightNum = m7.FlightNum`, from Figure 13-8 above, is one example of an equi-join.) 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 13-8 illustrates the earlier example of the inner join and two other Hive join types.

- Figure 13-9 illustrates how an inner join works using a Venn diagram, in case you're not familiar with the technique. The basic idea here is that an inner join returns the records that match between two tables. So an inner join is a per- fect analysis tool to determine which flights are the same from JFK (New York) to ORD (Chicago) in July of 2007 and July of 2008.

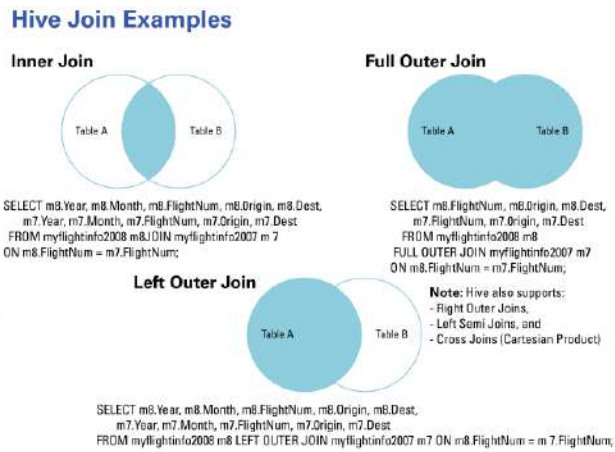


Figure 13-9: Hive inner join, full outer join, and left outer join.

Improving your Hive queries with indexes

- Creating an index is common practice with relational databases when you 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 you have selected.
- Indexes become even more essential when the tables grow extremely large
- Hive supports index creation on tables, though its functionality is still somewhat immature as of this writing.
- We can optimize Hive queries in at least five ways:
    - First, with a little research, you can often speed your joins by leveraging certain optimization techniques, as described on the Hive wiki.
    - Second, column-oriented storage options can be quite helpful. Remember that the ORC file format is new as of Hive 0.11.
    - Third, we demonstrate and discuss how to partition tables in Listing 13-16.
    - Fourth, the Hive community has provided indexing, as illustrated in Listing 13-18.
    - Finally, the `hive.exec.mode.local.auto` configuration variable we mention earlier, in the section "Seeing How the Hive Data Manipulation Language Works."
- In Listing 13-18, we list the steps necessary to index the `FlightInfo2008` table. This extremely large table has millions of rows, so it makes a good candidate for an index or two.
- **Listing 13-18: Creating an Index on 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__ Time taken: 0.079
seconds, Fetched: 1 row(s)
(D) hive (flightdata)> DESCRIBE compact
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(1) 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)
```

- 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.
- In Step (A) the keywords WITH DEFERRED REBUILD instructs Hive to first create an empty index;
- Step (B) is where you 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.
- Step (C) illustrates how you 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 you need to first create the index table and then build it to populate the table.
- Therefore, you 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 (as we demon- strate in Listing 13-18).

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. Examples are quite helpful when explaining windowing and aggregate functions.
- we first discovered this data set was, "**What exactly is the average flight delay per day?"** So we created a query in Listing 13-19 that produces the average departure delay per day in 2008.

Listing 13-19: Finding 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)
```

- Day 5 under the results in Step (B) — had the highest number of delays.
- Step (A): We want to point out that 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 .
- When a view is referenced in HiveQL, Hive executes the query and then uses the results which could be part of a larger query. This can be very useful to simplify complex queries and break them down into logical components.
- Additionally, the `GROUP BY` clause, which gathers all the days per week and allows the `AVG` aggregate function to provide a consolidated answer per day.
- After we answered our question above about average flight delays per day, we came up with another question **"What is the first flight between Airport X and Y?"**! Listing 13-20 provides you with a query that answers these questions.

Listing 13-20: Using Aggregate Window Functions on the Flight Data

```
(A) hive (flightdata)> SELECT f08.Month, f08.DayOfMonth,
cr.description, f08.Origin, f08.Dest, f08.FlightNum, f08.DepTime,
MIN(f08.DepTime)
OVER (PARTITION BY f08.DayOfMonth ORDER BY f08.DepTime) FROM flight
info2008 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 (41
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 5674  1224 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     31 JetBlue Airways          JFK ORD 919   48   48
1     31 JetBlue Airways          JFK ORD 903   635  48
1     31 Comair Inc.              JFK ORD 5447  650  48
1     31 American Airlines Inc.   JFK ORD 1323  840  48
1     31 JetBlue Airways          JFK ORD 907   921  48
1     31 JetBlue Airways          JFK ORD 917   1859 48
```

- 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. As you can see from the list- ing, on January 31, JetBlue has a nice, early flight at 12:48 a.m. — we'll opt for a later one, at 6:35 a.m.

Other key HiveQL features

The following list summarizes them for you:

✓ **Security:** Apache Hive provides a security subsystem that can be quite helpful in preventing accidental data corruption or compromise among trusted members of workgroups, but 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 can not 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. Several examples in this chapter use built-in operators as well as built-in aggregate functions (AVG, MIN, and COUNT, for example). 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.