

Hive



Data and Analytics Training

Big Data – Self-Study Guide (206)

© Copyright 2014 Avanade Inc. All Rights Reserved. This document contains confidential and proprietary information of Avanade and may be protected by patents, trademarks, copyrights, trade secrets, and/or other relevant state, federal, and foreign laws. Its receipt or possession does not convey any rights to reproduce, disclose its contents, or to manufacture, use or sell anything contained herein. Forwarding, reproducing, disclosing or using without specific written authorization of Avanade is strictly forbidden. The Avanade name and logo are registered trademarks in the US and other countries. Other brand and product names are trademarks of their respective owners.

This study guide serves as a basis for your self-study on the basics of Hive. The study guide gives you an overview on the main topics and presents a number of internal and external resources you should use to get a deeper understanding of Hive, the related terms and technical details.

Please especially use the external resources provided. They are a main part of this study guide and are required to get a good understanding of the discussed topics.

Estimated Completion Time

3 hours (including time for external resources)

Objectives

After completing this study guide, you will be able to understand

- The concept behind Hive and HiveQL
- Internal and external Hive tables
- Different options of loading data into these tables
- Using remote desktop to execute HiveQL
- Using PowerShell to execute HiveQL
- Using Hive ODBC to work with Hive tables in Excel

Contents

- Introduction 4
- Understanding Hive..... 5
 - Type System..... 7
 - Internal and external tables 8
- Creating tables (CREATE)..... 8
- Loading Data (LOAD and INSERT) 9
- Querying tables (SELECT) 11
- Using Hive in HDInsight 12
- Using Hive in PowerShell 17
- Importing Data from HDInsight..... 19
 - Hive ODBC Driver..... 21
 - Configure ODBC DSN 21
 - Loading Data into Excel..... 23
- Conclusion 26

Introduction

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Built on top of Apache Hadoop, it provides

- Tools to enable easy data extract/transform/load (ETL)
- A mechanism to impose structure on a variety of data formats
- Access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase

Hive defines a simple SQL-like query language, called QL, that enables users familiar with SQL to query the data. At the same time, this language also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language. QL can also be extended with custom scalar functions (UDF's), aggregations (UDAF's), and table functions (UDTF's).

Hive does not mandate read or written data be in the "Hive format" — there is no such thing. Hive works equally well on Thrift, control delimited, or your specialized data formats.

Hive is not designed for OLTP workloads and does not offer real-time queries or row-level updates. It is best used for batch jobs over large sets of append-only data (like web logs). What Hive values most are scalability (scale out with more machines added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and UDF/UDAF/UDTF), fault-tolerance, and loose-coupling with its input formats.¹

We are going to talk about how to process big data with Hive in this module. As we understand it though, since the early modules of this course, we are talking about how to do Big Data analysis and get results back. But we haven't really been able to make a lot of sense of the result. In the Pig module was shown how to create delimited files out of your data, that we could do little bit of analysis on. But really a lot of people with experience in databases that want to get into Big Data want to do something that is more familiar to them and process the information in a way that is more like in traditional databases. In this module we are going to take a look at Hive, that will help us leverage some of the concepts we are already familiar with databases on Big Data results.

¹ <https://cwiki.apache.org/confluence/display/Hive/Home#Home-GeneralInformationaboutHive>

Hive allows us to project tabular schemas on to our data and then query those tabular schemas using the same semantics that you might use with a relational database like SQL Server SELECT statements.

Apache Hive provides a way of running MapReduce job through an SQL-like scripting language, called HiveQL. Hive is a data warehouse system for Hadoop, which enables data summarisation, querying, and analysis of large volumes of data. In this module, we use HiveQL to query a sample data file that is provided as part of HDInsight cluster provisioning.²

You can find more details on the Hive architecture here:

<https://cwiki.apache.org/confluence/display/Hive/Design>

Understanding Hive

Meta and data are two words that go well together and that really describes exactly what Hive is. It's a metadata service that projects a tabular schemas over an HDFS folders. If you are familiar with SQL Server databases, you will be familiar with the idea that you create a table and then insert data into that table. And if the data doesn't fit the tabular schema, it can't be inserted into the table. In this case you get an error when you try to insert the data that doesn't match the schema.

Hive is kind of different. What you've got here is a folder that contains files. And it could contain a whole hierarchy of subfolders that contain files. And then we're projecting a tabular schema on top of that folder. And the data isn't validated against that schema until we actually try to read from the schema, until we do a SELECT statement that brings data out of that table. At that point, the schema is projected onto the data and the data that fits the schema is brought out, data that doesn't fit the schema is not. The important point is that we do not force our data in any particular schema in order to process, the point is that we project a schema on to the data in order to process it.

This idea that we have files that are put into folders, could be a whole hierarchy of folders, but then we want to process that by using semantics that we are familiar with. We want to use SELECT statements, GROUP BY statements, JOIN and all those things that we do in Transact-SQL with SQL Server. And, that's what Hive enables us to do. It enables us to treat those files in those folders as if they were tables in relational database.

² <http://azure.microsoft.com/en-us/documentation/articles/hdinsight-use-hive/>

Whenever we run a query, whenever we do anything in HDInsight, it fundamentally boils down to MapReduce. So what happens under the covers is the Hive query engine takes our query, converts it to appropriate jar and uses that jar to run the MapReduce job against the data in the folders.

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. An example of a table could be `page_views` table, where each row could comprise of the following columns (schema):
 - `timestamp` - which is of `INT` type that corresponds to a unix timestamp of when the page was viewed.
 - `userid` - which is of `BIGINT` type that identifies the user who viewed the page.
 - `page_url` - which is of `STRING` type that captures the location of the page.
 - `referer_url` - which is of `STRING` that captures the location of the page from where the user arrived at the current page.
 - `IP` - which is of `STRING` type that captures the IP address from where the page request was made.
- 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. For example, a `date_partition` of type `STRING` and `country_partition` of type `STRING`. Each unique value of the partition keys defines a partition of the Table. For example all "US" data from "2009-12-23" is a partition of the `page_views` table. Therefore, if you run analysis on only the "US" data for 2009-12-23, 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 2009-12-23 does not mean that it contains all or only data from that date; partitions are named after dates 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.³

³ <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

Type System

Hive supports primitive and complex data types, as described below. Types are associated with the columns in the tables. The following Primitive types are supported:

- Integers
 - TINYINT - 1 byte integer
 - SMALLINT - 2 byte integer
 - INT - 4 byte integer
 - BIGINT - 8 byte integer
- Boolean type
 - BOOLEAN - TRUE/FALSE
- Floating point numbers
 - FLOAT - single precision
 - DOUBLE - Double precision
- String type
 - STRING - sequence of characters in a specified character set

Complex Types can be built up from primitive types and other composite types using:⁴

- Structs: the elements within the type can be accessed using the DOT (.) notation. For example, for a column c of type STRUCT {a INT; b INT} the a field is accessed by the expression c.a
- Maps (key-value tuples): The elements are accessed using ['element name'] notation. For example in a map M comprising of a mapping from 'group' -> gid the gid value can be accessed using M['group']
- Arrays (indexable lists): The elements in the array have to be in the same type. Elements can be accessed using the [n] notation where n is an index (zero-based) into the array. For example for an array A having the elements ['a', 'b', 'c'], A[1] retruns 'b'.

Using the primitive types and the constructs for creating complex types, types with arbitrary levels of nesting can be created. For example, a type User may comprise of the following fields:

- gender - which is a STRING.
- active - which is a BOOLEAN.

Please watch this 16 minutes video on Hive provided by Hortonworks:

<http://www.youtube.com/watch?v=Pn7Sp2-hUXE>

⁴ <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

Internal and external tables

The other key concept, and this is a really important thing to understand when we are dealing with Hive tables, is that tables can be internal or external. And that's got nothing to do with where on the file system the table points to. It has nothing to do with the location of the folder. When we say a table is internal, what we mean is, its lifetime and the lifetime of the folder on which it is mapped is managed internally by Hive. So if we drop the table, it would delete the folder and all of its contents. So it behaves much like a regular relational database, where if you drop the table you also inherently lose the data.

We can also create external tables in Hive. And when we create external table, if we drop the table, the table metadata is deleted, but the data stays where it is in the HDFS environment. And it turns out that is actually quite important for big data processing, because one of the things we will do is bring up a cluster, do the processing, get the results, and then get rid of the cluster. Of course, it will remove Hive and drop all the Hive tables. But if we have an external table, that data stays in the HDFS storage, which is, of course, Windows Azure blob store. We can actually get rid of the cluster, but retain the data, we can delete the HDInsight cluster, but keep the Windows Azure blob store and keep the data. We've still got perhaps the result of the processing we've done with Hive by creating multiple internal tables and one result table which could be an external table. Now we can drop all the internal tables, get rid of the HDInsight cluster altogether, but can keep the resulting data.

Creating tables (CREATE)

```
CREATE TABLE table1
(col1 STRING,
 col2 INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
```

Here is some of the syntax for creating a table. Here, we are creating an internal table, we haven't specified internal or external. The default is internal. col1 is a string, col2 is an int. There are a number of data types that are supported by Hive and, of course, there are different versions of Hive. The latest version has a part of the technology for Hive called Stinger. (As hives are related to bees, and stinger is related to bees, we are in a wonderful world of names in Hadoop). Stinger supports things like datetime data types and supports varchar data types. So more familiar data types if you are used to working with relational databases. The previous versions of Hive didn't support date, didn't support

varchar, but they did support things like strings, integers, floating kind of numbers, all types of things you come across day-to-day.

We've got this internal table, and it simply says it has the ROW FORMAT and fields are delimited by space. That's the only information given there. We haven't specified any particular location. And that is going to be stored in default location which is /hive/warehouse and then the name of the table, in this case, table1.

```
CREATE TABLE table2
(col1 STRING,
 col2 INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
STORED AS TEXTFILE LOCATION '/data/table2';
```

In the second example for table2, we have actually specified it to be stored as a text file and we have given it a location. So it is still an internal table. It will still remove that folder and all of the data if we drop the table. But we are choosing to store at location of our choosing rather than the default location.

```
CREATE EXTERNAL TABLE table3
(col1 STRING,
 col2 INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
STORED AS TEXTFILE LOCATION '/data/table3';
```

And table3 is an external table. With table3 if we were to drop that table, if we were to decommission the cluster, it automatically gets rid of the Hive metastore, but the data store remains on the disk. So we have got all our data in our Azure blob store.

Loading Data (LOAD and INSERT)

We discussed the creation of tables, we talked about idea of them containing data. And of course they don't contain data, it is the folders that contain the data. The tables just project the schema on to it. But how do we get the data into the folders that are referenced by Hive tables?

There is a very simple way, we can just simply save the files into those folders. And actually if the folder where we create a table already contains data, that data is already in the table. It is just simply creating a metadata abstraction over a folder, we can move the files to and from the folder in order to populate them. That works perfectly well.

There is also a LOAD statement, in HiveQL that we can use. And what that will do is, depending on the way how we set things up, it will move or copy the files from a location in HDFS to the folder for the table. It is really a good way of taking data (perhaps that's been staged) to a particular staging folder and just moving it into the appropriate folder for the table.

```
LOAD DATA LOCAL INPATH '/data/source' INTO  
TABLE MyTable;
```

The other option you've got is the INSERT statement. But that doesn't work the way you might expect. If you are a Transact-SQL programmer, you do INSERT statements, then typically you insert row into a table. What the INSERT statement does in HiveQL, is more like an INSERT FROM semantics you would have in Transact-SQL where you take the results of a query and insert the results of that query into another table.

What we actually do with INSERT FROM is, we have a syntax that allow us to specify source table and insert the results of a query from that into a destination table.

```
FROM StagingTable  
INSERT INTO TABLE MyTable  
SELECT Col1, Col2;
```

You can actually see the syntax here. Notice it is kind of the other way round, from how it might be in Transact-SQL. It is from a source table, insert into destination table and then a query that specifies the data you are going to get. And that is a really common way of processing data with Hive. One of the things, that is quite difficult for we as database people to come to terms with, is that, we are used to thinking of databases as being places to store and query data. And that makes perfect sense. But what big data is about, is about taking data, which is too big or unmanageable, and in wrong shape

or wrong structure, and then transforming that into something we can work with. So quite often what we do with Hive is rather than just treat the tables, as somewhere to store the data and query it, we use Hive as a means of projecting a schema on the source data and then using query to transform that into the shape that we want. Quite often, you create the table on top of your source, you then use an insert statement to transform that source into another table, perhaps then into another table and so on until you get a format you want. And that's when you can do your analysis on. So that's kind of how it works. Think of it as being partly like a database for storing data, but it is almost partly an ETL system where we are transforming the data as a sequence of queries.

Querying tables (SELECT)

```
SELECT Col1, SUM(Col2) AS TotalCol2
FROM MyTable
WHERE Col1 >= '2013-06-01' AND Col1 <= '2013-06-30'
GROUP BY Col1
ORDER BY Col1;
```

When we are going to query these tables, we use the SELECT statement. It is pretty much the same as in any ANSI SQL language. And what that does is, it translate it into MapReduce jobs, then apply the tabular schema to the underlying data that brings out the data from there that we need. In the example here, it is a pretty typical SELECT statement. SELECT col1 and then SUM(Col2) as a total from the table. We've put in a WHERE clause filtering on a range of dates. And then we GROUP BY and ORDER BY, much as you would do with Transact-SQL. There's nothing really there that would be unfamiliar to a SQL Server programmer or any other database person.

To understand more details on the HiveQL and how hive commands relate to SQL commands, please have a look at this SQLtoHive cheat sheet by Hortonworks:

<http://hortonworks.com/wp-content/uploads/downloads/2013/08/Hortonworks.CheatSheet.SQLtoHive.pdf>

You should be able to understand and code the different commands and the resultset created by these queries and commands.

Using Hive in HDInsight

We've already talked about it enough. Now it's time to see it in action. We're going to dive and see how it all fits together.

In this sample you will use an HDInsight query that analyzes website log files to get insight into how customers use the website. With this analysis, you can see the frequency of visits to the website in a day from external websites, and a summary of website errors that the users experience.⁵

In this tutorial, you'll learn how to use HDInsight to:

- Connect to an Azure Storage Blob containing website log files
 - Create Hive tables to query those logs
 - Create Hive queries to analyze the data
 - Use Microsoft Excel to connect to HDInsight (using an ODBC connection) to retrieve the analyzed data
1. Connect the remote desktop session to the cluster.
 2. Let's see the data we are going to work with. The website log data is available under */HdiSamples/WebsiteLogSampleData/SampleLog* path. To confirm, open the Hadoop Command Line on the remote desktop and execute the following command:

```
C:\>hadoop fs -ls /HdiSamples/WebsiteLogSampleData/SampleLog
Found 1 items
-rwxrwxrwx 1      273431 2014-11-13 07:05 /HdiSamples/WebsiteLogSampleData/Samp
leLog/909f2b.1log
C:\>
```

If you open the log file to look at it, it is a typical log file from IIS. The data in it is semi-structured. The log data is actually structured, but the 2 header rows at the top (that start with # character) that don't fit the same tabular structure.

3. Bring up the Hive environment by typing the below command. Here, %HIVE_HOME% is the system variable that has been set to the installation folder for Hive. This folder is usually having a long name with version number, and to save the torture of remembering this long name, we have this system variable set for us.

```
C:\>%HIVE_HOME%\bin\hive_
```

⁵ https://<your_cluster_name>.azurehdinsight.net/GettingStarted/Tutorials/WeblogAnalysis/Introduction

4. Create an internal table that allows Hive to query data stored in Azure Blob Storage

```
CREATE TABLE weblogs_raw

(s_date date, s_time string, s_sitename string, cs_method string, cs_uristem string, cs_uriquery
string, s_port int, cs_username string, c_ip string, cs_useragent string, cs_cookie string, cs_referer
string, cs_host string, sc_status int, sc_substatus int, sc_win32status int, sc_bytes int, cs_bytes int,
s_timetaken int )

ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
```

It looks a pretty regular create table statement. We have specified that the fields are delimited by space, which is how the log file is structured. This table will be created in the default location.

5. To verify this, go to another Hadoop command line and run this command:

```
C:\>hadoop fs -ls /hive/warehouse
Found 2 items
drwxr-xr-x  - hdinternaluser supergroup          0 2014-11-13 04:26 /hive/ware
house/hivesampletable
drwxr-xr-x  - hdp supergroup                    0 2014-11-13 08:22 /hive/ware
house/weblogs_raw
```

We see a folder created under */hive/warehouse* that has the same name as the name of the table we just created. This folder has been created to hold the data for this table.

6. Return back to the Hive environment. Execute the below command to see the tables that already exist.

```
hive> show tables;
OK
hivesampletable
weblogs_raw
Time taken: 0.078 seconds, Fetched: 2 row(s)
hive> _
```

What we see is that we get our *weblogs_raw* table. And there is an existing *hivesampled* table. That's provided with every HDInsight cluster.

- Now we want to create another table and we are going to use that for the cleaned up log. When we get rid of the header rows and clean all that data up, we want to store the results in a different table called *weblogs_clean*. This table would be stored in the location in HDFS as specified in the query.

```
CREATE EXTERNAL TABLE weblogs_clean
(s_date date, s_time string, s_sitename string, cs_method string, cs_uristem string, cs_uriquery
string, s_port int, cs_username string, c_ip string, cs_useragent string, cs_cookie string, cs_referer
string, cs_host string, sc_status int, sc_substatus int, sc_win32status int, sc_bytes int, cs_bytes int,
s_timetaken int )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
STORED AS TEXTFILE LOCATION '/data/cleanlog/';
```

This is an external table. So the lifetime of *cleanlog* folder is managed by the HDFS and not by Hive. So, if we drop this table, it won't delete the folder. This gives us a little bit of more control over the lifetime of the data and the actual files. Because we know we are going to do something on this after the processing.

- We can verify our table has been created:

```
hive> show tables;
OK
hivesampletable
weblogs_clean
weblogs_raw
Time taken: 0.078 seconds, Fetched: 3 row(s)
hive>
```

- Switch to the Hadoop command line, and verify the creation of *cleanlog* folder:

```
C:\>hadoop fs -ls /data
Found 1 items
drwxr-xr-x - hdp supergroup 0 2014-11-13 09:30 /data/cleanlog
```

- Right now we don't have data in our *weblogs_raw* table. We would like to take data that is in our *SampleLog* folder (refer step 2), and load it in our *weblogs_raw* table. To do that, go back to Hive environment.

```
LOAD DATA INPATH '/HdiSamples/WebsiteLogSampleData/SampleLog' INTO TABLE  
weblogs_raw;
```

It loads the data, that is move the log file in the background to the weblogs_raw folder in HDFS.

11. Just to prove the point, if we go to our filesystem again and examine the SampleLog folder, we see the folder no longer exists.

```
C:\>hadoop fs -ls /HdiSamples/WebsiteLogSampleData/SampleLog  
ls: `/HdiSamples/WebsiteLogSampleData/SampleLog': No such file or directory
```

12. And if we look at weblogs_raw folder, we see the log file has been moved:

```
C:\>hadoop fs -ls /hive/warehouse/weblogs_raw  
Found 1 items  
-rwxrwxrwx 1 273431 2014-11-13 09:40 /hive/warehouse/weblogs_raw/909f2b.1o  
g
```

So, by executing the Hive statement, what it actually did was manipulated the file on the filesystem.

13. Return to Hive environment. What we are more interested in is if we can query the table and get some information out of it. We'll take data from the weblogs_raw and insert into the weblogs_clean, but we are going to get rid of any rows where there is a # sign in the beginning. In essence, this will get rid of all those comment rows that are at the beginning.

```
FROM weblogs_raw  
  
INSERT INTO TABLE weblogs_clean  
  
SELECT *  
  
WHERE SUBSTR(s_date, 1, 1) <> '#';
```

This query will take all of the rows from the raw table and insert them into the clean table where the first character of the column s_date is a #. So, it is projecting the schema, and of course that s_date column has whatever in the string before it hits the first space. So the cases

where the rows don't quite fit the schema, it will just fit it the best it can. And the first column will have the # symbol and we are excluding them in the query.

Because there is a SELECT statement, on running this it executes a MapReduce job. It is all about MapReduce.

14. What we should now be able to do, is actually select some rows from that clean log table.

```
SELECT s_date, COUNT(*) AS PageHits, SUM(sc_bytes) AS BytesReceived, SUM(cs_bytes) AS BytesSent  
  
FROM weblogs_clean  
  
GROUP BY s_date  
  
ORDER BY s_date;
```

Again, it is a SELECT. So we get a MapReduce job. And we get our results on the screen showing the page hits on that particular date and bytes sent and received on the server.

15. Let's see the effect of what happens when we start manipulating those tables and drop them and so on. We'll drop the weblogs_raw table first. It has done its job, it is really just a staging folder for our raw log file.

```
hive> drop table weblogs_raw;  
OK  
Time taken: 1.117 seconds  
hive>
```

During deletion of the table, it is actually deleting the folder. It was an internal table.

16. If we go now and have a look at the default location where that table was, we don't find its folder anymore

```
C:\>hadoop fs -ls /hive/warehouse  
Found 1 items  
drwxr-xr-x - hdpinternaluser supergroup          0 2014-11-13 04:26 /hive/warehouse/hivesampletable
```

So, it has actually deleted the folder and everything in it.

17. Now, let's compare that with what happens if we drop the clean log table:

```
hive> drop table weblogs_clean;  
OK  
Time taken: 0.853 seconds
```

This is comparatively fast, because all it had to do was get rid of metadata that describes the table.

18. The actual data is still in the HDFS. If we go and have the look

```
C:\>hadoop fs -ls /data/cleanlog  
Found 1 items  
-rw-r--r--  1 RemoteUser supergroup    272771 2014-11-13 09:59 /data/cleanlog/  
000000_0
```

We can see there's a file in it which was the output when we ran the INSERT statement. So the table is gone. The metadata that projected the schema on it has gone, but the data still remains.

In theory, we can take the entire cluster down now, leave our storage account up, the data will still be there and next week when we need to create that data again, we can bring our cluster back up, reconnect to the same storage and we can actually go and just simply run the CREATE TABLE statement, pointing to the existing folder where the data is already stored. And actually it will behave exactly as it did before. So, if we repeat our query on this table, it runs the MapReduce job and we get the results just the same as before. Despite having dropped the table, we didn't get rid of the data, we recreated the table, and the data is still there for anybody to use.

Additionally to the above example you may want to work through the tutorial provided by Hortonworks and using the Hortonworks sandbox environment:

<http://hortonworks.com/hadoop-tutorial/using-hive-data-analysis/>

Using Hive in PowerShell

We've done all of this in the desktop and we want to look doing this remotely using PowerShell. So, we talked in previous modules about using PowerShell for doing various things. We talked about using PowerShell to run a MapReduce job. And we had our cmdlet which was called New-AzureHDInsightMapReduceJobDefinition for doing MapReduce. Well there is a New-AzureHDInsightHiveJobDefinition cmdlet that we can use for Hive as well.

We can use it in exactly the same way that we did with generic MapReduce and with Pig, if we recall, with similar sort of the approach. We can do the same thing with Hive. We can create a job definition,

and we can point that out to a HiveQL query or we can point it out to a file that contains our HiveQL script. So it is exactly the same as we did with Pig in the previous module.

And we start the job with the Start-AzureHDInsightJob cmdlet.

If you've got, perhaps a series of things that should happen using HiveQL statement to manipulate the data, that's the way to do it. Put it all in a file, create it as a script, upload that file to HDFS and then you are just going to use this approach to execute that and you get all the metadata information of what's being happening as it executes and prints to the PowerShell environment.

What if you just want to query a table? You want to very quickly go in and run a query that brings back some data perhaps. Well, there is a kind of lighter-weight approach to that as well. There is a Invoke-Hive cmdlet. And with Invoke-Hive cmdlet, it gives us a very quick and easy way just to go and execute some HiveQL against the cluster. So, we can use either of these techniques from within our PowerShell environment.

1. Go to the client machine
2. Open up PowerShell and execute the following script:

```
$clusterName = "<your_cluster_name>"

$hiveQL = "DROP TABLE webactivity; CREATE EXTERNAL TABLE webactivity(log_date
STRING, page_hits INT, bytes_recvd INT, bytes_sent INT) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ' ' STORED AS TEXTFILE LOCATION '/data/webactivity'; FROM
weblogs_clean INSERT INTO TABLE webactivity SELECT s_date, COUNT(*), SUM(sc_bytes),
SUM(cs_bytes) GROUP BY s_date ORDER BY s_date;"

$jobDef = New-AzureHDInsightHiveJobDefinition -Query $hiveQL
$hiveJob = Start-AzureHDInsightJob -Cluster $clusterName -JobDefinition $jobDef
write-Host "HiveQL job submitted..."
wait-AzureHDInsightJob -Job $hiveJob -waitTimeoutInSeconds 3600
Get-AzureHDInsightJobOutput -Cluster $clusterName -JobId $hiveJob.JobId -
StandardError
```

The Hive query is doing three things, (1) it is dropping any existing table by the name webactivity, (2) it creates an external table, (3) runs the same query we had see previously to calculate the network traffic and puts that result in this webactivity table.

So, what we've done is from PowerShell remotely connected to our cluster, run this HiveQL query and what that actually has done is, if you think back to our bigger picture what we are trying to do with big data is take it from something which is big and unwieldy and unmanageable and reshape it into

something that is useful. So that's what this Hive job has done. It has taken that huge volume of data in our log file, and it has aggregated that down into something that I might want to work with.

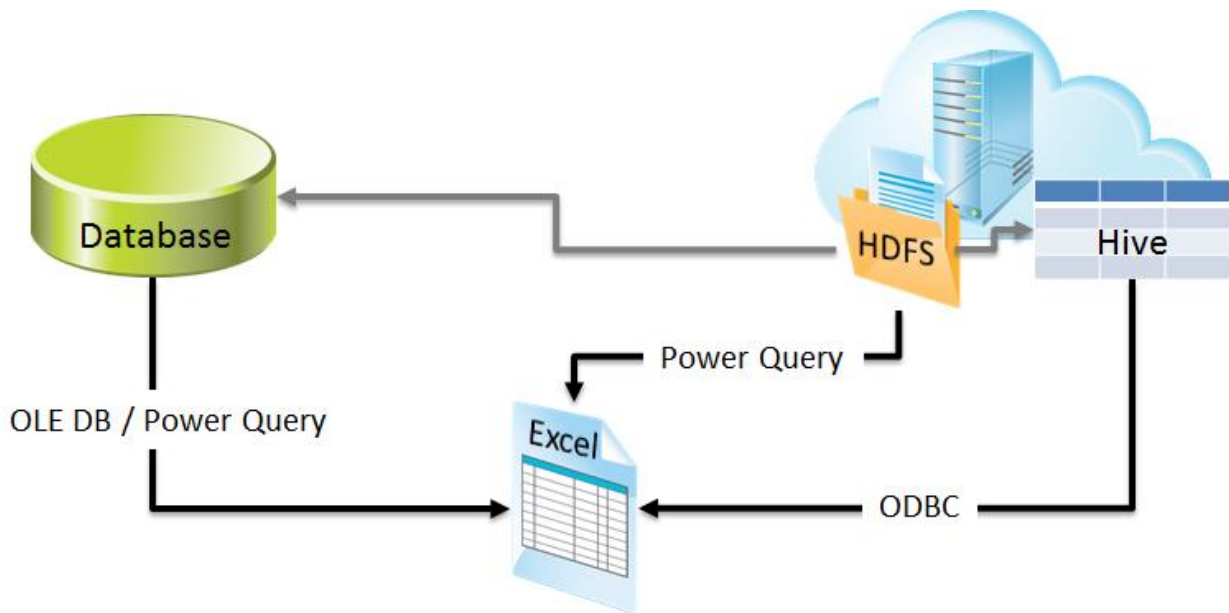
The great thing now is we can go and see the results. And because the results are in the table, we can actually query that table using Hive. So, let's have a look at how to do that. We could use the same technique, we could use the `AzureHDInsightHiveJobDefinition` in PowerShell. But there is a simpler way, a lighter way.

```
$clusterName = "<my_cluster_name>"  
$hiveQL = "SELECT * FROM webactivity;"  
Use-AzureHDInsightCluster $clusterName  
Invoke-Hive -Query $hiveQL
```

It is a much smaller script here. It is connecting to the cluster and pulls data from the table. On executing, it connects to the cluster and submits the Hive query. And it then enters into a MapReduce job. And finally gives the result of the query.

Importing Data from HDInsight

We can get the results of our big data processing from HDInsight into Excel. And it turns out there are number of paths we can use to actually do that. So, let's explore each of those on the figure below and have a look and see how that works.



We've got this notion of a Hive table and HDFS folder up here in our cluster. So, HDInsight cluster is up here in our cloud, and we've got an HDFS folder and have a Hive table based on that folder. So, we've seen that pretty much as the pattern as we've gone through the course.

Down underneath there's Excel where we are consuming the data. And one of the options that we've got is that we could use ODBC to go and query a Hive table. Just the same we would query a table with any other database source, we can connect to our HDInsight cluster and literally import data from tables and create queries and so forth. The technology that is used to connect is ODBC, because there is an ODBC driver for Hive that makes it easy to set up a data source connection and go and run that query.

Now that pre-supposes, of course, the cluster is up and running, because we need it running to service the Hive queries. Because when we run a Hive query, what actually happens is the Map/Reduce job. But what if we want to take the cluster down? What if we are finished with the processing, we want to take the cluster down, but we have left our data in our Windows Azure blob store that we want a hold of.

We have an option there, we can use Power Query to go and connect to HDFS folder in the blob store, and can bring that data down into Excel using Power Query. So rather than going through the Hive table, we can access the files directly in HDFS folder structure and then Power Query provides all sorts of extra capability for us to do further filtering and reshaping and aggregation of our data.

And then finally we've got the option of doing our data processing in HDInsight, but then exporting the results of that to a database. Just a regular SQL Server database or Windows Azure SQL database

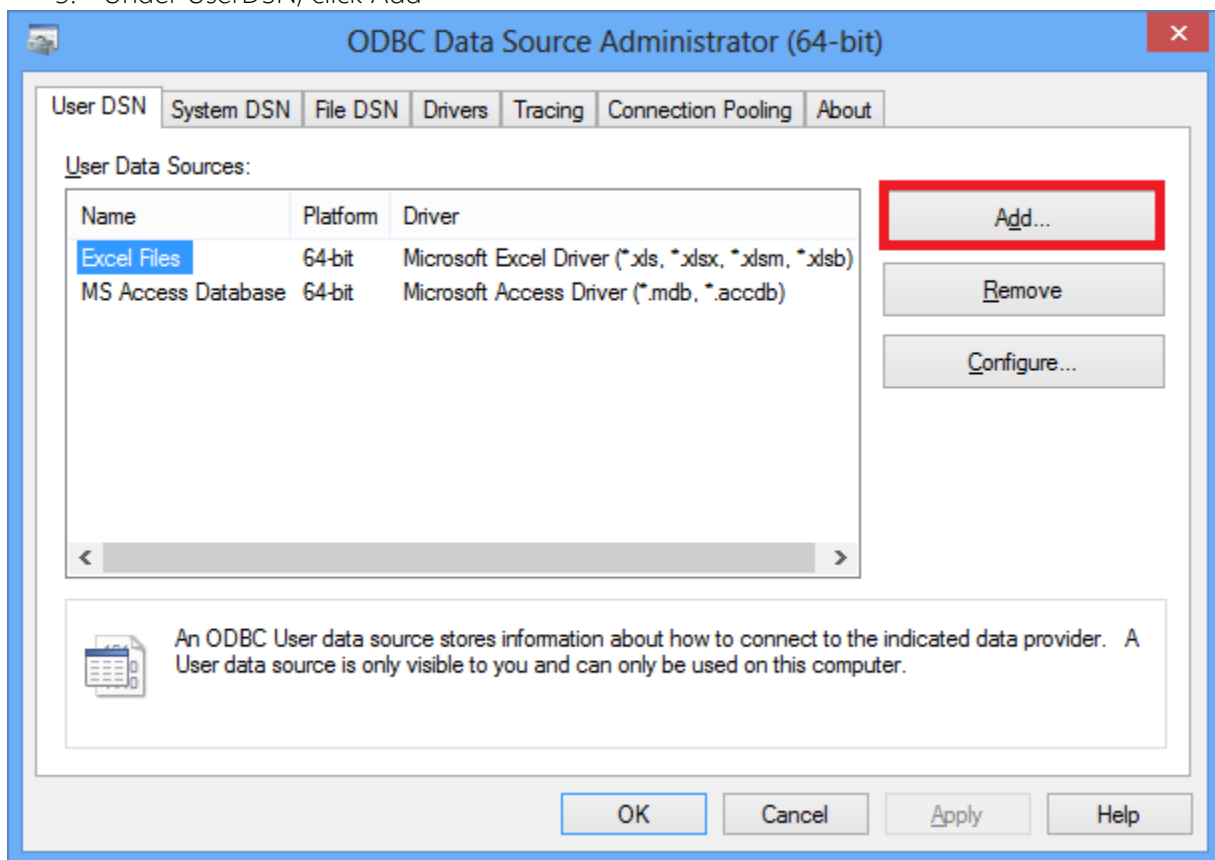
or someother database source we want to use. And then we can tear the cluster down, we don't need that anymore. And an Excel user could connect to the database where we put the results and bring the queries down from there.

Hive ODBC Driver

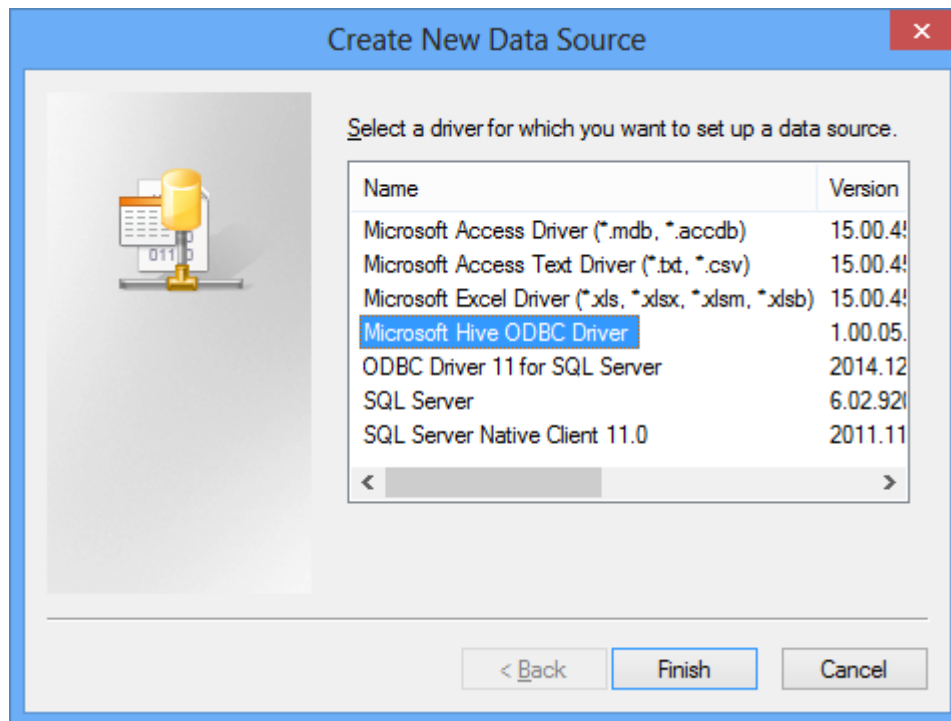
Let's dinally look at Hive ODBC driver. There's an ODBC driver that we can download and install from the Microsoft download centre. And once we've installed that driver we can create a data source name, which is usually the easiest way to configure ODBC connections. So, we create our data source name, and then can use that in the data connection wizard in Excel. And we can either use that directly in the regular Excel import data option or in PowerPivot within Excel to go and get our data.

Configure ODBC DSN

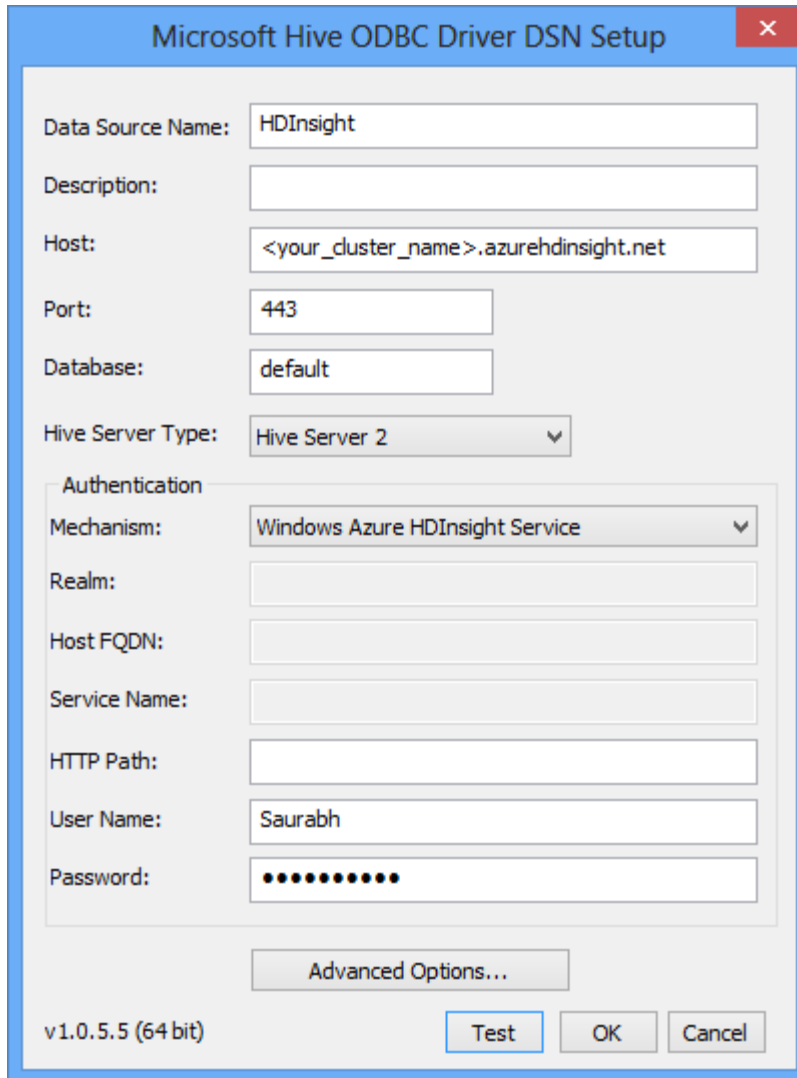
1. Download and install [Microsoft Hive ODBC Driver](#)
2. Configure ODBC driver by going to Administrative Tools | ODBC Data Sources
3. Under UserDSN, click Add



4. In the Create New Data Source window, select Microsoft Hive ODBC Driver and click Finish.



5. Give a name to the data source, fill in your host details, admin credentials for your cluster and test your connection. Once successful, click OK.



Microsoft Hive ODBC Driver DSN Setup

Data Source Name: HDInsight

Description:

Host: <your_cluster_name>.azurehdinsight.net

Port: 443

Database: default

Hive Server Type: Hive Server 2

Authentication

Mechanism: Windows Azure HDInsight Service

Realm:

Host FQDN:

Service Name:

HTTP Path:

User Name: Saurabh

Password:

Advanced Options...

v1.0.5.5 (64 bit)

Test OK Cancel

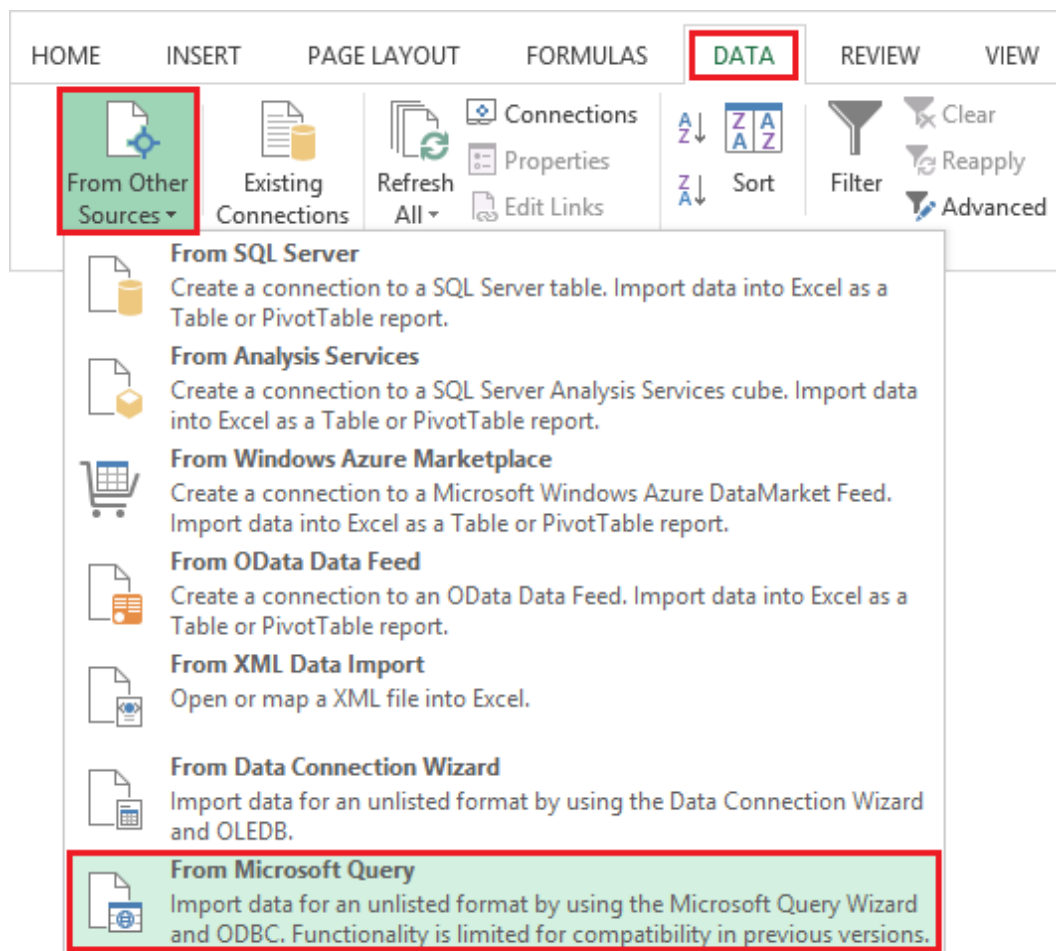
We are connecting to the default database. Hive can connect to multiple databases in the cluster. This completes the configuration of ODBC DSN called HDInsight. Lets now look at Excel.

Loading Data into Excel

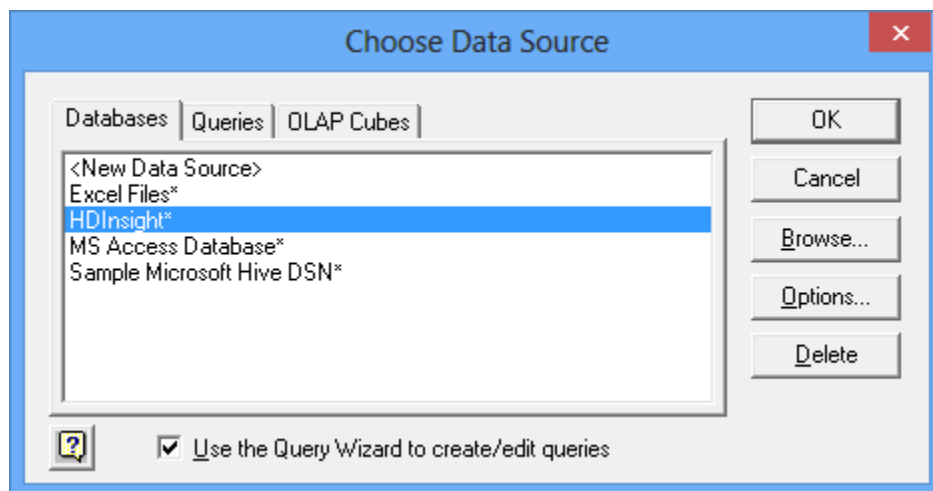
⁶You can use the Microsoft Hive ODBC Driver to import data from Hive into Excel. Use the following steps to connect to the table.

1. Open Excel and create a blank workbook
2. From the Data tab, select From Other Sources, and then select From Microsoft Query.

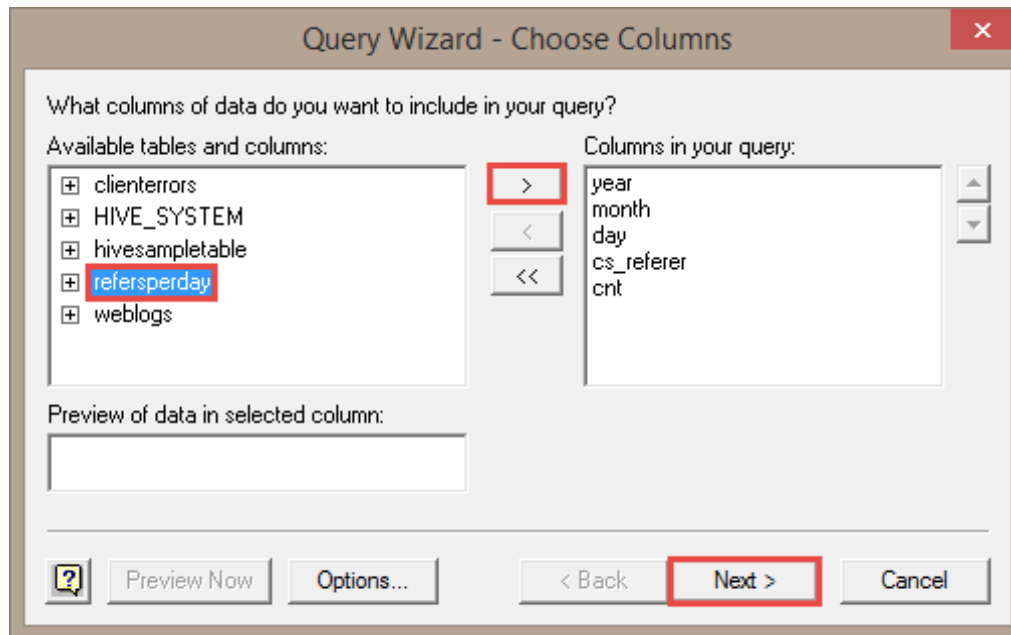
⁶ https://<your_cluster_name>.azurehdinsight.net/GettingStarted/Tutorials/WeblogAnalysis/ExcelLoading



- When prompted to Choose Data Source, select the HDInsight DSN we created in previous steps.



4. Enter the password for the cluster in the connection dialog box that comes and hit OK.
5. In the Query Wizard, select the `refersperday` table, and then select the > button.



6. Click Next to continue through the wizard, until you reach a dialog with a Finish button. Click Finish.
7. When the Import Data dialog appears, click OK to accept the defaults. After the query completes, the data will be displayed in Excel.

	A	B	C	D	E
1	year	month	day	cs_referer	cnt
2	2014	10	26	-	24
3	2014	10	26	http://weblogs.asp.net/archive/2007/02/26/tip-trick-url-rewriting-with-asp-net.aspx	8
4	2014	10	26	http://weblogs.asp.net/archive/2008/01/07/dynamic-linq-part-1-using-the-linq-dynamic-ql	7
5	2014	10	26	http://weblogs.asp.net/archive/2007/04/06/tip-trick-enabling-ssl-on-iis7-using-self-signed	5
6	2014	10	26	http://weblogs.asp.net/archive/2007/06/29/linq-to-sql-part-3-querying-our-database.aspx	4
7	2014	10	26	http://weblogs.asp.net/archive/2007/08/16/linq-to-sql-part-6-retrieving-data-using-stored	4
8	2014	10	26	http://weblogs.asp.net/archive/2007/05/19/using-linq-to-sql-part-1.aspx	3
9	2014	10	26	http://weblogs.asp.net/archive/2007/09/07/linq-to-sql-part-9-using-a-custom-linq-express	3
10	2014	10	26	http://weblogs.asp.net/archive/2008/11/24/new-asp-net-charting-control-It-asp-chart-run	3
11	2014	10	26	http://weblogs.asp.net/archive/2007/05/29/linq-to-sql-part-2-defining-our-data-model-cla	2
12	2014	10	26	http://weblogs.asp.net/archive/2007/08/23/linq-to-sql-part-7-updating-our-database-using	2
13	2014	10	26	http://weblogs.asp.net/archive/2007/12/09/asp-net-mvc-framework-part-4-handling-form	2
14	2014	10	26	http://weblogs.asp.net/archive/2010/09/18/important-asp-net-security-vulnerability.aspx	2
15	2014	10	26	http://anhndt.wordpress.com/	1
16	2014	10	26	http://asp.net.vn/Default.aspx?tabid=439f6137-d2ce-4d77-9df3-589775e08a36&mid=206962	1
17	2014	10	26	http://barnamenevis.org/showthread.php?95001-1001-%D9%86%DA%A9%D8%AA%D9%87-'	1
18	2014	10	26	http://barnamenevis.org/showthread.php?95001-1001-%D9%86%DA%A9%D8%AA%D9%87-'	1
19	2014	10	26	http://chicasharp.net/result2.aspx?target=LINQ+to+SQL+{E3%83%91E3%83%BC%E3%83%	1
20	2014	10	26	http://forum.vsoftgroup.com/showthread.php/18111-Mo-hinh-MVC-trong-ASPnet	1
21	2014	10	26	http://go4answers.webhost4life.com/Example/set-roles-particular-user-dynamically-94902	1
22	2014	10	26	http://http://localhost/	1
23	2014	10	26	http://lpcbaby.blog.163.com/blog/static/383238872007112711751381/	1
24	2014	10	26	http://my.opera.com/hivietnam/blog/?startidx=20	1

So we used the Hive ODBC driver to connect to the Hive table and pull down data from it into our Excel.

Conclusion

Hive is a great technology for doing Map/Reduce if you are familiar with SQL syntax. If the way you think about things is in terms of set based operations and tables and querying then it is a great way to do that Map/Reduce processing. One of the things that you have to get used to is the idea that it is not about storing your data and just querying it like you are doing on your database. Typically you are performing some changes to your data, you are transforming from a huge amount of data that is perhaps difficult to work with into a smaller subset of data that is perhaps aggregated, or perhaps just splitting off different values. Or filtered in some way. It is quite often the case if you create the table that you staged your data into then you use that to populate other tables you are going to use for your analytics.

We've only really scratched the surface of Hive in this module. We've shown you the fundamentals of how to use Hive, create tables and drop them and differences between internal tables and external tables. The key thing to remember is that a Hive table is just a metadata projection over a folder. It is not actually a structure for storage. It is projected when you read the data.

There are some other interesting things you can do with Hive. For example, you can partition your tables, and if you do that what will happen under the cover is Hive would generate sub folders. So if we partition a folder by year, it will create a folder for each year underneath the top level folder. And if we then further partition by month, for each year we would have month folders underneath there. So you can create hierarchies of folders to more efficiently store the data and give you more efficiencies when you query it. There is some pretty sophisticated stuff going on with Hive. And there is also indexing support in Hive. With small amounts of data indexing will make little difference, with large amounts of data you can actually see some serious performance gains.

You've got this notion of using Hive as a way of restructuring your data to something that's useful and using it to query that data using SQL semantics or you can just simply use Hive to transform that data into the files you need and then drop the cluster and you are left with the data files. And you can do with those whatever you need.

Then the final thing we showed you was, you can use PowerShell, just as you can with all the other end components that we had looked at so far to go and connect and run Hive statements on the cluster and do all that remotely from the client. You actually script out the entire Hive processing using PowerShell.

Exciting stuff, it is really amazing how powerful that is when you consider the setup of massive quantities of data. And it takes little bit time to run the process but what the analytics would be able to do over massive quantities of data is really-really impressive.