Tuto2 Ibm

Version 1.0

2019-02-01

DATASOFT CONSULTING

Historique des modifications du document

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2019-02-01 | 1.0 | Tuto2\_Ibm | Abdelaziz HANI |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

1. Introduction 4

1.1 Overview 4

2. Administering BigSQL 4

3. Exporting from DB2 5

4. General Information about Hive 9

4.1 What is Hive 9

4.2 What Hive is NOT 9

4.3 Hive Data Types 10

5. Working with Hive 10

5.1 Starting Hive 10

5.2 Managed Tables and External Tables 11

5.3 Altering and Browsing Tables 12

5.4 Simple Query 13

5.5 Exporting Data 15

5.5.1 Inserting data into Hive Tables 15

5.5.2 Inserting data into file system 15

5.6 Joins 18

5.7 Dropping Tables 20

6. Summary 20

# Introduction

Hadoop environments provide the ability to analyze huge amounts of data with MapReduce and various high level languages like JAQL and Pig. However they do not provide well formed interfaces to other tools and they require a lot of skill buildup. Providing a good SQL interface for a Hadoop cluster has therefore been a big priority for almost all companies involved in Hadoop. A good JDBC/ODBC connection with standard SQL would enable easy integration with pretty much any data management and analytic tool and allow the utilization of SQL and database skills present in pretty much any company. Hive is the open source technology that is most widely used in that matter. However it has a lot of drawbacks. These include missing function and datatype support, and mixed performance especially for small point queries and limits in data loading.

BigSQL(v3) is an IBM technology that aims to fix these problems. It is based on DB2 technology and provides much higher performance than Hive (on average ten times faster in the TPCH benchmark). In addition to that it uses a mature database engine that has excellent client support and features like stored procedures, federation, and sophisticated security.

BigSQL(v3) replaces BigSQL(v1) a Jaql based database engine that was added to BigInsights in version 2.1.0.

BigSQL(v1) is still shipped with the product for backwards compatibility and because it provides some features BigSQL(v3) does not yet have. The biggest ones are HBase support and the use of modules like R or text analytics. It is installed as a separate module (bigsql1) and running on a different port. However it can access the same tables as BigSQL(v3). In this lab we will refer to BigSQL(v3) as BigSQL, if we have to talk about BigSQL(v1) we will add the version.

## Overview

In this lab we will do a simple migration of a set of database tables from a DB2 database into BigSQL. We will first migrate the DDL statements and re-create the tables in BigSQL. We will then load the data. We will do this using two methods first loading data from delimited files then directly loading them using a JDBC connection. Finally we will port some SQL queries and show how easy it is to move data between a traditional database like DB2 and BigSQL. For the DDL we will utilize the Eclipse tooling provided with BigInsights.

# Administering BigSQL

In this chapter we will quickly go over the basics of working with BigSQL. BigSQL is sharing the storage and catalog infrastructure with Hive. It is reusing the Hive catalog and most Hive tables can simply be accessed in BigSQL as well.

Tables can be stored in Hive tables. All tables have their metadata stored in the Hive metastore. Hive table data is persisted in the Hive warehouse folder in HDFS. Since BigSQL is based on DB2 there is also a BigSQL catalog which is synced by the BigSQL scheduler. Normally you do not need to worry about it, however if you have created a Hive table outside of BigSQL it will not be accessible automatically. However you can import tables and whole schemas using a stored procedure.

Like in Hive we have a BigSQL server that is handling all types of queries. You can stop and start him with start.sh bigsql and stop.sh bigsql. This is handling both BigSQL and BigSQL1, however you can access only BigSQL(v1) using start.sh bigsql –bigsql1. You can also administer it in the Cluster status tab of the console. You can access BigSQL remotely using pretty much any JDBC/ODBC capable tool or using the jsqsh console. Jsqsh is a java based command line interface that provides a good command line environment. Let’s try it out.

1. Open a command line window for biadmin.
2. If you run the lab in a VM with less than 8GB of RAM you need to disable some unneeded services first



1. Start BigSQL with



You should see that the server is already started and the command should end successfully

1. Start the JSQSH command line utility



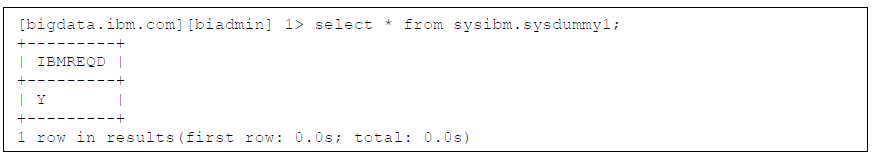
1. You need to connect to BigSQL using the following command \connect bigsql. Enter the password “passw0rd”

This works because the BigSQL server is the default connection of jsqsh in our environment. You could have other connections defined to other databases. You could also connect to BigSQL(v1) with \connect bigsql1

1. Run the following command to get a list of all tables in the database. At this point in time it will only show you the system catalog tables. We have not yet created any tables. \show tables
2. Finally let’s run a simple query. Sysdummy1 is a dummy table that is similar to the DUAL table in Oracle.



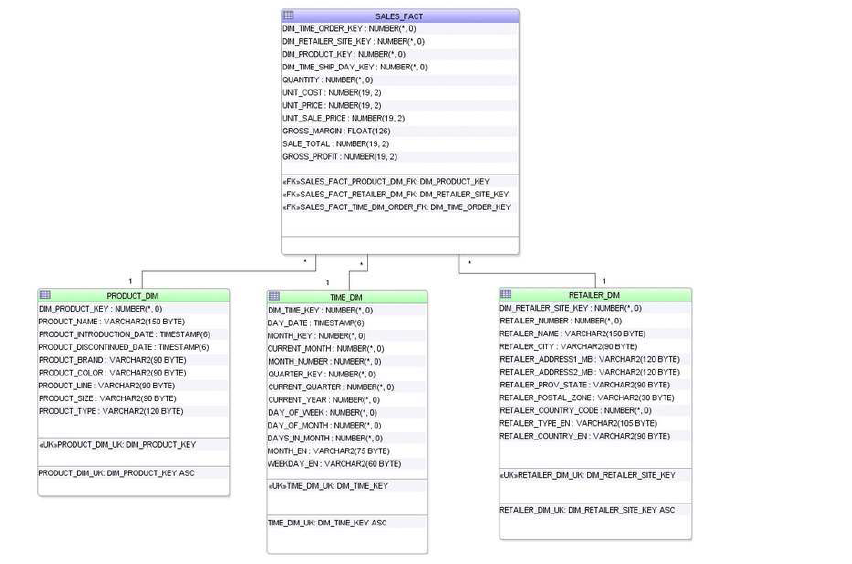
You will see the following results.



You have now run your first BigSQL query. We will now migrate a simple warehousing star schema from a DB2 database into BigSQL.

# Exporting from DB2

In this chapter we will investigate the data model and export the DDLs and data files from the DB2 database.



While we are in the command line lets investigate the data we want to transfer. Basically we have a very simple Warehousing Star Schema in DB2. It has one fact table SALES\_FACT which contains Point of Sales data. We have three dimension tables TIME\_DIM, PRODUCT\_DIM, and RETAILER\_DIM, which contain the time of sales, product that was sold and the store that sold it. BigSQL is a warehousing technology which is suitable for the aggregation of huge amounts of data. Keep in mind that while it is a big progress over Hive it is not as sophisticated as existing database optimizers.

Small data models that do not have too many branches are best.

1. Open a new Terminal window. (Right-Click Linux Desktop and Open Terminal)

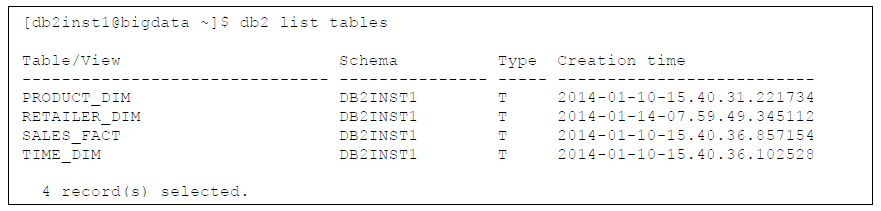
2. Switch to DB2INST1 with su – db2inst1, password is “passw0rd”

3. Start DB2 with db2start

4. Connect to the database SALES using db2 connect to SALES

5. Display the tables in the database using db2 list tables

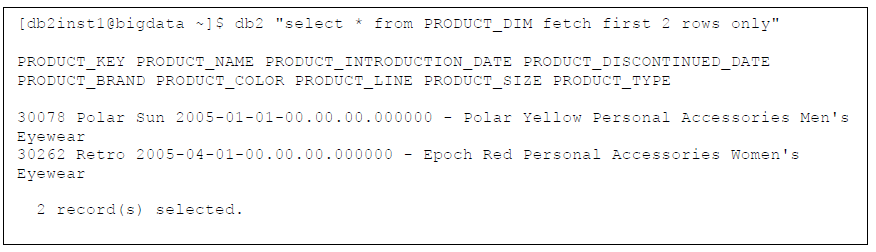
You should see the 4 tables of the Schema above.



6. Investigate the tables a bit. As an example:



You will see that the table contains information on the products sold in the store. In this example outdoor equipment like sunglasses, golfing goods, etc. The dimension table contains information about product line, size, colour etc.



1. Feel free to investigate the data model a bit more. It is a subset of the Cognos GoSales dataset. One good thing is to verify row counts so they can be verified after the migration. Let’s do this exemplary on SALES\_FACT the FACT table.

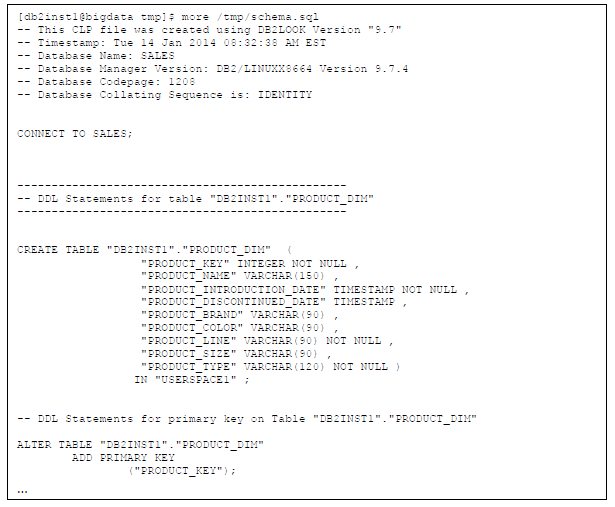


You will see that the table has a row count of 443439. After the migration we will verify that we have transferred all of them successfully.

1. To port a database schema we first need to retrieve the original schema. DB2 offers a handy utility for this called db2look. Execute the following command to retrieve the schema and save it in a file.

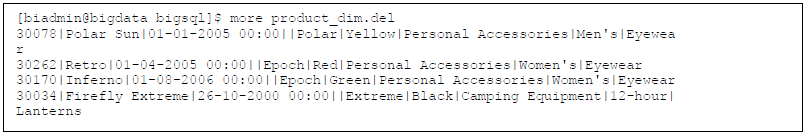


1. Have a look at the output with more /tmp/schema.sql



You can see some DB2 specific statements and then the 4 CREATE TABLE statements. We also have primary key definitions for the dimension tables. In the next step we will take this file and transform it into a BigSQL DDL file. We could do this in the command line but we will show the use of the Eclipse tooling to do it more easily. Just keep the location of the file in mind.

1. Finally we need to extract the data. The easiest form is delimited files which can be created with the db2 export or unload facility. We already did this for you, since this is a BigSQL and not a DB2 lab.
2. Switch back to your BIADMIN terminal window
3. Go to the lab directory cd /home/biadmin/labs/bigsql/
4. You can find the four delimited files here using ls
5. Take a look at one of the delimited files with more product\_dim.del



You can see the data for the product table we have investigated earlier. Columns are delimited with a “|” symbol and the dates have the european “dd-mm-yyyy hh:mm” format. (You can see this in the last line that has a value of 26 for the first two values) This will be important later.

15. BigSQL will load the files with the user bigsql to access them we need to either put them in HDFS or on an open directory. Copy them with cp \*.del /tmp

We have now created the DB2 DDL script and created the delimited files. In the next steps we will transform the DDLs into BigSQL. We will do this using the Eclipse Tooling.,although you could use command line tools like VIM as well(jsqsh).

# General Information about Hive

## 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 larges 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.

## 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).

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.

## 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)

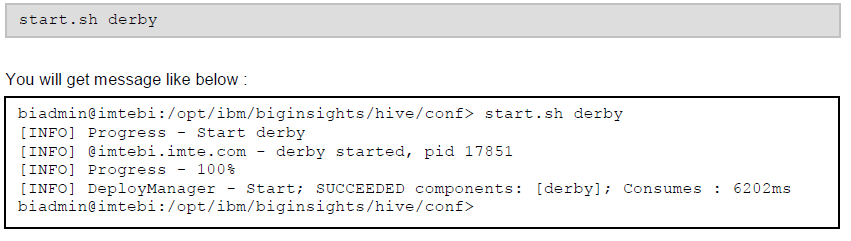
**Complex types:**

* arrays: ARRAY<data\_type>
* maps: MAP<primitive\_type, data\_type>

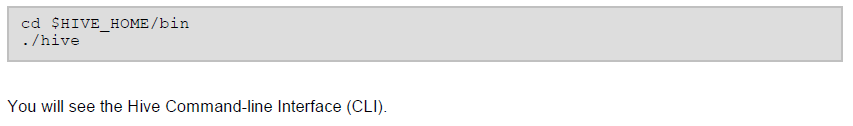
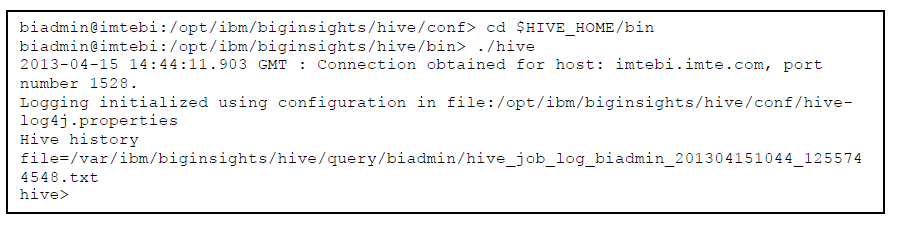
# Working with Hive

## Starting Hive

* 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 hive interactively. Change directory to $HIVE\_HOME/bin and launch Hive shell.



## 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.



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

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



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

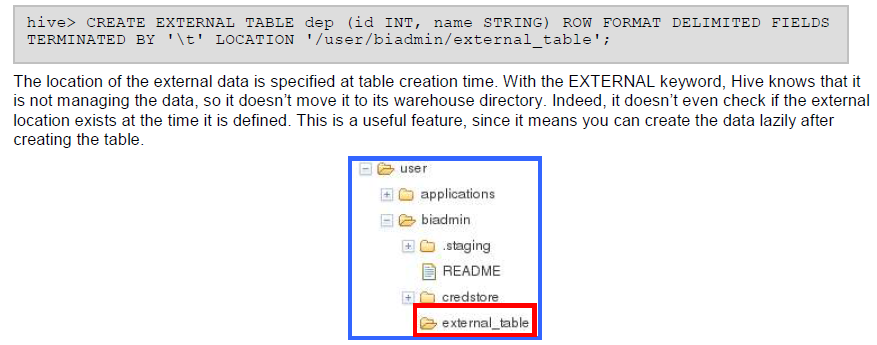
Console to check the directory.



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.



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 howthe 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.

## Altering and Browsing Tables

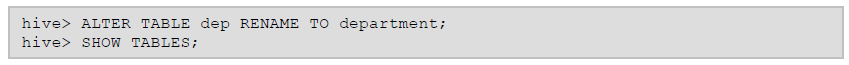
1. See the column details of the table.



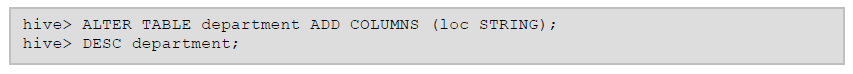
You should see the list of columns.



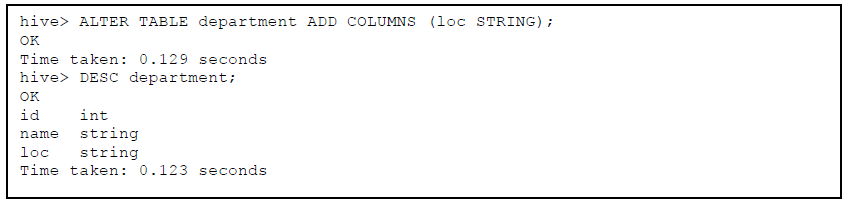
1. As for altering tables, table names can be changed and additional columns can be added or dropped.



3. Add a column to the table.



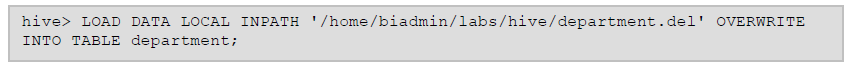
The new column ‘loc’ will be added to the table



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.

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



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.

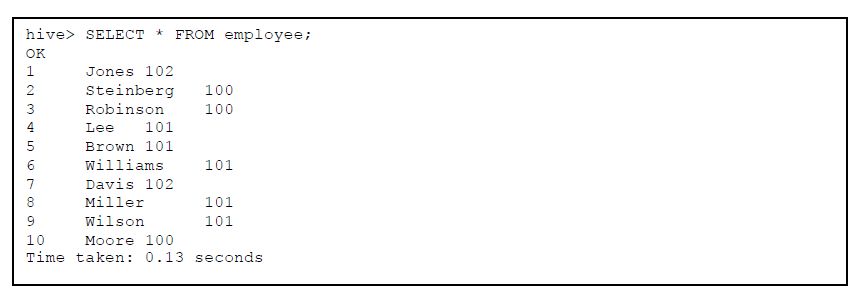
The keyword 'OVERWRITE' signifies that existing data in the table is deleted. If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.

## Simple Query

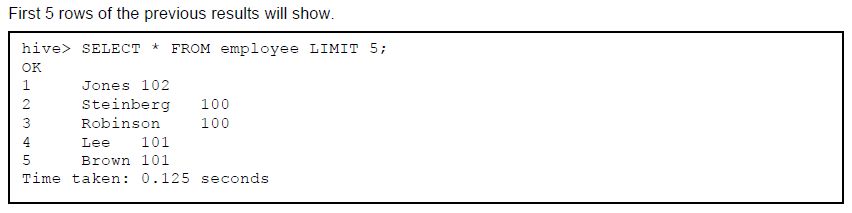
1. Select all columns and rows stored in employee table.



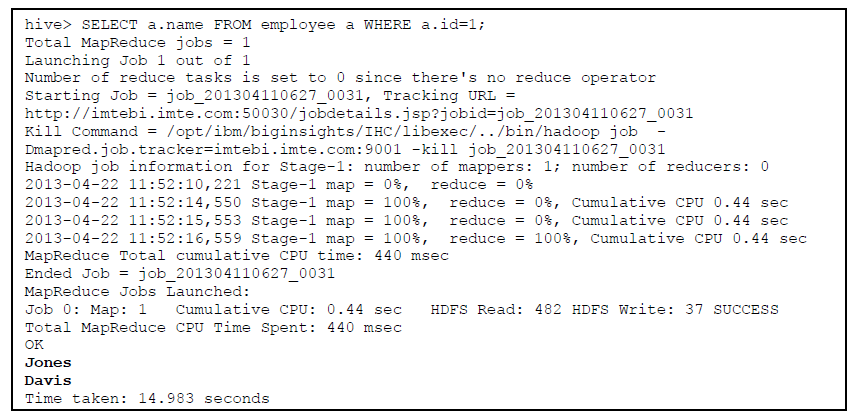
You will see the data loaded from employee.del file.



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



1. Filter the table.

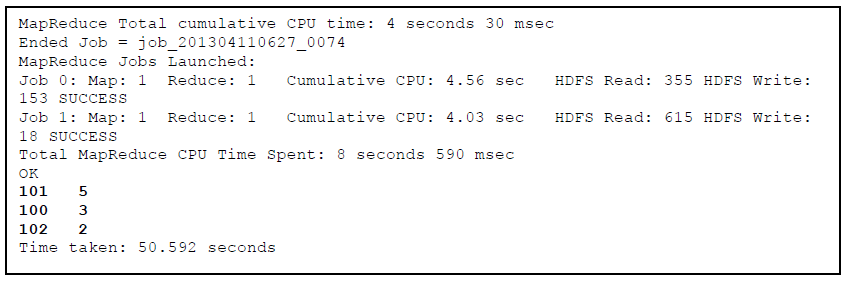


This query is internally compiled into a MapReduce job and fetches the requested data for us. We see that this simple conditional select query takes more time than a query on relational database. But, to gauge the actual performance, the framework has to be tested with huge data, during which MapReduce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

1. Use GROUP BY and ORDER BY clauses.



Result should be as below.



## 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

### 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.



1. Insert data into Hive table from a query.

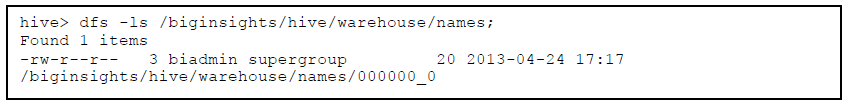


Query results can be inserted into tables by using the insert clause

1. See if we have a file created in HDFS



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

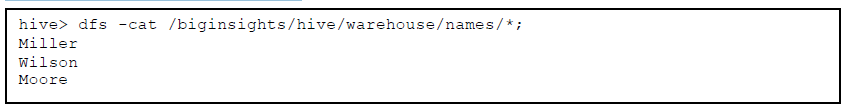


Note that the dfs command is build directly into the hive console so we do not need to switch to a different console to use hadoop fs.

1. Print the content of the file.



Expected contents are Wilson, and Moore.



### 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.



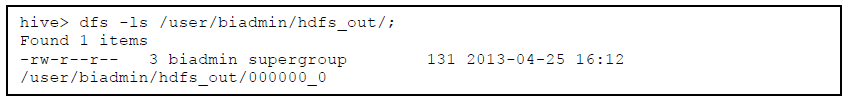
Query results can be inserted directly into file system directories.

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

1. List the content in the directory.



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



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.

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 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.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.

1. Create another table ‘emp\_pt’ for partitioning,



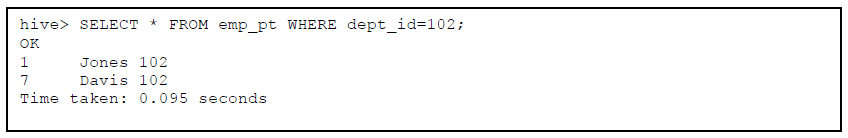
1. Insert data by loading it from the other table employee, and partition it by its department id.



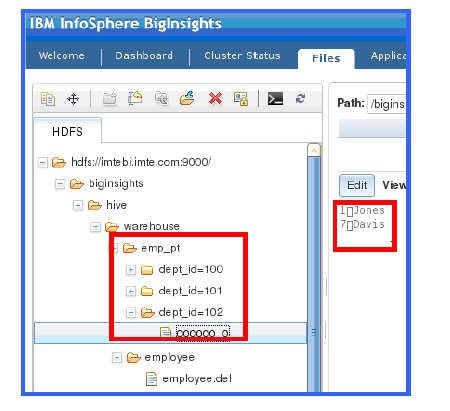
1. Run select statement.



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



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

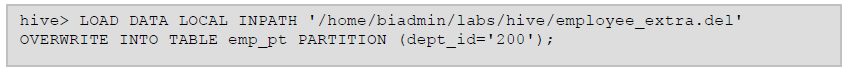


*emp\_pt* table is stored in multiple partitions unlike *employee*. Also, notice that the file doesn’t contain dept\_id data in it.

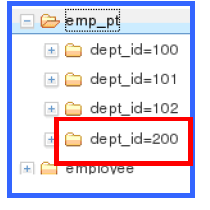
We mentioned it before that partition columns are virtual columns and they are not part of the data itself but are derived on load.

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.

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



1. You will notice in HDFS, another directory for dept\_id=200 partition is created.



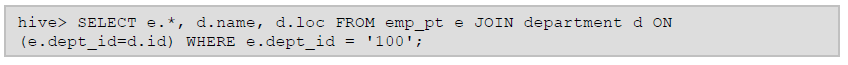
## Joins

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.

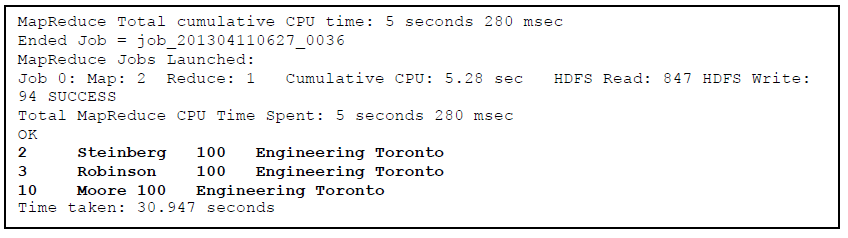
1. Select all from the department table.



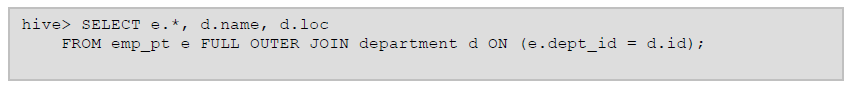
1. Insert data by loading it from the employee table, and partition employee table by its department id.



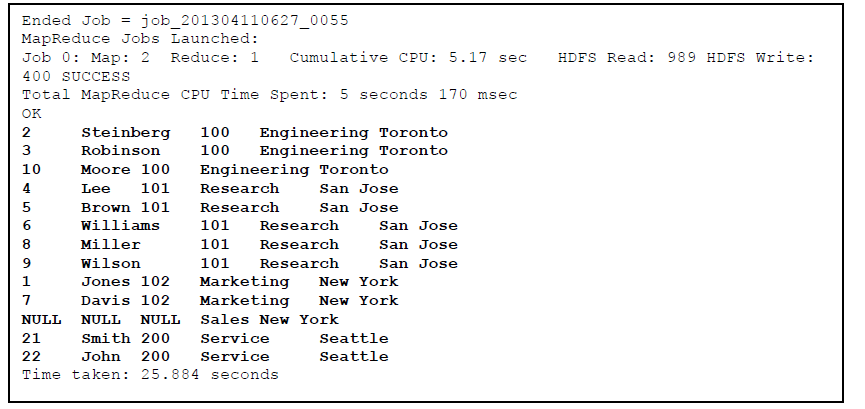
Result will look as below, and ones in bold are actual query results.



1. 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.

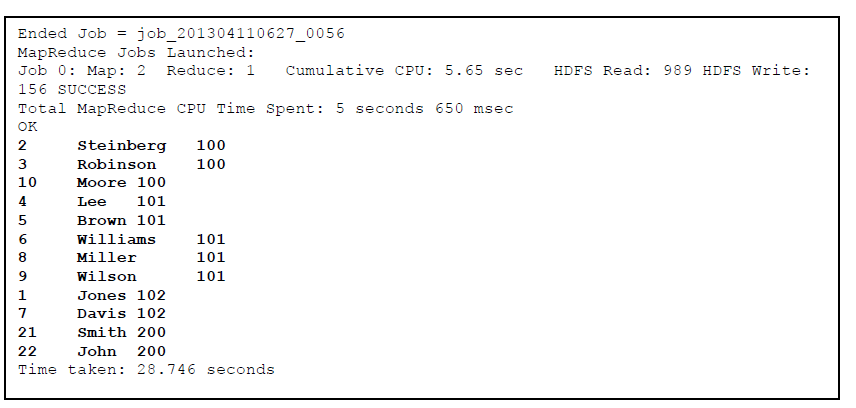
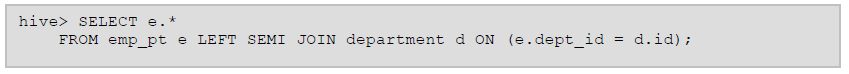


Result will be as below.



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

1. 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.



## Dropping Tables

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



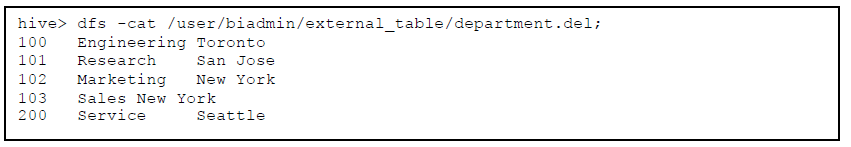
This table is a Hive managed table, thus the table information is removed from the metastore and the raw data is removed as if by 'hadoop dfs -rm'.

1. Drop an EXTERNAL table. (department)



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

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



1. Delete the content of the table. (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.

# 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