IBM Training

Demonstration 1

Working with data in HBase tables using Big SQL

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

- Create an HBase table using Big SQL
- Query an HBase table using Big SQL

Demonstration 1: Working with data in HBase tables using Big SQL

Demonstration 1: Working with data in HBase tables using Big SQL

Purpose:
In this demonstration, you will learn the basics of using HBase natively and with Big SQL, IBM's industry-standard SQL interface for data stored in its Hadoop-based platform. HBase is an open source key-value data storage mechanism commonly used in Hadoop environments. It features a column-oriented data model that enables programmers to efficiently retrieve data by key values from very large data sets. It also supports certain data modification operations, readily accommodates sparse data, and supports various kinds of data. Indeed, HBase doesn't have primitive data types – all user data is stored as byte arrays. With BigInsights, programmers can use SQL to query data in Big SQL tables managed by HBase.

Although HBase provides useful data storage and retrieval capabilities, it lacks a rich query language. Fortunately, IBM's Big SQL technology, a component of BigInsights, enables programmers to use industry-standard SQL to create, populate, and query Big SQL tables managed by HBase. Native HBase data retrieval operations (such as GET and SCAN) can be performed on these tables, too. This lab introduces you to the basics of using Big SQL with HBase

Estimated time: 60 minutes

User/Password: **biadmin/biadmin**
root/dalvm3
db2inst1/ibm2blue

Services Password: **ibm2blue**

Task 1. Ensure BigInsights is running.

You may skip this task if your environment is still up and running. Refer to past demonstrations if you need to do any of these.

1. Ensure BigInsights is running via Ambari.
2. Ensure you can access the BigInsights Home page
3. Ensure you can access the Big SQL page
4. For this demonstration, make sure HBase is running as well. If not, start it up via the Ambari console.

Task 2. Using Big SQL to create and query HBase tables.

To get started, you will create a Big SQL table named `reviews` to capture information about product reviews. A script is provided with the statements you will need to run:

The 1_Big SQL HBase statements.sql file is located in the */home/biadmin/labfiles/bigsql/Big_SQL_HBase* folder.

1. Use either JSqsh or the Big SQL console to create the table. You will use the **bigsql** user id and **ibm2blue** password. Create the table with this statement.

```
CREATE HBASE TABLE IF NOT EXISTS BIGSQLLAB.REVIEWS (
REVIEWID        varchar(10) primary key not null,
PRODUCT     varchar(30),
RATING      int,
REVIEWERNAME    varchar(30),
REVIEWERLOC     varchar(30),
COMMENT     varchar(100),
TIP         varchar(100)
)
COLUMN MAPPING
(
key     mapped by (REVIEWID),
summary:product mapped by (PRODUCT),
summary:rating  mapped by (RATING),
reviewer:name   mapped by (REVIEWERNAME),
reviewer:location  mapped by (REVIEWERLOC),
details:comment    mapped by (COMMENT),
details:tip    mapped by (TIP)
);
```

   This statement creates a Big SQL table named REVIEWS in the default BIGSQL schema and instructs Big SQL to use HBase as the underlying storage manager. The COLUMN MAPPING clause specifies how SQL columns are to be mapped to HBase columns in column families. For example, the SQL REVIEWERNAME column is mapped to the HBase column family:column of 'reviewer:name'.

For simplicity, this example uses a 1:1 mapping of SQL columns to HBase columns. It also uses a single SQL column as the HBase row key. As you'll see in a subsequent lab, you may want (or need) to map multiple SQL columns to one HBase column or to the HBase row key. Big SQL supports doing so.

Also for simplicity, you have accepted various defaults for this table, including default HBase column family property settings (for VERSIONS and others) and binary encoding for storage. Finally, note that the SQL REVIEWID column was defined as the SQL primary key. This is an informational constraint that can be useful for query optimization. However, it isn't actively enforced.

2. Verify that the table was created in the Hadoop file system. If necessary, open up a new terminal window and list the contents of the HBase data directory and confirm that the *bigsql.reviews* subdirectory exists.

```
hdfs dfs -ls /apps/hbase/data/data/default
```

The root HBase directory is determined at installation. The examples in this lab were developed for an environment in which HBase was configured to store data in /apps/hbase/data/data/default. If your HBase configuration is different, adjust the commands as needed to match your environment.

3. List the contents of your table's subdirectory. Observe that this subdirectory contains another subdirectory with a system-generated name.

```
hdfs dfs -ls
/apps/hbase/data/data/default/bigsql.reviews
```

```
[biadmin@ibmclass Desktop]$ hdfs dfs -ls /apps/hbase/data/data/default/bigsql.reviews
Found 3 items
drwxr-xr-x   - hbase hdfs          0 2015-08-14 11:33 /apps/hbase/data/data/default/bigsql.reviews/.tabledesc
drwxr-xr-x   - hbase hdfs          0 2015-08-14 11:33 /apps/hbase/data/data/default/bigsql.reviews/.tmp
drwxr-xr-x   - hbase hdfs          0 2015-08-14 11:33 /apps/hbase/data/data/default/bigsql.reviews/42a3277d4abf48ae2c9d37cec2551342
```

4. List the contents of the . . . /bigsql.reviews subdirectory with the system-generated name. (In the example above, this is the . . . / 42a3277d4abf48ae2c9d37cec2551342 subdirectory.) Adjust the path specification below to match your environment.

```
hdfs dfs -ls
/apps/hbase/data/data/default/bigsql.reviews/42a3277d4ab
f48ae2c9d37cec2551342
```

```
/apps/hbase/data/data/default/bigsql.reviews/42a3277d4abf48ae2c9d37cec2551342/.regioninfo
/apps/hbase/data/data/default/bigsql.reviews/42a3277d4abf48ae2c9d37cec2551342/details
/apps/hbase/data/data/default/bigsql.reviews/42a3277d4abf48ae2c9d37cec2551342/reviewer
/apps/hbase/data/data/default/bigsql.reviews/42a3277d4abf48ae2c9d37cec2551342/summary
```

Note that there are additional subdirectories for each of the 3 column families specified in the COLUMN MAPPING clause of your CREATE HBASE TABLE statement.

5. Back to your Big SQL execution environment. Insert a row into your Big SQL reviews table. Note that we are using the default bigsql schema here.

```
insert into reviews values
('198','scarf','2','Bruno',null,'Feels cheap',null);
```

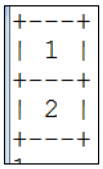
Note that this INSERT statement looks the same as any other SQL statement. You didn't need to insert one HBase cell value at a time, and you didn't need to understand the underlying HBase table structure.

6. Insert another row into your table, specifying values for only a subset of its columns.

```
insert into reviews (reviewid, product, rating,
reviewername) values ('298','gloves','3','Beppe');
```

7. Use SQL to count the number of rows stored in your table, verifying that 2 are present.

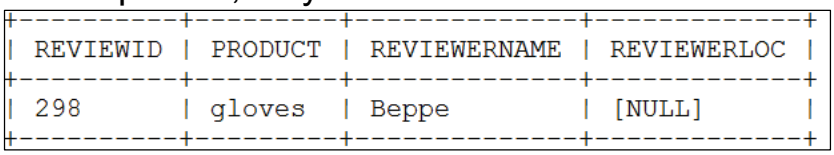
```
select count(*) from reviews;
```



8. Execute a simple query to return specific information about reviews of products rated 3 or higher.

```
select reviewid, product, reviewername, reviewerloc
from reviews
where rating >= 3;
```

As expected, only 1 row is returned.



Again, note that your SELECT statement looks like any other SQL SELECT - you didn't need to add any special code because the underlying storage manager is HBase.

9. Verify that you can work directly with the *bigsql.reviews* table from HBase. Launch the HBase shell. From the HBase home directory (such as `/usr/iop/current/hbase-client/bin`), issue this command:

```
hbase shell
```

10. Ignore any informational messages that may appear. Verify that the shell launched successfully and that your screen appears similar to this:

```
[biadmin@ibmclass bin]$ ./hbase shell
2015-08-14 11:49:33,678 INFO  [main] Configuration.deprecation: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.8_IBM_4-hadoop2, rUnknown, Fri Mar 27 21:53:57 PDT 2015

hbase(main):001:0>
```

11. Scan the table:

```
scan 'bigsql.reviews'
```

```
hbase(main):001:0> scan 'bigsql.reviews'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
ROW                                   COLUMN+CELL
 \x00198\x00                          column=details:comment, timestamp=1439570623068, value=\x00Feels cheap\x00
 \x00198\x00                          column=details:tip, timestamp=1439570623068, value=\x01
 \x00198\x00                          column=reviewer:location, timestamp=1439570623068, value=\x01
 \x00198\x00                          column=reviewer:name, timestamp=1439570623068, value=\x00Bruno\x00
 \x00198\x00                          column=summary:product, timestamp=1439570623068, value=\x00scarf\x00
 \x00198\x00                          column=summary:rating, timestamp=1439570623068, value=\x00\x80\x00\x00\x02
 \x00298\x00                          column=details:comment, timestamp=1439570671430, value=\x01
 \x00298\x00                          column=details:tip, timestamp=1439570671430, value=\x01
 \x00298\x00                          column=reviewer:location, timestamp=1439570671430, value=\x01
 \x00298\x00                          column=reviewer:name, timestamp=1439570671430, value=\x00Beppe\x00
 \x00298\x00                          column=summary:product, timestamp=1439570671430, value=\x00gloves\x00
 \x00298\x00                          column=summary:rating, timestamp=1439570671430, value=\x00\x80\x00\x00\x03
2 row(s) in 0.3280 seconds
```

As you would expect, the final line of output reports that there are 2 rows in your table. In case you're curious, \x00 is both the non-null marker and also the terminator used for variable length binary encoded values.

Task 3. Creating views over Big SQL HBase tables.

You can create views over Big SQL tables stored in HBase just as you can create views over Big SQL tables that store data in the Hive warehouse or in simple DFS files. Creating views over Big SQL HBase tables is straightforward, as you'll see in this task.

1. From your Big SQL query execution environment, create a view based on a subset of the `reviews` table that you created earlier.

```
create view testview as
select reviewid, product, reviewername, reviewerloc
from reviews
where rating >= 3;
```

Note that this view definition looks like any other SQL view definition. You didn't need to specify any syntax unique to Hadoop or HBase.

2. Query the view.

```
select reviewid, product, reviewername from testview;
```

```
+----------+---------+--------------+
| REVIEWID | PRODUCT | REVIEWERNAME |
+----------+---------+--------------+
| 298      | gloves  | Beppe        |
+----------+---------+--------------+
1 row in results(first row: 0.50s; total: 0.50s)
```

Task 4. Loading data into a Big SQL HBase table.

In this task, you'll explore how to use the Big SQL LOAD command to load data from a file into a Big SQL table managed by HBase. To do so, you will use sample data shipped with Big SQL that is typically installed with Big SQL client software. By default, this data is installed at /usr/ibmpacks/bigsql/4.0/bigsql/samples/data.
The sample data represents data exported from a data warehouse that tracks sales of outdoor products. It includes a series of FACT and DIMENSION tables. In this task, you will create 1 DIMENSION table and load sample data from 1 file into it.

1. Create a Big SQL table in HBase named sls_product_dim.

```
-- product dimension table
CREATE HBASE TABLE IF NOT EXISTS sls_product_dim
 ( product_key INT PRIMARY KEY 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
 )
COLUMN MAPPING
(
key                    mapped by (PRODUCT_KEY),
data:line_code         mapped by (PRODUCT_LINE_CODE),
data:type_key          mapped by (PRODUCT_TYPE_KEY),
data:type_code         mapped by (PRODUCT_TYPE_CODE),
data:number            mapped by (PRODUCT_NUMBER),
```

```
    data:base_key          mapped by (BASE_PRODUCT_KEY),
    data:base_number       mapped by (BASE_PRODUCT_NUMBER),
    data:color             mapped by (PRODUCT_COLOR_CODE),
    data:size              mapped by (PRODUCT_SIZE_CODE),
    data:brand_key         mapped by (PRODUCT_BRAND_KEY),
    data:brand_code        mapped by (PRODUCT_BRAND_CODE),
    data:image             mapped by (PRODUCT_IMAGE),
    data:intro_date        mapped by (INTRODUCTION_DATE),
    data:discon_date       mapped by (DISCONTINUED_DATE)
);
```

The HBase specification of this Big SQL statement placed nearly all SQL columns into a single HBase column family named 'data'. As you know, HBase creates physical files for each column family; this is something you should take into consideration when designing your table's structure. It's often best to keep the number of column families per table small unless your workload involves many queries over mutually exclusive columns. For example, if you knew that PRODUCT_IMAGE data was rarely queried or frequently queried alone, you might want to store it separately (i.e., in a different column family:column).

There's something else worth noting about this table definition: the HBase columns have shorter names than the SQL columns. HBase stores full key information (row key, column family name, column name, and timestamp) with the key value. This consumes disk space. If you keep the HBase column family and column names short and specify longer, more meaningful names for the SQL columns, your design will minimize disk consumption and remain friendly to SQL programmers.

Load data into the table.

2. This statement will return a warning message providing details on the number of rows loaded, etc.

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

3. Count the number of rows in the table to verify that 274 are present.

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

4. Optionally, query the table.

```
select product_key, introduction_date,
product_color_code
from sls_product_dim
where product_key > 30000
fetch first 5 rows only;
```

```
+-------------+-------------------------+--------------------+
| PRODUCT_KEY | INTRODUCTION_DATE       | PRODUCT_COLOR_CODE |
+-------------+-------------------------+--------------------+
|       30001 | 1995-02-15 00:00:00.000 |                908 |
|       30002 | 1995-02-15 00:00:00.000 |                906 |
|       30003 | 1995-02-15 00:00:00.000 |                924 |
|       30004 | 1995-02-15 00:00:00.000 |                923 |
|       30005 | 1995-02-15 00:00:00.000 |                923 |
+-------------+-------------------------+--------------------+
5 rows in results(first row: 0.69s; total: 0.69s)
```

Task 5. Dropping tables created in this lab.

1. Drop the reviews table and the sls_product_dim table.

```
drop table reviews;
drop table sls_product_dim;
```

2. Verify that these tables no longer exist. For example, query each table and confirm that you receive an error message indicating that the table name you provided is undefined (SQLCODE -204, SQLSTATE 42704).

```
select count(*) from reviews;
```

Results:
In this demonstration, you learned the basics of using HBase natively and with Big SQL, IBM's industry-standard SQL interface for data stored in its Hadoop-based platform.

This demonstration introduced you to the basics of using Big SQL with HBase.