IBM Software
Information Management

Hive

Hands-On Lab

Table of Contents

1 Introduction 3

2 About this Lab 3

3 Environment Setup Requirements..... Error! Bookmark not defined.

3.1 Getting Started ..... **Error! Bookmark not defined.**

4 General Information about Hive 3

4.1 What is Hive 3

4.2 What Hive is NOT 3

4.3 Data Units 4

4.4 Hive Data Types 4

5 Working with Hive 5

5.1 Starting Hive 5

5.2 Managed Tables and External Tables 6

5.3 Altering and Browsing Tables 7

5.4 Importing Data 9

5.5 Simple Query 9

5.6 Exporting Data 12

5.6.1 Inserting data into Hive Tables *12*

5.6.2 Inserting data into file system *13*

5.6.3 Export including metadata *13*

5.7 Partition Based Query 14

5.8 Joins 17

5.9 Dropping Tables 19

6 Configuration 20

6.1 Configuration management overview 20

6.2 Runtime Configuration 21

6.2.1 MapReduce and Local-Mode ..... ***Error! Bookmark not defined.***

7 Summary 22

1 Introduction

The overwhelming trend towards digital services, combined with cheap storage, has generated massive amounts of data that enterprises need to effectively gather, process, and analyze. Techniques from the data warehousing and high-performance computing communities are invaluable for many enterprises. However, often times their cost or complexity of scale-up discourages the accumulation of data without an immediate need. As valuable knowledge may nevertheless be buried in this data, related scaled-up technologies have been developed. Examples include Google's MapReduce, and the open-source implementation, Apache Hadoop.

Hadoop is an open-source project administered by the Apache Software Foundation. Hadoop's contributors work for some of the world's biggest technology companies. That diverse, motivated community has produced a collaborative platform for consolidating, combining and understanding data.

Technically, Hadoop consists of two key services: data storage using the Hadoop Distributed File System (HDFS) and large scale parallel data processing using a technique called MapReduce

2 About this Lab

After completing this hands-on lab, you'll be able to:

- Understand what Hive is
- Use Hive commands to run a simple MapReduce program on the Hadoop system
- Change configuration values

3 General Information about Hive

3.1 What is Hive

Hive is a data warehousing infrastructure based on the Hadoop. Hadoop provides massive scale out and fault tolerance capabilities for data storage and processing (using the MapReduce programming paradigm) on commodity hardware.

Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data. It provides a simple query language called Hive QL, which is based on SQL and which enables users familiar with SQL to do ad-hoc querying, summarization and data analysis easily. At the same time, Hive QL also allows traditional MapReduce programmers to be able to plug in their custom mappers and reducers to do more sophisticated analysis that may not be supported by the built-in capabilities of the language.

3.2 What Hive is NOT

Hadoop is a batch processing system and Hadoop jobs tend to have high latency and incur substantial overheads in job submission and scheduling. As a result - latency for Hive queries is generally very high (minutes) even when data sets involved are very small (say a few hundred megabytes). As a result it cannot be compared with systems such as Oracle where analyses are conducted on a significantly smaller amount of data but the analyses proceed much more iteratively with the response times between iterations being less than a few minutes. Hive aims to provide acceptable (but not optimal) latency for interactive data browsing, queries over small data sets or test queries.

Hive is not designed for online transaction processing and does not offer real-time queries and row level updates. It is best used for batch jobs over large sets of immutable data (like web logs).

3.3 Data Units

In the order of granularity - Hive data is organized into:

- **Databases**: Namespaces that separate tables and other data units from naming confliction.
- **Tables**: Homogeneous units of data which have the same schema.
- **Partitions**: Each Table can have one or more partition Keys which determines how the data is stored. Partitions - apart from being storage units - also allow the user to efficiently identify the rows that satisfy a certain criteria. Therefore, if you run analysis on a certain partition, you can run that query only on the relevant partition of the table thereby speeding up the analysis significantly. Note however, that just because a partition is named on a certain word, it does not mean that it contains all or only data from that word; partitions are named after that word for convenience but it is the user's job to guarantee the relationship between partition name and data content). Partition columns are virtual columns; they are not part of the data itself but are derived on load.
- **Buckets** (or **Clusters**): Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table. For example the page_views table may be bucketed by userid, which is one of the columns, other than the partitions columns, of the page_view table. These can be used to efficiently sample the data.

Note that it is not necessary for tables to be partitioned or bucketed, but these abstractions allow the system to prune large quantities of data during query processing, resulting in faster query execution.

3.4 Hive Data Types

Primitive types:

- TINYINT, SMALLINT, INT, BIGINT
- BOOLEAN
- FLOAT
- DOUBLE
- STRING
- BINARY (Note: Only available starting with Hive 0.8.0)
- TIMESTAMP (Note: Only available starting with Hive 0.8.0)
- DECIMAL (Note: Only available starting with Hive 0.11.0)

Complex types:

- arrays: ARRAY<data_type>
- maps: MAP<primitive_type, data_type>

- structs: STRUCT<col_name : data_type [COMMENT col_comment], ...>
- union: UNIONTYPE<data_type, data_type, ...>

Handling of NULL Values:

Missing values are represented by the special value NULL. To import data with NULL fields, check documentation of the SerDe used by the table. (The default Text Format uses LazySimpleSerDe which interprets the string \N as NULL when importing.)

4 Working with Hive

4.1 Starting Hive

1. Ensure the Apache Derby component is started. Apache Derby is the default database used as metastore in Hive. A quick way to verify if it is started, is to try to start it using:

```
start.sh derby
```

You will get message like below :

```
biadmin@imtebi:/opt/ibm/biginsights/hive/conf> start.sh derby
[INFO] Progress - Start derby
[INFO] @imtebi.imte.com - derby started, pid 17851
[INFO] Progress - 100%
[INFO] DeployManager - Start; SUCCEEDED components: [derby]; Consumes : 6202ms
biadmin@imtebi:/opt/ibm/biginsights/hive/conf>
```

2. Start hive interactively. Change directory to $HIVE_HOME/bin and launch Hive shell.

```
cd $HIVE_HOME/bin
./hive
```

You will see the Hive Command-line Interface (CLI).

```
biadmin@imtebi:/opt/ibm/biginsights/hive/conf> cd $HIVE_HOME/bin
biadmin@imtebi:/opt/ibm/biginsights/hive/bin> ./hive
2013-04-15 14:44:11.903 GMT : Connection obtained for host: imtebi.imte.com, port
number 1528.
Logging initialized using configuration in file:/opt/ibm/biginsights/hive/conf/hive-
log4j.properties
Hive history
file=/var/ibm/biginsights/hive/query/biadmin/hive_job_log_biadmin_201304151044_125574
4548.txt
hive>
```

4.2 Managed Tables and External Tables

When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory. Alternatively, you may create an *external table*, which tells Hive to refer to the data that is at an existing location outside the warehouse directory.

The difference between the two types of table is seen in the LOAD and DROP semantics. Let's consider a managed table first.

1. Creates a Hive table called employee with two columns, the first being an integer and the other a string.

```
hive> CREATE TABLE employee (id INT, name STRING, dept_id int);
```

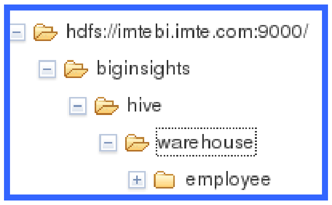
By default, tables are assumed to be of text input format and the delimiters are assumed to be ctrl-A(ASCII 001).

2. When you load data into a managed table, it is moved into Hive's warehouse directory.

```
hive> LOAD DATA LOCAL INPATH '/home/biadmin/labs/hive/employee.del' OVERWRITE
INTO TABLE employee;
```

This load query will move the file `'/home/biadmin/labs/hive/employee.del'` into Hive's warehouse directory, which is `hdfs://biginsights/hive/warehouse`. Hive creates a folder corresponding to each table that is created and dumps the data in the form of serialized files into it (which is abstracted to the user).

You can open up another terminal and run "`hadoop fs -ls /biginsights/hive/warehouse`" or open Web Console to check the directory.



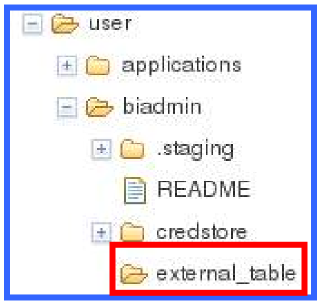
Check the file employee/employee.del and you will see that the file has not been modified only moved to the location. Hive whenever possible will not change the format of the file instead changing the serializers that read and write the data.

3. If the table is later dropped (will cover this part later in this lab), then the table, including its metadata and its data, is deleted. It bears repeating that since the initial `LOAD` performed a move operation, and the `DROP` performed a delete operation, the data no longer exists anywhere. This is what it means for Hive to manage the data.

4. Alternatively, we can create an external table which behaves differently. We will create external table with delimited row format.

```
hive> CREATE EXTERNAL TABLE dep (id INT, name STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\t' LOCATION '/user/biadmin/external_table';
```

The location of the external data is specified at table creation time. With the EXTERNAL keyword, Hive knows that it is not managing the data, so it doesn't move it to its warehouse directory. Indeed, it doesn't even check if the external location exists at the time it is defined. This is a useful feature, since it means you can create the data lazily after creating the table.



Also, the delimited row format specifies how the rows are stored in the hive table. In the case of the delimited format, this specifies how the fields are terminated, how the items within collections (arrays or maps) are terminated and how the map keys are terminated.

When you drop an external table, Hive will leave the data untouched and only delete the metadata.

So how do you choose which type of table to use? In most cases, there is not much difference between the two (except of course for the difference in DROP semantics), so it is a just a matter of preference. As a rule of thumb, if you are doing all your processing with Hive, then use managed tables, but if you wish to use Hive and other tools on the same dataset, then use external tables. A common pattern is to use an external table to access an initial dataset stored in HDFS (created by another process), then use a Hive transform to move the data into a managed Hive table. This works the other way around, too — an external table (not necessarily on HDFS) can be used to export data from Hive for other applications to use.

Another reason for using external tables is when you wish to associate multiple schemas with the same dataset.

4.3 Altering and Browsing Tables

1. See the column details of the table.

```
hive> DESCRIBE employee;
```

You should see the list of columns.

```
hive> DESCRIBE employee;
OK
id    int
name  string
dept_id     int
Time taken: 0.285 seconds
```

2. As for altering tables, table names can be changed and additional columns can be added or dropped.

```
hive> ALTER TABLE dep RENAME TO department;
hive> SHOW TABLES;
```

You will see the table name has been changed.

```
hive> ALTER TABLE dep RENAME TO department;
OK
Time taken: 0.165 seconds
hive> SHOW TABLES;
OK
department
employee
Time taken: 0.083 seconds
```

3. Add a column to the table.

```
hive> ALTER TABLE department ADD COLUMNS (loc STRING);
hive> DESC department;
```

The new column ‘loc’ will be added to the table.

```
hive> ALTER TABLE department ADD COLUMNS (loc STRING);
OK
Time taken: 0.129 seconds
hive> DESC department;
OK
id    int
name  string
loc   string
Time taken: 0.123 seconds
```

Note that a change in the schema (such as the adding of the columns), preserves the schema for the old partitions of the table in case it is a partitioned table. All the queries that access these columns and run over the old partitions implicitly return a null value or the specified default values for these columns.

In the later versions we can make the behavior of assuming certain values as opposed to throwing an error in case the column is not found in a particular partition configurable.

4.4 Importing Data

We've briefly introduced how to import the data to Hive in Section 5.2 by loading data to employee table. As you saw, Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move data files into locations corresponding to Hive tables, therefore, you must have table created in advance. Besides 'LOAD' command, we do have 'IMPORT' commands added since Hive 0.8.0, however, in order to use this command, it requires metadata along with data. For the purpose of this lab, we will only go over Load command.

1. The following query loads the data from the flat files on to Hive.

```
hive> LOAD DATA LOCAL INPATH '/home/biadmin/labs/hive/department.del' OVERWRITE
INTO TABLE department;
```

The *filepath* can be relative path, absolute path, or a full URI with scheme and (optionally) an authority. It can refer to a file or it can be a directory. In either case, *filepath* addresses a set of files.

The keyword 'LOCAL' signifies that the input file is on the local file system. If 'LOCAL' is omitted then it looks for the file in HDFS.

The keyword 'OVERWRITE' signifies that existing data in the table is deleted. If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.

ⓘ **Note:**

- NO verification of data against the schema is performed by the load command.
- If the file is in hdfs, it is moved into the Hive-controlled file system namespace.
  The root of the Hive directory is specified by the option hive.metastore.warehouse.dir
  in hive-default.xml. We advise users to create this directory before
  trying to create tables via Hive.

4.5 Simple Query

1. Select all columns and rows stored in employee table.

```
hive> SELECT * FROM employee;
```

You will see the data loaded from employee.del file.

```
hive> SELECT * FROM employee;
OK
1     Jones 102
2     Steinberg   100
3     Robinson    100
4     Lee   101
5     Brown 101
6     Williams    101
7     Davis 102
8     Miller      101
9     Wilson      101
10    Moore 100
Time taken: 0.13 seconds
```

2. Run a same query with limit to the number of output.

```
hive> SELECT * FROM employee LIMIT 5;
```

First 5 rows of the previous results will show.

```
hive> SELECT * FROM employee LIMIT 5;
OK
1     Jones 102
2     Steinberg   100
3     Robinson    100
4     Lee   101
5     Brown 101
Time taken: 0.125 seconds
```

3. Filter the table.

```
hive> SELECT name FROM employee WHERE dept_id=102;
```

The expected output is the ones in bold (Jones and Davis).

```
hive> SELECT a.name FROM employee a WHERE a.id=1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201304110627_0031, Tracking URL =
http://imtebi.imte.com:50030/jobdetails.jsp?jobid=job_201304110627_0031
Kill Command = /opt/ibm/biginsights/IHC/libexec/../bin/hadoop job  -
Dmapred.job.tracker=imtebi.imte.com:9001 -kill job_201304110627_0031
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2013-04-22 11:52:10,221 Stage-1 map = 0%,  reduce = 0%
2013-04-22 11:52:14,550 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.44 sec
2013-04-22 11:52:15,553 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.44 sec
2013-04-22 11:52:16,559 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 0.44 sec
MapReduce Total cumulative CPU time: 440 msec
Ended Job = job_201304110627_0031
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.44 sec   HDFS Read: 482 HDFS Write: 37 SUCCESS
Total MapReduce CPU Time Spent: 440 msec
OK
Jones
Davis
Time taken: 14.983 seconds
```

This query is internally compiled into a MapReduce job and fetches the requested data for us. We see that this simple conditional select query takes more time than a query on relational database. But, to gauge the actual performance, the framework has to be tested with huge data, during which MapReduce framework could be properly leveraged and we could feel the need for Hive in such circumstances.

4. Use GROUP BY and ORDER BY clauses.

```
hive> SELECT dept_id, count(*) c FROM employee GROUP BY dept_id ORDER BY c DESC;
```

Result should be as below.

```
MapReduce Total cumulative CPU time: 4 seconds 30 msec
Ended Job = job_201304110627_0074
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 4.56 sec   HDFS Read: 355 HDFS Write:
153 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 4.03 sec   HDFS Read: 615 HDFS Write:
18 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 590 msec
OK
101   5
100   3
102   2
Time taken: 50.592 seconds
```

4.6 Exporting Data

Hive supports multiple ways to export data into HDFS or local file system. INSERT and EXPORT statements are one of those ways. These methods are to accomplish the same objective, but output slightly different results

4.6.1 Inserting data into Hive Tables

Hive stores data as files, thus you can export data by creating another table and insert data into it. However, note that dropping a table will result in deleting the file as well.

1. Create a table.

```
hive> CREATE TABLE names (name STRING);
```

2. Insert data into Hive table from a query.

```
hive> INSERT OVERWRITE TABLE names SELECT name FROM employee WHERE id > 7;
```

Query results can be inserted into tables by using the insert clause

3. See if we have a file created in HDFS

```
hive> dfs -ls /biginsights/hive/warehouse/names;
```

You will see that the file is generated. (000000_00)

```
hive> dfs -ls /biginsights/hive/warehouse/names;
Found 1 items
-rw-r--r--   3 biadmin supergroup         20 2013-04-24 17:17
/biginsights/hive/warehouse/names/000000_0
```

Note that the dfs command is build directly into the hive console so we do not need to switch to a different console to use `hadoop fs`.

4. Print the content of the file.

```
hive> dfs -cat /biginsights/hive/warehouse/names/*;
```

Expected contents are Wilson, and Moore.

```
hive> dfs -cat /biginsights/hive/warehouse/names/*;
Miller
Wilson
Moore
```

4.6.2 Inserting data into file system

Instead of exporting data into tables, you can directly ingest data into file system (both HDFS and local). You can also achieve similar result with EXTERNAL tables.

1. Select all rows from employee table into an HDFS directory.

```
hive> INSERT OVERWRITE DIRECTORY '/user/biadmin/hdfs_out' SELECT * FROM employee;
```

Query results can be inserted directly into file system directories.

If LOCAL keyword is used - then Hive will write data to the directory on the local file system.

2. List the content in the directory.

```
hive> dfs -ls /user/biadmin/hdfs_out;
```

You will see that the file is generated. (000000_00)

```
hive> dfs -ls /user/biadmin/hdfs_out/;
Found 1 items
-rw-r--r--   3 biadmin supergroup        131 2013-04-25 16:12
/user/biadmin/hdfs_out/000000_0
```

Data written to the file system with this method is serialized as text with columns separated by ^A and rows separated by newlines. If any of the columns are not of primitive type - then those columns are serialized to JSON format.

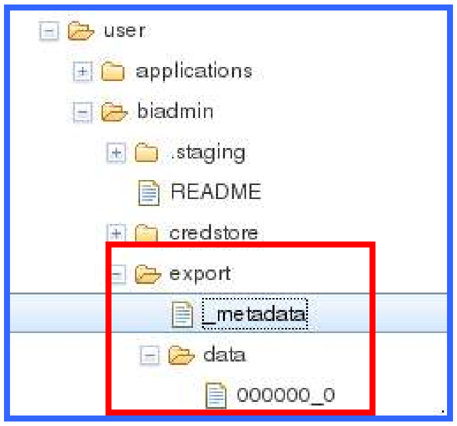
4.6.3 Export including metadata

Hive provides EXPORT command. The difference between insert commands is that Export command exports metadata along with data.

1. Select all rows from employee table into an HDFS directory.

```
hive> EXPORT TABLE names TO '/user/biadmin/export';
```

2. Check content in `/user/biadmin/export` directory using “`dfs –ls`” command as above examples or Web Console.



4.7 Partition Based Query

There are two types of partition columns.

- Static Partition columns: in DML/DDL involving multiple partitioning columns, the columns whose values are known at COMPILE TIME (given by user).
- Dynamic Partition columns: columns whose values are only known at EXECUTION TIME.

In order to load data to different partitions with static partition insert, you have to add an insert statement for each partition in the input data. This is very inconvenient since you have to have the priori knowledge of the list of partitions exist in the input data and create the partitions beforehand. It is also inefficient since each insert statement may be turned into a MapReduce Job.

We are going to demonstrate *Dynamic-partition insert* (or multi-partition insert) on this section. It is designed to solve this problem by dynamically determining which partitions should be created and populated while scanning the input table. This is a newly added feature that is only available from version 0.6.0. In the dynamic partition insert, the input column values are evaluated to determine which partition this row should be inserted into. If that partition has not been created, it will create that partition automatically. Using this feature you need only one insert statement to create and populate all necessary partitions. In addition, since there is only one insert statement, there is only one corresponding MapReduce job. This significantly improves performance and reduces the Hadoop cluster workload comparing to the multiple insert case.

1. First you need to change the configuration values.

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

hive.exec.dynamic.partition is whether or not to allow dynamic partition in DML/DLL, and if *hive.exec.dynamic.partition.mode* is in strict mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions.

2. Create another table 'emp_pt' for partitioning,

```
hive> CREATE TABLE emp_pt (id INT, name STRING) PARTITIONED BY (dept_id INT);
```

3. Insert data by loading it from the other table employee, and partition it by its department id.

```
hive> INSERT OVERWRITE TABLE emp_pt PARTITION (dept_id)
      SELECT id, name, dept_id FROM employee;
```

When there are already non-empty partitions exists for the dynamic partition columns, it will be overwritten if the dynamic partition insert saw the same value in the input data. This is in line with the 'insert overwrite' semantics. However, if the partition value does not appear in the input data, the existing partition will not be overwritten.

Dynamic partition insert could potentially resource hog in that it could generate a large number of partitions in a short time. To get yourself buckled, define two parameters hive.exec.max.dynamic.partitions (default value being 1000) which is the total number of dynamic partitions could be created by one DML and hive.exec.max.dynamic.partitions.pernode (default value being 100) which is the maximum dynamic partitions that can be created for each mapper or reducer. If one mapper or reducer created more than that the threshold, a fatal error will be raised from the mapper/reducer (through counter) and the whole job will be killed. If each mapper/reducer did not exceed the limit but the total number of dynamic partitions does, then an exception is raised at the end of the job before the intermediate data are moved to the final destination.

If the input column value is NULL or empty string, the row will be put into a special partition, whose name is controlled by the hive parameter hive.exec.default.dynamic.partition.name. The default value is `__HIVE_DEFAULT_PARTITION__`. Basically this partition will contain all "bad" rows whose values are not valid partition names. The caveat of this approach is that the bad value will be lost and is replaced by `__HIVE_DEFAULT_PARTITION__` if you select them Hive.

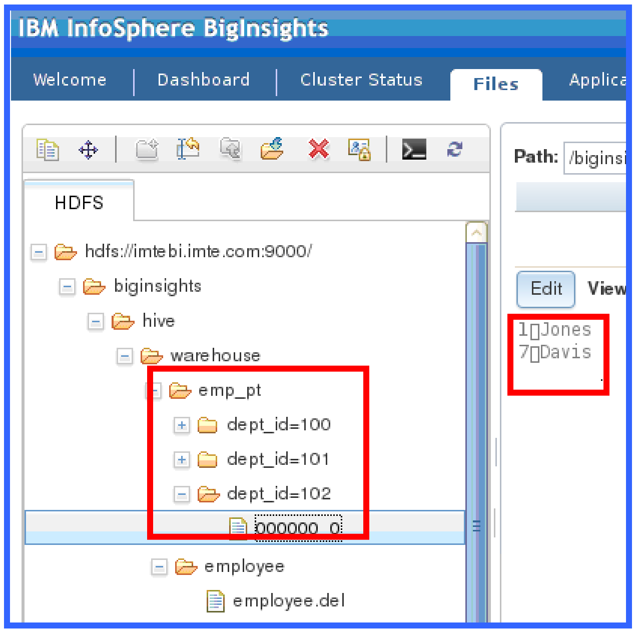
4. Run select statement.

```
hive> SELECT * FROM employee WHERE dept_id=102;
```

This will return the results only needing to read the single file containing the partition 102.

```
hive> SELECT * FROM emp_pt WHERE dept_id=102;
OK
1     Jones 102
7     Davis 102
Time taken: 0.095 seconds
```

5. Go to Files tab in Web Console to browse the HDFS. Click one of the partitioned file in the directory.



```
hdfs://imtebi.imte.com:9000/
  biginsights
    hive
      warehouse
        emp_pt
          dept_id=100
          dept_id=101
          dept_id=102
            000000_0
        employee
          employee.del
```

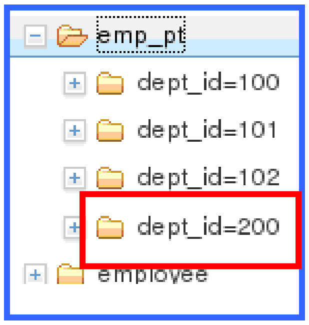
```
1Jones
7Davis
```

emp_pt table is stored in multiple partitions unlike *employee*. Also, notice that the file doesn't contain dept_id data in it. We mentioned it before that partition columns are virtual columns and they are not part of the data itself but are derived on load.
Since a Hive partition corresponds to a directory in HDFS, the partition value has to conform to the HDFS path format (URI in Java). Any character having a special meaning in URI (e.g., '%', ':', '/', '#') will be escaped with '%' followed by 2 bytes of its ASCII value. If the input column is a type different than STRING, its value will be first converted to STRING to be used to construct the HDFS path.

6. Load data from flat file using static partition by manually specifying the partition.

```
hive> LOAD DATA LOCAL INPATH '/home/biadmin/labs/hive/employee_extra.del'
OVERWRITE INTO TABLE emp_pt PARTITION (dept_id='200');
```

7. You will notice in HDFS, another directory for dept_id=200 partition is created.



```
emp_pt
  dept_id=100
  dept_id=101
  dept_id=102
  dept_id=200
employee
```

4.8 Joins

Only equi-joins, outer joins, and left semi joins are supported in Hive. Hive does not support join conditions that are not equality conditions as it is very difficult to express such conditions as a map/reduce job. Also, more than two tables can be joined in Hive, and it is best to put the largest table on the rightmost side of the join to get the best performance.

1. Select all from the department table.

```
hive> SELECT * from department;
```

```
hive> SELECT * FROM department;
OK
100   Engineering Toronto
101   Research    San Jose
102   Marketing   New York
103   Sales New York
200   Service     Seattle
Time taken: 5.852 seconds
```

2. Insert data by loading it from the employee table, and partition employee table by its department id.

```
hive> SELECT e.*, d.name, d.loc FROM emp_pt e JOIN department d ON
(e.dept_id=d.id) WHERE e.dept_id = '100';
```

Result will look as below, and ones in bold are actual query results.

```
MapReduce Total cumulative CPU time: 5 seconds 280 msec
Ended Job = job_201304110627_0036
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 5.28 sec   HDFS Read: 847 HDFS Write:
94 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 280 msec
OK
2     Steinberg   100   Engineering Toronto
3     Robinson    100   Engineering Toronto
10    Moore 100   Engineering Toronto
Time taken: 30.947 seconds
```

3. In order to do outer joins the user can qualify the join with LEFT OUTER, RIGHT OUTER or FULL OUTER keywords in order to indicate the kind of outer join (left preserved, right preserved or both sides preserved). For example, in order to do a full outer join in the query above, the corresponding syntax would look like the following query.

```
hive> SELECT e.*, d.name, d.loc
    FROM emp_pt e FULL OUTER JOIN department d ON (e.dept_id = d.id);
```

Result will be as below.

```
Ended Job = job_201304110627_0055
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 5.17 sec   HDFS Read: 989 HDFS Write:
400 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 170 msec
OK
2     Steinberg   100   Engineering Toronto
3     Robinson    100   Engineering Toronto
10    Moore 100   Engineering Toronto
4     Lee   101   Research    San Jose
5     Brown 101   Research    San Jose
6     Williams    101   Research    San Jose
8     Miller      101   Research    San Jose
9     Wilson      101   Research    San Jose
1     Jones 102   Marketing   New York
7     Davis 102   Marketing   New York
NULL  NULL  NULL  Sales New York
21    Smith 200   Service     Seattle
22    John  200   Service     Seattle
Time taken: 25.884 seconds
```

Notice that Sales department doesn't have any employee, thus shows NULL on the table.

4. In order check the existence of a key in another table, the user can use LEFT SEMI JOIN as illustrated by the following example. LEFT SEMI JOIN implements the correlated IN/EXISTS subquery semantics in an efficient way. Since Hive currently does not support IN/EXISTS subqueries, you can rewrite your queries using LEFT SEMI JOIN. The restrictions of using LEFT SEMI JOIN is that the right-hand-side table should only be referenced in the join condition (ON-clause), but not in WHERE- or SELECT-clauses etc.

```
hive> SELECT e.*
    FROM emp_pt e LEFT SEMI JOIN department d ON (e.dept_id = d.id);
```

Result will be as below.

```
Ended Job = job_201304110627_0056
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 5.65 sec   HDFS Read: 989 HDFS Write:
156 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 650 msec
OK
2     Steinberg   100
3     Robinson    100
10    Moore 100
4     Lee   101
5     Brown 101
6     Williams    101
8     Miller      101
9     Wilson      101
1     Jones 102
7     Davis 102
21    Smith 200
22    John  200
Time taken: 28.746 seconds
```

4.9 Dropping Tables

1. The DROP TABLE statement removes metadata and data for a table.

```
hive> DROP TABLE employee;
```

This table is a Hive managed table, thus the table information is removed from the metastore and the raw data is removed as if by 'hadoop dfs -rm'.

- **Note:** In Hive 0.70 or later, DROP returns an error if the table doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

2. Drop an EXTERNAL table. (department)

```
hive> DROP TABLE department;
```

In the case of external table, only the metadata is deleted, and data is left untouched, thus remains in the file system.

3. See if we still have the data within file system.

```
hive> dfs -cat /user/biadmin/external_table/department.del;
hive> dfs -cat /user/biadmin/external_table/department.del;
100   Engineering Toronto
101   Research    San Jose
102   Marketing   New York
103   Sales New York
200   Service     Seattle
```

4. Delete the content of the table. (emp_pt)

```
hive> dfs -rmr /biginsights/hive/warehouse/emp_pt;
```

If you want to only delete all the data in a table, but keep the table definition (like DELETE or TRUNCATE in SQL), then you can simply delete the data files.

5 Configuration

5.1 Configuration management overview

1. Hive configuration is an overlay on top of Hadoop - meaning the Hadoop configuration variables are inherited by default.
2. Hive default configuration is stored in **$HIVE_HOME/conf/hive-default.xml**
   Configuration can be manipulated by editing **$HIVE_HOME/conf/hive-site.xml** and defining any desired variables (including Hadoop variables) in it.
3. Log4j is an open source project based on the work of many authors. It allows the developer to control which log statements are output with arbitrary granularity. It is fully configurable at runtime using external configuration files. Log4j configuration is stored **$HIVE_HOME/conf/hive-log4j.properties.**
4. From the Linux command line (new window or by quitting the hive console with `quit;`

```
cd $HIVE_HOME/conf
ls
```

You will see the list of configuration files including (hive-default.xml, hive-site.xml, and hive-log4j.properties)

```
biadmin@imtebi:/opt/ibm/biginsights/bin> cd $HIVE_HOME/conf
biadmin@imtebi:/opt/ibm/biginsights/hive/conf> ls
hive-default.xml.template  hive-env.sh.template       hive-exec-
log4j.properties.template  hive-log4j.properties.template  keystore.jks
hive-env.sh                hive-exec-log4j.properties  hive-log4j.properties
hive-site.xml
biadmin@imtebi:/opt/ibm/biginsights/hive/conf>
```

5. The location of the Hive configuration directory can be changed by setting the HIVE_CONF_DIR environment variable.

5.2 Runtime Configuration

Hive queries are executed using map-reduce queries and, therefore, the behavior of such queries can be controlled by the Hadoop configuration variables. These runtime configuration is only valid during the current session of hive, which means if you ‘quit’ the session, all runtime configuration will be lost.

1. The ‘SET’ command can be used to set or check any Hadoop or Hive configuration variable. (restart hive console by executing $HIVE_HOME/bin/hive

```
hive> SET hive.exec.dynamic.partition.mode;
```

Current value of this configuration variable is printed.

```
hive> set hive.exec.dynamic.partition.mode;
hive.exec.dynamic.partition.mode=nonstrict
```

2. Change the value to ‘strict’.

```
hive> SET hive.exec.dynamic.partition.mode=strict;
```

3. Check the variable once more.

```
hive> SET hive.exec.dynamic.partition.mode;
```

The value is modified to strict.

```
hive> set hive.exec.dynamic.partition.mode;
hive.exec.dynamic.partition.mode=strict
```

4. With –v option, it prints all Hadoop and Hive configuration variables. Without –v option, only the variables that differ from the base Hadoop configuration are displayed.

```
hive> SET -v;
```

5.3 Local Mode

Many Hadoop jobs need the full scalability benefits of Hadoop to process large data sets. However, there are times when the input to Hive is very small. In these cases, the overhead of launching tasks for queries consumes a significant percentage of the overall job execution time. In many of these cases, Hive can leverage the lighter weight of the *local mode* to perform all the tasks for the job on a single machine and sometimes in the same process. The reduction in execution times can be dramatic for small data sets.

Hive compiler generates map-reduce jobs for most queries. These jobs are then submitted to the MapReduce cluster indicated by the ***mapred.job.tracker*** variable. By changing this MapReduce cluster to 'local', it enters the local mode.

Local mode execution is usually significantly faster than submitting jobs to a large cluster. Data is accessed transparently from HDFS. Conversely, local mode only runs with one reducer and can be very slow processing larger data sets.

5. Enable local mode temporarily.

```
hive> SET mapred.job.tracker=local;
```

In addition, mapred.local.dir should point to a path that's valid on the local machine. (Otherwise, the user will get an exception allocating local disk space).

6. Hive also supports a mode to run map-reduce jobs in local-mode automatically.

```
hive> SET hive.exec.mode.local.auto=true;
```

Note that this feature is *disabled* by default. If enabled - Hive analyzes the size of each map-reduce job in a query and may run it locally if the following thresholds are satisfied.

So, for queries over small data sets, or for queries with multiple map-reduce jobs where the input to subsequent jobs is substantially smaller (because of reduction/filtering in the prior job), jobs may be run locally.

- ⓘ **Note:** There may be differences in the runtime environment of hadoop server nodes and the machine running the hive client (because of different jvm versions or different software libraries). This can cause unexpected behavior/errors while running in local mode. Also note that local mode execution is done in a separate, child jvm (of the hive client). If the user so wishes, the maximum amount of memory for this child jvm can be controlled via the option hive.mapred.local.mem. By default, it's set to zero, in which case Hive lets Hadoop determine the default memory limits of the child jvm.

6 Summary

You have just completed the hands-on lab which is focused on Hive on Hadoop. You should now know how to perform the following basic tasks on the platform:

- Run simple Hive queries
- Create/drop table
- Import/Export data to/from HDFS or local file system
- Partition the table
- Modify configuration settings