IBM Training

Demonstration 1

Working with Big SQL data

At the end of this demonstratrion, you will be able to:

- Create Big SQL tables that use Hadoop text file and Parquet file formats.
- Populate Big SQL tables from local files and from the results of queries.
- Query Big SQL tables using projections, restrictions, joins, aggregations, and other popular expressions.
- Create and query a view based on multiple Big SQL tables.

Querying Hadoop data using Big SQL tables

Demonstration 1: Working with Big SQL data

Demonstration 1: Working with Big SQL data

Purpose:
Now you're ready to query your tables. Based on earlier demonstrations, you've already seen that you can perform basic SQL operations.

In this demonstration, you will create and run Big SQL queries that join data from multiple tables as well as perform aggregations and other SQL operations. Note that the queries included in this section are based on queries shipped with Big SQL client software as samples.

Estimated time: 60 minutes

User/Password: **biadmin/biadmin**

root/dalvm3

Services Password: **ibm2blue**

Task 1. Loading data into Big SQL tables.

Use either JSqsh or the Big SQL service via Ambari.

Load data into each of following tables using sample data provided. If necessary, change the SFTP and file path specifications in each of the following examples to match your environment. LOAD returns a **warning** message providing details on the number of rows loaded, etc.

The 2_Load Statements.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

1. Issue the first LOAD statement and verify that the operation completed successfully.

```
load hadoop using file url

'sftp://bigsql:ibm2blue@ibmclass.localdomain:22/usr/ibmp
acks/bigsql/4.0/bigsql/samples/data/GOSALESDW.GO_REGION_
DIM.txt' with SOURCE PROPERTIES ('field.delimiter'='\t')
INTO TABLE GO_REGION_DIM overwrite;
```

Each example loads data into a table using a file URL specification that relies on SFTP to locate the source file. In particular, the SFTP specification includes a valid user ID and password (yourID/yourPassword), the target host server and port (ibmclass.localdomain:22), and the full path of the data file on that system.

The WITH SOURCE PROPERTIES clause specifies that fields in the source data are delimited by tabs (“\t”). The INTO TABLE clause identifies the target table for the LOAD operation.
The OVERWRITE keyword indicates that any existing data in the table will be replaced by data contained in the source file. (If you wanted to simply add rows to the table’s content, you could specify APPEND instead.)

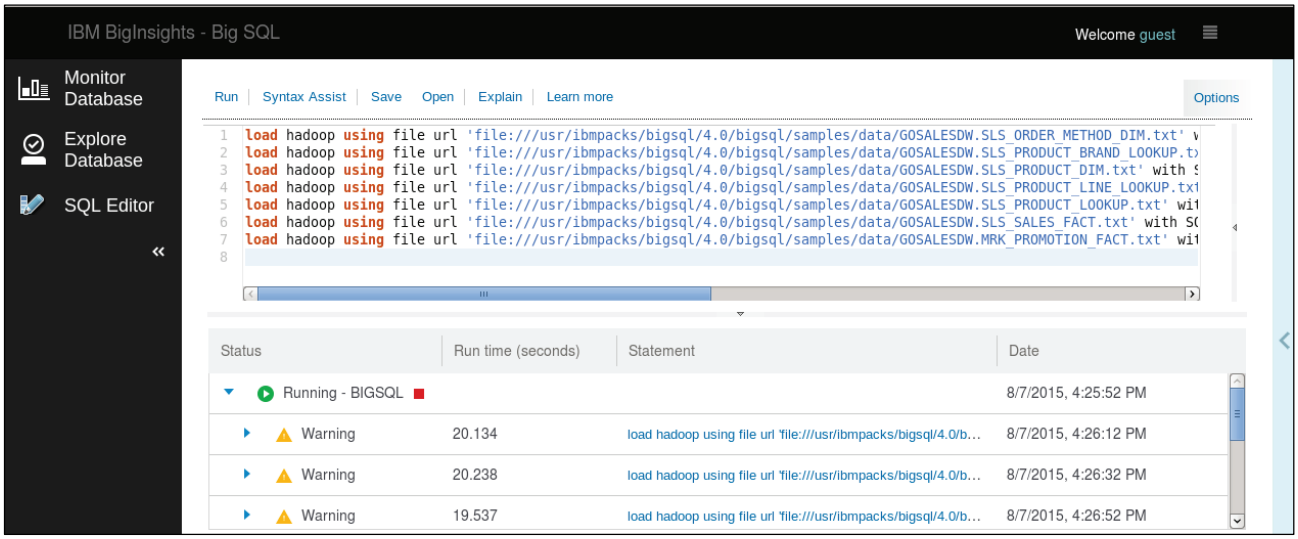
Using SFTP (or FTP) is one way in which you can invoke the LOAD command.

If your target data already resides in your distributed file system, you can provide the DFS directory information in your file URL specification. Indeed, for optimal runtime performance, you may prefer to take that approach. The remaining LOAD statements refers to the file locally, instead of using SFTP

2. Issue each LOAD statement and verify that the operation completed successfully.

```
load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_ORDER_METHOD_DIM.txt' with SOURCE PROPERTIES
('field.delimiter'='\t') INTO TABLE SLS_ORDER_METHOD_DIM
overwrite;

load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_PRODUCT_BRAND_LOOKUP.txt' with SOURCE
PROPERTIES ('field.delimiter'='\t') INTO TABLE
SLS_PRODUCT_BRAND_LOOKUP overwrite;

load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_PRODUCT_DIM.txt' with SOURCE PROPERTIES
('field.delimiter'='\t') INTO TABLE SLS_PRODUCT_DIM
overwrite;

load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_PRODUCT_LINE_LOOKUP.txt' with SOURCE
PROPERTIES ('field.delimiter'='\t') INTO TABLE
SLS_PRODUCT_LINE_LOOKUP overwrite;
```

```
load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_PRODUCT_LOOKUP.txt' with SOURCE PROPERTIES
('field.delimiter'='\t') INTO TABLE SLS_PRODUCT_LOOKUP
overwrite;

load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.SLS_SALES_FACT.txt' with SOURCE PROPERTIES
('field.delimiter'='\t') INTO TABLE SLS_SALES_FACT
overwrite;

load hadoop using file url
'file:///usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOS
ALESDW.MRK_PROMOTION_FACT.txt' with SOURCE PROPERTIES
('field.delimiter'='\t') INTO TABLE MRK_PROMOTION_FACT
overwrite;
```



3. Query the tables to verify that the expected number of rows was loaded into each table. Execute each query that follows individually and compare the results with the number of rows specified in the comment line preceding each query. You will need to expand each of the **Succeeded** lines to see the results.

   The 3_SELECT statements - validating LOAD.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

```
-- total rows in GO_REGION_DIM = 21
select count(*) from GO_REGION_DIM;

-- total rows in sls_order_method_dim = 7
select count(*) from sls_order_method_dim;

-- total rows in SLS_PRODUCT_BRAND_LOOKUP = 28
select count(*) from SLS_PRODUCT_BRAND_LOOKUP;

-- total rows in SLS_PRODUCT_DIM = 274
select count(*) from SLS_PRODUCT_DIM;

-- total rows in SLS_PRODUCT_LINE_LOOKUP = 5
select count(*) from SLS_PRODUCT_LINE_LOOKUP;

-- total rows in SLS_PRODUCT_LOOKUP = 6302
select count(*) from SLS_PRODUCT_LOOKUP;

-- total rows in SLS_SALES_FACT = 446023
select count(*) from SLS_SALES_FACT;

-- total rows gosalesdw.MRK_PROMOTION_FACT = 11034
select count(*) from MRK_PROMOTION_FACT;
```

Task 2. Querying the data with Big SQL.

1. Join data from multiple tables to return the product name, quantity and order method of goods that have been sold. For simplicity, limit the number of returns rows to 20. To achieve this, execute the following query:

   The 4_SELECT statements - querying data.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

```
--Fetch the product name, quantity, and order method of
products sold.
--
--Query 1
SELECT pnumb.product_name, sales.quantity,
meth.order_method_en
FROM
sls_sales_fact sales,
sls_product_dim prod,
sls_product_lookup pnumb,
sls_order_method_dim meth
WHERE
pnumb.product_language='EN'
AND sales.product_key=prod.product_key
AND prod.product_number=pnumb.product_number
AND meth.order_method_key=sales.order_method_key
fetch first 20 rows only;
```

   Data from four tables will be used to drive the results of this query (see the tables referenced in the FROM clause). Relationships between these tables are resolved through 3 join predicates specified as part of the WHERE clause. The query relies on 3 equi-joins to filter data from the referenced tables. (Predicates such as prod.product_number=pnumb.product_number help to narrow the results to product numbers that match in two tables.)

   For improved readability, this query uses aliases in the SELECT and FROM clauses when referencing tables. For example, pnumb.product_name refers to “pnumb,” which is the alias for the gosalesdw.sls_product_lookup table. Once defined in the FROM clause, an alias can be used in the WHERE clause so that you do not need to repeat the complete table name.

   The use of the predicate and pnumb.product_language='EN' helps to further narrow the result to only English output. This database contains thousands of rows of data in various languages, so restricting the language provides some optimization.

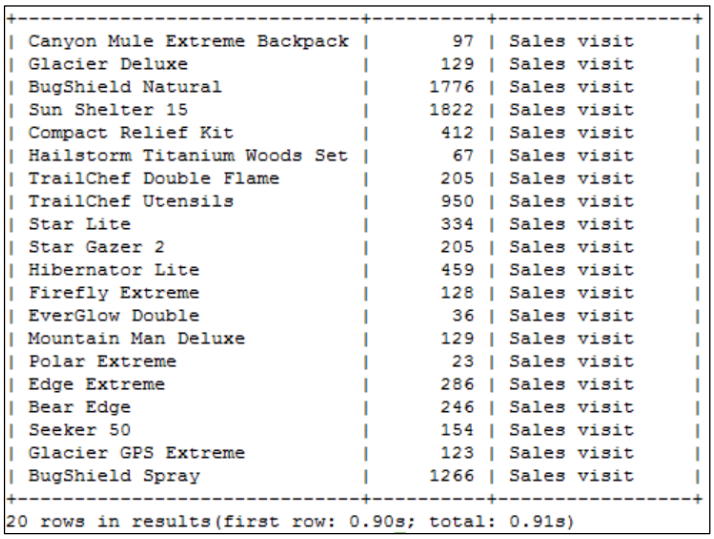
Results are shown from JSqsh, but you can use the Big SQL service via Ambari if you wish.

```
+-----------------------------+----------+-----------------+
| Compact Relief Kit          |      313 | Sales visit     |
| Course Pro Putter           |      587 | Telephone       |
| Blue Steel Max Putter       |      214 | Telephone       |
| Course Pro Gloves           |      576 | Telephone       |
| Glacier Deluxe              |      129 | Sales visit     |
| BugShield Natural           |     1776 | Sales visit     |
| Sun Shelter 15              |     1822 | Sales visit     |
| Compact Relief Kit          |      412 | Sales visit     |
| Hailstorm Titanium Woods Set |       67 | Sales visit     |
| Canyon Mule Extreme Backpack |       97 | E-mail          |
| TrailChef Canteen           |     1172 | Telephone       |
| TrailChef Cook Set          |      591 | Telephone       |
| TrailChef Deluxe Cook Set   |      338 | Telephone       |
| Star Gazer 3                |       97 | Telephone       |
| Hibernator                  |      364 | Telephone       |
| Hibernator Camp Cot         |      234 | Telephone       |
| Canyon Mule Cooler          |      603 | Telephone       |
| Firefly 4                   |      232 | Telephone       |
| EverGlow Single             |      450 | Telephone       |
| EverGlow Kerosene           |      257 | Telephone       |
+-----------------------------+----------+-----------------+
20 rows in results(first row: 1.2s; total: 1.2s)
```

2. Modify the query to restrict the order method to one type – those involving a Sales visit. To do so, add the following query predicate just before the FETCH FIRST 20 ROWS clause: AND order_method_en='Sales visit'

```
--Fetch the product name, quantity, and order method
--of products sold through sales visits.
--Query 2
SELECT pnumb.product_name, sales.quantity,
meth.order_method_en
FROM
sls_sales_fact sales,
sls_product_dim prod,
sls_product_lookup pnumb,
sls_order_method_dim meth
WHERE
pnumb.product_language='EN'
AND sales.product_key=prod.product_key
AND prod.product_number=pnumb.product_number
AND meth.order_method_key=sales.order_method_key
AND order_method_en='Sales visit'
FETCH FIRST 20 ROWS ONLY;
```

3. Inspect the results:



Canyon Mule Extreme Backpack	97	Sales visit
Glacier Deluxe	129	Sales visit
BugShield Natural	1776	Sales visit
Sun Shelter 15	1822	Sales visit
Compact Relief Kit	412	Sales visit
Hailstorm Titanium Woods Set	67	Sales visit
TrailChef Double Flame	205	Sales visit
TrailChef Utensils	950	Sales visit
Star Lite	334	Sales visit
Star Gazer 2	205	Sales visit
Hibernator Lite	459	Sales visit
Firefly Extreme	128	Sales visit
EverGlow Double	36	Sales visit
Mountain Man Deluxe	129	Sales visit
Polar Extreme	23	Sales visit
Edge Extreme	286	Sales visit
Bear Edge	246	Sales visit
Seeker 50	154	Sales visit
Glacier GPS Extreme	123	Sales visit
BugShield Spray	1266	Sales visit

20 rows in results(first row: 0.90s; total: 0.91s)

4. To find out which sales method of all the methods has the greatest quantity of orders, include a GROUP BY clause (group by pll.product_line_en, md.order_method_en). In addition, invoke the SUM aggregate function (sum(sf.quantity)) to total the orders by product and method. Finally, this query cleans up the output a bit by using aliases (e.g., as Product) to substitute a more readable column header.

```
--Query 3
SELECT pll.product_line_en AS Product,
md.order_method_en AS Order_method,
sum(sf.QUANTITY) AS total
FROM
sls_order_method_dim AS md,
sls_product_dim AS pd,
sls_product_line_lookup AS pll,
sls_product_brand_lookup AS pbl,
sls_sales_fact AS sf
WHERE
pd.product_key = sf.product_key
AND md.order_method_key = sf.order_method_key
AND pll.product_line_code = pd.product_line_code
AND pbl.product_brand_code = pd.product_brand_code
GROUP BY pll.product_line_en, md.order_method_en;
```

5. Inspect the results, which should contain 35 rows. A portion is shown below.

PRODUCT	ORDER_METHOD	TOTAL
Camping Equipment	E-mail	1413084
Camping Equipment	Fax	413958
Camping Equipment	Mail	348058
Camping Equipment	Sales visit	2899754
Camping Equipment	Special	203528
Camping Equipment	Telephone	2792588
Camping Equipment	Web	19230179
Golf Equipment	E-mail	333300
Golf Equipment	Fax	102651
Golf Equipment	Mail	80432
Golf Equipment	Sales visit	263788
Golf Equipment	Special	38585
Golf Equipment	Telephone	601506
Golf Equipment	Web	3693439
Mountaineering Equipment	E-mail	199214
Mountaineering Equipment	Fax	292408

Task 3. Creating and working with views.

Big SQL supports views (virtual tables) based on one or more physical tables. In this section, you will create a view that spans multiple tables. Then you'll query this view using a simple SELECT statement. In doing so, you'll see that you can work with views in Big SQL much as you can work with views in a relational DBMS.

The 5_VIEW statements.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

1. Create a view named MYVIEW that extracts information about product sales featured in marketing promotions. By the way, since the schema name is omitted in both the CREATE and FROM object names, the current schema (your user name), is assumed.

```
create view myview as
select product_name, sales.product_key, mkt.quantity,
    sales.order_day_key, sales.sales_order_key,
order_method_en
from
    mrk_promotion_fact mkt,
    sls_sales_fact sales,
    sls_product_dim prod,
    sls_product_lookup pnumb,
    sls_order_method_dim meth
where mkt.order_day_key=sales.order_day_key
    and sales.product_key=prod.product_key
    and prod.product_number=pnumb.product_number
    and pnumb.product_language='EN'
    and meth.order_method_key=sales.order_method_key;
```

2. Now query the view:

```
select * from myview
order by product_key asc, order_day_key asc
fetch first 20 rows only;
```

3. Inspect the results:

PRODUCT_NAME	PRODUCT_KEY	QUANTITY	ORDER_DAY_KEY	SALES_ORDER_KEY	ORDER_METHOD_EN
TrailChef Water Bag	30001	1350	20040112	195305	Sales visit
TrailChef Water Bag	30001	663	20040112	195305	Sales visit
TrailChef Water Bag	30001	495	20040112	195305	Sales visit
TrailChef Water Bag	30001	1260	20040112	195305	Sales visit
TrailChef Water Bag	30001	965	20040112	195305	Sales visit
TrailChef Water Bag	30001	1035	20040112	195305	Sales visit
TrailChef Water Bag	30001	990	20040112	195305	Sales visit

Task 4. Populating a table with 'INSERT INTO ... SELECT'.

Big SQL enables you to populate a table with data based on the results of a query. In this demonstration, you will use an INSERT INTO . . . SELECT statement to retrieve data from multiple tables and insert that data into another table. Executing an INSERT INTO . . . SELECT exploits the machine resources of your cluster because Big SQL can parallelize both read (SELECT) and write (INSERT) operations.

The 6_INSERT INTO SELECT statements.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

1. Execute the following statement to create a sample table named `sales_report`:

```
-- create a sample sales_report table
CREATE HADOOP TABLE sales_report
(
product_key       INT NOT NULL,
product_name VARCHAR(150),
quantity          INT,
order_method_en  VARCHAR(90)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

2. Now populate the newly created table with results from a query that joins data from multiple tables.

```
-- populate the sales_report data with results from a
query
INSERT INTO sales_report
SELECT sales.product_key, pnumb.product_name,
sales.quantity,
meth.order_method_en
FROM
sls_sales_fact sales,
sls_product_dim prod,
sls_product_lookup pnumb,
sls_order_method_dim meth
WHERE
pnumb.product_language='EN'
AND sales.product_key=prod.product_key
AND prod.product_number=pnumb.product_number
AND meth.order_method_key=sales.order_method_key
and sales.quantity > 1000;
```

3. Verify that the previous query was successful by executing the following query:

```
-- total number of rows should be 14441
select count(*) from sales_report;
```

Task 5. Storing data in an alternate file format (Parquet).

Until now, you've instructed Big SQL to use the TEXTFILE format for storing data in the tables you've created. This format is easy to read (both by people and most applications), as data is stored in a delimited form with one record per line and new lines separating individual records. It's also the default format for Big SQL tables.

However, if you'd prefer to use a different file format for data in your tables, Big SQL supports several formats popular in the Hadoop environment, including Avro, sequence files, RC (record columnar) and Parquet. While it's beyond the scope of this demonstration to explore these file formats, you'll learn how you can easily override the default Big SQL file format to use another format, in this case, Parquet. Parquet is a columnar storage format for Hadoop that's popular because of its support for efficient compression and encoding schemes. For more information on Parquet, visit http://parquet.io/.

The 7_Parquet statements.sql file is located in the */home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis* folder.

1. Create a table named big_sales_parquet.

```
CREATE HADOOP TABLE IF NOT EXISTS big_sales_parquet
(
product_key      INT NOT NULL,
product_name VARCHAR(150),
quantity     INT,
order_method_en   VARCHAR(90)
)
STORED AS parquetfile;
```

   With the exception of the final line (which specifies the PARQUETFILE format), all aspects of this statement should be familiar to you by now.

2. Populate this table with data based on the results of a query.

   Note that this query joins data from 4 tables you previously defined in Big SQL using a TEXTFILE format. Big SQL will automatically reformat the result set of this query into a Parquet format for storage.

```
insert into big_sales_parquet
SELECT sales.product_key, pnumb.product_name,
sales.quantity,
meth.order_method_en
FROM
sls_sales_fact sales,
sls_product_dim prod,
sls_product_lookup pnumb,
sls_order_method_dim meth
WHERE
pnumb.product_language='EN'
AND sales.product_key=prod.product_key
AND prod.product_number=pnumb.product_number
AND meth.order_method_key=sales.order_method_key
and sales.quantity > 5500;
```

3. Query the table.

   Note that your SELECT statement does not need to be modified in any way because of the underlying file format.

```
select * from big_sales_parquet;
```

4. Inspect the results.

   A portion of the results appear as follows:

PRODUCT_KEY	PRODUCT_NAME	QUANTITY	ORDER_METHOD_EN
30107	BugShield Extreme	5937	Sales visit
30107	BugShield Extreme	6282	E-mail
30107	BugShield Extreme	6121	Mail
30107	BugShield Extreme	7300	Sales visit
30107	BugShield Extreme	8772	Web
30090	Single Edge	6619	Special
30107	BugShield Extreme	5855	Sales visit
30107	BugShield Extreme	5523	Web
30107	BugShield Extreme	5658	Web
30107	BugShield Extreme	6948	Sales visit

   Remember that parquet files are not human-readable.

   Navigate to /apps/hive/warehouse/bigsql.db/big_sales_parquet directory in the hdfs if you want to see the table's actual content.

Task 6. Working with external tables.

The previous tasks in this demonstration caused Big SQL to store tables in a default location (in the Hive warehouse). Big SQL also supports the concept of an externally managed table – i.e., a table created over a user directory that resides outside of the Hive warehouse. This user directory contains all the table's data in files.
As part of this demonstration, you will create a DFS directory, upload data into it, and then create a Big SQL table that over this directory. To satisfy queries, Big SQL will look in the user directory specified when you created the table and consider all files in that directory to be the table's contents. Once the table is created, you'll query that table.

1. If necessary, open a terminal window.

2. Check the directory permissions for your DFS.

```
hdfs dfs -ls /
```

```
Found 8 items
drwxrwxrwx   - yarn   hadoop          0 2015-04-07 11:33 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:53 /apps
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 07:28 /biginsights
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:51 /iop
drwxr-xr-x   - mapred hdfs            0 2015-03-31 05:51 /mapred
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:51 /mr-history
drwxrwxrwx   - hdfs   hdfs            0 2015-03-31 05:53 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 07:28 /user
```

   If the /user directory cannot be written by the public (as shown in the example above), you will need to change these permissions so that you can create the necessary subdirectories for this task using your standard lab user account.

3. From the command line, issue this command to switch to the root user ID temporarily (if you are not already using the root user):

```
su root
```

4. If prompted, enter the password for this account. The password for root is **dalvm3**.

5. Then switch to the **hdfs** ID.

```
su hdfs
```

6. While logged in as user hdfs, issue this command:

```
hdfs dfs -chmod 777 /user
```

7. Confirm the effect of your change:

```
hdfs dfs -ls /
```

```
Found 8 items
drwxrwxrwx   - yarn   hadoop          0 2015-04-07 11:33 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:53 /apps
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 07:28 /biginsights
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:51 /iop
drwxr-xr-x   - mapred hdfs            0 2015-03-31 05:51 /mapred
drwxr-xr-x   - hdfs   hdfs            0 2015-03-31 05:51 /mr-history
drwxrwxrwx   - hdfs   hdfs            0 2015-03-31 05:53 /tmp
drwxrwxrwx   - hdfs   hdfs            0 2015-03-31 07:28 /user
```

8. Exit the hdfs user account:

```
exit
```

9. Exit the root user account and return to your standard user account.

```
exit
```

10. Create directories in your distributed file system for the source data files and ensure public read/write access to these directories. (If desired, alter the DFS information as appropriate for your environment.)

```
hdfs dfs -mkdir /user/bigsql_lab

hdfs dfs -mkdir /user/bigsql_lab/sls_product_dim

hdfs dfs -chmod -R 777 /user/bigsql_lab
```

11. Switch to the **bigsql** user with the password **ibm2blue**.

```
su bigsql
```

12. Upload the source data files into their respective DFS directories and change the local and DFS directories information below to match your environment.

```
hdfs dfs -copyFromLocal
/usr/ibmpacks/bigsql/4.0/bigsql/samples/data/GOSALESDW.SLS
_PRODUCT_DIM.txt
/user/bigsql_lab/sls_product_dim/SLS_PRODUCT_DIM.txt
```

13. List the contents of the DFS directories into which you copied the files to validate your work.

```
hdfs dfs -ls /user/bigsql_lab/sls_product_dim
```

```
Found 1 items
-rw-r--r--   3 bigsql hdfs      24089 2015-09-03 15:25 /user/bigsql_lab/sls_prod
uct_dim/SLS_PRODUCT_DIM.txt
```

14. Exit the **bigsql** user account and return to your standard user account:

```
exit
```

15. From your query execution environment (such as JSqsh or the SQL Editor), create an external Big SQL table for the sales product dimension (sls_product_dim_external).

    Note that the LOCATION clause in each statement references the DFS directory into which you copied the sample data.

    The 8_External tables.sql file is located in the /home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis folder

```
-- product dimension table stored in a DFS directory
external to Hive
CREATE EXTERNAL HADOOP TABLE IF NOT EXISTS
sls_product_dim_external
 ( product_key INT NOT NULL
 , product_line_code INT NOT NULL
 , product_type_key INT NOT NULL
 , product_type_code INT NOT NULL
 , product_number INT NOT NULL
 , base_product_key INT NOT NULL
 , base_product_number INT NOT NULL
 , product_color_code INT
 , product_size_code INT
 , product_brand_key INT NOT NULL
 , product_brand_code INT NOT NULL
 , product_image VARCHAR(60)
 , introduction_date TIMESTAMP
 , discontinued_date TIMESTAMP
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 LINES TERMINATED BY '\n'
 location '/user/bigsql_lab/sls_product_dim';
```

16. Query the table.

```
select product_key, introduction_date from
sls_product_dim_external
where discontinued_date is not null
fetch first 20 rows only;
```

17. Inspect the results.

PRODUCT_KEY	INTRODUCTION_DATE
30134	2003-01-01 00:00:00.000
30139	2003-01-01 00:00:00.000
30143	2003-01-01 00:00:00.000
30146	2003-01-01 00:00:00.000
30147	2003-01-01 00:00:00.000
30154	2003-01-01 00:00:00.000
30156	2003-01-01 00:00:00.000
30159	2003-01-01 00:00:00.000
30160	2003-01-01 00:00:00.000
30162	2003-01-01 00:00:00.000
30169	2003-01-01 00:00:00.000
30174	2003-01-01 00:00:00.000
30178	2003-01-01 00:00:00.000
30186	2003-01-01 00:00:00.000
30199	2003-01-01 00:00:00.000
30202	2003-01-01 00:00:00.000
30204	2003-01-01 00:00:00.000
30205	2003-01-01 00:00:00.000
30213	2003-01-01 00:00:00.000
30217	2003-01-01 00:00:00.000

20 rows in results(first row: 0.48s; total: 0.51s)

Results:
In this demonstration, you created and ran Big SQL queries that join data from multiple tables as well as perform aggregations and other SQL operations.