Use Data from a Hadoop Cluster with Oracle Database

Hands-On Lab

Lab Structure

Acronyms:

OLH: Oracle Loader for Hadoop
OSCH: Oracle SQL Connector for Hadoop Distributed File System (HDFS)

All files are located in `/home/oracle/movie/moviework/osch` and `/home/oracle/movie/moviework/olh`.

The files contain scripts that are complete and ready to run.

Lines in red highlight scripts to be executed.

`prompt>` means the script has to be run from the operating system prompt.

`SQL>` means the script has to be run from SQLPlus.

The scripts are not part of the product. They were created for this Hands-on Lab.

Part 1. The scripts in `/home/oracle/movie/moviework/osch` work with Oracle SQL Connector for HDFS. This connector is useful for bulk load of data and accessing data from Oracle Database via external tables.

Part 1 of the lab will show you how to

- Use Oracle SQL Connector for HDFS to query data in Hive or in files on HDFS from Oracle Database using external tables.
- Use SQL to load this data from external tables into Oracle Database.

Part 2. The scripts in `/home/oracle/movie/moviework/olh` work with Oracle Loader for Hadoop. This connector is useful for data that is coming in continuously and is loaded frequently.

Part 2 of the lab will show you how to

- Use Oracle Loader for Hadoop to load this data into Oracle Database. Oracle Loader for Hadoop pre-processes the data on Hadoop so that there is less impact on the database during the load.

Part 3. The scripts in `/home/oracle/movie/moviework/osch/preprocess` show how Oracle SQL Connector for HDFS and Oracle Loader for Hadoop can be used together for potential efficiency gains.

Environment variables

The following environment variables are required by the labs and have been set in your environment (or are set in `reset_conn.sh`, which is executed at the beginning of this lab).

For Oracle Loader for Hadoop

    $OLH_HOME: /u01/connectors/oraloader/

    ${OLH_HOME}/jlib/* has been added to HADOOP_CLASSPATH

For Oracle SQL Connector for HDFS

    $OSCH_HOME: /u01/connectors/orahdfs

    ${OSCH_HOME}/jlib/* has been added to HADOOP_CLASSPATH

For using Hive with these connectors

`${HIVE_HOME}/lib/*` and `${HIVE_HOME}/conf` have been added to `HADOOP_CLASSPATH`

Schema: We will use the `moviedemo` schema in this lab. The password is `welcome1`.

Setup

The following script cleans up any directories and files from previous runs of this lab and creates a new HDFS directory for storing output files.

```
prompt> sh /home/oracle/movie/moviework/reset/reset_conn.sh
```

(Note: Ignore any errors. If there were no prior runs of this lab on this VM there will be errors due to dropping tables that do not exist and deleting directories that do not exist.)

Part 1: Oracle SQL Connector for HDFS

Oracle SQL Connector for HDFS enables an external table in Oracle Database to access data on HDFS. The data can be in a Hive table, text files on HDFS, or Oracle Data Pump files (created by Oracle Loader for Hadoop) on HDFS. The external table can be queried in the database using SQL like any other table in the database. The external table can be joined with tables in the database. Rows can be selected from the external table and inserted into tables in the database. Note that external tables cannot be updated or indexed.

Definition of Terms

Configuration parameters: These properties are used by the connectors during execution. They can be specified in an XML file or on the command line. Examples of configuration parameters are the location of data files, database connection information, table name, schema name, and so on.

Location files: Location files are part of the LOCATION clause in an external table definition. These files are populated by Oracle SQL Connector for HDFS and will contain URLs of the data files on HDFS.

Publish location files: This step refers to running Oracle SQL Connector for HDFS with the `–publish` option to create and populate location files for an existing external table. As an example, this option will be used when new data files have been added and have to be accessed with an existing external table.

Data

Let us examine some of the data files on HDFS we will use in these examples.

```
prompt> hadoop fs -ls /user/oracle/moviework/data
```

There are 4 input files in the HDFS directory. Note that one file is larger than the other three files, this example is used to show automatic load balancing while reading data from text files. To see the sample input data in one of the smaller files:

```
prompt> hadoop fs -cat /user/oracle/moviework/data/part-00002
```

Sample input data:

```
1084372 16617    8       2012-10-01:01:29:34     0       11              1.99

1191532 59440    30      2012-10-01:01:30:27     1       4
```

```
1106264 28      18      2012-10-01:01:35:19     1       11              2.99

1061810 121     7       2012-10-01:01:56:42     1       11              2.99

1135508 240     8       2012-10-01:01:58:00     1       2

1135508 240     8       2012-10-01:02:04:15     1       5

1135508 1092    20      2012-10-01:02:10:23     0       5

1135508 240     8       2012-10-01:02:31:12     1       11              2.99

1191532 59440   30      2012-10-01:03:11:35     1       2

1191532 59440   30      2012-10-01:03:19:24     1       11              3.99
```

(The blank values are NULL.)

Part 1a: Accessing Hive Tables with Oracle SQL Connector for HDFS

We first access Hive tables from Oracle Database via external tables. The external table definition is **generated automatically** from the Hive table definition. Hive table data can be accessed by querying this external table. The data can be queried with Oracle SQL and joined with other tables in the database.

We will use the Hive table `movieapp_log_stage_1` that uses data from the MapReduce lab. We will access this Hive table from Oracle Database.

Step 1:

```
prompt> cd /home/oracle/movie/moviework/osch
```

Let us look at these files: `genloc_moviefact_hive.sh, moviefact_hive.xml.`

`genloc_moviefact_hive.sh` is a script to execute Oracle SQL Connector for HDFS. `moviefact_hive.xml` contains the configuration parameters used.

We can examine the script by viewing the contents of the file.

```
prompt> more genloc_moviefact_hive.sh
```

Oracle SQL Connector for HDFS is run with the `-createTable` option. This option generates the external table definition, creates the external table, and populates the location files in the LOCATION clause of the external table. After this step the external table is available for query.

Let us look at the configuration parameters in `moviefact_hive.xml`.

```
prompt> more moviefact_hive.xml
```

Some of the parameters are:

(i) The name of the external table that we want to create in Oracle Database to access the Hive table, the schema containing the table
(ii) Name of the Hive table
(iii) Database connection information
(iv) `oracle.hadoop.exttab.sourceType=hive`

Step 2: Run the script.

```
prompt> sh genloc_moviefact_hive.sh
```

You will be prompted for the password. The password is `welcome1`

You can see the external table definition from the output on screen, and also the contents of the location files. The location files contain the URIs of the data files on HDFS that contain the data in the Hive table.

The Hive table can now be queried by the database via the external table. It is as simple as that!

You can also look at the external table definition in SQLPlus:

```
prompt> sqlplus moviedemo/welcome1
```

```
SQL> describe movie_fact_ext_tab_hive;
```

You can try some queries on this external table, which will query data in the Hive table.

```
SQL> select count(*) from movie_fact_ext_tab_hive;

  COUNT(*)

----------

    296915
```

```
SQL> select custid from movie_fact_ext_tab_hive where rownum < 10;
```

You can join this table with a table in Oracle Database. Here we join it the `movie` table in the database, to list movie titles by `custid` (we list the first few rows here).

```
SQL> select custid, title from movie_fact_ext_tab_hive p, movie q
where p.movieid = q.movie_id and rownum < 10;
```

Note on Parallelizing Access

When a Hive table is an external table or a bucketed table, its data is likely to be in multiple data files. In such cases, the external table access can be parallelized, with parallel database threads reading the data files in parallel. This will be discussed further in Part 1b.

Populating an Oracle Database Table

The data in the Hive table can be inserted into a database table using SQL. If the Hive table is large and will be frequently queried, the data should be loaded into the database, as external tables have no indexes and no partitioning.

```
SQL> create table movie_fact_local as select * from
movie_fact_ext_tab_hive;
```

Part 1b: Accessing Files on HDFS with Oracle SQL Connector for HDFS

In this section we access text data files from Oracle Database via external tables.

Step 1:

Let us look at these files in `/home/oracle/movie/moviework/osch`: `genloc_moviefact_text.sh, moviefact_text.xml.`

`genloc_moviefact_text.sh` is a script to execute Oracle SQL Connector for HDFS. `moviefact_text.xml` contains the configuration parameters used.

We can examine the script by viewing the contents of the file.

```
prompt> more genloc_moviefact_text.sh
```

Again, we use Oracle SQL Connector for HDFS with the `-createTable` option. This step generates the external table definition, creates the external table, and populates the location files in the external table LOCATION clause with the URIs of the data files on HDFS.

Let us look at the configuration parameters in `moviefact_text.xml`.

```
prompt> more moviefact_text.xml
```

Some of the parameters are:

(i) The name of the table that we want to create in Oracle Database to access the files on HDFS, the schema containing the table

(ii) Location of the data files on HDFS
(iii) Database connection information
(iv) Schema containing the table
(v) `locationFileCount=2`
(vi) `oracle.hadoop.exttab.sourceType=text`
(vii) `oracle.hadoop.exttab.columnNames` (a list of column names the external table should use)

Step 2: Run the script.

```
prompt> sh genloc_moviefact_text.sh
```

You will be prompted for the password. The password is `welcome1`

You can see the external table definition from the output on screen, and also the location files. The location files contain the URIs of the data files on HDFS.

Step 3: The created external table `movie_fact_ext_tab_text` is now ready for query. It can be queried like any other table in the database.

```
prompt> sqlplus moviedemo/welcome1

SQL> select count(*) from movie_fact_ext_tab_file;

  COUNT(*)

----------

    300025

SQL> describe movie_fact_ext_tab_file;
```

Note that all the columns are of type varchar2, because there is no data type information in the data files on HDFS. This is the default behavior when accessing text files. Refer to the documentation to see how you can specify data types so that the external table columns will have user specified data types.

Select `cust_id` for a few rows in the table.

```
SQL> select cust_id from movie_fact_ext_tab_file where rownum < 10;
```

Join with the `movie` table in the database to list movie titles by cust_id (we list the first few rows here).

```
SQL> select cust_id, title from movie_fact_ext_tab_file p, movie q
where p.movie_id = q.movie_id and rownum < 10;
```

Parallel Access to Data

Note that there are two location files in the generated external table. This is because `moviefact_text.xml` contains the parameter `oracle.hadoop.exttab.locationFileCount` with a value of `2`, which specifies the number of location files in the LOCATION clause of the external table. Location files are read in parallel by the database. Location files contain the URIs of the data files on HDFS. Oracle SQL Connector for HDFS distributes the URIs of the data files between the location files so that to the extent possible each location file is pointing to the same volume of data. The goal is to ensure that (where possible) each parallel thread does a similar amount of work. Files are not split up, but they are grouped evenly (where possible) by size. This load balancing is automatically done by the connector based on the `locationFileCount` value.

The default value of `oracle.hadoop.exttab.locationFileCount` is 4. If the value specified by `locationFileCount` is greater than the number of data files being read then the number of location files will match the number of data files.

Part 1c: Updating Location Files

Location files for an existing external table will need to be updated when

(1) New data files are added to HDFS and need to be accessed by the same external table.
(2) The number of parallel threads that can be used to access the data has changed.

Oracle SQL Connector for HDFS can be executed with the `–publish` option to change location files. Existing location files will be deleted and new ones created and populated with the URIs to the datafiles. To use this option the external table must exist, created with the `–createTable` option.

Step 6: We will first add a new data file.

```
prompt> hadoop fs -put /home/oracle/movie/moviework/data/part-00005
  moviework/data
```

You can see the new data file on HDFS:

```
prompt> hadoop fs -ls moviework/data
```

Step 7: Run Oracle SQL Connector for HDFS with the `–publish` option.

```
prompt> sh genloc_moviefact_publish.sh
```

This script uses a configuration file `moviefact_publish.xml` with `locationFileCount=3`. This will change the degree of parallelism in the previous example to 3.

You can see the external table definition from the output on screen, and also the location files with the URIs of the data files on HDFS. You can see that there are three location files created, since we use `locationFileCount=3`. Two of the location files each contain the URI of a large data file, and the third location file contains the URIs of all the smaller data files. Three database threads read the data in parallel.

Querying the table again, we see the additional rows from the new data file.

```
prompt> sqlplus moviedemo/welcome1

SQL> select count(*) from movie_fact_ext_tab_file;

  COUNT(*)

----------

    359544
```

Additional Notes

Creating your own external table

You might want to create your own external table for some use cases, for example:

- Add some format transformation operations to the access parameters clause of the external table definition, for example to use DEFAULTIF clause to default to a specific value.

You can use Oracle SQL Connector for HDFS with the `--noexecute` option to see the definition of the external table without creating the table. An example of using `--noexecute` is in `genloc_moviefact_manual_tab.sh`.

```
prompt> more genloc_moviefact_manual_tab.sh

prompt> sh genloc_moviefact_manual_tab.sh
```

The external table displayed on screen can be copied and modified. After the updated external table is manually created the location files can be populated with the `-publish` option.

Other Useful Options

`-listLocations`: This option lists the contents of the location files of an external table that has already been created

For example:

```
prompt> hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
                oracle.hadoop.exttab.ExternalTable \
               -conf /home/oracle/movie/moviework/osch/moviefact_text.xml  \
               -listLocations
```

`-getDDL`: This option lists the definition of an external table that has already been created

For example:

```
prompt> hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
                 oracle.hadoop.exttab.ExternalTable \
                -conf /home/oracle/movie/moviework/osch/moviefact_text.xml   \
                -getDDL
```

Part 2: Oracle Loader for Hadoop

In this section we use Oracle Loader for Hadoop to load into a table in Oracle Database.

Oracle Loader for Hadoop is a MapReduce program that runs on the Hadoop cluster to pre-process the data. It can partition, sort, and convert data into Oracle data types in preparation for the load. This offloads some database cycles on to Hadoop, so that less database CPU is used during the load itself. In the *online mode* the pre-processed data is directly loaded into the database. In the *offline mode* Oracle Loader for Hadoop writes out the pre-processed data as data pump files on HDFS (we will work through an example of that in Part 3). In this section we use Oracle Loader for Hadoop in the *online mode.* There are two load options in the online mode, JDBC and direct path. We use the direct path load option here.

Step 1:

```
prompt> cd /home/oracle/movie/moviework/olh
```

Let us examine the files used in this section:

- `prompt> more moviesession.xml`

  This file contains the configuration parameters for the execution of Oracle Loader for Hadoop. You can examine the file to see the parameters such as
  - `mapreduce.inputformat.class`: Specifies the input format of the input data file (in this example the input data is delimited text, so the value for this parameter is the class name `oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`.)
  - `oracle.hadoop.loader.input.fieldTerminator`: Specifies the character used as a field terminator in the input data file. In this example it is Tab, represented by its hex value.
  - `mapreduce.outputformat.class`: Specifies the type of load. We specify here the value `OCIOutputFormat` to use the direct path online load option.
  - `mapred.input.dir`: Location of the input data file on HDFS.
  - `mapred.output.dir`: Specifies the HDFS directory where output files should be written, such as the _SUCCESS and _log files.

    - `oracle.hadoop.loader.loaderMapFile`: Specifies the name and location (typically on the client file system) of the *loaderMap* file. See next bullet point.

- `prompt> more loaderMap_moviesession.xml`

  This file specifies the target table that data will be loaded into, and the mapping of input data to columns in the target table. Note the date format in the DATE_ID column, which specifies the date format in the input data file. The Java date format should be used to specify the date format in the Loader Map file.
  If a Loader Map file is not specified, Oracle Loader for Hadoop will assume that all the columns in the table will be loaded, and will use the default date specification. If a Loader Map file is not specified the `oracle.hadoop.loader.targetTable` property and `oracle.hadoop.loader.input.fieldNames` property must be specified.

- `prompt> more runolh_session.sh`

  This is the script to invoke Oracle Loader for Hadoop, which will run as a MapReduce job on the Hadoop cluster. It uses `olh_moviefact.xml`, the file containing the configuration parameters.

Step 2: Let us create the target table in the database that we will be loading data into. This table is hash partitioned on column `cust_id`.

```
prompt>  sqlplus moviedemo/welcome1
```

```
SQL> @moviesession.sql
```

Step 3: Invoke the Oracle Loader for Hadoop job to load the session data.

```
prompt>  sh runolh_session.sh
```

This will start the MapReduce job that will load the data into the target table.

Step 4: After the MapReduce job completes check that the rows have been loaded.

Note that one row had a parse error. The .bad file containing the row and the error are logged in the `_olh` directory under the directory specified in `mapred.output.dir`. (In this example the date value is invalid – it is a time between 2.00 and 3.00am on the day we switch over to daylight savings time).

```
prompt> sqlplus moviedemo/welcome1
```

```
SQL> select count(*) from movie_sessions_tab;

  COUNT(*)
```

```
----------

      4526
```

The data is now available for query.

```
SQL> select cust_id from movie_sessions_tab where rownum < 10;

SQL> select max(num_browsed) from movie_sessions_tab;

SQL> select cust_id from movie_sessions_tab where num_browsed = 6;

SQL> select cust_id from movie_sessions_tab where num_browsed in
(select max(num_browsed) from movie_sessions_tab);
```

OPTIONAL

Part 3: Using Oracle SQL Connector for HDFS and Oracle Loader for Hadoop

Let us take a look at Part 1 again. In Part 1, we accessed and loaded input data files using Oracle SQL Connector for HDFS without any pre-processing. Alternatively, input data can first be pre-processed using Oracle Loader for Hadoop. The data can be partitioned, and sorted by primary key within each partition. The data can also be converted into Oracle Data Pump files, an Oracle binary format. One reason for pre-processing the data using Oracle Loader for Hadoop is to offload some of the processing done in the database on to Hadoop, reducing the use of database CPU time during the load itself.

The data pump files that are created can then be accessed by Oracle SQL Connector for HDFS.

****

NOTE: There is an error in the database connection information in olh_moviefact.xml. Database connection to the database in the VM fails without this fix.

Edit the file to make this change.

Change:

```
<property>
  <name>HOST</name>
  <value>bigdatalite.us.oracle.com</value>
</property>
```

to

```
<property>
  <name>HOST</name>
  <value>bigdatalite.localdomain</value>
</property>
```

****

Step 1: Create a partitioned table that is the target table for Oracle Loader for Hadoop.

```
prompt> cd /home/oracle/movie/moviework/osch/preprocess
```

```
prompt> sqlplus moviedemo/welcome1
```

```
SQL> @../moviefact_part.sql
```

(the file is in a directory that is one level above).

Step 2: Use Oracle Loader for Hadoop to partition the data and convert it into data pump files. Oracle Loader for Hadoop is used in the *offline mode* to pre-process the input data files and write output files to HDFS.

Let us examine the files used for Oracle Loader for Hadoop:

- `prompt> more olh_moviefact.xml`

  This file contains the configuration parameters for the execution of Oracle Loader for Hadoop. You can examine the file to see the parameters (several of them are Hadoop properties) such as
  - `mapreduce.inputformat.class`: Specifies the input format of the input data file (in this example the input data is delimited text, so the value for this parameter is `DelimitedTextInputFormat`).
  - `oracle.hadoop.loader.input.fieldTerminator`: Specifies the character used as the field terminator in the input data file. In this example it is Tab, represented by its hex value.
  - `mapreduce.outputformat.class`: Specifies the type of load. We specify here the value `DataPumpOutputFormat` to pre-process the text data into Oracle Data Pump files for efficient loading that uses less database CPU resources when compared to loading text data.
  - `mapred.output.dir`: Specifies the HDFS directory where Oracle Loader for Hadoop should write the output files. This directory will contain the _SUCCESS and _log files from the loader job, and the pre-processed files created by the loader.
  - Database connection information. **(Note: Specifying the password in clear text in the configuration file is insecure. Use Oracle Wallet and specify the wallet location in the configuration. Refer to the documentation for details.)**
  - `oracle.hadoop.loader.loaderMapFile`: Specifies the name and location (typically on the client file system) of the loaderMap file. See next bullet point.

- `prompt> more loaderMap_olh.xml`

  This file specifies the target table that data will be loaded into, and maps the input data to columns in the target table. If a Loader Map file is not specified, Oracle Loader for Hadoop will assume that all the columns in the table will be loaded. In that case, the `oracle.hadoop.loader.targetTable` property and `oracle.hadoop.loader.input.fieldNames` property should be specified.

- `prompt> more runolh_moviefact.sh`

  This is the script to invoke Oracle Loader for Hadoop, which will run as a MapReduce job on the Hadoop cluster. It uses `olh_moviefact.xml`, the file containing the configuration parameters.

Step 3: Now we will invoke the Oracle Loader for Hadoop job to load to pre-process the data and create output files.

```
prompt>  sh runolh_moviefact.sh
```

This will run as a MapReduce job and create the output files. It creates 3 output data files, one for each partition, as the target table specified in `olh_moviefact.xml` has 3 partitions.

When the MapReduce job completes, check that the output files have been created:

```
prompt> hadoop fs -ls /user/oracle/moviework/data_dp
```

This will show a listing of the _SUCCESS and log files from the map reduce job, and also the three output data files, one for each partition.

Step 4: Now let us see how we access these files from Oracle Database. We follow steps very similar to Part 1.

The 2 files we use (also in `/home/oracle/movie/moviework/osch/preprocess`) are: `moviefact_dp.xml` and `genloc_moviefact_dp.sh`. They are similar to the files we used in Part 1.

Step 5: Create the external table and publish the location files. This is done by executing Oracle SQL Connector for HDFS with the `-createTable` option.

```
prompt> sh genloc_moviefact_dp.sh
```

You will be prompted for the password. The password is `welcome1`

This creates the external table and populates the location files in the external table.

Step 6: You are now ready to query the table.

```
prompt> sqlplus moviedemo/welcome1
```

```
SQL> select count(*) from movie_fact_ext_tab_dp;

  COUNT(*)

----------

    359544
```

This table can be queried like any other table in the database. You can try other queries on the external table, and also try loading the data into a table in the database. Three streams of data from the three location files are read by the database to answer queries.

```
SQL> select cust_id from movie_fact_ext_tab_dp where rownum < 10;
```

Join with the `movie` table in the database to list movie titles by `cust_id` (we list the first few rows here).

```
SQL> select cust_id, title from movie_fact_ext_tab_dp p, movie q where
p.movie_id = q.movie_id and rownum < 10;
```

Step 5: Use SQL to load the data from the external table into the partitioned target table we had created. The three partitions will be loaded in parallel.

```
SQL> insert into movie_fact_db_tab select * from
movie_fact_ext_tab_dp;
```