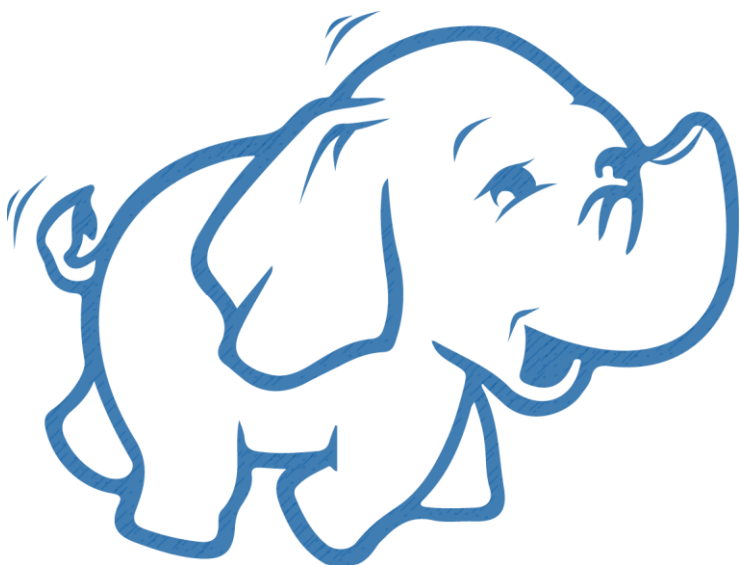




Understanding Hive Storage formats in Microsoft Azure HDInsight



Change management

| Version | Date Effective | Incorporated Changes | Requested By |
|---------|----------------|----------------------|-----------------|
| 1.0 | 06/01/2016 | Initial Version | Nishant Thacker |
| 1.1 | 01/29/2017 | TechReady 24 | Nishant Thacker |

Introduction

Hive is a data warehousing system that simplifies analyzing large datasets stored in Hadoop clusters, using SQL-Like language known as HiveQL. Hive converts queries to either map/reduce, Apache Tez or Apache Spark jobs.

To highlight how customers can efficiently leverage HDInsight Hive to analyze big data stored in Azure Blob Storage, this document describes how to optimize hive queries with file formats, SerDes, Compression, Partitioning and HiveQL optimization techniques. It is presumed that you have some prior understanding of basic Hive and HiveQL concepts.

The lab uses web transaction log data of an imaginary bookstore for the explanation purpose. All of the hive queries will be run against this data.

Takeaways

After completing this lab, you will

1. Understand different Hive File Formats and Compression.
2. Understand and Use custom SerDes.
3. Learn to use table partitioning to improve query performance.
4. Learn to use HiveQL optimization techniques to improve query performance.

Prerequisites

Azure Account Requirements

While carrying out all exercises within this hands-on lab, you will make use of the **Azure Preview portal** from <https://portal.azure.com/>.

To perform this lab, you will require a Microsoft Azure account.

If you do not have an Azure account, you can request for a free trial version by visiting <http://azure.microsoft.com/en-us/pricing/free-trial/>.

Within the one-month trial version, you can perform other SQL Server 2014 hands-on labs along with other tutorials available on Azure.

NOTE: - To sign up for a free trial, you will need a mobile device that can receive text messages and a valid credit card.

*Make sure you follow the **Roll back Azure changes** section at the end of this exercise after creating the Azure database so that you can make the most of your \$200 free Azure credit*

TechReady24 special instructions

HDInsight cluster usually take 15-20 minutes to create. As a work around to the cluster create time, we've pre-provisioned HDI clusters for this lab and are sharing the credentials below. Note that these clusters are only active for TechReady and will be deleted after the event, so if you're trying the labs after TR, you should create your own clusters and use the cluster properties to proceed with the lab.

These credentials are shared in good faith and the understanding is that attendees will not misuse these for any purposes, including but not limited to this lab. If you have any concerns, please close this lab now and do not proceed any further.

TechReady24 Cluster Credentials:

Note: The steps in the following section 'Provision HDInsight Linux Hadoop cluster with Azure Management Portal' should be ignored if you are provided a shared cluster. For TechReady you're provided a cluster with the following credentials:

Hive Cluster:

Cluster Name for Hive Cluster: nthdilabhive

Cluster URL (Ambari) for Hive Cluster: <https://nthdilabhive.azurehdinsight.net/>

Username: admin

Password: HDItut@123

Azure Credentials, if needed:

Username: analyticsdemo@outlook.com


Password: HDItut@123

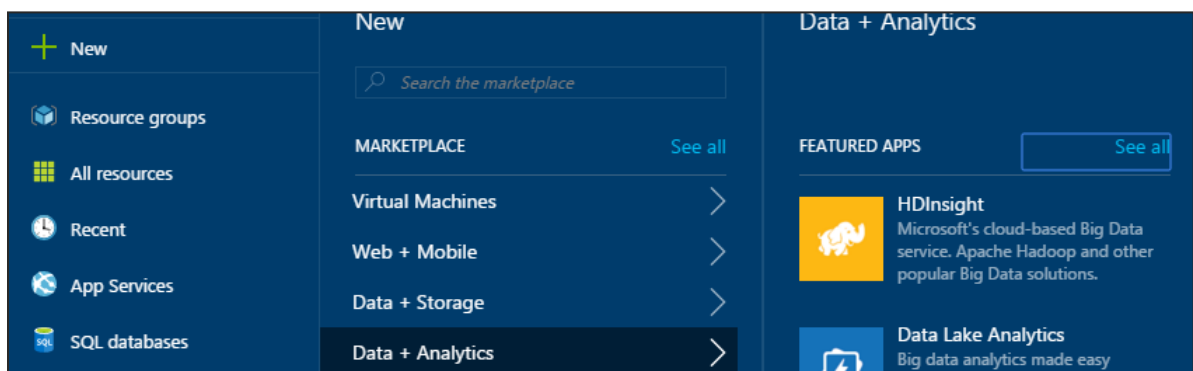
Class

Provision HDInsight Linux Hadoop cluster with Azure Management Portal

To provision HDInsight Windows Hadoop cluster with Azure Management Portal, perform the below steps.

Note: These steps should be ignored if you are provided a shared cluster.

1. Go to the Azure Preview Portal by clicking the **Preview Portal** link  **Preview Portal** on the IE favorites bar. Login using your azure account credentials.
2. Select **NEW -> Data Analytics -> HDInsight**



3. Enter or select the following values.
 - a. **Cluster Name:** Enter the cluster name. A green tick will appear if the cluster name is available.
 - b. **Cluster Type:** Select Hadoop as the cluster type.
 - c. **Cluster Operating System:** Select Linux as the cluster operating system.
 - d. **Version:** Select 3.5 as the cluster version.
 - e. **Cluster Tier:** Select the **Standard** cluster tier

- f. **Subscription:** Select the Azure subscription to create the cluster.
- g. **Resource Group:** Select an existing resource group or create a new resource group.
- h. **Credentials:** Configure the username and password for HDInsight cluster and the SSH connection. SSH connection is used to connect to HDInsight cluster through a SSH client such as Putty.

- i. **Data Source:** Create a new storage account and a default container.

- j. **Node Pricing Tiers:** Set the number of head node and worker nodes as shown below.

Note: You can select lowest pricing tier A3 nodes or reduce the number of worker nodes decrease the cluster cost.

- k. Leave other configuration options as default and click **Create** to provision HDInsight Hadoop cluster. It will take 15-20 minutes for cluster provisioning.

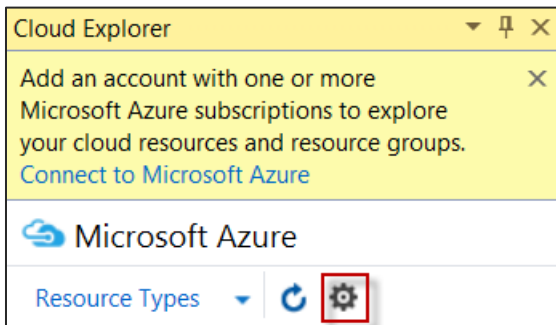
The HDInsight Linux Hadoop cluster is now ready to work with.

You can install HDInsight tools for visual studio using Web Platform Installer (<http://go.microsoft.com/fwlink/?linkid=255386&clcid=0x409>). HDInsight tools for visual studio are packaged with the Azure SDK for .Net. Install the one that matches your visual studio version.

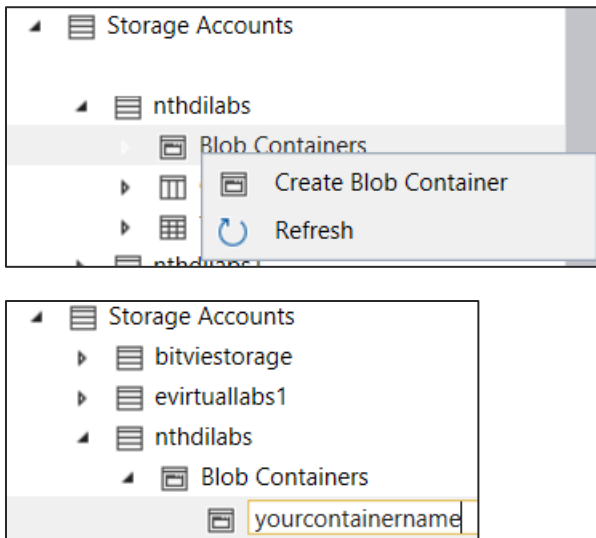
Create a new Azure Storage Container

In this step, you'll create a new storage container. This container will host the hive tables created in the later steps in the lab. To create a new storage container, follow the below steps.

1. Open visual studio, select View from the top menu and then select Cloud Explorer. Click on **Connect to Microsoft Azure** in the yellow tool tip or click on the settings icon (highlighted in red) to login to your Azure account.



2. On the cloud explorer, expand **Storage Accounts** node. This will list all the storage accounts in your Azure account. Expand **nthdilabs** storage account and right click on **Blob Containers** node. Select **Create Blob Container**. Enter your name as the blob container name (all small letters).



You'll use this blob container to store the hive tables created in later steps in the lab.

Note: If you are using your own HDInsight cluster and not **nthdilabhive**, your storage account name will be different.

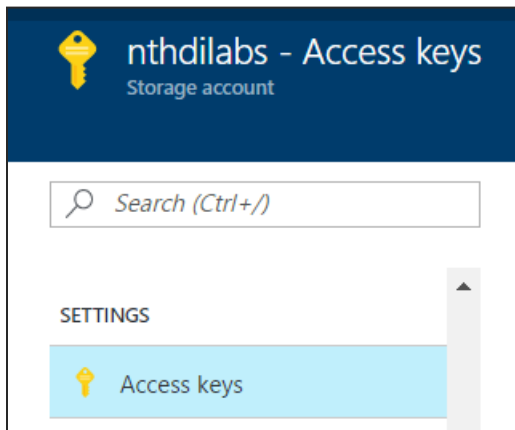
Copy lab data to the storage account

In this section, you'll copy the files required for the lab to your storage account. You'll copy the files between two storage account with the help of AzCopy utility. You can download the utility from here <http://aka.ms/downloadazcopy>

Note: These steps should be ignored if you are provided a shared cluster.

To copy the files, follow the below steps.

1. Copy your Azure Storage account access keys. This is required to copy data from the source Azure Storage account to your Azure Storage account. To get your storage account access key, navigate to your storage account on the Azure Management Portal and select **Access keys** under **Settings**.



2. Click on the copy icon to copy **Key1** from the **Access Keys** pane.

| NAME | KEY | |
|------|---|---|
| key1 | onLLmMI4n9zhQ0OgOzwyabeneOJbEcIVih/nTX+jzh8bpgYpTL |   ... |
| key2 | UDEidwroMarY3Iyisw6zKr7s2UQ/yMD1vaKi1D/4cp7EHgKwgin |   ... |

3. Press Window + R to open the run window. Type cmd and press enter to open a new command console window.
4. Change the directory to C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy.
5. Copy and paste the following command on the console window to transfer **staging_weblogs.csv** file from the source storage account to your storage account.

```
AzCopy /Source:https://nthdilab.blob.core.windows.net/hadooplabs/Lab1/
/Dest:https://nthdilabs.blob.core.windows.net/nthdilabllap/<yourname>/bookstore/
staging_weblogs/ /SourceKey
G1t6T13jn3K6w/4ZS2NG7RsTHe5YuusRLd9tKnRlJku7cjCwcRk5JxWAHmtrH1Dt03+nwttYbB2HuvHe
I/UiNw==
/DestKey:QfPYhbGxokkcqjjEZIkzbzyDdMb7QHSrUjA6UnpfuLjC0Np4PSSjkfrsnVhGyOxJsKWEK9C
XNvPZo/L1w7T0ow== /Pattern:staging_weblogs.csv
```

Note: Replace <yourstorageaccount> with the name of your storage account created in section "Provision HDInsight Hadoop Cluster with Azure Management Portal" and <yourstorageaccountkey> with your storage account access key copied in step 2.

6. Copy and paste the following command to copy the **customerfixed.csv** from the source storage account to your storage account.

```
AzCopy /Source:https://nthdilab.blob.core.windows.net/hadooplabs/Lab1/
/Dest:https://nthdilabs.blob.core.windows.net/nthdilabllap/<yourname>/booksto
re/ /SourceKey
G1t6T13jn3K6w/4ZS2NG7RsTHe5YuusRLd9tKnRlJku7cjCwcRk5JxWAHmtrH1Dt03+nwttYbB2Hu
vHeI/UiNw==
/DestKey:QfPYhbGxokkcqjjEZIkbzzyDdMb7QHSrUjA6UnpfuLjC0Np4PSSjkfrsnVhGyOxJsKWE
K9CXNvPZo/L1w7T0ow== /Pattern:customerfixed.csv
```

Note: Replace <yourname> with your name or a unique ID, so that your files do not get processed by other jobs.

7. Copy and paste the following command to copy the **customerjson.csv** from the source storage account to your storage account.

```
AzCopy /Source:https://nthdilab.blob.core.windows.net/hadooplabs/Lab1/
/Dest:https://nthdilabs.blob.core.windows.net/nthdilabllap/<yourname>/bookstore/
/SourceKey
G1t6T13jn3K6w/4ZS2NG7RsTHe5YuusRLd9tKnRlJku7cjCwcRk5JxWAHmtrH1Dt03+nwttYbB2HuvHe
I/UiNw==
/DestKey:QfPYhbGxokkcqjjEZIkbzzyDdMb7QHSrUjA6UnpfuLjC0Np4PSSjkfrsnVhGyOxJsKWEK9C
XNvPZo/L1w7T0ow== /Pattern:customerjson.csv
```

Note: Replace <yourname> with your name or a unique ID, so that your files do not get processed by other jobs.

Hive File Formats

A file format dictates how records are stored in a file. The file formats differ mainly by data encoding, disk I/O, compression rate and space utilization. In this section, we'll cover different file formats and their effect on query performance.

The query files are in HiveStorageFormats solution. Navigate to

C:\LabAssets\HDI\HiveStorageFormats\HiveStorageFormats folder and double click **HiveStorageFormats.sln** to open the solution.

In the solution explorer, click on **setup.hql** file. The script creates a hive database and tables in different hive file format as required for the lab execution. The description of the script is given below.

Create Database

```
-- Create database. Replace YourName with your fullname.
-- for example, if your name is John Doe then
-- the database name will be HDILABDB_johndoe
set newdb=HDILABDB_YourName;
DROP DATABASE IF EXISTS ${hiveconf:newdb};
CREATE DATABASE ${hiveconf:newdb};
```

Note: The above query creates a new hive database. All of the tables will be created in this database. Follow the instructions in the comments to modify the script before you execute it.

Staging Table

The query creates an external hive table for the web transaction log of an imaginary book store. The staging table is used to load data into other other tables used in the lab. The schema for the bookstore web transaction log is given below.

| Column | Description |
|-----------------|---|
| TransactionId | Unique Id assigned to a transaction |
| TransactionDate | The date of the transaction |
| CustomerId | Unique Id assigned to the customer |
| BookId | Unique id assigned to a book in the book store |
| PurchaseType | 1. Purchased: Customer bought the book 2. Browsed: Customer browsed but not purchased the book. 3. Added to Cart: Customer added the book to the shopping cart |
| OrderId | Unique order id |
| BookName | The name of the book accessed by the customer |
| CategoryName | The category of the book accessed by the customer |
| Quantity | Quantity of the book purchased. Valid only for PurchaseType = Purchased |
| ShippingAmount | Shipping cost |
| InvoiceNumber | Invoice number if a customer purchased the book |
| InvoiceStatus | The status of the invoice |
| PaymentAmount | Total amount paid by the customer. Valid only for PurchaseType = Purchased |
| City | The name of the city where the purchase is made. |
| State | The name of the state where the purchase is made. |

```
-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;

USE ${hiveconf:newdb};

-- create the staging table
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.staging_weblogs(
transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenumbr varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50) ,
    city varchar(50) ,
    state varchar(10)
) ROW FORMAT DELIMITED FIELDS TERMINATED by ',' lines TERMINATED by '\n'
```

```
STORED AS TEXTFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/staging_web
logs/';
```

Text Format

This is the default and the most common file format. Each row in text file is a data row with columns separated by a delimiter such as ",", "|" and many more. The structure depends on the column or the field order.

```
-- Text Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.text_weblogs(
    transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenummer varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50) ,
    city varchar(50) ,
    state varchar(10)
) ROW FORMAT DELIMITED FIELDS TERMINATED by ',' lines TERMINATED by '\n'
STORED AS TEXTFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/text_weblog
s/';
```

Note: The above query will create an external hive table stored as text format.

The storage or the file format is specified at the end of the table as "STORED AS". If nothing is specified, text file format is considered as a default storage option.

Text file compressed with Gzip or Bzip2 format can be directly imported as hive table. However, this has performance impact as hadoop can't split the compressed file into multiple blocks and run multiple maps in parallel.

```
-- Creates a compressed table
-- The gzip folder already has compressed file.
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.gzip_weblogs(
    transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
```

```

        invoicenumber varchar(50) ,
        invoicestatus varchar(50) ,
        paymentamount varchar(50) ,
        city varchar(50) ,
        state varchar(10)
    ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines TERMINATED BY '\n' LOCATION
    'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/gzip/';

```

Note: The above query will create an external hive table with GZip compression.

Sequence File

Compressing raw text files saves storage, however compressed files are not split able mainly because they can be only read from the beginning. Therefore, compressed files can't be broken down in parts and processed in parallel by multiple mappers.

The sequence file format can break the file into blocks (1 MB default block size) and can optionally compress the job output with one of the following options None, Record and Block. In Record compression, only value is compressed and in Block compression, a block of key and value is compressed. Block size is configurable.

Sequence file consists of binary key/value pairs which is split able and therefore can be processed in parallel by multiple mappers.

```

-- Sequence Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.seq_weblogs (
    transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenumber varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50) ,
    city varchar(50) ,
    state varchar(10)
) STORED AS SEQUENCEFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/sequence/';

```

Note: The above query will create an external hive table stored in sequence format.

Record Columnar(RC) File

Record Columnar (RC) file stores data in a column oriented way instead of row oriented way. This helps when a table has large number of columns and most queries use only few of the columns. In a row oriented approach, entire row will be scanned to get the required few columns however, in the column oriented only the required columns are scanned.

The rows are first partitioned horizontally into row splits, the vertically in a columnar way. The metadata of a row split is stored as key record and all the row data as value record.

Column oriented storage has better compression compared to row oriented storage. This is because in column oriented store, data being compressed is of same data type, for ex, compressing a column Bookname where all values are of string data type. This is further improved when a column has low cardinality i.e. few distinct values.

```
-- RC Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.rc_weblogs(
    transactionid varchar (50) ,
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenumber varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50) ,
    city varchar(50) ,
    state varchar(10)
) STORED AS RCFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/rcfile/';
```

Note: The above query will create an external hive table stored in rc format.

It's not possible to insert data directly into a table with RC file format. The data to be inserted should already be in a different table. RC file format is optimized for select queries however, writing data to RC file consumes more resources than the row oriented storage.

Optimized Record Columnar (ORC) File

ORC file format is a column oriented storage format that is optimized version of RC file format. It provides much better compression and faster query performance than RC file format. ORC file groups row data into stripes with stripe metadata stored in a file footer. Default stripe size is 250 MB.

The file footer contains list of stripes, number of rows per stripe, and column data type. It also contains light weight indexes with column level aggregations sum, max, min and count. At the end of a file, a postscript contains compression parameters and size of the compressed footer.

```
-- ORC Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.orc_weblogs(
    transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenumber varchar(50) ,
```



```

        invoicestatus varchar(50) ,
        paymentamount varchar(50) ,
        city varchar(50) ,
        state varchar(10)
    ) STORED AS ORC LOCATION
    'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/orcfile/'
    tblproperties ("orc.create.index"="true");

```

Note: The above query will create an external hive table stored in orc format with inline index set to true.

Parquet File

This is another columnar file format to be used with multiple hadoop projects such as Hive, Pig, Impala, Drill and Crunch. Parquet supports efficient compression and encoding schemes. It allows for specifying compression per column level and is scalable enough to add additional encodings as and when they are developed and implemented.

```

-- PARQUET Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.parq_weblogs(
    transactionid varchar(50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenumbr varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50) ,
    city varchar(50) ,
    state varchar(10)
) STORED AS PARQUET LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/parquet/';

```

Note: The above query will create an external hive table stored in parquet format.

The compression levels allowed are UNCOMPRESSED, GZIP and SNAPPY.

Avro File Format

Avro is a data serialization format which stores data in binary format. The data type definition is stored in file in Json format. It also contains markers to help map reduce job to split large files into multiple smaller files for faster parallel processing. Avro file format is smaller in size than the raw or text file.

Avro supports schema modification as opposed to Parquet and ORC file format. For example, if a new field is added Following is an example of an Avro schema.

```

{
  "type": "record",
  "name": "Employee",
  "fields": [
    {"name": "userName", "type": "string"},
    {"name": "Salary", "type": ["null", "long"]}
  ]
}

```

```

    {"name": "Profile",          "type": {"type": "array", "items": "string"}}
  ]
}

```

```

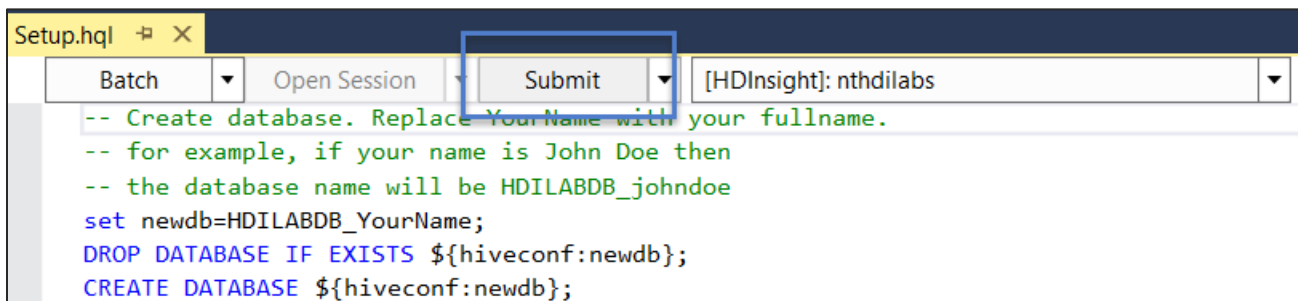
-- Avro Format
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.avro_weblogs(
  transactionid varchar (50),
  transactiondate varchar(50) ,
  customerid varchar(50) ,
  bookid varchar(50) ,
  purchasetype varchar(50) ,
  orderid varchar(50) ,
  bookname varchar(50) ,
  categoryname varchar(50) ,
  quantity varchar(50) ,
  shippingamount varchar(50) ,
  invoicenumbr varchar(50) ,
  invoicestatus varchar(50) ,
  paymentamount varchar(50) ,
  city varchar(50) ,
  state varchar(10)
) STORED AS AVRO LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/avro/';

```

Note: The above query will create an external hive table stored in avro format.

Execute Setup.hql

To execute the query, click on the dropdown on the top left corner of the query window. If asked, enter your Azure credentials in the resulting window. Select the HDInsight Cluster created earlier from the drop down and click on **Submit** button.



Comparing performance of different file formats

We have seen multiple file formats available in Hive. In this section we'll compare the performance of the discussed file format.

*Note: To check the execution time of a query, refresh the **Job View** window until query completes. Click on the **Job Log** at the bottom of the Job View window. Scroll to the end of the job log to get the query execution time*

Write Performance

In this section, we'll load data into the tables created above and note down the execution time.

In the solution explorer, click on **load_text_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into text_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
INSERT OVERWRITE TABLE ${hiveconf:newdb}.text_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

In the solution explorer, click on **load_seq_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into seq_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
INSERT OVERWRITE TABLE ${hiveconf:newdb}.seq_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

Note: The above query inserts data into seq_weblogs from staging_weblogs

In the solution explorer, click on **load_rc_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into rc_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
INSERT OVERWRITE TABLE ${hiveconf:newdb}.rc_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

Note: The above query inserts data into rc_weblogs from staging_weblogs

In the solution explorer, click on **load_orc_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into orc_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
INSERT OVERWRITE TABLE ${hiveconf:newdb}.orc_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

Note: The above query inserts data into orc_weblogs from staging_weblogs

In the solution explorer, click on **load_parq_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into parq_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
```

```
INSERT OVERWRITE TABLE ${hiveconf:newdb}.parq_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

Note: The above query inserts data into parq_weblogs from staging_weblogs

In the solution explorer, click on **load_avro_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Insert data into avro_weblogs table from staging_weblogs table
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
INSERT OVERWRITE TABLE ${hiveconf:newdb}.avro_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs;
```

Note: The above query inserts data into avro_weblogs from staging_weblogs

The below table lists down the execution time and the table size for the above created tables.

| Table Name | Execution Time (sec) | Total Size (MB) |
|--------------|----------------------|-----------------|
| text_weblogs | 50 | 123 |
| seq_weblogs | 46 | 123 |
| rc_weblogs | 45 | 123 |
| orc_weblogs | 52 | 20 |
| parq_weblogs | 48 | 34 |
| avro_weblogs | 43 | 138 |

Note: The above statistics are not the standard values and may vary depending on the type of data and the cluster configuration.

Fastest to write is the text format. The **PARQUET**, **ORC** and **AVRO** require additional parsing which increases the overall write time. **ORC** and **PARQUET** file formats occupy less space to store the data.

Read Performance

In this section, we'll read data from the different tables and note down the execution time. We'll run same select query against all of the tables. The query returns total revenue per category.

Execute the following queries one by one and note the execution time.

In the solution explorer, click on **read_text_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
```

```
FROM ${hiveconf:newdb}.text_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from text_weblogs table

In the solution explorer, click on **read_seq_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
FROM ${hiveconf:newdb}.text_seq_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from seq_weblogs table

In the solution explorer, click on **read_rc_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
FROM ${hiveconf:newdb}.rc_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from rc_weblogs table.

In the solution explorer, click on **read_orc_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
FROM ${hiveconf:newdb}.orc_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from orc_weblogs table.

In the solution explorer, click on **read_parq_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
FROM ${hiveconf:newdb}.parq_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from parq_weblogs table.

In the solution explorer, click on **read_avro_weblogs.hql** file. Click Submit to execute the query. Note the execution time of the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
```

```
set newdb=HDILABDB_YourName;
Select categoryname,Sum(Quantity) As quantitysold
FROM ${hiveconf:newdb}.avro_weblogs WHERE PurchaseType="Purchased" GROUP BY
CategoryName ORDER BY QuantitySold Desc;
```

Note: The above query reads data from avro_weblogs table.

The below table lists down the time taken to query the tables.

| Table Name | Execution Time (sec) |
|--------------|----------------------|
| text_weblogs | 45 |
| seq_weblogs | 50 |
| rc_weblogs | 48 |
| orc_weblogs | 44 |
| parq_weblogs | 42 |
| avro_weblogs | 51 |

Note: The above statistics are not the standard values and may vary depending on the type of data and the cluster configuration.

Fastest to read are the **PARQUET** and **ORC**, as the query only selects two columns out of the fifteen columns in the table. **AVRO** is a row oriented format and is optimized for scanning and returning large number of columns.

SerDes

SerDes is a short form for serializer/deserializer. Hive uses SerDes and File Format to read and write to tables. Users can write data to HDFS in any format. SerDes parse the unstructured bytes in a record, stored in a file, in a format that can used with hive. Serialization method is called when writing data to hive and Deserialization method is called when reading the data from hive. Custom SerDes can be written in Java to read and write custom record formats.

Hive comes in with many built in SerDes such as RegexSerde, Json SerDe, Avro SerDe and CSV SerDe.

In the solution explorer, click to open **regex_serde.hql**. Click Submit to execute.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;
```

```

CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.serde_customer_fixed
(
    customerid STRING,
    email STRING,
    firstname STRING,
    lastname STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES("input.regex" = "({50})({50})({50})({50})")
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/customerfixed/';

CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.customer_fixed
(
    customerid STRING,
    email STRING,
    firstname STRING,
    lastname STRING
)
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/customerfixed/';

```

The query creates two tables `serde_customer_fixed` and `customer_fixed`. The table `serde_customer_fixed` uses `regex_serde` to specify four fixed width columns of length 50.

In the solution explore, click to open **read_regex_serde.hql**. Click Submit to execute the query.

```

-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;
select customerid from ${hiveconf:newdb}.serde_customer_fixed Limit 1;
select customerid from ${hiveconf:newdb}.customer_fixed Limit 1;

```

Note: The query selects a row from `serde_customer_fixed` table and `customer_fixed` table. The `serde_customer_fixed` table uses `RegexSerde` to parse fixed width columns.

Once the job completes, click on Job Output link at the bottom of Job View window.

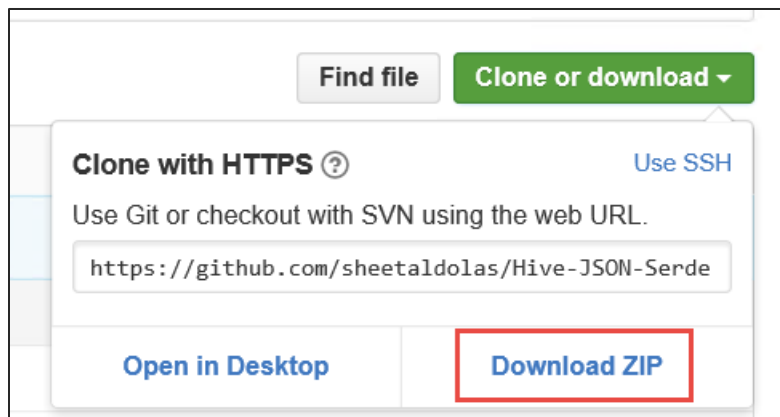
| Job Output | | | |
|------------|-----------------|------|-----------|
| JRK07IRCIJ | | | |
| JRK07IRCIJ | annef@yahoo.com | Anne | Fernandez |

The values are correctly stored in `serde_customer_fixed` table, however, in the `customer_fixed` table all of the values are stored in the first column, which is `customerid`.

Use custom Json Serde to read and write Json documents

To use a custom Json Serde to parse Json documents in hive, perform the below steps

1. Download the custom Json Serde from here <https://github.com/sheetaldolas/Hive-JSON-Serde/tree/master>.



2. Right click on the **Hive-JSON-Serde-master.zip** and select "Extract Here". Copy the extracted folder to C drive.
3. Download Maven 3.3.1. Right click on the apache-maven-3.3.1-bin.zip and select "Extract Here". Copy the extracted folder to apache-maven-3.3.1-bin to C drive.
4. Open command prompt and navigate to **C:\Hive-JSON-Serde-master** directory. Type **C:\apache-maven-3.3.9\bin\mvn package** in the command prompt and press enter. This will create the required jar files.

```
Administrator: C:\Windows\system32\cmd.exe
C:\Hive-JSON-Serde-master>C:\apache-maven-3.3.9\bin\mvn package_
```

5. Open **C:\LabAssets\HDI\HiveStorageFormats\Data\customerjson.txt** file. This is the Json data which is to analyzed.

```
{ "customerid": "JRK07IRCIJ", "firstname": "Anne",
  "lastname": "Fernandez", "city": "Charlotte", "state": "NC", "email": "annef@yahoo.com",
  "phone": { "work": "9202984230", "home": "1232432424" } },

{ "customerid": "JRK09IRKJIL", "firstname": "Adam", "lastname": "James", "city": "Seattle",
  "state": "WA", "email": "adamf@yahoo.com", "phone": { "work": "9205830444", "home": "2323232
323" } }
```

Note: Observe the nesting for the phone number column.

6. Open a new command prompt and enter the following command to copy the compiled jar file from virtual machine to the HDInsight cluster.

```
C:\LabAssets\HDI\HiveStorageFormats\pscp.exe "C:\Hive-JSON-Serde-
master\target\json-serde-1.1.9.9-Hive1.2-jar-with-dependencies.jar"
sshadmin@nthdilabhive-ssh.azurehdinsight.net:/tmp
```

*Note: Replace nthdilabhive with the name of your cluster if you aren't using the provided shared cluster. Copy **C:\Hive-JSON-Serde-master\target\json-serde-1.1.9.9-Hive1.2-jar-with-dependencies.jar** from your virtual machine to **tmp directory** on HDInsight cluster.*

7. In the solution explorer open **customerjson.hql**. Click Submit to execute the query.


```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;
-- add jar file to hive. This allows hive to interpret the json serde
add JAR /tmp/json-serde-1.1.9.9-Hive1.2-jar-with-dependencies.jar;
-- create table customerjson
CREATE TABLE ${hiveconf:newdb}.customerjson
(
    customerid STRING,
    firstname STRING,
    lastname STRING,
    city STRING,
    state STRING,
    email STRING,
    phone map<string,string>
) ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS TEXTFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/customerjson/';
```

Note: The query creates an external table customerjson in. The data for the table is stored in customerjson folder in the yourcontainer.

8. In the solution explore open **read_customerjson.hql**. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

add JAR /tmp/json-serde-1.1.9.9-Hive1.2-jar-with-dependencies.jar;
SELECT customerid, phone["work"] AS WorkPhone from ${hiveconf:newdb}.customerjson;
```

Note: You'll have to add the JAR everytime you use the customer SerDe. The query returns customerid and work phone. Remember from the json data a customer has a home and work phone.

You should get the following output from the above query.

| customerid | workphone |
|-------------|------------|
| JRK07IRCIJ | 9202984230 |
| JRK09IRKJIL | 9205830444 |

The Json serde successfully parses the Json data.

Partitioning

Hive allows for splitting data into two or more horizontal partitions based on column values. This improves queries which filter data on specific column values.

In the solution explorer, click on **create_partitioned_orc_weblogs.hql** file. Click Submit to execute the query.

```
-- Enable dynamic partitioning
SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode=non-strict;

set newdb=HDILABDB_YourName;
-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;

CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.partitioned_orc_weblogs(
    transactionid varchar(50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenummer varchar(50) ,
    invoicestatus varchar(50) ,
    paymentamount varchar(50)
)
PARTITIONED BY (
    state varchar(10),
    city varchar(50)
)
STORED AS ORC LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/partitioned
orc/'
tblproperties ("orc.compress"="SNAPPY","orc.create.index"="true");
```

Note: The above query creates an ORC format table partitioned on state and city column. With dynamic partitioning enabled, Hive automatically creates the partitions based on the specified column values. The setting `hive.exec.dynamic.partition.mode = non-strict` tells that there are no static partitions and all of the partitions are dynamic. If a table has both dynamic and static partition, then `hive.exec.dynamic.partition.mode` should be set to `strict`.

In the solution explorer, click on **load_partitioned_orc_weblogs.hql** file. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

SET hive.exec.dynamic.partition.mode=non-strict;
INSERT INTO TABLE ${hiveconf:newdb}.partitioned_orc_weblogs PARTITION (state,city)
SELECT transactionid ,
    transactiondate ,
    customerid ,
    bookid ,
    purchasetype ,
    orderid ,
    bookname ,
    categoryname ,
```

```

    quantity ,
    shippingamount ,
    invoicenum ,
    invoicestatus ,
    paymentamount ,
    trim(state) ,
    trim(city)
FROM ${hiveconf:newdb}.staging_weblogs;

```

Note: The above query dynamically creates necessary partitions and populates the partitions with appropriate records.

When the query completes, click on the Job log link in the Job view window to view the query log.

```

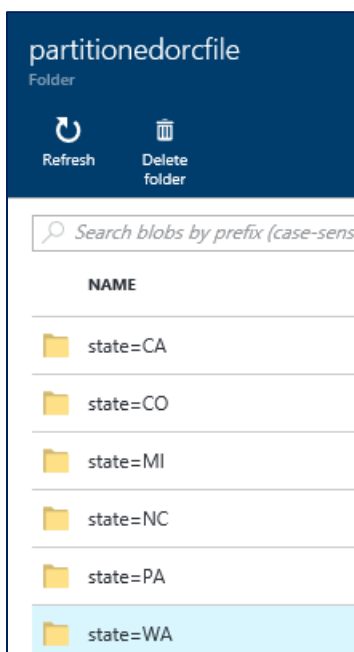
Loading data to table default.partitioned_orc_weblogs partition (state=null,
city=null)
    Time taken for load dynamic partitions : 15732
    Loading partition {state=NC, city=Charlotte}
    Loading partition {state=MI, city=Michigan City}
    Loading partition {state=CO, city=Denver}
    Loading partition {state=CA, city=Fremont}
    Loading partition {state=WA, city=Seattle}
    Loading partition {state=WA, city=Kirkland}
    Loading partition {state=PA, city=Philadelphia}

```

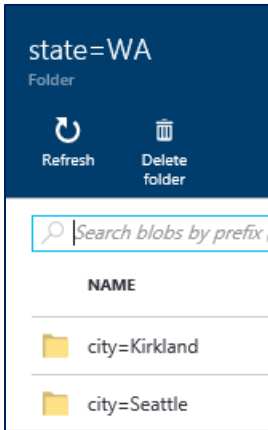
Hive identifies 7 distinct partitions and inserts data into the partitions.

Hive stores each of the partitions in their respective directory unlike non-partitioned table where all files are stored in a single directory.

Open <https://portal.azure.com> in internet explorer and login with your credentials. Navigate to the cluster storage account. Select blobs and then select **bookstore** container. Type partitioned in the search box and select **partitionedorc** directory.

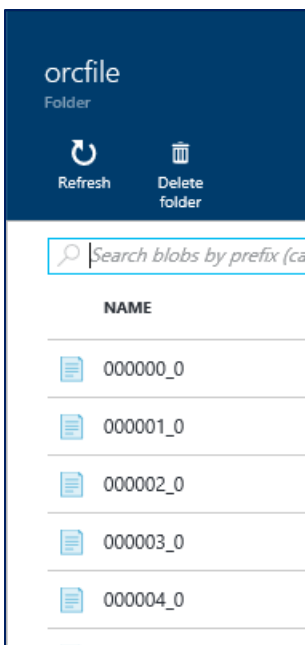


Observe that there is a separate directory for each partition. The directory name specifies the partition column and the value. Click on **state=WA** folder.



Observe that inside a state directory folder there is one directory each for a city in a state.

Close the **state=WA** folder and **partitionedorc** windows. Type orcfile in the search box and open the **orcfile** directory.



Observe that in non-partitioned table there are no sub directories. Similarly, check the different partition folders. Each of the partition have 8 different files.

In the solution explorer, click **read_partitioned_orc_weblogs.hql**. Click Submit to execute the query.

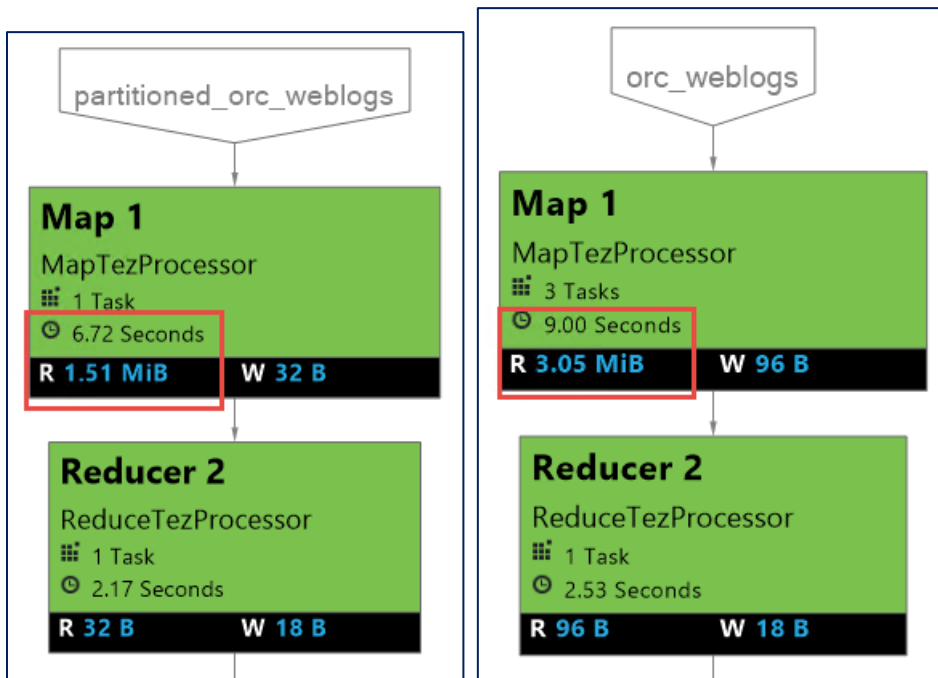
```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

SELECT    sum(PaymentAmount)
FROM      ${hiveconf:newdb}.partitioned_orc_weblogs
WHERE     state = "WA"
          AND city = "Seattle"
          AND PurchaseType = "Purchased";
```

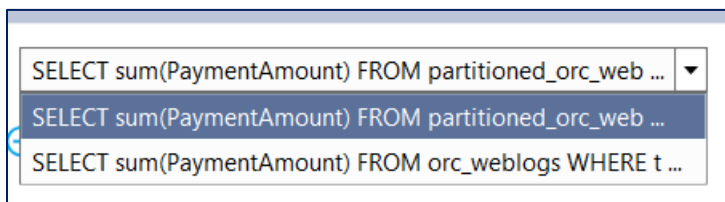
```
SELECT SUM(PaymentAmount)
FROM   ${hiveconf:newdb}.orc_weblogs
WHERE  trim(STATE) = "WA"
      AND city = "Seattle"
      AND PurchaseType = "Purchased";
```

*Note: The above query calculates the total sales made in state "WA" and city "Seattle", from **partitioned_orc_weblogs** and **orc_weblogs**.*

Once the query complete, compare the Tez execution graph for **partitioned_orc_weblogs** and **orc_weblogs** table.



*Note: Select the **orc_weblogs** query from the execution graph pane in the job view window to fetch Tez execution graph for **orc_weblogs** table.*



The query to the **partitioned_orc_weblogs** does half the I/O as of the **orc_weblogs** non-partitioned table.

Bucketed Tables

Partitioning will benefit if all the partitions have equal number of records. However, this may not be possible in all scenarios. In certain scenarios some partitions may have large number of rows compared to other partitions, which may result in uneven number of file per partition.

Bucketing overcomes this problem by splitting data into more manageable equal parts.

In the solution explorer, click **uneven_staging_weblogs.hql**. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;

-- create uneven_staging_weblogs table
CREATE TABLE ${hiveconf:newdb}.uneven_staging_weblogs
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/uneven_staging_weblogs/'
AS
SELECT * FROM ${hiveconf:newdb}.staging_weblogs;

-- insert records for state=PA
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
INSERT INTO ${hiveconf:newdb}.uneven_staging_weblogs SELECT * FROM
${hiveconf:newdb}.staging_weblogs where trim(state)="PA";
```

Note: The query inserts extra transactions for state "PA". Therefore, PA has more transactions than any other states.

In the solution explorer, click on **create_bucketed_orc_weblogs.hql** file. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;

CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.bucketed_orc_weblogs(
    transactionid varchar (50),
    transactiondate varchar(50) ,
    customerid varchar(50) ,
    bookid varchar(50) ,
    purchasetype varchar(50) ,
    orderid varchar(50) ,
    bookname varchar(50) ,
    categoryname varchar(50) ,
    quantity varchar(50) ,
    shippingamount varchar(50) ,
    invoicenummer varchar(50) ,
```

```

        invoicestatus varchar(50) ,
        paymentamount varchar(50) ,
        city VARCHAR(50)
    )
    PARTITIONED BY (
        state varchar(10)
    )
    CLUSTERED BY (city) SORTED BY (transactiondate) INTO 20 BUCKETS
    STORED AS ORC LOCATION
    'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/bucketedorc
    /'
    tblproperties ("orc.create.index"="true");

```

Note: The above query partitions the table on state column and buckets on city column. The data in each bucket is sorted on transactiondate column. The CLUSTERED BY clause specifies the bucketing column and the SORTED BY column specifies the sort column.

In the solution explorer, click on **load_bucketed_orc_weblogs.hql** file. Click Submit to execute the query.

```

SET hive.exec.dynamic.partition.mode=non-strict;

-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

INSERT INTO TABLE ${hiveconf:newdb}.bucketed_orc_weblogs PARTITION (state)
SELECT transactionid ,
       transactiondate ,
       customerid ,
       bookid ,
       purchasetype ,
       orderid ,
       bookname ,
       categoryname ,
       quantity ,
       shippingamount ,
       invoicenumber ,
       invoicestatus ,
       paymentamount ,
       city ,
       trim(state)
FROM ${hiveconf:newdb}.uneven_staging_weblogs;

```

When the query completes, click on the Job log link in the Job view window to view the query log.

```

Loading data to table default.bucketed_orc_weblogs partition (state=null)
Time taken for load dynamic partitions : 20988
Loading partition {state=WA}
Loading partition {state=CO}
Loading partition {state=CA}
Loading partition {state=PA}
Loading partition {state=MI}
Loading partition {state=NC}

```

Note: The query creates dynamic partitions and populates them with relevant data.

Open <https://portal.azure.com> in internet explorer and login with your credentials. Navigate to the cluster storage account. Select blobs and then select **yourcontainer** container. Type partitioned in the search box and select the **bucketedorc** directory.

Observe that each of the partitioned directories have 20 files each.

Compression

Hive supports multiple compression techniques to minimize the disk space required for files, network overhead and disk I/O. This is achieved on the cost of high CPU utilization. Therefore, compression is best suited for high disk I/O jobs involving TBs of data.

Hive supports following compression algorithms

GZip & BZip2

Highest compression output with highest CPU overhead and compression time. To be used when disk space and I/O utilization is high. Hadoop splits a single file into multiple smaller parts and each part is processed by a separate map process. This increases overall job performance. GZip is not splittable.

Snappy & LZO

Less compression ration with lowest compression time especially decompression time. To be used when disk space and I/O utilization are of less importance than fast decompression of commonly read data.

In the solution explorer, click on **snappy_parq_weblogs.hql** file. Click Submit to execute the query.

```
SET hive.exec.compress.output=true;
set parquet.compression=SNAPPY;

-- Set the value of newdb variable to the database you created in Setup.hql file
set newdb=HDILABDB_YourName;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;
CREATE TABLE ${hiveconf:newdb}.snappy_parq_weblogs STORED AS PARQUET
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/snappy_parq
_weblogs/'
AS
SELECT * FROM ${hiveconf:newdb}.staging_weblogs;
```

*Note: The above query compresses the **PARQUET** file format using Snappy compression and stores the data in **snappy_parq_weblogs** table.*

In the solution explorer, click on **gzip_parq_weblogs.hql** file. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;
```



```

SET hive.exec.compress.output=true;
SET parquet.compression=gzip;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;
CREATE TABLE ${hiveconf:newdb}.gzip_parq_weblogs STORED AS PARQUET
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/gzip_parq_w
eblogs/'
AS
SELECT * FROM ${hiveconf:newdb}.staging_weblogs;

```

Open <https://portal.azure.com> in internet explorer and login with your credentials. Navigate to the cluster storage account. Select blobs and then select **yourcontainer** container. Type **snappy** in the search box and select the **snappy_parq_weblogs** directory.

| NAME | MODIFIED | BLOB TYPE | SIZE |
|----------|----------------------|------------|---------|
| 000000_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000001_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000002_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000003_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000004_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000005_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000006_0 | 5/26/2016 6:49:57 PM | Block blob | 3.76 MB |
| 000007_0 | 5/26/2016 6:49:57 PM | Block blob | 3.84 MB |

Observe that there are 8 files of size 3.76 MB each. The total data is of 30 MB.

Close **snappy_parq_weblogs** directory and type **gzip** in the search box. Click to open **gzip_parq_weblogs** directory. Observe that there are 8 files of 2.44 MB each. The total data is of 19.5 MB.

Gzip offers better compression than Snappy.

In the solution explorer, click to open **read_compress_weblogs.hql**. Click Submit to execute the query.

```

-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SELECT categoryname ,

```

```

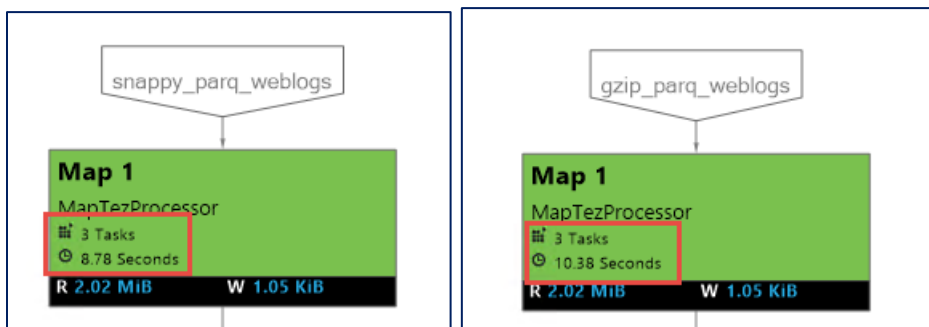
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.snappy_parq_weblogs
WHERE   PurchaseType = "Purchased"
GROUP BY CategoryName
ORDER BY quantitysold DESC;

SELECT  categoryname ,
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.gzip_parq_weblogs
WHERE   PurchaseType = "Purchased"
GROUP BY CategoryName
ORDER BY quantitysold DESC;

```

Note: The query selects data from snappy_parq_weblogs and gzip_parq_weblogs.

Once the job completes, observe the Tez execution graph of the queries.



The Map tasks in Snappy compression is faster than the Map task in Gzip compression. This is because decompression in Snappy is faster than that of Gzip.

HiveQL Optimization

In this section, we'll look at different techniques to further tune Hive queries.

Using Explain

Using Explain keyword in front of a hive query displays query plan and other information. Query plan can be read to diagnose the cause of a slow performant query.

In the solution explorer, click on **explain_staging_weblogs.hql** file.

```

-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

EXPLAIN SELECT SUM(CAST (paymentamount as INT)) AS TotalSales
FROM ${hiveconf:newdb}.staging_weblogs WHERE PurchaseType="Purchased";

```

Note: The EXPLAIN clause in front of the query will display the execution plan and not the query output

Once the query completes, click on the Job Output folder in the Job View window. The output will be a query plan as shown below.

A query is executed in multiple stages as shown below, stage 0 and stage 1. Stage 0 fetches the result and stage 1 does the bulk of the processing.

```

Stage-0
  Fetch Operator
    limit:-1
Stage-1
  Reducer 2
    File Output Operator [FS_7]
      compressed:false
      Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats:
NONE

table:{"serde":"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe","input
format":"org.apache.hadoop.mapred.TextInputFormat","output
format":"org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat"}
  Group By Operator [GBY_5]
    | aggregations:["sum(VALUE._col0)"]
    | outputColumnNames:["_col0"]
    | Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column
stats: NONE
  |<-Map 1 [SIMPLE_EDGE]
    Reduce Output Operator [RS_4]
      sort order:
      Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column
stats: NONE

    value expressions:_col0 (type: bigint)
    Group By Operator [GBY_3]
      aggregations:["sum(UDFToInteger(paymentamount))"]
      outputColumnNames:["_col0"]
      Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE
Column stats: NONE

    Select Operator [SEL_2]
      outputColumnNames:["paymentamount"]
      Statistics:Num rows: 672729 Data size: 67272933 Basic
stats: COMPLETE Column stats: NONE
      Filter Operator [FIL_8]
        predicate:(purchasetype = 'Purchased') (type: boolean)
        Statistics:Num rows: 672729 Data size: 67272933 Basic
stats: COMPLETE Column stats: NONE
        TableScan [TS_0]
          alias:staging_weblogs
          Statistics:Num rows: 1345459 Data size: 134545968
Basic stats: COMPLETE Column stats: NONE

```

A **TableScan** scans gets the total number of rows in the table and the **Filter Operator** applies the predicate to filter out **672729** rows with **purchasetype** value "Purchased". The **Select Operator** gets the **paymentamount** column from the filtered set of rows. The **UDFtoInteger** converts the data type of paymentamount column from varchar to integer and the **Group By Operator** applies the sum aggregation and produces an output column **_col0**. The **Fetch Operator** fetches the output of the query.

Indexes

Hive uses index to increase the performance of read queries. Index data for a table is stored in a separate table.

In the solution explorer, click on **total_sales.hql**. Execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SELECT SUM(CAST (paymentamount AS INT)) AS TotalSales
FROM    ${hiveconf:newdb}.staging weblogs
WHERE   PurchaseType = "Purchased";
```

Once the query completes, click on Job Log link in the Job View window. Observe the execution time of the query. The query takes **30 seconds** to complete.

In the solution explorer, click on **index_total_sales.hql**. Uncomment the Create Index statement and execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;
SET hive.execution.engine=mr;

-- Replace nthdilabhive/<yourname> with the container created in step "Create a new
Azure Storage Container"
SET Container=nthdilabhive/<yourname>;
-- specify the storage name if you have created your own HDInsight cluster
SET Storage=nthdilabs;

CREATE INDEX index_total_sales ON TABLE
${hiveconf:newdb}.staging_weblogs (purchasestype,paymentamount)
AS 'COMPACT' WITH DEFERRED REBUILD
LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/index_total_sales/';

--ALTER INDEX index_total_sales on ${hiveconf:newdb}.staging_weblogs REBUILD;
```

Note: The query creates index on purchasestype and paymentamount column. The CompactIndexHandler is the java class that implements indexing. The WITH DEFERRED REBUILD creates an empty index. It has to be rebuild later to populate with data.

Comment the Create Index query and Uncomment the ALTER REBUILD query. Execute the query to rebuild the index.

```
ALTER INDEX index_total_sales on staging_weblogs REBUILD;
```

*Note: The above query populates the index_total_sales with data. To verify, navigate to your azure storage account on azure portal and search of **index_total_sales** in the bookstore container.*

Once the job completes, navigate to **total_sales.hql** query again and execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SELECT SUM(CAST (paymentamount AS INT)) AS TotalSales
FROM    ${hiveconf:newdb}.staging_weblogs
WHERE   PurchaseType = "Purchased";
```

Once the query completes, click on Job Log link in the Job View window. Observe the execution time of the query. The query takes 27 seconds to complete.

Cost Based Optimization

Hive is a rule based optimization, that follows certain set of rules to determine how to execute a query. Cost based optimization calculates cost for multiple query plans and selects a plan with minimum costs.

In the solution explorer, select **enable_cbo_staging_weblogs.hql**. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

set hive.cbo.enable=true;
set hive.compute.query.using.stats=true;
set hive.stats.fetch.column.stats=true;
set hive.stats.fetch.partition.stats=true;
analyze table ${hiveconf:newdb}.staging_weblogs compute statistics for columns;
```

*Note: The above configuration statement enable cost based optimization and compute statistics for all columns of **staging_weblogs** table.*

In the solution explorer, select **explain_staging_weblogs.hql** and Click Submit to execute the query.

Once the query completes, in the Job View window, click Job Output to view the query plan.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

EXPLAIN SELECT SUM(CAST (paymentamount as INT)) AS TotalSales
FROM ${hiveconf:newdb}.staging_weblogs WHERE PurchaseType="Purchased";
```

Observer the Column Stats in the query plan. The value is Complete as compared to None in the last query plan with Cost Based Optimization disabled.

```
Stage-0
  Fetch Operator
    limit:-1
    Stage-1
      Reducer 2
        File Output Operator [FS_7]
          compressed:false
          Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats:
COMPLETE

table:{"serde":"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe","input
format":"org.apache.hadoop.mapred.TextInputFormat","output
format":"org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat"}
  Group By Operator [GBY_5]
    | aggregations:["sum(VALUE._col0)"]
```

```

      | outputColumnNames:["_col0"]
      | Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column
stats: COMPLETE
    |<-Map 1 [SIMPLE_EDGE]
      Reduce Output Operator [RS_4]
        sort order:
        Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE Column
stats: COMPLETE
      value expressions:_col0 (type: bigint)
      Group By Operator [GBY_3]
        aggregations:["sum(UDFToInteger(paymentamount))"]
        outputColumnNames:["_col0"]
        Statistics:Num rows: 1 Data size: 8 Basic stats: COMPLETE
Column stats: COMPLETE
      Select Operator [SEL_2]
        outputColumnNames:["paymentamount"]
        Statistics:Num rows: 672729 Data size: 122436678 Basic
stats: COMPLETE Column stats: COMPLETE
      Filter Operator [FIL_8]
        predicate:(purchasetype = 'Purchased') (type: boolean)
        Statistics:Num rows: 672729 Data size: 122436678 Basic
stats: COMPLETE Column stats: COMPLETE
      TableScan [TS_0]
        alias:staging_weblogs
        Statistics:Num rows: 1345459 Data size: 134545968
Basic stats: COMPLETE Column stats: COMPLETE

```

Vectorization

Vectorization causes hive to process a block 1024 rows at a time instead of a single row. This greatly reduces CPU utilization for scans, filters, joins and aggregations. Vectorization requires the data to be stored in ORC format.

In the solution explorer, open **enable_vectorization.hql**. Click Submit to execute the query.

```

-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SET hive.vectorized.execution.enabled=true;
SET hive.vectorized.execution.reduce.enabled = true;
SET hive.vectorized.execution.reduce.groupby.enabled = true;

SELECT SUM(CAST (paymentamount as INT)) AS TotalSales
FROM ${hiveconf:newdb}.orc_weblogs WHERE PurchaseType="Purchased";

```

*Note: The configuration parameters **hive.vectorized.execution.enabled** is set to true by default in hive version 0.13.0 and higher.*

Parallel Execution

A hive query is converted to multiple stages. These stages are executed one at a time. However, if a job consists of stages that are independent of each other, these stages can be executed in parallel to improve the query performance.

In the solution explorer, click to open **parallel_execution.hql**. Click Submit to execute the query.

```

-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

```

```
SET hive.exec.parallel=TRUE;

SET hive.exec.parallel.thread.number=16;

SELECT  SUM(CAST (paymentamount AS INT)) AS TotalSales
FROM    ${hiveconf:newdb}.orc_weblogs
WHERE   PurchaseType = "Purchased";
```

*Note: Observe that the configuration parameter **hive.exec.parallel** is set to true for parallel execution of the query. The parameter **hive.exec.parallel.thread.number** specifies the number of threads to be run in parallel.*

Tuning the number of reducers

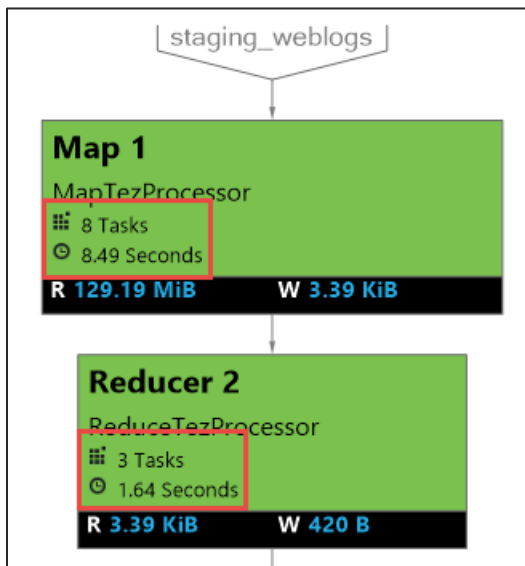
Tez decides the number of reducer tasks required to execute the query at run time. The number of reducers can be modified to get optimal performance.

In the solution explorer, click **tune_reducer.hql** and Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SELECT  categoryname ,
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.staging_weblogs
GROUP BY CategoryName
ORDER BY quantitysold DESC;
```

Once the query completes, observe the Tez execution graph. The default number of reducer task for **Reducer 2** are **Three**.



In the Job View window, click on Job Log link. Observe the query execution time. The query takes **45 seconds** to execute.

Note: The number of reducers may be different in your case.

Navigate to **tune_reducer.hql** and add the following configuration parameter right above the select query.

```
SET mapreduce.job.reduces = 2;
```

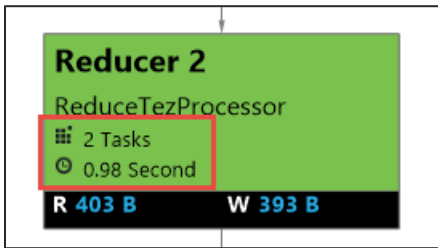
The final query should look like this,

```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SET mapreduce.job.reduces = 2;

SELECT  categoryname ,
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.staging_weblogs
GROUP BY CategoryName
ORDER BY quantitysold DESC;
```

Click Submit to execute the query again. Once the job completes, observe the Tez execution graph. The number of task for **Reducer 2** is changed to **Two**.



In the Job View window, click on Job Log link. Observer the query execution time. The query takes **46 seconds** to execute.

Navigate to **tune_reducer.hql** and replace mapreduce.job.reduces parameter with the following configuration parameter.

```
SET hive.exec.reducers.bytes.per.reducer=10000000;
```

The final query should look like this,

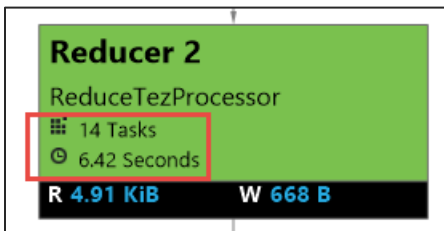
```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SET hive.exec.reducers.bytes.per.reducer=10000000;

SELECT  categoryname ,
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.staging_weblogs
GROUP BY CategoryName
ORDER BY quantitysold DESC;
```

Note: This parameter sets the number of bytes processed by a reducer to 10000000 bytes (10 MB). The default value for this configuration parameter is 67108864 bytes (64 MB).

Click Submit to execute the query. Once the job completes, observe the Tez execution graph. The number of tasks for **Reducer 2** is changed to **Fourteen**. The execution time of **Reducer 2** is now **6.42 seconds** as compared to **0.98 seconds** with 2 tasks.



Note: Increasing or decreasing number of reducers may or may not improve the query performance.

Tuning the number of mappers

Hadoop tries to split single file into multiple files and process the resulting files in parallel. The number of mappers depend on the number of splits.

In the solution explorer, click to open **tune_mappers.hql**.

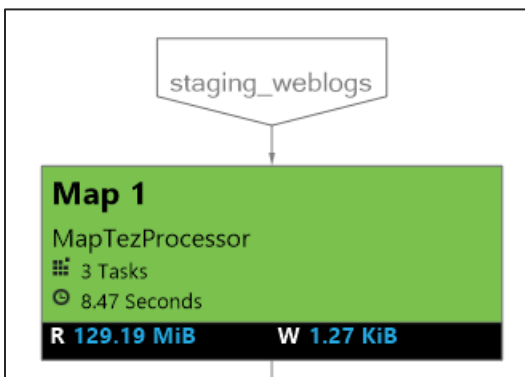
```
-- Set the value of newdb variable to the database you created in Setup.hql file
SET newdb=HDILABDB_YourName;

SET tez.grouping.split-count=2;

SELECT  categoryname ,
        SUM(Quantity) AS quantitysold
FROM    ${hiveconf:newdb}.staging_weblogs
GROUP BY CategoryName
ORDER BY quantitysold DESC;
```

*Note: The configuration parameter **tez.grouping.split-count** tell Tez to group splits into the specified number of groups.*

Click Submit to execute the query. Once the job completes, observe the Tez execution graph. The number of map tasks are now **Three** instead of **Eight** as shown in the previous section “**Tuning Reducers**”.



Navigate to **tune_mappers.hql** and replace the **tez.grouping.split-count** parameter with the following parameters.

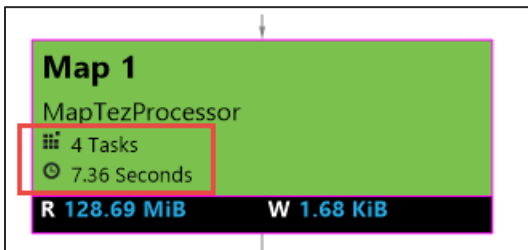
```
SET tez.grouping.min-size=33554432;  
SET tez.grouping.max-size=33554432;
```

The final query should look like this,

```
-- Set the value of newdb variable to the database you created in Setup.hql file  
SET newdb=HDILABDB_YourName;  
  
SET tez.grouping.min-size=33554432;  
SET tez.grouping.max-size=33554432;  
  
SELECT  categoryname ,  
        SUM(Quantity) AS quantitiesold  
FROM    ${hiveconf:newdb}.staging_weblogs  
GROUP BY CategoryName  
ORDER BY quantitiesold DESC;
```

Note: The above configuration parameter specified the minimum and maximum split size in bytes. In our example, the Map 1 reads around 128 MB of data. Therefore, setting minimum and maximum value to 32 MB should result in 4 map tasks.

Click Submit to execute the query. Once the job completes, observe the Tez execution graph. The number of map tasks are now **Four** based on out split size settings. The Map process now takes **7.36 seconds** to complete instead of **8.47 seconds** compared to the last execution. The query takes **40 seconds** to complete which is **5 seconds** faster than that of last execution.



Note: Increasing or decreasing number of mappers may or may not improve the query performance.

Multiple Inserts

Hive allows for loading data into two or more table in a single insert statement. This results in a single scan instead of multiple scan when inserting data individually into single tables.

In the solution explorer, click to open **multiple_inserts.hql**. Click Submit to execute the query.

```
-- Set the value of newdb variable to the database you created in Setup.hql file  
SET newdb=HDILABDB_YourName;  
  
-- Replace nthdilabhive/<yourname> with the container created in step "Create a new  
Azure Storage Container"  
SET Container=nthdilabhive/<yourname>;  
-- specify the storage name if you have created your own HDInsight cluster  
SET Storage=nthdilabs;  
-- create purchased table  
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.purchased(  
    bookname varchar(50) ,  
    categoryname varchar(50)
```

```

) ROW FORMAT DELIMITED FIELDS TERMINATED by ',' lines TERMINATED by '\n'
STORED AS TEXTFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/purchased/'
;

-- create browsed table
CREATE TABLE IF NOT EXISTS ${hiveconf:newdb}.browsed(
    bookname varchar(50) ,
    categoryname varchar(50)
) ROW FORMAT DELIMITED FIELDS TERMINATED by ',' lines TERMINATED by '\n'
STORED AS TEXTFILE LOCATION
'wasb://${hiveconf:Container}@${hiveconf:Storage}.blob.core.windows.net/browsed/';

-- Multiple Insert
FROM ${hiveconf:newdb}.staging_weblogs
INSERT OVERWRITE TABLE ${hiveconf:newdb}.Purchased SELECT bookname,categoryname
WHERE PurchaseType="Purchased"
INSERT OVERWRITE TABLE ${hiveconf:newdb}.Browsed SELECT bookname,categoryname WHERE
PurchaseType="Browsed";


```

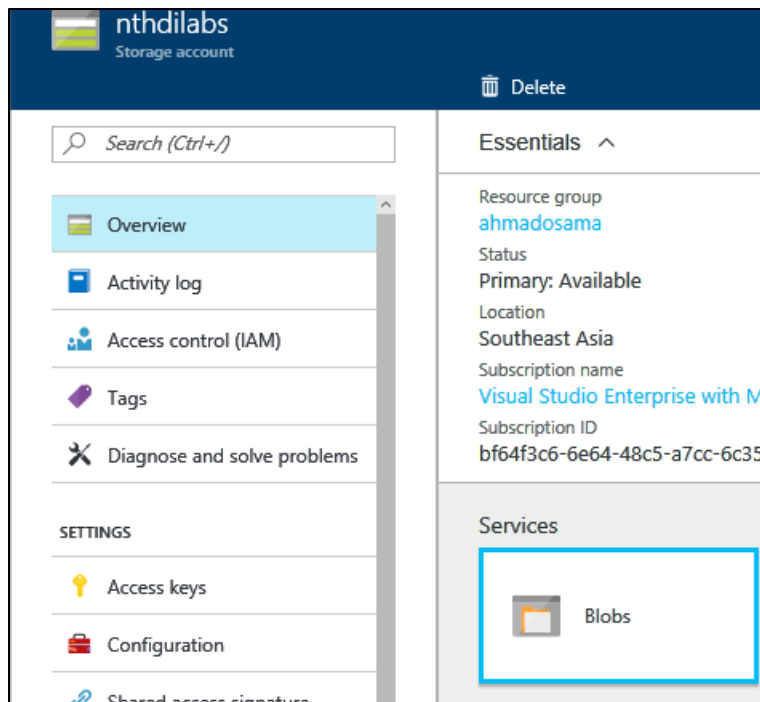
Note: The query inserts purchased and browsed transactions into Purchased and Browsed tables in a single query.

Roll Back Azure Changes

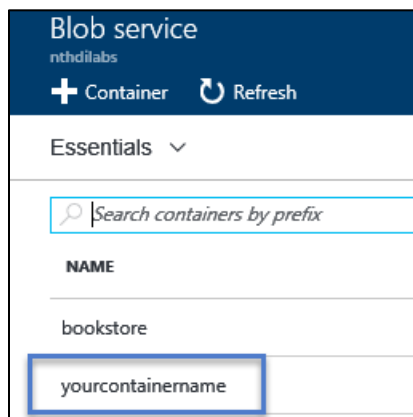
Now we should clean up the resources we have used during this hands-on lab. The following items should be deleted from your subscription.

Delete Azure Storage Container

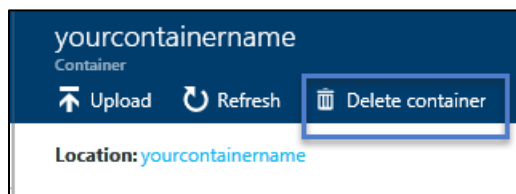
1. Go to the Azure Preview Portal by clicking the **Preview Portal** link  on the IE favorites bar.
2. Click **All Resources**. Locate and click on the storage account you created. In the storage blade, click blobs.



1. In the Blob Service blade, select yourcontainername.



2. In the yourcontainername blog, select delete to delete the container.

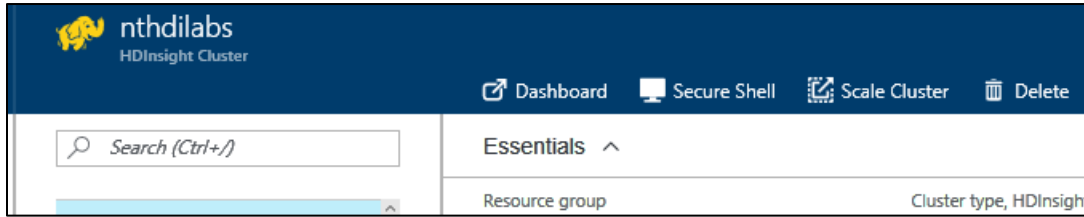


Delete HDInsight Hadoop Cluster

Note: These steps should be ignored if you are provided a shared cluster.

1. Go to the Azure Preview Portal by clicking the **Preview Portal** link  on the IE favorites bar.

2. Click **All Resources**. Locate and click on the HDInsight cluster you created. In the cluster blade, click delete.



3. Click Yes, in the delete confirmation dialog box to delete the cluster.

You can now close the lab.

Terms of use

© 2016 Microsoft Corporation. All rights reserved.

By using this hands-on lab, you agree to the following terms:

The technology/functionality described in this hands-on lab is provided by Microsoft Corporation in a "sandbox" testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the hands-on lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this hands-on lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality, and/or concepts described in this hands-on lab to Microsoft, you give to Microsoft, without charge, the right to use, share, and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies, and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.