An IBM Proof of Technology

Getting started with Big SQL on Hadoop

Creating tables, loading data, and issuing queries with Big SQL 3.0 on InfoSphere BigInsights

Last updated: Nov. 3, 2014

An IBM Proof of Technology

Catalog Number

Contents

LAB 1 **OVERVIEW** 5
1.1. PRE-REQUISITES 6
1.2. WHAT YOU'LL LEARN 7
1.3. GETTING STARTED WITH YOUR VMWARE IMAGE 7
LAB 2 **USING THE BIG SQL COMMAND LINE INTERFACE (JSQSH)** 11
2.1. LAUNCHING THE WEB CONSOLE TO VERIFY BIGINSIGHTS SERVICES ARE UP AND RUNNING 11
2.2. UNDERSTANDING JSQSH CONNECTIONS 12
2.3. OPTIONAL: CREATING YOUR OWN DATABASE CONNECTION 15
2.4. GETTING HELP FOR JSQSH 18
2.5. ISSUING JSQSH COMMANDS AND BIG SQL QUERIES 19
LAB 3 **USING ECLIPSE** 27
3.1. LAUNCHING THE WEB CONSOLE TO VERIFY BIGINSIGHTS SERVICES ARE UP AND RUNNING 27
3.2. CREATING A BIG SQL CONNECTION IN ECLIPSE 28
3.3. CREATING A PROJECT AND A SQL SCRIPT FILE 34
3.4. CREATING AND ISSUING QUERIES 36
LAB 4 **QUERYING STRUCTURED DATA WITH BIG SQL** 39
4.1. CREATING SAMPLE TABLES AND LOADING SAMPLE DATA 39
4.2. QUERYING THE DATA WITH BIG SQL 45
4.3. CREATING AND WORKING WITH VIEWS 49
4.4. POPULATING A TABLE WITH 'INSERT INTO … SELECT' 50
4.5. OPTIONAL: STORING DATA IN AN ALTERNATE FILE FORMAT (PARQUET) 51
4.6. OPTIONAL: USING BIG SQL FROM A JDBC CLIENT APPLICATION (ECLIPSE) 53
LAB 5 **ANALYZING SOCIAL MEDIA DATA IN BIGSHEETS WITH BIG SQL** 59
5.1. CREATING AND CUSTOMIZING A BIGSHEETS WORKBOOK 59
5.2. CREATING A BIG SQL TABLE DIRECTLY FROM A WORKBOOK 65
5.3. EXPORTING YOUR WORKBOOK 66
5.4. OPTIONAL: USING BIG SQL TO WORK WITH DATA EXPORTED FROM BIGSHEETS 69
LAB 6 **WORKING WITH NON-TRADITIONAL DATA** 71
6.1. REGISTERING A SERDE 71
6.2. CREATING, POPULATING, AND QUERYING A TABLE THAT USES A SERDE 72
LAB 7 **USING ADVANCED BIG SQL FEATURES** 75
7.1. UNDERSTANDING YOUR DATA ACCESS PLAN (EXPLAIN) – FROM THE COMMAND LINE 75
7.2. COLLECTING STATISTICS WITH THE ANALYZE TABLE COMMAND 79
7.3. ENHANCING SQL SECURITY WITH FINE-GRAINED ACCESS CONTROL 80
LAB 8 **DEVELOPING AND EXECUTING SQL USER-DEFINED FUNCTIONS** 91
8.1. UNDERSTANDING UDFS 91
8.2. PREPARE JSQSH TO CREATE AND EXECUTE UDFS 92
8.3. CREATING AND EXECUTING A SCALAR UDF 93
8.4. OPTIONAL: INVOKING UDFS WITHOUT PROVIDING FULLY-QUALIFIED NAME 95
8.5. INCORPORATING IF/ELSE STATEMENTS 96
8.6. INCORPORATING WHILE LOOPS 98
8.7. INCORPORATING FOR LOOPS 99
8.8. CREATING A TABLE UDF 100
8.9. OPTIONAL: OVERLOADING UDFS AND DROPPING UDFS 102
LAB 9 **EXPLORING BIG SQL LOAD AND HADOOP COMMANDS** 104
9.1. LOADING DATA INTO BIG SQL TABLES FROM A LOCAL FILE 104
9.2. TRACKING REJECTED RECORDS 106
9.3. PREPARING TO LOAD DATA DIRECTLY FROM A RELATIONAL DBMS 107
9.4. LOADING DATA DIRECTLY FROM A RELATIONAL DBMS TABLE 109
9.5. LOADING DATA DIRECTLY FROM A RELATIONAL DBMS WITH SELECT 111
9.6. EXPLORING ADDITIONAL LOAD SCENARIOS 112
9.7. USING HADOOP COMMANDS TO MOVE DATA INTO A TABLE 114
9.8. DROPPING TABLES YOU CREATED IN THIS LAB 117
LAB 10 **SUMMARY** 118

Lab 1 Overview

In this hands-on lab, you'll learn how to work with Big SQL, a component of InfoSphere BigInsights, IBM's big data platform based on Apache Hadoop. In particular, you'll use Big SQL to query traditional structured data as well as data derived from social media sites.

Big SQL enables IT professionals to create tables and query data in BigInsights using familiar SQL statements. To do so, programmers use standard SQL syntax and, in some cases, SQL extensions created by IBM to make it easy to exploit certain Hadoop-based technologies. Big SQL shares query compiler technology with DB2 (a relational DBMS) and, as such, offers a wide breadth of SQL capabilities.

Organizations interested in Big SQL often have considerable SQL skills in-house, as well as a suite of SQL-based business intelligence applications and query/reporting tools. The idea of being able to leverage existing skills and tools — and perhaps reuse portions of existing applications — can be quite appealing to organizations new to Hadoop. Indeed, some companies with large data warehouses built on relational DBMS systems are looking to Hadoop-based platforms as a potential target for offloading "cold" or infrequently used data in a manner that still allows for query access. In other cases, organizations turn to Hadoop to analyze and filter non-traditional data (such as logs, sensor data, social media posts, etc.), ultimately feeding subsets or aggregations of this information to their relational warehouses to extend their view of products, customers, or services.

1.1. Pre-requisites

Before beginning this lab, you should have a working BigInsights environment launched with a Big SQL 3.0 server active. You must be able to log into your system with an account that has administrative privileges. Prior knowledge of industry-standard SQL is useful.

This lab was developed for the InfoSphere BigInsights 3.0 Quick Start Edition VMware image. As such, its lab exercises are based on the following information:

	User	Password
VM Image root account	root	password
VM Image lab user account	biadmin	biadmin
BigInsights Administrator	biadmin	biadmin
Big SQL Administrator	bigsql	bigsql
Lab user	biadmin	biadmin

Property	Value
Host name	bivm.ibm.com
BigInsights Web Console URL	http://bivm.ibm.com:8080
Big SQL database name	bigsql
Big SQL port number	51000

.

About the screen captures, sample code, and environment configuration

Screen captures in this lab depict examples and results that may vary from what you see when you complete the exercises. In addition, some code examples may need to be customized to match your environment. For example, you may need to alter directory path information or user ID information.

Furthermore, some exercises presume you have access to a BigInsights administrative ID (e.g., biadmin) and/or a Big SQL administrative ID with SECADM authority (e.g., bigsql). If you don't have access to IDs with appropriate privileges, you may not be able to complete some exercises, or the results you see may differ from those depicted in this lab guide.

1.2. What you'll learn

After completing all exercises in this lab guide, you'll know how to

- Create a connection to your Big SQL 3.0 server
- Execute Big SQL statements and commands from a command line environment (JSqsh) and an Eclipse-based environment
- Create Big SQL tables
- Load data into Big SQL tables
- Query big data using Big SQL
- Develop and launch a JDBC client application for Big SQL
- Use a spreadsheet-style tool (BigSheets) to work with Big SQL data
- Explore enhanced SQL security features
- Work with non-traditional data formats using serializers / deserializers (SerDes)
- Explore the data access plan selected for your queries
- Collect statistical data useful for query optimization

Allow 6 - 7 hours to complete all sections of this lab.

1.3. Getting started with your VMware image

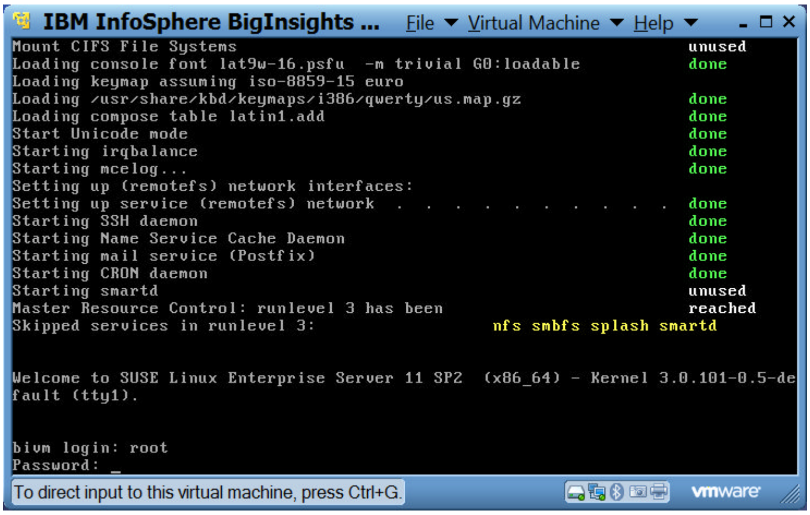
This section summarizes the steps you need to obtain a BigInsights 3.0 Quick Start Edition VMware image, install it, and launch it. If you or your instructor have already installed the VMware image and launched it, or if you will be using a different BigInsights environment for this lab, you may skip this section.

__1. If necessary, obtain a copy of the BigInsights 3.0 Quick Start Edition VMware image from your instructor or from IBM's external download site (http://www-01.ibm.com/software/data/infosphere/biginsights/quick-start/downloads.html). Use the image for the single-node cluster.

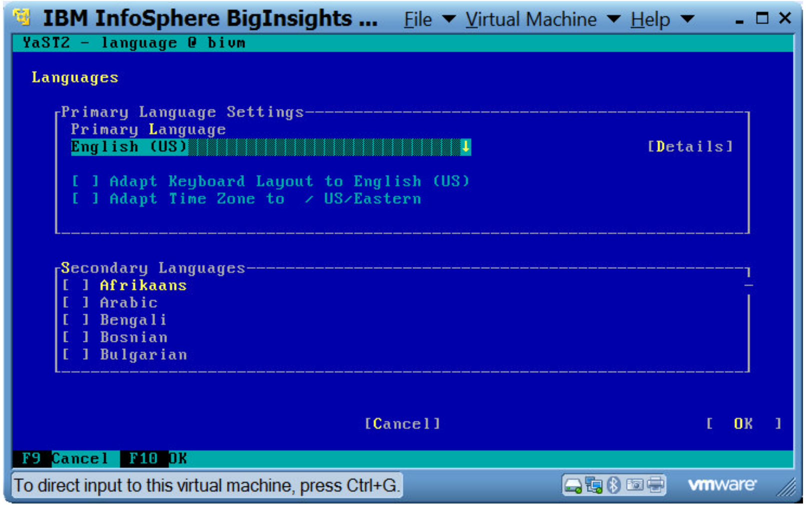
__2. Follow the instructions provided to decompress (unzip) the file and install the image on your laptop. Note that there is a README file with additional information.

__3. If necessary, install VMware player or other required software to run VMware images. Details are in the README file provided with the BigInsights VMware image.

__4. Launch the VMware image. When logging in for the first time, use the root ID (with a password of password). Follow the instructions to configure your environment, accept the licensing agreement, and enter the passwords for the root and biadmin IDs (root/password and biadmin/biadmin) when prompted. **This is a one-time only requirement.**

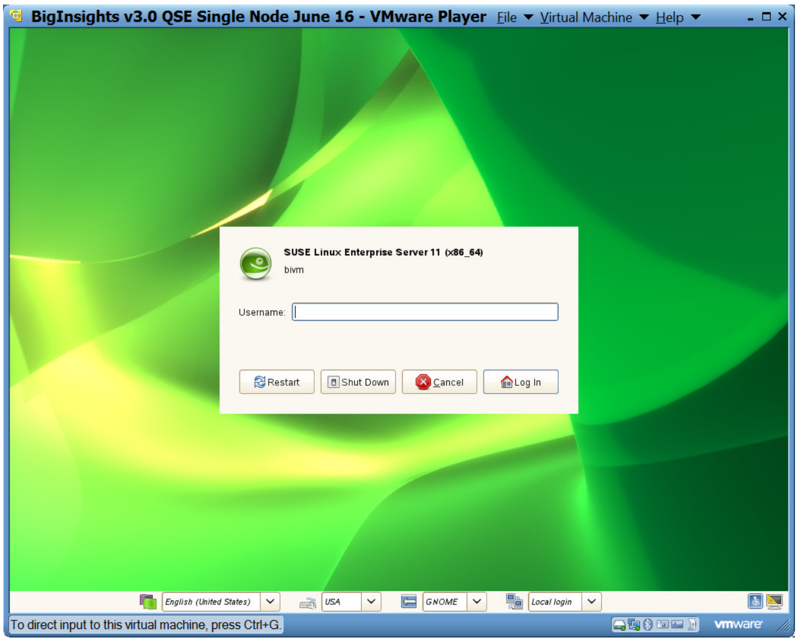

IBM InfoSphere BigInsights ...
File
Virtual Machine
Help
To direct input to this virtual machine, press Ctrl+G.
vmware

```
Mount CIFS File Systems                                                   unused
Loading console font lat9w-16.psfu  -m trivial G0:loadable                done
Loading keymap assuming iso-8859-15 euro
Loading /usr/share/kbd/keymaps/i386/qwerty/us.map.gz                      done
Loading compose table latin1.add                                          done
Start Unicode mode                                                        done
Starting irqbalance                                                       done
Starting mcelog...                                                        done
Setting up (remotefs) network interfaces:
Setting up service (remotefs) network  . . . . . . . . . .               done
Starting SSH daemon                                                       done
Starting Name Service Cache Daemon                                        done
Starting mail service (Postfix)                                           done
Starting CRON daemon                                                      done
Starting smartd                                                           unused
Master Resource Control: runlevel 3 has been                              reached
Skipped services in runlevel 3:                   nfs smbfs splash smartd


Welcome to SUSE Linux Enterprise Server 11 SP2  (x86_64) - Kernel 3.0.101-0.5-de
fault (tty1).


bivm login: root
Password: _
```


IBM InfoSphere BigInsights ...
File
Virtual Machine
Help
To direct input to this virtual machine, press Ctrl+G.
vmware

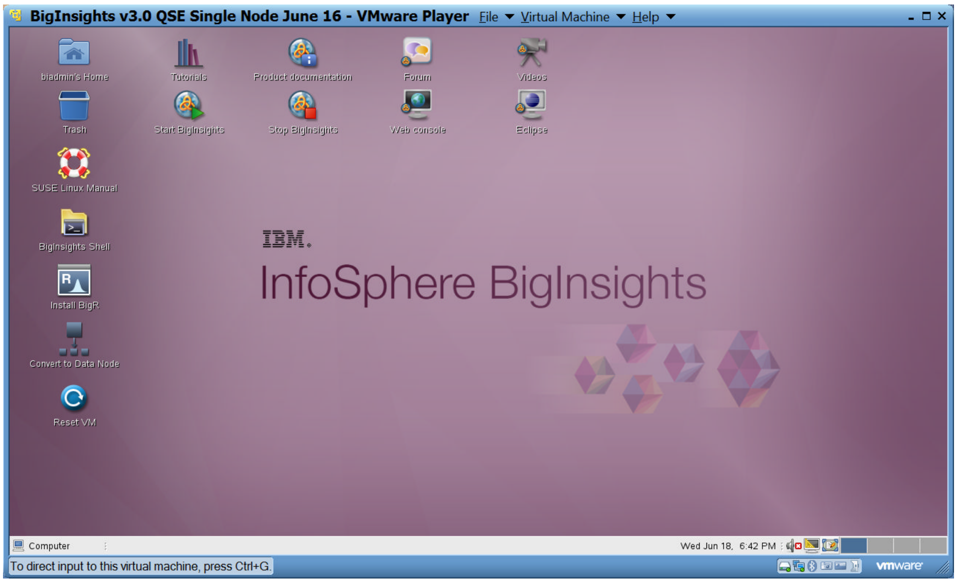
```
 YaST2 - language @ bivm

  Languages

     ┌Primary Language Settings──────────────────────────────────────────────┐
     │ Primary Language                                                      │
     │ English (US)                                 ↓             [Details]  │
     │                                                                       │
     │ [ ] Adapt Keyboard Layout to English (US)                             │
     │ [ ] Adapt Time Zone to  / US/Eastern                                  │
     │                                                                       │
     └───────────────────────────────────────────────────────────────────────┘

     ┌Secondary Languages─────────────────────────────────────────────────────┐
     │[ ] Afrikaans                                                           │
     │[ ] Arabic                                                              │
     │[ ] Bengali                                                             │
     │[ ] Bosnian                                                             │
     │[ ] Bulgarian                                                           │
     └───────────────────────────────────────────────────────────────────────┘

                                 [Cancel]                             [  OK  ]

 F9 Cancel  F10 OK
```

__5. When the one-time configuration process is completed, you will be presented with a SUSE Linux log in screen. Log in as biadmin/biadmin.



To direct input to this virtual machine, press Ctrl+G.

__6. Verify that your screen appears similar to this:



To direct input to this virtual machine, press Ctrl+G.

Your VMware image is now ready to use for this lab.

Ask your instructor for help if have any questions or need assistance getting your VMware image up and running. Or consult the README file and documentation provided for the VMWare image.

Detailed information about Big SQL and BigInsights is available online through the product documentation. Questions about Big SQL or BigInsights may be posted to the HadoopDev forum (https://developer.ibm.com/answers?community=hadoop).

This lab was developed by Cynthia M. Saracco (saracco@us.ibm.com), Carlos Renteria (crrenter@us.ibm.com), Raanon Reutlinger (reutlinger@il.ibm.com), and Uttam Jain (uttam@us.ibm.com).

Lab 2 Using the Big SQL command line interface (JSqsh)

BigInsights supports a command-line interface for Big SQL through the Java SQL Shell (JSqsh, pronounced "jay-skwish"). JSqsh is an open source project for querying JDBC databases. You may find it handy to become familiar with basic JSqsh capabilities, particularly if you don't expect to have access to an Eclipse environment at all times for your work.

In this section, you will learn how to

- Verify that the Biginsights services are running from the Web Console.
- Launch JSqsh.
- Issue Big SQL queries.
- Issue popular JSqsh commands to get help, retrieve your query history, and perform other functions.

If you prefer to only use an Eclipse-based environment to develop and execute your Big SQL queries, you can skip this section and continue to the next lab.

Allow 30 minutes to complete this section.

2.1. Launching the Web Console to verify BigInsights services are up and running

In this exercise, you will start all required BigInsights services and launch the Web console.

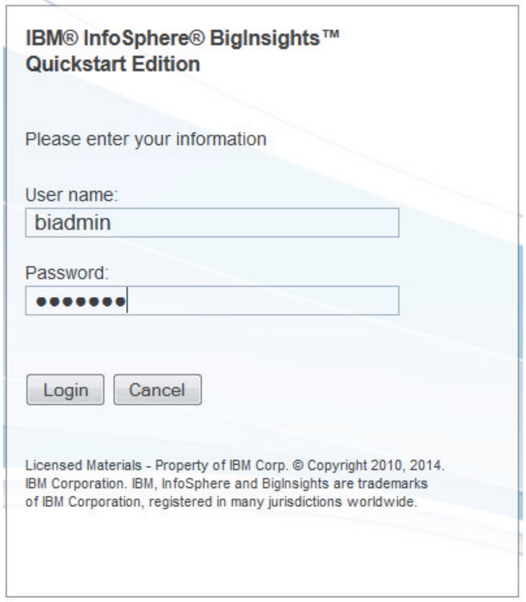
__1. Select the **Start BigInsights** icon to start all services. (Alternatively, you can open a terminal window and issue this command: `$BIGINSIGHTS_HOME/bin/start-all.sh`)

Wait until the operation completes. This may take several minutes, depending on your machine's resources.

__2. Verify that all required BigInsights services are up and running, including Big SQL.

__a. Launch the BigInsights Web console. (Direct your browser to http://bivm.ibm.com:8080 or select the Web Console icon on your desktop.)

__b. Log in with your user name and password.



IBM® InfoSphere® BigInsights™ Quickstart Edition

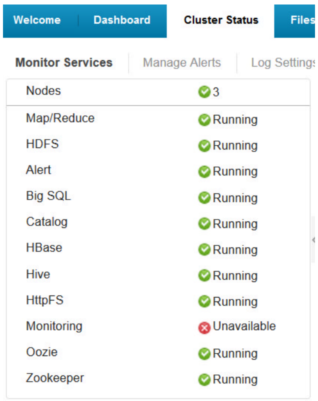
Please enter your information

User name: biadmin

Password: ●●●●●●●

Licensed Materials - Property of IBM Corp. © Copyright 2010, 2014. IBM Corporation. IBM, InfoSphere and BigInsights are trademarks of IBM Corporation, registered in many jurisdictions worldwide.

__c. Click on the **Cluster Status** tab to inspect the running services. Monitoring and Alert services do not need to be active for this lab.

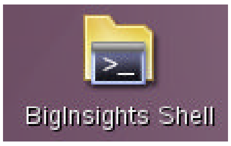


Nodes	✓ 3
Map/Reduce	✓ Running
HDFS	✓ Running
Alert	✓ Running
Big SQL	✓ Running
Catalog	✓ Running
HBase	✓ Running
Hive	✓ Running
HttpFS	✓ Running
Monitoring	✗ Unavailable
Oozie	✓ Running
Zookeeper	✓ Running

2.2. Understanding JSqsh connections

To issue Big SQL commands from JSqsh, you need to define a connection to a Big SQL server. The BigInsights VMware image has some predefined for you. Let's examine them.

__1. Open a terminal window. If desired, used the desktop icons. First, open the **BigInsights Shell** folder.

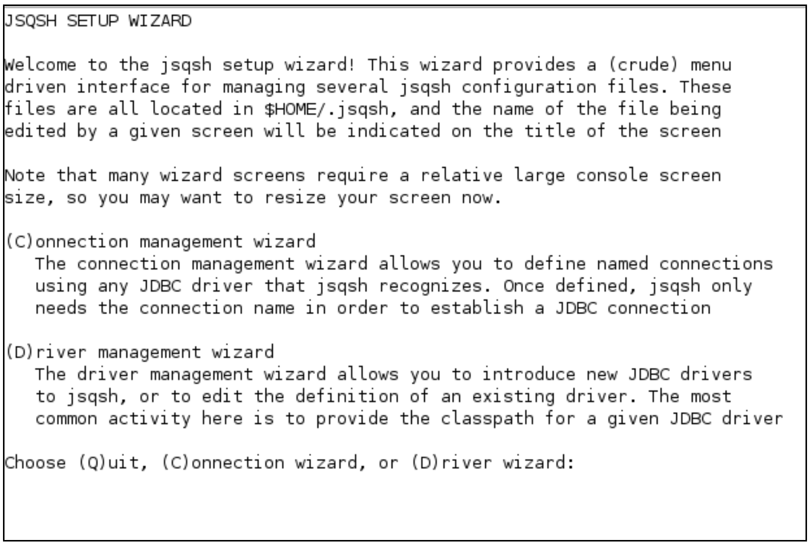


Then, click on **Terminal** icon.

__2. Launch the JSqsh shell.

```
$JSQSH_HOME/bin/jsqsh
```

__3. If this is the first time you launched JSqsh, a welcome screen will display. When prompted, enter c to launch the connection wizard.

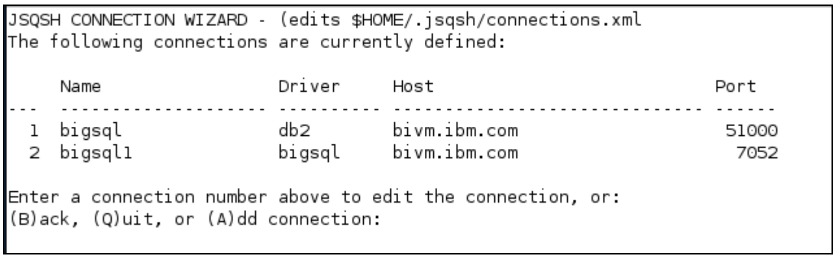


```
JSQSH SETUP WIZARD

Welcome to the jsqsh setup wizard! This wizard provides a (crude) menu
driven interface for managing several jsqsh configuration files. These
files are all located in $HOME/.jsqsh, and the name of the file being
edited by a given screen will be indicated on the title of the screen

Note that many wizard screens require a relative large console screen
size, so you may want to resize your screen now.

(C)onnection management wizard
   The connection management wizard allows you to define named connections
   using any JDBC driver that jsqsh recognizes. Once defined, jsqsh only
   needs the connection name in order to establish a JDBC connection

(D)river management wizard
   The driver management wizard allows you to introduce new JDBC drivers
   to jsqsh, or to edit the definition of an existing driver. The most
   common activity here is to provide the classpath for a given JDBC driver

Choose (Q)uit, (C)onnection wizard, or (D)river wizard:
```

In the future, you can enter `\setup connections` in the JSqsh shell to invoke this wizard.

__4. Inspect any existing connections in your environment.

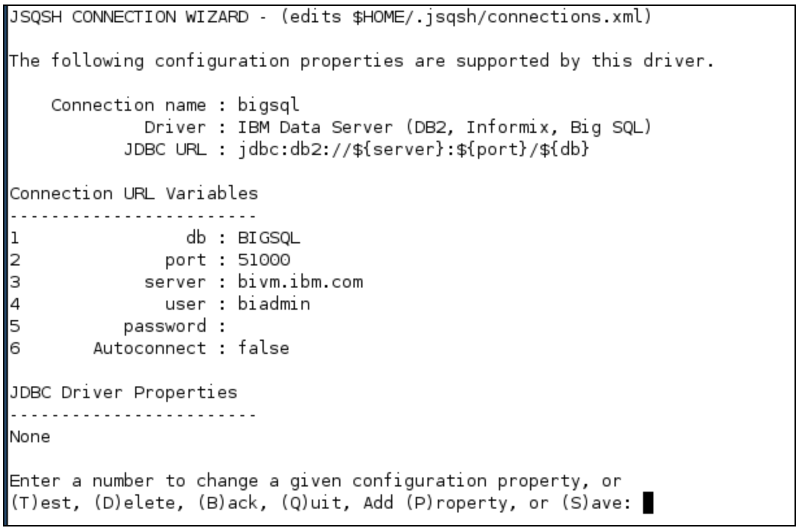
```
\setup connections
```



```
JSQSH CONNECTION WIZARD - (edits $HOME/.jsqsh/connections.xml
The following connections are currently defined:

    Name                 Driver     Host                           Port
--- -------------------- ---------- ------------------------------ ------
  1 bigsql               db2        bivm.ibm.com                    51000
  2 bigsql1              bigsql     bivm.ibm.com                     7052

Enter a connection number above to edit the connection, or:
(B)ack, (Q)uit, or (A)dd connection:
```

Throughout this lab, you will work with the `bigsql` database at port 51000, which is accessible through the DB2 JDBC driver provided with BigInsights.

__5. Examine the connection details for `bigsql`. Provide the connection number displayed by the wizard (in this case, 1) and hit **Enter**.



```
JSQSH CONNECTION WIZARD - (edits $HOME/.jsqsh/connections.xml)

The following configuration properties are supported by this driver.

    Connection name : bigsql
             Driver : IBM Data Server (DB2, Informix, Big SQL)
           JDBC URL : jdbc:db2://${server}:${port}/${db}

Connection URL Variables
------------------------
1                db : BIGSQL
2              port : 51000
3            server : bivm.ibm.com
4              user : biadmin
5          password :
6       Autoconnect : false

JDBC Driver Properties
------------------------
None

Enter a number to change a given configuration property, or
(T)est, (D)elete, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: █
```

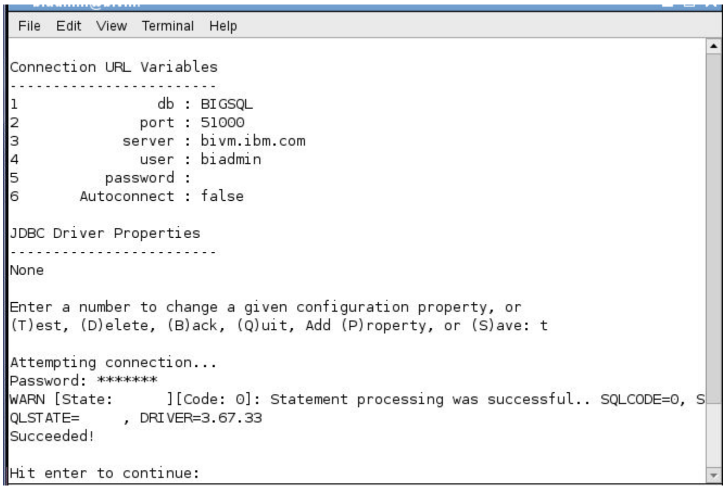
About the database name

Note that the Big SQL database name is a constant; it was defined during the installation of BigInsights.

__6. Enter `t` to test your configuration.

__7. When prompted for a password enter **biadmin**.

__8. Verify that the test succeeded, and hit **Enter**.



```
Connection URL Variables
-----------------------
1               db : BIGSQL
2             port : 51000
3           server : bivm.ibm.com
4             user : biadmin
5         password :
6      Autoconnect : false

JDBC Driver Properties
-----------------------
None

Enter a number to change a given configuration property, or
(T)est, (D)elete, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: t

Attempting connection...
Password: *******
WARN [State:      ][Code: 0]: Statement processing was successful.. SQLCODE=0, S
QLSTATE=     , DRIVER=3.67.33
Succeeded!

Hit enter to continue:
```

__9. Exit this current session. Enter q and then `quit`.

2.3. Optional: Creating your own database connection

The BigInsights Quick Start Edition VMware image is pre-configured with a connection to your Big SQL server with access for the biadmin account. However, you can also create your own JSqsh connections to Big SQL. This can be handy if you want to test capabilities available to different user accounts on your platform.

In this exercise, you'll learn how to create a Big SQL database connection in JSqsh. This section is optional – you can skip to the next module in this lab if you'd like.

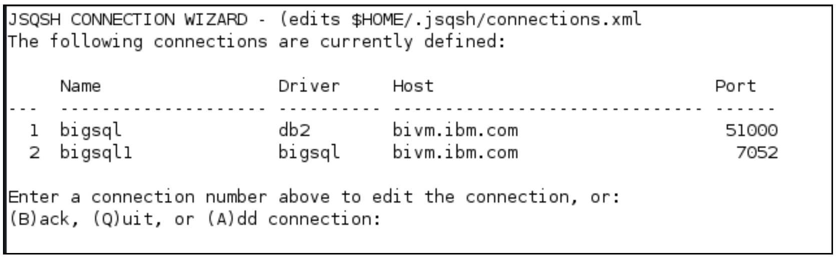
__1. If necessary, open a terminal window and launch the JSqsh shell.

```
$JSQSH_HOME/bin/jsqsh
```

__2. Invoke the setup wizard by entering the \setup connections command.

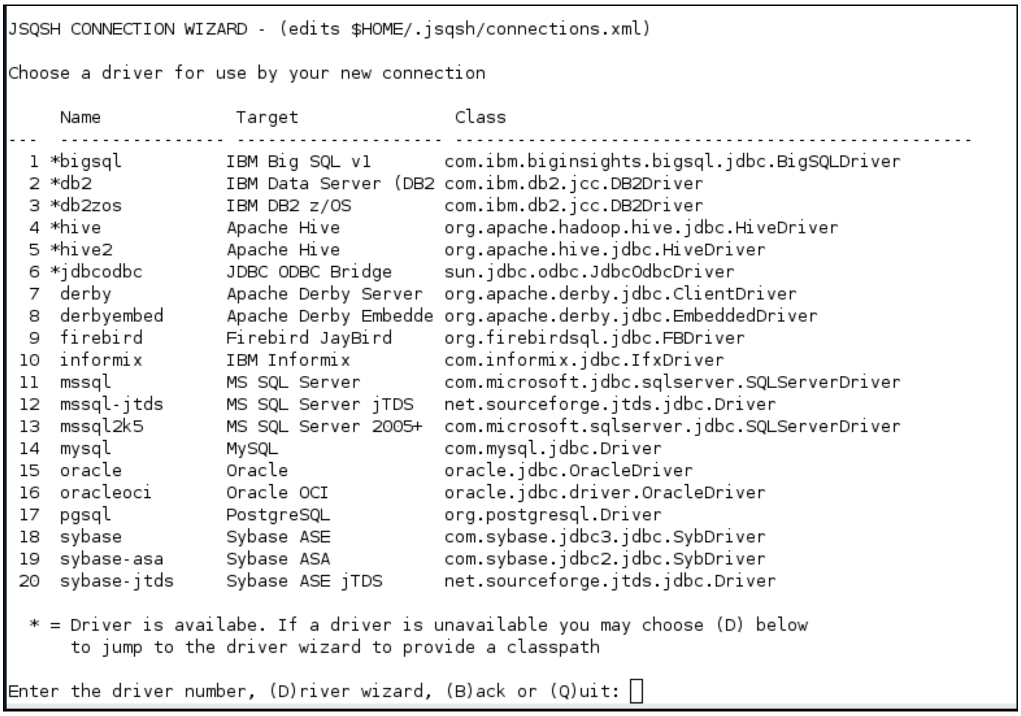
```
\setup connections
```

__3. When prompted, enter a to add a connection.



```
JSQSH CONNECTION WIZARD - (edits $HOME/.jsqsh/connections.xml
The following connections are currently defined:

    Name                 Driver     Host                           Port
--- -------------------- ---------- ------------------------------ ------
  1 bigsql               db2        bivm.ibm.com                    51000
  2 bigsql1              bigsql     bivm.ibm.com                     7052

Enter a connection number above to edit the connection, or:
(B)ack, (Q)uit, or (A)dd connection:
```

__4. Inspect the list of drivers displayed by the wizard, and note the number for the `db2` driver (not the `db2zos` driver). Depending on the size of your command window, you may need to scroll up to see the full list of drivers. In the screen capture below, the correct DB2 driver is 2. The order of your drivers may differ, as pre-installed drivers are listed first.



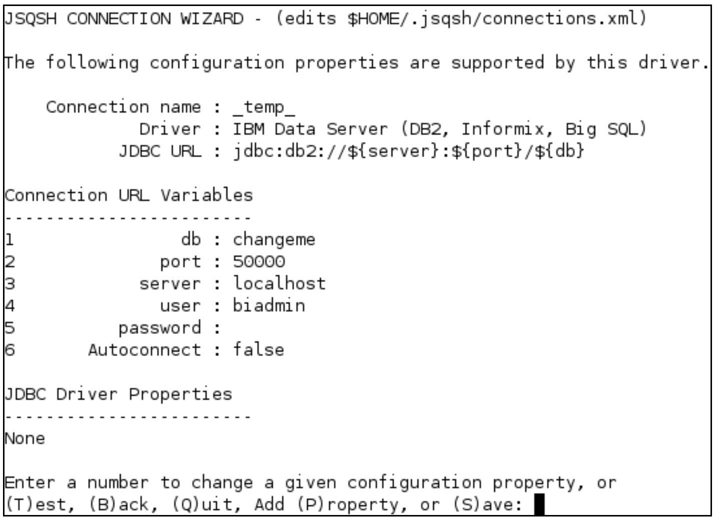
```
JSQSH CONNECTION WIZARD - (edits $HOME/.jsqsh/connections.xml)

Choose a driver for use by your new connection

     Name             Target              Class
--- --------------- ------------------- ------------------------------------------------
  1 *bigsql          IBM Big SQL v1      com.ibm.biginsights.bigsql.jdbc.BigSQLDriver
  2 *db2             IBM Data Server (DB2 com.ibm.db2.jcc.DB2Driver
  3 *db2zos          IBM DB2 z/OS        com.ibm.db2.jcc.DB2Driver
  4 *hive            Apache Hive         org.apache.hadoop.hive.jdbc.HiveDriver
  5 *hive2           Apache Hive         org.apache.hive.jdbc.HiveDriver
  6 *jdbcodbc        JDBC ODBC Bridge    sun.jdbc.odbc.JdbcOdbcDriver
  7  derby           Apache Derby Server  org.apache.derby.jdbc.ClientDriver
  8  derbyembed      Apache Derby Embedde org.apache.derby.jdbc.EmbeddedDriver
  9  firebird        Firebird JayBird    org.firebirdsql.jdbc.FBDriver
 10  informix        IBM Informix        com.informix.jdbc.IfxDriver
 11  mssql           MS SQL Server       com.microsoft.jdbc.sqlserver.SQLServerDriver
 12  mssql-jtds      MS SQL Server jTDS  net.sourceforge.jtds.jdbc.Driver
 13  mssql2k5        MS SQL Server 2005+ com.microsoft.sqlserver.jdbc.SQLServerDriver
 14  mysql           MySQL               com.mysql.jdbc.Driver
 15  oracle          Oracle              oracle.jdbc.OracleDriver
 16  oracleoci       Oracle OCI          oracle.jdbc.driver.OracleDriver
 17  pgsql           PostgreSQL          org.postgresql.Driver
 18  sybase          Sybase ASE          com.sybase.jdbc3.jdbc.SybDriver
 19  sybase-asa      Sybase ASA          com.sybase.jdbc2.jdbc.SybDriver
 20  sybase-jtds     Sybase ASE jTDS     net.sourceforge.jtds.jdbc.Driver

  * = Driver is availabe. If a driver is unavailable you may choose (D) below
      to jump to the driver wizard to provide a classpath

Enter the driver number, (D)river wizard, (B)ack or (Q)uit: 
```

About the driver selection

You may be wondering why this lab uses the DB2 driver rather than the Big SQL driver. In 2014, IBM released a common SQL query engine as part of its DB2 and BigInsights offerings. Doing so provides for greater SQL commonality across its relational DBMS and Hadoop-based offerings. It also brings a greater breadth of SQL function to Hadoop (BigInsights) users. This common query engine is accessible through the "DB2" driver listed. The Big SQL driver remains operational and offers connectivity to an earlier, BigInsights-specific SQL query engine. This lab focuses on using the common SQL query engine.

__5. At the prompt line, enter the number of the DB2 driver.

__6. The connection wizard displays some default values for the connection properties and prompts you to change them. (Your default values may differ from those shown below.)

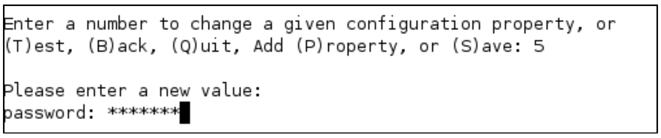


```
JSQSH CONNECTION WIZARD - (edits $HOME/.jsqsh/connections.xml)

The following configuration properties are supported by this driver.

    Connection name : _temp_
             Driver : IBM Data Server (DB2, Informix, Big SQL)
           JDBC URL : jdbc:db2://${server}:${port}/${db}

Connection URL Variables
-----------------------
1                db : changeme
2              port : 50000
3            server : localhost
4              user : biadmin
5          password :
6       Autoconnect : false

JDBC Driver Properties
-----------------------
None

Enter a number to change a given configuration property, or
(T)est, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: 
```

__7. Change each variable as needed, one at a time. To do so, enter the variable number and specify a new value when prompted. For example, to change the value of the `password` variable (which is null by default)

(1) Specify variable number 5 and hit **Enter**.

(2) Enter the password value and hit **Enter.**

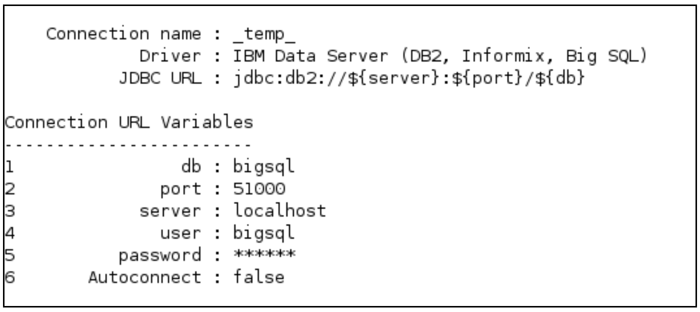


```
Enter a number to change a given configuration property, or
(T)est, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: 5

Please enter a new value:
password: *******
```

(3) Inspect the variable settings that are displayed again to verify your change.

Repeat this process as needed for each variable that needs to be changed. In particular, you may need to change the values for the `db` (database), `port`, and `server` variables.

After making all necessary changes, the variables should reflect values that are accurate for your environment. Here is an example of a connection created for the bigsql user account (password bigsql) that will connect to the database named “bigsql” at port 51000 on the localhost server.

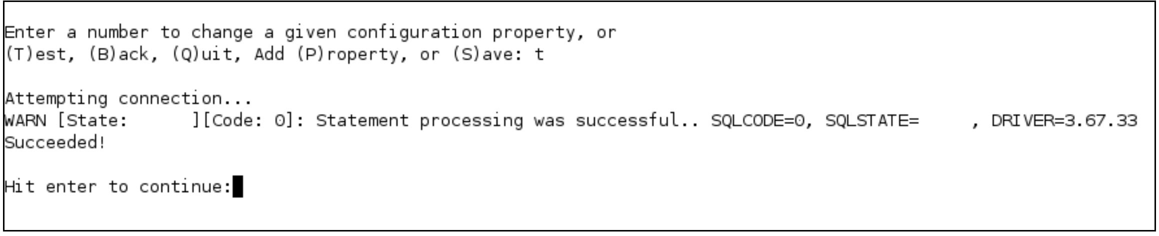


```
    Connection name : _temp_
             Driver : IBM Data Server (DB2, Informix, Big SQL)
           JDBC URL : jdbc:db2://${server}:${port}/${db}

Connection URL Variables
-----------------------
1                db : bigsql
2              port : 51000
3            server : localhost
4              user : bigsql
5          password : ******
6       Autoconnect : false
```

The Big SQL database is defined during the installation of BigInsights. The default is bigsql. In addition, a Big SQL database administrator account is also defined at installation. This account has SECADM (security administration) authority for Big SQL. By default, that user account is bigsql.

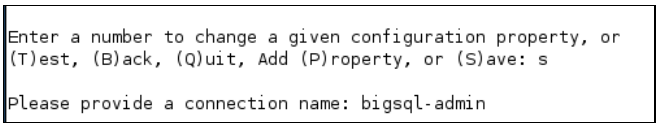
__8. Enter `t` to test your configuration.

__9. Verify that the test succeeded, and hit **Enter**.



```
Enter a number to change a given configuration property, or
(T)est, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: t

Attempting connection...
WARN [State:      ][Code: 0]: Statement processing was successful.. SQLCODE=0, SQLSTATE=     , DRIVER=3.67.33
Succeeded!

Hit enter to continue:
```

__10. Save your connection. Enter s, provide a name for your connection (such as bigsql-admin), and hit **Enter**.



```
Enter a number to change a given configuration property, or
(T)est, (B)ack, (Q)uit, Add (P)roperty, or (S)ave: s

Please provide a connection name: bigsql-admin
```

__11. Finally, quit the connection wizard when prompted. (Enter q.)

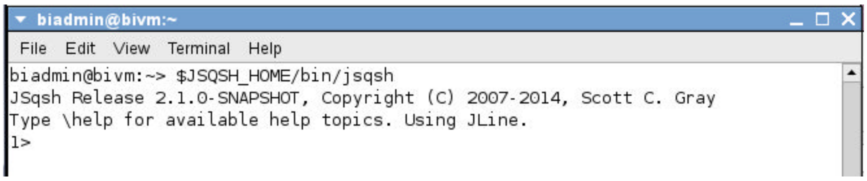
2.4. Getting Help for JSqsh

Now that you're familiar with JSqsh connections, you're ready to work further with the shell.

__1. Launch the JSqsh shell from a terminal without any parameters

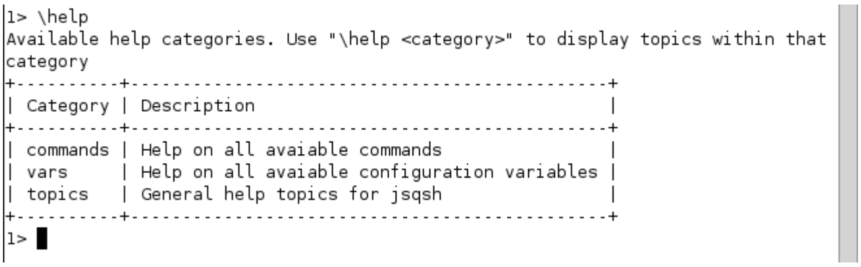
```
$JSQSH_HOME/bin/jsqsh
```

__2. Verify that the JSqsh command prompt of `1>` appears.



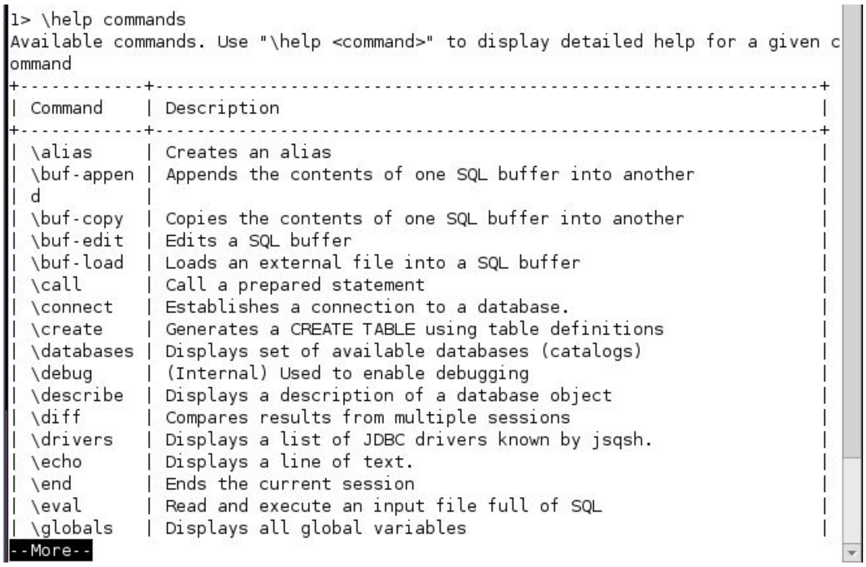
```
biadmin@bivm:~> $JSQSH_HOME/bin/jsqsh
JSqsh Release 2.1.0-SNAPSHOT, Copyright (C) 2007-2014, Scott C. Gray
Type \help for available help topics. Using JLine.
1>
```

__3. 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> 
```

__4. Optionally, 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.

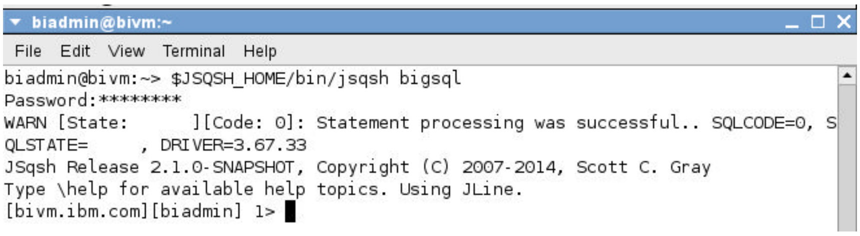
__5. Enter `quit` to exit the JSqsh shell.

2.5. Issuing JSqsh commands and Big SQL queries

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

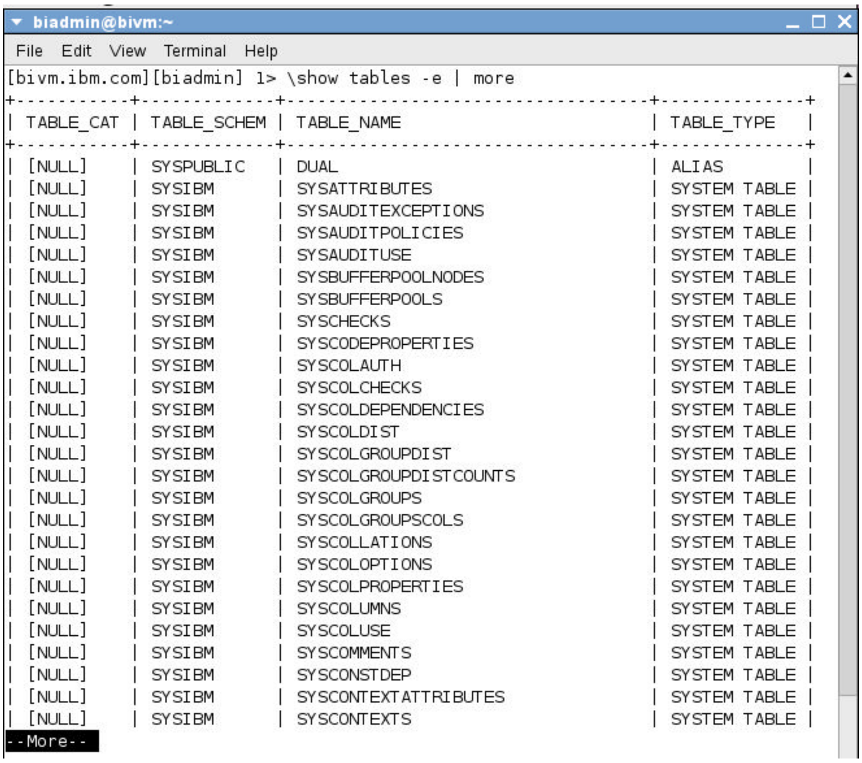
__1. Launch the JSqsh shell and connect to your Big SQL server by specifying the connection name on the command line. When prompted, enter your password.

```
$JSQSH_HOME/bin/jsqsh bigsql
```



```
biadmin@bivm:~> $JSQSH_HOME/bin/jsqsh bigsql
Password:********
WARN [State:      ][Code: 0]: Statement processing was successful.. SQLCODE=0, S
QLSTATE=      , DRIVER=3.67.33
JSqsh Release 2.1.0-SNAPSHOT, Copyright (C) 2007-2014, Scott C. Gray
Type \help for available help topics. Using JLine.
[bivm.ibm.com][biadmin] 1>
```

__2. Type `\show tables -e | more` to display essential information about all available tables one page at a time. The structure of your output will be similar to the results shown below, although the specific information will vary depending on the tables available in your environment. Press space bar to continue `scrolling` or q to stop `scrolling.`



```
[bivm.ibm.com][biadmin] 1> \show tables -e | more
+-----------+------------+---------------------------------+--------------+
| TABLE_CAT | TABLE_SCHEM | TABLE_NAME                     | TABLE_TYPE   |
+-----------+------------+---------------------------------+--------------+
| [NULL]    | SYSPUBLIC  | DUAL                            | ALIAS        |
| [NULL]    | SYSIBM     | SYSATTRIBUTES                   | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSAUDITEXCEPTIONS              | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSAUDITPOLICIES                | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSAUDITUSE                     | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSBUFFERPOOLNODES              | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSBUFFERPOOLS                  | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCHECKS                       | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCODEPROPERTIES               | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLAUTH                      | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLCHECKS                    | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLDEPENDENCIES              | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLDIST                      | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLGROUPDIST                 | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLGROUPDISTCOUNTS           | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLGROUPS                    | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLGROUPSCOLS                | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLLATIONS                   | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLOPTIONS                   | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLPROPERTIES                | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLUMNS                      | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOLUSE                       | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCOMMENTS                     | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCONSTDEP                     | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCONTEXTATTRIBUTES            | SYSTEM TABLE |
| [NULL]    | SYSIBM     | SYSCONTEXTS                     | SYSTEM TABLE |
--More--
```

__3. Next, cut and paste 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 your user name. This is equivalent to

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

where `yourID` is your user name.

We've intentionally created a very simple Hadoop table for this exercise so that you can concentrate on working with JSqsh. In later modules, you'll learn more about CREATE TABLE options supported by Big SQL. For example, you'll learn about the LOCATION clause of CREATE TABLE. In these examples, where LOCATION is omitted, the default Hadoop directory path for these tables are at `/biginsights/hive/warehouse/<schema>.db/<table>`.

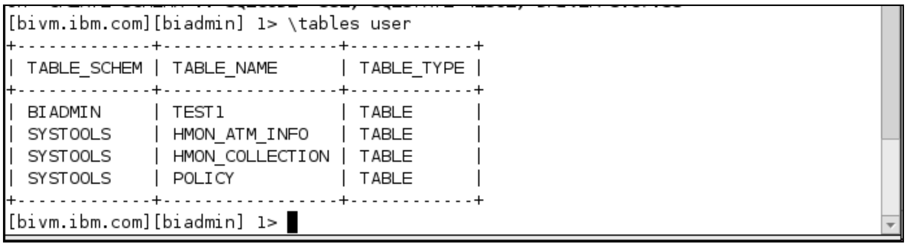
Big SQL 3.0 enables users with appropriate authority to create their own schemas by issuing a command such as

```
create schema if not exists testschema;
```

Authorized users can then create tables in that schema as desired. Furthermore, users can also create a table in a different schema, and if it doesn't already exist it will be implicitly created.

__4. Display all user tables (avoiding views and system tables) with the `\tables user` command. Note that with this command you may also see tables defined by other users, but you won't have the privileges to query them.

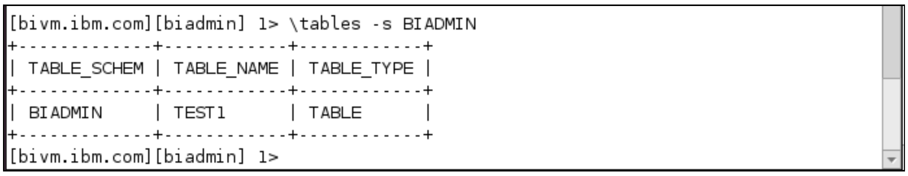
```
\tables user
```



```
[bivm.ibm.com][biadmin] 1> \tables user
+-------------+----------------+------------+
| TABLE_SCHEM | TABLE_NAME     | TABLE_TYPE |
+-------------+----------------+------------+
| BIADMIN     | TEST1          | TABLE      |
| SYSTOOLS    | HMON_ATM_INFO  | TABLE      |
| SYSTOOLS    | HMON_COLLECTION | TABLE      |
| SYSTOOLS    | POLICY         | TABLE      |
+-------------+----------------+------------+
[bivm.ibm.com][biadmin] 1>
```

__5. If your output contains too many user tables from other users, you can narrow your results by specifying a schema with the command `\tables -s BIADMIN`. The schema name should be provided in upper case since it will be used directly to filter the list of tables.

```
\tables -s BIADMIN
```



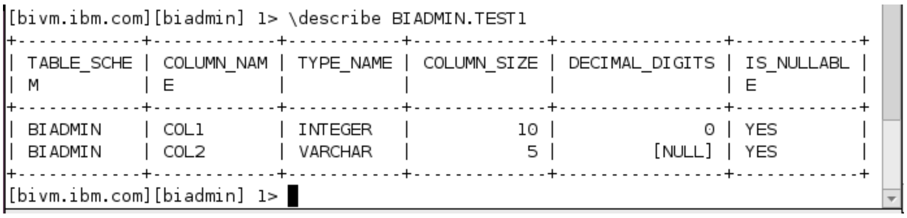
```
[bivm.ibm.com][biadmin] 1> \tables -s BIADMIN
+-------------+------------+------------+
| TABLE_SCHEM | TABLE_NAME | TABLE_TYPE |
+-------------+------------+------------+
| BIADMIN     | TEST1      | TABLE      |
+-------------+------------+------------+
[bivm.ibm.com][biadmin] 1>
```

__6. Try inserting 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 meta data about a table, use the \describe command with the fully qualified table name in upper case.

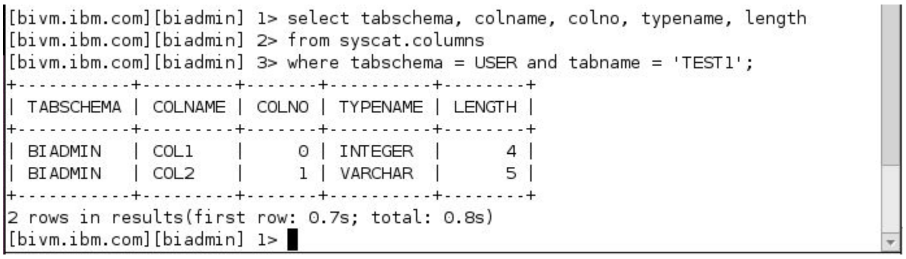


```
[bivm.ibm.com][biadmin] 1> \describe BIADMIN.TEST1
+-----------+-----------+----------+------------+----------------+-----------+
| TABLE_SCHE | COLUMN_NAM | TYPE_NAME | COLUMN_SIZE | DECIMAL_DIGITS | IS_NULLABL |
| M          | E          |           |             |                | E          |
+-----------+-----------+----------+------------+----------------+-----------+
| BIADMIN    | COL1       | INTEGER   |          10 |              0 | YES        |
| BIADMIN    | COL2       | VARCHAR   |           5 |         [NULL] | YES        |
+-----------+-----------+----------+------------+----------------+-----------+
[bivm.ibm.com][biadmin] 1>
```

__8. Optionally, you can query the system for metadata about this table:

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

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.



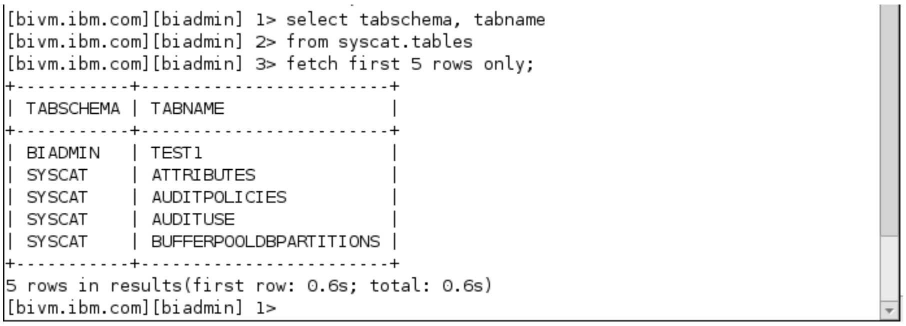
```
[bivm.ibm.com][biadmin] 1> select tabschema, colname, colno, typename, length
[bivm.ibm.com][biadmin] 2> from syscat.columns
[bivm.ibm.com][biadmin] 3> where tabschema = USER and tabname = 'TEST1';
+-----------+---------+-------+----------+--------+
| TABSCHEMA | COLNAME | COLNO | TYPENAME | LENGTH |
+-----------+---------+-------+----------+--------+
| BIADMIN   | COL1    |     0 | INTEGER  |      4 |
| BIADMIN   | COL2    |     1 | VARCHAR  |      5 |
+-----------+---------+-------+----------+--------+
2 rows in results(first row: 0.7s; total: 0.8s)
[bivm.ibm.com][biadmin] 1>
```

In case you're wondering, `syscat.columns` is one of a number of views supplied over system catalog data automatically maintained for you by the Big SQL service.

Once again, notice that we 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.

__9. Issue a query that restricts the number of rows returned to 5. For example, select the first 5 rows from `syscat.tables`:

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

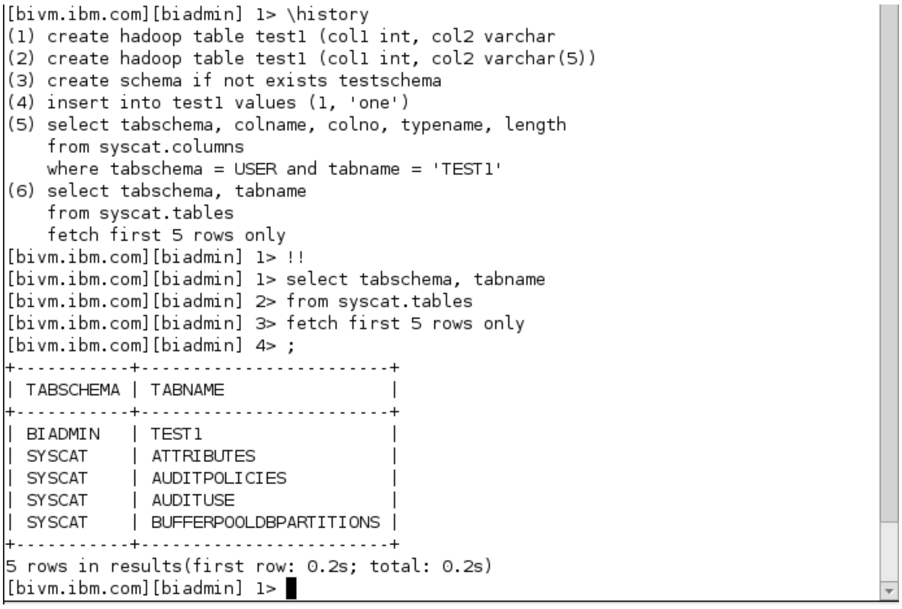


```
[bivm.ibm.com][biadmin] 1> select tabschema, tabname
[bivm.ibm.com][biadmin] 2> from syscat.tables
[bivm.ibm.com][biadmin] 3> fetch first 5 rows only;
+-----------+-----------------------+
| TABSCHEMA | TABNAME               |
+-----------+-----------------------+
| BIADMIN   | TEST1                 |
| SYSCAT    | ATTRIBUTES            |
| SYSCAT    | AUDITPOLICIES         |
| SYSCAT    | AUDITUSE              |
| SYSCAT    | BUFFERPOOLDBPARTITIONS |
+-----------+-----------------------+
5 rows in results(first row: 0.6s; total: 0.6s)
[bivm.ibm.com][biadmin] 1>
```

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

__10. Review the history of commands you recently executed in the JSqsh shell. Type `\history` and **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. Enter `!!` (two exclamation points, without spaces) to recall the previously run statement. In the example below, the previous statement selects the first 5 rows from `syscat.tables.` To run the statement, type a semi-colon on the following line.



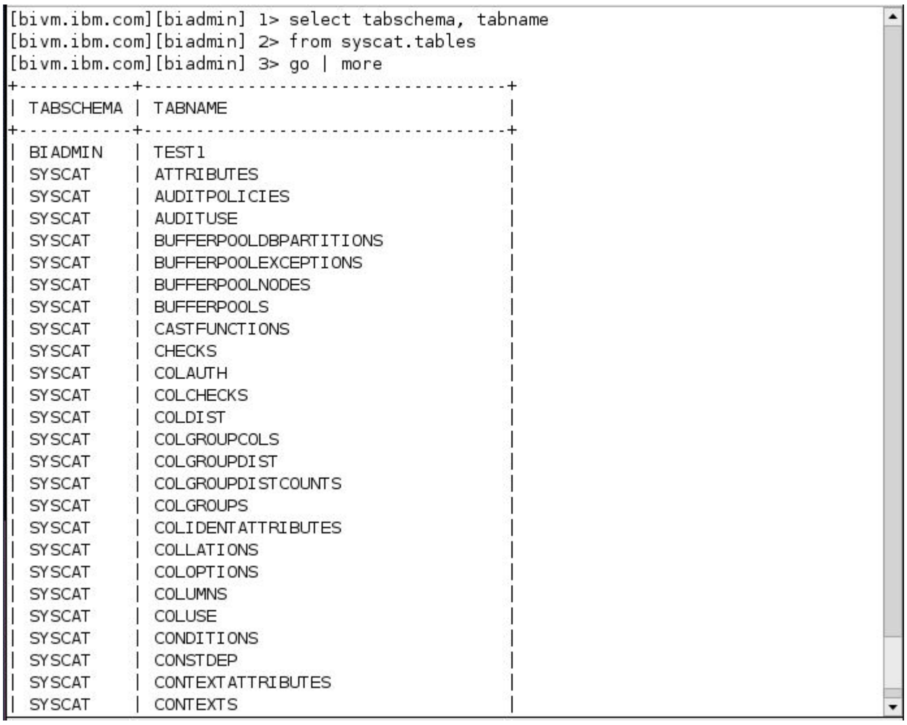
```
[bivm.ibm.com][biadmin] 1> \history
(1) create hadoop table test1 (col1 int, col2 varchar
(2) create hadoop table test1 (col1 int, col2 varchar(5))
(3) create schema if not exists testschema
(4) insert into test1 values (1, 'one')
(5) select tabschema, colname, colno, typename, length
    from syscat.columns
    where tabschema = USER and tabname = 'TEST1'
(6) select tabschema, tabname
    from syscat.tables
    fetch first 5 rows only
[bivm.ibm.com][biadmin] 1> !!
[bivm.ibm.com][biadmin] 1> select tabschema, tabname
[bivm.ibm.com][biadmin] 2> from syscat.tables
[bivm.ibm.com][biadmin] 3> fetch first 5 rows only
[bivm.ibm.com][biadmin] 4> ;
+-----------+-----------------------+
| TABSCHEMA | TABNAME               |
+-----------+-----------------------+
| BIADMIN   | TEST1                 |
| SYSCAT    | ATTRIBUTES            |
| SYSCAT    | AUDITPOLICIES         |
| SYSCAT    | AUDITUSE              |
| SYSCAT    | BUFFERPOOLDBPARTITIONS |
+-----------+-----------------------+
5 rows in results(first row: 0.2s; total: 0.2s)
[bivm.ibm.com][biadmin] 1>
```

__12. 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.

__13. Experiment with JSqsh's ability to support piping of output to an external program. 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:



```
[bivm.ibm.com][biadmin] 1> select tabschema, tabname
[bivm.ibm.com][biadmin] 2> from syscat.tables
[bivm.ibm.com][biadmin] 3> go | more
+-----------+--------------------------------+
| TABSCHEMA | TABNAME                        |
+-----------+--------------------------------+
| BIADMIN   | 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                         |
| SYSCAT    | CONDITIONS                     |
| SYSCAT    | CONSTDEP                       |
| SYSCAT    | CONTEXTATTRIBUTES              |
| SYSCAT    | CONTEXTS                       |
```

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.

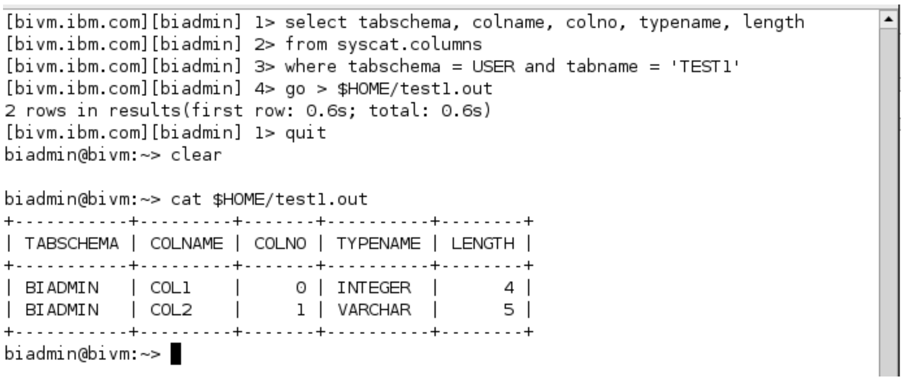
__14. Experiment with JSqsh's ability to redirect output to a local file rather than the console display. Enter the following two lines on the command shell, adjusting the path information on the second line as needed for your environment:

```
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.

__15. Exit the shell and view the output file:

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



```
[bivm.ibm.com][biadmin] 1> select tabschema, colname, colno, typename, length
[bivm.ibm.com][biadmin] 2> from syscat.columns
[bivm.ibm.com][biadmin] 3> where tabschema = USER and tabname = 'TEST1'
[bivm.ibm.com][biadmin] 4> go > $HOME/test1.out
2 rows in results(first row: 0.6s; total: 0.6s)
[bivm.ibm.com][biadmin] 1> quit
biadmin@bivm:~> clear

biadmin@bivm:~> cat $HOME/test1.out
+-----------+---------+-------+----------+--------+
| TABSCHEMA | COLNAME | COLNO | TYPENAME | LENGTH |
+-----------+---------+-------+----------+--------+
| BIADMIN   | COL1    |     0 | INTEGER  |      4 |
| BIADMIN   | COL2    |     1 | VARCHAR  |      5 |
+-----------+---------+-------+----------+--------+
biadmin@bivm:~>
```

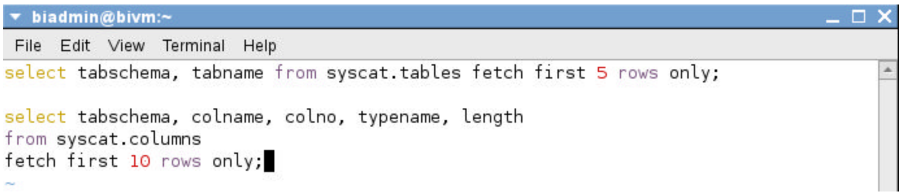
__16. Invoke JSqsh using an input file containing Big SQL commands to be executed. Maintaining SQL script files can be quite handy for repeatedly executing various queries.

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

```
vi test.sql
```

__b. 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;
```



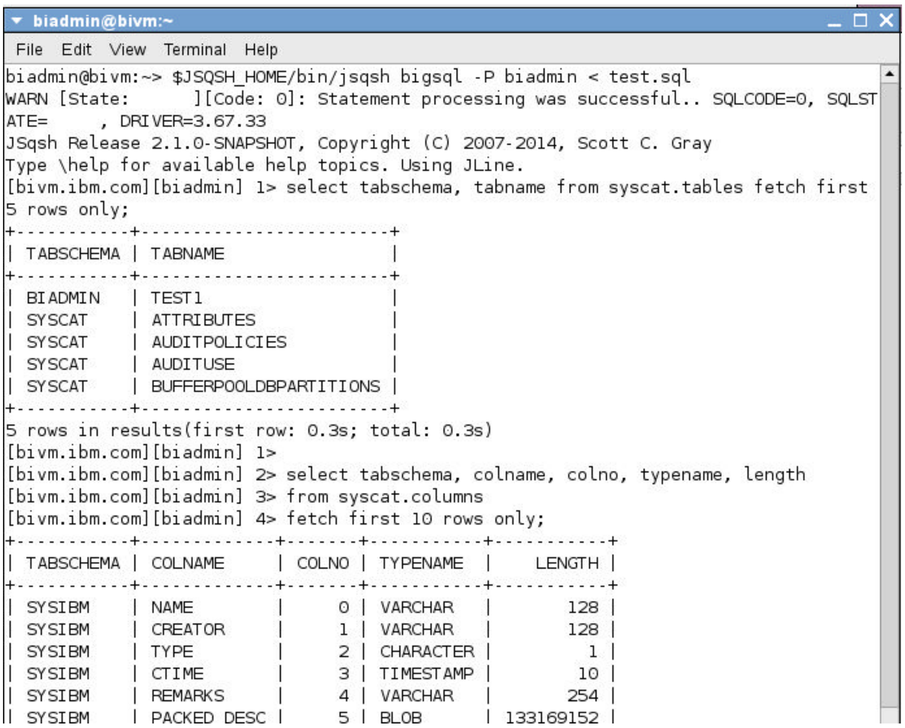
```
select tabschema, tabname from syscat.tables fetch first 5 rows only;

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

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

__d. Invoke JSQSH, instructing it to connect to your bigsql database and execute the contents of the script you just created:

```
$JSQSH_HOME/bin/jsqsh bigsql –P biadmin < test.sql
```

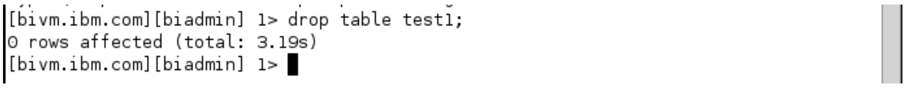
__e. Inspect the output. As you will see, JSQSH executes each instruction and displays its output. (Partial results are shown below.)



```
biadmin@bivm:~> $JSQSH_HOME/bin/jsqsh bigsql -P biadmin < test.sql
WARN [State:      ][Code: 0]: Statement processing was successful.. SQLCODE=0, SQLST
ATE=      , DRIVER=3.67.33
JSqsh Release 2.1.0-SNAPSHOT, Copyright (C) 2007-2014, Scott C. Gray
Type \help for available help topics. Using JLine.
[bivm.ibm.com][biadmin] 1> select tabschema, tabname from syscat.tables fetch first
5 rows only;
+-----------+-----------------------+
| TABSCHEMA | TABNAME               |
+-----------+-----------------------+
| BIADMIN   | TEST1                 |
| SYSCAT    | ATTRIBUTES            |
| SYSCAT    | AUDITPOLICIES         |
| SYSCAT    | AUDITUSE              |
| SYSCAT    | BUFFERPOOLDBPARTITIONS |
+-----------+-----------------------+
5 rows in results(first row: 0.3s; total: 0.3s)
[bivm.ibm.com][biadmin] 1>
[bivm.ibm.com][biadmin] 2> select tabschema, colname, colno, typename, length
[bivm.ibm.com][biadmin] 3> from syscat.columns
[bivm.ibm.com][biadmin] 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 |
```

__17. Finally, clean up the your database:

```
$JSQSH_HOME/bin/jsqsh bigsql

drop table test1;
```



```
[bivm.ibm.com][biadmin] 1> drop table test1;
0 rows affected (total: 3.19s)
[bivm.ibm.com][biadmin] 1>
```

Consult the JSqsh documentation (http://sourceforge.net/apps/mediawiki/jsqsh/index.php?title=Main_Page) for more information about using this command line tool.

Lab 3 Using Eclipse

You can develop and execute Big SQL queries using an appropriate version of Eclipse (4.2.2) and some additional software. Some people prefer to use Eclipse over JSqsh, as query result sets are formatted in an easy-to-read fashion and queries are typically organized in scripts within projects.

In this section, you will learn how to:

- Configure Eclipse to work with Big SQL.
- Create a connection to your Big SQL 3.0 server.
- Create projects and Big SQL scripts.
- Issue Big SQL queries.

If you prefer to only use JSqsh to develop and execute your Big SQL queries, you can skip this section and continue to the next lab.

Allow 30 - 45 minutes to complete the configuration and query exercises in this lab. Allow additional time to install an appropriate Eclipse shell if you don't have one already available. You must have access to a running BigInsights 3.0 cluster before beginning this lab.

3.1. Launching the Web Console to verify BigInsights services are up and running

In this section, you'll launch all required BigInsights services and use the Web console to verify that all required BigInsights services are up and running. You can skip this section if you have already completed this work as part of an earlier lab or if your instructor tells you that this work has been done for you.

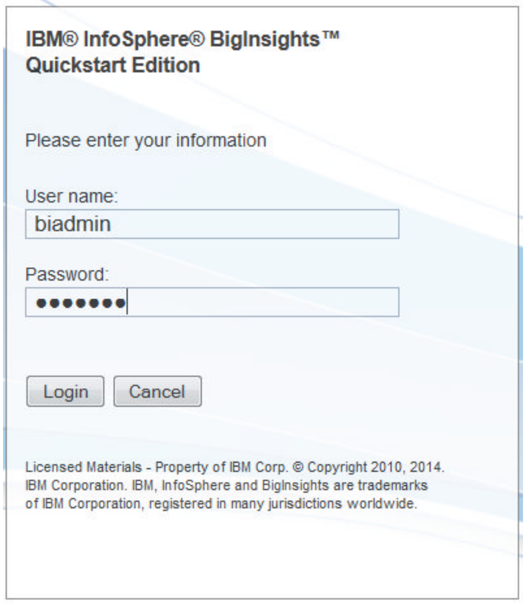
__1. Select the **Start BigInsights** icon to start all services. (Alternatively, you can open a terminal window and issue this command: `$BIGINSIGHTS_HOME/bin/start-all.sh`)

Wait until the operation completes. This make take several minutes, depending on your machine resources.

__2. Verify that all required BigInsights services are up and running, including Big SQL.

__a. Launch the BigInsights Web console. (Direct your browser to http://bivm.ibm.com:8080 or select the Web Console icon on your desktop.)

__b. Log in with your user name and password.



IBM® InfoSphere® BigInsights™
Quickstart Edition

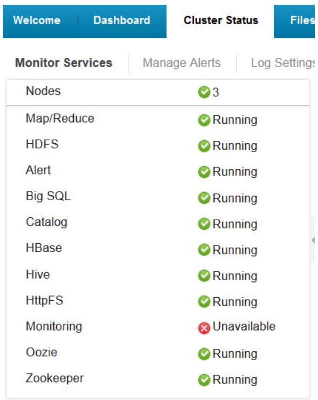
Please enter your information

User name: biadmin

Password: •••••••

Licensed Materials - Property of IBM Corp. © Copyright 2010, 2014. IBM Corporation. IBM, InfoSphere and BigInsights are trademarks of IBM Corporation, registered in many jurisdictions worldwide.

__c. Click on the **Cluster Status** tab to inspect the running services. Monitoring and Alert services do not need to be active for this lab.



Nodes	✓ 3
Map/Reduce	✓ Running
HDFS	✓ Running
Alert	✓ Running
Big SQL	✓ Running
Catalog	✓ Running
HBase	✓ Running
Hive	✓ Running
HttpFS	✓ Running
Monitoring	✗ Unavailable
Oozie	✓ Running
Zookeeper	✓ Running

3.2. Creating a Big SQL Connection in Eclipse

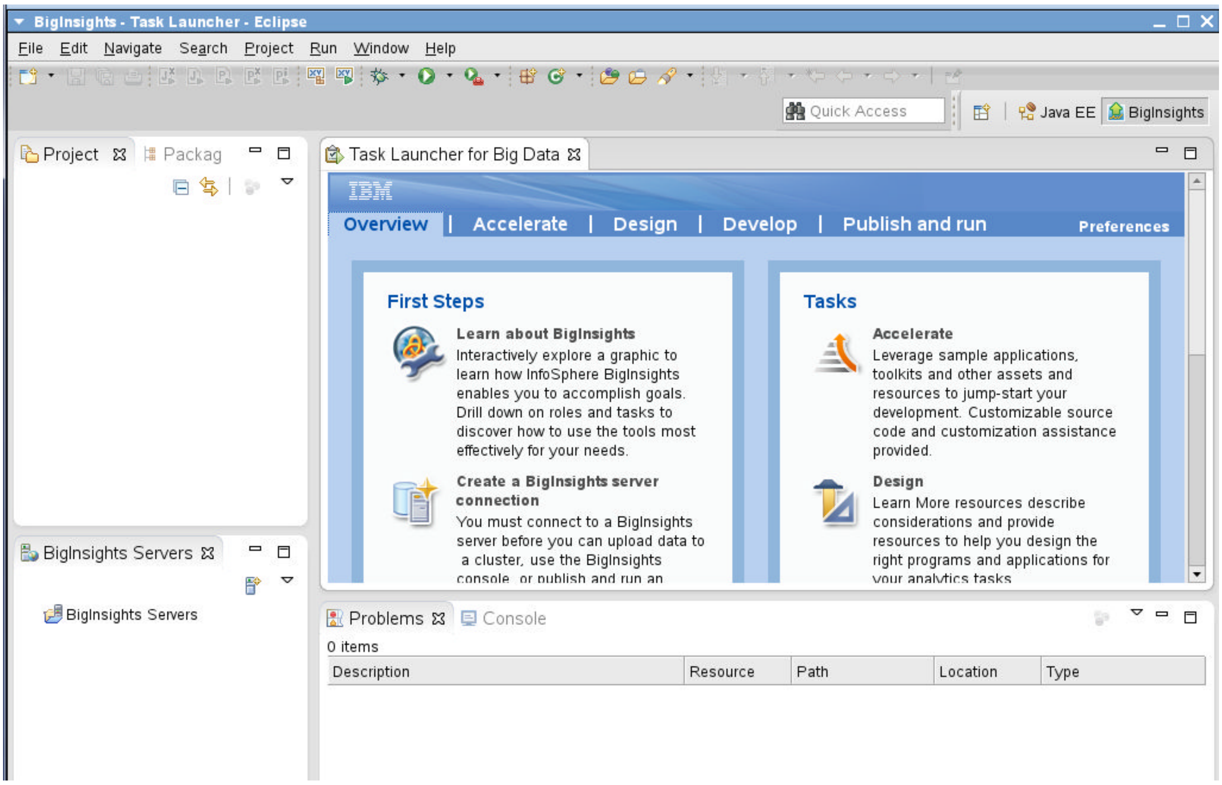
Certain tasks require a live connection to a Big SQL server within the BigInsights cluster. This section explains how you can define a JDBC connection to your Big SQL server.

If you're working with the Quick Start Edition VMware image, this section is optional. (The image is pre-configured with a Big SQL connection.)

__1. Launch Eclipse using this icon on your Desktop.

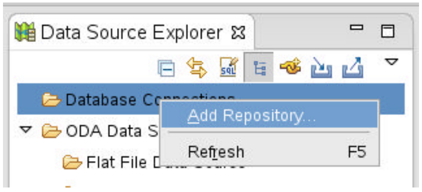
__2. Accept the default workspace name when prompted.

__3. Verify that your workspace appears similar to this:

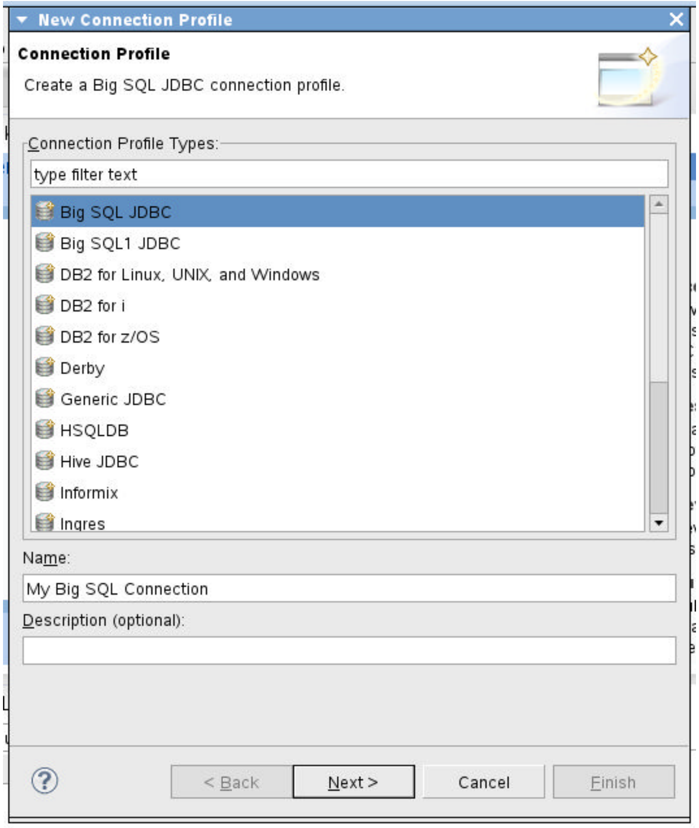

BigInsights - Task Launcher - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Java EE
BigInsights
Project
Packag
Task Launcher for Big Data
IBM
Overview | Accelerate | Design | Develop | Publish and run
Preferences
First Steps
Learn about BigInsights
Interactively explore a graphic to learn how InfoSphere BigInsights enables you to accomplish goals. Drill down on roles and tasks to discover how to use the tools most effectively for your needs.
Create a BigInsights server connection
You must connect to a BigInsights server before you can upload data to a cluster, use the BigInsights console, or publish and run an
Tasks
Accelerate
Leverage sample applications, toolkits and other assets and resources to jump-start your development. Customizable source code and customization assistance provided.
Design
Learn More resources describe considerations and provide resources to help you design the right programs and applications for your analytics tasks
BigInsights Servers
BigInsights Servers
Problems
Console
0 items
Description
Resource
Path
Location
Type

__4. Open the Database Development perspective. **Window > Open Perspective > Other > Database Development.**

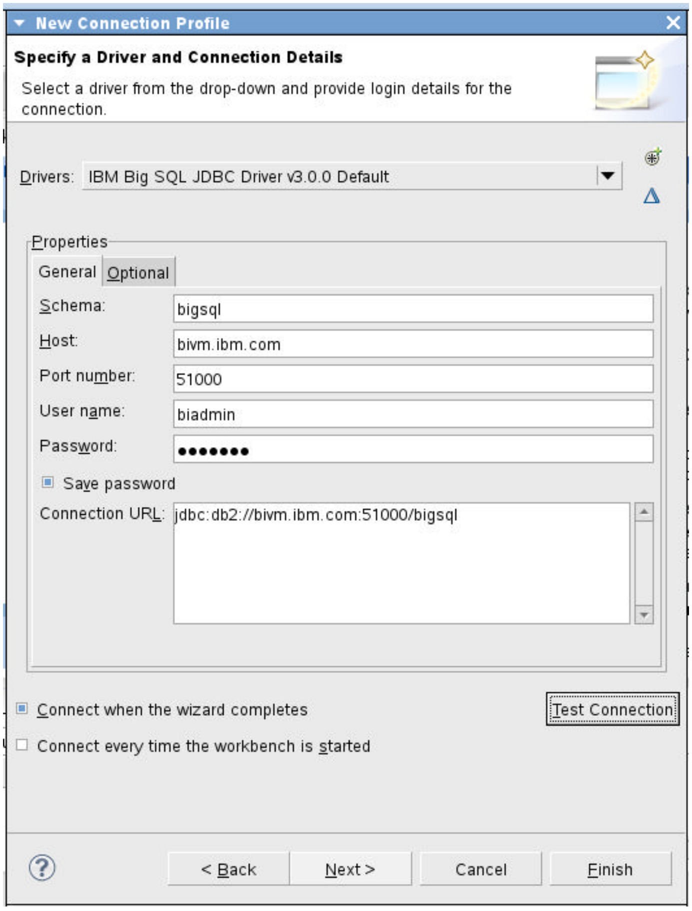
__5. In the Data Source Explorer pane, right click on **Database Connections > Add Repository**



__6. In the **New Connection Profile** menu, select **Big SQL JDBC Driver** and enter a name for your new driver (e.g., My Big SQL Connection). Click **Next.**



__7. Enter the appropriate connection information for your environment, including the host name, port number (51000, by default) user ID, and password. Verify that you have selected the correct JDBC driver at the top. The information shown below contains information for the BigInsights Quick Start Edition VMware image (V3.0).



Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: IBM Big SQL JDBC Driver v3.0.0 Default

Properties

Schema: bigsql

Host: bivm.ibm.com

Port number: 51000

User name: biadmin

Password: ●●●●●●●

☑ Save password

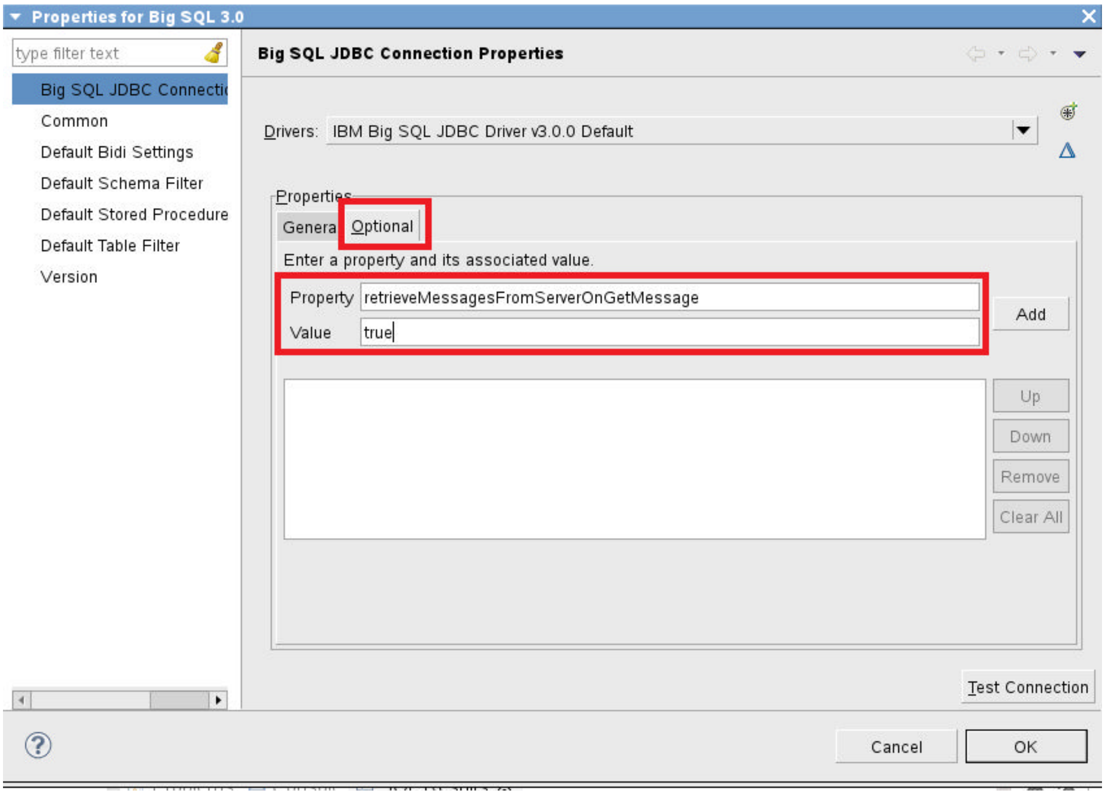
Connection URL: jdbc:db2://bivm.ibm.com:51000/bigsql

☑ Connect when the wizard completes

☐ Connect every time the workbench is started

__8. Click the Optional tab under the Properties heading to expose another menu that allows you to add more properties to the connection.

__9. In the **Property** field, enter `retrieveMessagesFromServerOnGetMessage`, In the **Value** field, enter `true`.



Drivers: IBM Big SQL JDBC Driver v3.0.0 Default

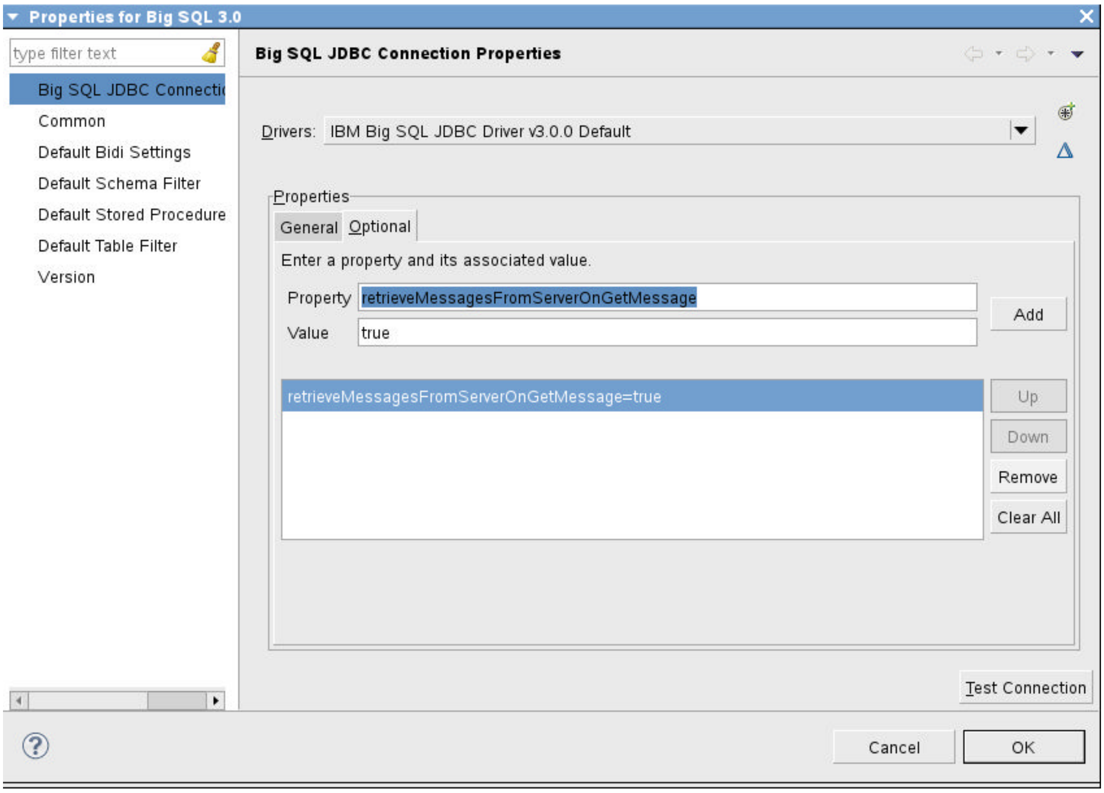
Properties

Enter a property and its associated value.

Property retrieveMessagesFromServerOnGetMessage

Value true

__10. Click **Add**. Verify that your screen appears similar to this:



Properties for Big SQL 3.0

Big SQL JDBC Connection Properties

Drivers: IBM Big SQL JDBC Driver v3.0.0 Default

Properties

Enter a property and its associated value.

Property retrieveMessagesFromServerOnGetMessage

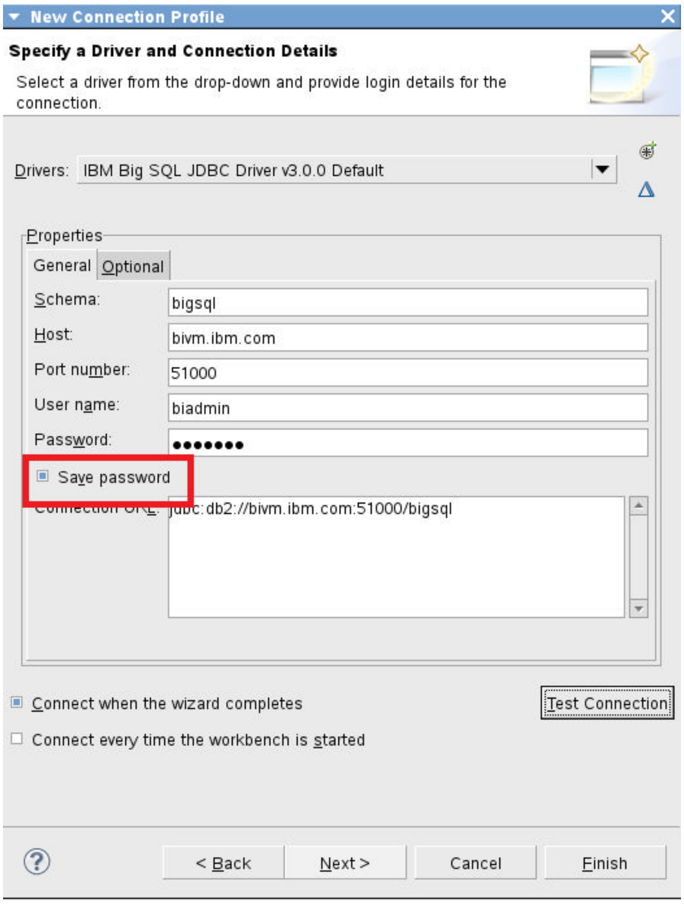
Value true

retrieveMessagesFromServerOnGetMessage=true

__11. Click the General tab again.

__12. Click the **Test connection** button and verify that you can successfully connect to your target Big SQL server.

__13. Click the **Save password** box and **Finish**.



New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: IBM Big SQL JDBC Driver v3.0.0 Default

Properties

General | Optional

Schema: bigsql

Host: bivm.ibm.com

Port number: 51000

User name: biadmin

Password: ●●●●●●●

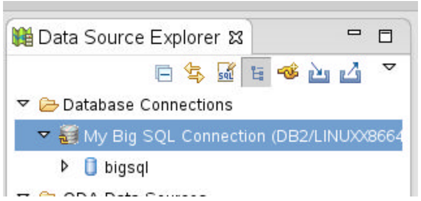
☑ Save password

Connection URL: jdbc:db2://bivm.ibm.com:51000/bigsql

☑ Connect when the wizard completes

☐ Connect every time the workbench is started

__14. In the Data Source Explorer, expand the list of data sources and verify that your Big SQL connection appears.



__15. Return to the BigInsights perspective.

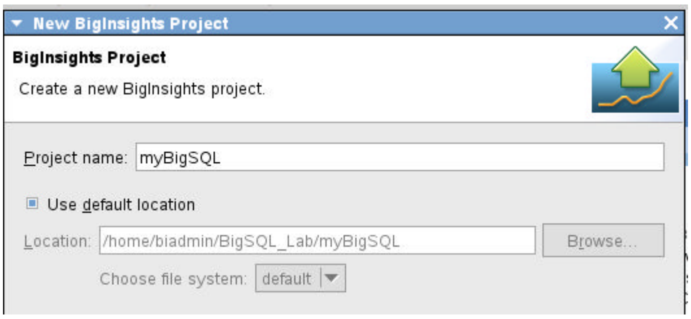
You’re now ready to create and query Big SQL tables.

3.3. Creating a project and a SQL script file

To begin, create a BigInsights project and Big SQL script.

__1. Create a BigInsights project for your work. From the Eclipse menu bar, click **File > New > Other.** Expand the BigInsights folder, and select BigInsights Project, and then click **Next** .

__2. Type myBigSQL in the Project name field, and then click **Finish.**



New BigInsights Project

BigInsights Project

Create a new BigInsights project.

Project name: myBigSQL

■ Use default location

Location: /home/biadmin/BigSQL_Lab/myBigSQL

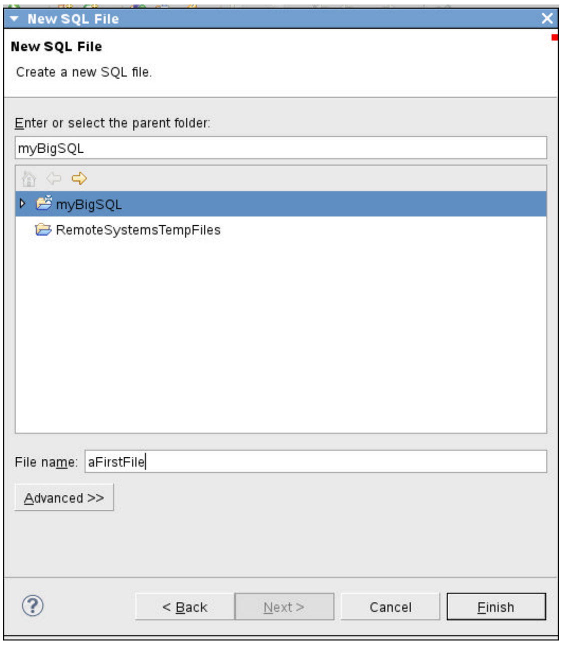
Choose file system: default

__3. If you are not already in the BigInsights perspective, a Switch to the BigInsights perspective window opens. Click **Yes** to switch to the BigInsights perspective.

__4. Create a new SQL script file. From the Eclipse menu bar, click **File > New > Other**. Expand the BigInsights folder, and select SQL script, and then click **Next.**

__5. In the New SQL File window, in the **Enter or select the parent folder** field, select myBigSQL. Your new SQL file is stored in this project folder.

__6. In the File name field, type aFirstFile. The .sql extension is added automatically. Click **Finish.**



New SQL File

New SQL File

Create a new SQL file.

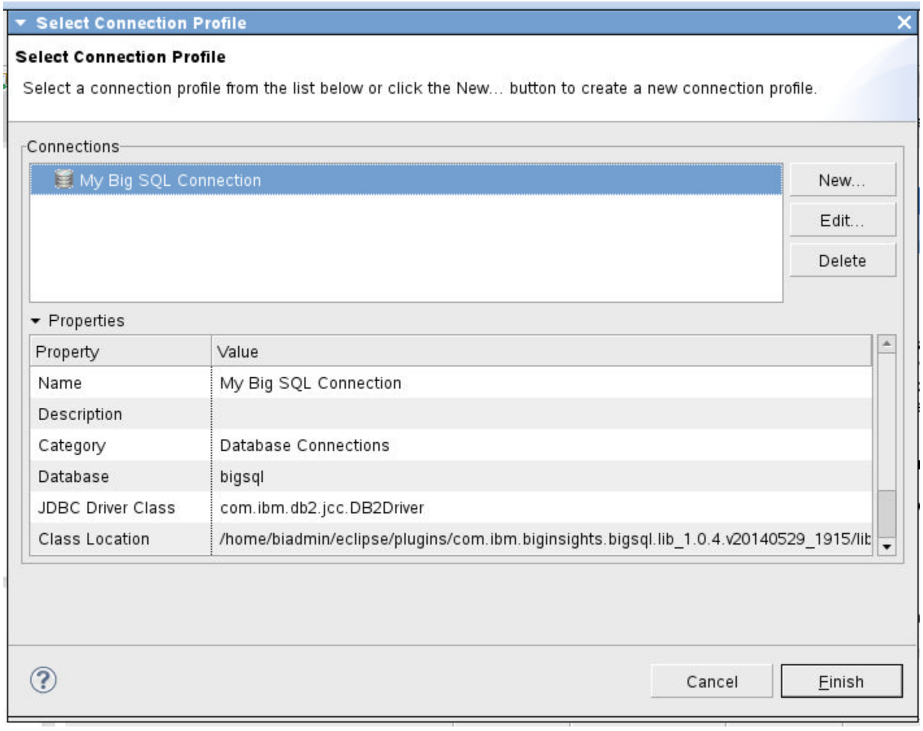
Enter or select the parent folder:

myBigSQL

File name: aFirstFile

__7. In the Select Connection Profile window, select the Big SQL connection. The properties of the selected connection display in the Properties field. When you select the Big SQL connection, the Big SQL database-specific context assistant and syntax checks are activated in the editor that is used to edit your SQL file.

Verify that the connection uses the JDBC driver and database name shown in the Properties pane here.



Select Connection Profile

Select a connection profile from the list below or click the New... button to create a new connection profile.

Connections

My Big SQL Connection

Properties

Property	Value
Name	My Big SQL Connection
Description	
Category	Database Connections
Database	bigsql
JDBC Driver Class	com.ibm.db2.jcc.DB2Driver
Class Location	/home/biadmin/eclipse/plugins/com.ibm.biginsights.bigsql.lib_1.0.4.v20140529_1915/lib

About the driver selection

You may be wondering why you are using a connection that employs the com.ibm.com.db2.jcc.DB2 driver class. In 2014, IBM released a common SQL query engine as part of its DB2 and BigInsights offerings. Doing so provides for greater SQL commonality across its relational DBMS and Hadoop-based offerings. It also brings a greater breadth of SQL function to Hadoop (BigInsights) users. This common query engine is accessible through the DB2 driver. The Big SQL driver remains operational and offers connectivity to an earlier, BigInsights-specific SQL query engine. This lab focuses on using the common SQL query engine.

__8. Click **Finish**.

3.4. Creating and issuing queries

Now you're ready to add some Big SQL statements to the empty script file that you just created. Once you've added some statements, you will execute them and inspect the results.

In some cases, the Eclipse SQL editor may flag certain Big SQL statements as containing syntax errors. Ignore these false warnings and continue with your lab exercises.

__1. Copy the following statement into the SQL script you created earlier:

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

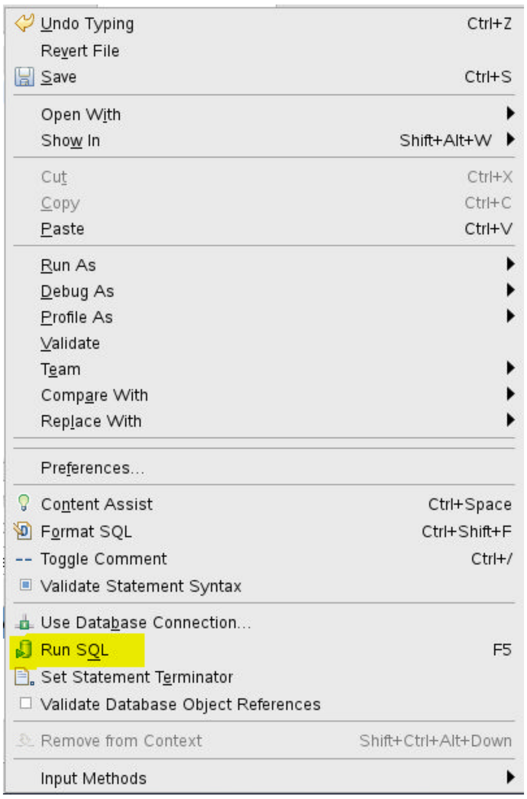
Because you didn't specify a schema name for the table it was created in your default schema, which is your user name. This is equivalent to:

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

where `biadmin` is the current user name.

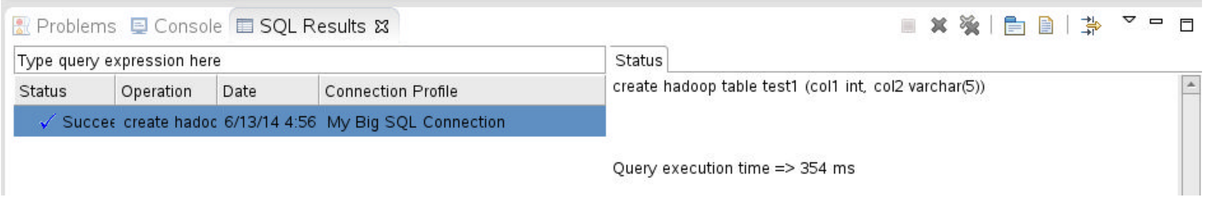
__2. Save your file (press **Ctrl + S** or click **File > Save**).

__3. Right mouse click anywhere in the script to display a menu of options.



__4. Select **Run SQL** or press **F5**. This causes all statements in your script to be executed.

__5. Inspect the SQL Results pane that appears towards the bottom of your display. (If desired, double click on the SQL Results tab to enlarge this pane. Then double click on the tab again to return the pane to its normal size.) Verify that the statement executed successfully. Your Big SQL database now contains a new table named BIADMIN.TEST1 where *BIADMIN* is the name of the current user. Note that your schema and table name were folded into upper case.



Status	Operation	Date	Connection Profile
✓ Succes	create hadoc	6/13/14 4:56	My Big SQL Connection

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

Query execution time => 354 ms

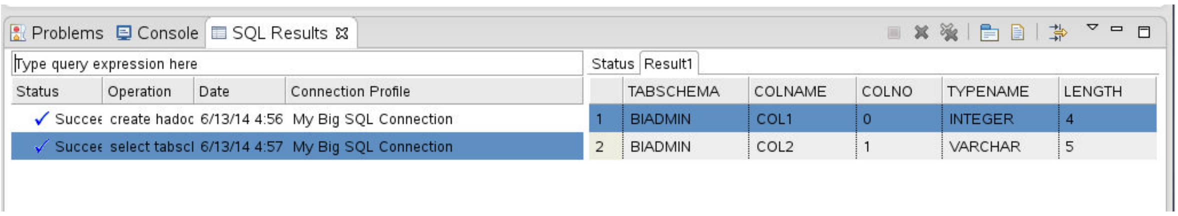
For the remainder of this lab, to execute each SQL statement individually highlight the statement and press **F5**. When you're developing a SQL script with multiple statements, it's generally a good idea to test each statement one at a time to verify that each is working as expected.

__6. From your Eclipse project, query the system for meta data about your `test1` table:

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

In case you're wondering, `syscat.columns` is one of a number of views supplied over system catalog data automatically maintained for you by the Big SQL service.

__7. Inspect the SQL Results to verify that the query executed successfully, and click on the Result1 tab to view its output.



Status	Operation	Date	Connection Profile
✓ Succes	create hadoc	6/13/14 4:56	My Big SQL Connection
✓ Succes	select tabscl	6/13/14 4:57	My Big SQL Connection

	TABSCHEMA	COLNAME	COLNO	TYPENAME	LENGTH
1	BIADMIN	COL1	0	INTEGER	4
2	BIADMIN	COL2	1	VARCHAR	5

__8. Finally, clean up the object you created in the database.

```
drop table test1;
```

__9. Save your file. If desired, leave it open to execute statements for subsequent exercises.

Now that you've set up your Eclipse environment and know how to create SQL scripts and execute queries, you're ready to develop more sophisticated scenarios using Big SQL. In the next lab, you will create a number of tables in your schema and use Eclipse to query them.

Lab 4 Querying structured data with Big SQL

In this lab, you will execute Big SQL queries to investigate data stored in Hadoop. Big SQL provides broad SQL support based on the ISO SQL standard. You can issue queries using JDBC or ODBC drivers to access data that is stored in InfoSphere BigInsights in the same way that you access relational databases from your enterprise applications. Multiple queries can be executed concurrently. The SQL query engine supports joins, unions, grouping, common table expressions, windowing functions, and other familiar SQL expressions.

This tutorial uses sales data from a fictional company that sells and distributes outdoor products to third-party retailer stores as well as directly to consumers through its online store. It maintains its data in a series of FACT and DIMENSION tables, as is common in relational data warehouse environments. In this lab, you will explore how to create, populate, and query a subset of the star schema database to investigate the company's performance and offerings. Note that BigInsights provides scripts to create and populate the more than 60 tables that comprise the sample GOSALESDW database. You will use fewer than 10 of these tables in this lab.

Prior to starting this lab, you must have completed at least one of the two previous labs on JSqsh or Eclipse. In particular, you must be familiar with how to execute queries in your target development platform (JSqsh or Eclipse), and you must have established a connection to your Big SQL 3.0 database. Screen captures shown in this lab are based on Eclipse, as query result sets are generally easier to read.

After you complete the lessons in this module, you will understand how 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.
- Create and run a JDBC client application for Big SQL using Eclipse.

Allow 1.5 hours to complete this lab.

4.1. Creating sample tables and loading sample data

In this lesson, you will create several sample tables and load data into these tables from local files.

__1. Determine the location of the sample data in your local file system and make a note of it. If necessary, ask your instructor for the location of the sample GOSALESDW data used in this lab. You will need to use this path specification when issuing LOAD commands later in this lab.

> Subsequent examples in this section presume your sample data is in the `/opt/ibm/biginsights/bigsql/samples/data` directory. This is the location of the data on the BigInsights VMware image, and it is the default location in typical BigInsights installations.

__2. Create several tables in this schema. Issue each of the following CREATE TABLE statements one at a time, and verify that each completed successfully:

```
-- 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_en          VARCHAR(90) NOT NULL
, country_en         VARCHAR(90) NOT NULL
, region_de          VARCHAR(90), country_de   VARCHAR(90), region_fr   VARCHAR(90)
, country_fr         VARCHAR(90), region_ja    VARCHAR(90), country_ja  VARCHAR(90)
, region_cs          VARCHAR(90), country_cs   VARCHAR(90), region_da   VARCHAR(90)
, country_da         VARCHAR(90), region_el    VARCHAR(90), country_el  VARCHAR(90)
, region_es          VARCHAR(90), country_es   VARCHAR(90), region_fi   VARCHAR(90)
, country_fi         VARCHAR(90), region_hu    VARCHAR(90), country_hu  VARCHAR(90)
, region_id          VARCHAR(90), country_id   VARCHAR(90), region_it   VARCHAR(90)
, country_it         VARCHAR(90), region_ko    VARCHAR(90), country_ko  VARCHAR(90)
, region_ms          VARCHAR(90), country_ms   VARCHAR(90), region_nl   VARCHAR(90)
, country_nl         VARCHAR(90), region_no    VARCHAR(90), country_no  VARCHAR(90)
, region_pl          VARCHAR(90), country_pl   VARCHAR(90), region_pt   VARCHAR(90)
, country_pt         VARCHAR(90), region_ru    VARCHAR(90), country_ru  VARCHAR(90)
, region_sc          VARCHAR(90), country_sc   VARCHAR(90), region_sv   VARCHAR(90)
, country_sv         VARCHAR(90), region_tc    VARCHAR(90), country_tc  VARCHAR(90)
, region_th          VARCHAR(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_descriptionVARCHAR(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.

__3. Load data into each of these tables using sample data provided in files. One at a time, issue each of the following LOAD statements and verify that each completed successfully. Remember to change the SFTP and file path specifications (if needed) to match your environment. The statements will return a warning message providing details on the number of rows loaded, etc.

```
load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.GO_REGION_
DIM.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE GO_REGION_DIM overwrite;
```

```
load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_ORDER_
METHOD_DIM.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE
SLS_ORDER_METHOD_DIM overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_PRODUC
T_BRAND_LOOKUP.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE
SLS_PRODUCT_BRAND_LOOKUP overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_PRODUC
T_DIM.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE SLS_PRODUCT_DIM
overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_PRODUC
T_LINE_LOOKUP.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE
SLS_PRODUCT_LINE_LOOKUP overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_PRODUC
T_LOOKUP.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE SLS_PRODUCT_LOOKUP
overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.SLS_SALES_
FACT.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE SLS_SALES_FACT
overwrite;

load hadoop using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/GOSALESDW.MRK_PROMOT
ION_FACT.txt' with SOURCE PROPERTIES ('field.delimiter'='\t') INTO TABLE MRK_PROMOTION_FACT
overwrite;
```

Let’s explore the LOAD syntax shown in these examples briefly. Each example loads data into a table using a file URL specification that relies on SFTP to locate the source file (in this case, in a file on your local VM). In particular, the SFTP specification includes a valid user ID and password (biadmin/biadmin), the target host server and port (bivm:22), and the full path of the data file on that system. Note that the path is local to the Big SQL server (not your Eclipse client). 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. See the BigInsights Knowledge Center (product documentation) for details. In addition, you can load data directly from a remote relational DBMS via a JDBC connection, will be discussed in a future lab.

__4. 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.

```
-- 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;
```

4.2. Querying the data with Big SQL

Now you're ready to query your tables.   Based on earlier exercises, you've already seen that you can perform basic SQL operations, including projections (to extract specific columns from your tables) and restrictions (to extract specific rows meeting certain conditions you specified). Let's explore a few examples that are a bit more sophisticated.

In this lesson, 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 BigInsights as samples.

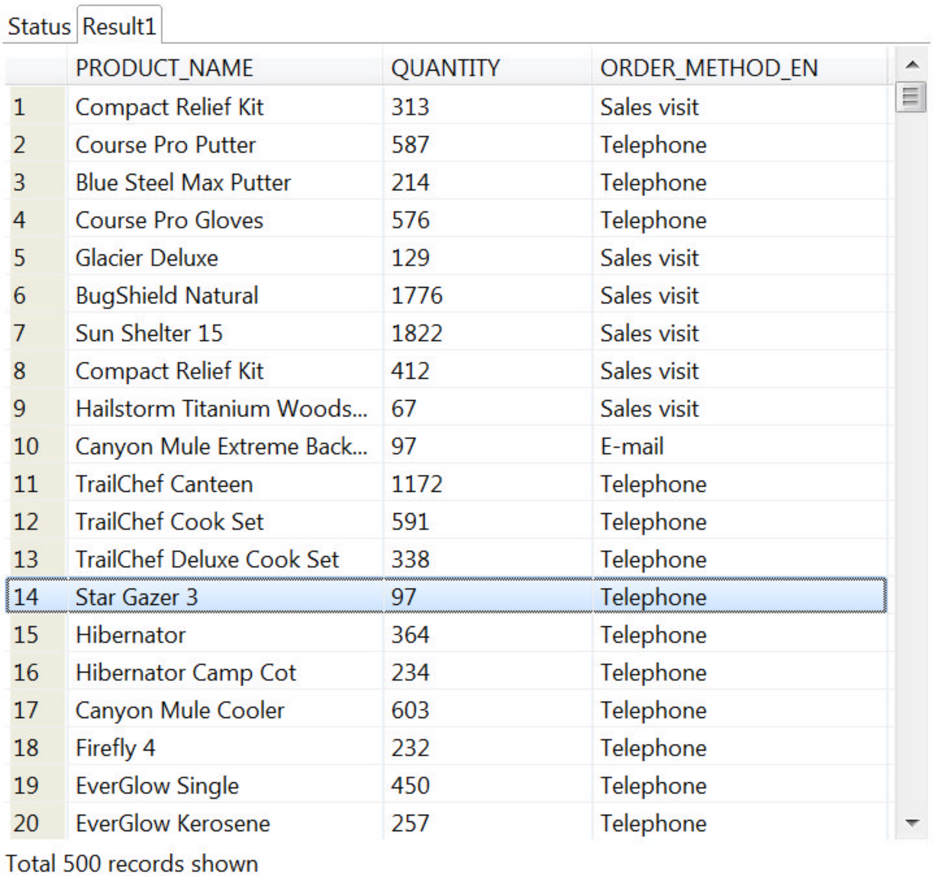
You may find it easiest to use Eclipse to issue the following Big SQL statements. You can also execute them from the JSqsh shell, but some return hundreds of thousands of rows. The Eclipse SQL Results page limits output to only 500 rows. (You can change that value in the Data Management preferences.)

__1. Join data from multiple tables to return the product name, quantity and order method of goods that have been sold. To do so, execute the following query.

```
-- 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;
```

Let’s review a few aspects of this query briefly:

- 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.



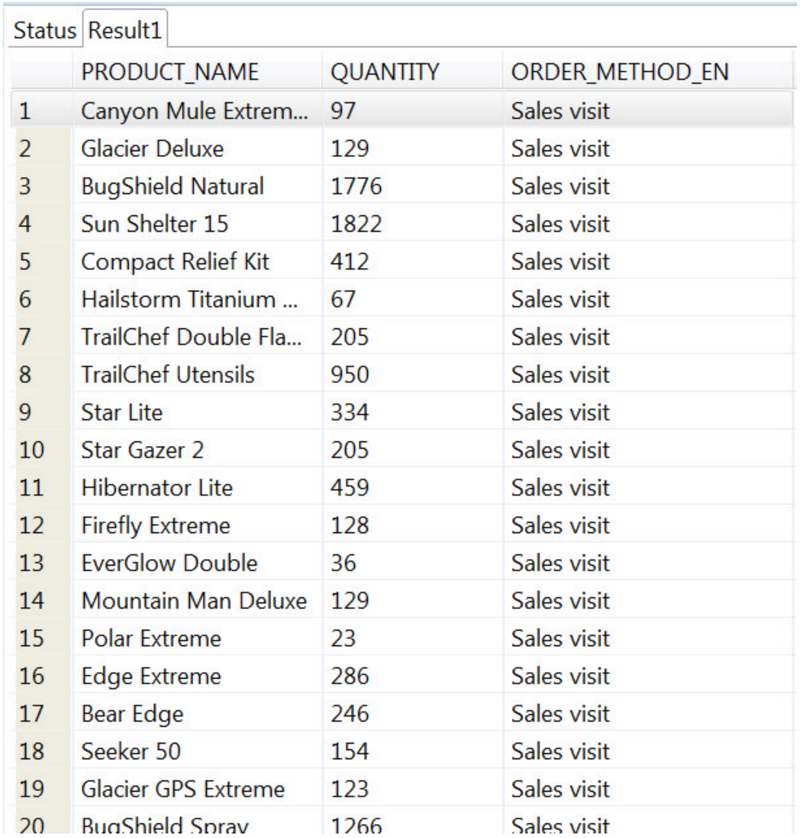
	PRODUCT_NAME	QUANTITY	ORDER_METHOD_EN
1	Compact Relief Kit	313	Sales visit
2	Course Pro Putter	587	Telephone
3	Blue Steel Max Putter	214	Telephone
4	Course Pro Gloves	576	Telephone
5	Glacier Deluxe	129	Sales visit
6	BugShield Natural	1776	Sales visit
7	Sun Shelter 15	1822	Sales visit
8	Compact Relief Kit	412	Sales visit
9	Hailstorm Titanium Woods...	67	Sales visit
10	Canyon Mule Extreme Back...	97	E-mail
11	TrailChef Canteen	1172	Telephone
12	TrailChef Cook Set	591	Telephone
13	TrailChef Deluxe Cook Set	338	Telephone
14	Star Gazer 3	97	Telephone
15	Hibernator	364	Telephone
16	Hibernator Camp Cot	234	Telephone
17	Canyon Mule Cooler	603	Telephone
18	Firefly 4	232	Telephone
19	EverGlow Single	450	Telephone
20	EverGlow Kerosene	257	Telephone

Total 500 records shown

__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 semi-colon:

```
AND order_method_en='Sales visit'
```

__3. Inspect the results, a subset of which is shown below:

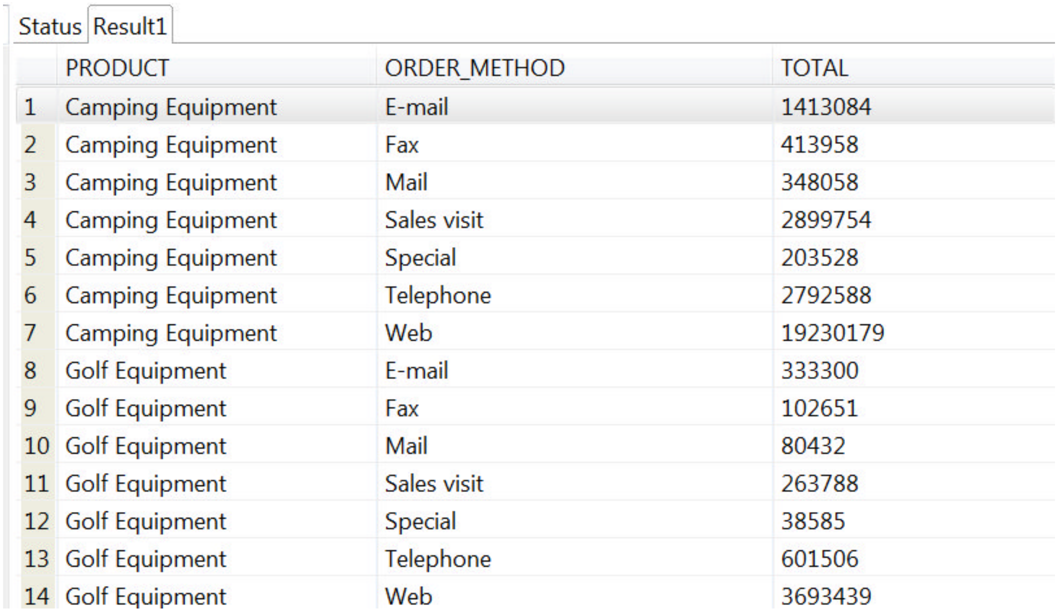


	PRODUCT_NAME	QUANTITY	ORDER_METHOD_EN
1	Canyon Mule Extrem...	97	Sales visit
2	Glacier Deluxe	129	Sales visit
3	BugShield Natural	1776	Sales visit
4	Sun Shelter 15	1822	Sales visit
5	Compact Relief Kit	412	Sales visit
6	Hailstorm Titanium ...	67	Sales visit
7	TrailChef Double Fla...	205	Sales visit
8	TrailChef Utensils	950	Sales visit
9	Star Lite	334	Sales visit
10	Star Gazer 2	205	Sales visit
11	Hibernator Lite	459	Sales visit
12	Firefly Extreme	128	Sales visit
13	EverGlow Double	36	Sales visit
14	Mountain Man Deluxe	129	Sales visit
15	Polar Extreme	23	Sales visit
16	Edge Extreme	286	Sales visit
17	Bear Edge	246	Sales visit
18	Seeker 50	154	Sales visit
19	Glacier GPS Extreme	123	Sales visit
20	BugShield Spray	1266	Sales visit

__4. To find out which sales method of all the methods has the greatest quantity of orders, add 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
1	Camping Equipment	E-mail	1413084
2	Camping Equipment	Fax	413958
3	Camping Equipment	Mail	348058
4	Camping Equipment	Sales visit	2899754
5	Camping Equipment	Special	203528
6	Camping Equipment	Telephone	2792588
7	Camping Equipment	Web	19230179
8	Golf Equipment	E-mail	333300
9	Golf Equipment	Fax	102651
10	Golf Equipment	Mail	80432
11	Golf Equipment	Sales visit	263788
12	Golf Equipment	Special	38585
13	Golf Equipment	Telephone	601506
14	Golf Equipment	Web	3693439

4.3. 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.

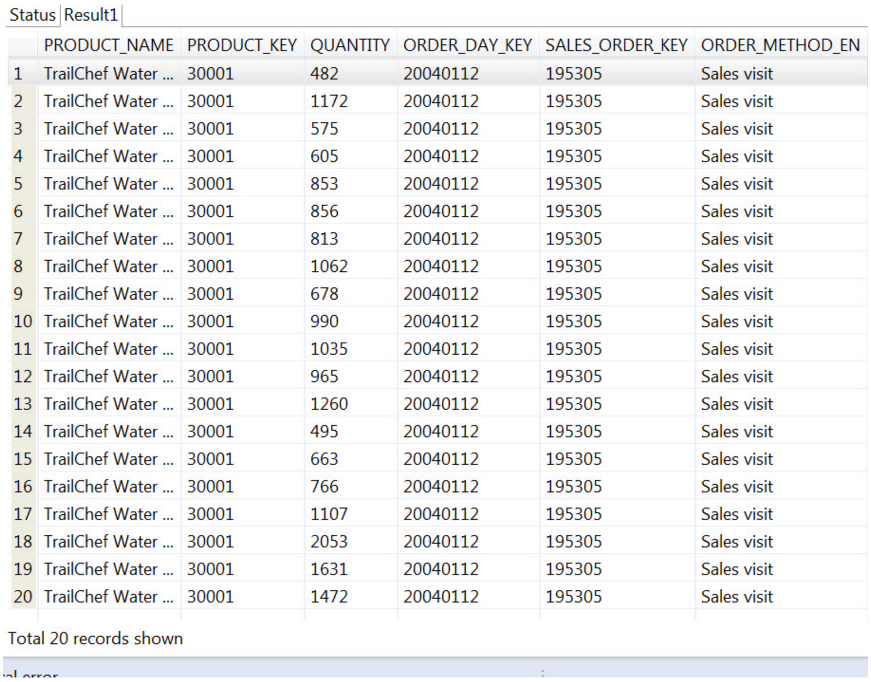
__1. Create a view named MYVIEW that extracts information about product sales featured in marketing promotions. By the way, since the schema name is omitted in both the CREATE and FROM object names, the current schema (your user name), is assumed.

```
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:



Status | Result1

	PRODUCT_NAME	PRODUCT_KEY	QUANTITY	ORDER_DAY_KEY	SALES_ORDER_KEY	ORDER_METHOD_EN
1	TrailChef Water ...	30001	482	20040112	195305	Sales visit
2	TrailChef Water ...	30001	1172	20040112	195305	Sales visit
3	TrailChef Water ...	30001	575	20040112	195305	Sales visit
4	TrailChef Water ...	30001	605	20040112	195305	Sales visit
5	TrailChef Water ...	30001	853	20040112	195305	Sales visit
6	TrailChef Water ...	30001	856	20040112	195305	Sales visit
7	TrailChef Water ...	30001	813	20040112	195305	Sales visit
8	TrailChef Water ...	30001	1062	20040112	195305	Sales visit
9	TrailChef Water ...	30001	678	20040112	195305	Sales visit
10	TrailChef Water ...	30001	990	20040112	195305	Sales visit
11	TrailChef Water ...	30001	1035	20040112	195305	Sales visit
12	TrailChef Water ...	30001	965	20040112	195305	Sales visit
13	TrailChef Water ...	30001	1260	20040112	195305	Sales visit
14	TrailChef Water ...	30001	495	20040112	195305	Sales visit
15	TrailChef Water ...	30001	663	20040112	195305	Sales visit
16	TrailChef Water ...	30001	766	20040112	195305	Sales visit
17	TrailChef Water ...	30001	1107	20040112	195305	Sales visit
18	TrailChef Water ...	30001	2053	20040112	195305	Sales visit
19	TrailChef Water ...	30001	1631	20040112	195305	Sales visit
20	TrailChef Water ...	30001	1472	20040112	195305	Sales visit

Total 20 records shown

4.4. Populating a table with ‘INSERT INTO … SELECT’

With Big SQL v3.0 you can populate a table with data based on the results of a query. In this exercise, 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.

__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;
```

4.5. Optional: 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 lab 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/.

__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.

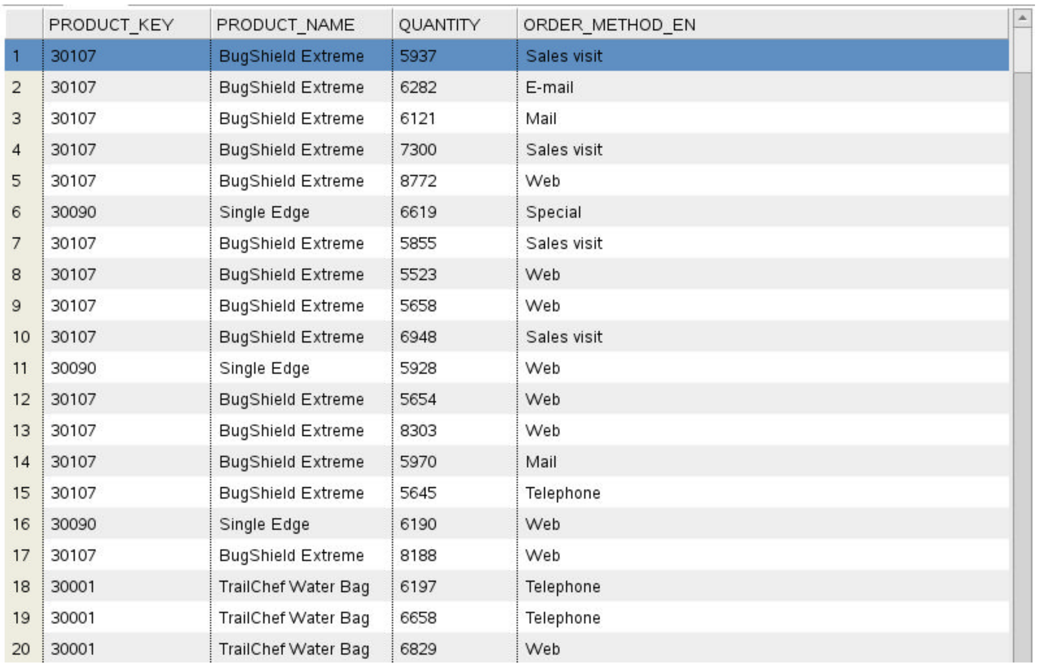
__2. Populate this table with data based on the results of a query. Note that this query joins data from 4 tables you previously defined in Big SQL using a TEXTFILE format. Big SQL will automatically reformat the result set of this query into a Parquet format for storage.

```
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.

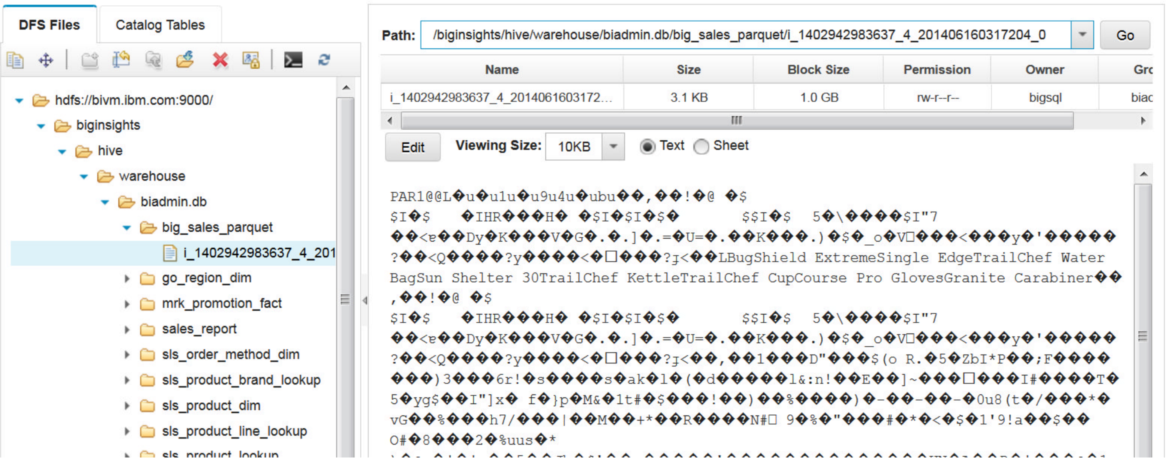
```
select * from big_sales_parquet;
```

__4. Inspect the results. A subset are shown in the screen capture below. The query should return 471 rows.



	PRODUCT_KEY	PRODUCT_NAME	QUANTITY	ORDER_METHOD_EN
1	30107	BugShield Extreme	5937	Sales visit
2	30107	BugShield Extreme	6282	E-mail
3	30107	BugShield Extreme	6121	Mail
4	30107	BugShield Extreme	7300	Sales visit
5	30107	BugShield Extreme	8772	Web
6	30090	Single Edge	6619	Special
7	30107	BugShield Extreme	5855	Sales visit
8	30107	BugShield Extreme	5523	Web
9	30107	BugShield Extreme	5658	Web
10	30107	BugShield Extreme	6948	Sales visit
11	30090	Single Edge	5928	Web
12	30107	BugShield Extreme	5654	Web
13	30107	BugShield Extreme	8303	Web
14	30107	BugShield Extreme	5970	Mail
15	30107	BugShield Extreme	5645	Telephone
16	30090	Single Edge	6190	Web
17	30107	BugShield Extreme	8188	Web
18	30001	TrailChef Water Bag	6197	Telephone
19	30001	TrailChef Water Bag	6658	Telephone
20	30001	TrailChef Water Bag	6829	Web

__5. Optionally, open the Files tab of the Web console, and navigate to the directory containing your table (e.g., `/biginsights/hive/warehouse/biadmin.db/big_sales_parquet).` Note that the contents appears in a format that cannot be read as plain text.



Path: /biginsights/hive/warehouse/biadmin.db/big_sales_parquet/i_1402942983637_4_201406160317204_0

Name	Size	Block Size	Permission	Owner	Gro
i_1402942983637_4_2014061603172...	3.1 KB	1.0 GB	rw-r--r--	bigsql	biad

Viewing Size: 10KB ● Text ○ Sheet

```
PAR1@@L�u�ulu�u9u4u�ubu��,��!�@ �$
$I�$   �IHR���H� �$I�$I�$�       $$I�$  5�\����$I"7
��<e��Dy�K���V�G�.�.]�.=�U=�.��K���.)�$�_o�V□���<���y�'�����
?��<Q����?y����<�□���?J<��LBugShield ExtremeSingle EdgeTrailChef Water
BagSun Shelter 30TrailChef KettleTrailChef CupCourse Pro GlovesGranite Carabiner��
,��!�@ �$
$I�$   �IHR���H� �$I�$I�$�       $$I�$  5�\����$I"7
��<e��Dy�K���V�G�.�.]�.=�U=�.��K���.)�$�_o�V□���<���y�'�����
?��<Q����?y����<�□���?J<��,��1���D"���$(o R.�5�ZbI*P��;F����
���)3���6r!�s����s�ak�l�(�d�����l&:n!��E��]~���□����I#����T�
5�yg$��I"]x� f�}p�M&�1t#�$���!��)��%����)�-��-��-�0u8(t�/���*�
vG��%���h7/���|��M��+*��R����N#□ 9�%�"���#�*�<�$�1'9!a��$��
O#�8���2�%uus�*
```

__6. Optionally, click on the Catalog Tables tab (next to the DFS Files tab), expand the folder associated with your user ID (biadmin), and click on `big_sales_parquet` entry. Note that the data is displayed as a draft BigSheets workbook. BigSheets' HCatalog Reader can process the Big SQL table stored in Parquet. In a later lab, you'll learn more about BigSheets, a spreadsheet-style tool for BigInsights that enables analysts to explore and manipulate data without writing code.

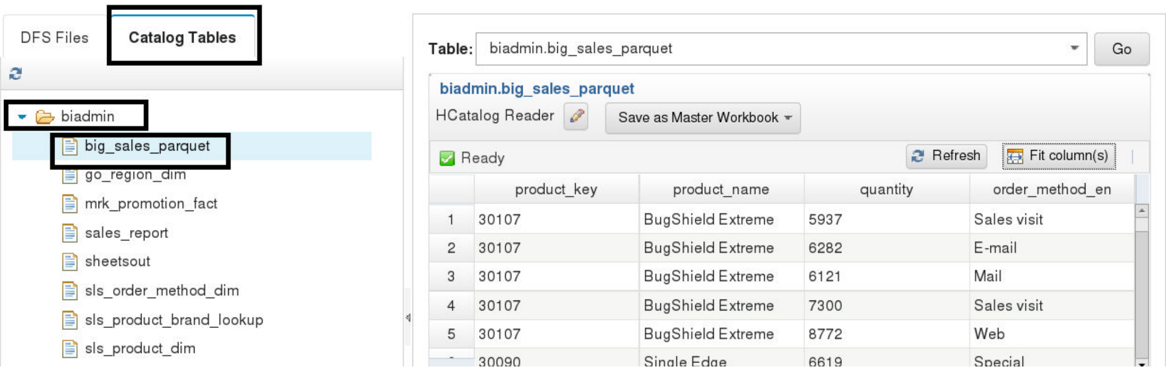


Table: biadmin.big_sales_parquet

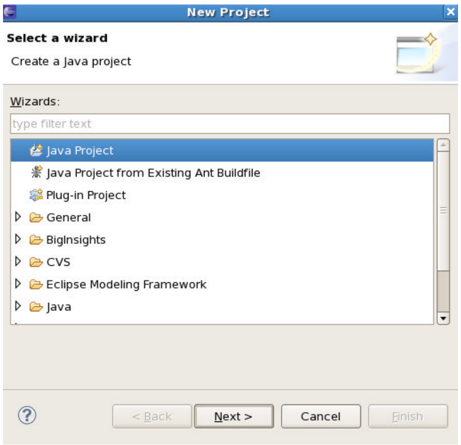
biadmin.big_sales_parquet

	product_key	product_name	quantity	order_method_en
1	30107	BugShield Extreme	5937	Sales visit
2	30107	BugShield Extreme	6282	E-mail
3	30107	BugShield Extreme	6121	Mail
4	30107	BugShield Extreme	7300	Sales visit
5	30107	BugShield Extreme	8772	Web
6	30090	Single Edge	6619	Special

4.6. Optional: Using Big SQL from a JDBC client application (Eclipse)

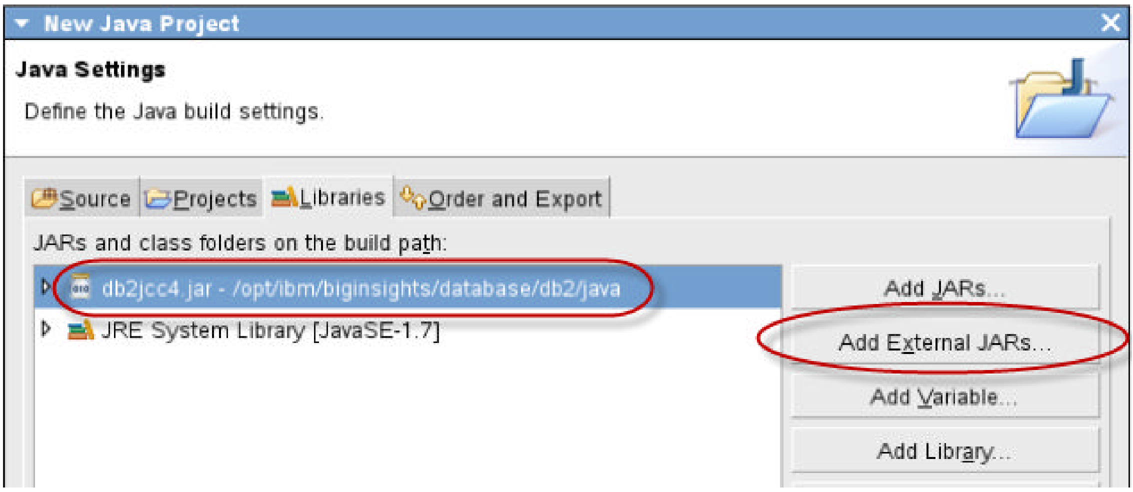
You can write a JDBC client application that uses Big SQL to open a database connection, execute queries, and process the results. In this optional exercise, you'll see how writing a client JDBC application for Big SQL is like writing a client application for any relational DBMS that supports JDBC access.

__1. In the IBM InfoSphere BigInsights Eclipse environment, create a Java project by clicking **File > New >Project**. From the New Project window, select **Java Project**. Click **Next**.



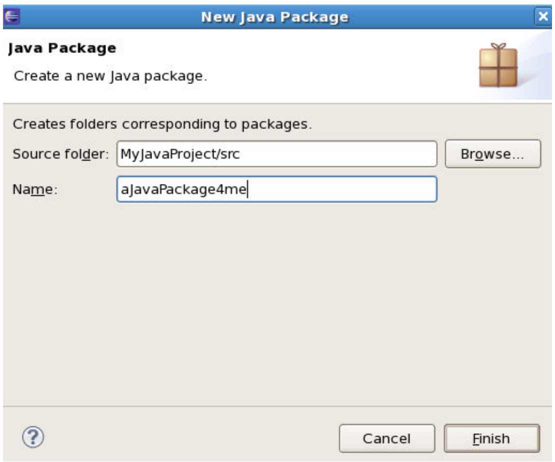
__2. Type a name for the project in the Project Name field, such as MyJavaProject. Click **Next**.

__3. Open the Libraries tab and click **Add External Jars**. Add the DB2 JDBC driver for BigInsights, located at `/opt/ibm/biginsights/database/db2/java/db2jcc4.jar`.


New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
db2jcc4.jar - /opt/ibm/biginsights/database/db2/java
JRE System Library [JavaSE-1.7]
Add JARs...
Add External JARs...
Add Variable...
Add Library...

__4. Click **Finish**. Click **Yes** when you are asked if you want to open the **Java** perspective.

__5. Right-click the MyJavaProject project, and click **New > Package**. In the Name field, in the New Java Package window, type a name for the package, such as aJavaPackage4me. Click **Finish**.



New Java Package

Java Package

Create a new Java package.

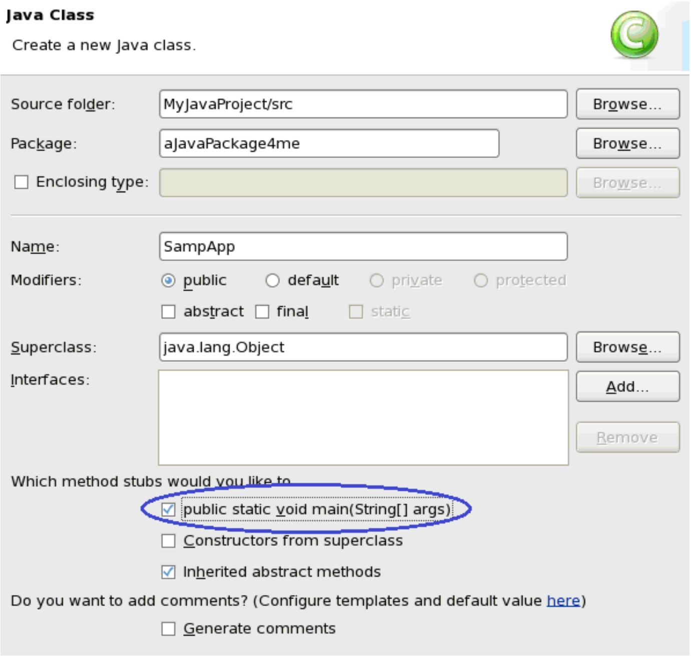
Creates folders corresponding to packages.

Source folder: MyJavaProject/src

Name: aJavaPackage4me

__6. Right-click the aJavaPackage4me package, and click **New > Class**.

__7. In the New Java Class window, in the Name field, type SampApp. Select the public static void main(String[] args) check box. Click **Finish**.



Java Class

Create a new Java class.

Source folder: MyJavaProject/src

Package: aJavaPackage4me

□ Enclosing type:

Name: SampApp

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to

☑ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

__8. Replace the default code for this class and copy or type the following code into the SampApp.java file (you'll find the file in `/opt/ibm/biginsights/bigsql/samples/data/SampApp.java`):

```
package aJavaPackage4me;

//a. Import required package(s)
import java.sql.*;

public class SampApp {
```

```
/**
* @param args
*/

//b. set JDBC & database info
//change these as needed for your environment
static final String db = "jdbc:db2://YOUR_HOST_NAME:51000/bigsql";
static final String user = "YOUR_USER_ID";
static final String pwd = "YOUR_PASSWORD";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
System.out.println("Started sample JDBC application.");

try{
//c. Register JDBC driver -- not needed for DB2 JDBC type 4 connection
// Class.forName("com.ibm.db2.jcc.DB2Driver");

//d. Get a connection
conn = DriverManager.getConnection(db, user, pwd);
System.out.println("Connected to the database.");

//e. Execute a query
stmt = conn.createStatement();
System.out.println("Created a statement.");
String sql;
sql = "select product_color_code, product_number from sls_product_dim " +
      "where product_key=30001";
ResultSet rs = stmt.executeQuery(sql);
System.out.println("Executed a query.");

//f. Obtain results
System.out.println("Result set: ");
while(rs.next()){
//Retrieve by column name
int product_color = rs.getInt("PRODUCT_COLOR_CODE");
int product_number = rs.getInt("PRODUCT_NUMBER");
//Display values
System.out.print("* Product Color: " + product_color + "\n");
System.out.print("* Product Number: " + product_number + "\n");
}

//g. Close open resources
rs.close();
stmt.close();
conn.close();

}catch(SQLException sqlE){
// Process SQL errors
sqlE.printStackTrace();
}catch(Exception e){
// Process other errors
e.printStackTrace();
}
finally{
// Ensure resources are closed before exiting
try{
if(stmt!=null)
stmt.close();
}catch(SQLException sqle2){
} // nothing we can do

try{
if(conn!=null)
conn.close();
}
catch(SQLException sqlE){
sqlE.printStackTrace();
}// end finally block
}// end try block
System.out.println("Application complete");
}}
```

__*a.* After the package declaration, ensure that you include the packages that contain the JDBC classes that are needed for database programming (import java.sql.*;).

__*b.* Set up the database information so that you can refer to it. *Be sure to change the user ID, password, and connection information as needed for your environment.*

__c. Optionally, register the JDBC driver. The class name is provided here for your reference. When using the DB2 Type 4.0 JDBC driver, it's not necessary to specify the class name.

__d. Open the connection.

__e. Run a query by submitting an SQL statement to the database.

__f. Extract data from result set.

__g. Clean up the environment by closing all of the database resources.

__9. Save the file and right-click the Java file and click **Run > Run as > Java Application**.

__10. The results show in the Console view of Eclipse:

```
Started sample JDBC application.
Connected to the database.
Created a statement.
Executed a query.
Result set:
* Product Color: 908
* Product Number: 1110
Application complete
```

4.7. Optional: Creating and querying the full sample database

BigInsights ships with sample SQL scripts for creating, populating, and querying more than 60 tables. These tables are part of the GOSALESDW schema -- a schema that differs from the one used in this lab. (You created tables in the default schema, which is your user ID's schema. In this lab, you logged in as `biadmin`, so all SQL statements defaulted to the `biadmin` schema.)

If desired, use standard Linux operation system facilities to inspect the SQL scripts and sample data for the GOSALESDW schema in the samples directory of $BIGSQL_HOME. By default, this location is `/opt/ibm/biginsights/bigsql/samples.` Within this directory, you'll find subdirectories containing (1) the full sample data for the GOSALESDW tables and (2) a collection of SQL scripts for creating, loading, and querying these tables. Feel free to use the supplied scripts to create the full set of GOSALESDW tables, load data into these tables, and query these tables. Depending on your machine resources, it may take 20 minutes or more to create and populate all the tables. Note that when you query tables in the GOSALESDW schema, you will need to reference the full table name -- e.g., GOSALESDW.GO_REGION_DIM.

Lab 5 Analyzing social media data in BigSheets with Big SQL

In this lesson, you will use a BigSheets workbook as input to Big SQL tables. Unlike previous lessons, the sample data used here isn't typical of data you'd find in a data warehouse. Instead, the sample data for this lesson uses social media data collected from the public domain.

BigInsights provides several sample applications to collect social media data, including a basic application for accessing Boardreader services. However, because a separate license key is required for using third party data collection services (such as Boardreader), you will use some sample social media data from this application that is available in the public domain. Indeed, a developerWorks article on Analyzing Social Media and Structured Data with InfoSphere Biginsights includes social media postings about "IBM Watson" collected by the BigInsights sample Boardreader application for a specific period of time. (The URL for this is http://www.ibm.com/developerworks/data/library/techarticle/dm-1206socialmedia/index.html?ca=dat) This data is in a JSON format, and you will use this data to create a simple BigSheets workbook for exploratory analysis. Quite often, such exploratory work leads to a desire for deeper exploration using a query language such as Big SQL. You'll see how you can accomplish that in this lab exercise.

It's helpful, though not required, to have some knowledge of BigSheets before attempting this lab. A detailed BigSheets lab is available separately.

After you complete the lessons in this module, you will understand how to:

- Create Big SQL tables directly from BigSheets workbooks and query these tables using Big SQL.
- Export data in BigSheets workbooks into one of several common file formats.
- Create a Big SQL table for data you exported from BigSheets.

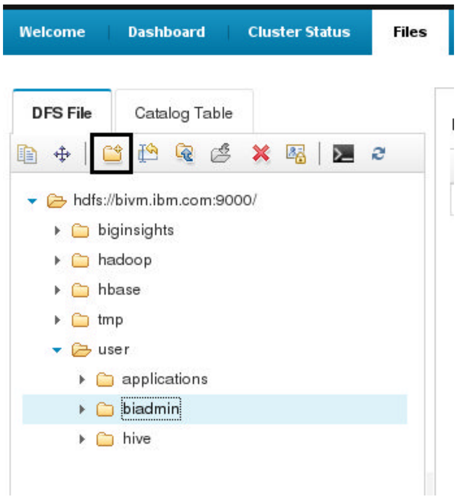
Allow ½ to 1 hour to complete this lab.

5.1. Creating and customizing a BigSheets workbook

In this exercise, you will download sample social media data, create a BigSheets workbook from one of its files, and customize the workbook (in this case, by deleting information that isn't of interest). If your instructor has provided you with a USB drive containing the sample data, you can skip the first step.

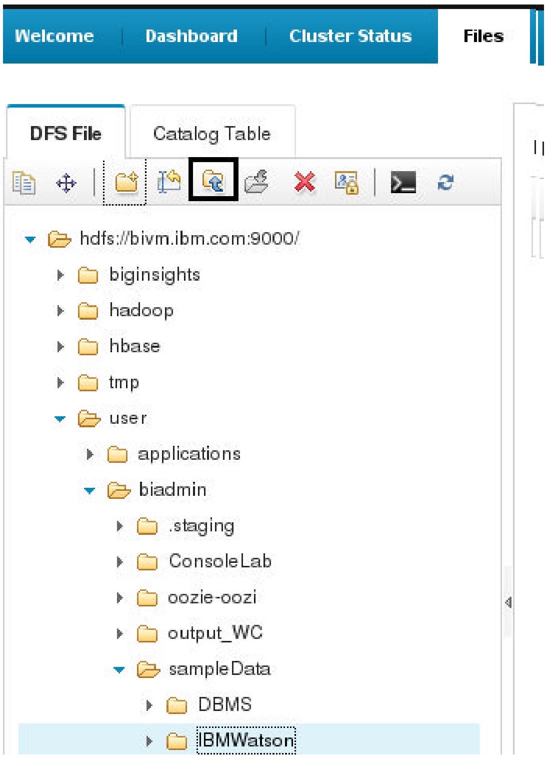
__1. Download the .zip file containing the sample data from the bottom half of the article referenced in the introduction. Unzip the file into a directory on your local file system, such as `/home/biadmin`. You will be working with the blogs-data.txt file.

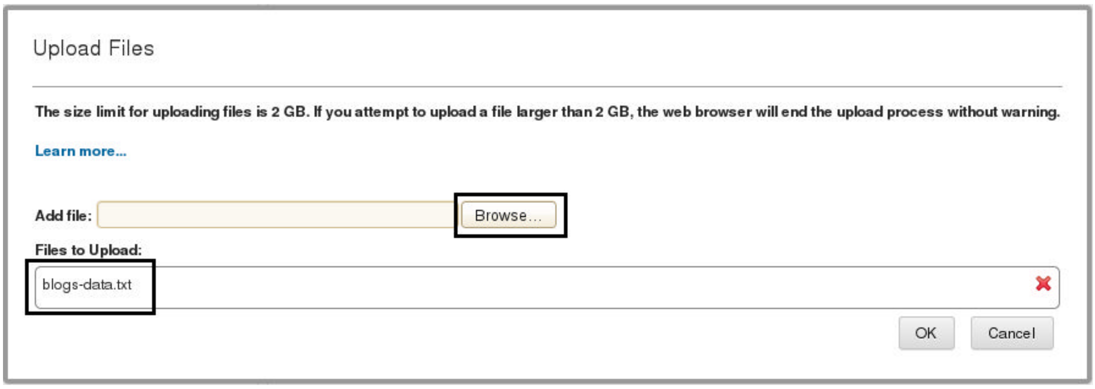
__2. From the Files tab of the Web console, navigate to the `/user/biadmin` directory of your distributed file system. Use the create directory button to create a subdirectory named `sampleData`.



__3. Within the `/user/biadmin/sampleData` directory, repeat the process to create another subdirectory for `IBMWatson`.

__4. Highlight the `IBMWatson` subdirectory and use the upload button to upload the blogs-data.txt file from your local file system. (When prompted, **Browse** through your local file system to locate the file. Then click **OK**.)



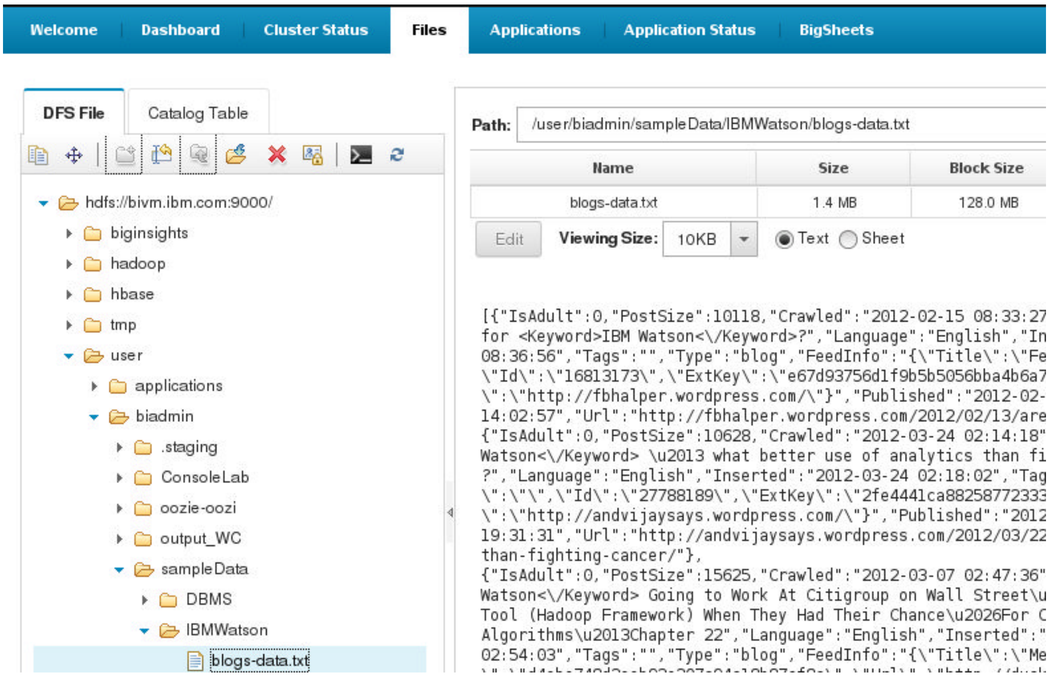


The size limit for uploading files is 2 GB. If you attempt to upload a file larger than 2 GB, the web browser will end the upload process without warning.

Add file:

Files to Upload: blogs-data.txt

__5. Use the file system navigator in the Web console to verify that the file was successfully uploaded into your target DFS directory.



Path: /user/biadmin/sampleData/IBMWatson/blogs-data.txt

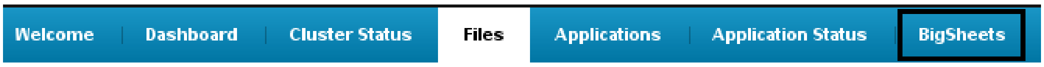
Name	Size	Block Size
blogs-data.txt	1.4 MB	128.0 MB

Viewing Size: 10KB ● Text ○ Sheet

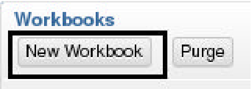
```
[{"IsAdult":0,"PostSize":10118,"Crawled":"2012-02-15 08:33:27
for <Keyword>IBM Watson<\/Keyword>?","Language":"English","In
08:36:56","Tags":"","Type":"blog","FeedInfo":"{\"Title\":\"Fe
\"Id\":\"16813173\",\"ExtKey\":\"e67d93756d1f9b5b5056bba4b6a7
\":\"http://fbhalper.wordpress.com/\"}","Published":"2012-02-
14:02:57","Url":"http://fbhalper.wordpress.com/2012/02/13/are
{"IsAdult":0,"PostSize":10628,"Crawled":"2012-03-24 02:14:18"
Watson<\/Keyword> \u2013 what better use of analytics than fi
?","Language":"English","Inserted":"2012-03-24 02:18:02","Tag
\":\"\",\"Id\":\"27788189\",\"ExtKey\":\"2fe4441ca88258772333
\":\"http://andvijaysays.wordpress.com/\"}","Published":"2012
19:31:31","Url":"http://andvijaysays.wordpress.com/2012/03/22
than-fighting-cancer/"},
{"IsAdult":0,"PostSize":15625,"Crawled":"2012-03-07 02:47:36"
Watson<\/Keyword> Going to Work At Citigroup on Wall Street\u
Tool (Hadoop Framework) When They Had Their Chance\u2026For C
Algorithms\u2013Chapter 22","Language":"English","Inserted":"
02:54:03","Tags":"","Type":"blog","FeedInfo":"{\"Title\":\"Me
```

With the file uploaded into your DFS, you can now begin exploring its contents in BigSheets.

__6. Click on the BigSheets tab of your Web console.

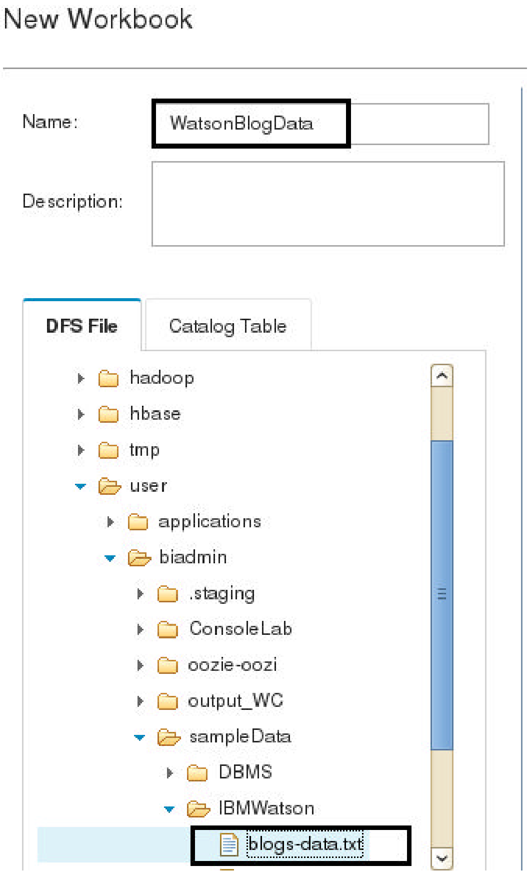


__7. Click **New Workbook**.



__8. In the Name field, type WatsonBlogData.

__9. In the File field, expand the Distributed File System directory, browse to and select the blogs-data-txt file.

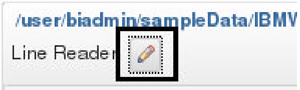


New Workbook

Name: WatsonBlogData

Description:

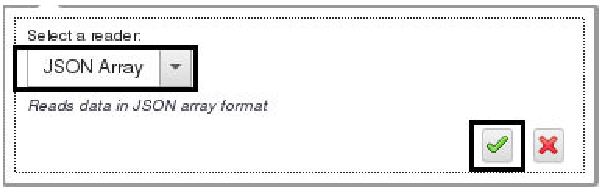
__10. In the Preview area of the screen, select a new reader to map the data to the spreadsheet format. Click the edit icon that looks like a pencil.



/user/biadmin/sampleData/IBMV

Line Reader

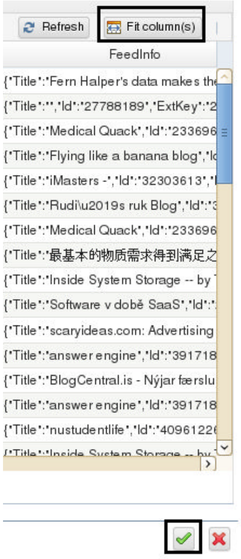
__11. The data in the blogs-data.txt is formatted in a JSON Array structure. Select the JSON Array reader from the list, and click the check mark inside the Select a reader box to apply the reader.



Select a reader: JSON Array

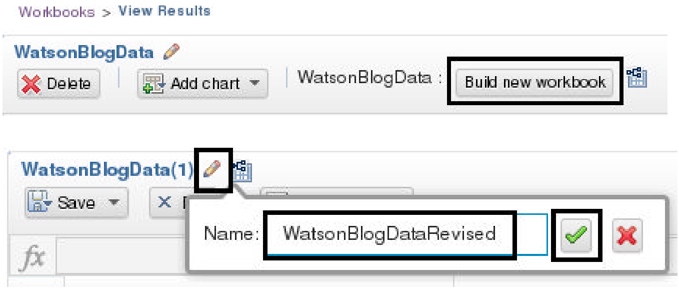
Reads data in JSON array format

__12. Since the data columns exceed the viewing space, click Fit column(s). The first eight columns display in the Preview area. Click the check mark to save the workbook.



```
FeedInfo
{"Title":"Fern Halper's data makes th
{"Title":"","Id":"27788189","ExtKey":"2
{"Title":"Medical Quack","Id":"233696
{"Title":"Flying like a banana blog","Id
{"Title":"iMasters -","Id":"32303613","
{"Title":"Rudi\u2019s ruk Blog","Id":"3
{"Title":"Medical Quack","Id":"233696
{"Title":"最基本的物质需求得到满足之
{"Title":"Inside System Storage -- by
{"Title":"Software v době SaaS","Id":"
{"Title":"scaryideas.com: Advertising
{"Title":"answer engine","Id":"391718
{"Title":"BlogCentral.is - Nýjar færslu
{"Title":"answer engine","Id":"391718
{"Title":"nustudentlife","Id":"4096122
{"Title":"Inside System Storage -- by
```

__13. Click Build new workbook. Rename the workbook by clicking the edit icon, entering the new name of WatsonBlogDataRevised, and clicking the green check mark.

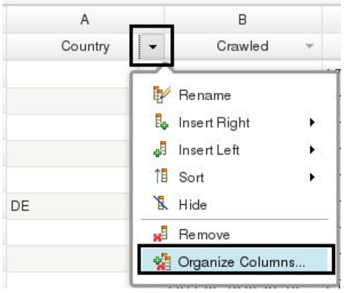


Name: WatsonBlogDataRevised

__14. To more easily see the columns, click Fit column(s). Now columns A through H fit within the width of the sheet.

There are several columns that you do not need to use in your Big SQL table. Remove multiple columns by following these steps:

__15. Click the down arrow in any column heading and select Organize columns.



__16. Click the X next to the following columns to mark them for removal:

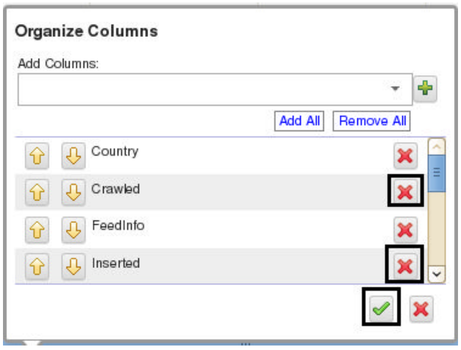
__a. Crawled

__b. Inserted

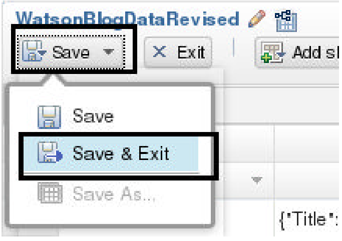
__c. IsAdult

__d. PostSize

__17. Click the green check mark to remove the marked columns.



__18. Click **Save and Exit**, and then Run the workbook. . In the Save workbook dialog, click **Save**. Click **Exit** to start the run process. Click **Run** to run the workbook.



Customizing BigSheets workbooks

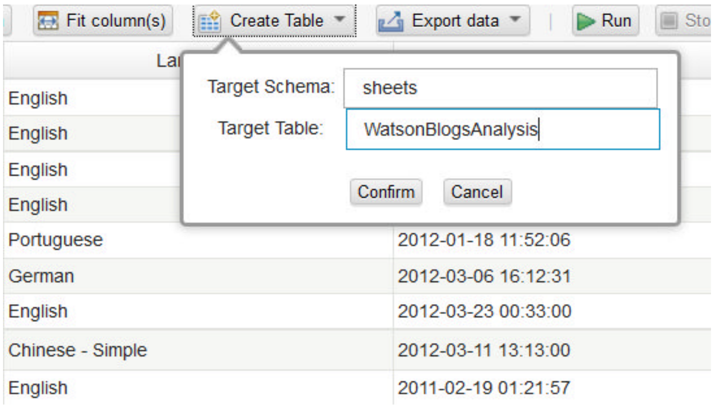
BigSheets enables you to customize your workbooks in much more sophisticated ways than shown here. For example, you can aggregate data, apply formulas to transform your data, join or union data across multiple workbooks, filter data, sort data, analyze data in text-based fields, and so on. Business analysts often use BigSheets to explore potential information of interest, manipulating data in workbooks in various ways before sharing their results with colleagues or downstream applications. A separate lab is available that delves into many popular BigSheets functions.

5.2. Creating a Big SQL table directly from a workbook

BigSheets provides streamlined integration with Big SQL. In this exercise, you will use that feature to create a Big SQL table with data from your workbook and query that data.

__1. If necessary, open the WatsonBlogDataRevised workbook you just created. (In the BigSheets tab of the Web console, double click on the workbook's name to open it.)

__2. Click the **Create Table** button. When prompted, enter WatsonBlogsAnalysis as the Target Table and click **Confirm**. (Leave "sheets" as the Target Schema.)



Target Schema: sheets

Target Table: WatsonBlogsAnalysis

La	
English	
English	
English	
English	
Portuguese	2012-01-18 11:52:06
German	2012-03-06 16:12:31
English	2012-03-23 00:33:00
Chinese - Simple	2012-03-11 13:13:00
English	2011-02-19 01:21:57

__3. Verify that the table is present in the catalog. In the Files tab of the Web console, click the **Catalog Table** tab in the DFS navigator and expand the schema folder for sheets. If desired, click on the watsonblogsanalysis table to display a subset of its contents.

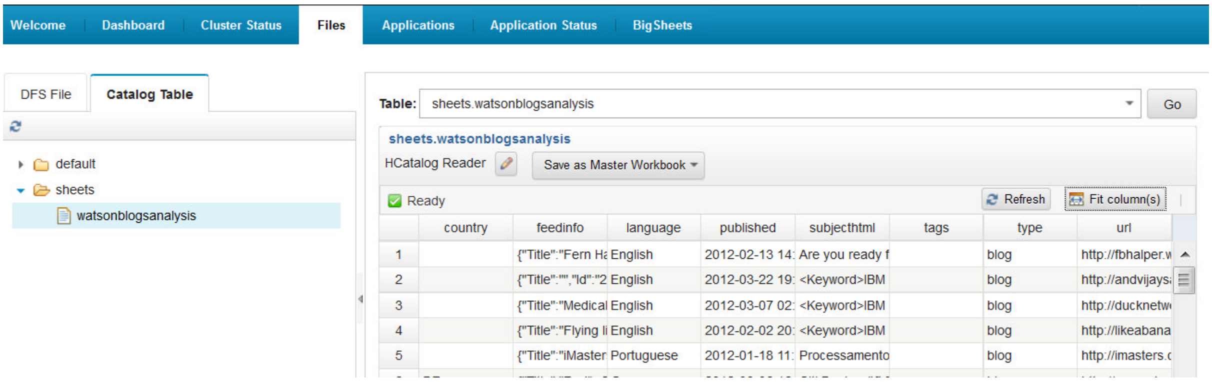


Table: sheets.watsonblogsanalysis

sheets.watsonblogsanalysis

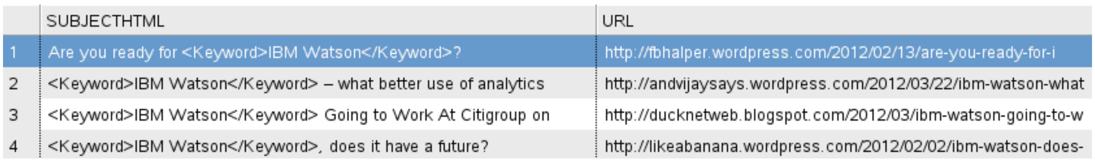
Ready

	country	feedinfo	language	published	subjecthtml	tags	type	url
1		{"Title":"Fern Ha	English	2012-02-13 14:	Are you ready f		blog	http://fbhalper.w
2		{"Title":"","Id":"2	English	2012-03-22 19:	<Keyword>IBM		blog	http://andvijays
3		{"Title":"Medical	English	2012-03-07 02:	<Keyword>IBM		blog	http://ducknetw
4		{"Title":"Flying li	English	2012-02-02 20:	<Keyword>IBM		blog	http://likeabana
5		{"Title":"iMaster	Portuguese	2012-01-18 11:	Processamento		blog	http://imasters.c

__4. Optionally, execute the following query from JSqsh or Eclipse:

```
select subjecthtml, url from sheets.watsonblogsanalysis

fetch first 4 rows only;
```

Verify that 4 rows are returned.

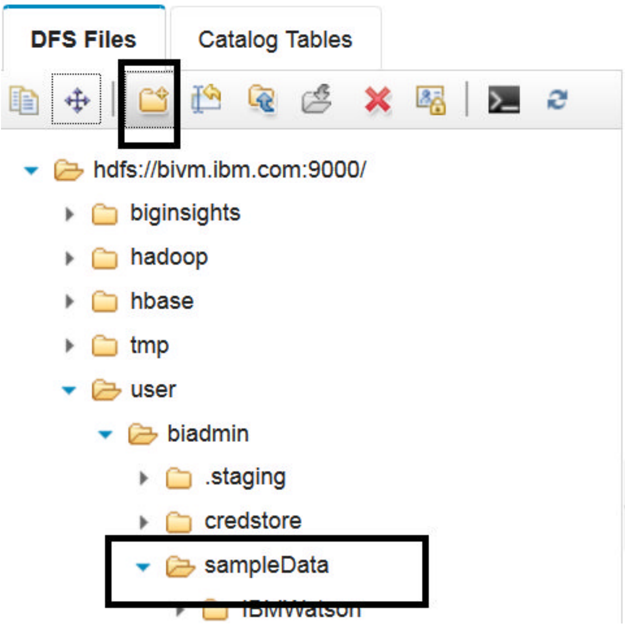


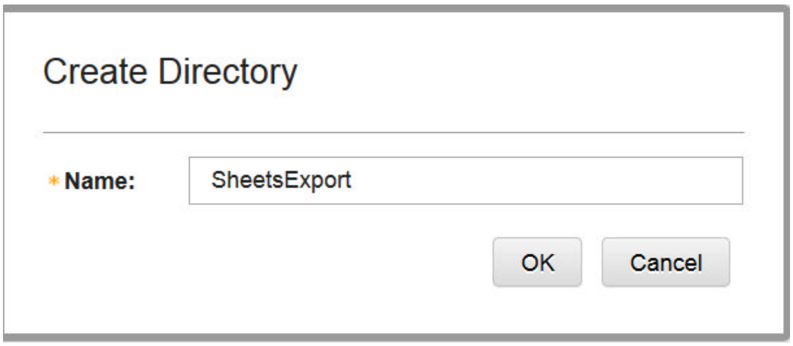
	SUBJECTHTML	URL
1	Are you ready for <Keyword>IBM Watson</Keyword>?	http://fbhalper.wordpress.com/2012/02/13/are-you-ready-for-i
2	<Keyword>IBM Watson</Keyword> – what better use of analytics	http://andvijaysays.wordpress.com/2012/03/22/ibm-watson-what
3	<Keyword>IBM Watson</Keyword> Going to Work At Citigroup on	http://ducknetweb.blogspot.com/2012/03/ibm-watson-going-to-w
4	<Keyword>IBM Watson</Keyword>, does it have a future?	http://likeabanana.wordpress.com/2012/02/02/ibm-watson-does-

5.3. Exporting your workbook

Let's explore how you can export the contents of your workbook into one of several common formats so the data can be easily shared with other applications. BigSheets includes an export function that supports a variety of data formats to suit various application needs. In this exercise, you will export your workbook to a location in your DFS in a tab-separated values (TSV) file format.

__1. In the Files tab of the Web console, create a new subdirectory named SheetsExport under the sampleData directory for your user account.

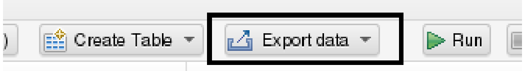



Create Directory

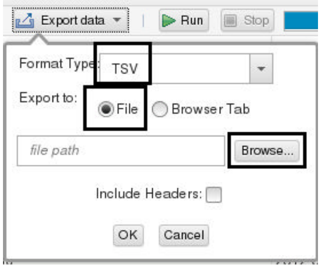
* Name: SheetsExport

__2. Return to BigSheets (click the BigSheets tab) and open the WatsonBlogDataRevised workbook you created earlier.

__3. In the menu bar of the WatsonBlogDataRevised workbook, click **Export as**.



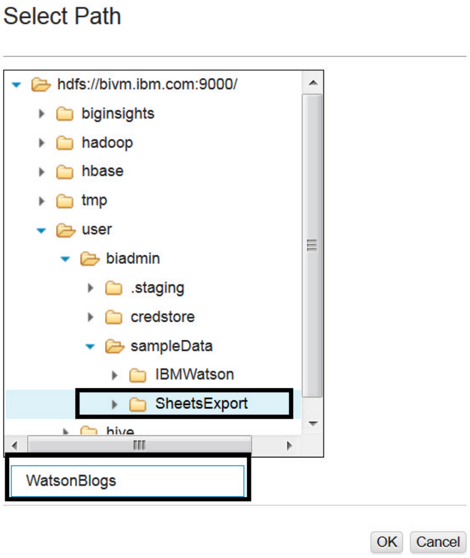
__4. In the drop-down window, select TSV in the Format Type field and click the radio button to Export to File.


Format Type: TSV

Export to: ● File ○ Browser Tab

Include Headers: □

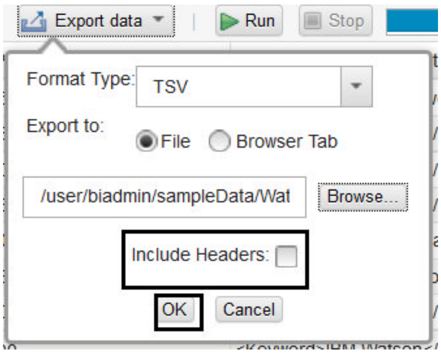
__5. Click **Browse** to select a destination directory. Select your path (`/user/biadmin/sampleData/SheetsExport`) and type a name for the new file, such as WatsonBlogs. Click **OK**.



Select Path

WatsonBlogs

__6. Remove the check mark from the Include Headers box, as you only want to export the data. Click **OK**.



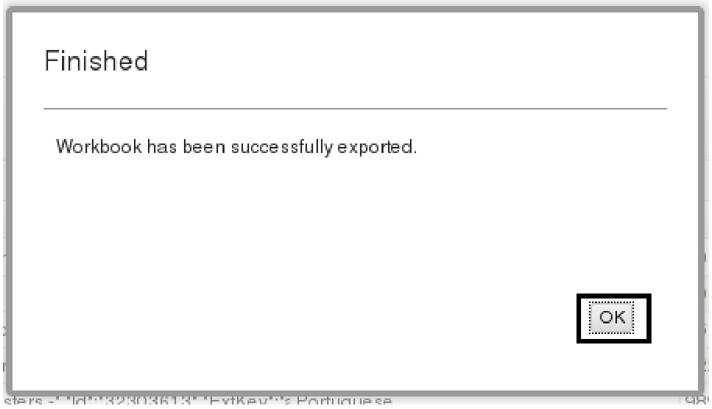
Format Type: TSV

Export to: ● File ○ Browser Tab

/user/biadmin/sampleData/Wat

Include Headers: □

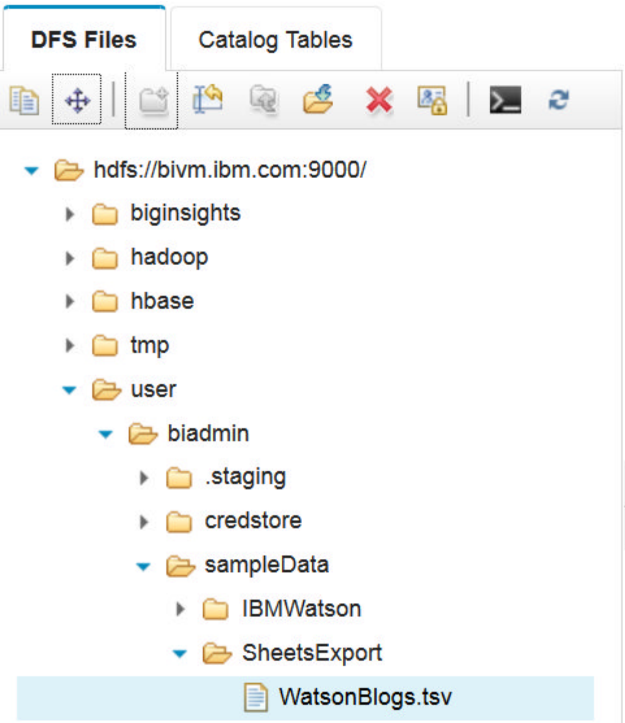
__7. A message dialog shows that the workbook is successfully exported. Click **OK** to close that dialog.



Finished

Workbook has been successfully exported.

__8. Optionally, use the DFS navigator in the Files tab of the Web console to verify that your WatsonBlogs.tsv file was exported to your target directory.



```
hdfs://bivm.ibm.com:9000/
  biginsights
  hadoop
  hbase
  tmp
  user
    biadmin
      .staging
      credstore
      sampleData
        IBMWatson
        SheetsExport
          WatsonBlogs.tsv
```

5.4. Optional: Using Big SQL to work with data exported from BigSheets

In some cases, you may want to share data from BigSheets workbooks with a variety of applications, including Big SQL applications. Rather than creating a Big SQL table directly (as you did in a previous exercise), you may find it convenient to work directly with an exported file, such as the TSV file you created earlier. This optional section explores how you can do so.

As a reminder, the BigSheets workbook that you exported earlier into a TSV file with these fields:

- Country - a two-letter country identifier.

- FeedInfo - information from web feeds, with varying lengths.
- Language - string that identifies the language of the feed.
- Published – date and time of publication.
- SubjectHtml – a string-based subject of varying length.
- Tags - a string of varying length that provides categories.
- Type – a string identifying the source of the web feed, e.g., blog or news feed.
- URL - the web address of the feed, with varying length.

In this section, you will create a Big SQL table for this data that points to the DFS directory where you exported your workbook. In effect, you will be layering a Big SQL schema definition over all files in the directory and creating a table that is managed externally from the Hive warehouse. Later, if you were to drop the Big SQL table, this directory and its contents would remain.

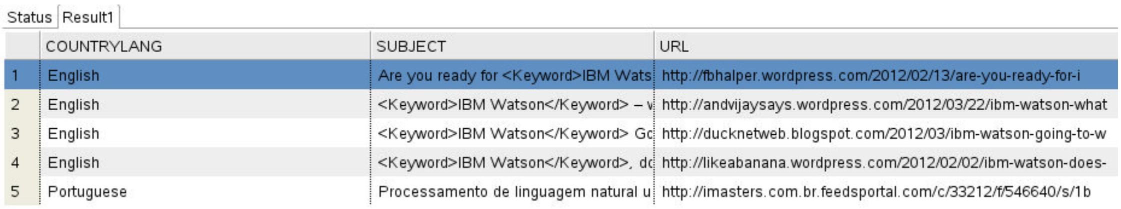
__1. Issue the following CREATE TABLE statement:

```
-- Create an external table based on BigSheets data exported to your DFS.
-- Before running this statement,
-- update the location info as needed for your system
create hadoop table sheetsOut
 (country varchar(2),
 FeedInfo varchar(300),
  countryLang varchar(25),
  published varchar(25),
  subject varchar(300),
  mediatype varchar(20),
  tags varchar(100),
  url varchar(100))
  row format delimited fields terminated by '\t'
  location '/user/biadmin/sampleData/SheetsExport';
```

__2. Query the table.

```
select countrylang, subject, url from sheetsOut fetch first 5 rows only;
```

__3. Inspect the results.



	COUNTRYLANG	SUBJECT	URL
1	English	Are you ready for <Keyword>IBM Wats	http://fbhalper.wordpress.com/2012/02/13/are-you-ready-for-i
2	English	<Keyword>IBM Watson</Keyword> – v	http://andvijaysays.wordpress.com/2012/03/22/ibm-watson-what
3	English	<Keyword>IBM Watson</Keyword> Gc	http://ducknetweb.blogspot.com/2012/03/ibm-watson-going-to-w
4	English	<Keyword>IBM Watson</Keyword>, dc	http://likeabanana.wordpress.com/2012/02/02/ibm-watson-does-
5	Portuguese	Processamento de linguagem natural u	http://imasters.com.br.feedsportal.com/c/33212/f/546640/s/1b

Lab 6 Working with Non-Traditional Data

While data structured in CSV and TSV columns are often stored in BigInsights and loaded into Big SQL tables, you may also need to work with other types of data – data that might require the use of a serializer / deserializer (SerDe). SerDes are common in the Hadoop environment. You'll find a number of SerDes available in the public domain, or you can write your own following typical Hadoop practices.

Using a SerDe with Big SQL is pretty straightforward. Once you develop or locate the SerDe you need, just add its JAR file to the appropriate BigInsights subdirectories. Then stop and restart the Big SQL service, and specify the SerDe class name when you create your table. (Note: If needed, look in your JAR file to determine the class name of the SerDe you'll be using. The CREATE TABLE statement requires the class name, not the JAR file name.)

In this lab exercise, you will use a SerDe to define a table for blog data collected in a JSON (JavaScript Object Notation) format. JSON files have a nested, varied structure defined by the user or application that created them. The JSON-based blog file for this exercise is the same blog file you used as input to BigSheets in a prior lab. As you'll recall, this data was generated by a BigInsights sample application that collects social media data from various public Web sites. The sample data is available for free download as part of a developerWorks article on Analyzing Social Media and Structured Data with InfoSphere Biginsights. (The URL for this article is http://www.ibm.com/developerworks/data/library/techarticle/dm-1206socialmedia/index.html?ca=dat) Before beginning this lab, be sure that you have a copy of the blogs-data.txt file stored in your local file system.

After you complete the lessons in this module, you will understand how to:

- Register a SerDe with Big SQL and Hive
- Create a Big SQL table that uses a SerDe for processing JSON data
- Populate a Big SQL table with JSON data
- Query this Big SQL table

Allow ¼ - ½ hour to complete this lab.

6.1. Registering a SerDe

In this exercise, you will provide a JSON-based SerDe to Big SQL and Hive so that you can later create a table that relies on this SerDe.

__1. Download the hive-json-serde-0.2.jar into a directory of your choice on your local file system, such as `/home/biadmin/sampleData.` (As of this writing, the full URL for this SerDe is https://code.google.com/p/hive-json-serde/downloads/detail?name=hive-json-serde-0.2.jar)

__2. Register the SerDe with BigInsights.

__a. Stop the Big SQL server. (You can do this from a terminal window with the command `$BIGINSIGHTS_HOME/bin/stop.sh bigsql` or you can use the Cluster Status tab of the BigInsights Web console.)

__b. Copy the SerDe .jar file to the `$BIGSQL_HOME/userlib` and `$HIVE_HOME/lib` directories.

__c. Restart the Big SQL server. (You can do this from a terminal window with the command `$BIGINSIGHTS_HOME/bin/start.sh bigsql` or you can use the Cluster Status tab of the Web console.)

6.2. Creating, populating, and querying a table that uses a SerDe

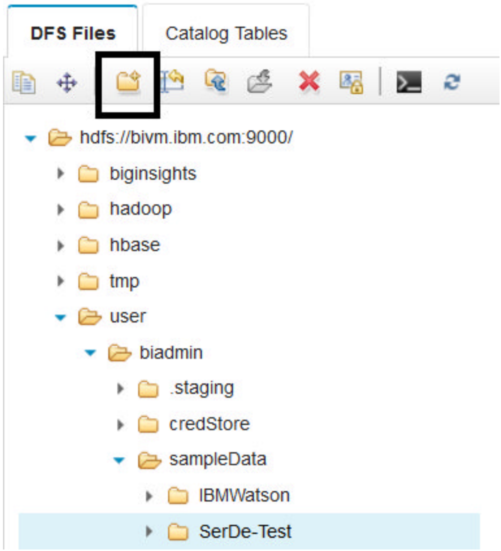
Now that you've registered your SerDe, you're ready to use it. In this section, you will create a table that relies on the SerDe you just registered. For simplicity, this will be an externally managed table – i.e., a table created over a user directory that resides outside of the Hive warehouse. This user directory will contain all the table's data in files. As part of this exercise, you will upload the sample blogs-data.txt file into the target DFS directory.

Creating a Big SQL table over an existing DFS directory has the effect of populating this table with all the data in the 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. This is consistent with the Hive concept of an externally managed table.

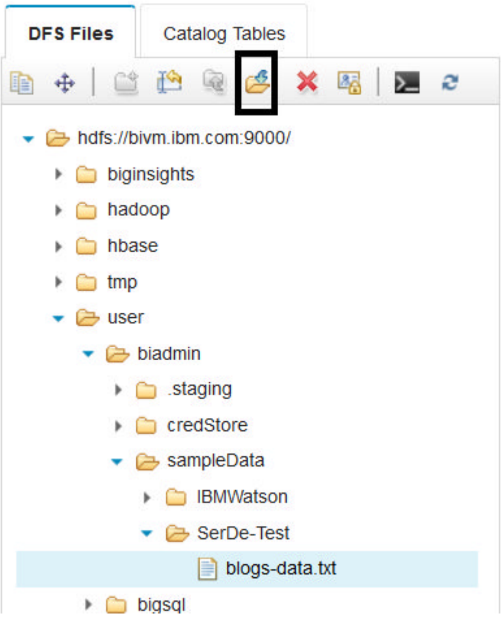
Once the table is created, you'll query that table. In doing so, you'll note that the presence of a SerDe is transparent to your queries.

__1. If necessary, download the .zip file containing the sample data from the bottom half of the article referenced in the introduction. Unzip the file into a directory on your local file system, such as `/home/biadmin`. You will be working with the blogs-data.txt file.

From the Files tab of the Web console, navigate to the `/user/biadmin/sampleData` directory of your distributed file system. Use the create directory button to create a subdirectory named `SerDe-Test`.



__2. Upload the blogs-data.txt file into `/user/biadmin/sampleData/SerDe-Test.`



__3. Return to the Big SQL execution environment of your choice (JSqsh or Eclipse).

__4. Execute the following statement, which creates a TESTBLOGS table that includes a LOCATION clause that specifies the DFS directory containing your sample blogs-data.txt file:

```
create hadoop table if not exists testblogs (
Country String,
Crawled String,
FeedInfo String,
Inserted String,
IsAdult int,
Language String,
Postsize int,
Published String,
SubjectHtml String,
Tags String,
Type String,
Url String)
row format serde  'org.apache.hadoop.hive.contrib.serde2.JsonSerde'
location '/user/biadmin/sampleData/SerDe-Test';
```

About this code . . . .

The CREATE HADOOP TABLE statement specifies the class in the SerDe .jar file that is responsible for processing the input record into a "row" that Big SQL (and Hive) can understand. Because you copied the SerDe .jar file into the appropriate Big SQL and Hive directories earlier, the runtime engine will be able to locate this class in the .jar file and successfully execute the CREATE HADOOP TABLE statement.

Quite commonly, new users will specify the .jar file in the CREATE HADOOP TABLE statement instead of the class file. Doing so will result in a runtime error.

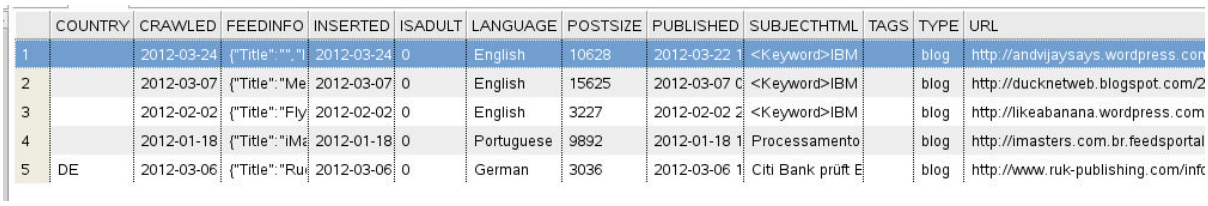
You will also notice the LOCATION clause as used in the previous lab. If you do not have the input file already in your DFS at this path, you will have to manually move or copy the file to this location.

__5. Finally, query the table using the following statement.

```
select * from testblogs

where subjecthtml is not null

fetch first 5 rows only;
```

Note that the SELECT syntax does not reference the SerDe in any way.

__6. Inspect the results.



	COUNTRY	CRAWLED	FEEDINFO	INSERTED	ISADULT	LANGUAGE	POSTSIZE	PUBLISHED	SUBJECTHTML	TAGS	TYPE	URL
1		2012-03-24	{"Title":"","I	2012-03-24	0	English	10628	2012-03-22 1	<Keyword>IBM		blog	http://andvijaysays.wordpress.con
2		2012-03-07	{"Title":"Me	2012-03-07	0	English	15625	2012-03-07 0	<Keyword>IBM		blog	http://ducknetweb.blogspot.com/2
3		2012-02-02	{"Title":"Fly	2012-02-02	0	English	3227	2012-02-02 2	<Keyword>IBM		blog	http://likeabanana.wordpress.com
4		2012-01-18	{"Title":"iMa	2012-01-18	0	Portuguese	9892	2012-01-18 1	Processamento		blog	http://imasters.com.br.feedsportal
5	DE	2012-03-06	{"Title":"Ru	2012-03-06	0	German	3036	2012-03-06 1	Citi Bank prüft E		blog	http://www.ruk-publishing.com/infc

Lab 7 Using advanced Big SQL features

This lab explores some of the advanced features that are new to Big SQL 3.0. Big SQL employs a powerful database optimization engine to execute your queries. You will need to do previous labs to load data in your schema before attempting this lab.

In the first exercise, you will examine the data access plan Big SQL will use to retrieve your data using a feature called EXPLAIN. There are many ways to organize your data, such as partitioning and indexing, which will speed up your queries. Some of the details about your data are maintained automatically during runtime any time you query it, but in order to give the optimizer a more complete picture you'll want to collect meta data statistics using the ANALYZE TABLE command. This is highly recommended when dealing with large volumes of data (but less critical for this sample lab).

A later exercise enables you to explore fine-grained access control mechanisms in Big SQL. These mechanisms, implemented through the definition of ROLES and use of GRANT/REVOKE statements, enable an administrator to define specific column- and row-based access restrictions. For example, only managers might be permitted to see information in the PROFIT column of a table. Similarly, brokers might be permitted to see only portfolio information for their clients.

Before starting this lab, you should be familiar with how to execute commands from JSqsh or Eclipse. Earlier labs provided information on these topics.

Allow 1 – 1.5 hours to complete this lab.

7.1. Understanding your data access plan (EXPLAIN) – from the command line

The EXPLAIN feature enables you to inspect the data access plan selected by the Big SQL optimizer for your query. Such information is highly useful for performance tuning. This exercise introduces you to EXPLAIN, a Big SQL feature that stores meta data in a set of EXPLAIN tables.

__1. Launch the JSqsh shell from a command window for your BigInsights instance.

```
$JSQSH_HOME/bin/jsqsh
```

__2. Connect to your Big SQL 3.0 database. For example, if you created a connection named “bigsql” earlier, you would issue this command:

```
\connect bigsql -P biadmin
```

__3. You need to create the EXPLAIN tables, call the SYSINSTALLOBJECTS procedure. In this invocation, the tables will be created only for your user account. By casting a NULL in the last parameter, a single set of EXPLAIN tables can be created in schema SYSTOOLS, which can be used for all users.

```
CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN', 'C', CAST (NULL AS VARCHAR(128)), CAST
(NULL AS VARCHAR(128)));
```

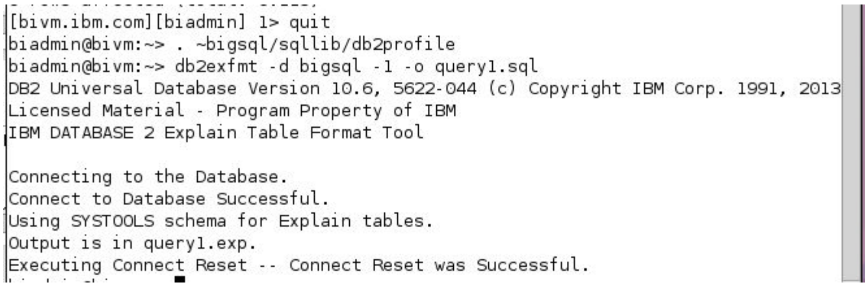
__4. Now, let's capture the EXPLAIN access plan for a query. One way to do this is by prefixing the query with the command `EXPLAIN PLAN WITH SNAPSHOT FOR`. In this way, the query isn't executed, but the access plan is saved in the EXPLAIN tables. Copy paste the following command and run it:

```
explain plan with snapshot for
select distinct product_key, introduction_date
from sls_product_dim;
```

Information about the data access strategy for this query is stored in the EXPLAIN tables, which you'll explore shortly. There are various tools to view the "explained" access plan. For example, you could use the Data Studio, IBM Query Tuning perspective and Query Tuner Project. In this lab, we will use a DB2 utility called db2exfmt, executed from the bash shell. Exit the JSqsh shell (enter `quit` on the command line).

__5. Invoke db2exfmt with the -1 option, which is a handy way to retrieve the plan from the LAST statement which was "explained" by the current user.

```
. ~bigsql/sqllib/db2profile
db2exfmt -d bigsql -1 -o query1.exp
```



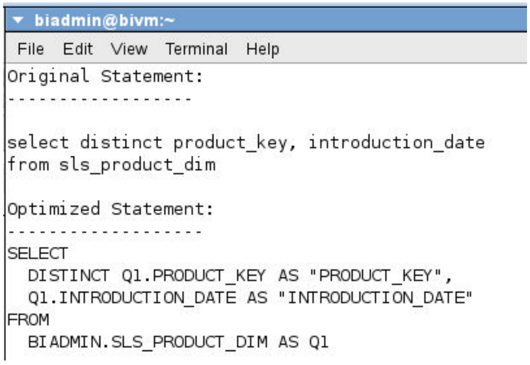
```
[bivm.ibm.com][biadmin] 1> quit
biadmin@bivm:~> . ~bigsql/sqllib/db2profile
biadmin@bivm:~> db2exfmt -d bigsql -1 -o query1.sql
DB2 Universal Database Version 10.6, 5622-044 (c) Copyright IBM Corp. 1991, 2013
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Using SYSTOOLS schema for Explain tables.
Output is in query1.exp.
Executing Connect Reset -- Connect Reset was Successful.
```

__6. Investigate the contents of the query1.exp file. For example, type

```
more query1.exp
```

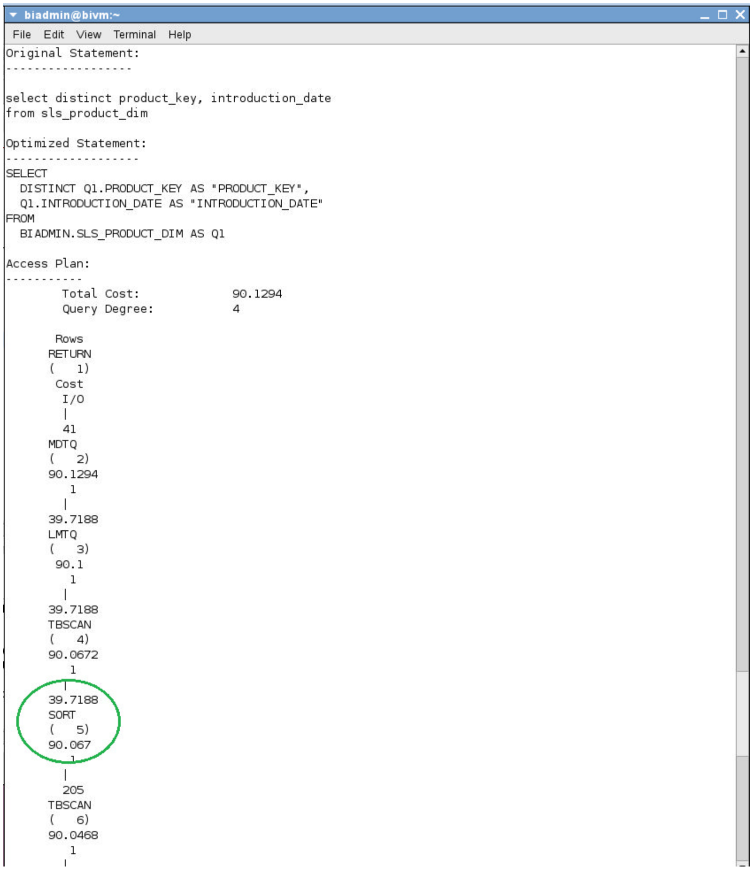
on the command line. Press the space bar to scroll forward through the output one page at a time, and enter b to page backward.

__7. Original Statement vs. Optimized Statement. Sometimes, the optimizer may decide to rewrite the query in a more efficient manner. For example, replacing IN lists with JOINS. In this lab, the optimized Statement show that no further optimization has been done.



```
Original Statement:
------------------

select distinct product_key, introduction_date
from sls_product_dim

Optimized Statement:
-------------------
SELECT
  DISTINCT Q1.PRODUCT_KEY AS "PRODUCT_KEY",
  Q1.INTRODUCTION_DATE AS "INTRODUCTION_DATE"
FROM
  BIADMIN.SLS_PRODUCT_DIM AS Q1
```

__8. Notice the SORT operation and the total number of operations for this explained Access Plan.



```
Original Statement:
------------------

select distinct product_key, introduction_date
from sls_product_dim

Optimized Statement:
-------------------
SELECT
  DISTINCT Q1.PRODUCT_KEY AS "PRODUCT_KEY",
  Q1.INTRODUCTION_DATE AS "INTRODUCTION_DATE"
FROM
  BIADMIN.SLS_PRODUCT_DIM AS Q1

Access Plan:
-----------
        Total Cost:             90.1294
        Query Degree:           4

         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
          41
        MDTQ
        (   2)
        90.1294
           1
          |
        39.7188
        LMTQ
        (   3)
         90.1
           1
          |
        39.7188
        TBSCAN
        (   4)
        90.0672
           1
          |
        39.7188
        SORT
        (   5)
        90.067
           1
          |
          205
        TBSCAN
        (   6)
        90.0468
           1
          |
```

__9. Next we will alter the table. Again launch the JSQSH shell:

```
$JSQSH_HOME/bin/jsqsh bigsql -P biadmin
```

__10. Execute the following alter command:

```
alter table sls_product_dim
add constraint newPK primary key (product_key) not enforced;
```

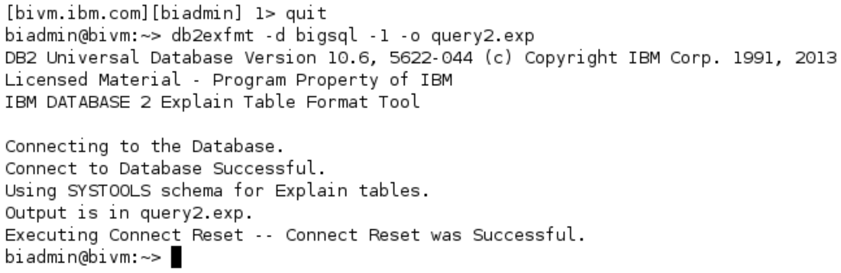
This will alter the table to have a non-enforced PK constraint.

__11. Now lets do another explain command on the altered table:

```
explain plan with snapshot for
select distinct product_key, introduction_date
from sls_product_dim;
```

__12. Again quit the JSQSH shell by entering **quit** and invoke the following command:

```
db2exfmt -d bigsql -1 -o query2.exp
```



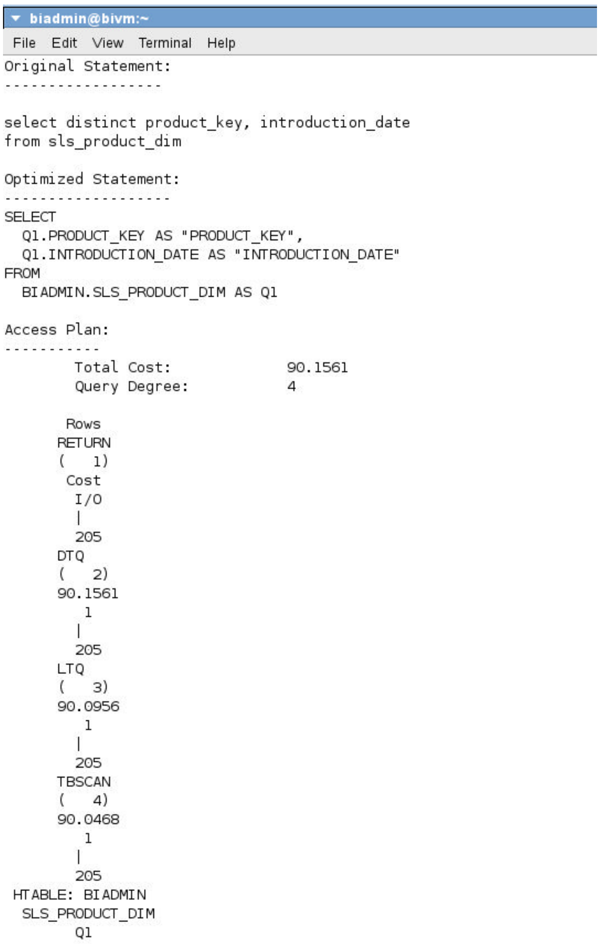
```
[bivm.ibm.com][biadmin] 1> quit
biadmin@bivm:~> db2exfmt -d bigsql -1 -o query2.exp
DB2 Universal Database Version 10.6, 5622-044 (c) Copyright IBM Corp. 1991, 2013
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Using SYSTOOLS schema for Explain tables.
Output is in query2.exp.
Executing Connect Reset -- Connect Reset was Successful.
biadmin@bivm:~> 
```

__13. Investigate the contents of the query2.exp file. For example, type

```
more query2.exp
```

on the command line. Press the space bar to scroll forward through the output one page at a time, and enter b to page backward.

__14. Again you will see an Original Statement vs Optimized Statement they will not differ, but take a look at the new Access Plan. The SORT operation is no longer used and there are fewer operations in total.



```
Original Statement:
-----------------

select distinct product_key, introduction_date
from sls_product_dim

Optimized Statement:
------------------
SELECT
  Q1.PRODUCT_KEY AS "PRODUCT_KEY",
  Q1.INTRODUCTION_DATE AS "INTRODUCTION_DATE"
FROM
  BIADMIN.SLS_PRODUCT_DIM AS Q1

Access Plan:
-----------
        Total Cost:             90.1561
        Query Degree:           4

       Rows
      RETURN
      (   1)
       Cost
        I/O
        |
        205
      DTQ
      (   2)
      90.1561
         1
        |
        205
      LTQ
      (   3)
      90.0956
         1
        |
        205
      TBSCAN
      (   4)
      90.0468
         1
        |
        205
 HTABLE: BIADMIN
  SLS_PRODUCT_DIM
        Q1
```

7.2. Collecting statistics with the ANALYZE TABLE command

The ANALYZE TABLE command collects statistics about your Big SQL data. These statistics influence query optimization, enabling the Big SQL query engine to select an efficient data access path to satisfy your query.

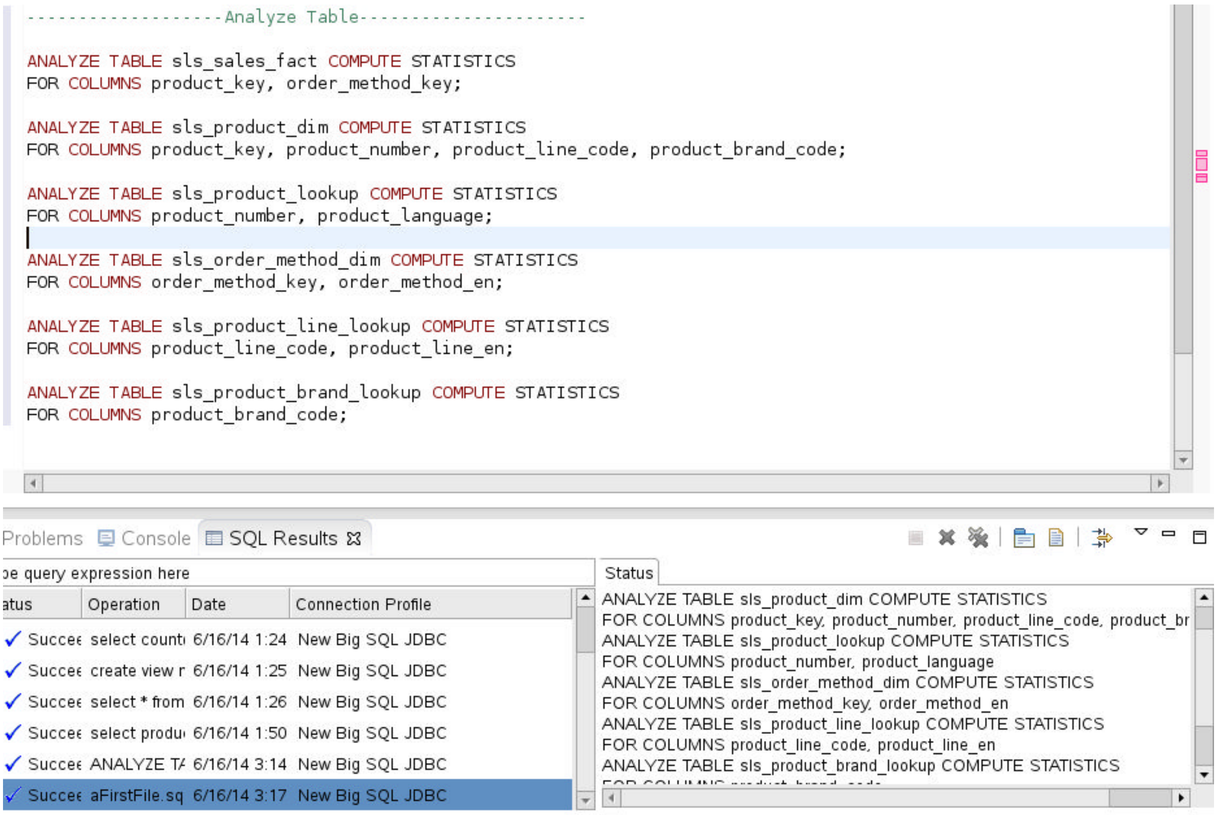
__1. Providing a list of columns to the ANALYZE TABLE command is optional but gives valuable information to the optimizer.

```
ANALYZE TABLE sls_sales_fact COMPUTE STATISTICS
FOR COLUMNS product_key, order_method_key
;
ANALYZE TABLE sls_product_dim COMPUTE STATISTICS
FOR COLUMNS product_key, product_number, product_line_code, product_brand_code
;
ANALYZE TABLE sls_product_lookup COMPUTE STATISTICS
FOR COLUMNS product_number, product_language
;
ANALYZE TABLE sls_order_method_dim COMPUTE STATISTICS
FOR COLUMNS order_method_key, order_method_en
;
ANALYZE TABLE sls_product_line_lookup COMPUTE STATISTICS
FOR COLUMNS product_line_code, product_line_en
;
ANALYZE TABLE sls_product_brand_lookup COMPUTE STATISTICS
FOR COLUMNS product_brand_code
;
```

ANALYZE TABLE syntax

It is recommended to include FOR COLUMNS and a list of columns to the ANALYZE TABLE command. Choose those columns found in your WHERE, ORDER BY, GROUP BY and DISTINCT clauses.

__2. Copy and paste the previous ANALYZE TABLE commands into your Eclipse editor.



```
-------------------Analyze Table---------------------

ANALYZE TABLE sls_sales_fact COMPUTE STATISTICS
FOR COLUMNS product_key, order_method_key;

ANALYZE TABLE sls_product_dim COMPUTE STATISTICS
FOR COLUMNS product_key, product_number, product_line_code, product_brand_code;

ANALYZE TABLE sls_product_lookup COMPUTE STATISTICS
FOR COLUMNS product_number, product_language;

ANALYZE TABLE sls_order_method_dim COMPUTE STATISTICS
FOR COLUMNS order_method_key, order_method_en;

ANALYZE TABLE sls_product_line_lookup COMPUTE STATISTICS
FOR COLUMNS product_line_code, product_line_en;

ANALYZE TABLE sls_product_brand_lookup COMPUTE STATISTICS
FOR COLUMNS product_brand_code;
```

atus	Operation	Date	Connection Profile
✓ Succee	select count	6/16/14 1:24	New Big SQL JDBC
✓ Succee	create view r	6/16/14 1:25	New Big SQL JDBC
✓ Succee	select * from	6/16/14 1:26	New Big SQL JDBC
✓ Succee	select produ	6/16/14 1:50	New Big SQL JDBC
✓ Succee	ANALYZE TA	6/16/14 3:14	New Big SQL JDBC
✓ Succee	aFirstFile.sq	6/16/14 3:17	New Big SQL JDBC

Status

```
ANALYZE TABLE sls_product_dim COMPUTE STATISTICS
FOR COLUMNS product_key, product_number, product_line_code, product_br
ANALYZE TABLE sls_product_lookup COMPUTE STATISTICS
FOR COLUMNS product_number, product_language
ANALYZE TABLE sls_order_method_dim COMPUTE STATISTICS
FOR COLUMNS order_method_key, order_method_en
ANALYZE TABLE sls_product_line_lookup COMPUTE STATISTICS
FOR COLUMNS product_line_code, product_line_en
ANALYZE TABLE sls_product_brand_lookup COMPUTE STATISTICS
```

This may take a few minutes to run. Verify that each command succeeded.

7.3. Enhancing SQL security with fine-grained access control

Big SQL offers administrators additional SQL security control mechanisms for row and column access through the definition of ROLES and GRANT/REVOKE statements. In this exercise, you will mask information about gross profits for sales from all users who are not a MANAGER. That is, all users with SELECT privileges will be able to query the `SLS_SALES_FACT` table, but information in the GROSS_PROFIT column will display as 0.0 unless the user was granted the role of a MANAGER. Any user who is a MANAGER will be able to see the underlying data values for GROSS_PROFIT.

To complete this lab, you must have access to multiple user accounts. Examples in this lab are based on the following user IDs (in addition to `biadmin`):

- `bigsql`, which has SECADM authority for your database environment.
- `user1` and `user2`, which have USER privileges for BigInsights.

These accounts are part of the default configuration for the BigInsights 3.0 VMware image. The `bigsql` account has a password of `bigsql`, while the `user1` and `user2` accounts both have passwords of `passw0rd`. If you're using an environment with a different configuration and different accounts, you will need to adjust the Big SQL examples in this section to match your environment.

In addition, prior to starting this lab, you must have created the SLS_SALES_FACT table and populated it with data, as described in an earlier lab.

As background, the fine-grained access control supported by BigInsights is based on row and column access control mechanisms that first became available in DB2. These mechanisms involve row permissions and column masks. Once activated, no database user is automatically exempt (including the table owner). Details about row and column access control (RCAC) are beyond the scope of this lab. However, very briefly,

- A row permission captures a row access control rule for a specific table. It's basically a search condition that describes which rows a user can access. An example of such a rule may be that managers can only see rows for their employees.

- A column mask is a column access control rule for a specific column in a specific table. When defining a mask, you use a CASE expression to describe what a user sees when accessing the column. For example, a mask could be defined so that a teller can see only the last 4 digits of a credit card number.

Row permissions and column masks require no SQL application changes; row and column access control is based on specific rules that are transparent to existing SQL applications.

With that backdrop, you're ready to get started. You can use either JSqsh or Eclipse for most of the work in this lab. Most of the screen captures included here are based on an Eclipse environment.

Initially, you'll implement a column-based access control scenario.

__1. If you haven't already done so, create a Big SQL database connection in JSqsh or Eclipse that logs in as the bigsql user. (The bigsql user ID has specific privileges required to execute certain commands that follow.) If necessary, review earlier lab exercises that described how to create a new database connection in JSqsh or Eclipse.

__2. Create two new roles: one for MANAGER and one for STAFF.

```
-- column based access control example
-- allow only managers to see gross profit for sales
--
-- commands must be executed from bigsql user ID (or ID with equivalent authority)
-- valid IDs must exist for user1 and user 2 accounts
-- in this lab, user1 is a manager and user2 is a staff member
--
-- first, create the roles
CREATE ROLE manager;

CREATE ROLE staff;
```

__3. Grant SELECT (read) access to the table to users:

```
-- grant read access to appropriate users
GRANT SELECT ON biadmin.sls_sales_fact TO USER user1;
GRANT SELECT ON biadmin.sls_sales_fact to USER user2;
grant select on biadmin.sls_sales_fact to user biadmin;
```

__4. Issue GRANT statements that assign appropriate roles to desired users.

```
-- assign users appropriate roles
GRANT ROLE MANAGER TO USER user1;
GRANT ROLE STAFF TO USER user2;
GRANT ROLE MANAGER TO USER biadmin;
```

__5. Create a column access control rule. Specifically, create a mask called PROFIT_MASK for the GROSS_PROFIT column of the BIADMIN.SLS_SALES_FACT table that will display a value of 0.0 to any user who queries this column that is not a MANAGER.

```
-- create a mask for the gross_profit column that allows only managers to see values
for this column
CREATE MASK PROFIT_MASK ON
biadmin.sls_sales_fact
FOR COLUMN gross_profit
RETURN
CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'MANAGER') = 1
THEN gross_profit
ELSE 0.0
END
ENABLE;
```

__6. Grant SECADM authority to the biadmin user ID.

```
grant secadm on database to user biadmin;
```

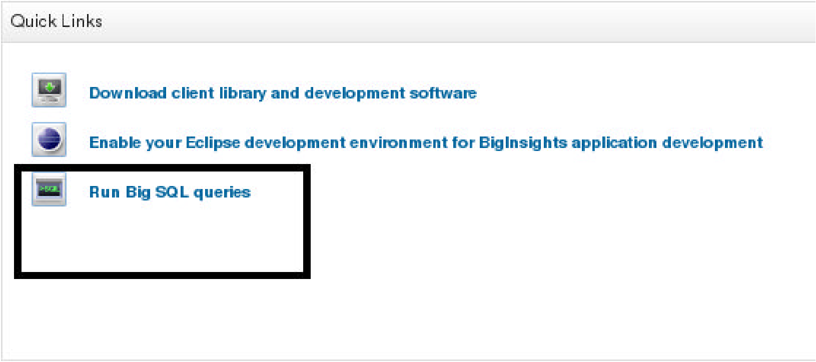
__7. Change your database connection so that you are connected to your bigsql database as biadmin.

__8. While connected to the bigsql database as biadmin, issue the following ALTER TABLE statement to activate the column based access control restriction.

```
-- Activate column access control.
-- (Prior to executing this statement, biadmin must have SECADM authority.)
-- Connect to database as biadmin and activate access control.
ALTER TABLE sls_sales_fact ACTIVATE COLUMN ACCESS CONTROL;
```

__9. Now you're ready to test the results of your work by querying the biadmin.sls_sales_fact table using different user accounts. It's easy to do this through the Web console. First, log into the Web console as user1 (password = passw0rd).

__10. On the Welcome page, click the Run Big SQL queries link in the Quick Links pane.


Quick Links

Download client library and development software

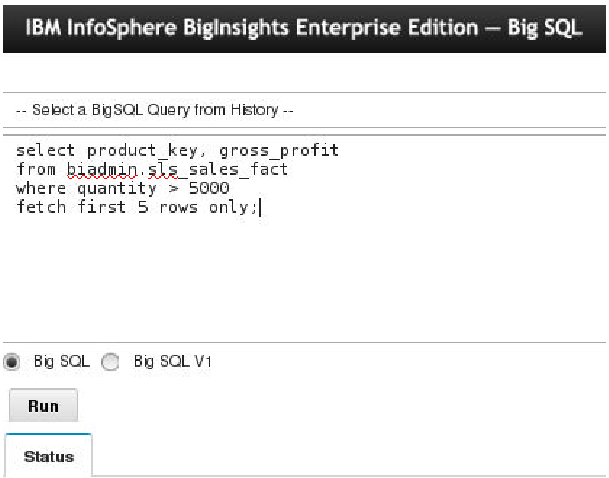
Enable your Eclipse development environment for BigInsights application development

Run Big SQL queries

A new tab will appear.

__11. Enter the following query into interface, verify that the Big SQL button at left is on, and click Run.

```
select product_key, gross_profit
from biadmin.sls_sales_fact
where quantity > 5000 fetch first 5 rows only;
```

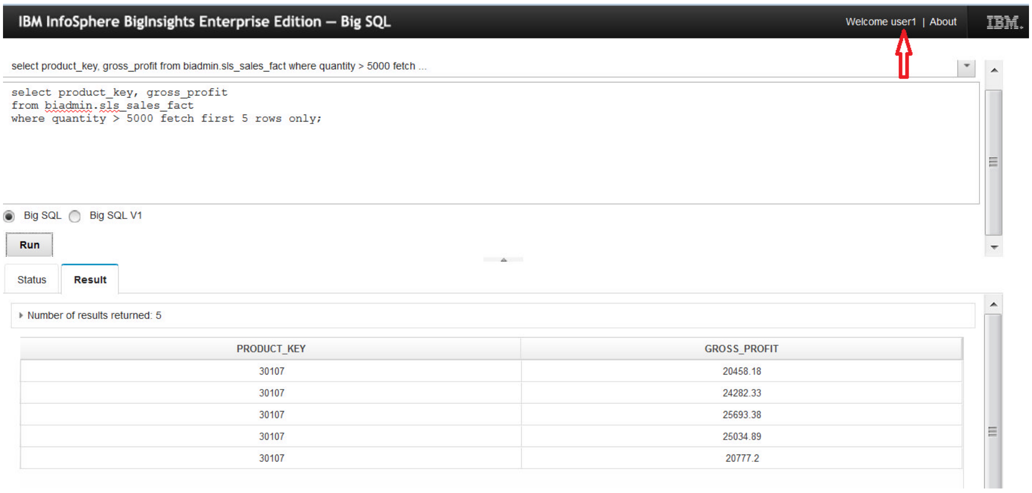

IBM InfoSphere BigInsights Enterprise Edition – Big SQL

-- Select a BigSQL Query from History --

```
select product_key, gross_profit
from biadmin.sls_sales_fact
where quantity > 5000
fetch first 5 rows only;
```

● Big SQL ○ Big SQL V1

__12. Inspect the results, and note that various data values appear in the GROSS_PROFIT column. This is what you should expect, because USER1 is a MANAGER, and your column mask rule allows MANAGERs to see the data values for this column.



IBM InfoSphere BigInsights Enterprise Edition — Big SQL
Welcome user1 | About
IBM.
select product_key, gross_profit from biadmin.sls_sales_fact where quantity > 5000 fetch ...

```
select product_key, gross_profit
from biadmin.sls_sales_fact
where quantity > 5000 fetch first 5 rows only;
```

● Big SQL ○ Big SQL V1
Run
Status
Result

Number of results returned: 5

PRODUCT_KEY	GROSS_PROFIT
30107	20458.18
30107	24282.33
30107	25693.38
30107	25034.89
30107	20777.2

__13. Close the Big SQL query tab.

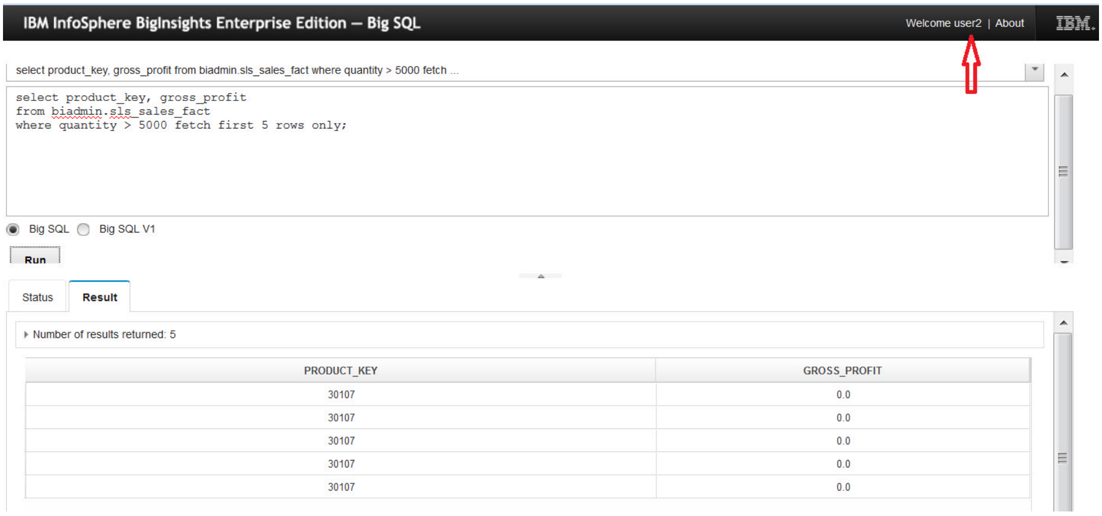
__14. Log off of the Web console as user1, and log in again as user2 (password = passw0rd).

__15. Click on the Run Big SQL query link in the Welcome page.

__16. Issue the same query.

```
select product_key, gross_profit
from biadmin.sls_sales_fact
where quantity > 5000 fetch first 5 rows only;
```

__17. Inspect the results. Note that the GROSS_PROFIT values are masked (appearing as 0.0). Again, this is what you should expect, because USER2 has a STAFF role. Any users who aren't MANAGERs are not allowed to see values for this column.



IBM InfoSphere BigInsights Enterprise Edition — Big SQL
Welcome user2 | About
IBM.
select product_key, gross_profit from biadmin.sls_sales_fact where quantity > 5000 fetch ...

```
select product_key, gross_profit
from biadmin.sls_sales_fact
where quantity > 5000 fetch first 5 rows only;
```

● Big SQL ○ Big SQL V1
Run
Status
Result

Number of results returned: 5

PRODUCT_KEY	GROSS_PROFIT
30107	0.0
30107	0.0
30107	0.0
30107	0.0
30107	0.0

__18. Optionally, while connected with the USER2 account, see what happens when you apply a function or calculation to the GROSS_PROFIT column in a query, such as one or both of these:

```
select product_key, gross_profit+10 as new from biadmin.sls_sales_fact where
quantity > 5000 fetch first 5 rows only;
```

```
select avg(gross_profit) as avg_gp from biadmin.sls_sales_fact where quantity >
5000;
```

As you might expect, the underlying results for values based on GROSS_PROFIT are masked from you. The first query results a value of 10 for the second column in the result set, as this is the result of adding 0.0 (the masked value for GROSS_PROFIT) with 10. The second query returns 0 (the average of 0 across all qualifying rows).

__19. Optionally, return to JSqsh or Eclipse. Connect to your bigsql database as biadmin, and deactivate the column access restriction.

```
ALTER TABLE sls_sales_fact DEACTIVATE COLUMN ACCESS CONTROL;
```

The effort required to implement row-based access control rules is similar. Let's explore a row-based scenario now using the `biadmin.mrk_promotion_fact` table you created and populated in an earlier lab. After implementing this example, you'll see that SELECT statements issued by `user1` will only return rows related to a specific retailer key.

__20. Create a new role named CONSULT.

```
-- row based access control example
-- restrict consultants (CONSULT role users) to accessing only rows
-- for retailer key 7166
--
-- commands must be executed from bigsql user ID (or ID with equivalent authority)
-- a valid ID must exist for user1
-- in this lab, user1 is a consultant
--
-- first, create the roles
CREATE ROLE CONSULT;
```

__21. Grant SELECT (read) access to the table to user1 and user2:

```
-- grant read access to appropriate user(s)
GRANT SELECT ON biadmin.mrk_promotion_fact TO USER user1;
GRANT SELECT ON biadmin.mrk_promotion_fact TO USER user2;
```

__22. Issue GRANT statements that assign appropriate roles to desired users. In this case, assign user1 CONSULT role. Do not assign any role to user2.

```
-- assign CONSULT role to user1
```

```
GRANT ROLE CONSULT TO USER user1;
```

__23. Create a row access control rule. Specifically, restrict read operations on `biadmin.mrk_promotion_fact` to users with the CONSULT role. Furthermore, allow such users to only see rows in which RETAILER_KEY column values are 7166.

```
-- create persmission for accessing data related to specific retailer
CREATE PERMISSION RETAILER_7166
ON biadmin.mrk_promotion_fact
FOR ROWS WHERE(VERIFY_ROLE_FOR_USER(SESSION_USER,'CONSULT') = 1
AND
retailer_key = 7166)
ENFORCED FOR ALL ACCESS
ENABLE;
```

__24. Grant SECADM authority to the biadmin user ID.

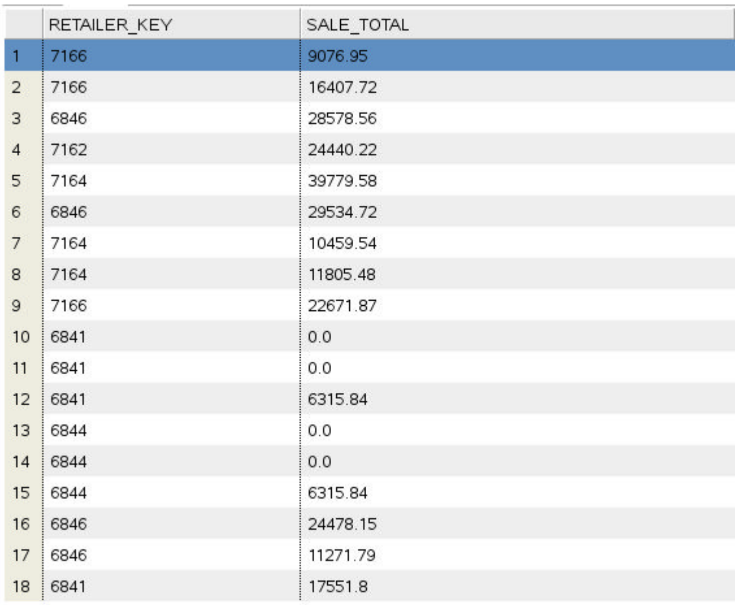
```
-- This statement is redundant because you already granted
-- SECADM authority to biadmin in the column-based access control exercise.
-- However, it is included here for clarity.
-- Reissuing the statement will not cause an error.
grant secadm on database to user biadmin;
```

__25. Change your database connection so that you are connected to your bigsql database as biadmin.

__26. Issue the following query:

```
select retailer_key, sale_total

from biadmin.mrk_promotion_fact

where rtl_country_key = 90010

fetch first 100 rows only;
```

__27. Note that the results include rows for with RETAILER_KEY values other than 7166.



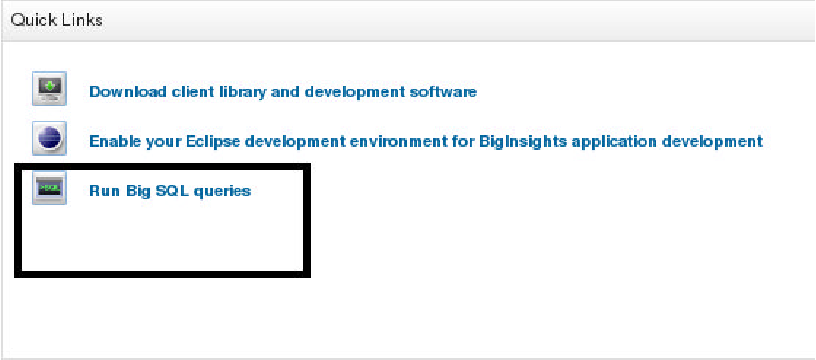
	RETAILER_KEY	SALE_TOTAL
1	7166	9076.95
2	7166	16407.72
3	6846	28578.56
4	7162	24440.22
5	7164	39779.58
6	6846	29534.72
7	7164	10459.54
8	7164	11805.48
9	7166	22671.87
10	6841	0.0
11	6841	0.0
12	6841	6315.84
13	6844	0.0
14	6844	0.0
15	6844	6315.84
16	6846	24478.15
17	6846	11271.79
18	6841	17551.8

__28. While connected to the bigsql database as biadmin, issue the following ALTER TABLE statement to activate the row based access control restriction.

```
-- activate row access control while logged in as biadmin.
-- prior to executing this statement, biadmin must have been granted SECADM
-- authority for the database
ALTER TABLE mrk_promotion_fact ACTIVATE ROW ACCESS CONTROL;
```

__29. Now you're ready to test the results of your work by querying the table. It's easy to do this through the Web console. First, log into the Web console as user1 (password = passw0rd).

__30. On the Welcome page, click the Run Big SQL queries link in the Quick Links pane.



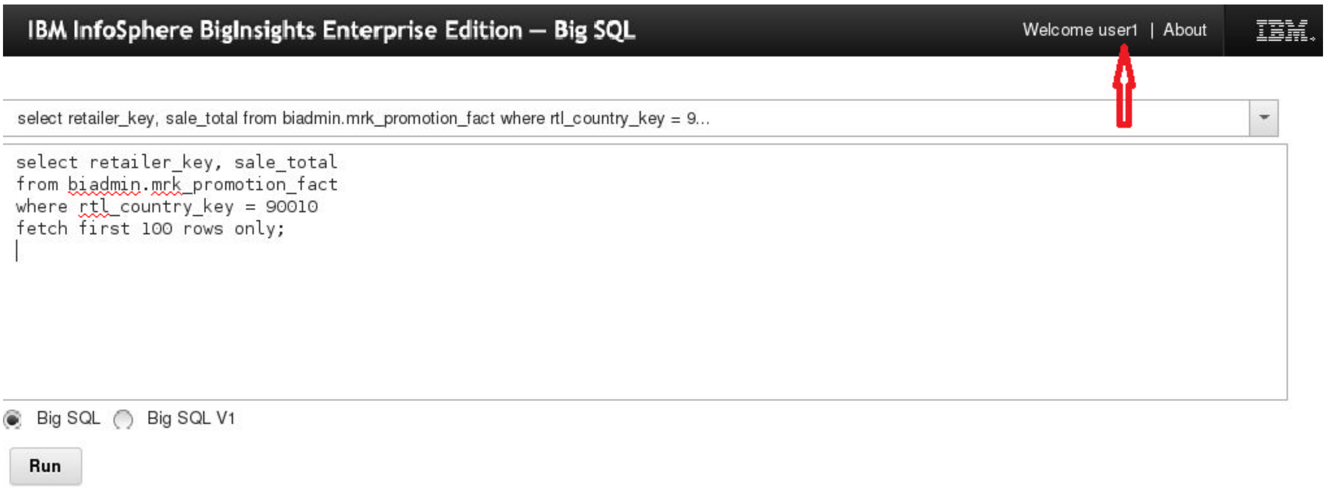
Quick Links
Download client library and development software
Enable your Eclipse development environment for BigInsights application development
Run Big SQL queries

A new tab will appear.

__31. Enter the same query you just entered as biadmin before you activated row access control on the table. Verify that the Big SQL button at left is on, and click Run.

```
select retailer_key, sale_total
```

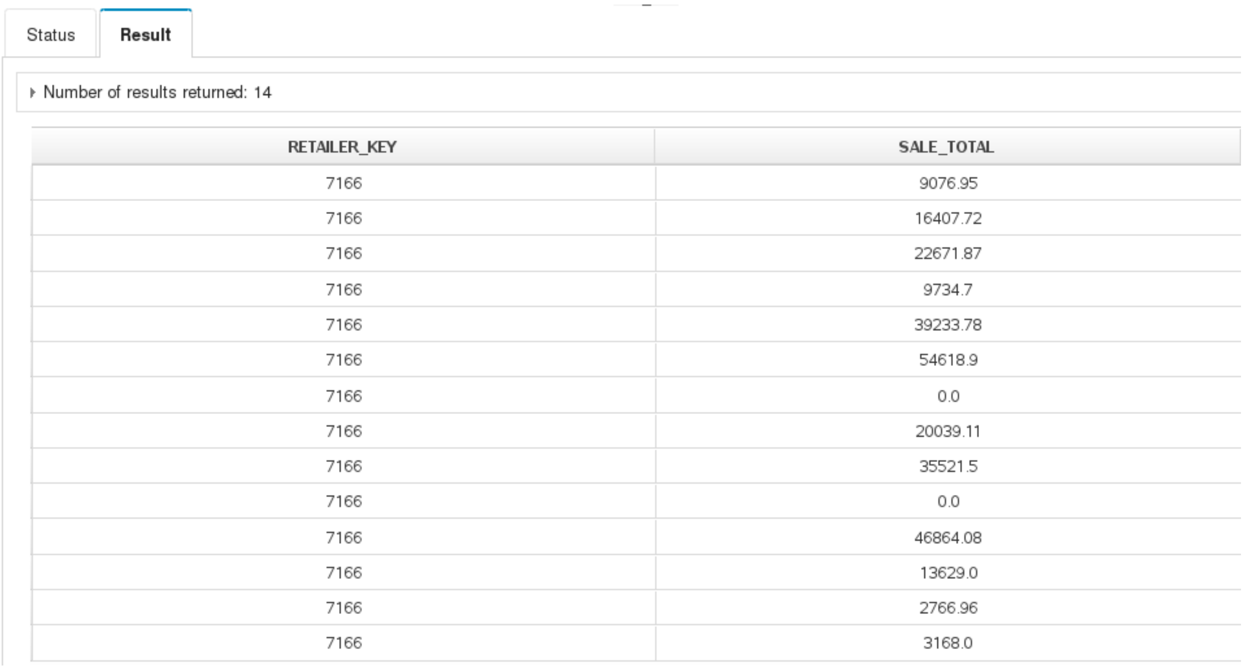
```
from biadmin.mrk_promotion_fact

where rtl_country_key = 90010

fetch first 100 rows only;
```



select retailer_key, sale_total from biadmin.mrk_promotion_fact where rtl_country_key = 9...

```
select retailer_key, sale_total
from biadmin.mrk_promotion_fact
where rtl_country_key = 90010
fetch first 100 rows only;
```

__32. Inspect the results, and note that only rows for retailer 7166 appear. This is what you should expect, because USER1 is associated with the CONSULT row.



Number of results returned: 14

RETAILER_KEY	SALE_TOTAL
7166	9076.95
7166	16407.72
7166	22671.87
7166	9734.7
7166	39233.78
7166	54618.9
7166	0.0
7166	20039.11
7166	35521.5
7166	0.0
7166	46864.08
7166	13629.0
7166	2766.96
7166	3168.0

__33. Close the Big SQL query tab.

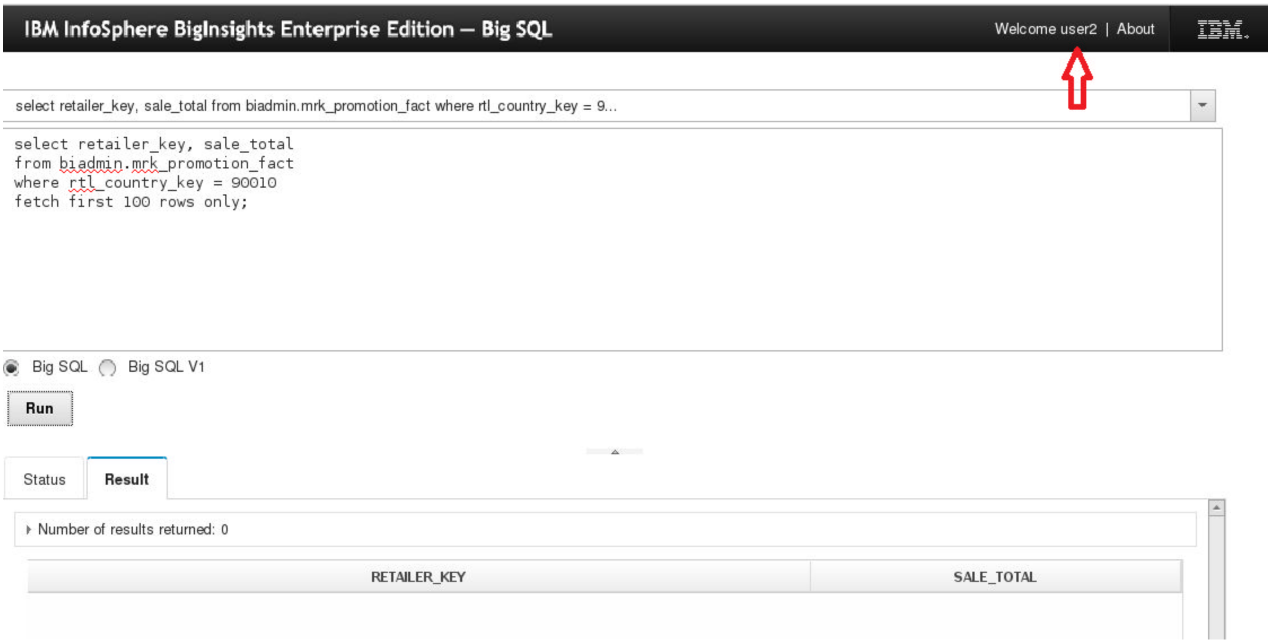
__34. Log off of the Web console as user1, and log in again as user2 (password = passw0rd) or as biadmin.

__35. Click on the Run Big SQL query link in the Welcome page.

__36. Issue the same query.

```
select retailer_key, sale_total

from biadmin.mrk_promotion_fact

where rtl_country_key = 90010

fetch first 100 rows only;
```

__37. Inspect the results. Note that the query runs successfully but returns no rows. Why? The row access control mechanism you implemented specified that only users of the CONSULT row are permitted to see data from the data and that data would be restricted to rows related to a specific RETAILER_KEY value.


IBM InfoSphere BigInsights Enterprise Edition – Big SQL
Welcome user2 | About
select retailer_key, sale_total from biadmin.mrk_promotion_fact where rtl_country_key = 9...
● Big SQL ○ Big SQL V1
Run
Status
Result

```
select retailer_key, sale_total
from biadmin.mrk_promotion_fact
where rtl_country_key = 90010
fetch first 100 rows only;
```

Number of results returned: 0

RETAILER_KEY	SALE_TOTAL

__38. Optionally, return to JSqsh or Eclipse. Connect to your bigsql database as biadmin, and deactivate the column access restriction.

```
ALTER TABLE mrk_promotion_fact DEACTIVATE ROW ACCESS CONTROL;
```

__39. Optionally, log into the Web console again with any valid user ID (biadmin, user1, or user2). Issue the same query again, and note that the results contain rows related to a number of different RETAILER_KEY values.

Lab 8 Developing and executing SQL user-defined functions

Big SQL enables users to create their own SQL functions that can be invoked in queries. User-defined functions (UDFs) promote code re-use and reduce query complexity. They can be written to return a single (scalar) value or a result set (table). Programmers can write UDFs in SQL or any supported programming languages (such as Java and C). For simplicity, this lab focuses on SQL UDFs.

After you complete this lab, you will understand how to:

- Create scalar and table UDFs written in SQL
- Incorporate procedural logic in your UDFs
- Invoke UDFs in Big SQL queries

Allow 1 - 1.5 hours to complete this lab.

Please note that this lab discusses only some of the capabilities of Big SQL scalar and table UDFs. For an exhaustive list of all the capabilities, please see the BigInsights 3.0 knowledge center (http://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.welcome.doc/doc/welcome.html).

Prior to starting this lab, you must be familiar with how to use the Big SQL command line (JSqsh), and you must have created the sample GOSALESDW tables. If necessary, work through earlier lab exercises on these topics. For example, the JSqsh lab is available at: https://developer.ibm.com/hadoop/docs/tutorials/big-sql-hadoop-tutorial/big-sql-hadoop-lab-2-big-sql-command-line-interface/

Furthermore, creating and populating the GOSALESDW tables is covered in this lab: https://developer.ibm.com/hadoop/docs/tutorials/big-sql-hadoop-tutorial/big-sql-hadoop-lab-4-querying-structured-data/

This UDF lab was developed by Uttam Jain (uttam@us.ibm.com) with contributions from Cynthia M. Saracco. Please post questions or comments to the forum on Hadoop Dev at https://developer.ibm.com/hadoop/support/.

8.1. Understanding UDFs

Big SQL provides many built-in functions to perform common computations. An example is `dayname()`, which takes a date/timestamp and returns the corresponding day name, such as Friday.

Often, organizations need to perform some customized or complex operation on their data that's beyond the scope of any built-in-function. Big SQL allows users to embed their customized business logic inside a user-defined function (UDF) and write queries that call these UDFs.

As mentioned earlier, Big SQL supports two types of UDFs:

1. **Scalar UDF**: These functions take one or more values as input and return a single value as output. For example, a scalar UDF can take three values (price of an item, percent discount on that item, and percent sales tax) to compute the final price of that item.

2. **Table UDF**: These functions take one or more values as input and return a whole table as output. For example, a table UDF can take single value (department-id) as input and return a table of employees who work in that department. This result set could have multiple columns, such as employee-id, employee-first-name, employee-last-name.

Once created, UDFs can be incorporated into queries in a variety of ways, as you'll soon see.

In this lab, you will first set up your environment for UDF development and then explore how to create and invoke UDFs through various exercises.

Ready to get started?

8.2. Prepare JSqsh to create and execute UDFs

In this section, you will set up your JSqsh environment for UDF development.

__1. If necessary, launch JSqsh using the connection to your bigsql database. (This was covered in an earlier lab.)

```
$JSQSH_HOME/bin/jsqsh bigsql
```

__2. Reset the default SQL terminator character to "@":

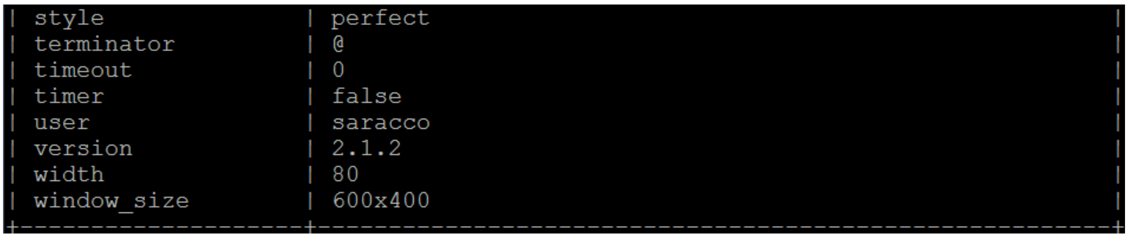
```
\set terminator = @;
```

Because some of the UDFs you will be developing involve multiple SQL statements, you must reset the JSqsh default termination character so that the semi-colon following each SQL statement in your UDF is not interpreted as the end of the CREATE FUNCTION statement for your UDF.

__3. Validate that the terminator was effectively reset:

```
\set @
```

__4. Inspect the output from the command (a subset of which is shown below), and verify that the terminator property is set to @.



```
| style              | perfect                                              |
| terminator         | @                                                    |
| timeout            | 0                                                    |
| timer              | false                                                |
| user               | saracco                                              |
| version            | 2.1.2                                                |
| width              | 80                                                   |
| window_size        | 600x400                                              |
+--------------------+------------------------------------------------------+
```

You're now ready to create your first Big SQL UDF.

8.3. Creating and executing a scalar UDF

In this section, you will create a scalar SQL UDF to compute final price of a particular item that was sold. Your UDF will require several input parameters:

• unit sale price:	Price of one item
• quantity:	Number of units of this item being sold in this transaction
• % discount:	Discount on the item (computed before tax)
• % sales-tax:	Sales tax (computed after discount)

As you might expect, your UDF will return a single value – the final price of the item.

After creating and registering the UDF, you will invoke it using some test values to ensure that it behaves correctly. Afterwards, you will invoke it in a query, passing in values from columns in a table as input to your function.

__1. Set your environment to use a schema that's different from your user ID. In this case, you want to create your UDFs in the GOSALESDW schema, so issue this command:

```
use gosalesdw@
```

Although you can create UDFs in your default schema (which is your user ID), it's quite common for programmers to create UDFs in a different schema, which is what you will do in this lab.

__2. Create a UDF named new_final_price:

```
CREATE OR REPLACE FUNCTION new_final_price
(
  quantity INTEGER,
  unit_sale_price DOUBLE,
  discount_in_percent DOUBLE,
  sales_tax_in_percent DOUBLE
)
RETURNS DOUBLE
LANGUAGE SQL
RETURN (quantity * unit_sale_price) * DOUBLE(1 - discount_in_percent / 100.0) * DOUBLE(1 +
sales_tax_in_percent / 100.0) @
```

__3. Review the logic of this function briefly. This first line creates the function, which is defined to take four input parameters. The RETURNS clause indicates that a single (scalar) value of type DOUBLE will be returned. The function's language is as SQL. Finally, the last two lines include the function's logic, which simply performs the necessary arithmetic operations to calculate the final sales price of an item.

__4. After creating the function, test it using some sample values. A simple way to do this is with the VALUES clause shown here:

```
VALUES gosalesdw.new_final_price (1, 10, 20, 8.75)@
```

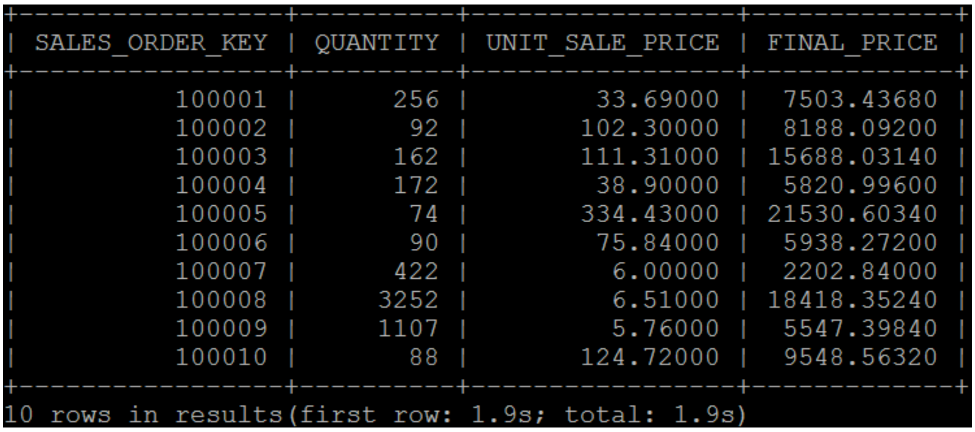
__5. Verify that result returned by your test case is

```
8.70000
```

__6. Next, use the UDF in a query to compute the final price for items listed in sales transactions in the SLS_SALES_FACT table. Note that this query uses values from two columns in the table as input for the quantity and unit price and two user-supplied values as input for the discount rate and sales tax rate.

```
SELECT sales_order_key, quantity, unit_sale_price, gosalesdw.new_final_price(quantity,
unit_sale_price, 20, 8.75) as final_price
FROM sls_sales_fact
ORDER BY sales_order_key
FETCH FIRST 10 ROWS ONLY@
```

__7. Inspect the results.

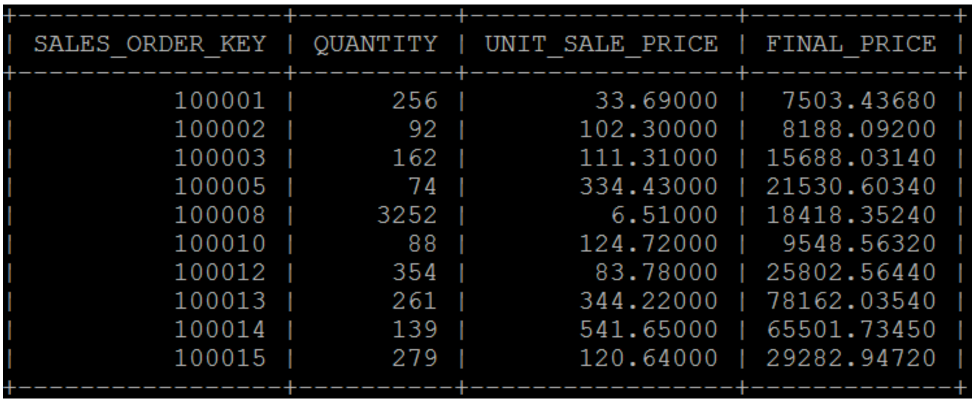


```
+-----------------+----------+-----------------+-------------+
| SALES_ORDER_KEY | QUANTITY | UNIT_SALE_PRICE | FINAL_PRICE |
+-----------------+----------+-----------------+-------------+
|          100001 |      256 |        33.69000 |  7503.43680 |
|          100002 |       92 |       102.30000 |  8188.09200 |
|          100003 |      162 |       111.31000 | 15688.03140 |
|          100004 |      172 |        38.90000 |  5820.99600 |
|          100005 |       74 |       334.43000 | 21530.60340 |
|          100006 |       90 |        75.84000 |  5938.27200 |
|          100007 |      422 |         6.00000 |  2202.84000 |
|          100008 |     3252 |         6.51000 | 18418.35240 |
|          100009 |     1107 |         5.76000 |  5547.39840 |
|          100010 |       88 |       124.72000 |  9548.56320 |
+-----------------+----------+-----------------+-------------+
10 rows in results(first row: 1.9s; total: 1.9s)
```

__8. Now invoke your UDF in the WHERE clause of a query. (Scalar UDFs can be included anywhere in a SQL statement that a scalar value is expected.) This query is similar to your previous query expect that it includes a WHERE clause to restrict the result set to items with a file price of greater than 7000.

```
-- scalar UDF can be used wherever a scalar value is expected,
-- for example in WHERE clause
SELECT sales_order_key, quantity, unit_sale_price,
gosalesdw.new_final_price(quantity, unit_sale_price, 20, 8.75) as final_price
FROM sls_sales_fact
WHERE gosalesdw.new_final_price(quantity, unit_sale_price, 20, 8.75) > 7000
ORDER BY sales_order_key
FETCH FIRST 10 ROWS ONLY@
```

__9. Note that your results no longer include rows with items priced at 7000 or below.



```
+-----------------+----------+----------------+-------------+
| SALES_ORDER_KEY | QUANTITY | UNIT_SALE_PRICE | FINAL_PRICE |
+-----------------+----------+----------------+-------------+
|          100001 |      256 |       33.69000 |  7503.43680 |
|          100002 |       92 |      102.30000 |  8188.09200 |
|          100003 |      162 |      111.31000 | 15688.03140 |
|          100005 |       74 |      334.43000 | 21530.60340 |
|          100008 |     3252 |        6.51000 | 18418.35240 |
|          100010 |       88 |      124.72000 |  9548.56320 |
|          100012 |      354 |       83.78000 | 25802.56440 |
|          100013 |      261 |      344.22000 | 78162.03540 |
|          100014 |      139 |      541.65000 | 65501.73450 |
|          100015 |      279 |      120.64000 | 29282.94720 |
+-----------------+----------+----------------+-------------+
```

8.4. Optional: Invoking UDFs without providing fully-qualified name

In the previous lab, you used the fully-qualified UDF name (GOSALESDW.NEW_FINAL_PRICE) in your VALUES or SELECT statements. (GOSALESDW is the schema name and NEW_FINAL_PRICE is the function name.)

A UDF with the same name and input parameters can be specified in more than one schema, so providing Big SQL with the fully qualified function name identifies the function you want to execute. With Big SQL, you can also specify a list of schemas in a special register called “CURRENT PATH” (also called “CURRENT FUNCTION PATH”). When Big SQL encounters an unqualified UDF (in which no schema name specified), it will look for the UDF in the schemas specified in CURRENT PATH.

In this lab, you’ll learn how to set the CURRENT PATH and invoke your function without specifying a schema name.

__1. To begin, determine the values of your current function path by issuing either of these two statements:

```
VALUES CURRENT PATH@

VALUES CURRENT FUNCTION PATH@
```

__2. Verify that the results are similar to this:

```
"SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","BIADMIN"
```

__3. Add the GOSALESDW schema to the current path:

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, "GOSALESDW"@
```

__4. Inspect your function path setting again:

```
VALUES CURRENT FUNCTION PATH@
```

__5. Verify that the GOSALESDW schema is now in the path:

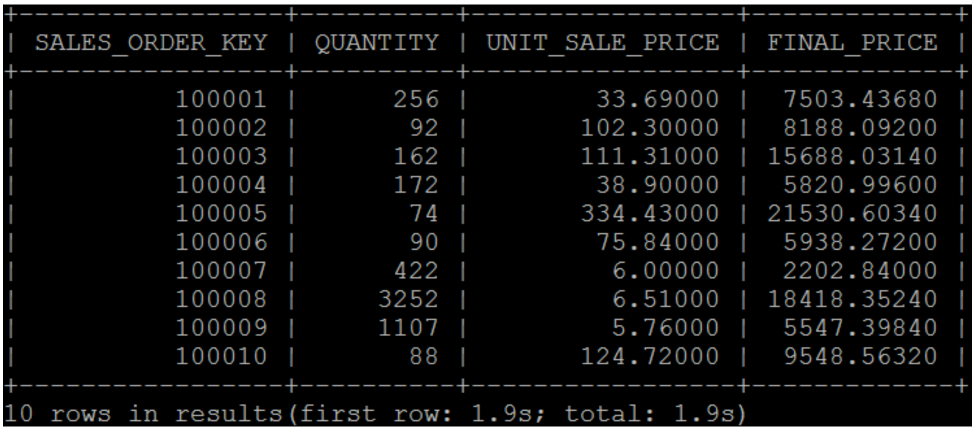
```
"SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","BIADMIN","GOSALESDW"
```

__6. Re-run the query you executed earlier, but this time remove the GOSALESDW schema from the function name with you invoke it:

```
SELECT sales_order_key, quantity, unit_sale_price,
new_final_price(quantity, unit_sale_price, 20, 8.75) as final_price
FROM sls_sales_fact
ORDER BY sales_order_key
FETCH FIRST 10 ROWS ONLY@
```

Note that Big SQL will automatically locate your UDF and successfully execute your query.

__7. Inspect the results.



```
+-----------------+----------+-----------------+-------------+
| SALES_ORDER_KEY | QUANTITY | UNIT_SALE_PRICE | FINAL_PRICE |
+-----------------+----------+-----------------+-------------+
|          100001 |      256 |        33.69000 |  7503.43680 |
|          100002 |       92 |       102.30000 |  8188.09200 |
|          100003 |      162 |       111.31000 | 15688.03140 |
|          100004 |      172 |        38.90000 |  5820.99600 |
|          100005 |       74 |       334.43000 | 21530.60340 |
|          100006 |       90 |        75.84000 |  5938.27200 |
|          100007 |      422 |         6.00000 |  2202.84000 |
|          100008 |     3252 |         6.51000 | 18418.35240 |
|          100009 |     1107 |         5.76000 |  5547.39840 |
|          100010 |       88 |       124.72000 |  9548.56320 |
+-----------------+----------+-----------------+-------------+
10 rows in results(first row: 1.9s; total: 1.9s)
```

8.5. Incorporating IF/ELSE statements

Quite often, you may find it useful to incorporate conditional logic in your UDFs. In this section, you will learn how to include IF/ELSE statements to calculate the final price of an item based on a varying discount rate. To keep your work simple, you will create a modified version of the previous UDF that includes the following logic:

- If the unit price is 0 to 10, use a discount rate of X%
- If the unit price is 10 to 100, use a discount rate of Y%
- If the unit price is greater than 100, use a discount rate of Z%

The three different discount rates (X, Y, and Z) are based on input parameters.

__8. Create a UDF named new_final_price_v2:

```
CREATE OR REPLACE FUNCTION new_final_price_v2
(
  quantity INTEGER,
  unit_sale_price DOUBLE,
  discount_in_percent_if_price_0_t0_10 DOUBLE,
  discount_in_percent_if_price_10_to_100 DOUBLE,
  discount_in_percent_if_price_greater_than_100 DOUBLE,
  sales_tax_in_percent DOUBLE
)
RETURNS DOUBLE
```

```
LANGUAGE SQL
BEGIN ATOMIC
  DECLARE final_price DOUBLE;
  SET final_price = -1;

  IF unit_sale_price <= 10
  THEN
    SET final_price = (quantity * unit_sale_price) * DOUBLE(1 -
discount_in_percent_if_price_0_t0_10 / 100.0) * DOUBLE(1 + sales_tax_in_percent / 100.0) ;
  ELSEIF unit_sale_price <= 100
  THEN
    SET final_price = (quantity * unit_sale_price) * DOUBLE(1 -
discount_in_percent_if_price_10_to_100 / 100.0) * DOUBLE(1 + sales_tax_in_percent / 100.0) ;
  ELSE
    SET final_price = (quantity * unit_sale_price) * DOUBLE(1 -
discount_in_percent_if_price_greater_than_100 / 100.0) * DOUBLE(1 + sales_tax_in_percent /
100.0) ;
  END IF;

  RETURN final_price;
END @
```

__9. Review the logic of this function briefly. As shown on lines 3 – 8, the function requires 6 input parameters. The first two represent the quantity ordered and the base unit price of each. The next three parameters specify different discount rates. The final input parameter represents the sales tax. The body of this function uses various conditional logic clauses (IF, THEN, ELSEIF, and ELSE) to calculate the final price of an item based on the appropriate discount rate and sales tax. Note that the unit price of the item determines the discount rate applied.

__10. Test you function's logic using sample data values:

```
VALUES gosalesdw.new_final_price_v2 (1, 100, 10, 20, 30, 8.75)@
```

__11. Verify that the result is

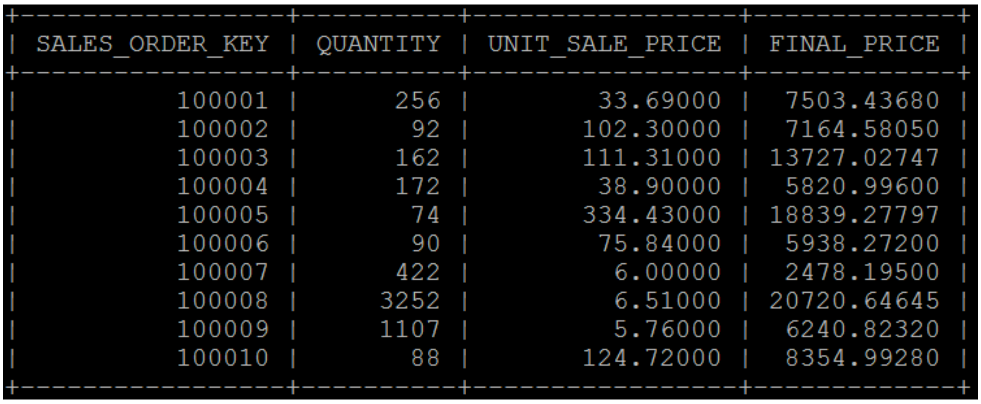
```
87.00000
```

If desired, review the function's logic to confirm that this is the correct value based on the input parameters. Note that 1 item was ordered at a price of $100, qualifying it for a 20% discount (to $80). Sales tax of 8.75% on $80 is $7, which results in a final item price of $87.

__12. Now invoke your UDF in a query to report the final sales prices for various items recorded in your SLS_SALES_FACT table:

```
SELECT sales_order_key, quantity, unit_sale_price,
gosalesdw.new_final_price_v2(quantity, unit_sale_price, 10,20,30, 8.75) as final_price
FROM shared.sls_sales_fact
ORDER BY sales_order_key
FETCH FIRST 10 ROWS ONLY @
```

__13. Inspect the results.



```
+-----------------+----------+-----------------+-------------+
| SALES_ORDER_KEY | QUANTITY | UNIT_SALE_PRICE | FINAL_PRICE |
+-----------------+----------+-----------------+-------------+
|          100001 |      256 |        33.69000 |  7503.43680 |
|          100002 |       92 |       102.30000 |  7164.58050 |
|          100003 |      162 |       111.31000 | 13727.02747 |
|          100004 |      172 |        38.90000 |  5820.99600 |
|          100005 |       74 |       334.43000 | 18839.27797 |
|          100006 |       90 |        75.84000 |  5938.27200 |
|          100007 |      422 |         6.00000 |  2478.19500 |
|          100008 |     3252 |         6.51000 | 20720.64645 |
|          100009 |     1107 |         5.76000 |  6240.82320 |
|          100010 |       88 |       124.72000 |  8354.99280 |
+-----------------+----------+-----------------+-------------+
```

8.6. Incorporating WHILE loops

Big SQL enables you to include loops in your scalar UDFs. In this section, you'll use a WHILE loop to create a mathematical function for factorials. As a reminder, the factorial of a non-negative integer N is the product of all positive integers less than or equal to N. In other words,

```
factorial(N) = N * (N-1) * (N-2) …....* 1
```

As an example,

factorial(5) = 5 * 4 * 3 * 2 * 1 = 120

__1. Create a scalar UDF named `factorial` that uses a WHILE loop to perform the necessary multiplication operations.

```
-- WHILE-DO loop in scalar UDF
-- This example is independent of gosalesdw tables
-- Given a number n (n >= 1), returns its factorial
-- as long as it is in INTEGER range.
--------------------------------
-- Create scalar UDF with WHILE-DO loop
CREATE OR REPLACE FUNCTION factorial(n INTEGER)
RETURNS INTEGER
LANGUAGE SQL
BEGIN ATOMIC
  DECLARE n2 INTEGER;
  DECLARE res INTEGER;
  SET res = n;
  SET n2 = n;

  loop1:
  WHILE (n2 >= 2)
  DO
    SET n2 = n2 - 1;
    SET res = res * n2;
  END WHILE  loop1;

  RETURN res;
```

```
END @
```

__2. Review the logic of this function. Note that two variables are declared and set to the value of the input parameter. The first variable (`res`) holds the result of the computation. Its value changes as the body of the WHILE loop is executed. The second variable (`n2`) controls the loop's execution and serves as part of the calculation of the factorial.

__3. Test your function supplying different input parameters:

```
-- The output of factorial(5) should be 120

VALUES gosalesdw.factorial(5)@



-- The output of factorial(7) should be 5040

VALUES gosalesdw.factorial(7)@
```

__4. Optionally, drop your function.

```
drop function gosalesdw.factorial@
```

Note that if you try to invoke your function again, you will receive an error message similar to this:

```
No authorized routine named "FACTORIAL" of type "FUNCTION" having compatible
arguments was found.. SQLCODE=-440, SQLSTATE=42884, DRIVER=3.68.61

[State: 56098][Code: -727]: An error occurred during implicit system action type
"2". Information returned for the error includes SQLCODE "-440", SQLSTATE "42884"
and message tokens "FACTORIAL|FUNCTION".. SQLCODE=-727, SQLSTATE=56098,
DRIVER=3.68.61
```

8.7. Incorporating FOR loops

As you might expect, Big SQL also supports FOR-DO loops in SQL-bodied UDFs. In this exercise, you'll create a function to calculate the sum of the top 5 sales for a given day.

__1. Create a scalar UDF named `sum_sale_total_top_5`. Note that this UDF references the SLS_SALES_FACT table that you created in an earlier lab in the `biadmin` schema because you were logged in as biadmin. If you created this table in a different schema, modify the table reference in the FROM clause of the FOR block as needed to match your environment.

```
-- FOR-DO loop and a SELECT statement inside scalar UDF
-- Given order_day_key, returns sum of sale_total for first 5 sales with given
order_day_key. Order by sale_total
--------------------------------
```

```
-- Create UDF with FOR-DO loop and a SELECT statement inside
CREATE OR REPLACE FUNCTION sum_sale_total_top_5(input_order_day_key INTEGER)
RETURNS DOUBLE
LANGUAGE SQL
READS SQL DATA
BEGIN ATOMIC
  DECLARE result DOUBLE;
  DECLARE counter INTEGER;
  SET result = 0;
  SET counter = 5;

  FOR v1 AS
    SELECT sale_total
    FROM biadmin.sls_sales_fact
    WHERE order_day_key = input_order_day_key
    ORDER BY  sale_total DESC

  DO
    IF counter > 0
    THEN
      SET result = result + sale_total;
      SET counter = counter - 1;
    END IF;

  END FOR;

  RETURN result;
END @
```

__2. Review the logic of this function. Note that the FOR loop begins by retrieving SALE_TOTAL values from the SLS_SALES_FACT table based on the order key day provided as input. These results are ordered, and the DO block uses a counter to control the number of times it will add a SALE_TOTAL value to the result. In this example, that will occur 5 times.

__3. Finally, use this UDF to compute the sum of the top 5 sales on a specific order day key (20040112).

```
-- The output of this function call should be 925973.09000
VALUES (gosalesdw.sum_sale_total_top_5(20040112)) @
```

8.8. Creating a table UDF

Now that you've created several scalar UDFs, it's time to explore how you can create a simple UDF that will return a result set. Such UDFs are called table UDFs because they can return multiple columns and multiple rows.

In this lab, you will create a table UDF that returns information about the items sold on a given day input by the user. The result set will include information about the sales order, the quantity of items, the pre-discounted sales price, and the final sales price (including tax and a discount). In doing so, your table UDF will call a scalar UDF you created previously: `new_final_price_v2`.

__1. Create a table UDF named `sales_summary`. Note that this UDF references the SLS_SALES_FACT table that you created in an earlier lab in the `biadmin` schema because you were logged in as biadmin. If you created this table in a different schema, modify the table reference in this UDF to match your environment.

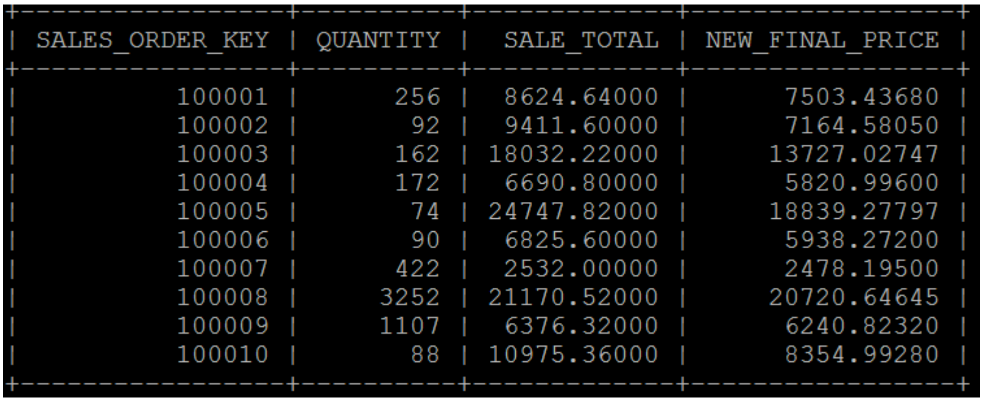
```
-- Table UDF
-- given an order_day_key, returns some desired fields and
-- new_final_price for that order_day_key
--------------------------------

-- Create a simple table UDF
CREATE OR REPLACE FUNCTION sales_summary(input_order_day_key INTEGER)
RETURNS TABLE(sales_order_key INTEGER, quantity INTEGER, sale_total DOUBLE, new_final_price
DOUBLE)
LANGUAGE SQL
READS SQL DATA
RETURN
  SELECT sales_order_key, quantity, sale_total, gosalesdw.new_final_price_v2(quantity,
unit_sale_price, 10,20,30, 8.75)
  FROM sls_sales_fact
  WHERE order_day_key = input_order_day_key
@
```

__2. Inspect the logic in this function. Note that it includes a READS SQL DATA clause (because the function SELECTs data from a table) and that the RETURNS clause specifies a TABLE with columns and data types. Towards the end of the function is the query that drives the result set that is returned. As mentioned earlier, this query invokes a scalar UDF that you created earlier.

__3. Invoke your table UDF in the FROM clause of a query, supplying an input parameter of `20040112` to your function for the order day key.

```
-- use it in the FROM clause
SELECT t1.*
FROM TABLE (gosalesdw.sales_summary(20040112)) AS t1
ORDER BY sales_order_key
FETCH FIRST 10 ROWS ONLY
@
```

__4. Inspect your output.



```
+-----------------+----------+-------------+-----------------+
| SALES_ORDER_KEY | QUANTITY |  SALE_TOTAL | NEW_FINAL_PRICE |
+-----------------+----------+-------------+-----------------+
|          100001 |      256 |  8624.64000 |      7503.43680 |
|          100002 |       92 |  9411.60000 |      7164.58050 |
|          100003 |      162 | 18032.22000 |     13727.02747 |
|          100004 |      172 |  6690.80000 |      5820.99600 |
|          100005 |       74 | 24747.82000 |     18839.27797 |
|          100006 |       90 |  6825.60000 |      5938.27200 |
|          100007 |      422 |  2532.00000 |      2478.19500 |
|          100008 |     3252 | 21170.52000 |     20720.64645 |
|          100009 |     1107 |  6376.32000 |      6240.82320 |
|          100010 |       88 | 10975.36000 |      8354.99280 |
+-----------------+----------+-------------+-----------------+
```

As you might imagine, the bodies of table UDFs aren't limited to queries. Indeed, you can write table UDFs that contain IF/ELSE, WHILE/DO, FOR-DO, and many more constructs. Consult the BigInsights Knowledge Center for details.

8.9. Optional: Overloading UDFs and dropping UDFs

As you saw in an earlier exercise, you can drop UDFs with the DROP FUNCTION statement. In addition, you can create multiple UDFs with the same name (even in the same schema) if their input parameters differ enough so that Big SQL can identify which should be called during a query. Such UDFs are said to be "overloaded". When working with overloaded UDFs, you must use the DROP SPECIFIC FUNCTION statement to properly identify which UDF bearing the same name should be dropped.

In this lab, you'll explore the concepts of overloading functions and dropping a specific function. To keep things simple and focused on the topics at hand, the UDFs will be trivial – they will simply increment a supplied INTEGER or DOUBLE value by 1.

__1. Create a scalar UDF that increments an INTEGER value.

```
-- Create a scalar UDF
CREATE FUNCTION increment_by_one(p1 INT)
RETURNS INT
LANGUAGE SQL
SPECIFIC increment_by_one_int
RETURN p1 + 1 @
```

Note that the SPECIFIC clause provides a unique name for the function that we can later reference it when we need to drop this function.

__2. Create a scalar UDF that increments a DOUBLE value.

```
-- Create another scalar UDF with same name (but different specific name)
CREATE FUNCTION increment_by_one(p1 DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
SPECIFIC increment_by_one_double
RETURN p1 + 1 @
```

__3. Attempt to drop the `increment_by_one` function without referencing the specific name you included in each function.

```
-- If we try to drop the function using DROP FUNCTION statement,
-- Big SQL will throw Error : SQLCODE=-476, SQLSTATE=42725, because
-- Big SQL needs to know which function should be dropped
DROP FUNCTION increment_by_one@
```

Note that this statement will fail because Big SQL isn't certain which of the two `increment_by_one` functions you intended to drop.

__4. Drop the function that requires an INTEGER as its input parameter. Reference the function's specific name in a DROP SPECIFIC FUNCTION statement.

```
-- User must drop using specific name
DROP SPECIFIC FUNCTION increment_by_one_int@
```

__5. Now drop the remaining `increment_by_one` function. Since we only have 1 function by this name in this schema, we can issue a simple DROP FUNCTION statement:

```
-- Now we have only one function with this name, so we can use
-- simple DROP FUNCTION statement.
DROP FUNCTION increment_by_one@
```

What if you didn't include a SPECIFIC clause (i.e., a specific name) in your UDF definition? Big SQL will explicitly provide one, and you can query the system catalog tables to identify it. Let's explore that scenario.

__6. Create a simple scalar UDF again.

```
-- Create a UDF
CREATE FUNCTION increment_by_one(p1 INT)
RETURNS INT
LANGUAGE SQL
RETURN p1 + 1 @
```

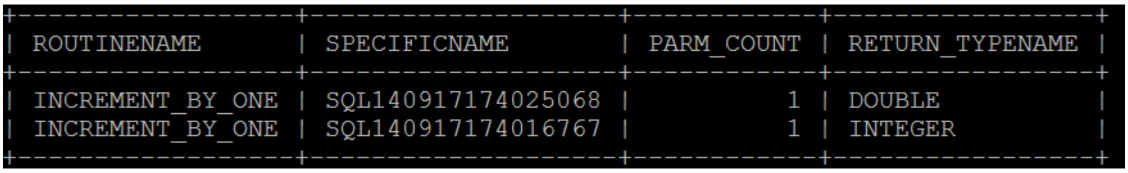
__7. Create another scalar UDF with the same name (but different input parameter)

```
-- Create another scalar UDF with same name (but different input parm)
CREATE FUNCTION increment_by_one(p1 DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
RETURN p1 + 1 @
```

__8. Query the Big SQL catalog for specific names for these functions:

```
-- Query catalog for specific name:
SELECT ROUTINENAME, SPECIFICNAME, PARM_COUNT, RETURN_TYPENAME
FROM SYSCAT.ROUTINES
WHERE  ROUTINESCHEMA = 'GOSALESDW' AND ROUTINENAME = 'INCREMENT_BY_ONE' @
```

__9. Inspect the output, noting the different names assigned to your functions. (Your output may vary from that shown below.)



```
+------------------+--------------------+------------+-----------------+
| ROUTINENAME      | SPECIFICNAME       | PARM_COUNT | RETURN_TYPENAME |
+------------------+--------------------+------------+-----------------+
| INCREMENT_BY_ONE | SQL140917174025068 |          1 | DOUBLE          |
| INCREMENT_BY_ONE | SQL140917174016767 |          1 | INTEGER         |
+------------------+--------------------+------------+-----------------+
```

__10. If desired, drop each of these UDFs. Remember that you will need to reference the specific name of the first UDF that you drop when you execute the DROP SPECIFIC FUNCTION statement.

Lab 9 Exploring Big SQL LOAD and Hadoop Commands

BigInsights offers a LOAD command for populating Big SQL tables with data. The LOAD command can read data from files or directly from specific relational DBMSs and import this data into a previously-defined Big SQL table.

This lab introduces you to the LOAD command and explores several aspects of its syntax. Examples that rely on RDBMS access are based on IBM DB2 for Linux, Unix, and Windows (LUW). You will need access to a DB2 server to complete those exercises. If necessary, download and install a free copy of DB2 Express-C. (The full URL is http://www-01.ibm.com/software/data/db2/express-c/index.html). Alternatively, consult with your instructor to determine if a DB2 server has been made available for your use with this lab.

An alternative approach to LOAD is to copy a data file directly to the Hadoop filesystem directory defined as the LOCATION for a Big SQL table.

You should be familiar with BigInsights V3.0 and Big SQL before beginning this lab. In particular, you should be able to create Big SQL tables, issue Big SQL commands and queries, and inspect the results of your work.

After completing this hands-on lab, you'll be able to:

- Load data into a Big SQL table from a comma-delimited file stored in your local file system.
- Load data into a Big SQL table directly from an RDBMS server (in our examples, a DB2 LUW server).
- Track records rejected by a LOAD operation.
- Use Hadoop commands to copy data to a table's Hadoop path and explore the contents of that directory, using Hadoop commands and from the Web Console.

Prior to starting this lab, you must have completed at least one of the prior labs on JSqsh or Eclipse. In particular, you must be familiar with how to execute queries in your target development platform (JSqsh or Eclipse), and you must have established a connection to your Big SQL 3.0 database. Most screen captures shown in this lab are based on Eclipse.

In addition, to complete the lab exercises that involve direct RDBMS connectivity, you will need access to a DB2 LUW server.

Allow 1 to 1.5 hours to complete this lab.

9.1. LOADing data into Big SQL Tables from a local file

In this section, you'll learn how to load data from a delimited file into a Big SQL table that uses Hadoop as its underlying storage mechanism. Verify that you have access to the sample file named `db2export_media.del` before completing this section. Examples in this lab presume that you have uploaded this file to the `/opt/ibm/biginsights/bigsql/samples/data/db2export` directory.

As background, this file was created by running the DB2 EXPORT facility using default values. You'll be loading data from this file into a Big SQL table.

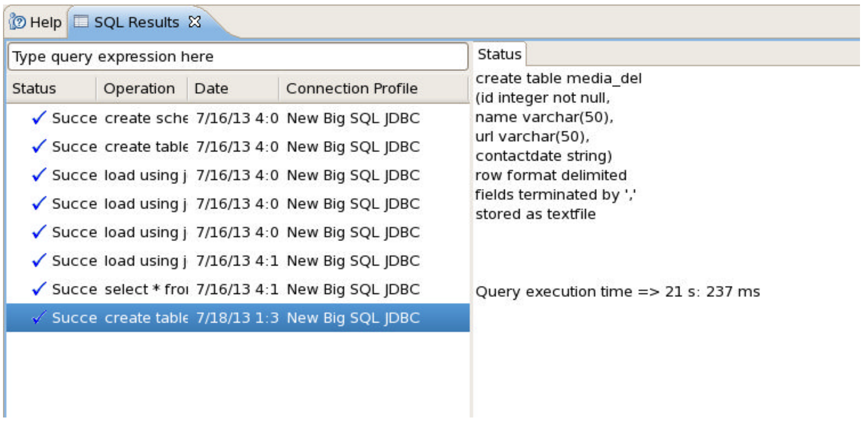
__1. Execute the following statement to create an appropriate Big SQL table for the sample data:

```
create hadoop table media_del
(id integer not null,
name varchar(50),
url varchar(50),
contactdate string)
row format delimited
fields terminated by ','
stored as textfile;
```

About this CREATE TABLE statement

The NOT NULL clause for the ID column is advisory only – it is not enforced by Big SQL or LOAD. In addition, the CONTACTDATE column is defined here as a String type because the input values aren't in ISO-compliant TIMESTAMP format. The final 3 lines of this statement reflect the fact that our input file is in a comma-delimited text format.

__2. Verify that the operation completed successfully.



Status	Operation	Date	Connection Profile
✓ Succe	create sche	7/16/13 4:0	New Big SQL JDBC
✓ Succe	create table	7/16/13 4:0	New Big SQL JDBC
✓ Succe	load using j	7/16/13 4:0	New Big SQL JDBC
✓ Succe	load using j	7/16/13 4:0	New Big SQL JDBC
✓ Succe	load using j	7/16/13 4:0	New Big SQL JDBC
✓ Succe	load using j	7/16/13 4:1	New Big SQL JDBC
✓ Succe	select * froı	7/16/13 4:1	New Big SQL JDBC
✓ Succe	create table	7/18/13 1:3	New Big SQL JDBC

```
create table media_del
(id integer not null,
name varchar(50),
url varchar(50),
contactdate string)
row format delimited
fields terminated by ','
stored as textfile


Query execution time => 21 s: 237 ms
```

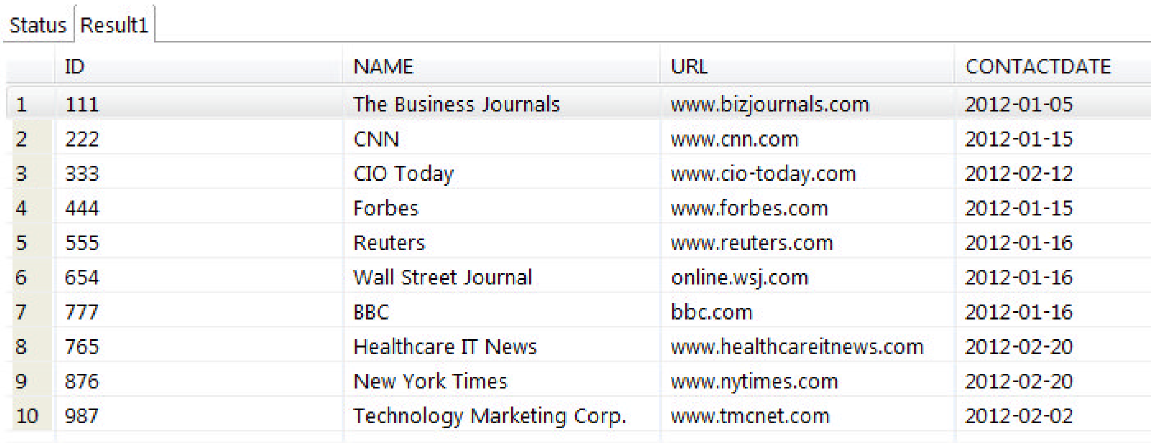
__3. Load data from the `db2export_media.del` file into the table. Adjust the file path specification to match where you have the db2export_media.del file stored in your local file system.

```
load hadoop
using file url 'sftp://biadmin:biadmin@bivm:22/path/to/file/db2export_media.del'
INTO TABLE MEDIA_DEL overwrite;
```

__4. Verify that the operation completed successfully.

__5. Finally, execute the following SELECT statement and inspect the results:

```
select * from media_del;
```



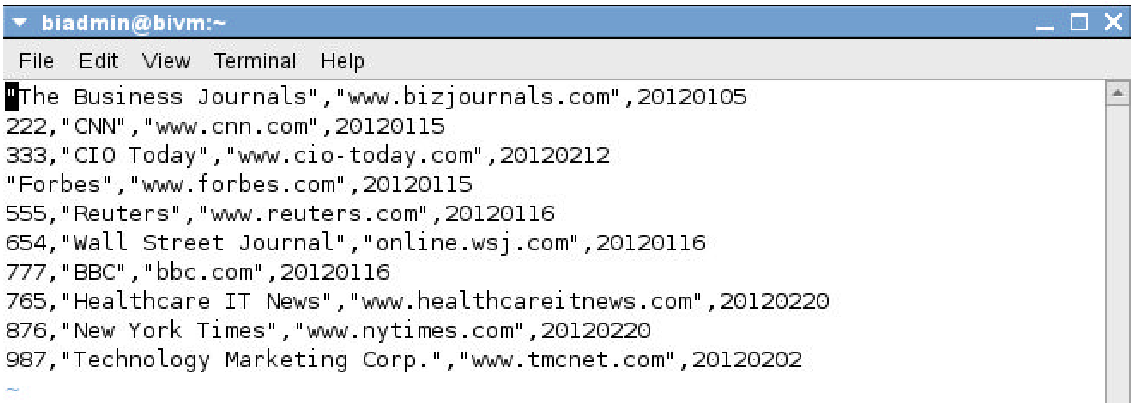
	ID	NAME	URL	CONTACTDATE
1	111	The Business Journals	www.bizjournals.com	2012-01-05
2	222	CNN	www.cnn.com	2012-01-15
3	333	CIO Today	www.cio-today.com	2012-02-12
4	444	Forbes	www.forbes.com	2012-01-15
5	555	Reuters	www.reuters.com	2012-01-16
6	654	Wall Street Journal	online.wsj.com	2012-01-16
7	777	BBC	bbc.com	2012-01-16
8	765	Healthcare IT News	www.healthcareitnews.com	2012-02-20
9	876	New York Times	www.nytimes.com	2012-02-20
10	987	Technology Marketing Corp.	www.tmcnet.com	2012-02-02

9.2. Tracking rejected records

The LOAD command enables you to direct any rejected records into a directory in your DFS, if desired. Doing so enables you to assess and correct any problems with the records. In this section, you will explore how to set and use the `rejected.records.dir` property to capture rejected records.

__1. Copy the db2_export.del file into a new file named db2export_media_error.del.

__2. Edit the new file using an editor of your choice. Delete the ID fields from the first and fourth lines of the file. Your file's contents should look like this:



```
"The Business Journals","www.bizjournals.com",20120105
222,"CNN","www.cnn.com",20120115
333,"CIO Today","www.cio-today.com",20120212
"Forbes","www.forbes.com",20120115
555,"Reuters","www.reuters.com",20120116
654,"Wall Street Journal","online.wsj.com",20120116
777,"BBC","bbc.com",20120116
765,"Healthcare IT News","www.healthcareitnews.com",20120220
876,"New York Times","www.nytimes.com",20120220
987,"Technology Marketing Corp.","www.tmcnet.com",20120202
~
```

Note that the records for `The Business Journals` (line 1) and `Forbes` (line 4). Because they lack ID field values, these rows will be rejected by the LOAD operation.

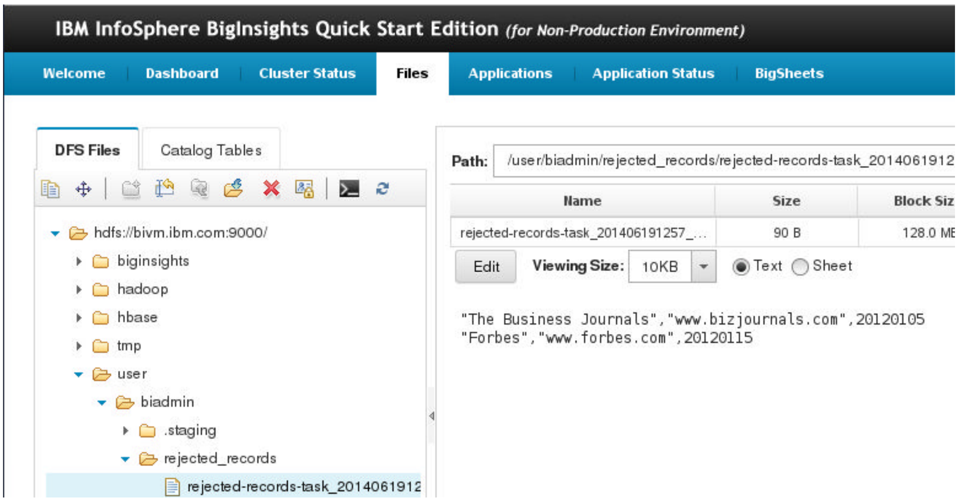
__3. Save the file and return to your SQL execution environment (e.g., JSqsh or Eclispe).

__4. Execute the following LOAD command so that the rejected records will be stored in an appropriate directory of your DFS. If needed, alter the path specifications to match your environment. For example, if you stored the source file (db2export_media_error.del) in a different directory, adjust the file url specification. If you are using a different BigInsights user ID than biadmin, change the DFS directory path in the final line of the statement below to match your user ID.

```
load hadoop
using file url
'sftp://biadmin:biadmin@bivm:22/opt/ibm/biginsights/bigsql/samples/data/db2export_medi
a_error.del'
into table media_del overwrite
with load properties
('rejected.records.dir' = '/user/biadmin/rejected_records');
```

__5. Run the Load statement.

__6. Open the BigInsights Web Console. From the Files tab, navigate to the directory you specified in the load statement. You will see the rejected records.



Path: /user/biadmin/rejected_records/rejected-records-task_2014061912

Name	Size	Block Siz
rejected-records-task_201406191257_...	90 B	128.0 MB

Viewing Size: 10KB

```
"The Business Journals","www.bizjournals.com",20120105
"Forbes","www.forbes.com",20120115
```

9.3. Preparing to load data directly from a relational DBMS

The LOAD command can dynamically read data from a table or view in a supported relational DBMS. Behind the scenes, LOAD uses JDBC to establish a DBMS connection and invokes open source Sqoop technology (included with BigInsights) to complete the data transfer.

This exercise, and several that follow, help you understand how you can use the LOAD command to dynamically retrieve data from a DB2 LUW database server. In addition to DB2 LUW, BigInsights supports loading data directly from Netezza, Teradata, Oracle, and other RDBMS sources into Big SQL tables.

You must have access to a DB2 server before attempting this lab. Furthermore, the server must contain a MEDIA table populated with data from the `db2export_media.del` file.

To begin, add the appropriate JDBC driver file(s) to Sqoop on BigInsights:

__1. Copy the DB2 JDBC .jar file (db2jcc4.jar) to the `$BIGINSIGHTS_HOME/sqoop/lib` directory. This can be found in the `../java` directory where your DB2 server is installed.

About the DB2 JDBC driver file

If you downloaded the current version of DB2 Express-C for this use with this exercise or have access to a DB2 LUW 10.5 server or later version, you can copy the db2jcc4.jar file from the `$BIGINSIGHTS_HOME/database/db2/java` directory into your Sqoop library.

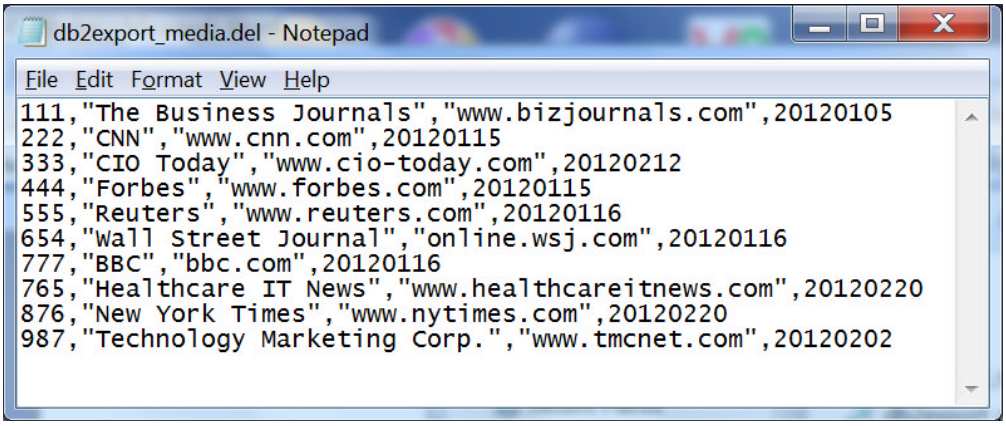
__2. If BigInsights is running, stop and restart the Big SQL service. (From a terminal window, issue these commands:

__a. `$BIGINSIGHTS_HOME/bin/stop.sh -bigsql`

__b. `$BIGINSIGHTS_HOME/bin/start.sh -bigsql`

Next, ensure your DB2 server has a MEDIA table defined and that the table contains data loaded from the `db2export_media.del` file required by this lab.

__3. Locate the `db2export_media.del` sample file and copy it to a location accessible to your DB2 server. Optionally, open the file using a text editor or operating system facility to inspect its contents, and close the file when you're done.



```
111,"The Business Journals","www.bizjournals.com",20120105
222,"CNN","www.cnn.com",20120115
333,"CIO Today","www.cio-today.com",20120212
444,"Forbes","www.forbes.com",20120115
555,"Reuters","www.reuters.com",20120116
654,"Wall Street Journal","online.wsj.com",20120116
777,"BBC","bbc.com",20120116
765,"Healthcare IT News","www.healthcareitnews.com",20120220
876,"New York Times","www.nytimes.com",20120220
987,"Technology Marketing Corp.","www.tmcnet.com",20120202
```

__4. From a DB2 command window, issue SQL statements to create a MEDIA table and populate it with data contained in the `db2export_media.del` file supplied with this lab. Alter the file specifications in the IMPORT statement as needed to match your environment.

```
CREATE TABLE MEDIA (
ID INTEGER,
NAME VARCHAR(50),
URL VARCHAR(50),
CONTACTDATE DATE);
```

```
IMPORT FROM "C:\Downloads\big data labs\LOAD\db2export_media.del" OF DEL
METHOD P (1, 2, 3, 4) MESSAGES "C:\Downloads\big data labs\LOAD\db2import-
msgs.txt" INSERT INTO TEST.MEDIA (ID, NAME, URL, CONTACTDATE);
```

__5. Verify that 10 rows were imported successfully into DB2.

SELECT * FROM MEDIA;

9.4. LOADing data directly from a relational DBMS table

Now you're ready to LOAD data directly from your relational DBMS table into a Big SQL table.

__1. In your Big SQL execution environment (JSqsh or Eclipse), create a table for the DB2 data.

```
create hadoop table media_db2table (
id integer not null,
name varchar(50),
url varchar(50),
contactdate varchar(30))
row format delimited
fields terminated by ','
stored as textfile;
```

About this CREATE TABLE statement

The NOT NULL clause for the ID column is advisory only – it is not enforced by Big SQL or LOAD. In addition, the CONTACTDATE column is defined here as a String type because the input values aren't in ISO-compliant TIMESTAMP format. The final 3 lines of this statement reflect the fact that our input file is in a comma-delimited text format.

__2. Run the following LOAD statement to your script.

```
load hadoop
using jdbc connection url 'jdbc:db2://your.server.com:portNum/sampledb'
with parameters (user='shared', password='shared123')
from table MEDIA
into table media_db2table overwrite
with load properties ('num.map.tasks' = 1);
```

About this LOAD USING JDBC connection . . . command

This form of the LOAD command establishes a live connection to a DB2 server and dynamically retrieves the contents of the specified table or view. Because we did not specify a "split column" property in this example, we must set the number of Map tasks for this LOAD operation to 1. Identifying a split column (such as ID) helps ensure that the LOAD operation is parallelized. You'll see an example of this shortly.

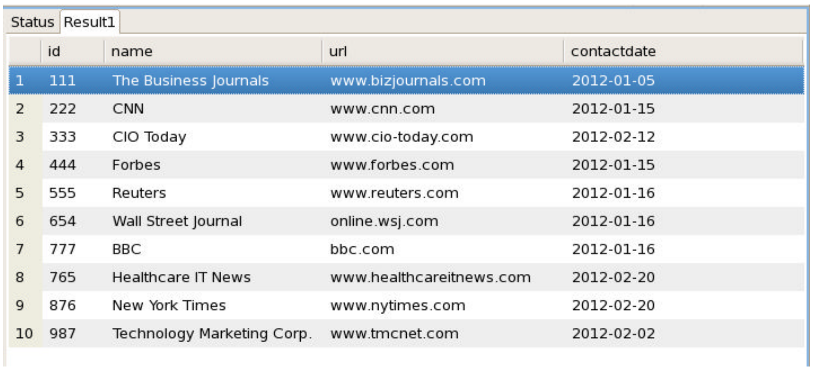
The LOAD command treats DB2 object names in a case-sensitive manner. Since DB2 folds names into upper case, the `from table` clause of this command must reference the DB2 table name in upper case.

__3. Verify that the operation completed successfully.

__4. Query the table:

```
select * from media_db2table;
```

__5. Inspect the results. Note that the name and URL values are not surrounded by double quotes. Because this form of the LOAD command uses Sqoop technology, VARCHAR and CHAR data are loaded without double quotes.



	id	name	url	contactdate
1	111	The Business Journals	www.bizjournals.com	2012-01-05
2	222	CNN	www.cnn.com	2012-01-15
3	333	CIO Today	www.cio-today.com	2012-02-12
4	444	Forbes	www.forbes.com	2012-01-15
5	555	Reuters	www.reuters.com	2012-01-16
6	654	Wall Street Journal	online.wsj.com	2012-01-16
7	777	BBC	bbc.com	2012-01-16
8	765	Healthcare IT News	www.healthcareitnews.com	2012-02-20
9	876	New York Times	www.nytimes.com	2012-02-20
10	987	Technology Marketing Corp.	www.tmcnet.com	2012-02-02

__6. Optionally, perform an equivalent LOAD operation using the default number of Map tasks (which is 4). For example, run the following LOAD command:

```
load hadoop
using jdbc connection url 'jdbc:db2:// your.server.com:portNum/sampledb'
with parameters (user='shared', password='shared123')
from table MEDIA
split column ID
into table media_db2table overwrite;
```

Note that this command uses the ID column of the DB2 table for splitting work across Map tasks.

9.5. LOADing data directly from a relational DBMS with SELECT

In some cases, it's more practical to issue a SELECT statement to identify the relational data you'd like to dynamically load into your Big SQL table. Big SQL's LOAD command supports such syntax. This exercise introduces you to using a relational query specification as part of your Big SQL LOAD command. In doing so, you can project and restrict data returned from your relational source table as well as join data from multiple tables.

You'll begin by performing a query-based load operation that's logically equivalent to the table-based load operation that you performed in the previous section.

__1. Run the CREATE TABLE statement and verify that the operation completes successfully.

```
create hadoop table media_db2select (
id integer not null,
name varchar(50),
url varchar(50),
contactdate varchar(30))
row format delimited
fields terminated by ','
stored as textfile;
```

__2. Execute the following LOAD statement:

```
load hadoop
using jdbc connection url 'jdbc:db2:// your.server.com:portNum/sampledb'
with parameters (user='shared', password='shared123')
from sql query
'select id, name, url, contactdate from media
where $CONDITIONS'
split column ID
into table media_db2select overwrite;
```

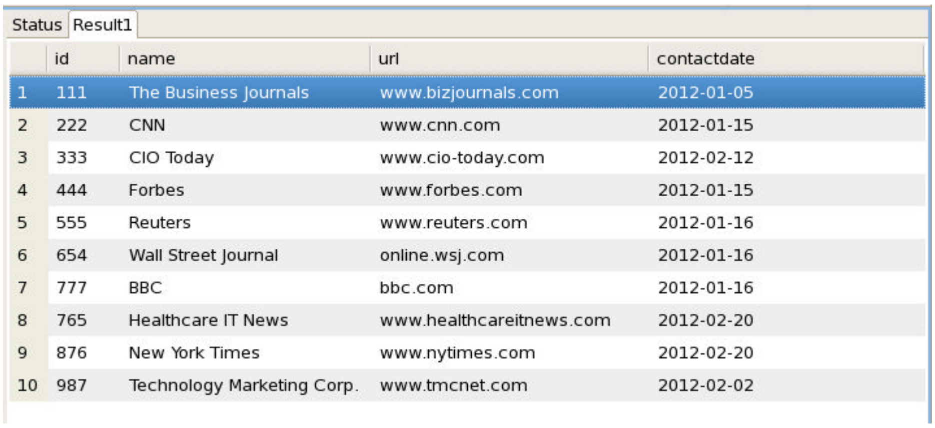
> **About this LOAD command**
>
> When using a SQL query as part of the LOAD command, you must include a WHERE clause that contains a $CONDITIONS marker as shown here. At runtime, this marker is replaced with a unique condition expression for each Map task.

__3. Verify that the operation completed successfully.

__4. Run the following SELECT statement.

```
select * from media_db2select;
```

__5. Inspect the results.



	id	name	url	contactdate
1	111	The Business Journals	www.bizjournals.com	2012-01-05
2	222	CNN	www.cnn.com	2012-01-15
3	333	CIO Today	www.cio-today.com	2012-02-12
4	444	Forbes	www.forbes.com	2012-01-15
5	555	Reuters	www.reuters.com	2012-01-16
6	654	Wall Street Journal	online.wsj.com	2012-01-16
7	777	BBC	bbc.com	2012-01-16
8	765	Healthcare IT News	www.healthcareitnews.com	2012-02-20
9	876	New York Times	www.nytimes.com	2012-02-20
10	987	Technology Marketing Corp.	www.tmcnet.com	2012-02-02

9.6. Exploring additional LOAD scenarios

You're now familiar with the basics of Big SQL's LOAD command. In this exercise, you'll explore a few additional scenarios that involve certain syntax variations.

You've already seen how to load data directly from a remote relational table. Let's refine that work a bit more by incorporating projection and restriction operations as part of your LOAD operation. Specifically, you will create a Big SQL table with columns for the ID and name of each media company contacted in January 2012.

__1. Run the following CREATE TABLE statement:

```
create hadoop table media_db2table_jan (
id integer not null,
name varchar(50)
)
row format delimited
fields terminated by ','
stored as textfile;
```

__2. Execute this LOAD statement:

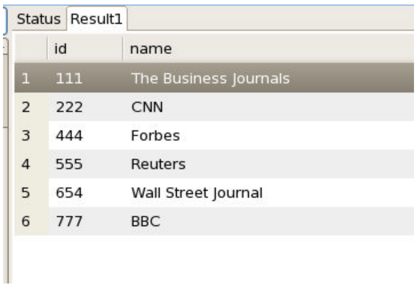
```
load hadoop
using jdbc connection url 'jdbc:db2:// your.server.com:portNum/sampledb'
with parameters (user='shared', password='shared123')
from table MEDIA columns (ID, NAME)
where 'CONTACTDATE < ''2012-02-01'''
into table media_db2table_jan overwrite
with load properties ('num.map.tasks' = 1);
```

__3. Verify that the operation completed successfully.

__4. Run the following SELECT statement:

```
select * from media_db2table_jan;
```

__5. Inspect the results and verify that the following 6 rows are present.



Status Result1

	id	name
1	111	The Business Journals
2	222	CNN
3	444	Forbes
4	555	Reuters
5	654	Wall Street Journal
6	777	BBC

Next, you'll execute a similar LOAD command using query-based syntax. In this example, you'll change the ordering of the ID and name columns so that the name column is defined first in the Big SQL table.

__6. Run the CREATE TABLE statement and verify that the operation completes successfully.

```
create hadoop table media_db2select_jan (
name varchar(50),
id integer not null
)
row format delimited
fields terminated by ','
stored as textfile;
```

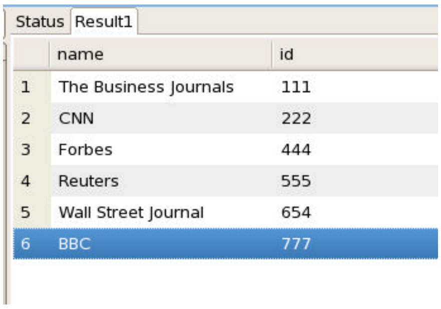
__7. Run the following LOAD statement:

```
load hadoop
using jdbc connection url 'jdbc:db2:// your.server.com:portNum/sampledb'
with parameters (user='shared', password='shared123')
from sql query
'select name, id from media
where $CONDITIONS and CONTACTDATE < ''2012-02-01'' '
split column ID
into table media_db2select_jan overwrite;
```

__8. Execute the following SELECT statement:

```
select * from media_db2select_jan;
```

__9. Inspect the results and verify that the following 6 rows are present.



Status Result1

	name	id
1	The Business Journals	111
2	CNN	222
3	Forbes	444
4	Reuters	555
5	Wall Street Journal	654
6	BBC	777

9.7. Using Hadoop commands to move data into a table

Since the data for Big SQL tables reside on the Hadoop Distributed File System (HDFS), it's possible to simply overlay the table definition onto an existing file or files, or likewise, copy a file into the path designated for the table. If you have a file which is already in the format designated by the CREATE TABLE statement, then this approach can save the time and processing of the LOAD command. (Examples of table features which would make LOAD a simpler approach are partitioning or Parquet format, which are beyond the scope of this document.)

__1. In the first exercise of this lab you created a table called `media_del`. Let's create the same table, but called `media_external`, and designate its location (path) to be different from the default.

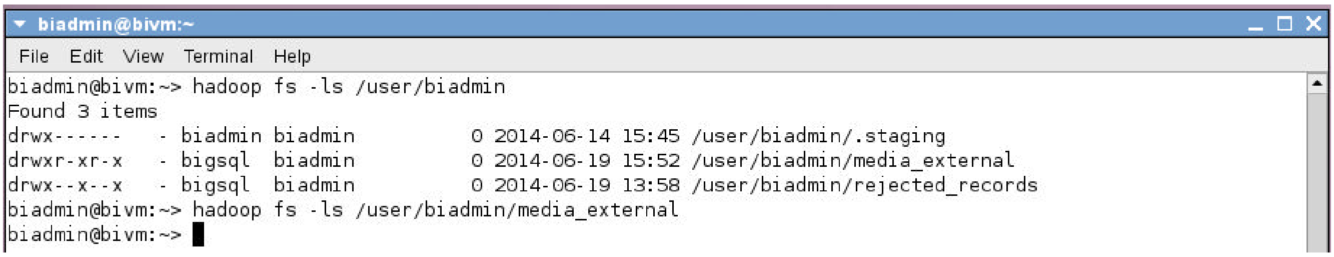
> The default Hadoop directory path (LOCATION) for Big SQL tables is at `/biginsights/hive/warehouse/<schema>.db/<table>`. In this beta release of the Technology Preview environment, this path is managed strictly by Big SQL (and Hive), meaning that its contents cannot be viewed and files cannot be copied there.
>
> Many code examples in this section use `userN` as part of the directory path specification. Replace this sample user with your user ID before executing the statements shown.

You can execute this CREATE TABLE statement in your Data Studio session.

```
create hadoop table media_external
(id integer not null,
name varchar(50),
url varchar(50),
contactdate string)
row format delimited fields terminated by ','
location '/user/biadmin/media_external';
```

__2. Use the Hadoop -ls command from the shell prompt to find the HDFS directory which has been created for this table. Note that the owner of the directory is bigsql and that it is currently empty.

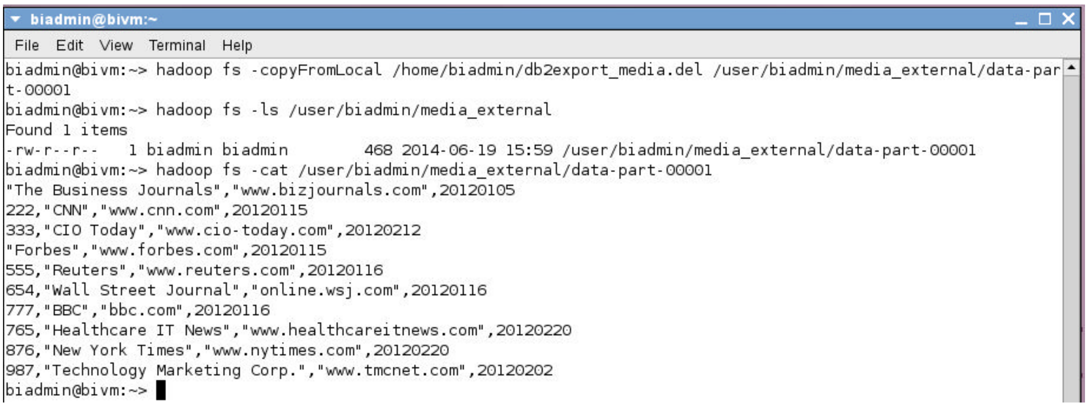
```
hadoop fs -ls /user/biadmin
hadoop fs -ls /user/biadmin/media_external
```



```
biadmin@bivm:~> hadoop fs -ls /user/biadmin
Found 3 items
drwx------   - biadmin biadmin          0 2014-06-14 15:45 /user/biadmin/.staging
drwxr-xr-x   - bigsql  biadmin          0 2014-06-19 15:52 /user/biadmin/media_external
drwx--x--x   - bigsql  biadmin          0 2014-06-19 13:58 /user/biadmin/rejected_records
biadmin@bivm:~> hadoop fs -ls /user/biadmin/media_external
biadmin@bivm:~> 
```

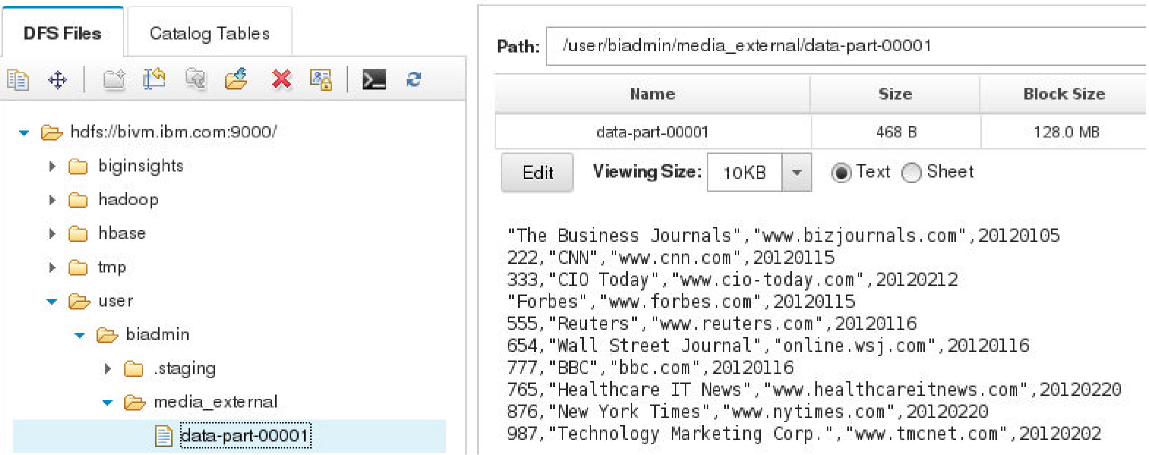
__3. Instead of LOADing data from the export file as we did before, use the Hadoop -copyFromLocal command to copy the file to the table's location (here, we are also giving the copied file a new name). Then verify that the directory is no longer empty and use the Hadoop -cat command to display the file contents. (Be aware that some commands below have wrapped onto two lines. Each hadoop command should be typed on a single line, and don't forget to replace userN.)

```
hadoop fs -copyFromLocal /path/to/file/db2export_media.del
/user/biadmin/media_external/data-part-00001
hadoop fs -ls /user/biadmin/media_external
hadoop fs -cat /user/biadmin/media_external/data-part-00001
```



```
biadmin@bivm:~> hadoop fs -copyFromLocal /home/biadmin/db2export_media.del /user/biadmin/media_external/data-par
t-00001
biadmin@bivm:~> hadoop fs -ls /user/biadmin/media_external
Found 1 items
-rw-r--r--   1 biadmin biadmin        468 2014-06-19 15:59 /user/biadmin/media_external/data-part-00001
biadmin@bivm:~> hadoop fs -cat /user/biadmin/media_external/data-part-00001
"The Business Journals","www.bizjournals.com",20120105
222,"CNN","www.cnn.com",20120115
333,"CIO Today","www.cio-today.com",20120212
"Forbes","www.forbes.com",20120115
555,"Reuters","www.reuters.com",20120116
654,"Wall Street Journal","online.wsj.com",20120116
777,"BBC","bbc.com",20120116
765,"Healthcare IT News","www.healthcareitnews.com",20120220
876,"New York Times","www.nytimes.com",20120220
987,"Technology Marketing Corp.","www.tmcnet.com",20120202
biadmin@bivm:~> 
```

__4. You can also use the Biginsights Web Console to view this file. Click on Files on the horizontal menu and in the DFS File tab open the path to your file.



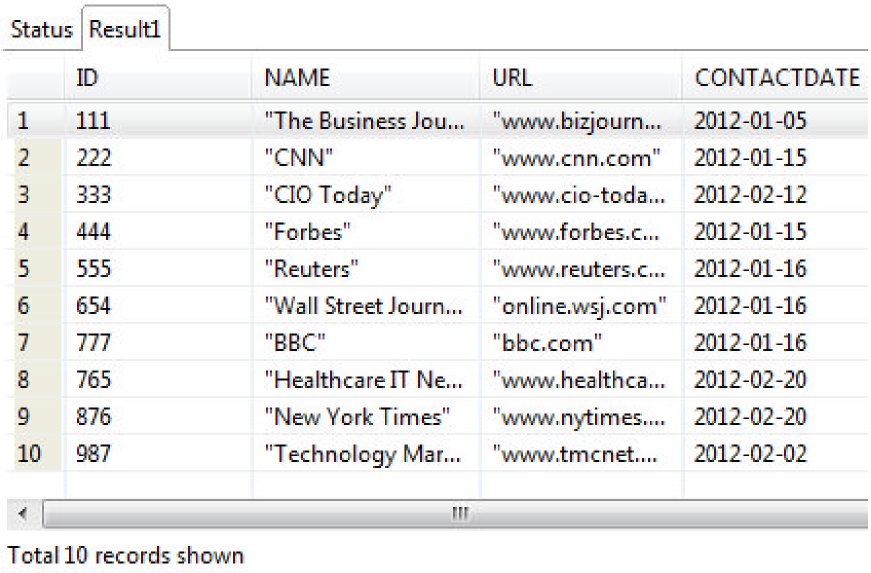
Path: /user/biadmin/media_external/data-part-00001

Name	Size	Block Size
data-part-00001	468 B	128.0 MB

```
"The Business Journals","www.bizjournals.com",20120105
222,"CNN","www.cnn.com",20120115
333,"CIO Today","www.cio-today.com",20120212
"Forbes","www.forbes.com",20120115
555,"Reuters","www.reuters.com",20120116
654,"Wall Street Journal","online.wsj.com",20120116
777,"BBC","bbc.com",20120116
765,"Healthcare IT News","www.healthcareitnews.com",20120220
876,"New York Times","www.nytimes.com",20120220
987,"Technology Marketing Corp.","www.tmcnet.com",20120202
```

__5. Now you can already query the data from the Big SQL interface, as well.

```
select * from media_external;
```



	ID	NAME	URL	CONTACTDATE
1	111	"The Business Jou...	"www.bizjourn...	2012-01-05
2	222	"CNN"	"www.cnn.com"	2012-01-15
3	333	"CIO Today"	"www.cio-toda...	2012-02-12
4	444	"Forbes"	"www.forbes.c...	2012-01-15
5	555	"Reuters"	"www.reuters.c...	2012-01-16
6	654	"Wall Street Journ...	"online.wsj.com"	2012-01-16
7	777	"BBC"	"bbc.com"	2012-01-16
8	765	"Healthcare IT Ne...	"www.healthca...	2012-02-20
9	876	"New York Times"	"www.nytimes....	2012-02-20
10	987	"Technology Mar...	"www.tmcnet....	2012-02-02

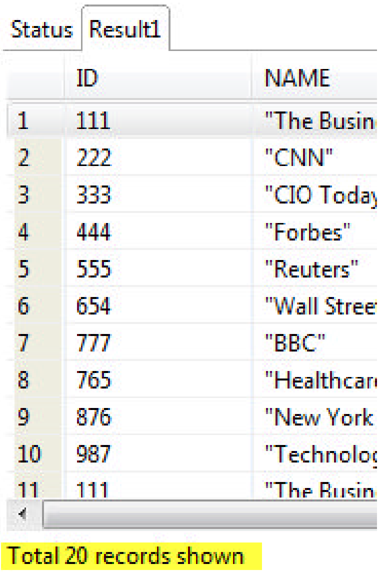
Total 10 records shown

Any file which matches the data types for these four columns and are separated by the terminator ',', and is placed in this Hadoop path, will be available to a Big SQL query.

__6. Copy the data file again, with a different file name, into the table's directory. (This command should be typed from the bash shell on *one* line. Don't forget to change userN.)

```
hadoop fs -copyFromLocal /path/to/file/db2export_media.del
/user/biadmin/media_external/data-part-00002
```

__7. Query the table again and confirm that now 20 rows were retrieved. If only 10 rows were retrieved, execute the next exercise and then repeat your query.



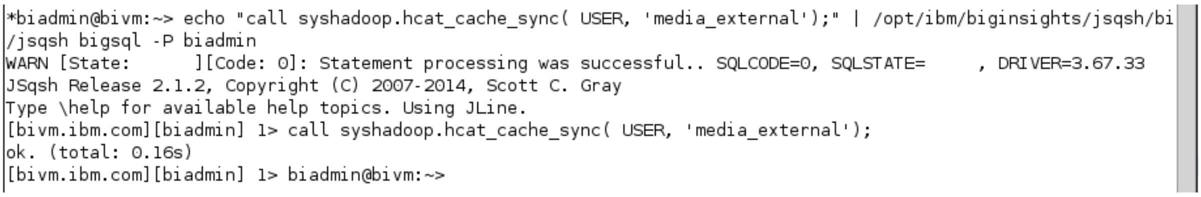
	ID	NAME
1	111	"The Busin
2	222	"CNN"
3	333	"CIO Today
4	444	"Forbes"
5	555	"Reuters"
6	654	"Wall Stree
7	777	"BBC"
8	765	"Healthcar
9	876	"New York
10	987	"Technolo
11	111	"The Busin

Total 20 records shown

__8. Caches are maintained at various levels within the software stack in order to speed up your queries. Although caches are flushed periodically, since the last query was probably executed very recently, you may not see your new data yet. Since we know that new data has arrived at the HDFS level, we will call a Big SQL procedure from JSqsh in order to flush the Big SQL cache for this table. (Note yet another method for executing a (single) command in JSqsh.)

```
echo "call syshadoop.hcat_cache_sync( USER, 'media_external');" |
/opt/ibm/biginsights/jsqsh/bin/jsqsh bigsql -P bigsql
```

The first parameter of this procedure is the table's schema name, so we are using the contents of the USER registry variable. The second parameter is the table name. (Don't forget the semi-colon at the end of the statement.)



```
*biadmin@bivm:~> echo "call syshadoop.hcat_cache_sync( USER, 'media_external');" | /opt/ibm/biginsights/jsqsh/bi
/jsqsh bigsql -P biadmin
WARN [State:      ][Code: 0]: Statement processing was successful.. SQLCODE=0, SQLSTATE=     , DRIVER=3.67.33
JSqsh Release 2.1.2, Copyright (C) 2007-2014, Scott C. Gray
Type \help for available help topics. Using JLine.
[bivm.ibm.com][biadmin] 1> call syshadoop.hcat_cache_sync( USER, 'media_external');
ok. (total: 0.16s)
[bivm.ibm.com][biadmin] 1> biadmin@bivm:~>
```

9.8. Dropping tables you created in this lab

If you'd like to drop the tables you created in this lab, execute these statements:

```
drop table media_del;
drop table media_new;
drop table media_db2table;
drop table media_db2select;
drop table media_db2table_jan;
drop table media_db2select_jan;
drop table media_external;
```

Lab 10 Summary

Congratulations! You've just learned many important aspects of Big SQL, IBM's query interface for big data. To expand your skills and learn more, enroll in free online courses offered by Big Data University (http://www.bigdatauniversity.com/) or work through free tutorials included in the BigInsights product documentation. The HadoopDev web site (https://developer.ibm.com/hadoop/) contains links to these and other resources.

NOTES

NOTES