Introduction to Apache Hive

Part 1

Hive Basic Concepts and HiveQL Data Definition

What is Apache Hive

- Hive is an open source, petabyte scale data warehouse infrastructure built on top of Hadoop for big data summarization, query, and analysis. It was initially developed by Facebook and is a component of Hortonworks Data Platform (HDP).
- Hive provides a SQL-like interface (HiveQL or HQL) to the data stored in HDP. It has a query interface to a Hadoop-based data warehouse and can be easily adopted by the users who are used to using SQL.
- Hive supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems such as Amazon S3 file system.
- Hive stores metadata in an embedded Apache Derby database (a relational database, which significantly reduces the time to perform semantic checks during query execution).
- Hive currently supports four file formats TEXTFILE, SEQUENCEFILE, ORC and RCFILE.

Hive is …

- not a relational database:

  Hive uses a relational database to store metadata, but the actual data that Hive processes is stored in HDFS.
- not designed for online transaction processing
  - ✧ Hive runs on Hadoop, which is a batch-processing system where jobs can have latency with substantial overhead.
  - ✧ Even for small jobs the latency of Hive queries is generally high.
- not for real-time queries and low-level updates

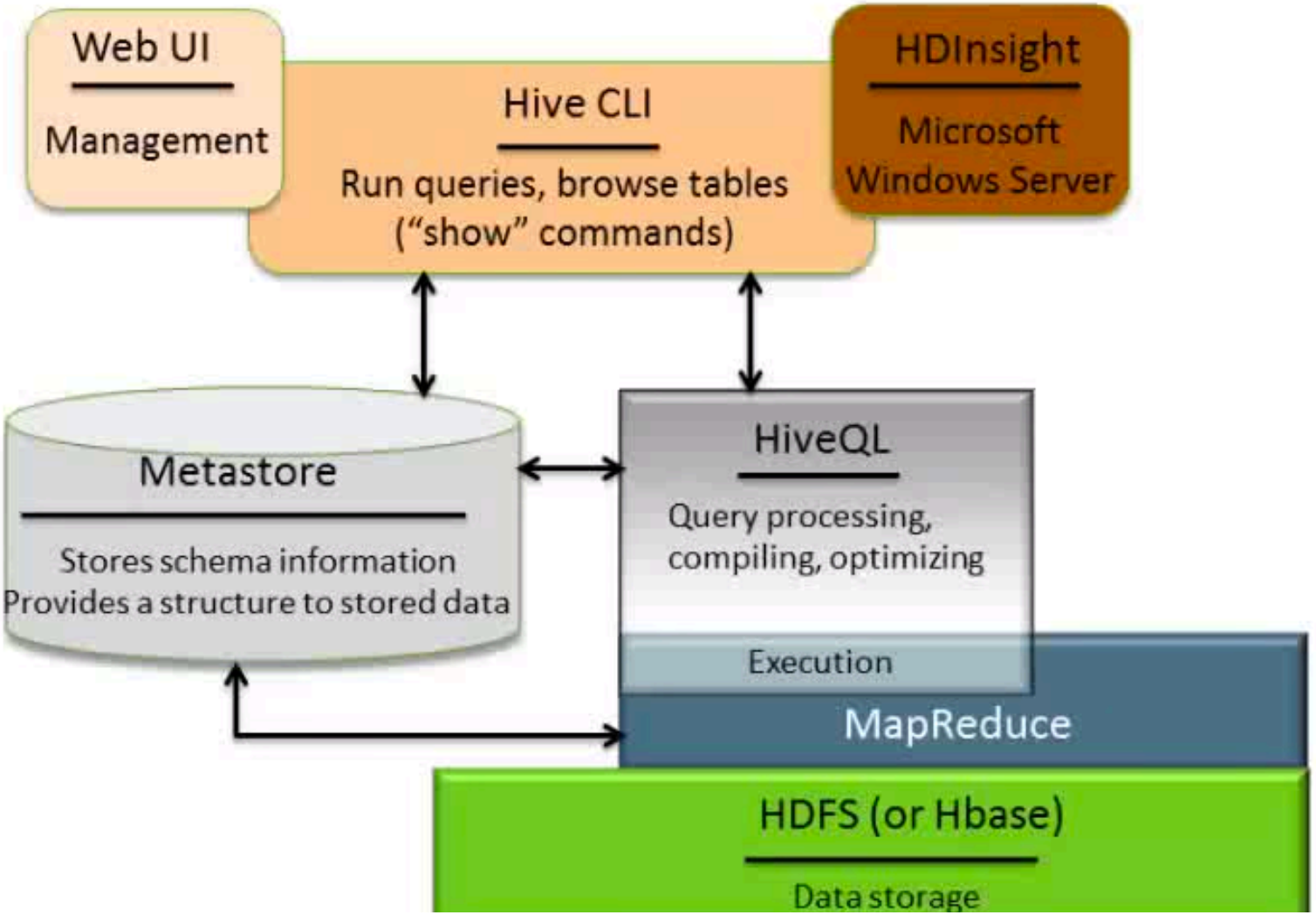
  Hive is best used for batch jobs over large sets of immutable data (e.g., web logs and historical data).
- not a full database (similar to OLAP, Online Analytic Processing) and does not provide record-level update, insert, or delete; however, it can be integrated with NoSQL databases such as HBase (but now Hive is integrated with HBase) and Cassandra.

A Typical Use Case of Hive

- A typical use of Hive is to take a large amount of unstructured data and to place it into a structure and a view that is comfortable to be used by data analysts or tools. HiveQL can also be extended by UDFs, etc.
- Hive supports uses such as:
  - Ad-hoc queries
  - Summarization
  - Data analysis

Hive Architecture Basics

System for querying and managing structured data



Two Ways to Use Hive

- The command line interface (or the Hive shell).
  - Do it interactively

```
$ hive
hive>
```

  - Use the -f flag to specify a file that contains a Hive script.

```
$ hive –f myquery.hive
```

- Some software tools, such as Microsoft HDInsight, which provides a Hive Console (it is a paid service), Cloudera's open source Hue, and Ambari's Hive view, etc.

The Hive Query Language (HiveQL)

- Is similar to other SQLs.
  - Uses familiar relational database concepts (tables, rows, columns and schemas).
  - Differs in various ways from the familiar SQL dialects provided by Oracle, MySQL, and SQL Server.
  - Based on the SQL-92 specification.
- Supports multi-table inserts via a code API: accesses “Big Data” via tables.
- Converts HQL queries into MapReduce jobs (so you even do not need to know MapReduce and how to map familiar data operations to the low-level MapReduce. Hive does this dirty work, so you can focus on the query itself).
- Also supports plugging custom MapReduce scripts into queries through the TRANSFORM clause.

Hive *vs.* Pig

Pig and Hive are both complete, so you can do all required data manipulations in each of them. They work together well and many businesses use both.

- **Hive is Good for:**
  - querying data
  - answering a specific question
  - making use of SQL knowledge to begin with

  (Any Hive query made, table created, or data copied persists from query to query. All the data are live, providing a data workbench for examining, modifying and manipulating the data in Hadoop.)
- **Pig is good for:**
  - ETL
  - preparing data for easier analysis
  - a long series of steps to place the data into a form that is accessible to the analysts.

  (In Pig, all data objects exist and are operated on in the script. Once the script is complete all data objects are deleted unless they are specifically copied out to storage.)

Hive Data Types and File Formats

- Hive supports many of the *primitive* data types you find in relational databases, as well as three *collection* data types that are rarely found in relational databases.
- It provides great flexibility in how data is encoded in files.
- Hive lets the users control how data is persisted to disk and its life cycle, which makes it easier to manage and process data with a variety of tools.

Hive Primitive Data Types*

Type	Size	Example
TINYINT	1 byte signed integer	10 (range: -128 to 127)
SMALLINT	2 byte signed integer	10 (range: -32768 to 32767)
INT	4 byte signed integer	10 (range: -2147483648 to …)
BIGINT	8 byte signed integer	10 (a huge range)
BOOLEAN	Boolean true or false	TRUE
FLOAT	Single precision floating point	3.14159265
DOUBLE	Double precision floating point	3.14159265
STRING	Sequence of characters.	'Hello World', "Hello World"
TIMESTAMP	Integer, float, or string	1327882394 1327882394.123456789 '2012-02-03 2:34:56.123456789'
BINARY	Array of bytes	N/A (max size: 2 billion bytes)

*Each of these data types is implemented in Java.

Hive Collection Data Types

Hive supports columns that are structs, maps, and arrays.

Type	Description	Example
STRUCT	Analogous to a C struct or an "object." Fields can be accessed using the "dot" notation. They can be of any data type and different data types.	struct{"FirstName" :'John', "LastName":'Smith '}
MAP	A collection of key-value tuples, where the fields are accessed using array notation (e.g., ['key']). Keys must be of primitive data types, but values can be any type. However, for a particular map, the keys must be of the same type and the values must have the same type.	map{'first':'John', 'last': 'Smith'}
ARRAY	Ordered sequences of the ***same*** *data* type that are indexable using zero-based integers.	array['John', 'Smith']

An Example of Using Hive Data Types

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING, city:STRING, state:STRING,
    zip:INT>);
```

Text File Encoding of Data Values

Delimiter	Description
\n	For text files, each line is a record, so the line feed character separates records.
^A ("Control" A)	Separates all fields (columns). Written using the octal code **\001** when explicitly specified in CREATE TABLE statements.
^B	Separate the elements in an ARRAY or STRUCT, or the key-value pairs in a MAP. Written using the octal code **\002** when explicitly specified in CREATE TABLE statements.
^C	Separate the key from the corresponding value within MAP key-value pairs. Written using the octal code **\003** when explicitly specified in CREATE TABLE statements.

Some Records for the Employees Table

John Smith^A100000.0^AMary Smith^BTodd Jones^AFederal Taxes^C.2^BState Taxes^C.05^BInsurance^C.1^A1 Michigan Ave.^BChicago^BIL^B60600

Mary Smith^A80000.0^ABill King^AFederal Taxes^C.2^BState Taxes^C. 05^BInsurance^C.1^A100 Ontario St.^BChicago^BIL^B60601

Todd Jones^A70000.0^AFederal Taxes^C.15^BState Taxes^C.03^BInsurance^C. 1^A200 Chicago Ave.^BOak Park^BIL^B60700

Bill King^A60000.0^AFederal Taxes^C.15^BState Taxes^C.03^BInsurance^C. 1^A300 Obscure Dr.^BObscuria^BIL^B60100

The 1st Line Data in the Table Schema (in JSON)


```
{
        "name": "John Smith",
        "salary": 100000.0,
        "subordinates": ["Mary Smith", "Todd Jones"],
        "deductions": {
                "Federal Taxes": .2,
                "State Taxes": .05,
                "Insurance": .1
        },
        "address": {
                "street": "1 Michigan Ave.",
                "city": "Chicago",
                "state": "IL",
                "zip": 60600
        }
}
```

Table Declaration with Formats Explicitly Specified

```
CREATE TABLE employees (
  name STRING,
  salary FLOAT,
  subordinates ARRAY<STRING>,
  deductions MAP<STRING, FLOAT>,
  address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\001'
  COLLECTION ITEMS TERMINATED BY '\002'
  MAP KEYS TERMINATED BY '\003'
  LINES TERMINATED BY '\n'
  STORED AS TEXTFILE;
```

Here the defaults are used for demonstration.

Override the Default Delimiters

You can override these default delimiters. This might be necessary if another application writes the data using a different convention. Here is a table definition where the data contain comma-delimited fields.

```
CREATE TABLE some_data (
    first FLOAT,
    second FLOAT,
    third FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Table Declaration with Formats Explicitly Specified

-- Override the default delimiters

```
CREATE TABLE employees (
  name STRING,
  salary FLOAT ,
  subordinates ARRAY<STRING>,
  deductions MAP<STRING, FLOAT>,
  address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '^'
  COLLECTION ITEMS TERMINATED BY '#'
  MAP KEYS TERMINATED BY '@'
  LINES TERMINATED BY '\n'
  STORED AS TEXTFILE;
```

HiveQL: Data Definition Language (DDL)

- *HiveQL* is the Hive query language. Hive adds extensions to provide better performance in the context of Hadoop and to integrate with custom extensions and even external programs.
- The *data definition language (DDL)* parts of HiveQL are used for creating, altering, and dropping databases, tables, views, functions, and indexes.

Hive Databases and Tables

- The Hive concept of a database is essentially a catalog or namespace of tables. The simplest syntax for creating a database is:

  hive> **CREATE DATABASE** financials;

  (hive> **CREATE DATABASE** IF **NOT EXISTS** financials;)

- Hive creates a directory for each database under a top-level directory based on your Hive settings. Tables in that database will be stored in the subdirectories. You can also specify (or create) a location for the new directory:

  hive> **CREATE DATABASE** financials
  > **LOCATION** '/mydata/preferred_directory';

The Default Hive Database Directory

The default directory in Hortonworks Sandbox HDFS:

/apps/hive/warehouse/

For example,

hive> create database jason;

By default, Hive creates the database at the /apps/hive/warehouse/jason.db directory.

The DESCRIBE DATABASE Command

You can use the DESCRIBE DATABASE statement to show the directory location of your databases:

DESCRIBE DATABASE jason;

DESCRIBE DATABASE financials;

DESCRIBE DATABASE default;

```
hive> create database financials
    > location '/mydata';
OK
Time taken: 0.042 seconds
hive> create database jason;
OK
Time taken: 0.048 seconds
hive> describe database jason;
OK
jason           hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/jason.db
Time taken: 0.029 seconds, Fetched: 1 row(s)
hive> describe database financials;
OK
financials              hdfs://sandbox.hortonworks.com:8020/mydata       root    USER
Time taken: 0.025 seconds, Fetched: 1 row(s)
hive> describe database default;
OK
default Default Hive database   hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse
Time taken: 0.034 seconds, Fetched: 1 row(s)
```

Show the Existing Databases

- Use the **SHOW DATABASES** command to see all your databases that already exist.

```
hive> show databases;
OK
default
financials
jason
xademo
Time taken: 0.023 seconds, Fetched: 4 row(s)
```

- Or use the **LIKE** keyword and the **.*** wildcard to show some specific databases.

```
hive> create database john;
OK
Time taken: 0.032 seconds
hive> show databases like 'j.*';
OK
jason
john
Time taken: 0.028 seconds, Fetched: 2 row(s)
```

The USE and SHOW TABLES Commands

- The USE command sets a database as your working database, analogous to changing working directories in a filesystem:

```
USE financials;
USE default;
```

- The SHOW TABLES command lists all the tables in your working database:

```
SHOW TABLES;
```

```
hive> show tables;
OK
geolocation
sample_07
sample_08
trucks
Time taken: 0.286 seconds, Fetched: 4 row(s)
```

Drop a Database

- Use one of the following commands to drop financials database:

  DROP DATABASE financials;

  DROP DATABASE IF EXISTS financials;

  DROP DATABASE IF EXISTS financials **CASCADE**;

- The CASCADE keyword will cause Hive to drop the tables in the database first, and then drop the DB.
- When a database is dropped, its directory is also deleted.

Hive Tables

- A Hive table consists of:
  - Data: typically a file(s) in HDFS.
  - Schema: in the form of metadata stored in a relational database.
- Schema and data are separate
  - A schema can be defined for existing data
  - Data can be added or removed independently
  - Hive can be pointed at existing data
- You have to define a schema if you have existing data in HDFS that you want to use in Hive.

Creating Tables: The CREATE TABLE Statement

```
CREATE TABLE IF NOT EXISTS financials.employees (
  name STRING COMMENT 'Employee name',
  salary FLOAT COMMENT 'Employee salary',
  subordinates ARRAY<STRING> COMMENT 'Names of subordinates' ,
  deductions MAP<STRING, FLOAT>
      COMMENT 'Keys are deductions names, values are percentages',
  address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
      COMMENT 'Home address' )
COMMENT 'Description of the table'
TBLPROPERTIES ('creator'='xy', 'created_at'='2019-03-06 09:05:00');
```

Create a Table From an Existing Table

```
CREATE TABLE IF NOT EXISTS financials.employees2
LIKE financials.employees;
```

(Copy the schema but not the data.)

```
CREATE TABLE t3 AS
SELECT * FROM trucks;
```

(Create the same table, including the schema and the data.)

The DESCRIBE EXTENDED Command

```
hive> describe extended financials.employees2;
OK
name                    string                  Employee name
salary                  float                   Employee salary
subordinates            array<string>           Names of subordinates
deductions              map<string,float>       Keys are deductions names, values are percenta
ges
address                 struct<street:string,city:string,state:string,zip:int>  Home address


Detailed Table Information      Table(tableName:employees2, dbName:financials, owner:root, cre
ateTime:1520329616, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name
:name, type:string, comment:Employee name), FieldSchema(name:salary, type:float, comment:Emplo
yee salary), FieldSchema(name:subordinates, type:array<string>, comment:Names of subordinates)
, FieldSchema(name:deductions, type:map<string,float>, comment:Keys are deductions names, valu
es are percentages), FieldSchema(name:address, type:struct<street:string,city:string,state:str
ing,zip:int>, comment:Home address)], location:hdfs://sandbox.hortonworks.com:8020/apps/hive/w
arehouse/financials.db/employees2, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outpu
tFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBucke
ts:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.Lazy
SimpleSerDe, parameters:{serialization.format=1}), bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), s
toredAsSubDirectories:false), partitionKeys:[], parameters:{totalSize=0, numRows=0, rawDataSiz
e=0, COLUMN_STATS_ACCURATE={"BASIC_STATS":"true"}, numFiles=0, transient_lastDdlTime=152032961
6}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGED_TABLE)
Time taken: 0.547 seconds, Fetched: 7 row(s)
```

The DESCRIBE FORMATTED Command

```
hive> describe formatted financials.employees2;
OK
# col_name              data_type               comment

name                    string                  Employee name
salary                  float                   Employee salary
subordinates            array<string>           Names of subordinates
deductions              map<string,float>       Keys are deductions names, values are percentages
address                 struct<street:string,city:string,state:string,zip:int>  Home address

# Detailed Table Information
Database:               financials
Owner:                  root
CreateTime:             Tue Mar 06 09:46:56 UTC 2018
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/financials.db/emplo

Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1520329616

# Storage Information
```

The Managed Tables and External Tables

- **Managed Table:** A table that is created under Hive's control. Hive stores the data in a subdirectory and manages the lifecycle of the data. The keyword "CREATE TABLE" is used when the table is created.

- **External Table:** A table using a storage location and contents that are outside of Hive's control. It is convenient for sharing data with other tools, but it is up to other processes to manage the life cycle of the data. The keyword "CREATE EXTERNAL TABLE" is used when the table is created.

- Can use the DESCRIBE EXTENDED or DESCRIBE FORMATTED command to check the table type.

External Tables: An Example

```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
  exchange          STRING,
  symbol            STRING,
  ymd               STRING,
  price_open        FLOAT,
  price_high        FLOAT,
  price_low         FLOAT,
  price_close       FLOAT,
  volume            INT,
  price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/data/stocks';
```

Partitioned, Managed Tables

```
CREATE TABLE employees (
    name            STRING,
    salary          FLOAT,
    subordinates    ARRAY<STRING>,
    deductions      MAP<STRING, FLOAT>,
    address         STRUCT<street:STRING, city:STRING,
    state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

Effects of Partitioning a Table

- Partitioning tables changes how Hive structures the data storage and creates subdirectories reflecting the partitioning structure.

```
...
.../employees/country=CA/state=AB
.../employees/country=CA/state=BC
...
.../employees/country=US/state=AL
.../employees/country=US/state=AK
...
```

- Once created, the partition keys (country and state, in this case) behave like regular columns.

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'IL';
```

The SHOW PARTITIONS Command

```
hive> SHOW PARTITIONS employees;
...
Country=CA/state=AB
country=CA/state=BC
...
country=US/state=AL
country=US/state=AK
...
```

```
hive> SHOW PARTITIONS employees PARTITION(country='US');
country=US/state=AL
country=US/state=AK
...
```

External Partitioned Tables

```
CREATE EXTERNAL TABLE IF NOT EXISTS log_messages (
  hms             INT,
  severity        STRING,
  server          STRING,
  process_id      INT,
  message         STRING)
PARTITIONED BY (year INT, month INT, day INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

```
hive> SHOW PARTITIONS log_messages;
...
year=2011/month=12/day=31
year=2012/month=1/day=1
year=2012/month=1/day=2
...
```

Dropping Tables

```
DROP TABLE IF EXISTS employees;
```

- For managed tables, the table metadata and data are deleted
- Use with caution

Alter Table

- Most table properties can be altered with ALTER TABLE statements, which change metadata about the table but not the data itself. These statements can be used to fix mistakes in schema, move partition locations, etc.
- It's up to you to ensure that any modifications are consistent with the actual data.

Alter Table Statements

- **Renaming a Table**

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

- **Adding, Modifying, and Dropping a Table 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'
PARTITION (year = 2011, month = 1, day = 3) LOCATION '/logs/2011/01/03' ;

ALTER TABLE log_messages PARTITION(year = 2011, month = 12, day = 2)
SET LOCATION 's3n://ourbucket/logs/2011/12/02';

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

- **Changing Columns**

```
ALTER TABLE log_messages
CHANGE COLUMN hms hours_minutes_seconds INT
COMMENT 'The hours, minutes, and seconds part of the timestamp'
AFTER severity;
```

Alter Table Statements (cont'd)

- **Adding Columns**

```
ALTER TABLE log_messages ADD COLUMNS (
app_name STRING COMMENT 'Application name',
session_id LONG COMMENT 'The current session id');
```

- **Deleting or Replacing Columns:** removes all the existing columns and replaces them with the new columns specified

```
ALTER TABLE log_messages REPLACE COLUMNS (
hours_mins_secs INT COMMENT 'hour, minute, seconds from timestamp',
severity STRING COMMENT 'The message severity'
message STRING COMMENT 'The rest of the message');
```

- **Alter Table Properties**

```
ALTER TABLE log_messages SET TBLPROPERTIES (
'notes' = 'The process id is no longer captured; this column is always NULL');
```

- **Alter Storage Properties**

```
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1)
SET FILEFORMAT SEQUENCEFILE;
```

Alter Table Statements (cont'd)

- **Prevent the partition from being dropped and queried:**

```
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) ENABLE NO_DROP;

ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) ENABLE OFFLINE;
```

- **To reverse:**

```
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) DISABLE NO_DROP;

ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) DISABLE OFFLINE;
```