HiveQL - basic

IRAJ HEDAYATI

HiveQL

What is SQL dialect? Structured Query Language (SQL) is a programming language for Relational Database Management System (RDBMS) initially developed by IBM in 1970s. Since then, different dialects are introduced to improve the language such as SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016.

On the other hand, most of the commercial RDBMS also introduced their own dialect. For example PL/SQL from Oracle or T-SQL from Microsoft that are mainly extensions to SQL to enable procedural programming.

Hive has its own dialect that is called **HiveQL**. It is perhaps closest to MySQL's dialect, but with significant differences.

- No support for *INSERT*, *UPDATE* and *DELETE*
- No support for transactions
- Extensions for better performance in working with Hadoop

Database

Database in Hive is more a concept of *namespace*. It is for organizing the tables. If you don't specify a database, the **`default`** is used.

Create a database:

```
hive> CREATE DATABASE financials;
```

Hive will throw an error if database exists.

```
hive> CREATE DATABASE IF NOT EXISTS financials;
```

```
hive> SHOW DATABASES;
default
financials
```

Database - location

Hive creates a directory for each database under its **`warehouse`** folder specified by `hive.metastore.warehouse.dir` configuration key (default is `/user/hive/warehouse`).

```
# hdfs dfs -ls /user/hive/warehouse/
```

Tables will be stored in subdirectories.

It is possible to override the directory of database.

```
hive> CREATE DATABASE financials;
    > LOCATION '/user/root/financials';
hive> DESCRIBE DATABASE financials;
```

Note that you can also use relative path (that goes under your user’s directory on HDFS)

Database - workspace

By default, your working database is **`default`** unless you change it using **`USE`** command.

```
hive> USE financials;
```

There is no command to show what is current working database, but you can set Hive CLI to show you in the prompt.

```
hive> SET hive.cli.print.current.db=true;

hive (financials)>
```

Database - drop

Finally, you can drop a database

```
hive> DROP DATABASE IF EXISTS financials;
```

Note that `IF EXISTS` is optional.

You can’t drop a database if it contains tables. You need to tell Hive if it’s necessary.

```
hive> DROP DATABASE IF EXISTS financials CASCADE;
```

You also are able to alter a database. But, it only is applicable to properties.

Database properties

Provide database metadata using a custom key-value. It doesn’t have any functionality and just more information in describe database command.

```
hive> CREATE DATABASE financials > WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date'
= '2012-01-02'); hive> DESCRIBE DATABASE financials; financials hdfs://master-
server/user/hive/warehouse/financials.db hive> DESCRIBE DATABASE EXTENDED financials;
financials hdfs://master-server/user/hive/warehouse/financials.db {date=2012-01-02,
creator=Mark Moneybags);
```

(Page 51).

Table

HiveQL DDL follows SQL with more options in creating table.

Recap:

```
CREATE TABLE table_name (
 field_1_name field_1_type COMMENT 'field comment’,
 field_2_name field_2_type COMMENT 'field comment’,
 ...
)
COMMENT 'Table comment'
```

Data types

Hive supports most of the *primitive* data types that other relational databases have.

Unlike other relational databases, Hive doesn't tend to have full control over data file encodings.

Type	Size	Literal syntax examples
TINYINT	1 byte signed integer	20
SMALLINT	2 bytes signed integer	20
INT	4 bytes signed integer	20
BIGINT	8 bytes signed integer	20
BOOLEAN	Boolean true or false	TRUE
FLOAT	Single precision floating point	3.14159
DOUBLE	Double precision floating point	3.14159
STRING	Sequence of characters. The character set can be specified. Single or double quotes can be used.	'Now is the time', "for all good men"

Data types (v0.8.0+)

Type	Size	Literal syntax examples
TIMESTAMP	Integer, float, or string	1327882394 (Unix epoch seconds), 1327882394.123456789 (Unix epoch seconds plus nanoseconds), and '2012-02-03 12:34:56.123456789' (JDBCcompliant java.sql.Timestamp format)
BINARY	Array of bytes	

Casting

Implicit casting (automatically done by Hive)

```
TINYINT -> SMALLINT -> INT -> DOUBLE
```

```
FLOAT -> DOUBLE
```

Explicit casting (provided by user)

```
… cast (s AS INT) … ;
```

Table – STM GTFS data

For demonstration, let's create **Agency** table form STM GTFS data.

```
hive> CREATE DATABASE stm_gtfs;

hive> CREATE TABLE stm_gtfs.table_name (
    > agency_id STRING,
    > agency_name STRING,
    > agency_url STRING,
    > agency_timezone STRING,
    > agency_lang STRING,
    > agency_phone STRING,
    > agency_fare_url STRING
    > )
```

Table – related

You can have **`TBLPROPERTIES`** same as **`DBPROPERTIES`**

You can show list of tables in current database using **`SHOW TABLES`**

You can override default directory on HDFS using **`LOCATION`** (same as DATABASE)

You can copy the schema of table using **`CREATE TABLE table_2 LIKE table_1;`**

You can print table information using **`DESCRIBE table_1`** or get more information using **`DESCRIBE EXTENDED table_1.`** To get more readable information (but more verbose output), you can use **`DESCRIBE FORMATTED table_1`**

Table types in Hive

There are two types of tables in Hive:

- Managed
- External

Managed tables (those that we created so far) or also called *internal* tables are those controlled and managed by Hive. Hive has control on their lifecycles. If you drop a managed table, its data is also gone.

External tables on the other hand is good when you just need to run some queries but don't want to give the ownership of data to Hive.

External table

Assume you are analyzing STM GTFS data daily basis. Here is a possible ETL scenario:

1. Go to STM website and download the file (could be automated)
2. Extract the files
3. Put the files on HDFS
4. ???

One option is to implement a Java or Scala application to parse the data and use Hive JDBC to insert them into the table. The other option is to create an external table on top of the **directory** that you have just created with correct schema.

External table - Agency

Assume you have put **agency.txt** into /user/root/agency/ directory on HDFS.

```
hive> USE stm_gtfs;

hive> CREATE EXTERNAL TABLE stm_gtfs.table_name (
    > agency_id STRING,
    > agency_name STRING,
    > agency_url STRING,
    > agency_timezone STRING,
    > agency_lang STRING,
    > agency_phone STRING,
    > agency_fare_url STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ‘,’
    > STORED AS TEXTFILE
    > LOCATION ‘/user/root/agency’;
```

Alter table

Rename

```
ALTER TABLE log_messages RENAME TO logmsgs;
```

Add partition

```
ALTER TABLE log_messages ADD IF NOT EXISTS
PARTITION (year = 2011, month = 1, day = 1) LOCATION '/logs/2011/01/01'
PARTITION (year = 2011, month = 1, day = 2) LOCATION '/logs/2011/01/02'
```

Drop partition

```
ALTER TABLE log_messages DROP IF EXISTS
PARTITION(year = 2011, month = 12, day = 2);
```

You can also add, change and drop a column

Join restriction

There is nothing special in DML part. Just some consideration in joins.

Hive only supports *equi-join*.

Wikipedia:

- An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

An *equi-join*

```
SELECT * FROM orders o JOIN customers c ON o.cust_id = c.cust_id;
```

A *non-equi-join*

```
SELECT * FROM orders o  JOIN payments p ON o.date > p.date;
```

Note that Hive only supports `OR` in `ON` clauses.

Join using multiple jobs

For joins that more than two tables are involved, Hive uses one MapReduce job for each.

```
SELECT *
FROM orders o JOIN customers c ON o.cust_id=c.cust_id
              JOIN employees e ON o.emp_id=e.emp_id;
```

Here, it would use one MapReduce job to join `o` and `c` and one MapReduce job to join `e` and `o` and will do it from left to right.

Join optimization - Join on same key

When joining three or more tables, if every `ON` clause uses the same join key, a single MapReduce job will be used.

```
SELECT *
FROM orders o JOIN customers c ON o.cust_id=c.cust_id
              JOIN payments p ON p.cust_id=c.cust_id;
```

In above query, the same key of `c.cust_id` is used in both `ON` clauses. Hence, there would be only one MapReduce job.

Join optimization – streaming hint

It is also possible to give Hive a hint instead of trying to work on order of tables.

```
SELECT /*+ STREAMABLE(o) */ *
FROM orders o JOIN customers c ON o.cust_id=c.cust_id
              JOIN payments p ON p.cust_id=c.cust_id;
```

In this example, `orders` is the largest one. In the order it comes first but using the hint of `STREAMABLE`, we tell Hive that it is the largest table and has to be streamed.

Join optimization – map-side join

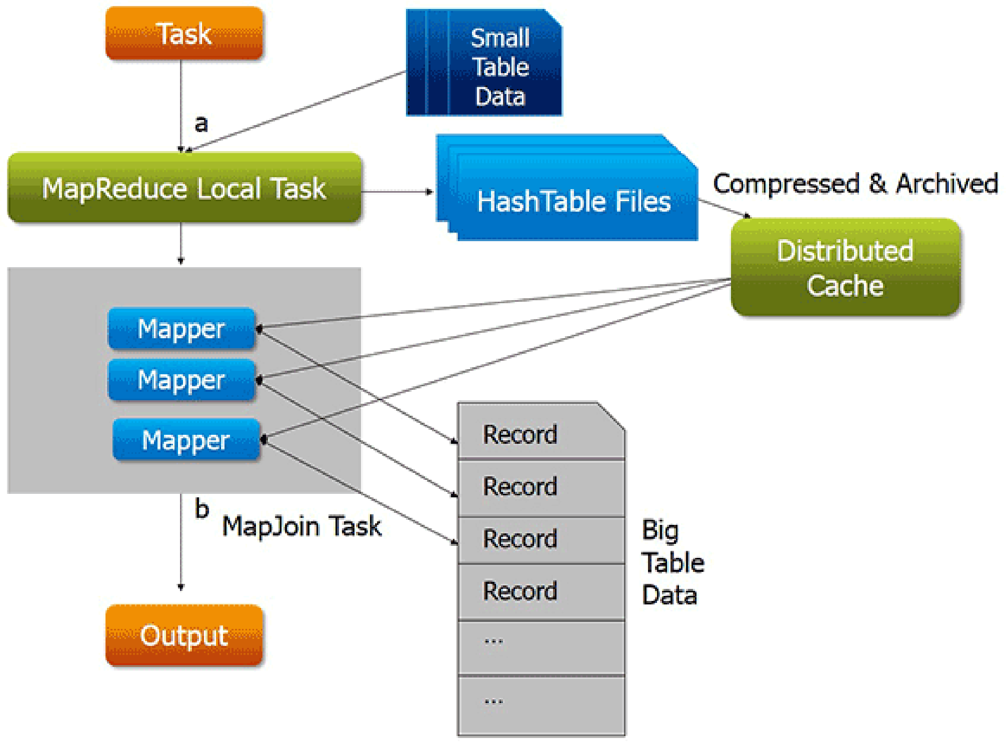
One of the ways to optimize JOIN is to cache small tables in the memory and then stream the largest table in the Map side of a MapReduce job. It eliminates reduce phase. Hive assumes that the *last* table in the query is the largest. (Note: try to always put the largest table at the end.)

In order to enable this feature:

```
SET hive.auto.convert.join=true
```

You also are able to change the definition of small table:

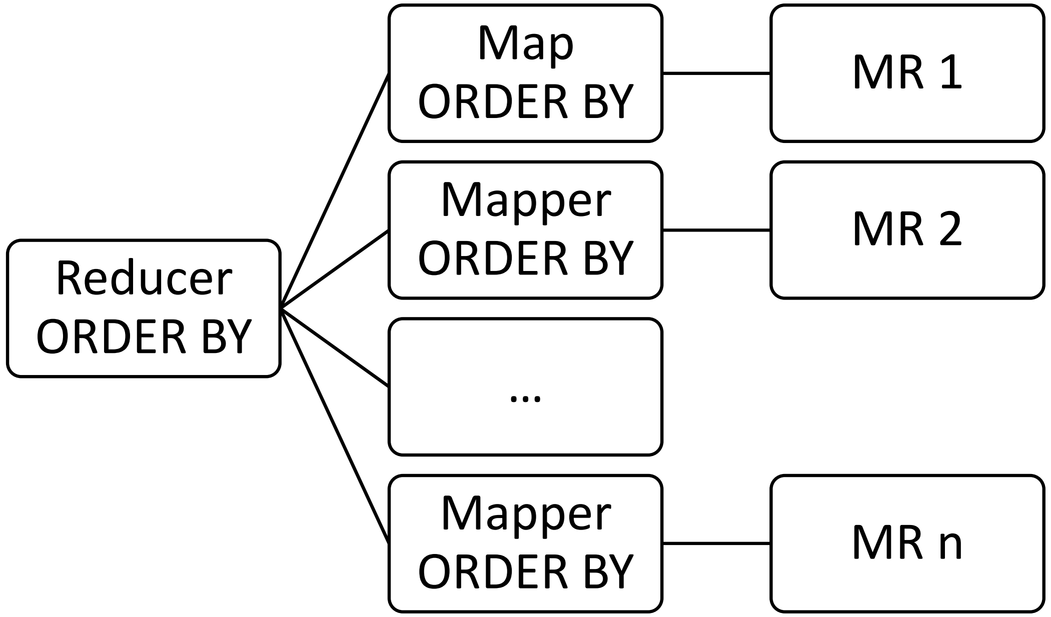
```
SET hive.mapjoin.smalltable.filesize=25000000
```



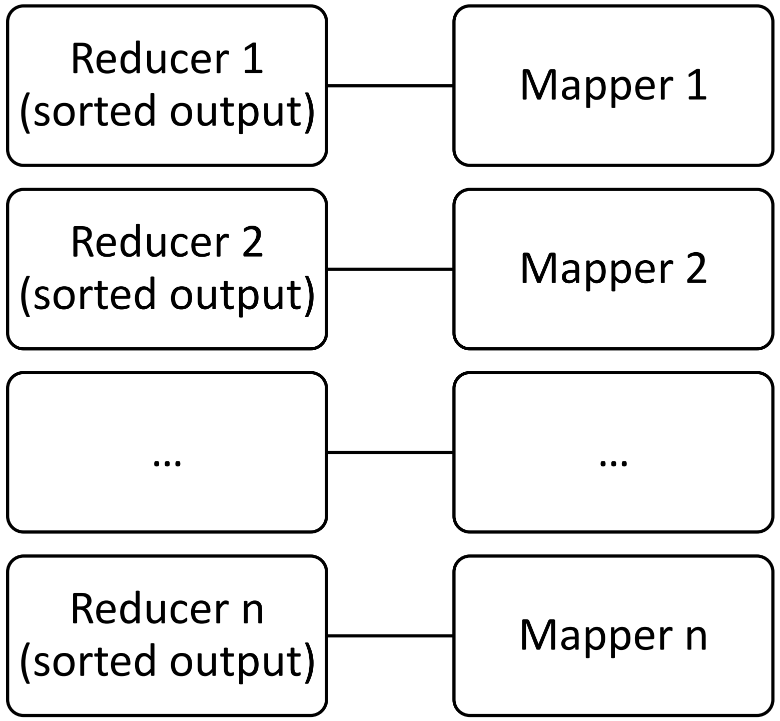
Ref.: https://www.edureka.co/blog/map-side-join-vs-join/

ORDER BY and SORT BY

ORDER BY (ASSUME ONE JOB DEDICATED FOR)



SORT BY


Reducer 1 (sorted output)
Mapper 1
Reducer 2 (sorted output)
Mapper 2
…
…
Reducer n (sorted output)
Mapper n

ORDER BY and SORT BY

Using ORDER BY takes considerably long time as all the records has to pass through one reducer

```
SELECT * FROM orders ORDER BY order_id;
```

Using SORT BY is much faster but it won't give us a *total ordering*

```
SELECT * FROM orders SORT BY order_id;
```

By default, if `hive.mapred.mode=strict`, Hive requires a `LIMIT` clause if you use `ORDER BY`.

NOTE: as you see the configuration key hive.mapred.mode has quite a lot of impact everywhere

References

"Programming Hive", Dean Wampler, Jason Rutherglen, Edward Capriolo