ORACLE
BIG DATA

May 2015

Oracle Loader for Hadoop and Oracle SQL Connector for HDFS

Examples

ORACLE

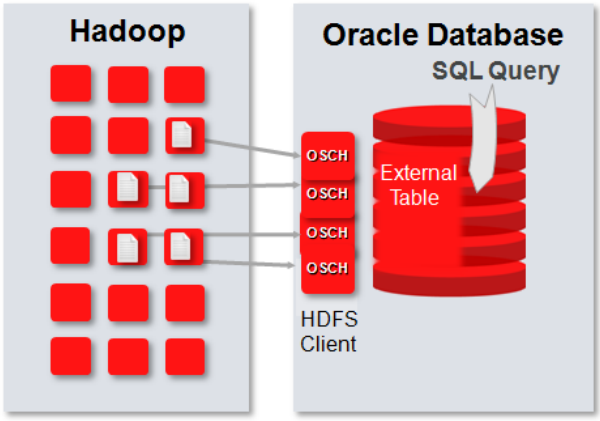
Introduction

Oracle Loader for Hadoop and Oracle SQL Connector for Hadoop Distributed File System (HDFS) enable high performance load and access of data from a Hadoop platform to Oracle Database. These efficient connectors, optimized for Hadoop and Oracle Database, make it easy to acquire and organize unstructured data on Hadoop and bring it together with data in Oracle Database, so that applications can analyze all data in the enterprise.

This document describes examples to work with Oracle Loader for Hadoop and Oracle SQL Connector for HDFS. It accompanies a kit that contains the examples and sample data.

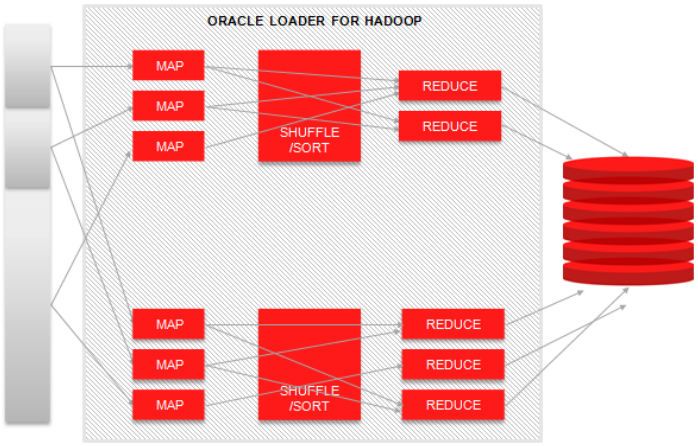
Oracle SQL Connector for HDFS

Oracle SQL Connector for HDFS uses external tables to provide Oracle Database with read access to Hive tables, delimited text files, and Oracle Data Pump files on Hadoop clusters. A utility available with Oracle SQL Connector for HDFS generates an external table for data on Hadoop. Oracle SQL can then be used to query this external table and load data from this table into the database. The data is accessed and loaded in parallel for very high speed load into the database.



Oracle Loader for Hadoop

Oracle Loader for Hadoop is an efficient and high-performance loader for fast movement of data from a Hadoop cluster into Oracle database. It pre-processes the data on Hadoop to optimize the load. The pre-processing includes partitioning the data by partition key, sorting the data, and transforming it into a database-ready format. Performing these operations on Hadoop leverages the parallel processing framework of Hadoop to perform operations typically performed on the database server as part of the load process. Offloading the operations to Hadoop reduces the CPU requirements on the database server, thereby lessening the performance impact on other database tasks.



Framework

Acronyms

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

Operating system prompts are represented in this document as '`prompt>`'
SQL*Plus prompts are represented in this document as '`SQL>`'

Examples

Part 1. Part 1 contains examples with Oracle SQL Connector for HDFS.

- Use Oracle SQL Connector for HDFS to query text files in-place in HDFS
- Use Oracle SQL Connector for HDFS to query data in Hive
- Use Oracle SQL Connector for HDFS query select Hive partitions
- Illustrate performance tuning with Oracle SQL Connector for HDFS

Part 2. Part 2 contains examples with Oracle Loader for Hadoop.

- Use Oracle Loader for Hadoop to load data from delimited text files in HDFS into Oracle Database
- Use Oracle Loader for Hadoop to load data from Hive tables into Oracle Database
- Use Oracle Loader for Hadoop to load data from Apache log files in HDFS into Oracle Database

Product Installation

These examples work with the Oracle Big Data Lite VM, which has installed versions of Oracle Loader for Hadoop and Oracle SQL Connector for HDFS. If you are using this VM you can skip the rest of this section.

If you would like to use the examples in your own environment, below are the Hadoop and Oracle Database versions to use, the download locations for the products, and environment variables.

Hadoop and Hive Versions

The Hadoop and Hive versions on your cluster should be:

- Hadoop: A distribution based on Apache Hadoop 2.x (certified Hadoop distributions are listed here).
- Hive: 010.0 or above

Contact us if you are interested in using the connectors but would like to use a distribution that is not yet certified.

Oracle Database

Install Oracle Database 11.2.0.3 or higher. Oracle Database 12.1.0.2 is available for download at the Oracle Technology Network. Contact us if you need help.

Oracle Loader for Hadoop

Download the kit from the Oracle Technology Network page:
http://www.oracle.com/technetwork/bdc/big-data-connectors/downloads/big-data-downloads-1451048.html

Use Oracle Loader for Hadoop 3.0 or higher.

See Section 1.5 in the Oracle Big Data Connectors User's Guide for installation instructions, follow links from http://www.oracle.com/technetwork/database/bigdata-appliance/documentation/bigdata-1454976.html

As described in the documentation this is installed on a node from which you submit MapReduce jobs.

Oracle SQL Connector for HDFS

Download the kit from the Oracle Technology Network page:
http://www.oracle.com/technetwork/bdc/big-data-connectors/downloads/big-data-downloads-1451048.html

Use Oracle SQL Connector for HDFS 3.0 or higher.

See Section 1.4 in the Oracle Big Data Connectors User's Guide for installation instructions, follow links from http://www.oracle.com/technetwork/database/bigdata-appliance/documentation/bigdata-1454976.html

As described in the documentation, Oracle SQL Connector for HDFS must be installed and configured on the database node. Additionally, it is recommended that you install and configure Oracle SQL Connector for HDFS on a system configured as Hadoop client. This is necessary if accessing Hive tables.

Also as described in the documentation, a Hadoop client (minimally the HDFS client) has to be installed on the database node.

Set Required Environment Variables

The following environment variables are required. These are described in the installation instructions in the Oracle Big Data Connectors User's Guide, they are repeated here for emphasis.

For Oracle Loader for Hadoop, in the Hadoop node environment (where you will submit Oracle Loader for Hadoop jobs):

#Set OLH_HOME to the directory where Oracle Loader for Hadoop is installed.

For example, set `$OLH_HOME` to `/u01/connector/loader/oraloader-3.3.0-h2`

(Change version number depending on your installation)

##Add to HADOOP_CLASSPATH
`${OLH_HOME}/jlib/*`

For Oracle SQL Connector for HDFS, in the Hadoop node environment and the database node environment:

#Set OSCH_HOME to the directory where Oracle SQL Connector for HDFS is installed.

For example, set `$OSCH_HOME` to `/u01/connector/hdfs/orahdfs-3.2.0`

(Change version number depending on your installation)

##Add to HADOOP_CLASSPATH
`${OSCH_HOME}/jlib/*`

If accessing Hive Tables using OLH and OSCH, also add the following to HADOOP_CLASSPATH
`${HIVE_HOME}/lib/*`
`${HIVE_HOME}/conf`

Setup and Cleanup

Examples Scripts and Data

The kit contains setup scripts, example scripts, and sample data. In the Big Data Lite VM unzip example_scripts.zip to `/home/oracle/movie/moviework`. This will contain `osch/`, `olh/`, `data/` and `setup/` directories. Setup and reset scripts are located in `setup/`

Data files are located in `data/`

All examples for Oracle SQL Connector for HDFS are located in `osch/`

All examples for Oracle Loader for Hadoop are located in `olh/`

Note: If you are not using the VM, then you should edit the examples accordingly to change NFS location, HDFS directory, and database connection information.

Cleanup for Re-run

Example scripts can be repeated by cleaning up directories and files from previous runs by running `reset_conn.sh`.

Ignore warning and error messages while running this script. They are messages indicating an object does not exist (which would be the case if an example has not been run).

Data

There are five data files in the HDFS directory `/user/oracle/data`. The data contains data of type string, integer, float, and date. A sample from file part-00002 is below:

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.)

Definition of Terms

Configuration parameters: These properties are used by the connectors during execution of Oracle SQL Connector for HDFS and Oracle Loader for Hadoop. They can be specified in an XML file or on the command line (using -D). Examples of configuration parameters are the locations 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 URIs of the data files on HDFS.

Part 1: Oracle SQL Connector for HDFS

Oracle SQL Connector for HDFS enables Oracle SQL access to data in Hadoop, via external tables. The data can be in a Hive table, text files in HDFS, or Oracle Data Pump files on HDFS.

Oracle SQL Connector for HDFS creates database objects to access data in Hadoop. Querying these objects will access data in Hadoop. When the data is in text files, non-partitioned Hive tables and Oracle Data Pump files, Oracle SQL Connector for HDFS creates a single external table to access the data. When the data is in partitioned Hive tables Oracle SQL Connector for HDFS creates multiple database objects - one external table and one database view for each partition. The external tables map

to the data columns in the Hive table (without the partition column values), and database views map to the data columns plus the partition column values. An application will use external tables to query text files, non-partitioned Hive tables and Oracle Data Pump files, and database views to query partitioned Hive tables.

External tables and views can be queried with full Oracle SQL functionality. They cannot be updated.

There are performance speedups while querying a Hive partitioned table since only the required partitions are queried. For example, consider a Hive table containing monthly sales data partitioned by date. An application that needs sales data for the 15th can query the view that corresponds to that partition.

Example

A Hive table is a monthly sales table partitioned by date. There are 30 partitions in the Hive table, one for each day in the month.

Oracle SQL Connector for HDFS will create 30 external tables and 30 database views, one for each partition. It will also create a metadata table that contains information about the external tables and views to identify which view to query.

Load

Oracle SQL can be used to query the external table or database view and insert into tables in the database for high speed load.

Specifying Hive Table Partitions

External tables and views are created for only selected partitions.

Partitions of interest are specified using a HiveQL predicate using the property `oracle.hadoop.exttab.partitionFilter`. Note that the predicate **can only contain partition columns**. Predicates with other column values are not supported, and can result in unexpected results.

When this property is not specified external tables and views are created for all partitions of the Hive table.

Using the Connector

There are two steps while using this connector.

- The command-line tool creates the database objects. This step needs access to HDFS and if accessing Hive tables, to the Hive metastore. So it is recommended to run this step on a Hadoop node. (If HDFS and Hive clients are installed on the database node then this step can be run on that node if necessary.)
- Querying data from the database, by querying the created database objects with Oracle SQL.

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

In this section we access text data files in HDFS from Oracle Database via external tables. The VM is single-node, so both Hadoop and Oracle Database are on the same node. In a multi-node system you will run Step 1 on Hadoop, and Step 2 (querying the data) from Oracle Database.

Step 1: Create the external table in Oracle Database to access files in HDFS.

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

Execute the script:

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

You will be prompted for the password for `moviedemo` (this is `welcome1`).

or (run the script directly from the command line):

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

Step 2: From the database

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 in HDFS.

The data can now be queried by the database via the external table.

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

```
prompt> sqlplus moviedemo@orcl/welcome1
SQL> describe movie_fact_ext_tab_file;
```

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

```
prompt> sqlplus moviedemo@orcl/welcome1
SQL> select count(*) from movie_fact_ext_tab_file;
  COUNT(*)
----------
    300025
```

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

You can try joining this table with tables in Oracle Database.

Notes:

This example used Oracle SQL Connector for HDFS with the `-createTable` option. This creates the external table and populates the location files in the external table LOCATION clause with the URIs of the data files on HDFS.

Take a look at the configuration parameters in `moviefact_text.xml`.

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

Some of the parameters are:

- The name of the table that we want to create in Oracle Database to access the files on HDFS
- Schema containing the table
- Location of the data files in HDFS
- Database connection information
- `locationFileCount=2`
- `oracle.hadoop.exttab.sourceType=text`
- `oracle.hadoop.exttab.columnNames` (a list of column names the external table should use)

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

Access data in a non-partitioned Hive table from Oracle Database

This example uses a Hive table created as part of the setup.

Step 1: Creates the external table in Oracle Database to access this Hive table

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

Execute the script:

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

You will be prompted for the password for `moviedemo` (this is `welcome1`).

or (run the script directly from the command line):

```
prompt> hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
       oracle.hadoop.exttab.ExternalTable \
```

```
       -conf /home/oracle/movie/moviework/osch/moviefact_hive.xml
\
       -createTable
```

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 in HDFS that contain the data in the Hive table.

Step 2: From the database

The Hive table can now be queried by the database via the external table.

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

```
prompt> sqlplus moviedemo@orcl/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.

```
prompt> sqlplus moviedemo@orcl/welcome1
SQL> select count(*) from movie_fact_ext_tab_hive;
  COUNT(*)
----------
    6063
SQL> select custid from movie_fact_ext_tab_hive where rownum < 10;
```

You can try joining this table with tables in Oracle Database.

Notes:

This example used Oracle SQL Connector for HDFS with the `-createTable` option. This creates the external table and populates the location files in the LOCATION clause of the external table.

Take a look at the configuration parameters in `moviefact_hive.xml`.

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

Some of the parameters are:

- The name of the external table that we want to create in Oracle Database to access the Hive table
- Schema containing the table
- Name of the Hive table

- Database connection information
- `oracle.hadoop.exttab.sourceType=hive`

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

Access Select Hive Partitions from a Partitioned Hive Table

This example uses a partitioned Hive table created as part of the setup. The Hive table is partitioned by column `activity`.

Step 1: This step creates the database objects (external tables, views on the external table with additional partition columns, and a metadata table) in Oracle Database to access partitions in the Hive table.

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

Step 1b:

Execute the script:

```
prompt> sh genloc_moviefact_hivepart.sh
```

You will be prompted for the password for `moviedemo` (this is `welcome1`).

or (run the script directly from the command line):

```
prompt> hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
       oracle.hadoop.exttab.ExternalTable \
       -conf
/home/oracle/movie/moviework/osch/moviefact_hivepart.xml \
     -D oracle.hadoop.exttab.hive.partitionFilter='activity < 4' \
       -createTable
```

You can see a summary of the database objects created in the output on screen.

The property `oracle.hadoop.exttab.hive.partitionFilter` identifies the Hive partitions to access. `activity` is the Hive partition column. The value of this property identifies that Hive partitions with `activity < 4` are of interest.

Another difference is the value for the property `oracle.hadoop.exttab.tableName` in `moviefact_hivepart.xml`. In this example the value is the name of the metadata table that is created for partitioned Hive tables. (For non-partitioned tables this is the name of the external table).

```
<property>
<name>oracle.hadoop.exttab.tableName</name>
        <value>MOVIE_FACT_META_PART</value>
 </property>
```

Step 2: From the database

A specific Hive partition can be accessed by querying the associated database view.

You can look at the metadata table in SQLPlus. The name of the metadata table was specified in property `oracle.hadoop.exttab.tableName` in `moviefact_hivepart.xml`.

```
prompt> sqlplus moviedemo@orcl/welcome1
SQL> describe movie_fact_meta_part;
--------------------------------- -------- --------------
 VIEW_NAME                        NOT NULL VARCHAR2(30)
 EXT_TABLE_NAME                   NOT NULL VARCHAR2(30)
 HIVE_TABLE_NAME                  NOT NULL VARCHAR2(4000)
 HIVE_DB_NAME                     NOT NULL VARCHAR2(4000)
 HIVE_PART_FILTER                          VARCHAR2(4000)
 ACTIVITY                                  NUMBER(38)
```

The metadata table contains one row for each Hive partition. Note the last column `activity`. This contains the partition values associated with each partition. You can query this table to identify the view for the partition with `activity = 2`.

```
SQL> select view_name from movie_fact_meta_part where activity =
2;
VIEW_NAME
------------------------------
MOVIE_FACT_META_PART_1
```

Querying this view will query data in the Hive table.

```
prompt> sqlplus moviedemo@orcl/welcome1
SQL> select count(*) from movie_fact_meta_part_1;
```

```
  COUNT(*)
----------
    4390
SQL> select custid from movie_fact_meta_part_1 where rownum < 10;
```

You can try joining this table with tables in Oracle Database.

Notes:

This example used Oracle SQL Connector for HDFS with the `–createTable` option. This creates the database objects discussed above.

Take a look at the configuration parameters in `moviefact_hivepart.xml`.

```
prompt> more moviefact_hivepart.xml
```

Some of the parameters are:

- The **name of the metadata table** created in Oracle Database. This contains the names of the external tables and views, one each for each Hive partition
- The partition filter value specified using Hive QL
- Schema containing the database objects created
- Name of the Hive table
- Database connection information
- `oracle.hadoop.exttab.sourceType=hive`

Populate an Oracle Database table

The data in the Hive table or in text files in HDFS can be inserted into a database table using SQL.

```
SQL> create table osch_ora_local_tab_genre_id_2 as select * from
movie_fact_meta_part_1;
```

or

```
SQL> create table osch_ora_local_tab as select * from
movie_fact_ext_tab_file;
```

Partition Pruning with an UNION ALL view

You can create a UNION ALL view on the views generated by Oracle SQL Connector for HDFS. This enables partition pruning. The predicate of a query on the UNION ALL view automatically

determines which views (which relate to partitions) to query. Data in HDFS is accessed only from those partitions. So querying the view eliminates partitions not relevant to the query and reduces the volume of data accessed in HDFS, resulting in greatly improved query results while using Oracle SQL Connector for HDFS.

For convenience, Oracle SQL Connector for HDFS includes a sample script to create the UNION ALL view. The sample script is described in the documentation (Section 2.5.4.3).

Syntax:

```
SQL> @mkhive_unionall_view <metadata table name> <schema> <UNION
all viewname> <predicate>
```

Example:

```
SQL> @mkhive_unionall_view MOVIE_FACT_META_PART null
MOVIE_FACT_HIVEPART_UNIONALL null
```

After you create the view, the following queries automatically prune the partitions that are not relevant:

```
SQL> select count(*) from MOVIE_FACT_HIVEPART_UNIONALL where activity <>
1;

  COUNT(*)

----------

     4390


SQL> select count(*) from MOVIE_FACT_HIVEPART_UNIONALL where activity =
4;

  COUNT(*)

----------

        0
```

In the first example, the query only accesses data where `activity = 2` and `activity = 3` (the external tables and views were created for partitions where `activity = 1, 2` or `3`). It does not access data where `activity = 1`.

In the second example, `activity = 4` does not match the values in any view/partition. This query is extremely fast, as it returns the result without accessing any data.

Part 1d: Performance Tuning for Oracle SQL Connector for HDFS

Oracle SQL Connector for HDFS performance increases as the degree of parallelism increases. The heuristic for best performance is as follows:

Number of data files = Number of location files = DOP in the database.

To arrive at the best set of numbers, start with the maximum DOP in the database that is possible. Set the number of location files to be equal to the DOP. Then look at the number of data files. If the number of data files is less than the database DOP, try to increase the number of data files by splitting up the data files or by generating an increased number of data files in the upstream Hadoop job. The goal is to make the number of data files equal to the number of location files and DOP.

If that is not possible, the number of data files will determine the parallelism while using Oracle SQL Connector for HDFS.

This example uses a bucketed Hive table created as part of the setup. The bucketed Hive table was created to increase the number of data files. Assuming a database DOP of 8, this bucketed Hive table is a table over 8 data files.

The file moviefact_hivepartb.xml includes a property to set the number of location files to 8.

- `locationFileCount=8`

Step 1: Creates the external table in Oracle Database to access this Hive table

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

Execute the script:

```
prompt> sh genloc_moviefact_hivepartb.sh
```

You will be prompted for the password for `moviedemo` (this is `welcome1`).

or (run the script directly from the command line):

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

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 in HDFS that contain the data in the Hive table. You can see there are 8 location files, because we set `locationFileCount=8` and because the Hive table has 8 data files.

Step 2: From the database

The Hive table can now be queried by the database via the external table.

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

```
prompt> sqlplus moviedemo@orcl/welcome1

SQL> describe movie_fact_hive_b;
```

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

The use of the PARALLEL hint is critical to ensure correct use of database DOP. There is an example in `parallel_query.sql`

```
prompt> sqlplus moviedemo@orcl/welcome1

SQL> select /*+ PARALLEL(t,8) */ count(*) from movie_fact_hive_b
t;

  COUNT(*)
----------
    39716

SQL> select /*+ PARALLEL(t,8) */ custid from movie_fact_hive_b
where rownum < 10;
```

You can try joining this with tables in Oracle Database.

Notes:

This example used Oracle SQL Connector for HDFS with the `-createTable` option. This creates the external table and populates the location files in the LOCATION clause of the external table.

Take a look at the configuration parameters in `moviefact_hivepartb.xml`.

```
prompt> more moviefact_hivepartb.xml
```

Some of the parameters are:

- The name of the external table that we want to create in Oracle Database to access the Hive table
- Schema containing the table
- Name of the Hive table
- `locationFileCount=8`
- Database connection information
- `oracle.hadoop.exttab.sourceType=hive`

Part 2: Oracle Loader for Hadoop

This section contains examples using Oracle Loader for Hadoop to load data from Hadoop into a table in Oracle Database.

About Oracle Loader for Hadoop:

Oracle Loader for Hadoop runs on the Hadoop cluster to pre-process input 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 Oracle Data Pump files on HDFS.

The examples in this section use Oracle Loader for Hadoop in the *online mode*.

Oracle Loader for Hadoop can work with many different input formats. The examples here include delimited text input format, Hive input format, and regular expression input format to load from Apache log files.

Part 2a: Load delimited text files in HDFS with Oracle Loader for Hadoop

The target table `movie_sessions_tab` was created as part of the setup. This table is range partitioned on column `time_id`. If you would like to create it manually, you can do so as shown below.

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

prompt>  sqlplus moviedemo@orcl/welcome1

SQL> @moviesession.sql
```

Step 1

Submit a Oracle Loader for Hadoop job to load data into this table from text files.

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

prompt> sh runolh_session.sh
```

Notes: Examine the configuration parameters used in this example:

- `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.job.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 this is Tab, represented by its hex value.
- `mapreduce.job.outputformat.class`: Specifies the type of load. We specify here the value `OCIOutputFormat` to use the online direct path load option.
- `mapreduce.input.fileinputformat.inputdir`: Location of the input data files on HDFS.
- `mapreduce.output.fileoutputformat.outputdir`: Specifies the HDFS directory for the output files of the MapReduce job, such as the _SUCCESS and _log files.
- `oracle.hadoop.loader.loaderMap.targetTable`: Name of the target table we are loading into.
- `oracle.hadoop.loader.input.fieldNames`: Target table column names we are loading into.

Step 3: On the database node

After the Oracle Loader for Hadoop job completes check that the rows have been loaded.

```
prompt> sqlplus moviedemo@orcl/welcome1

SQL> select count(*) from movie_sessions_tab;

  COUNT(*)
----------
      4526
```

The data is now available for query.

Part 2b: Load Hive tables with Oracle Loader for Hadoop

The target table `movie_local_tab` was created as part of the setup. This table is range partitioned on column `time_id`. If you would like to create the table manually, you can do so as shown below.

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

prompt>  sqlplus moviedemo@orcl/welcome1
```

```
SQL> @movie_tab.sql
```

Step 1

Submit Oracle Loader for Hadoop to load data into this table from text files.

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

prompt> sh runolh_hive_part.sh
```

Notes: Examine the configuration parameters used in this example:

- `prompt> more olh_hive_part.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.job.inputformat.class`: Specifies the input format of the input data file (in this example the input data is a Hive table, so the value for this parameter is the class name `oracle.hadoop.loader.lib.input.HivetoAvroInputFormat`.)
  - `oracle.hadoop.loader.input.hive.databaseName`: Specifies the Hive database.
  - `oracle.hadoop.loader.input.hive.tableName`: Specifies the Hive database.
  - `mapreduce.job.outputformat.class`: Specifies the type of load. We specify here the value `OCIOutputFormat` to use the online direct path load option.
  - `mapreduce.output.fileoutputformat.outputdir`: Specifies the HDFS directory for the output files of the MapReduce job, such as the _SUCCESS and _log files.

Step 2: On the database After the Oracle Loader for Hadoop job completes check that the rows have been loaded.

```
prompt> sqlplus moviedemo@orcl/welcome1

SQL> select count(*) from movie_local_tab;

  COUNT(*)
----------
      10453
```

The data is now available for query.

Part 2c: Load from Apache Log Files Oracle Loader for Hadoop

The regular expression input format enables parsing and loading of data, so that select columns can be loaded directly from log files such as Apache log files.

The target table `movie_apache_combined_log` was created as part of the setup. This table is range partitioned on column `time_id`. If you would like to create the table manually, you can do so as shown below.

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

prompt>  sqlplus moviedemo@orcl/welcome1

SQL> @reg_exp.sql
```

Step 1

Submit Oracle Loader for Hadoop to load data into this table from text files.

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

prompt> sh reg_exp.sh
```

Notes: Examine the configuration parameters used in this example:

- `prompt> more reg_exp.xml`
    - `mapreduce.job.inputformat.class`: Specifies the input format of the input data file (in this example we use `oracle.hadoop.loader.lib.input.RegexInputFormat`.)
    - `oracle.hadoop.loader.input.regexPattern`: Specifies the regular expression pattern used to filter the input data file.
    - `mapreduce.job.outputformat.class`: Specifies the type of load. We specify here the value `OCIOutputFormat` to use the online direct path load option.
    - `mapreduce.output.fileoutputformat.outputdir`: Specifies the HDFS directory for the output files of the MapReduce job, such as the _SUCCESS and _log files.

Step 2: On the database After the Oracle Loader for Hadoop job completes check that the rows have been loaded.

```
prompt> sqlplus moviedemo@orcl/welcome1

SQL> select count(*) from movie_apache_combined_log;

  COUNT(*)
```

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

         8
```

The data is now available for query.

Conclusion

The examples in this document contain end-to-end examples for using the connectors Oracle Loader for Hadoop and Oracle SQL Connector for HDFS.

White Paper Title
May 2015
Author: Melliyal Annamalai

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together