Migrating Data

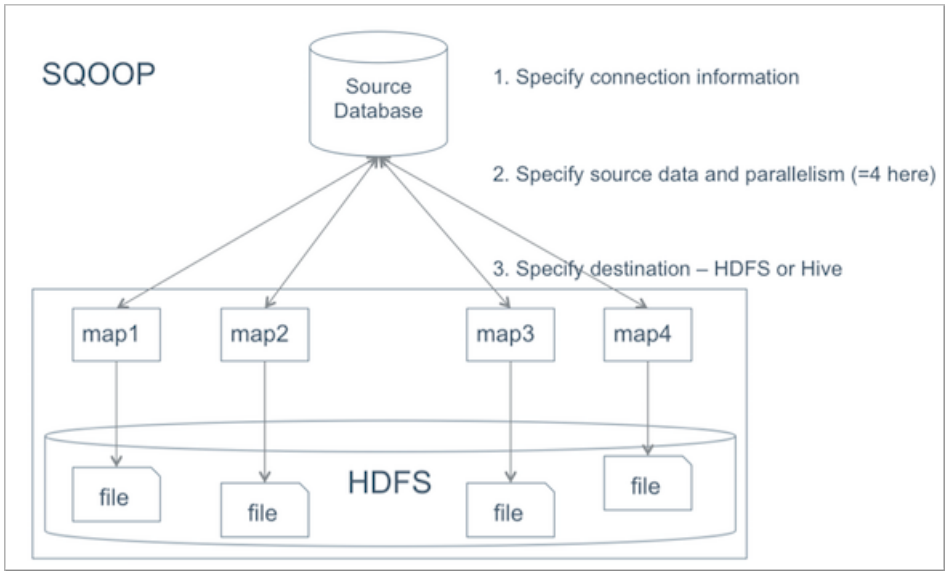
Date of Publish: 2019-04-15

https://docs.hortonworks.com/

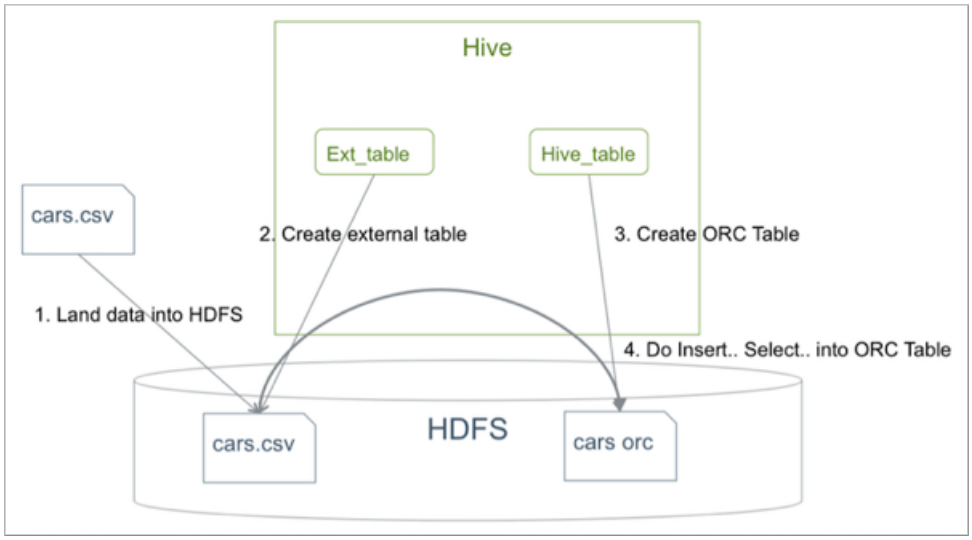
Contents

Data migration to Apache Hive

Sqoop is a tool for bulk importing and exporting data from diverse data sources to HDFS and Hive. The following diagram shows the process for moving data into Hive:



HDFS is typically the source of legacy system data that needs to undergo an extract, transform, and load (ETL) process. You can also import data in delimited text (default) or SequenceFile format, and then convert data to ORC format recommended for Hive. Generally, for querying the data in Hive, ORC is the preferred format because of the performance enhancements ORC provides. The following diagram shows an example of a common parallel and distributed conversion of data to ORC for querying in Hive:



Moving data from databases to Apache Hive

You can use Sqoop to import data from a relational database into HDFS for use with Hive or directly to Hive.

Related Information
Sqoop User Guide

Create a Sqoop import command

You create a single Sqoop import command that imports data from diverse data sources, such as a relational database, into Hive using Apache Sqoop.

About this task

You enter the Sqoop import command on the command line of your Hive cluster to import data from a data source into HDFS and Hive. The import can includes the following information, for example:

- Database connection information: database URI, database name, and connection protocol, such as jdbc:mysql:
- The data to import
- Parallel processing directives for performant data transfer
- Destination for imported data

Procedure

1. Create an import command that specifies the Sqoop connection to the RDBMS.
   - To enter a password for the data source on the command line, use the -P option in the connection string.
   - To specify a file where the password is stored, use the --password-file option.

   Password on command line:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
   <data to import> \
   --username <username> \
   -P
   ```

   Specify password file:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
   --table EMPLOYEES \
   --username <username> \
   --password-file ${user.home}/.password
   ```

   Note: Sqoop is tested to work with Connector/J 5.1. If you have upgraded to Connector/J 8.0, and want to use the zeroDateTimeBehavior property to handle values of '0000-00-00\' in DATE columns, explicitly specify zeroDateTimeBehavior=CONVERT_TO_NULL in the connection string. For example, jdbc:mysql://<MySQL host>/<DB>?zeroDateTimeBehavior=CONVERT_TO_NULL

2. Specify the data to import in the command.
   - Import an entire table.
   - Import a subset of the columns.
   - Import data using a free-form query.

   Entire table:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com:3306/bar \
   --table EMPLOYEES
   ```

Subset of columns:

```
sqoop import
--connect jdbc:mysql://db.foo.com:3306/bar \
--table EMPLOYEES \
--columns "employee_id,first_name,last_name,job_title"
```

Free-form query to import the latest data:

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3306/bar \
--table EMPLOYEES \
--where "start_date > '2018-01-01'"
```

3. Optionally, specify write parallelism in the import statement to execute a number of map tasks in parallel:
   - Set mappers: If the source table has a primary key, explicitly set the number of mappers using --num-mappers.
   - Split by: If primary keys are not evenly distributed, provide a split key using --split-by
   - Sequential: If you do not have a primary key or split key, import data sequentially using --num-mappers 1 or --autoreset-to-one-mapper in query.
   - Set mappers:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
      --table EMPLOYEES \
      --num-mappers 8 \
     ```

   - Split by:

     ```
     sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
      --table EMPLOYEES \
      --split-by dept_id
     ```

   - Setting mappers evenly splits the primary key range of the source table.
   - Split by evenly splits the data using the split key instead of a primary key.
4. Specify importing the data into Hive using Hive default delimiters by specifying the import option --hive-import.
5. Specify the Hive destination of the data.
   - If you think the table does not already exist in Hive, name the table using --hive-table <db>.<table_name> and use the --create-hive-table option.
   - If you want to insert the imported data into an existing Hive external table, name the table using --hive-table <db>.<table_name>. Do not use the --create-hive-table option.

   This command imports the MySQL EMPLOYEES table to a new Hive table named in the default HDFS location /user/hive/warehouse.

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
   --table EMPLOYEES \
   --hive-import \
   --create-hive-table \
   --hive-table mydb.newtable
   ```

   This command imports the MySQL EMPLOYEES table to an external table in HDFS.

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
   --table EMPLOYEES \
   --hive-import \
   ```

```
--hive-table mydb.myexternaltable
```

Related Information
Sqoop User Guide

Import RDBMS data into Hive

You can test the Apache Sqoop import command and then execute the command to import relational database tables into Hive.

About this task

You enter the Sqoop import command on the command line of your Hive cluster to import data from a data source to Hive. You can test the import statement before actually executing it.

Before you begin

- Apache Sqoop is installed and configured.
- A Hive Metastore is associated with your HDFS cluster.

Procedure

1. Optionally, test the import command before execution using the eval option.

   ```
   sqoop eval --connect jdbc:mysql://db.foo.com/bar \
   --query "SELECT * FROM employees LIMIT 10"
   ```

   The output of the select statement appears listing 10 rows of data from the RDBMS employees table.
2. Execute a Sqoop import command that specifies the Sqoop connection to the RDBMS, the data you want to import, and the destination Hive table name.
   This command imports the MySQL EMPLOYEES table to a new Hive table named in the default HDFS location /user/hive/warehouse.

   ```
   sqoop import --connect jdbc:mysql://db.foo.com:3306/corp \
   --table EMPLOYEES \
   --hive-import \
   --create-hive-table \
   --hive-table mydb.newtable
   ```

Related Information
Sqoop User Guide

Moving data from HDFS to Apache Hive

You can import data from diverse data sources into HDFS, perform ETL processes, and then query the data in Apache Hive.

Import RDBMS data to HDFS

You create a single Sqoop import command that imports data from diverse data sources, such as a relational database, into HDFS.

About this task

You enter the Sqoop import command on the command line of your cluster to import data from a data source into HDFS. In HDFS, you can perform ETL on the data, move the data into Hive, and query the data. The import command needs to include the database URI, database name, and connection protocol, such as jdbc:mysql: and the data to import. Optionally, the command can include parallel processing directives for performant data transfer, the HDFS destination directory for imported data, data delimiters, and other information. The default directory is used if you do not specify another location. Fields are comma-delimited and rows are line-delimited.You can test the import statement before actually executing it.

Before you begin

- Apache Sqoop is installed and configured.

Procedure

1. Create an import command that specifies the Sqoop connection to the data source you want to import.
   - If you want to enter a password for the data source on the command line, use the -P option in the connection string.
   - If you want to specify a file where the password is stored, use the --password-file option.

   Password on command line:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com/bar \
   <data to import> \
   --username <username> \
   -P
   ```

   Specify password file:

   ```
   sqoop import --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --username <username> \
   --password-file ${user.home}/.password
   ```

2. Specify the data to import in the command.
   - Import an entire table.
   - Import a subset of the columns.
   - Import data using a free-form query.

   Entire table:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES
   ```

   Subset of columns:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --columns "employee_id,first_name,last_name,job_title"
   ```

   Free-form query to import the latest data:

   ```
   sqoop import \
   --connect jdbc:mysql://db.foo.com/bar \
   --table EMPLOYEES \
   --where "start_date > '2018-01-01'"
   ```

3. Specify the destination of the imported data using the --target-dir option.
   This command appends data imported from the MySQL EMPLOYEES table to the output files in the HDFS target directory using default text file delimiters.

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--table EMPLOYEES \
--where "id > 100000" \
--target-dir /incremental_dataset \
--append
```

   This command splits imported data by column and specifies importing the data into output files in the HDFS target directory.

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--query 'SELECT a.*, b.* \
FROM a JOIN b on (a.id == b.id) \
WHERE $CONDITIONS' \
--split-by a.id \
--target-dir /user/foo/joinresults
```

   This command executes once and imports data serially using a single map task as specified by the -m 1 options:

```
sqoop import \
--connect jdbc:mysql://db.foo.com:3600/bar \
--query \
'SELECT a.*, b.* \
FROM a \
JOIN b on (a.id == b.id) \
WHERE $CONDITIONS' \
-m 1 \
--target-dir /user/foo/joinresults
```

4. Optionally, specify write parallelism in the import statement to execute a number of map tasks in parallel:
   - Set mappers: If the source table has a primary key, explicitly set the number of mappers using --num-mappers.
   - Split by: If primary keys are not evenly distributed, provide a split key using --split-by
   - Sequential: If you do not have a primary key or split key, import data sequentially using --num-mappers 1 or --autoreset-to-one-mapper in query.
   - Set mappers:

```
sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
--table EMPLOYEES \
--num-mappers 8
```

   - Split by:

```
sqoop import --connect jdbc:mysql://db.foo.com:3306/bar \
--table EMPLOYEES \
--split-by dept_id
```

   - Setting mappers evenly splits the primary key range of the source table.
   - Split by evenly splits the data using the split key instead of a primary key.

5. Optionally, test the import command before execution using the eval option.

```
sqoop eval --connect jdbc:mysql://db.foo.com:3306/bar \
--query "SELECT * FROM employees LIMIT 10"
```

The output of the select statement appears.

Related Information
Sqoop User Guide

Convert an HDFS file to ORC

To query data in HDFS in Hive, you apply a schema to the data and then store data in ORC format.

About this task

To convert data stored in HDFS into the recommended format for querying in Hive, you create a schema for the HDFS data by creating a Hive external table, and then create a Hive-managed table to convert and query the data in ORC format. The conversion is a parallel and distributed action, and no standalone ORC conversion tool is necessary. Suppose you have the following CSV file that contains a header line that describes the fields and subsequent lines that contain the data.

```
Name,Miles_per_Gallon,Cylinders,Displacement,Horsepower,Weight_in_lbs,Acceleration,Year,
"chevrolet chevelle malibu",18,8,307,130,3504,12,1970-01-01,A
"buick skylark 320",15,8,350,165,3693,11.5,1970-01-01,A
"plymouth satellite",18,8,318,150,3436,11,1970-01-01,A
"amc rebel sst",16,8,304,150,3433,12,1970-01-01,A
"ford torino",17,8,302,140,3449,10.5,1970-01-01,A
```

Before you begin

You removed the header from the CSV file.

Procedure

1. Create an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS Cars(
        Name STRING,
        Miles_per_Gallon INT,
        Cylinders INT,
        Displacement INT,
        Horsepower INT,
        Weight_in_lbs INT,
        Acceleration DECIMAL,
        Year DATE,
        Origin CHAR(1))
    COMMENT 'Data about cars from a public database'
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    location '/user/<username>/visdata';
```

2. Create a Hive-managed table to convert the data to ORC.

```
CREATE TABLE IF NOT EXISTS mycars(
Name STRING,
Miles_per_Gallon INT,
Cylinders INT,
Displacement INT,
```

```
Horsepower INT,
Weight_in_lbs INT,
Acceleration DECIMAL,
Year DATE,
Origin CHAR(1))
COMMENT 'Data about cars from a public database'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

3. Insert the data from the external table into the Hive-managed table.

```
INSERT OVERWRITE TABLE mycars SELECT * FROM cars;
```

4. Verify that you imported the data into the ORC-formatted table correctly:

```
hive> SELECT * FROM mycars LIMIT 3;
OK
"chevrolet chevelle malibu" 18 8 307 130 3504 12 1970-01-01 A
"buick skylark 320" 15 8 350 165 3693 12 1970-01-01 A
"plymouth satellite" 18 8 318 150 3436 11 1970-01-01 A
Time taken: 0.144 seconds, Fetched: 3 row(s)
```

Incrementally update an imported table

Updating imported tables involves importing incremental changes made to the original table using Sqoop and then merging changes with the tables imported into Hive.

About this task

After ingesting data from an operational database to Hive, you usually need to set up a process for periodically synchronizing the imported table with the operational database table. The base table is a Hive-managed table that was created during the first data ingestion. Incrementally updating Hive tables from operational database systems involves merging the base table and change records to reflect the latest record set. You create the incremental table as a Hive external table, typically from .CSV data in HDFS, to store the change records. This external table contains the changes (INSERTs and UPDATEs) from the operational database since the last data ingestion. Generally, the table is partitioned and only the latest partition is updated, making this process more efficient.

You can automate the steps to incrementally update data in Hive by using Oozie.

Before you begin

- The first time the data was ingested into hive, you stored entire base table in Hive in ORC format.
- The base table definition after moving it from the external table to a Hive-managed table has the following schema:

```
CREATE TABLE base_table (
id STRING,
field1 STRING,
modified_date DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Procedure

1. Store the incremental table as an external table in Hive and to fetch records newer than last_import_date, which is the date of the last incremental data update.

   You frequently import incremental changes since the last time data was updated and then merging it.

- using --check-column to fetch records
- use --query to fetch records

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
 --connection-manager org.apache.sqoop.teradata.TeradataConnManager
 --username dbc --password dbc --table SOURCE_TBL --target-dir /user/
hive/incremental_table -m 1 --check-column modified_date --incremental
 lastmodified --last-value {last_import_date}
```

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail --
connection-manager org.apache.sqoop.teradata.TeradataConnManager --
username dbc --password dbc --target-dir /user/hive/incremental_table
 -m 1 --query 'select * from SOURCE_TBL where modified_date >
 {last_import_date} AND $CONDITIONS'
```

2. After the incremental table data is moved into HDFS using Sqoop, you can define an external Hive table over it with the following command

```
CREATE EXTERNAL TABLE incremental_table (
         id STRING,
         field1 STRING,
         modified_date DATE)
         ROW FORMAT DELIMITED
         FIELDS TERMINATED BY ','
         STORED AS TEXTFILE
         location '/user/hive/incremental_table';
```

3. Use the MERGE command to merge the data and reconcile the base table records with the new records:

```
                  merge into base_table
                  using incremental_table on base.id =
 incremental_table.id
                  when matched then update set
                  fieldl1=incremental_table.email,
                  modified_date=incremental_table.state
                  when not matched then insert
                  values(incremental_table.id, incremental_table.field1,
 incremental_table.modified_data);
```

Hive import command options

A number Sqoop command options facilitate importing data into Hive.

Table 1: Sqoop Command Options for Importing Data into Hive

Sqoop Command Option	Description
--hive-home <directory>	Overrides $HIVE_HOME.
--hive-import	Imports tables into Hive using Hive's default delimiters if none are explicitly set.
--hive-overwrite	Overwrites existing data in the Hive table.

Sqoop Command Option	Description
--create-hive-table	Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to false by default.
--hive-table <table_name>	Specifies the table name to use when importing data into Hive.
--hive-drop-import-delims	Drops the delimiters \n, \r, and \01 from string fields when importing data into Hive.
--hive-delims-replacement	Replaces the delimiters \n, \r, and \01 from strings fields with a user-defined string when importing data into Hive.
--hive-partition-key	Specifies the name of the Hive field on which a sharded database is partitioned.
--hive-partition-value <value>	A string value that specifies the partition key for data imported into Hive.
--map-column-hive <map>	Overrides the default mapping from SQL type to Hive type for configured columns.