Spring,2015

Apache Hive

BY NATIA MAMAIASHVILI , LASHA AMASHUKELI & ALEKO CHAKHVASHVILI

SUPERVAIZOR: PROF. NODAR MOMTSELIDZE

Contents:

- Briefly About Big Data Management
- What is hive?
- Hive Architecture
- Working of Hive
- Installation
- Data Types
- Creating Database, table…etc
- Partitioning
- Built-in Operators and Functions
- Queries
- Differences Between HiveQL and SQL
- Example of World Database

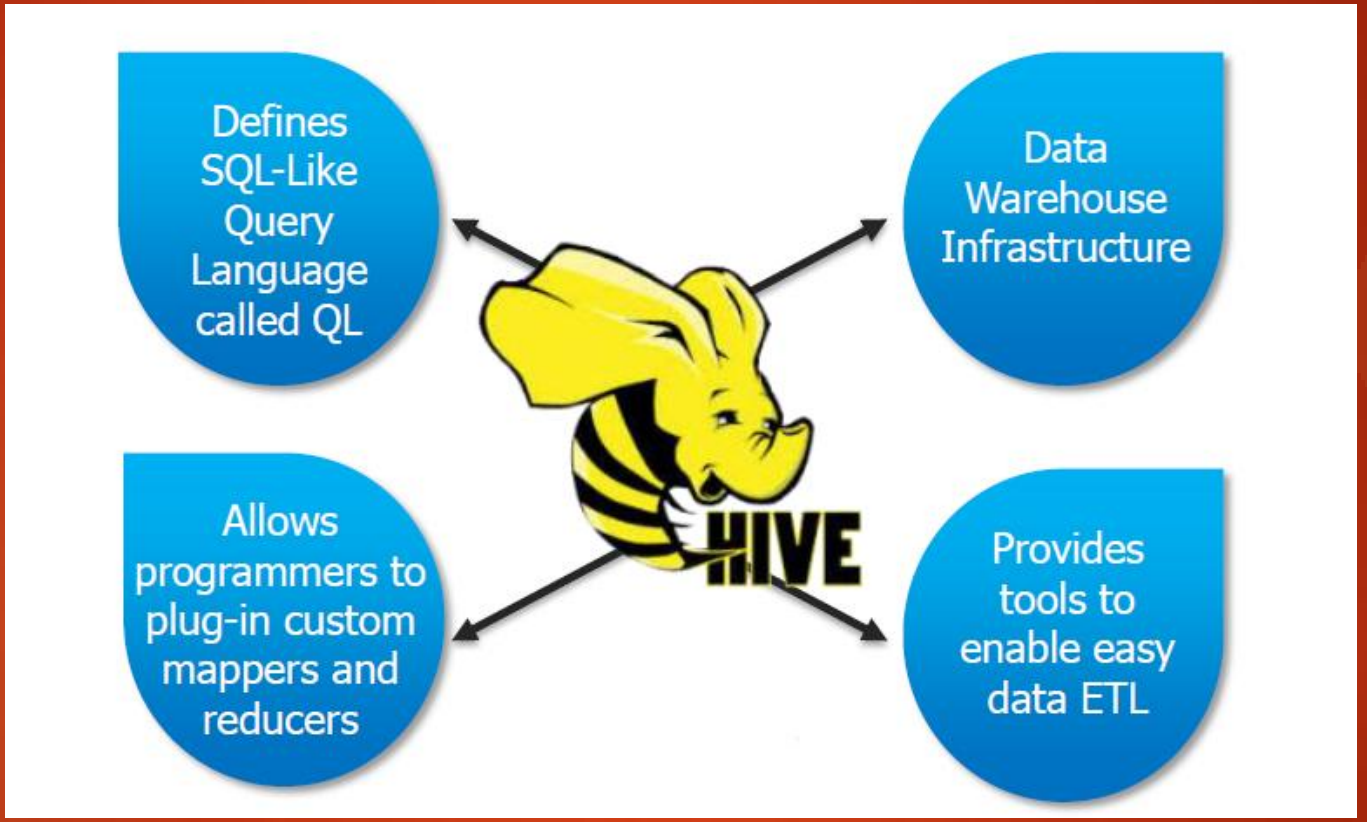
Briefly About Big Data Management

- Apache Software Foundation introduced a framework called Hadoop to solve Big Data management and processing challenges.
- It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).
- The Hadoop ecosystem contains different sub-projects(tools) one of them is Hive, which is a platform used to develop a script for MapReduce operations.

What is Hive?

FEATURES:

- It stores schema in a database and processed data into HDFS.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

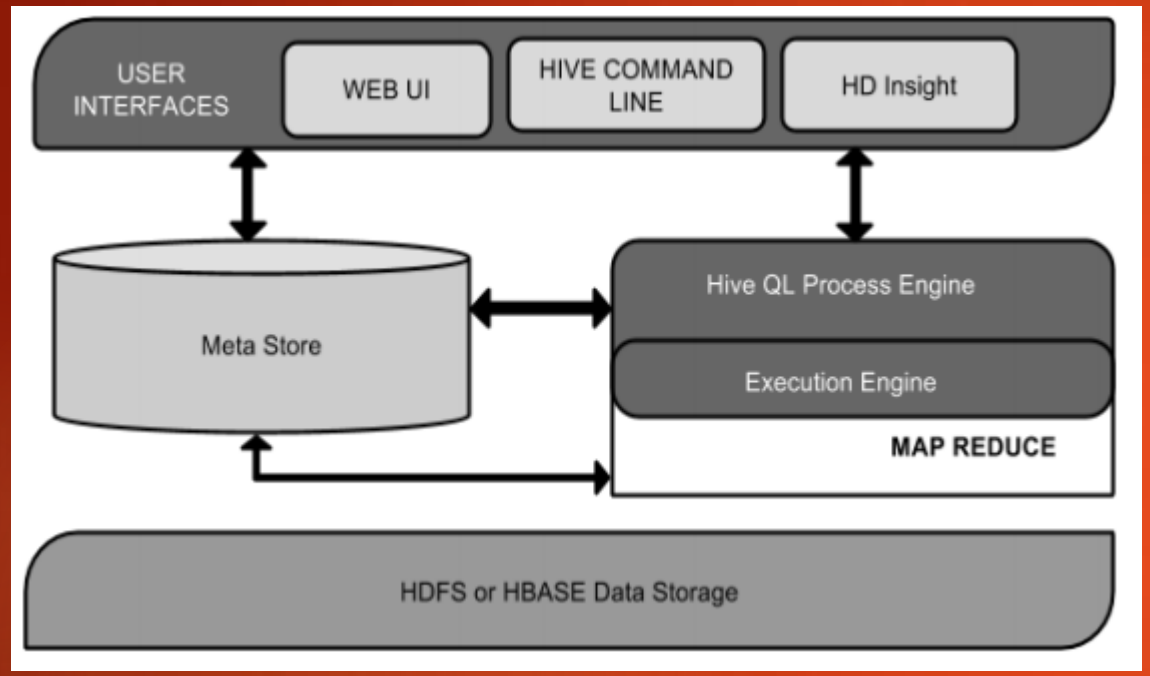


- Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Hive is not :

- A relational database
- A design for Online Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Hive Architecture



HDFS or HBASE -Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

UI - interaction between user and HDFS; Supports WEB UI HCL and HD Insight (for Win Server

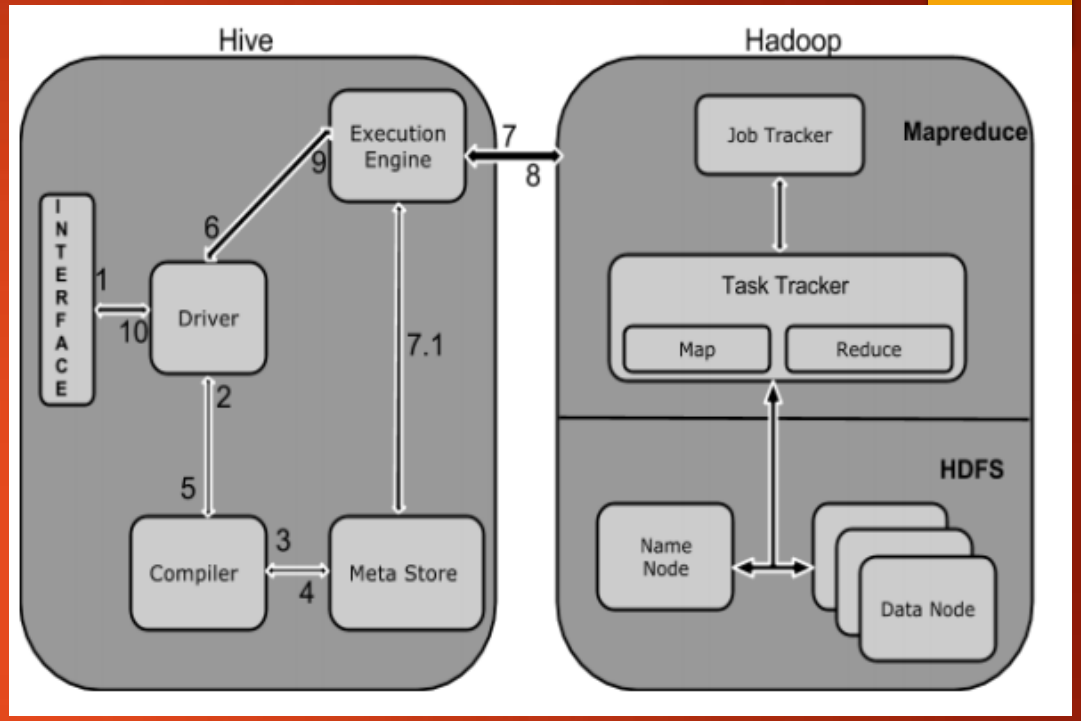
MetaStore – place to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.

HiveQL Process Engine - Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

Execution Engine - The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.

Working of Hive

- Executing Query - The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
- Get Plan - The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
- Get Metadata - The compiler sends metadata request to Metastore (any database).
- Send Metadata - Metastore sends metadata as a response to the compiler.
- Send Plan - The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
- Execute Plan - The driver sends the execute plan to the execution engine.
- Execute Job(Mapreduce) -The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Query executes MapReduce job.



- Metadata Ops Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
- Fetch Result The execution engine receives the results from Data nodes.
- Sending The execution engine sends those resultant values to the driver.
- Sending The driver sends the results to Hive Interfaces.

Installation

- Step 1: Installing VM, which runs VMWare or VirtualBox;
- Step 2: Installing Java: as Hive requires it;
- Step 3: Ensuring that Java is in your path and the JAVA_HOME environment variable is set.
- On Linux systems, the following instructions set up a bash file in the /etc/profile.d/ directory that defines JAVA_HOME for all users

```
$ /usr/java/latest/bin/java -version
java version "1.6.0_23"
Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
Java HotSpot(TM) 64-Bit Server VM (build 19.0-b09, mixed mode)
$ sudo echo "export JAVA_HOME=/usr/java/latest" > /etc/profile.d/java.sh
$ sudo echo "PATH=$PATH:$JAVA_HOME/bin" >> /etc/profile.d/java.sh
$ . /etc/profile
$ echo $JAVA_HOME
/usr/java/latest
```

- Step 4: Download and extract tarball for hive

```
$ cd ~                    # or use another directory of your choice.
$ curl -o http://archive.apache.org/dist/hive/hive-0.9.0/hive-0.9.0-bin.tar.gz
$ tar -xzf hive-0.9.0.tar.gz
$ sudo mkdir -p /user/hive/warehouse
$ sudo chmod a+rwx /user/hive/warehouse
```

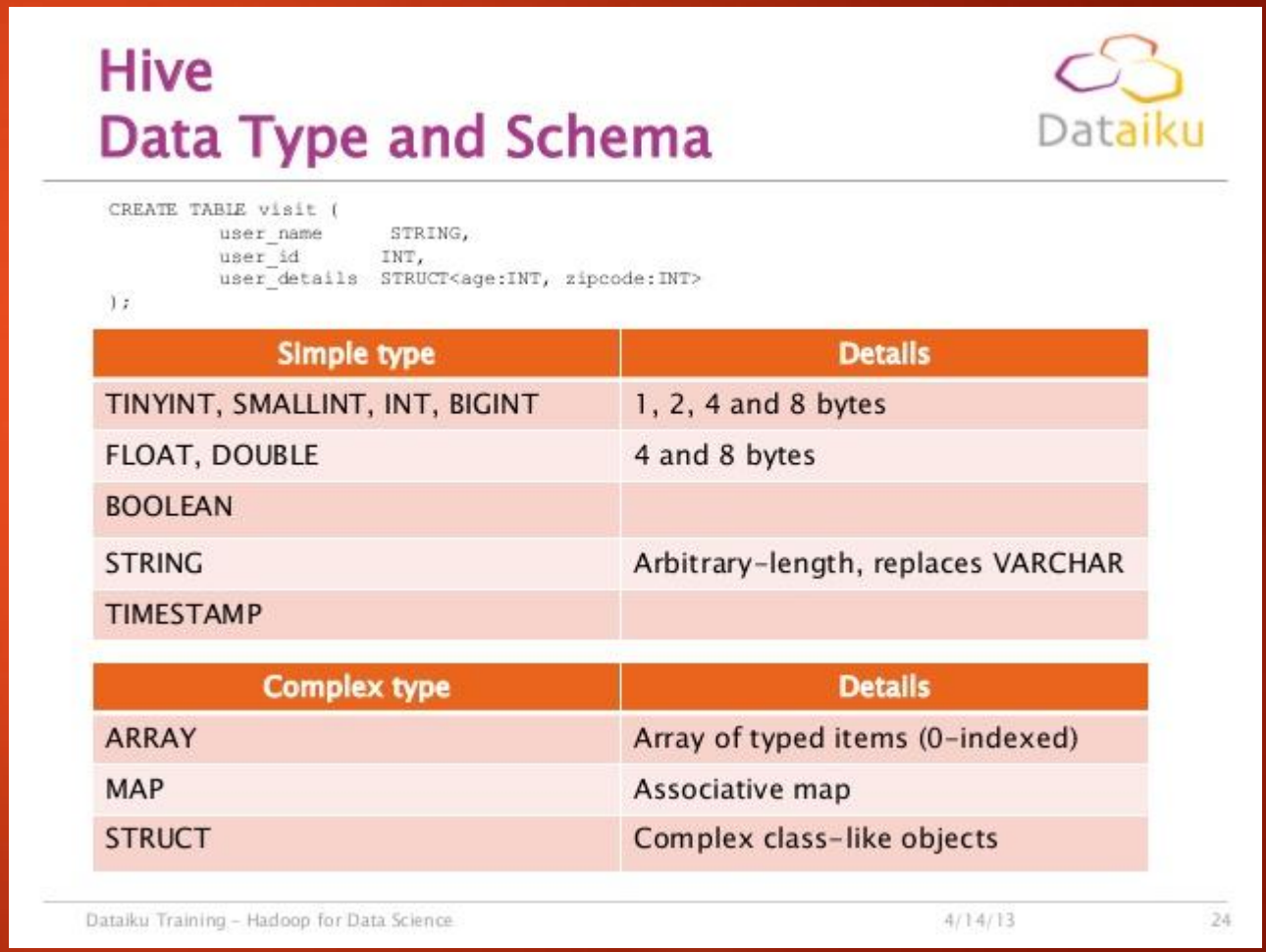
Data Types

Primitive types

- Integers: TINYINT, SMALLINT, INT, BIGINT.
- Boolean: BOOLEAN.
- Floating point numbers: FLOAT(Implemented By Java Float), DOUBLE
- String: STRING (Implemented By Java String)
- Char: Varchar,Char
- Timestamp,date…

Complex Types

- ARRAY<data_type>
- MAP<primitive_type, data_type>
- STRUCT<col_name : data_type [COMMENT col_comment], ...>



Databases in Hive

- It is a catalog or namespace of tables. Used for avoiding table name collisions.
- SYNTAX: hive>CREATE DATABASE <name>;

you can see the databases that already exist as follows:

- hive> SHOW DATABASES;

setting a database as your working database:

- hive> USE <name>;

If not specified, the default database is used.

- Create table structure and syntax:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]
table_name

[(col_name data_type [COMMENT col_comment], ...)]

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]
```

Loading Data:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

- We can also drop and alter databases and tables.
- When altering tables we can:
  - Replace columns
  - Add columns
  - Change table name
  - Change column name

Partitioning

- Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns.
- Tables or partitions are sub-divided into buckets, to provide extra structure to the data that may be used for more efficient querying.

/tab1/employeedata/file1

id, name, dept, yoj

1, gopal, TP, 2012

2, kiran, HR, 2012

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

The above data is partitioned into two files using year.

/tab1/employeedata/2012/file2

1, gopal, TP, 2012

2, kiran, HR, 2012

/tab1/employeedata/2013/file3

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

Built-in Operators and Functions

Operator	Operand	Description
A = B	all primitive types	TRUE if expression A is equivalent to expression B otherwise FALSE.
A != B	all primitive types	TRUE if expression A is *not* equivalent to expression B otherwise FALSE.
A < B	all primitive types	TRUE if expression A is less than expression B otherwise FALSE.
A <= B	all primitive types	TRUE if expression A is less than or equal to expression B otherwise FALSE.
A > B	all primitive types	TRUE if expression A is greater than expression B otherwise FALSE.
A >= B	all primitive types	TRUE if expression A is greater than or equal to expression B otherwise FALSE.
A IS NULL	all types	TRUE if expression A evaluates to NULL otherwise FALSE.
A IS NOT NULL	all types	FALSE if expression A evaluates to NULL otherwise TRUE.

- There are four types of operators in Hive:
  - 1. Relational Operators
  - 2. Arithmetic Operators
  - 3. Logical Operators
  - 4. Complex Operators

Operator	Operand	Description
A[n]	A is an Array and n is an int	It returns the nth element in the array A. The first element has index 0.
M[key]	M is a Map<K, V> and key has type K	It returns the value corresponding to the key in the map.
S.x	S is a struct	It returns the x field of S.

A LIKE B	Strings	TRUE if string pattern A matches to B otherwise FALSE.
A RLIKE B	Strings	NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B , otherwise FALSE.
A REGEXP B	Strings	Same as RLIKE.

Operators	Operand	Description
A + B	all number types	Gives the result of adding A and B.
A - B	all number types	Gives the result of subtracting B from A.
A * B	all number types	Gives the result of multiplying A and B.
A / B	all number types	Gives the result of dividing B from A.
A % B	all number types	Gives the reminder resulting from dividing A by B.
A & B	all number types	Gives the result of bitwise AND of A and B.
A | B	all number types	Gives the result of bitwise OR of A and B.
A ^ B	all number types	Gives the result of bitwise XOR of A and B.

Operators	Operands	Description
A AND B	boolean	TRUE if both A and B are TRUE, otherwise FALSE.
A && B	boolean	Same as A AND B.
A OR B	boolean	TRUE if either A or B or both are TRUE, otherwise FALSE.
A || B	boolean	Same as A OR B.
NOT A	boolean	TRUE if A is FALSE, otherwise FALSE.
!A	boolean	Same as NOT A.

Built-in Functions

- Some built-in functions:

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.
BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
double	rand(), rand(int seed)	It returns a random number that changes from row to row.
string	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.
string	substr(string A, int start)	It returns the substring of A starting from start position till the end of string A.
string	substr(string A, int start, int length)	It returns the substring of A starting from start position with the given length.
string	upper(string A)	It returns the string resulting from converting all characters of A to upper case.
string	ucase(string A)	Same as above.
string	lower(string A)	It returns the string resulting from converting all characters of B to lower case.
string	lcase(string A)	Same as above.
string	trim(string A)	It returns the string resulting from trimming spaces from both ends of A.
string	ltrim(string A)	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	rtrim(string A)	It returns the string resulting from trimming spaces from the end (right hand side) of A.
string	regexp_replace(string A, string B, string C)	It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.
int	size(Map<K.V>)	It returns the number of elements in the map type.
int	size(Array<T>)	It returns the number of elements in the array type.
value of <type>	cast(*<expr>* as *<type>*)	It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.
string	from_unixtime(int unixtime)	convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00"
string	to_date(string timestamp)	It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"
int	year(string date)	It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
int	month(string date)	It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
int	day(string date)	It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.

Queries

- Select … Where
- Select Order By
- Nested Select etc.
- Group By
- Joins (JOIN, LEFT OUTER JOIN,RIGHT OUTER JOIN,FULL OUTER JOIN)
- LIKE and RLIKE
- Limit Clause

```
hive> SELECT avg(price_close)
    > FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL';
```

```
hive> SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd);
```

```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

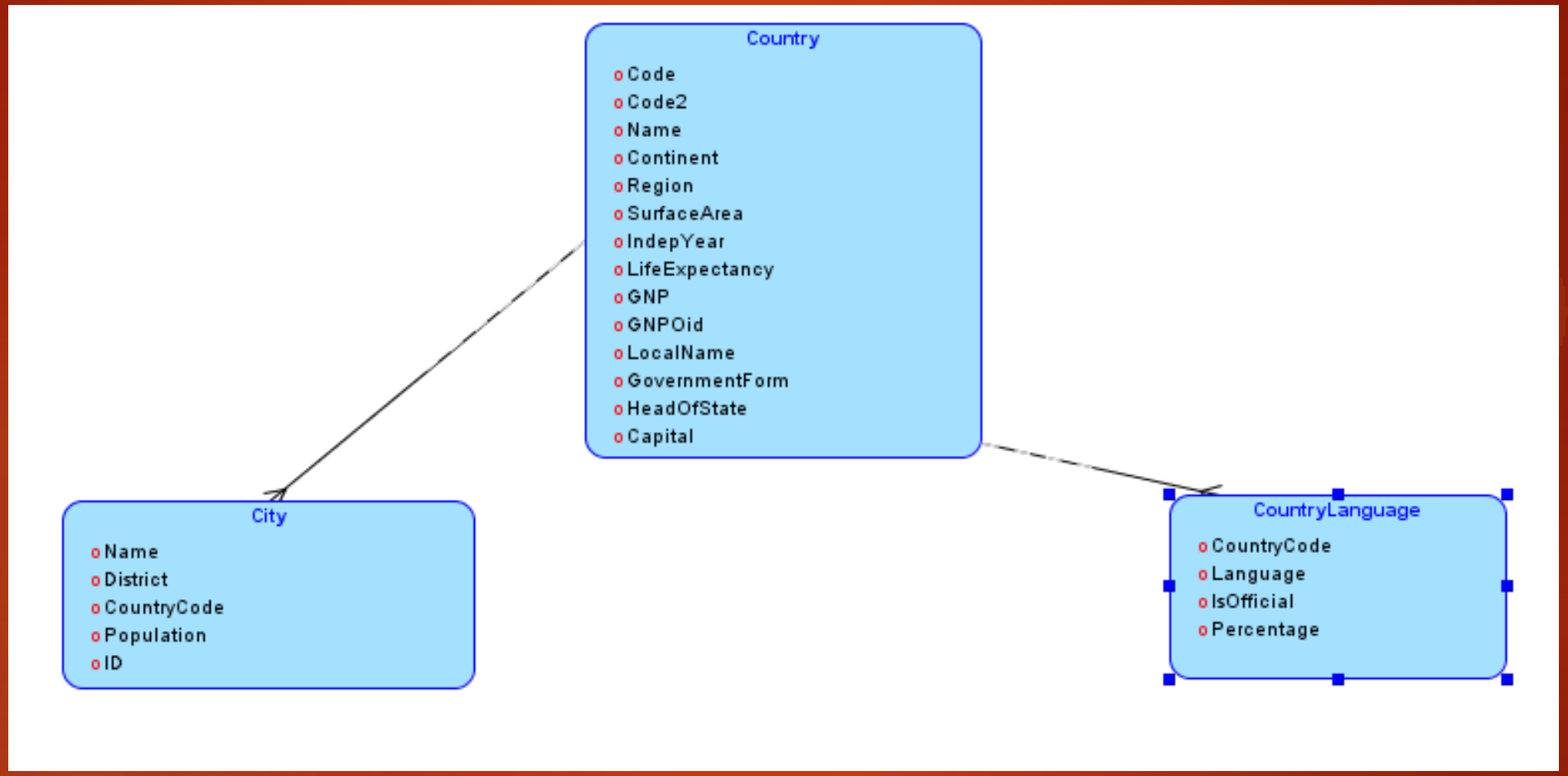
```
hive> SELECT  c.ID, c.NAME, o.AMOUNT, o.DATE
    > FROM CUSTOMERS c
    > LEFT OUTER JOIN ORDERS o
    > ON (c.ID = o.CUSTOMER_ID);
```

Differences Between HiveQL and SQL

As we know HiveQL and SQL are quite similar, but there are some differences. Let's describe distinguished ones:

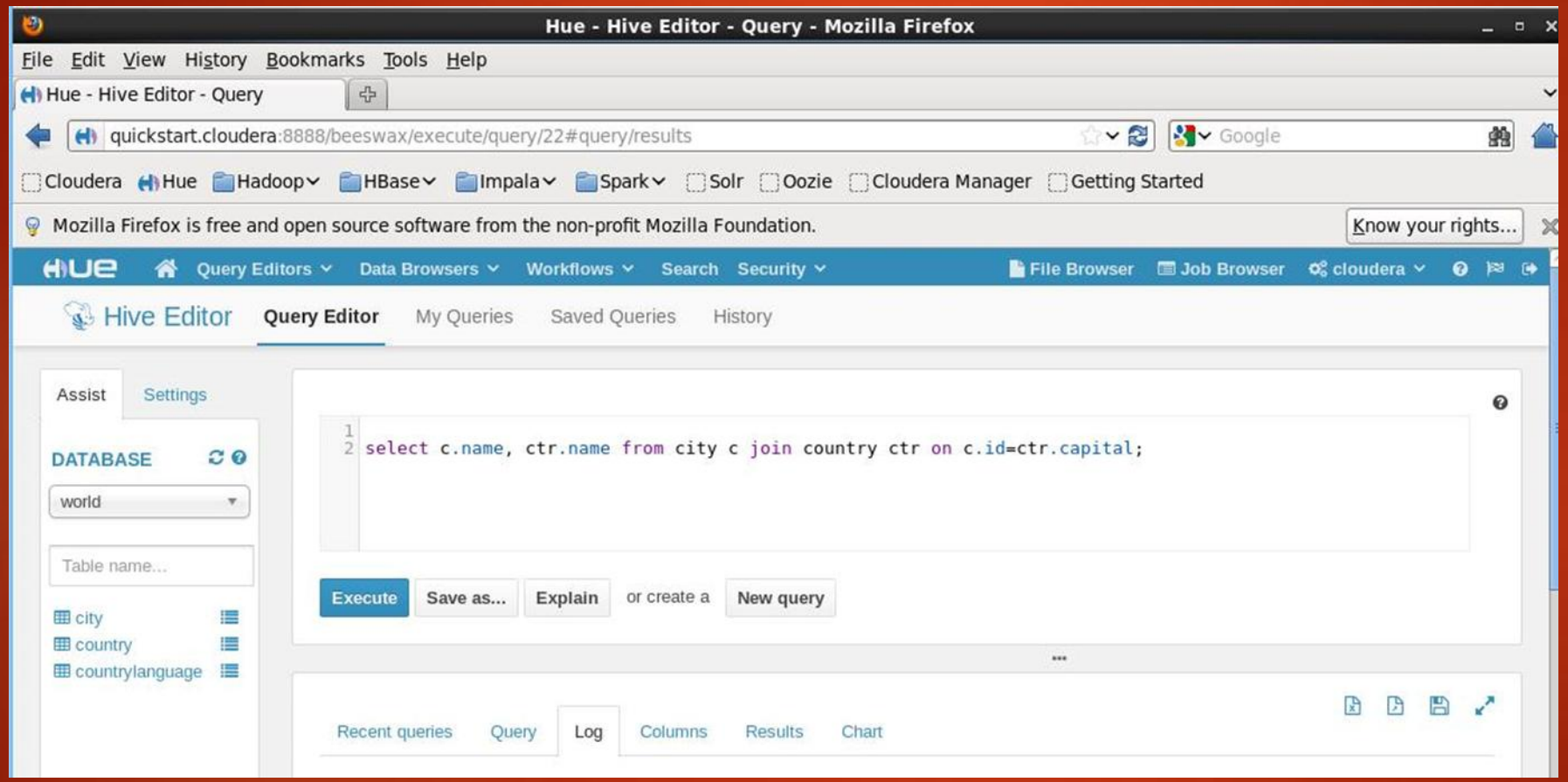
- Join Syntax
- Largest Table Last
- Group By
- Limit

Example of World Database



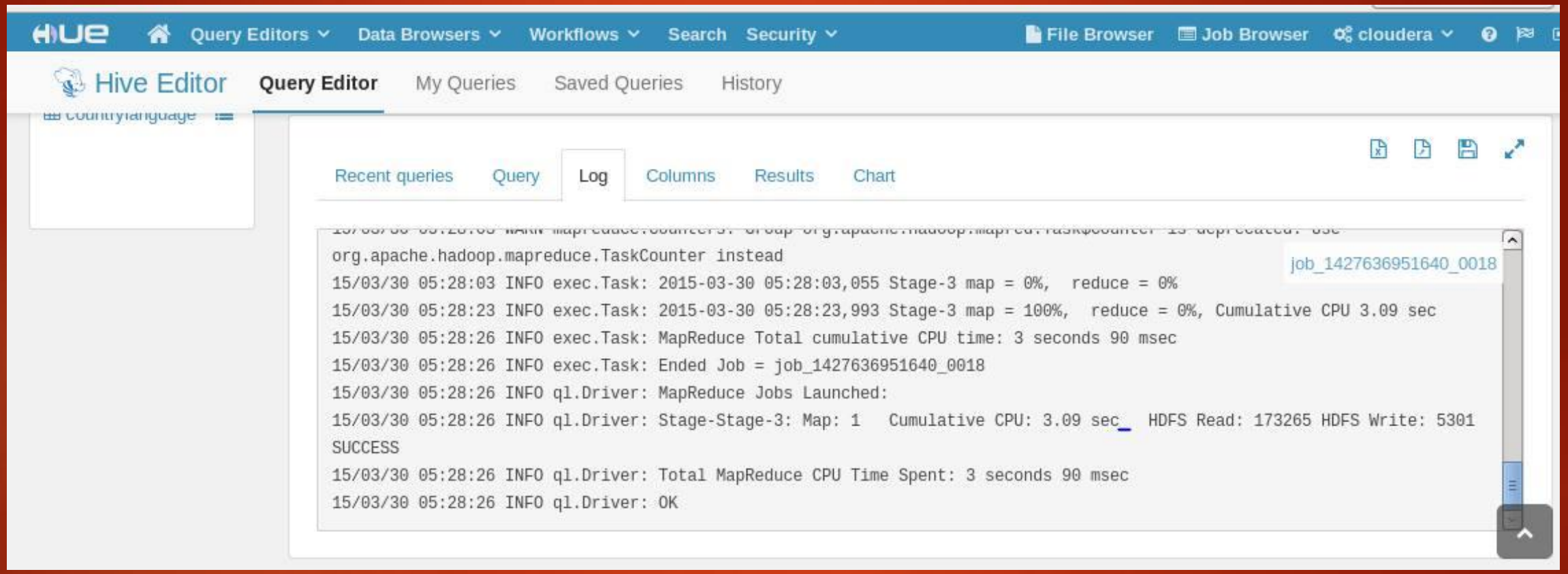
We will show implementation of this database practically, in cloudera quickstart VM

Select Example and Implementation in Hue



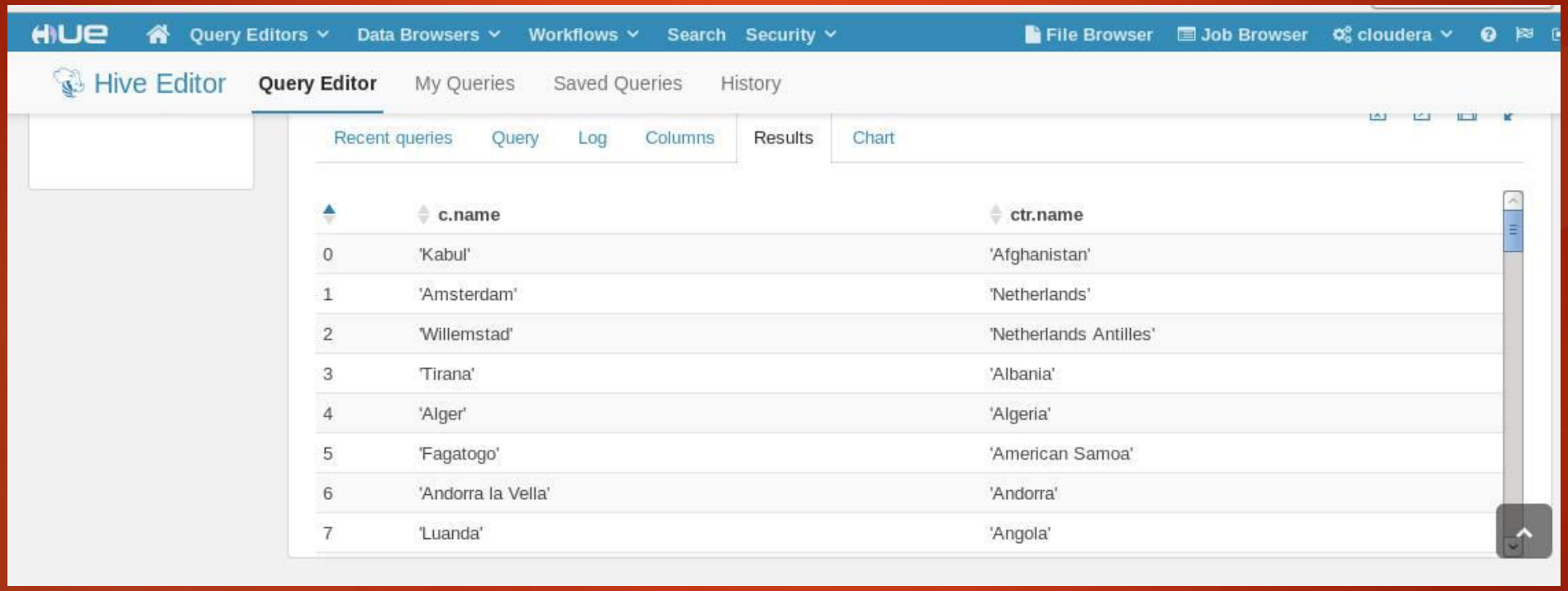
- This select joins two tables: countries and cities and outputs countries and its capitals

Log



- Log shows tasks of map-reducing – in other words how is hive database communicating with Hadoop through Hues

Result _ Output



Thanks For Your Attention