IBM Training

Checkpoint solutions

1. What is one of the reasons to use Big SQL?
   - Want to access your Hadoop data without using MapReduce
   - Do not want to learn new languages like Hive or JAQL
   - No deep learning curve because Big SQL uses standard 2011 query structure.
2. List the two ways you can work with Big SQL.
   - JSqsh
   - Web tooling from DSM
3. When creating a Big SQL table, what is the keyword that you need to use to ensure that Big SQL creates the table within the HDFS?
   - Hadoop

Checkpoint solutions

IBM Training

Demonstration 1

Connecting to the IBM Big SQL Server

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

- Connect to the IBM Big SQL Server
- Use JSqsh or the Web Tooling to run SQL queries
- Create Big SQL schemas and tables

Demonstration 1: Connecting to the IBM Big SQL Server

Demonstration 1: Connecting to the IBM Big SQL Server

Purpose:

You will see how to connect to the Big SQL Server, a component of IBM BigInsights. In particular, you will connect to the Big SQL server using JSqsh, a CLI for Big SQL. JSqsh, pronounce "jay-skwish" is an open source project for querying JDBC databases, such as Big SQL. In this demonstration, you will see how to get up and running with JSqsh. This demonstration will also show you how to explore the Big SQL service through Apache Ambari, another open source project for cluster manager.

Estimated time: 60 minutes

User/Password: **biadmin/biadmin**

root/dalvm3

Services Password: **ibm2blue**

Task 1. Configure your image.

Important: Occasionally, when you suspend and resume the VM image, the network may assign a different IP address than the one you had configured. In these instances, the Ambari console and the services will not run. You will need to update /etc/hosts file with the newly assigned IP address to continue working with the image. No restart of the VM image is necessary - just give it a couple of minutes, at most. In some cases, you may need to restart the Ambari server, using *ambari-server restart* from the command line.

1. To open a new terminal, right-click the desktop, and then click **Open in Terminal**.
2. Type `ifconfig` to check for the current assigned IP address.
3. Take note of the IP address next to **inet**.

   Next, you need to edit the /etc/hosts file to map the hostname to the IP address.
4. To switch to root user, type `su -` and if prompted for a password, type `dalvm3`.
5. To open the */etc/hosts* file, type `gedit /etc/hosts`.
6. Ensure that the contents of the file are similar to the following:

   ```
   10.0.0.118 ibmclass.localdomain ibmclass
   127.0.0.1 localhost.localdomain localhost
   ```
7. Update with the IP address for ibmclass from step 3.
8. Save and exit the file, and then close the terminal.

Task 2. Start the BigInsights components.

If all of the components have been started already, you may skip this task. Otherwise, follow the steps in this task to start up all of the components.

You will start up all the services via the Ambari console to ensure that everything is ready for the lab. You may stop what you don't need later, but for now, you will start everything.

1. Launch **Firefox**, and then if necessary, navigate to the **Ambari** login page, **http://ibmclass.localdomain:8080**.
2. Log in to the **Ambari** console as **admin/admin**.

   On the left side of the browser are the statuses of all the services. If any are currently yellow, wait for several minutes for them to become red before you start them up.
3. Once all the statuses are red, at the bottom of the left side, click **Actions** and then click **Start All** to start up the services.
4. In the Confirmation dialog, click **OK**.

   This will take a while to complete to complete.
5. When the services have started successfully, click **OK**.

   If your Web browser returns an error instead of a sign in screen, verify that the Ambari server has been started and that you have the correct URL for it. If needed, launch Ambari manually: log into the node containing the Ambari server as `root` and issue this command: `ambari-server start`

Task 3. Start up the Big SQL service.

1. Inside the Ambari console, ensure that **BigInsights - Big SQL** has been started.

   Big SQL requires that the monitoring utility package is started as well.
2. Open a terminal and switch to the root user.

   `su -`

   If prompted for a password: `dalvm3`
3. Change directory to the following path:

   `cd /usr/ibmpacks/bigsql/4.0/dsm/1.1/ibm-datasrvrmgr/bin/`
4. Run the script `/dsmKnoxSetup.sh -knoxHost ibmclass.localdomain`

   ibmclass.localdomain is the host where the Knox gateway is running.

   Remember, whenever you restart the Knox server, you will have to run the dsmKnoxSetup script again.

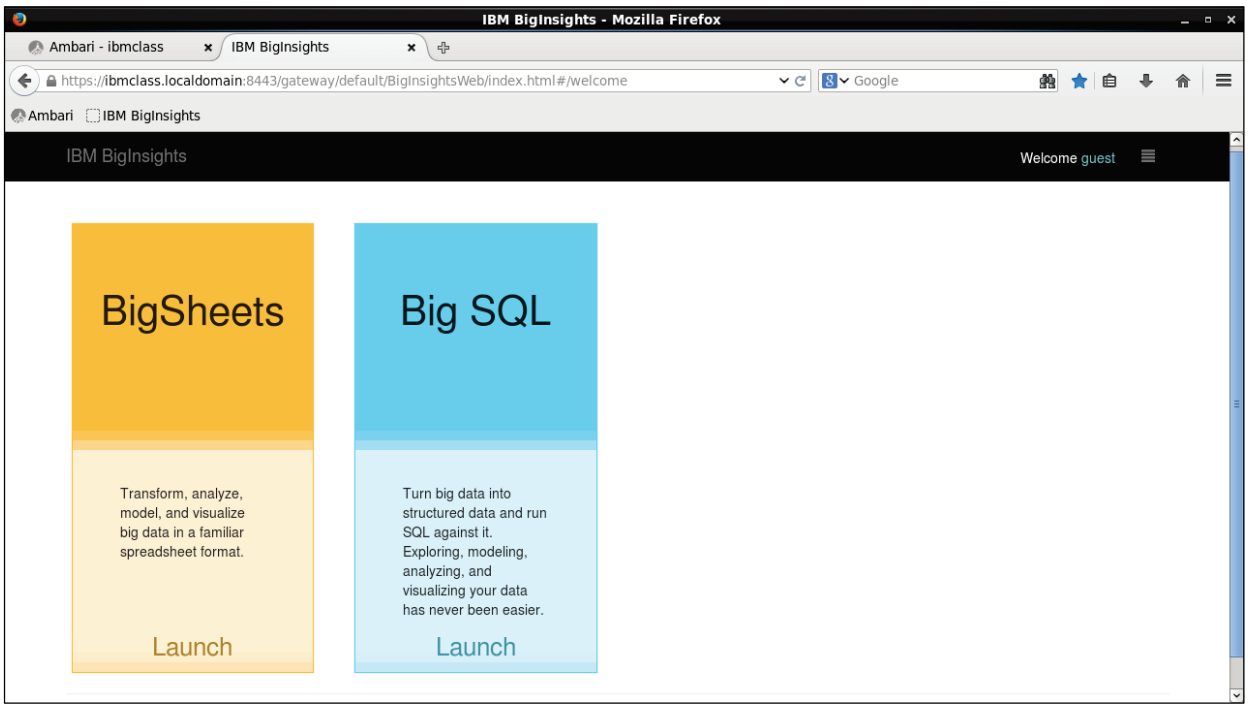
5. If prompted with the message, Do you wish to continue running with the above value (select 1 or 2), select **1**.

   To use Big SQL, you will need to access BigInsights Home. The BigInsights Home requires the LDAP server to be started.
6. Click the **Knox** component.
7. Under the **Service Actions** dropdown on the upper right, select **Start Demo LDAP**.
8. Click **OK** to close the confirmation window.
9. In Firefox, open a new tab, and navigate to the **Web UI (BigInsights Home)** page with the following URL.

   https://ibmclass.localdomain:8443/gateway/default/BigInsightsWeb/index.html
10. If promoted to login, enter **guest / guest-password**.

    It should be saved in the Firefox browser so you can click ok to continue with the login.

    The results appear as follows:



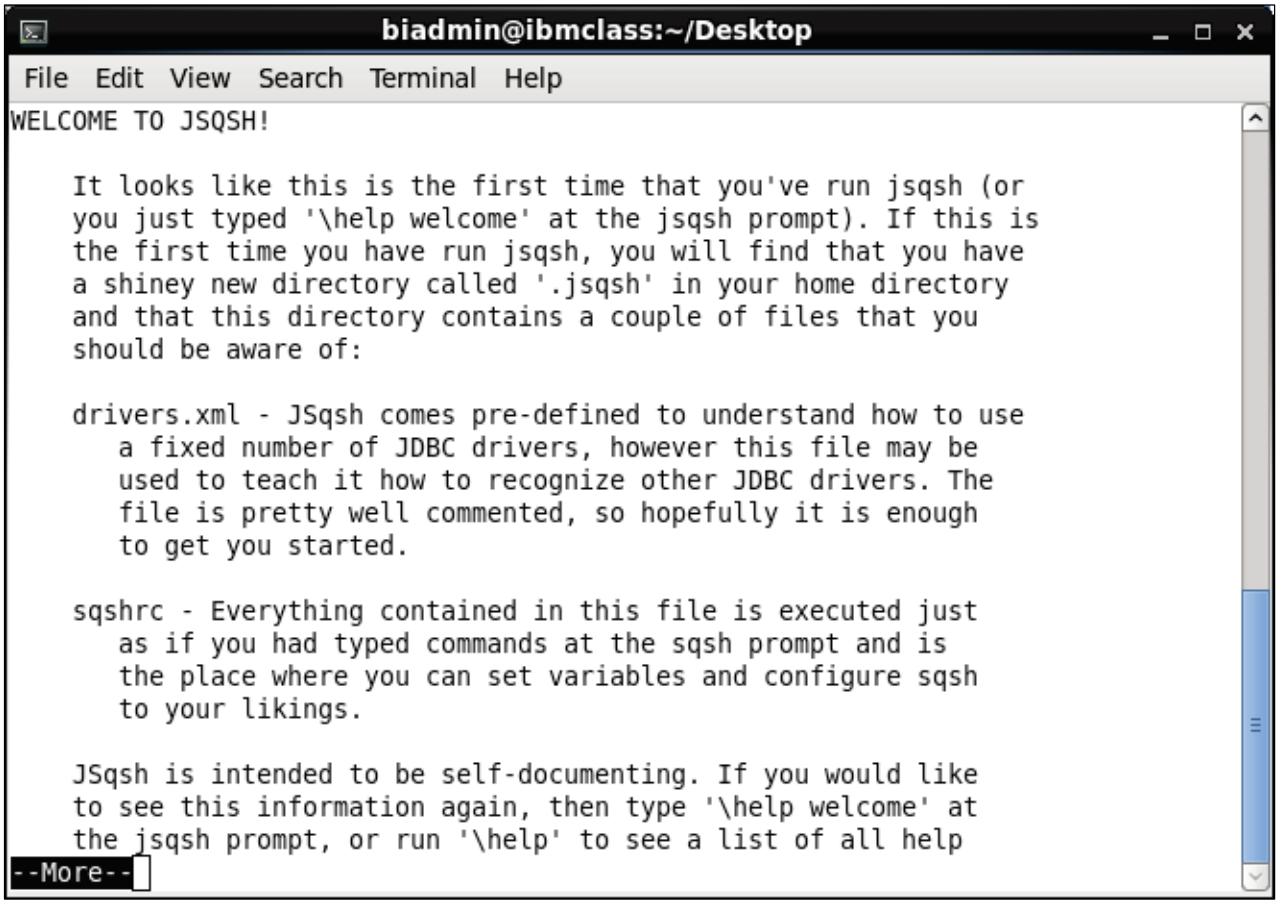
You will now be able to access Big SQL via the **BigInsights Home** page. Check to see that it is available.

Task 4. Connecting to Big SQL using JSqsh.

1. Right-click on the desktop and select **Open in Terminal**.
2. Launch the JSqsh shell:

   `/usr/ibmpacks/common-utils/jsqsh/2.14/bin/jsqsh`

   If it is your first time, you will see a welcome screen for a setup.
3. Press **Enter** to continue and enter **c** to launch the connection wizard.

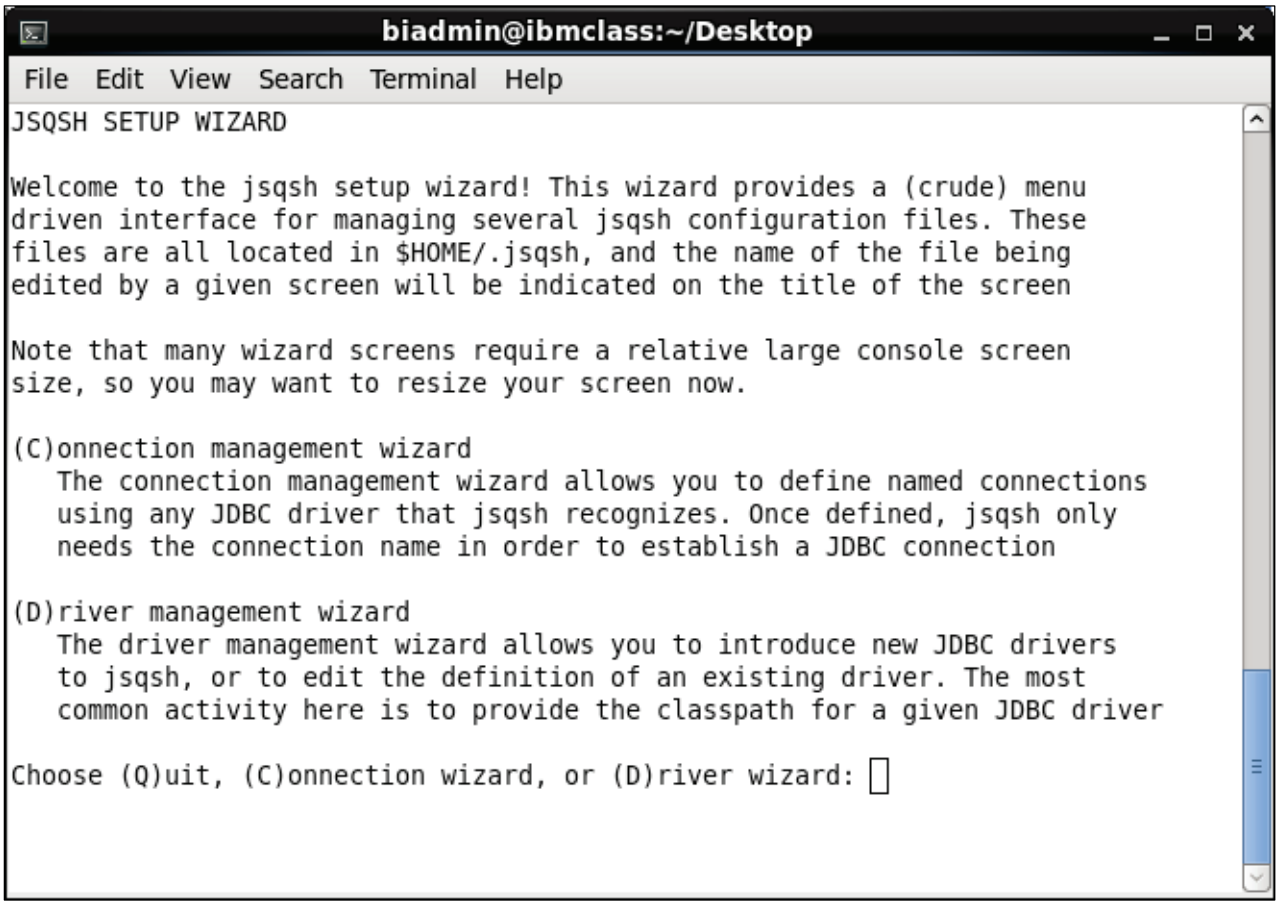


If this is not your first time, you will get a prompt that looks like this:

You want to run the setup to make sure you have the right connection information.

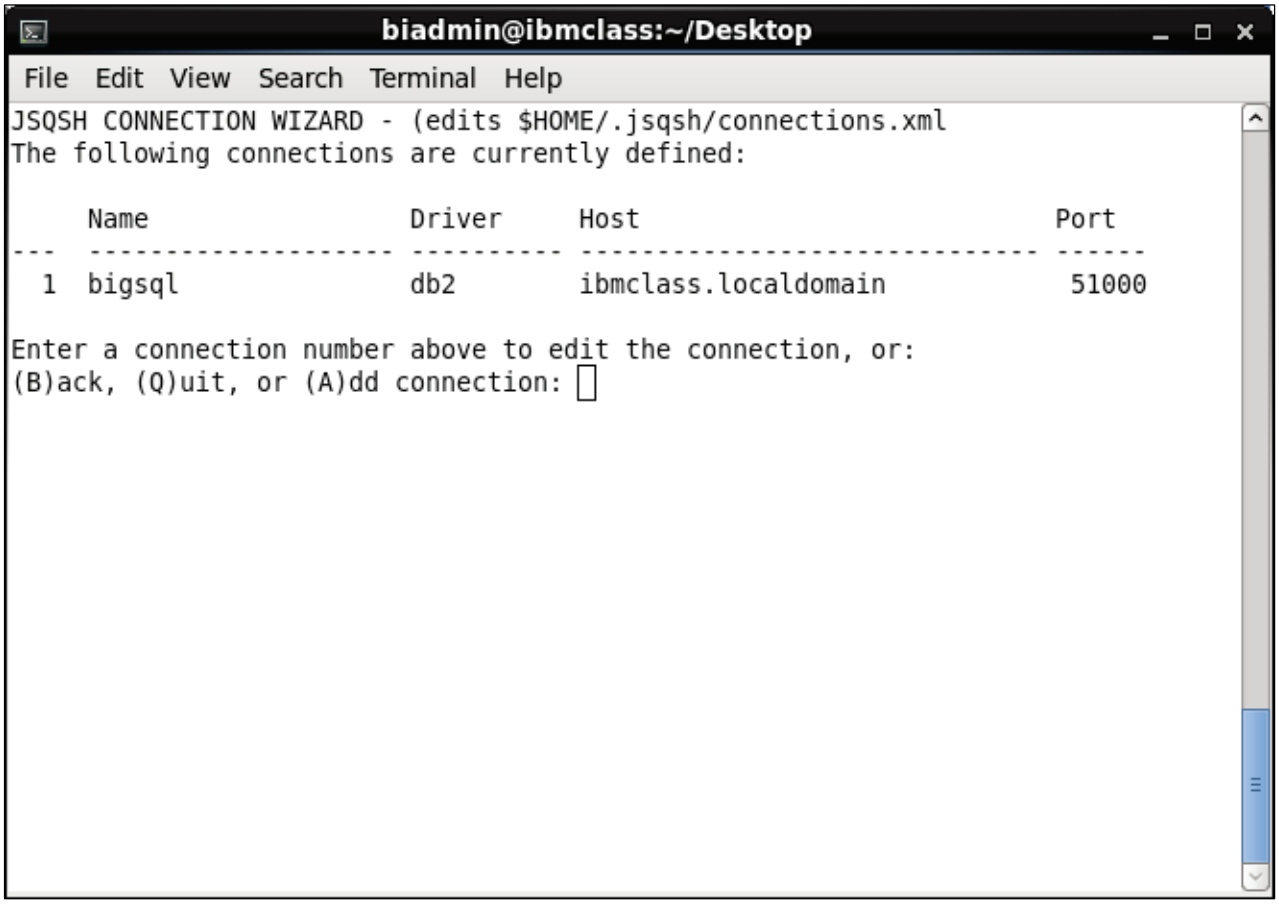
4. Type:

```
\setup
```

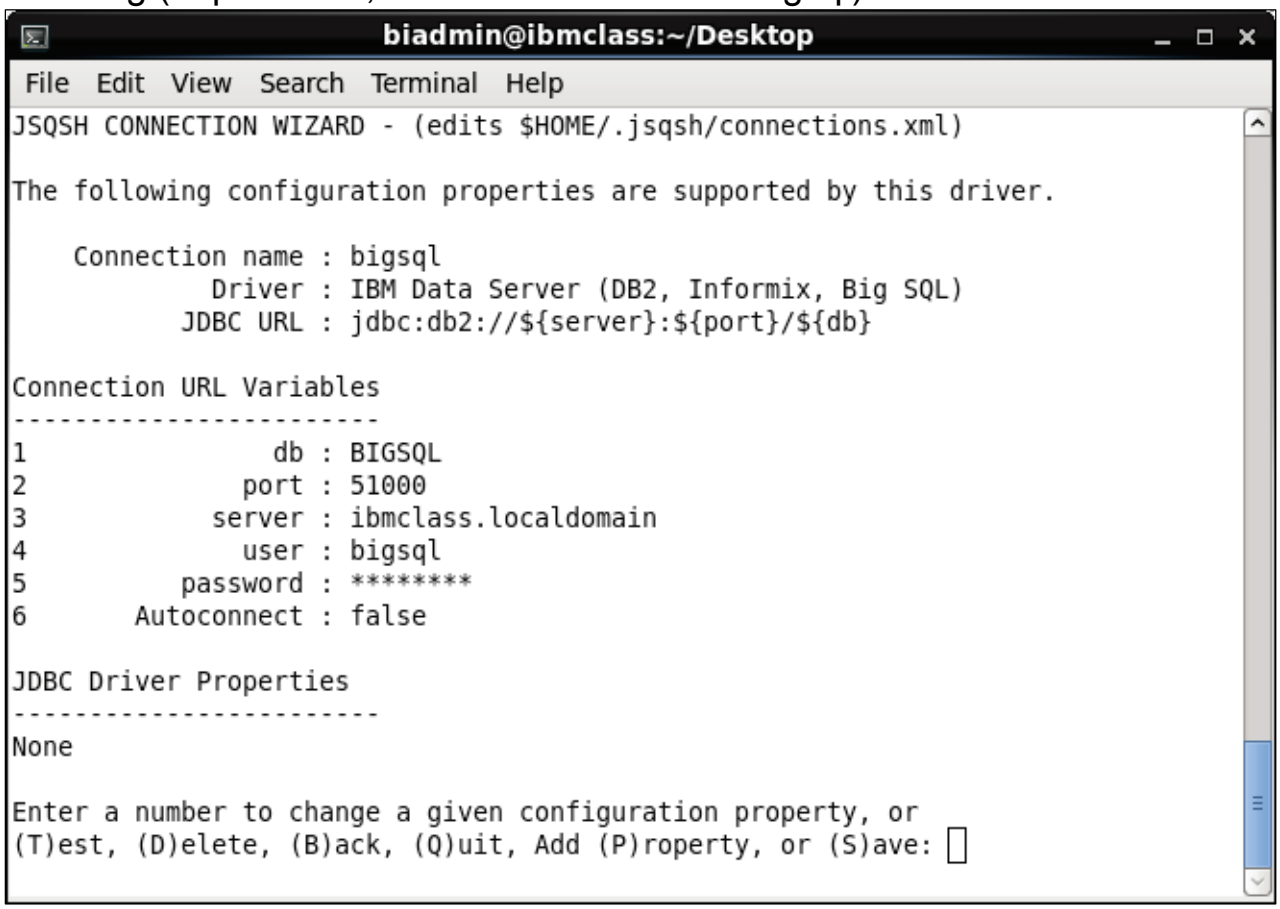


5. Enter **C** to start the connection wizard.

6. Enter **1**, to select the bigsql connection:



For each of the Connection URL variables, ensure yours matches the following (In particular, make sure the user is bigsql):



You want to enter in and save the password so that you can connect automatically for the purposes of this demonstration.

7. Enter **5**, and type in the bigsql service password: **ibm2blue**.
8. Enter **T** to test the connection.
9. Enter **S** to save your configurations.
10. Enter **Q** to quit.

    At this point, you have successfully connected to the *bigsql* schema/database.

Task 5. Getting help for JSqsh.

1. From JSqsh, type **\help** to display a list of available help categories.

```
1> \help
Available help categories. Use "\help <category>" to display topics within that
category
+----------+----------------------------------------------+
| Category | Description                                  |
+----------+----------------------------------------------+
| commands | Help on all avaiable commands                |
| vars     | Help on all avaiable configuration variables |
| topics   | General help topics for jsqsh               |
+----------+----------------------------------------------+
1>
```

2. Type **\help commands** to display help for supported commands. A partial list of supported commands is displayed on the initial screen.

```
1> \help commands
Available commands. Use "\help <command>" to display detailed help for a given c
ommand
+------------+-----------------------------------------------------------------+
| Command    | Description                                                     |
+------------+-----------------------------------------------------------------+
| \alias     | Creates an alias                                                |
| \buf-appen | Appends the contents of one SQL buffer into another             |
| d          |                                                                 |
| \buf-copy  | Copies the contents of one SQL buffer into another              |
| \buf-edit  | Edits a SQL buffer                                              |
| \buf-load  | Loads an external file into a SQL buffer                        |
| \call      | Call a prepared statement                                       |
| \connect   | Establishes a connection to a database.                         |
| \create    | Generates a CREATE TABLE using table definitions                |
| \databases | Displays set of available databases (catalogs)                  |
| \debug     | (Internal) Used to enable debugging                             |
| \describe  | Displays a description of a database object                     |
| \diff      | Compares results from multiple sessions                         |
| \drivers   | Displays a list of JDBC drivers known by jsqsh.                 |
| \echo      | Displays a line of text.                                        |
| \end       | Ends the current session                                        |
| \eval      | Read and execute an input file full of SQL                      |
| \globals   | Displays all global variables                                   |
--More--
```

Press the space bar to display the next page or q to quit the display of help information.

Task 6. Executing basic Big SQL statements.

In this section, you will execute simple JSqsh commands and Big SQL queries so that you can become familiar with the JSqsh shell.

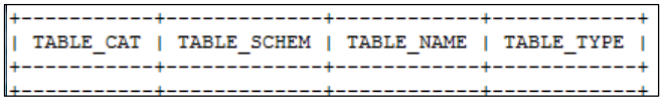
1. From the **JSqsh** shell, connect to your Big SQL server using the connection **bigsql** you created in a previous task.

```
\connect bigsql
```

If prompted for a password, type ibm2blue.

2. Type **\show tables -e | more** to display essential information about all available tables one page at a time.

   If you're working with a newly installed Big SQL server, your results will appear similar to those below.



```
+-----------+-------------+------------+------------+
| TABLE_CAT | TABLE_SCHEM | TABLE_NAME | TABLE_TYPE |
+-----------+-------------+------------+------------+
+-----------+-------------+------------+------------+
```

3. Type the following command into JSqsh to create a simple Hadoop table:

```
create hadoop table test1 (col1 int, col2 varchar(5));
```

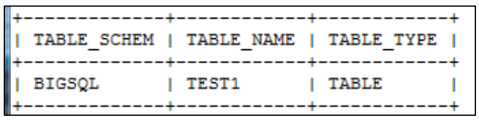
   Because you didn't specify a schema name for the table it was created in your default schema, which is the user name specified in your JDBC connection. This is equivalent to

```
create hadoop table yourID.test1 (col1 int, col2
varchar(5));
```

   Where *yourID* is the user name for your connection. In a previous task, you created a connection using the bigsql user ID, so your table is BIGSQL.TEST1.

4. Display all tables in the current schema with the **\tables** command.

```
\tables
```



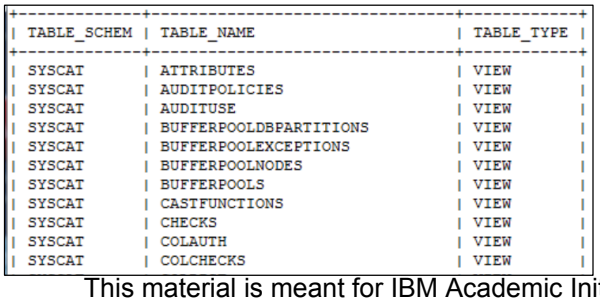
```
+-------------+------------+------------+
| TABLE_SCHEM | TABLE_NAME | TABLE_TYPE |
+-------------+------------+------------+
| BIGSQL      | TEST1      | TABLE      |
+-------------+------------+------------+
```

   Optionally, display information about tables and views created in other schemas, such as the SYSCAT schema used for the Big SQL catalog.

5. Specify the schema name in upper case since it will be used directly to filter the list of tables.

```
\tables -s SYSCAT
```

   Partial results are shown below.



```
+-------------+-------------------------------+------------+
| TABLE_SCHEM | TABLE_NAME                    | TABLE_TYPE |
+-------------+-------------------------------+------------+
| SYSCAT      | ATTRIBUTES                    | VIEW       |
| SYSCAT      | AUDITPOLICIES                 | VIEW       |
| SYSCAT      | AUDITUSE                      | VIEW       |
| SYSCAT      | BUFFERPOOLDBPARTITIONS        | VIEW       |
| SYSCAT      | BUFFERPOOLEXCEPTIONS          | VIEW       |
| SYSCAT      | BUFFERPOOLNODES               | VIEW       |
| SYSCAT      | BUFFERPOOLS                   | VIEW       |
| SYSCAT      | CASTFUNCTIONS                 | VIEW       |
| SYSCAT      | CHECKS                        | VIEW       |
| SYSCAT      | COLAUTH                       | VIEW       |
| SYSCAT      | COLCHECKS                     | VIEW       |
```

6. Insert a row into your table.

```
insert into test1 values (1, 'one');
```

This form of the INSERT statement (INSERT INTO … VALUES …) should be used for test purposes only because the operation will not be parallelized on your cluster. To populate a table with data in a manner that exploits parallel processing, use the Big SQL LOAD command, INSERT INTO … SELECT FROM statement, or CREATE TABLE AS … SELECT statement. You'll learn more about these commands later.

7. To view the metadata about a table, use the **\describe** command with the fully qualified table name in upper case.

```
\describe BIGSQL.TEST1
```

```
+-------------+-------------+-----------+-------------+----------------+-------------+
| TABLE_SCHEM | COLUMN_NAME | TYPE_NAME | COLUMN_SIZE | DECIMAL_DIGITS | IS_NULLABLE |
+-------------+-------------+-----------+-------------+----------------+-------------+
| BIGSQL      | COL1        | INTEGER   |          10 |              0 | YES         |
| BIGSQL      | COL2        | VARCHAR   |           5 |         [NULL] | YES         |
+-------------+-------------+-----------+-------------+----------------+-------------+
```

8. Optionally, query the system for metadata about this table:

```
select tabschema, colname, colno, typename, length
from syscat.columns
where tabschema = USER and tabname= 'TEST1';
```

Once again, notice that you used the table name in upper case in these queries and \describe command. This is because table and column names are folded to upper case in the system catalog tables.

You can split the query across multiple lines in the JSqsh shell if you'd like. Whenever you press **Enter**, the shell will provide another line for you to continue your command or SQL statement. A semi-colon or **go** command causes your SQL statement to execute.

```
+-----------+---------+-------+----------+--------+
| TABSCHEMA | COLNAME | COLNO | TYPENAME | LENGTH |
+-----------+---------+-------+----------+--------+
| BIGSQL    | COL1    |     0 | INTEGER  |      4 |
| BIGSQL    | COL2    |     1 | VARCHAR  |      5 |
+-----------+---------+-------+----------+--------+
2 rows in results(first row: 0.1s; total: 0.1s)
```

SYSCAT.COLUMNS is one of a number of views supplied over system catalog tables automatically maintained for you by the Big SQL service.

Issue a query that restricts the number of rows returned to 5.

9. Select the first 5 rows from SYSCAT.TABLES:

```
select tabschema, tabname from syscat.tables fetch
first 5 rows only;
```

```
+-----------+------------------------+
| TABSCHEMA | TABNAME                |
+-----------+------------------------+
| BIGSQL    | TEST1                  |
| SYSCAT    | ATTRIBUTES             |
| SYSCAT    | AUDITPOLICIES          |
| SYSCAT    | AUDITUSE               |
| SYSCAT    | BUFFERPOOLDBPARTITIONS |
+-----------+------------------------+
5 rows in results(first row: 0.1s; total: 0.1s)
```

Restricting the number of rows returned by a query is a useful development technique when working with large volumes of data.

If you plan to use JSqsh frequently, it's worth exploring some additional features. This demonstration shows you how to recall previous commands, redirect output to local files, and execute scripts.

Review the history of commands you recently executed in the JSqsh shell.

10. Type `\history` and press **Enter**.

Note that previously run statements are prefixed with a number in parentheses. You can reference this number in the JSqsh shell to recall that query.

11. Type `!!` (two exclamation points, without spaces) to recall the previously run statement.

The previous statement selects the first 5 rows from SYSCAT.TABLES.

12. To run the statement, type a semi-colon on the following line.

```
5 rows in results(first row: 0.1s; total: 0.1s)
[bdvs1052.svl.ibm.com][bigsql] 1> \history
(1) create hadoop table test1 (col1 int, col2 varchar(5))
(2) insert into test1 values (1, 'one')
(3) select tabschema, colname, colno, typename, length
    from syscat.columns
    where tabschema = USER and tabname= 'TEST1'
(4) select tabschema, tabname from syscat.tables fetch first 5 rows only
[bdvs1052.svl.ibm.com][bigsql] 1> !!
[bdvs1052.svl.ibm.com][bigsql] 1> select tabschema, tabname from syscat.tables fetch first 5 rows only
[bdvs1052.svl.ibm.com][bigsql] 2> ;
+-----------+------------------------+
| TABSCHEMA | TABNAME                |
+-----------+------------------------+
| BIGSQL    | TEST1                  |
| SYSCAT    | ATTRIBUTES             |
| SYSCAT    | AUDITPOLICIES          |
| SYSCAT    | AUDITUSE               |
| SYSCAT    | BUFFERPOOLDBPARTITIONS |
+-----------+------------------------+
5 rows in results(first row: 0.1s; total: 0.2s)
```

Recall a previous SQL statement by referencing the number reported via the \history command. For example, if you wanted to recall the 4th statement, you would enter !4. After the statement is recalled, add a semi-column to the final line to run the statement.

You will experiment with JSqsh's ability to support piping of output to an external program.

13. Enter the following two lines on the command shell:

```
select tabschema, tabname from syscat.tables

go | more
```

The go statement in the second line causes the query on the first line to be executed. (Note that there is no semi-colon at the end of the SQL query on the first line. The semi-colon is a Big SQL short cut for the JSqsh go command.) The | more clause causes the output that results from running the query to be piped through the Unix/Linux more command to display one screen of content at a time.

Your results should look similar to this:

```
| TABSCHEMA | TABNAME                          |
+-----------+----------------------------------+
| BIGSQL    | TEST1                            |
| SYSCAT    | ATTRIBUTES                       |
| SYSCAT    | AUDITPOLICIES                    |
| SYSCAT    | AUDITUSE                         |
| SYSCAT    | BUFFERPOOLDBPARTITIONS           |
| SYSCAT    | BUFFERPOOLEXCEPTIONS             |
| SYSCAT    | BUFFERPOOLNODES                  |
| SYSCAT    | BUFFERPOOLS                      |
| SYSCAT    | CASTFUNCTIONS                    |
| SYSCAT    | CHECKS                           |
| SYSCAT    | COLAUTH                          |
| SYSCAT    | COLCHECKS                        |
| SYSCAT    | COLDIST                          |
| SYSCAT    | COLGROUPCOLS                     |
| SYSCAT    | COLGROUPDIST                     |
| SYSCAT    | COLGROUPDISTCOUNTS               |
| SYSCAT    | COLGROUPS                        |
| SYSCAT    | COLIDENTATTRIBUTES               |
| SYSCAT    | COLLATIONS                       |
| SYSCAT    | COLOPTIONS                       |
| SYSCAT    | COLUMNS                          |
| SYSCAT    | COLUSE                           |
--More--443 rows in results(first row: 0.1s; total: 0.8s)
[bdvs1052.svl.ibm.com][bigsql] 1>
```

14. Since there are more than 400 rows to display in this example, enter `q` to quit displaying further results and return to the JSqsh shell.

Experiment with JSqsh's ability to redirect output to a local file rather than the console display.

15. Enter the following two lines on the command shell, adjusting the path information on the final line as needed for your environment:

```
select tabschema, colname, colno, typename, length
from syscat.columns
where tabschema = USER and tabname= 'TEST1'
go > $HOME/test1.out
```

This example directs the output of the query shown on the first line to the output file test1.out in your user's home directory.

16. Exit the shell:

```
quit
```

17. From a terminal window, view the output file:

```
cat $HOME/test1.out
```

```
[bigdata@bdvs1054 ~]$ cat $HOME/test1.out
+-----------+---------+-------+----------+--------+
| TABSCHEMA | COLNAME | COLNO | TYPENAME | LENGTH |
+-----------+---------+-------+----------+--------+
| BIGSQL    | COL1    |     0 | INTEGER  |      4 |
| BIGSQL    | COL2    |     1 | VARCHAR  |      5 |
+-----------+---------+-------+----------+--------+
```

Invoke JSqsh using an input file containing Big SQL commands to be executed. Maintaining SQL script files can be quite handy for repeatedly executing various queries.

From the Unix/Linux command line, use any available editor to create a new file in your local directory named test.sql.

18. Type:

```
vi test.sql
```

a. Add the following 2 queries into your file

```
select tabschema, tabname from syscat.tables fetch
first 5 rows only;

select tabschema, colname, colno, typename, length
from syscat.columns
fetch first 10 rows only;
```

b. Save your file (select ‘esc’ to exit INSERT mode then type :wq) and return to the command line.

c. Invoke JSQSH, instructing it to connect to your Big SQL database and execute the contents of the script you just created. (You may need to adjust the path or user information shown below to match your environment.)

```
/usr/ibmpacks/common-utils/jsqsh/2.14/bin/jsqsh
bigsql < test.sql
```

In this example, bigsql is the name of the database connection you created in an earlier task.

d. Inspect the output.

As you will see, JSQSH executes each instruction and displays its output. (Partial results are shown below.)

```
[bigdata@bdvs1054 ~]$ /usr/ibmpacks/bigsql/4.0/jsqsh/bin/jsqsh bigsql -P bigsql < test.sql
JSqsh Release 2.11, Copyright (C) 2007-2015, Scott C. Gray
Type \help for available help topics. Using JLine.
[bdvs1052.svl.ibm.com][bigsql] 1> select tabschema, tabname from syscat.tables fetch first 5 rows only
+-----------+-------------------------+
| TABSCHEMA | TABNAME                 |
+-----------+-------------------------+
| BIGSQL    | TEST1                   |
| SYSCAT    | ATTRIBUTES              |
| SYSCAT    | AUDITPOLICIES           |
| SYSCAT    | AUDITUSE                |
| SYSCAT    | BUFFERPOOLDBPARTITIONS  |
+-----------+-------------------------+
5 rows in results(first row: 0.3s; total: 0.3s)
[bdvs1052.svl.ibm.com][bigsql] 1>
[bdvs1052.svl.ibm.com][bigsql] 2> select tabschema, colname, colno, typename, length
[bdvs1052.svl.ibm.com][bigsql] 3> from syscat.columns
[bdvs1052.svl.ibm.com][bigsql] 4> fetch first 10 rows only;
+-----------+-------------+-------+-----------+-----------+
| TABSCHEMA | COLNAME     | COLNO | TYPENAME  |    LENGTH |
+-----------+-------------+-------+-----------+-----------+
| SYSIBM    | NAME        |     0 | VARCHAR   |       128 |
| SYSIBM    | CREATOR     |     1 | VARCHAR   |       128 |
| SYSIBM    | TYPE        |     2 | CHARACTER |         1 |
| SYSIBM    | CTIME       |     3 | TIMESTAMP |        10 |
| SYSIBM    | REMARKS     |     4 | VARCHAR   |       254 |
| SYSIBM    | PACKED_DESC |     5 | BLOB      | 133169152 |
| SYSIBM    | VIEW_DESC   |     6 | BLOB      |   4190000 |
| SYSIBM    | COLCOUNT    |     7 | SMALLINT  |         2 |
| SYSIBM    | FID         |     8 | SMALLINT  |         2 |
| SYSIBM    | TID         |     9 | SMALLINT  |         2 |
+-----------+-------------+-------+-----------+-----------+
10 rows in results(first row: 0.1s; total: 0.1s)
```

Next, you will clean up the database.

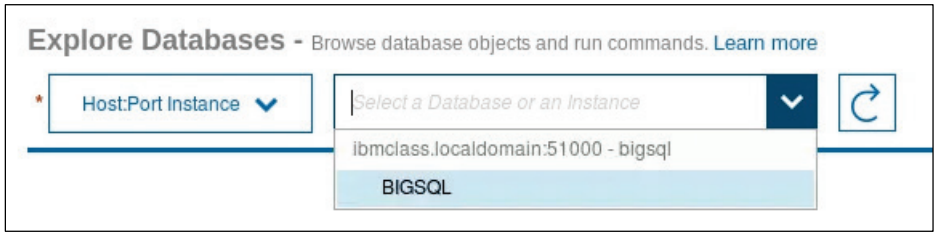
19. Launch the JSqsh shell again and issue this command:

```
drop table test1;
```

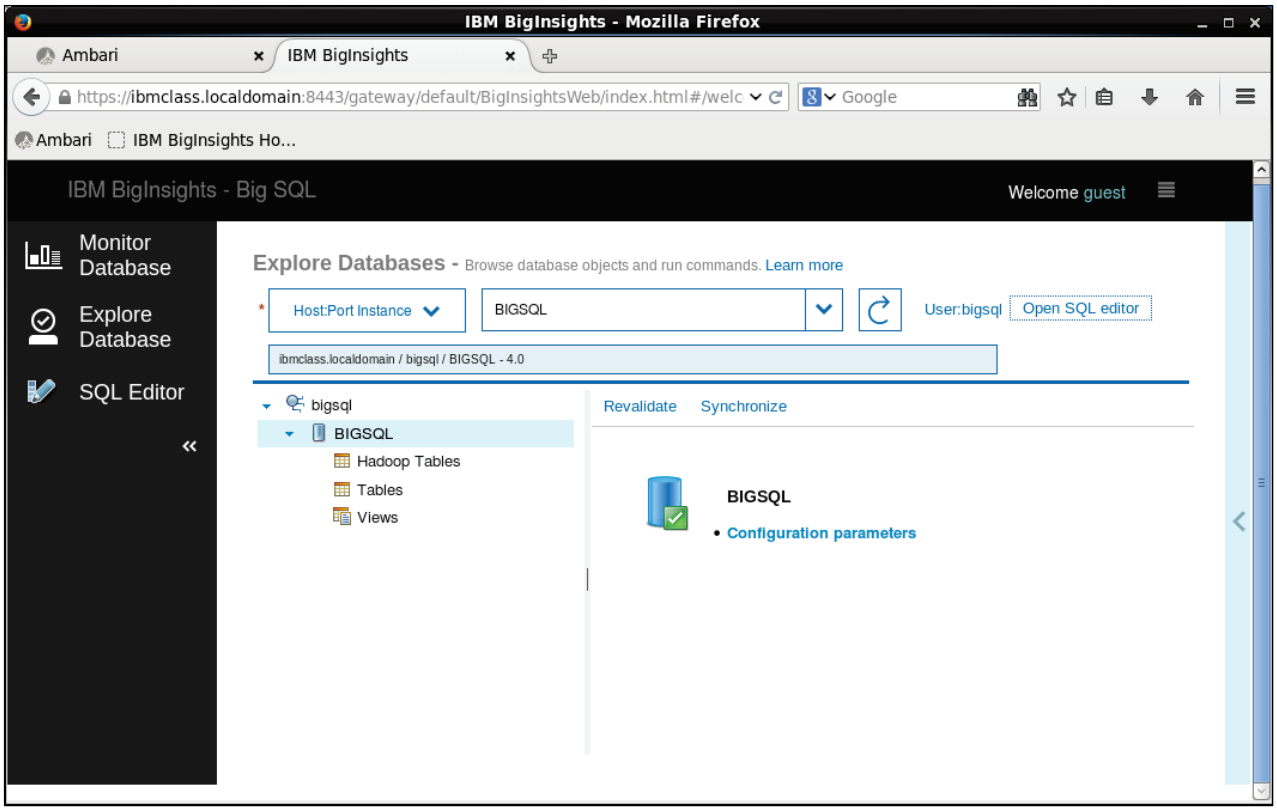
There's more to JSqsh than this short lab can cover. Visit the JSqsh wiki (https://github.com/scgray/jsqsh/wiki) to learn more.

Task 7. Exploring Big SQL through Ambari.

1. Launch BigInsights Home (https://ibmclass.localdomain:8443/gateway/default/BigInsightsWeb/index.html#/welcome) and click on the Big SQL service to open up the Big SQL page.
2. Click on **Explore Database** to see the default BIGSQL table that you created earlier.
3. Select the BIGSQL instance:



4. Login using the Big SQL credentials, use **bigsql/ibm2blue**.

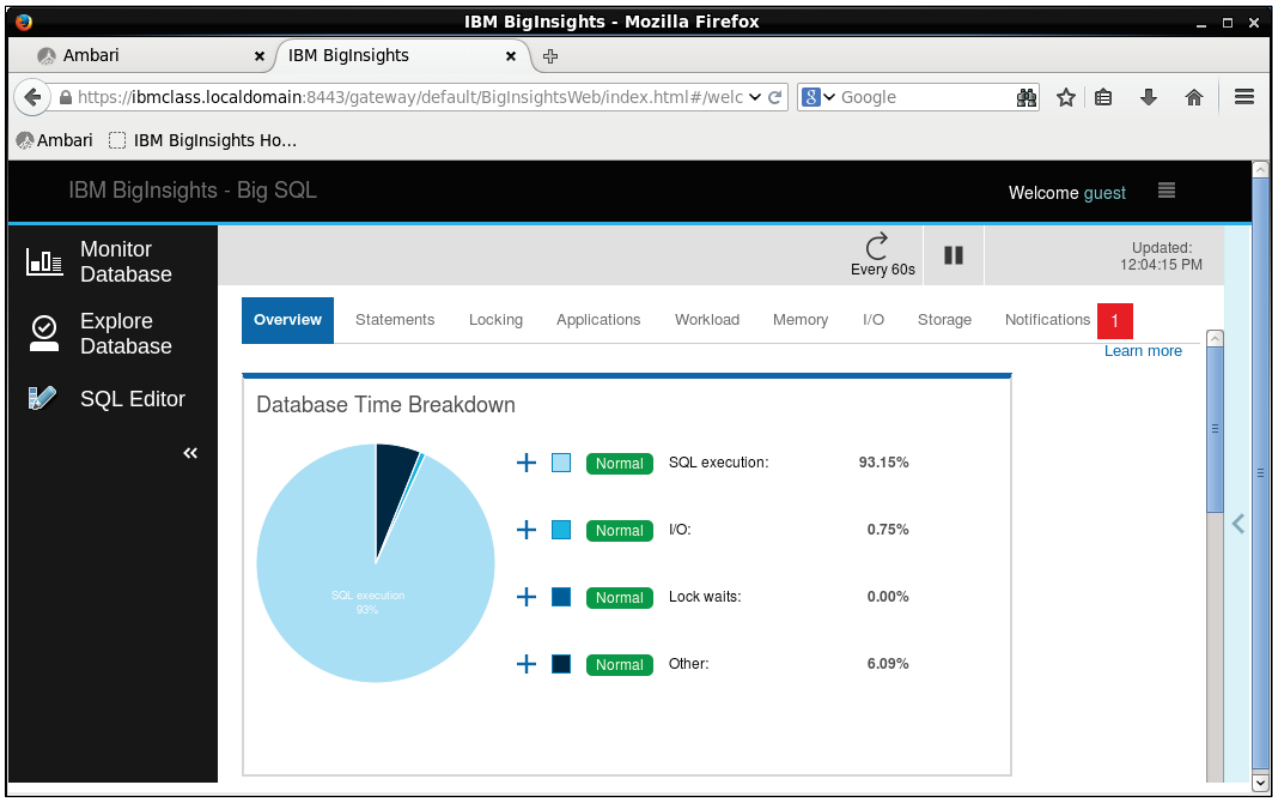


From there, you can explore your Big SQL database.

5. Click on the **Hadoop Tables** link to load the view of the tables.

   When you create Big SQL tables using the hadoop keyword, the tables would show up here. If you omit that keyword, and you did not specify the SYSHADOOP.COMPATIBILITY_MODE environment variable, then the tables would show up under Tables.

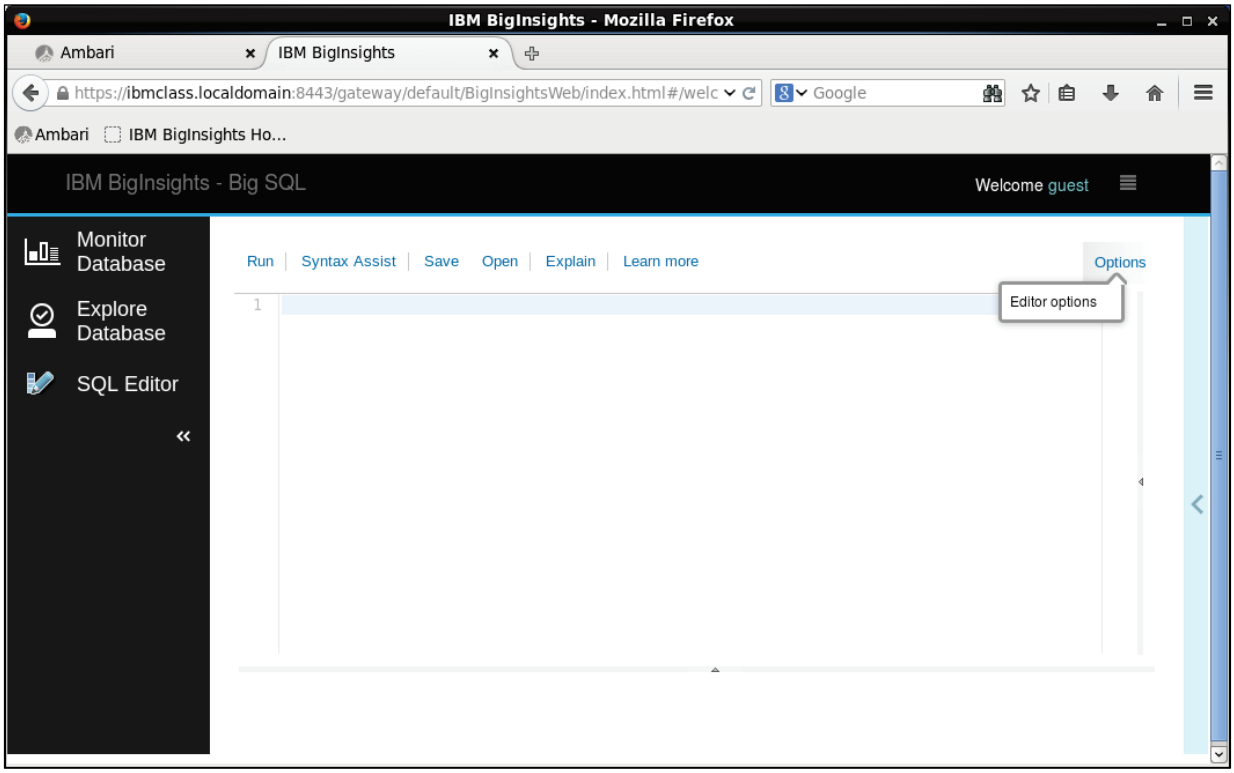
6. On the left panel, click on the **Monitor Database** to view the performance metrics for the database.



   There are number of tabs on the monitoring page that you can explore. Click on each of them to find out more.

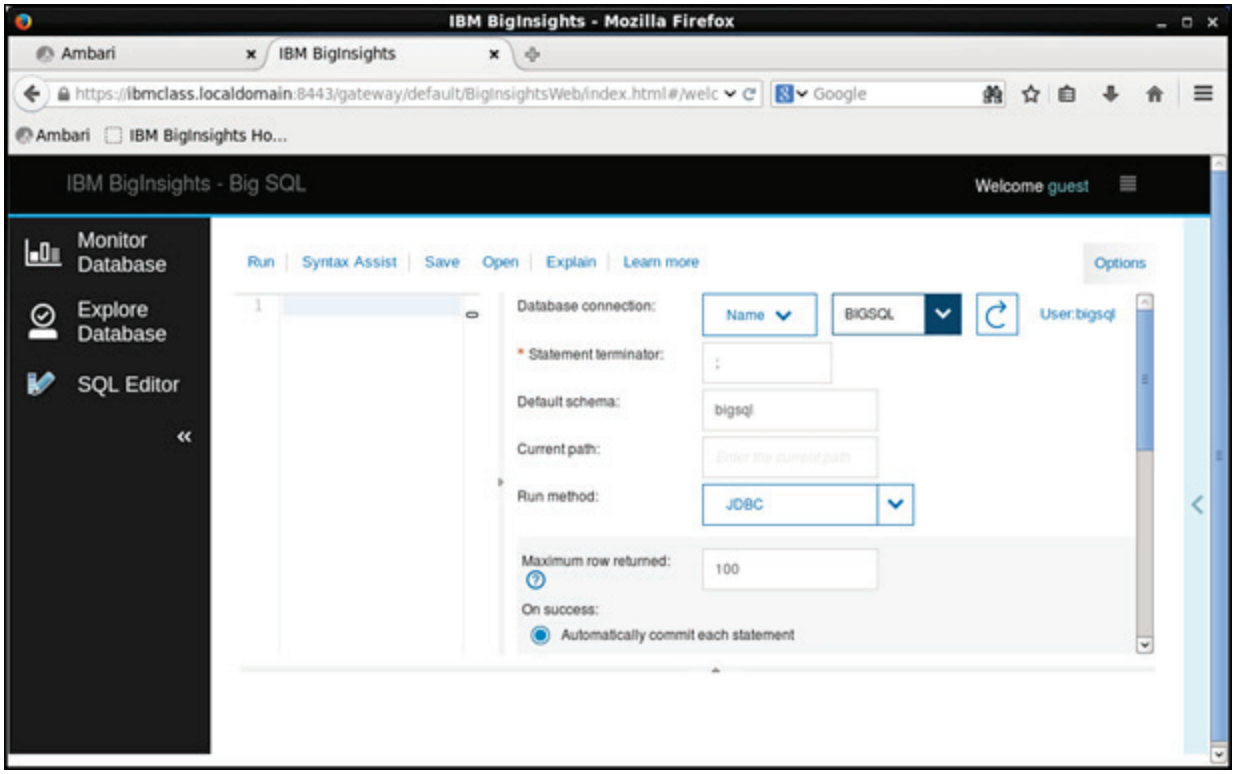
7. From the left panel, click **SQL Editor**.

8. Explore the **Options** menu to specify the database connection to use.



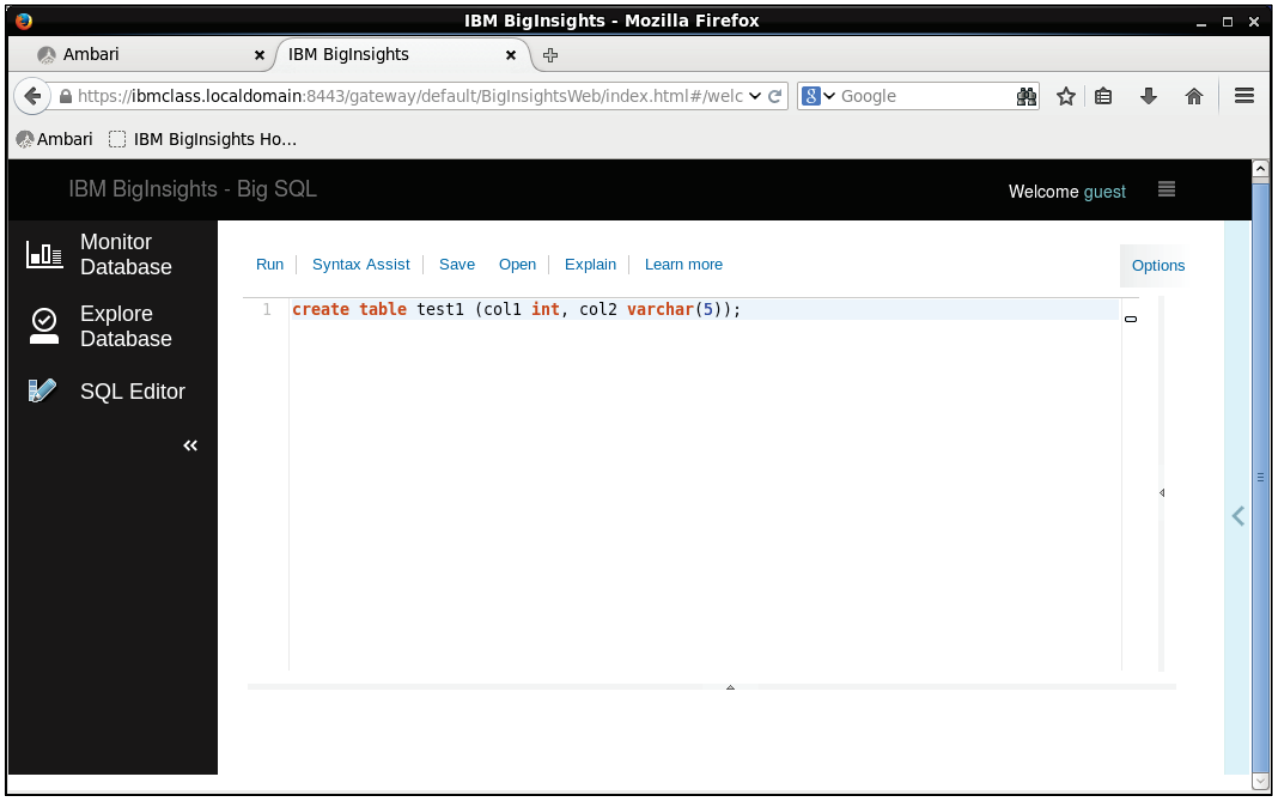
9. From the Database connection dropdown, select BIGSQL and in the Default schema field, type BIGSQL.

The results appear as follows:



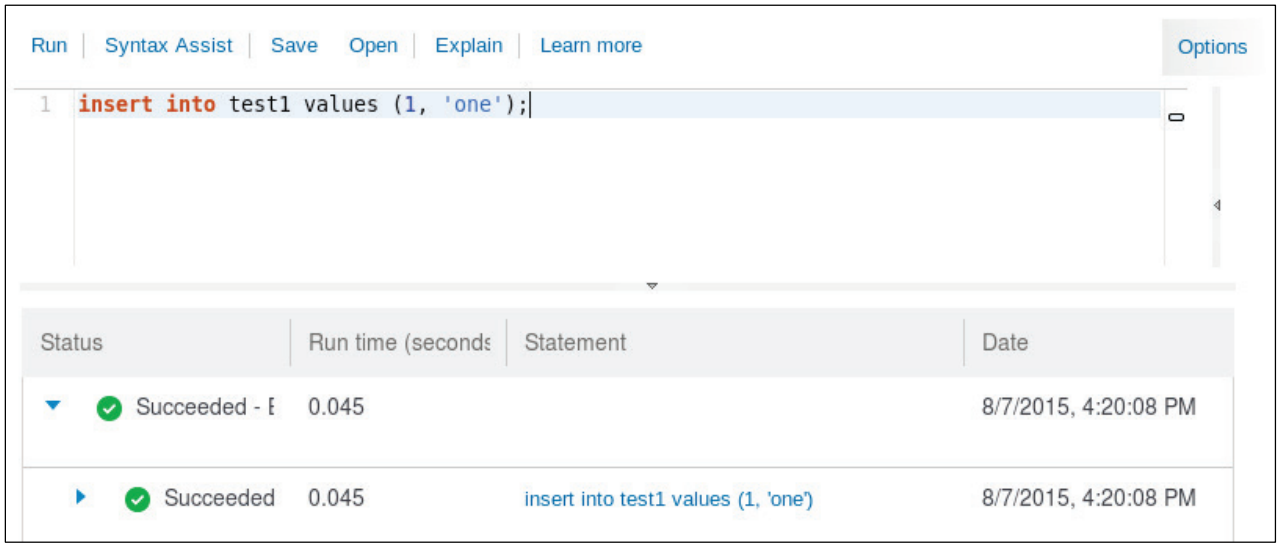
10. Collapse the Options pane after you are done.
11. In the Run query field, type:

```
create table test1 (col1 int, col2 varchar(5));
```



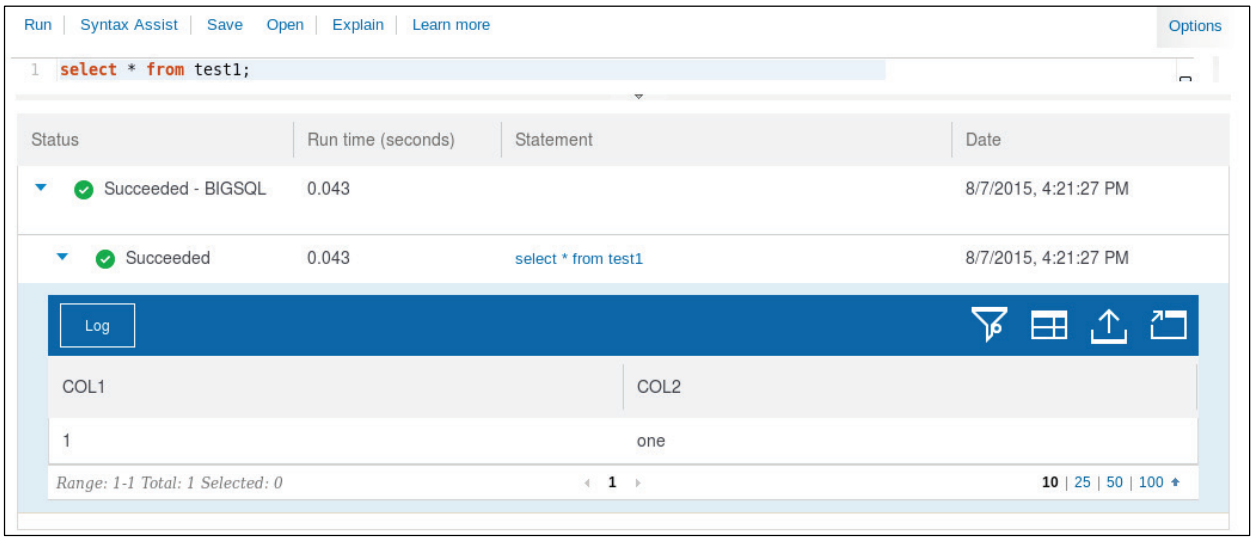
12. Click **Run** to execute the query.
13. Insert a row into the table:

```
insert into test1 values (1, 'one');
```



14. Select from the table:

```
select * from test1;
```



You are no longer require the table.

15. Drop the table:

```
drop table test1;
```

Task 8. Creating sample tables.

In this task, you will create several sample tables.

Determine the location of the sample data in your local file system and make a note of it. You will need to use this path specification when issuing LOAD commands later in this demonstration.

Subsequent examples in this section presume your sample data is in the **/usr/ibmpacks/bigsql/4.0/bigsql/samples/data** directory.

This is the location of the data in typical Big SQL installations.

Also note that the statements in this demonstration are provided in the scripts that are located in **/home/biadmin/labfiles/bigsql/BIG_SQL_Data_Analysis** for you to copy and paste the statements into your console of choice.

1. Using your choice of either JSqsh or the Big SQL service via Ambari, establish a connection to your Big SQL server following the standard process for your environment.

2. Create 8 tables in your default schema (bigsql). Issue each of the following CREATE TABLE statements one at a time, and verify that each completed successfully. If you are not able to create tables, refer to the troubleshooting section.

```
-- dimension table for region info
CREATE HADOOP TABLE IF NOT EXISTS go_region_dim
( country_key  INT NOT NULL
, country_code INT NOT NULL
, flag_image    VARCHAR(45)
, iso_three_letter_code  VARCHAR(9) NOT NULL
, iso_two_letter_code        VARCHAR(6) NOT NULL
, iso_three_digit_code   VARCHAR(9) NOT NULL
, region_key   INT NOT NULL
, region_code  INT NOT NULL
, region_enVARCHAR(90) NOT NULL
, country_en   VARCHAR(90) NOT NULL
, region_deVARCHAR(90), country_de VARCHAR(90), region_fr VARCHAR(90)
, country_fr   VARCHAR(90), region_ja   VARCHAR(90), country_ja VARCHAR(90)
, region_csVARCHAR(90), country_cs VARCHAR(90), region_da VARCHAR(90)
, country_da   VARCHAR(90), region_el   VARCHAR(90), country_el VARCHAR(90)
, region_esVARCHAR(90), country_es VARCHAR(90), region_fi VARCHAR(90)
, country_fi   VARCHAR(90), region_hu   VARCHAR(90), country_hu VARCHAR(90)
, region_idVARCHAR(90), country_id VARCHAR(90), region_it VARCHAR(90)
, country_it   VARCHAR(90), region_ko   VARCHAR(90), country_ko VARCHAR(90)
, region_msVARCHAR(90), country_ms VARCHAR(90), region_nl VARCHAR(90)
, country_nl   VARCHAR(90), region_no   VARCHAR(90), country_no VARCHAR(90)
, region_plVARCHAR(90), country_pl VARCHAR(90), region_pt VARCHAR(90)
, country_pt   VARCHAR(90), region_ru   VARCHAR(90), country_ru VARCHAR(90)
, region_scVARCHAR(90), country_sc VARCHAR(90), region_sv VARCHAR(90)
, country_sv   VARCHAR(90), region_tc   VARCHAR(90), country_tc VARCHAR(90)
, region_thVARCHAR(90), country_th VARCHAR(90)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
;

-- dimension table tracking method of order for the sale (e.g., Web, fax)
CREATE HADOOP TABLE IF NOT EXISTS sls_order_method_dim
( order_method_key   INT NOT NULL
, order_method_code  INT NOT NULL
, order_method_en    VARCHAR(90) NOT NULL
, order_method_de    VARCHAR(90), order_method_fr   VARCHAR(90)
, order_method_ja    VARCHAR(90), order_method_cs   VARCHAR(90)
, order_method_da    VARCHAR(90), order_method_el   VARCHAR(90)
, order_method_es    VARCHAR(90), order_method_fi   VARCHAR(90)
, order_method_hu    VARCHAR(90), order_method_id   VARCHAR(90)
, order_method_it    VARCHAR(90), order_method_ko   VARCHAR(90)
, order_method_ms    VARCHAR(90), order_method_nl   VARCHAR(90)
, order_method_no    VARCHAR(90), order_method_pl   VARCHAR(90)
, order_method_pt    VARCHAR(90), order_method_ru   VARCHAR(90)
, order_method_sc    VARCHAR(90), order_method_sv   VARCHAR(90)
, order_method_tc    VARCHAR(90), order_method_th   VARCHAR(90)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
```

```
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
;

-- look up table with product brand info in various languages
CREATE HADOOP TABLE IF NOT EXISTS sls_product_brand_lookup
 ( product_brand_code     INT NOT NULL
 , product_brand_en  VARCHAR(90) NOT NULL
 , product_brand_de  VARCHAR(90), product_brand_fr  VARCHAR(90)
 , product_brand_ja  VARCHAR(90), product_brand_cs  VARCHAR(90)
 , product_brand_da  VARCHAR(90), product_brand_el  VARCHAR(90)
 , product_brand_es  VARCHAR(90), product_brand_fi  VARCHAR(90)
 , product_brand_hu  VARCHAR(90), product_brand_id  VARCHAR(90)
 , product_brand_it  VARCHAR(90), product_brand_ko  VARCHAR(90)
 , product_brand_ms  VARCHAR(90), product_brand_nl  VARCHAR(90)
 , product_brand_no  VARCHAR(90), product_brand_pl  VARCHAR(90)
 , product_brand_pt  VARCHAR(90), product_brand_ru  VARCHAR(90)
 , product_brand_sc  VARCHAR(90), product_brand_sv  VARCHAR(90)
 , product_brand_tc  VARCHAR(90), product_brand_th  VARCHAR(90)
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 LINES TERMINATED BY '\n'
 STORED AS TEXTFILE
 ;

 -- product dimension table
 CREATE HADOOP TABLE IF NOT EXISTS sls_product_dim
 ( 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'
 STORED AS TEXTFILE
 ;

 -- look up table with product line info in various languages
 CREATE HADOOP TABLE IF NOT EXISTS sls_product_line_lookup
 ( product_line_code      INT NOT NULL
 , product_line_en   VARCHAR(90) NOT NULL
 , product_line_de   VARCHAR(90), product_line_fr   VARCHAR(90)
 , product_line_ja   VARCHAR(90), product_line_cs   VARCHAR(90)
 , product_line_da   VARCHAR(90), product_line_el   VARCHAR(90)
 , product_line_es   VARCHAR(90), product_line_fi   VARCHAR(90)
 , product_line_hu   VARCHAR(90), product_line_id   VARCHAR(90)
 , product_line_it   VARCHAR(90), product_line_ko   VARCHAR(90)
```

```
, product_line_ms   VARCHAR(90), product_line_nl   VARCHAR(90)
, product_line_no   VARCHAR(90), product_line_pl   VARCHAR(90)
, product_line_pt   VARCHAR(90), product_line_ru   VARCHAR(90)
, product_line_sc   VARCHAR(90), product_line_sv   VARCHAR(90)
, product_line_tc   VARCHAR(90), product_line_th   VARCHAR(90)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

-- look up table for products
CREATE HADOOP TABLE IF NOT EXISTS sls_product_lookup
( product_number    INT NOT NULL
, product_language  VARCHAR(30) NOT NULL
, product_name VARCHAR(150) NOT NULL
, product_description    VARCHAR(765)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

-- fact table for sales
CREATE HADOOP TABLE IF NOT EXISTS sls_sales_fact
( order_day_key     INT NOT NULL
, organization_key  INT NOT NULL
, employee_key           INT NOT NULL
, retailer_key           INT NOT NULL
, retailer_site_key      INT NOT NULL
, product_key  INT NOT NULL
, promotion_key     INT NOT NULL
, order_method_key  INT NOT NULL
, sales_order_key   INT NOT NULL
, ship_day_key           INT NOT NULL
, close_day_key     INT NOT NULL
, quantity     INT
, unit_cost    DOUBLE
, unit_price   DOUBLE
, unit_sale_price   DOUBLE
, gross_margin           DOUBLE
, sale_total   DOUBLE
, gross_profit           DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
;

-- fact table for marketing promotions
CREATE HADOOP TABLE IF NOT EXISTS mrk_promotion_fact
( organization_key  INT NOT NULL
, order_day_key     INT NOT NULL
, rtl_country_key   INT NOT NULL
, employee_key           INT NOT NULL
, retailer_key           INT NOT NULL
, product_key  INT NOT NULL
, promotion_key     INT NOT NULL
```

```
, sales_order_key   INT NOT NULL
, quantity     SMALLINT
, unit_cost    DOUBLE
, unit_price   DOUBLE
, unit_sale_price   DOUBLE
, gross_margin          DOUBLE
, sale_total   DOUBLE
, gross_profit          DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Let's briefly explore some aspects of the CREATE TABLE statements shown here. If you have a SQL background, the majority of these statements should be familiar to you. However, after the column specification, there are some additional clauses unique to Big SQL – clauses that enable it to exploit Hadoop storage mechanisms (in this case, Hive). The ROW FORMAT clause specifies that fields are to be terminated by tabs ("\t") and lines are to be terminated by new line characters ("\n"). The table will be stored in a TEXTFILE format, making it easy for a wide range of applications to work with. For details on these clauses, refer to the Apache Hive documentation.

Task 9. Troubleshooting - Optional.

Big SQL has some limitations running on a single node cluster (actually, anything less than a 3 node cluster) and for good reasons too. You do not ever want to have a single node cluster for your production environment.

In the training world, simpler is better. Your lab environment should have been configured with this fix to allow **creating** Big SQL tables, but if it wasn't, you can run this yourself:

1. Open up a new terminal.
2. Switch to the **bigsql** user.
3. Run this command to connect to the bigsql database:

```
db2 connect to bigsql

Database Connection Information
Database server        = DB2/LINUXX8664 10.6.3
SQL authorization ID   = BIGSQL
Local database alias   = BIGSQL
```


4. Run this command for the fix:

```
db2set DB2_DYNAMIC_PMAP=INCLUDE_HEAD_NODE
```

5. Restart the **Big SQL** service from **Ambari**.
6. This will allow you to create Big SQL tables (if you were not able to before).

Results:

You connected to the Big SQL Server, a component of IBM BigInsights. In particular, you connected to the Big SQL server using JSqsh, a CLI for Big SQL. You also so how to get up and running with JSqsh. This demonstration will also showed you how to explore the Big SQL service through Apache Ambari, another open source project for cluster manager.

IBM Training

Unit summary

- Understand how Big SQL fits in the Hadoop architecture
- Accessing and using Big SQL
- Creating Big SQL schemas and tables

Unit Summary