The Ultimate Guide to Programming Apache Hive

A Reference Guide Document – Straight from the trenches, with real world lessons, tips and tricks to help you take on and start analyzing Big Data with Apache Hive

- FRU NDE -


THE ULTIMATE GUIDE TO PROGRAMMING HIVE
- FRU NDE -
THE ULTIMATE GUIDE TO PROGRAMMING APACHE HIVE
A REFERENCE GUIDE DOCUMENT - STRAIGHT FROM THE TRENCHES, WITH REAL WORLD LESSONS, TIPS AND TRICKS TO HELP YOU TAKE ON AND START ANALYZING BIG DATA WITH APACHE HIVE
- FRU NDE -

THE ULTIMATE GUIDE TO PROGRAMMING APACHE HIVE

A Reference Guide Document – Straight from the trenches, with real world lessons, tips and tricks included to help you take on and start analyzing BigData with Apache Hadoop.

Note: All effort has been made to keep this book vendor agnostic. It doesn't matter what vendor or distribution you use for hadoop and hive, the concepts in here should be accessible to you.

First Edition

Printed in the, United States of America

Let’s Get Started!

I want to thank you and congratulate you for downloading this book "The Ultimate Guide to Programming Apache Hive".

This book contains proven steps and strategies on how to become a truly savvy Hive Programmer. You will be able to increase your profitability, areas of opportunity, and so much more by unleashing the Hive tips this book will teach you.

Here’s an inescapable fact: you will need to practice your way through mastering Hive.

Because Hive is basically SQL on Hadoop, you would need to develop your SQL skills along the way.

If you do not develop your SQL skills, you will not master Hive. SQL is at the heart of Hive, and this book will help you get on the right path.

It’s time for you to become an amazing Big Data professional, who can crunch massive amounts of data with the help of the information compiled within this book.

Introduction

If you are already familiar with SQL then you may well be thinking about how to add Hadoop skills to your tool belt as an option for data processing.

From a querying perspective, using Apache Hive is the next logical step you can take. Using Hive provides a familiar interface to data held in a Hadoop cluster and is a great way to get started with minimal learning curve.

Apache Hive a is data warehouse infrastructure built on top of Apache Hadoop for providing data summarization, ad-hoc query, and analysis of large datasets. It provides a mechanism to project structure onto the data in Hadoop and to query that data using a SQL-like language called Hive Query Language (HiveQL or HQL).

In this book, I will take you deep down into Hive. I will do what many other books haven't successfully done. Provide you with High level information about Apache Hive that will allow you to strategically shine in that next board room presentation, while leaving you with enough details to tactically take on BigData projects and amaze your peers in the process.

Audience

This tutorial is prepared for professionals aspiring to make a career in Big Data Analytics using Hadoop Framework. Administrators, ETL developers, Data Analysts and professionals who are into analytics on Hadoop using Apache Hive will be able to use this tutorial to good effect.

Prerequisites

Before proceeding with this tutorial, you need a basic knowledge of concepts of Structured Query Language (SQL), Data Warehousing Concepts of tables, columns, facts and dimensions, and any understanding of Linux operating systems, massively parallel processing would be a plus. It also assumes you have a working installation of Apache Hive. You can either accomplish this by downloading and installing the raw tarball from the Hive website, or by going through any one of the vendored packages such as Hortonworks, Microsoft, Cloudera, MapR, Pivotal, or more.

About Apache Hive

Apache Hive (http://hive.apache.org) is an open source volunteer project under the Apache Software Foundation (http://www.apache.org/). Apache Hive (commonly called Hive) provides data warehousing package/infrastructure built on top of Hadoop.

Brief History

Hive was originally an internal project by the Facebook Data team. After being used internally, it was contributed to the Apache Software Foundation and made freely available as an open source software. The project then quickly matured into a full blown Apache Project. Its home page URL is. http://hive.apache.org

Initially, it was a subproject of Apache Hadoop, but has now graduated to become a top-level project of its own. Hive is extremely powerful because it provides a SQL query language, called Hive Query Language (HQL) for querying data stored in a Hadoop cluster, either directly in HDFS or in other data storage systems such as Apache Hbase.

The Anatomy of Hive

Hive has all the structures similar to a RDBMS like tables, joins, partitions. This similarity with traditional SQL enables users familiar with SQL to easily query underlying data in Hadoop. Hive itself is NOT a database solution like MySQL, HBase, and Cassandra. Instead, it is simply a query system on top of HDFS, and behind the scenes, every hive tables are simply references to files on HDFS.

This unique characteristic of Hive being able to directly query underlying files in HDFS is good because it allows for other applications to modify Hive files within HDFS independently of the Hive application. But this advantage comes with some draw backs. Other applications having access to those files means that Hive can never be certain if the data being read matches the schema. Therefore, Hive enforces **schema on read**. If the underlying data format has changed, Hive will do its best to compensate when it reads, but users can likely get unexpected results.

HiveQL

HiveQL does not fully conform to any particular revision of the ANSI SQL standard.

Until Hive 0.14, HiveQL was not ACID (an acronym for Atomicity, Consistency, Isolation and Durability) complaint, offered no support for row level inserts, updates,

and deletes. But that has been resolved as of the 0.14 release of Hive. Now, Hive allows users to modify data using insert, update and delete SQL statements. It also provides snapshot isolation and uses locking for writes -- allowing users to make corrections to fact tables and changes to dimension tables with relative ease.

As of now (the writing of this book), Hive doesn't support transactions. But who knows; the Hadoop ecosystem and Hive are under very active development and support for transactions might be introduced with time.

Not your traditional RDBMS

Hive is not designed for online transaction processing and does not offer real-time queries. As a result, most people tend to not compare it with RDBMS i.e. the architecture of Hive is different from the Architecture of RDBMS.

> *According to the Apache Hive wiki, "Hive is not designed for OLTP workloads and does not offer real-time queries or row-level updates. It is best used for batch jobs over large sets of append-only data (like web logs)."*

Data Warehousing Solution built on top of Hadoop. It is very well suited for batch processing data like: Log processing, Text mining, Document indexing, Customer-facing business intelligence, Predictive modeling, hypothesis testing etc.

Summary:

- Hive Provides SQL-like query language named HiveQL
- Early Hive development work started at Facebook in 2007
- Today Hive is an Apache project that was previously under Hadoop, but is now a project on its own
- With Hive, multiple schemas can be projected on the same data -- no ETL required
- With Hive, popular query tools like Power Pivot, MicroStrategy, Tableau, SAS e.t.c. and be connected and provided access to the underlying data via JDBC and ODBC

- Hive allows users to query very large data sets interactively using MapReduce, Tez, and more recently the Spark Query engine.

Why Hive?

Since most data professionals already know SQL and most Data Warehouse applications today support SQL, Hive minimizes the effort of analyzing data at scale on top of the Hadoop platform.

Hive is massively scalable with Hadoop. When users write queries with HQL, Hive takes the query statements and automatically translates them into one of (MapReduce jobs, TEZ, or Spark – depending on the execution engine being used). The translated query is then executed against the data in the Hadoop cluster and the results are returned to the user.

This saves Analysts time from having to write their own individual Map and Reduce tasks before being able to execute a single query.

Typical Hive Workflow

When using Hive, users typically perform the following functions or workflow steps:

1. Create tables
2. Load data from source system in to Hive table or HDFS directories
3. Analyze/Process data by writing HiveQL

Key Benefits of Hive

Hive is ideal for data warehouse teams migrating to Hadoop, because it gives them a familiar SQL language that hides the complexity of MR programming. It also provides teams with the ability to bring structure to various data formats such as Flat files, logs and more.

Hive supports text files (also called flat files), SequenceFiles (flat files consisting of binary key/value pairs) and RCFiles (Record Columnar Files which store columns of a table as a columnar database)

Summary Benefits of Hive

- Familiar SQL dialect. (This characteristic alone makes Hive indispensable for most business users, who are already SQL users).
- Analysis of large datasets on Hadoop using (MapReduce, TEZ, or Spark jobs).
- Simple interface for ad hoc querying, analyzing and summarizing large amounts of data
- Access to files on various data stores such as HDFS and Hbase.

Key Disadvantages of Hive

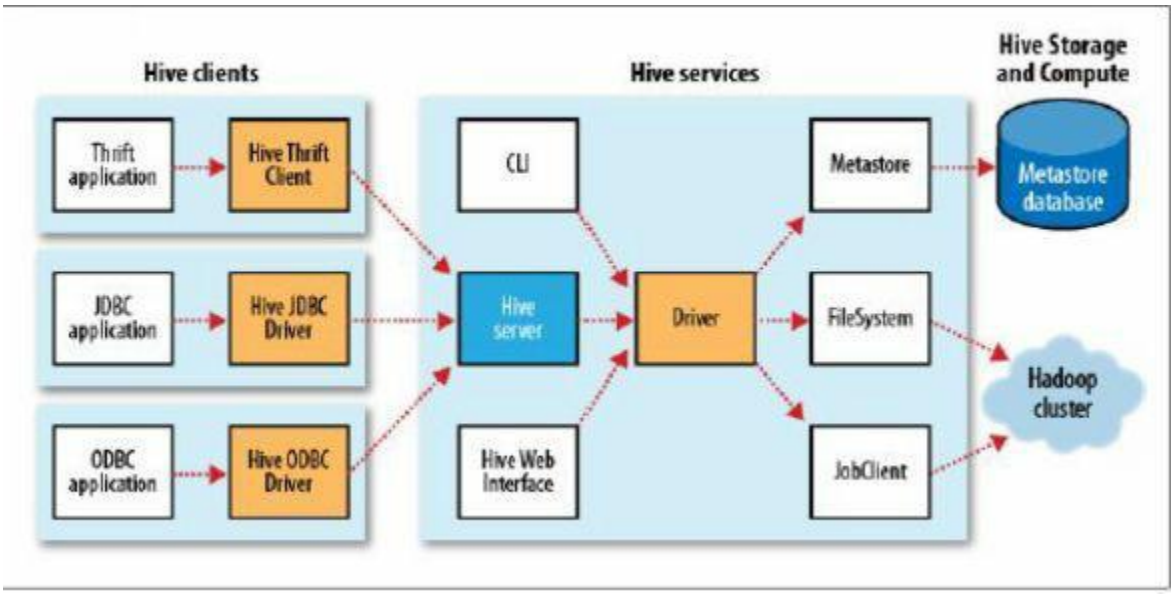
Despite the benefits of Hive, it's not without its draw backs. When it comes to system performance, Hive has several downsides. First, the batch nature of Map / Reduce makes Hive perform poorly when you need low-latency execution for simple queries.

The Apache Hive community is working effortlessly to address some of these issues with using Hive as a replacement for traditional RDBMS systems, but as it stands today, Hive still lacks some of the functionalities commonly present in database systems for supporting high-performance analytical queries.

Summary Disadvantages of Hive

- Not a real SQL Database.
- Up until Hive 0.14, HiveQL wasn't ACID complaint.
- Designed for scalability and ease-of-use. But not ideal for low latency, sub-second, or real-time queries
- Even querying small amounts of data may take minutes. (Results are much improved when using TEZ or Spark Execution engines).
- Hive is useful only if the data is structured. With unstructured data, Hive is not a good tool. Instead, by directly writing pure MapReduce code, users can work on any kind of dataset (structured or unstructured).

Hive Architecture



http://4.bp .blogspot.com/-Yy3-cDDuiqk/UAkBvWXmz9I/AAAAAAAAGWk/RM 1rJ5nWi9M /s1600/HiveClient.png

Some of the key components of the Hive Architecture are the MetaStore, the interfaces users use, and the client services that support all interactions. These Hive components would all be discussed in detail in the sections below.

Hive MetaStore

The Hive MetaStore is a very important piece of the Hive Architecture. To support features like schema(s) and data partitioning Hive keeps its metadata in a MetaStore.

This metadata contains information about what tables exist, their columns, privileges, column types, owners, key-value data, statistics and more.

The Hive MetaStore data is persisted in an ACID database such as Derby, Oracle DB or MySQL. And all Hive processes communicate with the MetaStore service using Thrift to read off the Metadata.

Hadoop NameNode vs Hive MetaStore

When most people are first introduced to Hadoop and the MPP architecture, they are taught about the NameNode, DataNode and their roles. The common scenario is that the NameNode is responsible for storing the directory tree of all files in the HDFS file system. In addition, the NameNode also performs the following functions:

Hive Name Node Functionalities

- It also tracks where Hive data (not metadata) is spread across Hadoop HDFS DataNode servers (Typically, each block of data is replicated on 3 different DataNodes).
- It tracks which DataNodes are dead and which ones are Live through heartbeat mechanism.
- It helps clients perform reads and writes on HDFS by receiving their requests and redirecting them to the appropriate DataNode.

But despite this, the Hadoop NameNode is NOT responsible for storing Hive Metadata. Instead, this information is managed by the Hive Service outside and stored in a separate relational database systems; typically Derby, MySQL, Oracle, PostgreSQL or any other relational database management system.

The Hive MetaStore contains information about Hive data such as:

- IDs of Database, Tables and Indexes
- The time of creation of a Table
- The time of creation of an Index
- Statistics of number of records stored in each tables at a point in
- time IDs of roles assigned to a particular user InputFormat used for
- a Table
- OutputFormat used for a Table

Example: NameNode vs MetaStore

Assume you go to Hive and create a new table as such

```
hive >
CREATE TABLE log_table (
        id BIGINT
        ,NAME string
        ) partitioned BY (server string);
```

When the query above executed, information about the newly created table would be persisted in the Hive MetaStore (not NameNode).

Then, when you insert data into the Hive log_table, the actual data inserted will be stored on Hadoop Data and Name Nodes (not MetaStore).

Tips on Hive MetaStore

- Some Hive queries have to be compiled in order to generate MapReduce jobs. But, some Hive operations don't invoke Hadoop processes at all. Some very simple queries will just either read or write updates directly to the MetaStore database.
- Example: Queries like *SELECT count (*) from tbl_TableName* just reads the table statistics stored in the MetaStore, and so returns results very fast.
- Out of the box Hive comes with Derby database as the default MetaStore location. The out of the box Derby database (a lightweight embedded SQL DB) used by Hive is only effective for a single user and a single process at a time. For large scale clusters, or in production, users need to set up a MySQL or PostgreSQL database for Hive's metadata.

- It is not mandatory to have MetaStore in the cluster itself. Any machine (inside or outside the cluster) having a JDBC-compliant database can be used for the MetaStore.

Hive Interface Options

Users and analysts can interact with Hive using several methods, including the CLI, HUE, and JDBC/ODBC.

Command Line Interface (CLI) and Beeline

– CLI and Beeline are one to the top primary interfaces through which developers work with Hive data.

– Generally speaking, CLI and Beeline perform similar functions i.e. execute queries against Hive, but the primary difference between the two involves how the clients connect to Hive. The Hive CLI connects directly to the Hive Driver and requires that Hive be installed on the same machine as the client. However, Beeline connects to HiveServer2 and does not require the installation of Hive libraries on the same machine as the client. Beeline is a thin client that uses the Hive JDBC driver and executes queries through HiveServer2, which allows multiple concurrent client connections and supports authentication.

– When using the CLI, the following properties can be set to help make the results that are printed on the screen such as column names and database context more clear and intuitive.

```
-- print column headers for query results
SET hive.cli.PRINT.header = true;

-- print default db. i.e. the current database context
SET hive.cli.PRINT.CURRENT.db = true;

Pro Tip: Note that, a common source of error occurs when users write code us-
ing an outside IDE and try to copy/paste it to the CLI. When pasting hive
code to the CLI, avoid lines that begin with tabs. Otherwise, this will cause
the CLI interface to not execute the hive script and error messages would be
displayed.
```

Hadoop User Experience (HUE)

– HUE (http://gethue.com/) is an open-source Web user interface that supports Apache Hadoop and its ecosystem, licensed under the Apache

v2 license. Hue allows technical and non-technical users to take advantage of Hive, Pig, and many of the other tools that are part of the Hadoop ecosystem.

– Hue is great alternative to running 'show tables' or 'show extended tables' from the CLI because it provides a very web-based GUI for users to quickly browse the schema of Hive tables, and also write queries.

– With Hue, the Hive metadata is presented in a hierarchical manner allowing users to start at the database level and click to get information about tables including the SerDe, column names, and column types.

Java/Open Database Connectivity (JDBC/ODBC)

– Hive also provides JDBC and ODBC drivers that allow users to connect popular Business Intelligence (BI) tools such as Excel, SAS, MicroStrategy, Tableau, e.t.c. to query, analyze and visualize data stored within Hive.

– Hive JDBC URLs have the following format:

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?<hiveConfs>#<hiveVars>
```

– Depending on which vendor distribution of Hive you are using (Hortonworks, Cloudera, MapR, e.t.c.), there might be vendor specific Hive ODBC drivers available you can download and use to connect to hive cluster, instead of using the native out-of-the box driver provided by Hive.

Hive Query Execution Engines

There are several ways users can execute queries on Hive. Most queries use MapReduce jobs, or more recently Tez and Spark execution engine.

Hive on MapReduce

Originally the MapReduce algorithm was very tightly coupled into the workings of Hadoop's cluster management infrastructure. MapReduce is very powerful, because of its divide and conquer approach – i.e. it divides the query work and the data up amongst the numerous commodity servers in its cluster, facilitates teamwork between them, and gets the answer.

Traditionally, Hive would run using the MapReduce engine. This meant, Hive generates MapReduce jobs to implement all but the simplest queries. Unfortunately, with this approach, the benefits of MapReduce (horizontal scalability), was quickly overshadowed by the high latency in query performance. Users who were used to sub-second queries from the RDBMS world simply weren't getting it with Hadoop.

In Hadoop 2.0, YARN (an acronym for "yet another resource negotiator") was introduced as a processing algorithm-independent cluster management layer. YARN allows for Hadoop cluster to run MapReduce jobs, but it can host an array of other engines, as well.

This separation of the cluster management layer with the introduction of YARN let to the up-spring of new execution engines such as Tez and Spark that compete and even eclipse traditional MapReduce in many benchmark tests.

Depending on what version of Hive you use, MapReduce might be the default (or the only) execution engine available. To execute Hive jobs using the MapReduce execution engine, set (or change) the following property:

```
SET hive.execution.engine = mr;
```

Hive on Tez

Tez (http://tez.apache.org/) is a new application framework built on Hadoop Yarn that can execute complex directed acyclic graphs of general data processing tasks. In many ways it can be thought of as a more flexible and powerful successor of the MapReduce framework. To use the Tez execution Engine on Hive, set the hive.execution.engine property to Tez before any queries.

```
SET hive.execution.engine = tez;
```

You can do this from the Hive CLI or as an option in your Hive command. Setting this property will enable your Hive CLI to pick up the hive.execution.engine setting that tells it to use Tez rather than MapReduce.

Hive on Spark

Apache Spark (https://spark.apache.org/) is an open source big data processing framework built around speed, ease of use, and sophisticated analytics. It was originally developed in UC Berkeley's AMPLab (Source: http://www.infoq.com/articles/apache-spark-introduction), and open sourced in 2010 as an Apache project.

Spark can run on Hadoop clusters and, because it uses memory instead of disk, can also avoid long query execution times experienced with MapReduce's batch mode. Spark provides support for a SQL language (Spark SQL). It also provides command-line interfaces and ODBC/JDBC server.

Spark claims to run 100× faster than MapReduce. For this reason, we see most people are having interest or moving altogether to Spark.

To enable and use spark execution engine, set the following property.

```
SET hive.execution.engine = spark;
```

You can do this from the Hive CLI or as an option in your Hive command. Setting this property will enable your Hive CLI to pick up the hive.execution.engine setting that tells it to use Spark rather than MapReduce or Tez.

Pro Tip: Putting settings into a .hiverc file

While it is possible to set Hive properties from the Hive CLI or put them into a Hive command, alternatively, you can create a *.hiverc file* in your home directory on the Hadoop cluster and set the execution engine parameter in that file. In the latter case, all your Hive jobs will execute with the properties that are set in the *.hiverc file*.

Setting Hive properties using the Hive CLI or Hive command will only affect individual Hive jobs. But, if you have properties that you want to set across all your jobs, *.hiverc file* would be a more efficient way to go.

Each user can have a *.hiverc file* in his or her own home directory to set up their hive environment.

Example:

```
$cat /home/Jane_Doe/.hiverc
SET hive.execution.engine = tez;
SET hive.job.queuename = elt;
ADD /home/Jane_Doe/Foo/JAR CoolHive.jar;
```

In this case, we have setup a *.hiverc file* in Jane_Doe's home directory. When Jane executes any Hive commands from the CLI, these properties will first be set before the command is executed. Making things much easier and consistent.

Anatomy of HiveQL

Hive is largely in synch with the traditional Relational Databases concepts. In Hive as in other RDBMS systems, we have the concept of Databases, Tables, Rows, Columns, e.t.c.

- **Database:** Set of Tables, used for name conflicts resolution
- **Table:** Set of Rows that have the same schema (same columns)
- **Row:** A single record; a set of columns
- **Column:** A unique instance and type for a single value

Along with the similarities Hive has with other RDBMS systems, HiveQL also wields many important similarities to traditional SQL. (The ultimate goal of the Hive project is to get it fully ACID compliant, and that goal was achieved as of Hive 0.14).

Also, the good thing with Hive is that users get most of the usual suspects of SQL statements such as SELECT, WHERE, e.t.c. to use for their data analysis.

Below are some to the common syntax similarities HiveQL shares with native SQL.

- **SELECT** Scans the table specified by the **FROM** clause
- **JOIN** joins results sets from one table to another. Supports Inner, Left, Right, and Full Outer Joins. Default Join in Hive is Inner Join. In Inner Join, rows are joined where the keys match. Rows that do not match are not included in the result.
- **GROUP BY** Gives a list of columns which specify how to aggregate the records. A GROUP BY clause is frequently used with aggregate functions, to group the result set by columns and apply aggregate functions over each group.

- **WHERE** Gives the condition of what to filter. A WHERE clause is used to filter the result set by using predicate operators and logical operators.
- **Having** A HAVING clause lets you filter the groups produced by GROUP BY, by applying predicate operators to each groups.
- **CLUSTER BY, DISTRIBUTE BY, SORT** BY specify the sort order and algorithm
- **LIMIT** specifies and limits the number of records to retrieve.

Hive Commands

In Hive, *Commands* are non-SQL statement such as setting a property or adding a resource. They can be used in HiveQL scripts or directly in the CLI or Beeline.

List of Hive Commands

Command	Description
quit exit	Use quit or exit to leave the interactive shell.
reset	Resets the configuration to the default values. Any configuration parameters that were set using the set command or -hiveconf parameter in hive command line will get reset to default value.
set <key>=<value>	Sets the value of a particular configuration variable (key). **Note:** If you misspell the variable name, the CLI will not show an error.
set	Prints a list of configuration variables that are overridden by the user or Hive.
set -v	Prints all Hadoop and Hive configuration variables.
Add, List, Delete FILE[S] <filepath>	Adds, Lists or Removes one or more files, jars, or archives to the list of resources in the distributed cache.

! <command>	Executes a shell command from the Hive shell.
dfs <dfs command>	Executes a dfs command from the Hive shell.
<query string>	Executes a Hive query and prints results to standard output.
source FILE <filepath>	Executes a script file inside the CLI.

The full list and details of Hive Commands can be viewed at

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Commands

Sample Usage of Hive Commands

```
hive> set hive.execution.engine=tez;
hive> set;
hive> select * from log_table limit 10;
hive> !ls;
hive> dfs -ls;
```

HiveQL Data Types

Hive, being ACID compliant now has a lot of the data types you would find in any other traditional SQL database.

In Hive, data types are categorized in to two main Categories, viz; primitive and complex data types.

HiveQL Primitive Data Types

The primitive data types include Integers, Boolean, Floating point numbers and strings. The below table lists the size of each data type:

- **Numeric Types**
  - TINYINT: 1-byte (signed integer, from -128 to 127)
  - SMALLINT: 2-byte (signed integer, from -32,768 to 32,767)
  - INT: 4-byte (signed integer, from -2,147,483,648 to 2,147,483,647)
  - BIGINT: 8-byte (signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
  - FLOAT: 4-byte (single precision floating point number)
  - DOUBLE: 8 byte (double precision floating point numbers)
  - DECIMAL: Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The range of decimal type is approximately -10^{-308} to 10^{308}. The syntax and example is as follows: DECIMAL (precision, scale). Example: decimal (10,0).
- **Date/Time Types**

– TIMESTAMP: Hive supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.fffffffff”.

– DATE: DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

- **String Types**

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

– STRING: Max size is 2GB.

– VARCHAR: 1 to 65355 in length

– CHAR: 255 in length

- **Misc Types**

– BOOLEAN: TRUE/FALSE value

– BINARY

Example Syntax: Hive Primitive Data Type

Syntax:

```
CREATE TABLE simple_dt (
        Log string
        ,ServerNames string
        ,Regions BIGINT
        ,Measure1 TINYINT
        ,Measure2 DOUBLE
        ,Zip_Code INT
        ,Load_date DateTime
        );
```

HiveQL Complex Data Types

The complex data types in Hive include Arrays, Maps and Structs. Complex data types are all built on using the primitive data types.

– **Arrays:** Contain a list of elements of the same data type. These

elements are accessed by using an index. For example an array, “tweets”, containing a list of elements [‘happytweet’, ’oktweet’, ‘joyfultweet’], the element “happytweet” in the array can be accessed by specifying tweets.

```
Syntax: ARRAY < data_type >
```

(Note: negative values and non-constant expressions are allowed as of Hive 0.14.)

– **Maps:** Maps are key-value pairs. The elements are accessed by using the keys. For example a map, “log_list” containing the “servername” as key and “serverlocation” as value, the location of the server can be accessed by specifying log_list[‘servername’].

```
Syntax: Map < primitive_type, data_type >
```

(Note: negative values and non-constant expressions are allowed as of Hive 0.14.)

– **Structs:** Structs are like “objects” or “c-style structs”. They contain elements of different data types. The elements can be accessed by using the dot notation. For example in a struct, “log”, the server of the log can be retrieved as specifying *log.server*.

```
Syntax: STRUCT < col_type: data_type [COMMENT col_comment], ... >
```

– **Union Types**: Union types can at any one point hold exactly one of their specified data types. You can create an instance of this type using the create_union UDF:

```
Syntax: UNIONTYPE < data_type, data_type, ... >
```

(Note: Only available starting with Hive 0.7.0.)

Example: Hive Complex Data Type

Syntax:

```
CREATE TABLE complex_dt (
        NAME STRING
        ,id BIGINT
        ,isStudent BOOLEAN
        ,ROLE VARCHAR(64)
        ,salary DECIMAL(8,2)
        ,phones ARRAY <INT>
        ,Course MAP <CHAR,FLOAT>
        ,address STRUCT <street:STRING,city:STRING,state:STRING,zip:INT>
        ,Teacher UNIONTYPE <FLOAT,BOOLEAN,STRING>
        ,misc BINARY
        ) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001' COLLECTION ITEMS
TERMINATED BY '\002' MAP KEYS TERMINATED BY '\003' LINES TERMINATED BY '\n';
```

Hive External vs Internal Tables

In Hive tables can be created as EXTERNAL or INTERNAL (managed tables). The choice of how the table is created affects how data is loaded, controlled, and managed.

Key Differences

Internal table data stores in Warehouse folder, External table data points to location you mentioned in table creation. So when you delete the internal table it delete Schema as well as Data under Warehouse folder, but in external table its only Schema going to lose. When you want table back you can create a table with schema again and point the location.

With external tables, Hive does not delete data from HDFS when you drop the base table. It doesn’t also rename the directory when the base table is renamed. Furthermore, external tables can't be archived and hive will not do any operations that affect the underlying files if the tables is declared external.

Hive EXTERNAL table

– External tables are preferred when the data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.

– Data remains in the underlying location even after a DROP TABLE. This can apply if you are pointing multiple schemas (tables or views) at a single data set or if you are iterating through various possible schemas.

– In external tables, if you drop it, it deletes only schema of the table, table data exists in physical location. So to delete the data use *hadoop fs -rmr table_name*.

Hive INTERNAL (Managed) table

– Hive completely manages the lifecycle of the table and data.

– Unlike external tables where users have control on it, in Managed tables, hive has full control on table.

– When you drop an internal table, it drops the data, and it also drops the metadata. (So, use caution when dropping Hive INTERNAL tables).

HiveQL and Examples

Hive – Data Definition Language – DDL

Create Managed Table

```
hive> CREATE TABLE managed_table (Region STRING);
        LOAD DATA INPATH '/user/Foo/data.txt'
        INTO TABLE managed_table;

--Note that LOCATION is not specified, and so hive uses the default Warehouse
location
```

Create External Table

```
hive> CREATE TABLE external_table (Region STRING)
        LOCATION '/user/Foo/external_tab'
        LOAD DATA INPATH '/user/Foo/data.txt'
        INTO TABLE external_table;

--Note that LOCATION is now specified, and so hive uses it.
```

Create database

```
hive> CREATE Databases Sales;
```

Create schema

```
hive> CREATE SCHEMA Sales.Customers;
```

Create Tables

```
hive> CREATE TABLE Sales.Customers.customer (
        age INT
        ,address STRING
        );

--In this case, customer is the table name.
```

Dropping Tables

```
hive> DROP TABLE IF EXISTS employees;
```

Renaming a Table

```
hive> ALTER TABLE employees RENAME TO guests;
```

Adding, Modifying and Dropping a table partition (aka. Partition swapping)

```
hive> ALTER TABLE customers ADD  --Add New Partition
        IF NOT EXISTS PARTITION (
                year = 2016, month = 01, day = 01
                ) LOCATION '/customers/2016/01/01';

hive> ALTER TABLE customers PARTITION (--Change partition location
        year = 2016, month = 01, day = 01
        ) SET LOCATION '/customers/2016/01/01;

hive> ALTER TABLE customers  (--Drop existing partition
        DROP IF EXISTS PARTITION (
                year = 2016, month = 1, day = 1);
```

Changing Columns

```
hive> ALTER TABLE customers CHANGE COLUMN oldname newname COMMENT 'Customer
New Name' AFTER someothercolumnname;
```

Adding Columns

```
hive> ALTER TABLE customers ADD columns (
        firstname STRING COMMENT 'First Name'
        ,lastname STRING COMMENT 'Last Name'
        );
```

Deleting or Replacing columns

```
hive> ALTER TABLE customers REPLACE columns (
        newfirstname STRING COMMENT 'My New First Name'
        ,newlastname STRING COMMENT 'My New Last Name'
        );
```

Note:

- Replaces original columns in the table with the new defined columns
- Can only be used in tables that use the native SerDe: DynamicSerDe.

Alter table properties

```
hive> ALTER TABLE customers SET TBLPROPERTIES('notes' ='This tables houses
customers information AND IS refreshed daily');
```

Alter Storage properties

```
hive> ALTER TABLE customers PARTITION (
        year = 2016
        ,month = 01
        ,day = 01
        )SET FILEFORMAT SEQUENCEFILE;
```

Partitions

```
hive> CREATE TABLE customer (
        age INT
        ,address STRING
        ) PARTITIONED BY (STATE string);
```

Show table

```
hive> SHOW Tables;
```

Describe table

```
hive> Describe Customer;
```

Alter Table

```
hive> ALTER TABLE customer ADD columns (age INT);
```

Drop Table

```
hive> DROP TABLE Customer;
```

More Hive queries

Exchange Partitions

```
hive> ALTER TABLE Region1_Logs exchange PARTITION (Year = '2015')
        WITH TABLE Region2_Logs
```

The semantics of the above statement is that the data is moved from the target table to the source table. Both the tables should have the same schema. The operation fails in the presence of an index. The source table should not have that partition.

Archive Partitions

```
hive> ALTER TABLE ... ARCHIVE PARTITION;
```

Archiving is a feature to moves a partition's files into a Hadoop Archive (HAR). Note that only the file count will be reduced; HAR does not provide any compression.

Touch Partitions

```
hive> ALTER TABLE ... TOUCH PARTITION;
```

TOUCH reads the metadata, and writes it back. This has the effect of causing the pre/post execute hooks to fire. An example use case is if you have a hook that logs all the tables/partitions that were modified, along with an external script that alters the files on HDFS directly. Since the script modifies files outside of hive, the modification wouldn't be logged by the hook. The external script could call TOUCH to fire the hook and mark the said table or partition as modified.

Hive Hooks

Hooking involves techniques for intercepting function calls or messages or events in an operating system, application, and other software components.

A Hook is code that handles intercepted function calls, events, or messages

More on hooking available at: http://www.slideshare.net/julingks/apache-hive-hooks-min-wookim130813

Hive – Data Modification Language (DML)

Hive Tables

Simple Select statement: Similar to SQL

```
hive> SELECT a.age FROM customer a WHERE a.LastName = 'Louie';
```

Loading flat files into Hive

```
hive> LOAD DATA LOCAL INPATH './data/home/myfile.txt' OVERWRITE INTO TABLE
Customer;
```

Note:

- There is no verification of incoming Data
- LOAD DATA LOCAL ... *copies* the local data to the final location in HDFS, while LOAD DATA ... (i.e. without LOCAL) *moves* the data to the final destination. LOCAL says the file is a file in your regular file system (not HDFS)
- Default column delimiter is \A and row delimiter \n, so typical files become a table having a single column and as many rows as lines

Insert Data into Tables from Queries

```
hive> INSERT OVERWRITE TABLE employees PARTITION (
        firstname = 'fur'
        ,lastname = 'edn'
        ) SELECT * FROM src_employees emp;
```

Note 1: The OVERWRITE keyword causes hive to overwrite all data in the table. Ignore it if you just want to append the data.

Dynamic Partition Inserts

```
hive> INSERT OVERWRITE TABLE employees PARTITION (
        firstname
        ,lastname
        ) SELECT ..., emp.firstname, emp.lastname FROM src_employees emp;
```

Note: The following is very important for dynamic partitioning

All Partition columns (firstname, lastname) MUST be the last columns in the select statement.

Properties to enable Dynamic Partition Inserts

Use the following properties in the query to allow dynamic partition.

```
hive> SET hive.EXEC.DYNAMIC.PARTITION = true; --Enables DYNAMIC partitioning

hive>SET hive.EXEC.DYNAMIC.PARTITION.mode = nonstrict; --Enables partitioned
to be determined dynamically

hive>SET hive.EXEC.max.DYNAMIC.partitions.pernode = 1000; --The number OF par-
titions allowed TO be created BY each mapper.
```

Creating and Loading tables in One Query

```
hive> CREATE TABLE sub_employees AS
        SELECT firstname
               ,lastname
               ,address
        FROM employees
        WHERE emp.type = 'sub';
```

Exporting data from Hive

Write results from a select statement to an HDFS directory

```
hive> INSERT OVERWRITE DIRECTORY '/directory/customerResults'
      SELECT a.*
      FROM customer a
      WHERE a.Birthdate >= '01/01/1980';
```

To Single Directory

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/user/tmp_employees'
        SELECT firstname
               ,lastname
               ,address
        FROM employees emp
        WHERE emp.Type = 'tmp';
```

To Multiple Directories/Files

```
hive > FROM src_employees emp

        INSERT OVERWRITE DIRECTORY '/user/tmp_employees'
        SELECT *
        WHERE emp.Type = 'temp'

        INSERT OVERWRITE DIRECTORY '/user/full_employees'
        SELECT *
        WHERE emp.Type = 'full'

        INSERT OVERWRITE DIRECTORY '/user/exec_employees'
        SELECT *
        WHERE emp.Type = 'executive';
```

Hive Views

Used to reduce complex queries

Create View

```
hive> CREATE VIEW IF NOT EXISTS top_customers AS
        SELECT *
        FROM customers cust
        WHERE cust.ranking >= 89;
```

HiveQL Indexes

Creating Indexes

```
hive> CREATE INDEX employees_index ON TABLE empoloyees (country) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
        WITH DEFERRED REBUILD IDXPROPERTIES('creator = ' me ')
        IN TABLE employees_index_table
        PARTITIONED BY (country, name);
```

BitMap Indexes

```
hive> CREATE INDEX employees_index ON TABLE empoloyees (country) AS 'BITMAP'
        WITH DEFERRED REBUILD IDXPROPERTIES('creator = 'me') IN TABLE
        employees_index_table PARTITIONED BY (country, name);
```

Rebuilding Indexes

```
hive> ALTER INDEX employees_index ON TABLE employees PARTITION (country =
'name') REBUILD;
```

Showing Indexes

```
hive > SHOW FORMATTED INDEX ON employees;
```

Dropping Indexes

```
hive > DROP INDEX IF EXISTS employees_index ON TABLE employees;
```

Hive Queries that sample Data

```
hive > SELECT * FROM customers TABLESMAPLE(BUCKET 1 OUTPUT OF 2 ON customerid) cust;

hive > SELECT * FROM customers TABLESMAPLE(BUCKET 1 OUTPUT OF 2 ON rand()) cust; --Bucket Sampling

hive >SELECT * FROM customers TABLESMAPLE(0.1 PERCENT) cust;
```

HiveQL Table Generating Functions

Normal user-defined functions, such as CONCAT (), take in a single input row and output a single output row. In contrast, table-generating functions transform a single input row to multiple output rows. Hive table generating functions include; Explode, Split, Lateral View.

```
hive> SELECT pageid
        ,adid
        FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

Hive's table generating functions allow a single row to expand to multiple rows; it takes an array and generate a row for each item in the array (split is a function that splits a string into an array)

HiveQL User Defined Functions (UDFs)

```
hive> ADD JAR MyUDFs.jar;
        CREATE TEMPORARY FUNCTION net_salary AS 'com.example.MyUDFs';
        SELECT EmpName, net_tax(salary, deductions) FROM employees;
```

With UDFs, it's easy for users to build their own custom functions, put in a jar, and then use them in queries.

The above UDF function takes the employee's salary and deductions, then computes the net tax.

Hive Calling Out to External Programs

In hive, it is possible to call out to external programs to perform map and reduce operations. See example below.

```
hive> FROM (
        FROM logs MAP message USING '/user/Jane_Doe' AS log_feed
                ,count CLUSTER BY log_feed
        ) it

        INSERT OVERWRITE TABLE Log_Data REDUCE lg.error_name
        ,lg.count USING '/user/log_analyzer.py' AS log_feed
        ,counts;
```

Note the MAP…USING and REDUCE…USING.

Key Hive Query Optimization Tips

Hive from the ground up is designed like SQL engines do. But given the challenges that come with working on the Massively Parallel Processing (MPP) architecture, it turns out that Hive in some cases does not work like SQL engines.

So, special techniques must be employed to tune and optimize hive queries so they can run efficiently. Let's discuss such best practices to get maximum performance benefits from hive.

Optimize ON JOINS

- When joining three or more tables, if every ON clause uses the same join key, a single MapReduce join will be used. Also note that Hive allows only equality Joins.
- Hive always assumes that the last table in the query is the largest. It attempts to buffer the other tables and stream the last table. So, the last table in queries should be the largest.
- Use small tables on the left side of your joins. Enable auto optimization, and take advantage of improved query performance with Map/Side joins.

```
hive> SET hive.auto.convert.inner.join = true;

-- This property causes hive at execution time to cache small tables and al-
lowfor mapper-only joins which are much faster.

hive> SET hive.mapjoin.smalltable.filesize = 1000000;

-- This property affects the size of what hive considers small enough to
cache. Note: This setting can only be limited by the amount of memory availa-
ble to the JVM

hive> SET hive.groupby.orderby.position.alias = true;

--If you want to be able to reference your column names by position (i.e.
group by 1,2) instead of by name (i.e. group by date, location), then use
this property.
```

- In every map/reduce stage of the join, the last table in the sequence is streamed through the reducers whereas the others are buffered. Therefore, it helps to reduce the memory needed in the reducer for buffering the rows for a particular value of the join key by organizing the tables such that the largest tables appear last in the sequence.
- Read more on Hive Joins at: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+J

Optimize on ORDER BY vs SORT BY

- SORT BY query produces multiple independent files. ORDER BY - just one file. This is because unlike SORT BY when doing ORDER BY, everything is piped through one reducer; an approach that might take a very long time for large data sets.
- In SORT BY, each reducer handles a subset of the data, so joining the files together would be globally sorted (in most cases).

```
--A total ordering - one reducer. (Slower)

hive>SELECT NAME
        ,salary
        FROM employees
        ORDER BY salary ASC;

--A LOCAL ordering - sorts within each reducer. (Faster!)

hive>SELECT NAME
         ,salary
         FROM employees SORT BY salary ASC;
```

- Always avoid using ORDER BY in queries. ORDER BY is slow and results in one final reducer sorting the data. Instead, use SORT BY or CLUSTER BY which organizes data on a per reducer (local) level and hence results in much faster queries.

Optimize On Partitions

- Partitioning works on both managed and external tables.
- Partitioned data are physically stored under separate directories. You can store the data in a subdirectory tree like year/month/day or if it is geographic data country/state/zipcode and so forth. That allows queries to quickly traverse and to skip the irrelevant data, saving lots of CPU time.
- When inserting data to partitioned tables, the partition columns must be specified. The partition columns, must be the last columns in the query statement.

```
hive > INSERT INTO TABLE partitioned_user PARTITION (country , STATE)
       SELECT firstname, lastname, address, country, STATE
       FROM temp_user;

--Notice how country and state used in the partition columns are listed as
the last columns in the select statement. Not sure why Hive was designed this
way, but that's just how it is. Ignore this rule, and your statement will
fail.
```

- Use caution not to over partition your data within Hive. It's important to consider the cardinality of the column that will be partitioned on. HDFS performs better when it has smaller set of large files instead of larger set of smaller files. Selecting a column with high cardinality will result in fragmentation of data and put strain on the name node to manage all the underlying structures in HDFS.

Best Practice Parameters for Loading and Querying Hive Partitioned Tables

```
hive> SET hive.mapred.mode = strict;

hive>SET hive.enforce.bucketing = true;

hive>SET hive.enforce.sorting = true;

hive>SET hive.EXEC.DYNAMIC.PARTITION.mode = nonstrict;

hive>SET hive.EXEC.max.DYNAMIC.partitions.pernode = 200000;

hive>SET hive.EXEC.max.DYNAMIC.partitions = 200000;

hive>SET hive.EXEC.max.created.files = 2000000;

hive>SET hive.EXEC.parallel  = true;

hive>SET hive.EXEC.reducers.max = 4000;

hive>SET hive.stats.autogather  = true;

hive>SET hive.OPTIMIZE.sort.DYNAMIC.PARTITION = true;

hive>SET mapred.job.reduce.input.buffer.PERCENT = 0.0;

hive>SET mapreduce.input.fileinputformat.split.minsizee = 300000000;

hive>SET mapreduce.input.fileinputformat.split.minsize.per.node = 300000000;

hive>SET mapreduce.input.fileinputformat.split.minsize.per.rack = 300000000;

--Of course, these numbers would not necessarily translate directly to your
cluster. You would need to examine and tune the values to match your particu-
lar cluster size and configuration.
```

Optimize On Bucketing

- Bucketed tables are fantastic in that they allow much more efficient sampling than do non-bucketed tables, and they may later allow for time saving operations such as mapside joins.
- Use the following command to enforce bucketing:

```
hive> SET hive.enforce.bucketing = true;
```

- Bucketing provides mechanism to query and examine random samples of data.
- With Bucketing, Data is broken into a set of buckets based on a hash function of a "bucket column".
- Bucketing offers capability to execute queries on a sub-set of random data
- Hive does not automatically enforce bucketing. User is required to specify the number of buckets by setting # of reducer.

Optimize On Statistics and Cost Based Optimization (CBO)

- Statistics such as the number of rows of a table or partition and the histograms of a particular interesting column are important in many ways. One of the key use cases of statistics is query optimization.
- For newly created tables and/or partitions (that are populated through the INSERT OVERWRITE command), statistics are automatically computed by default. The user has to explicitly set the Boolean variable hive.stats.autogather to false so that statistics are not automatically computed and stored into Hive MetaStore.

```
hive> SET hive.stats.autogather = false;
```

- Another feature of Hive 0.13 and beyond is support for Cost Based Optimization (CBO) that leverages statistics collected on Hive tables. To collect statistics for the Hive query optimizer, set two parameters:

```
hive> SET hive.COMPUTE.query.using.stats = true;
hive> SET hive.stats.dbclass = fs;
```

- For existing tables and/or partitions, the user can issue the ANALYZE command to gather statistics and write them into Hive MetaStore. The syntax for that command is described below:

```
hive> Analyze TABLE t [partition p] COMPUTE STATISTICS
FOR [columns c,...];
```

- Statistics serve as the input to the cost functions of the optimizer so that it can compare different plans and choose among them.

- Statistics may sometimes meet the purpose of the users' queries. Users can quickly get the answers for some of their queries by only querying stored statistics rather than firing long-running execution plans.

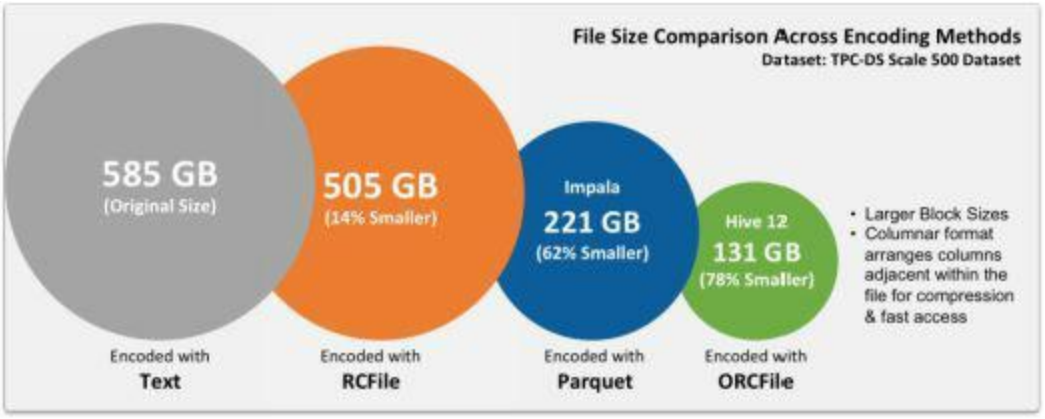
Optimize On Compression

- Text is where Hive started, but it has evolved into handling just about any text file. Today, ORC (Optimized Row Columnar) files are the new, highly optimized format for data that was recently released.
- Using ORCFile or converting existing data to ORCFile is simple. To use it just add STORED AS orc to the end of your create table statements like this:

```
hive> CREATE TABLE server_logs (
        ip STRING
        ,hostname STRING
        ,OS STRING
        ,load_date DATE
        ) PARTITIONED BY (load_date STRING) STORED AS ORC TBLPROPERTIES
       ("orc.compress" = "SNAPPY");

--This command creates a table stored ORC Hive. It also uses Snappy compres-
sion instead OF Zlib.
```

- ORC files predate Hive 0.13. The ORC format first showed up in Hive 0.11, but they work particularly well in conjunction with the new features of Hive 0.13.



Source: http://hortonworks.com/blog/orcfile-in-hdp -2-better-compression-better-performance/

- ORC (Optimized Row Columnar) is a columnar file format optimized to improve performance of Hive. Inserting data into

ORC compressed tables might be a little slower, but if you want good query speed, while compressing the data as much as possible, then ORC is for you.

Optimize On Vectorization

- Vectorization allows Hive to process a batch of rows together as a unit instead of processing one row at a time. When used, Vectorization greatly reduces the CPU usage for typical query operations like scans, filters, aggregates, and joins. A standard query execution system processes one row at a time. A vectorized query execution on the other hand streamlines operations by processing a block of 1024 rows at a time.
- To use vectorized query execution, the data must be stored in ORC format. Vectorized execution is turned off by default, so your queries only utilize it if this variable is turned on. To turn vectorization on, use the following property.

```
hive> SET hive.vectorized.execution.enabled = true;
```

- To disable vectorized execution and go back to standard execution, do the following:

```
hive> SET hive.vectorized.execution.enabled = false;
```

- More details on Hive Vectorization can be found at this link: https://cwiki.apache.org/confluence/display/Hive/Vectorized+Que

Optimize On Queues and Schedulers

- In most cases, as Hadoop usage grows and many users take on the tasks of executing Hive queries on the Hadoop cluster, contention for the Hadoop resources will inevitably become an issue. In order to mitigate this issue, Hive provides the concepts of queues and schedulers multi-tenant use and workload management.
- There are two prominent schedulers for multi-user workloads in Hive: the Fair Scheduler, developed at Facebook, and the Capacity Scheduler, developed at Yahoo. (Setting up queues and schedulers is beyond the scope of this book. More information on that can be found at this links: http://blog.cloudera.com/blog/2008/11/job-scheduling-in-hadoop/)
- Once you have the scheduler and queues setup, you can then manage users queries and direct their workloads to a particular queue by using this property.

```
hive> SET hive.job.queuename = elt;

--In this case, the job runs under the "elt" queue, and as such, only has ac-
cess to the capacity allocated to that queue. When used correctly, queues
can prevent users from sending rogue queries that can clog cluster resources
and hold up other more valuable jobs from running.
```

- In a cluster that is setup to use queues, if you do not specify the queue name in your job options, the job will be submitted to the default queue. Any jobs submitted to the default queue will have a lower priority than jobs submitted to other queues.

Hive Functions

Like with traditional SQL, there are lots of pre-built functions for performing analytical tasks within Hive. These range from simple Date functions, Mathematical functions, String functions, to more advanced windowing functions that can be used for Ranking, Aggregation, Navigation Time Series Analysis, Allocations, Data Densification and Linear Regression.

Simple functions can even be chained for more complex analysis.

The list below provides links to articles that lists all built-in aggregate functions. Because these list might constantly evolve and change, it's better to provide the raw URLs and point you to the direct source rather than pasting the functions in here. Please visit the links to check out the functions.

Date Functions

- http://docs.treasuredata.com/articles/hive-functions#date-functions

Mathematical Functions

- http://docs.treasuredata.com/articles/hive-functions#mathematical-functions

Arithmetic Operators

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LangLogicalOperators

Aggregate functions

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LangBuilt-inAggregateFunctions(UDAF)

Table Generating Functions

- Explode
- http://docs.treasuredata.com/articles/hive-functions#collection-functions

String Functions

- http://docs.treasuredata.com/articles/hive-functions#string-functions

Misc Functions

- http://docs.treasuredata.com/articles/hive-functions#misc-functions

Resources

- Apache Hive Project
  - http://hive.apache.org/
- Apache Hive Wiki
  - https://cwiki.apache.org/confluence/display/Hive/Home
- Apache Hive On GitHub
  - https://github.com/apache/hive
- Books
  - Capriolo, Edward, Dean Wampler, and Jason Rutherglen. *Programming Hive*. Sebastopol, CA: O'Reilly & Associates, 2012. Print.
  - Lam, Chuck. *Hadoop in Action*. Greenwich, CT: Manning, 2011. Print. (Chapter About Hive)
  - Holmes, Alex. *Hadoop in Practice*. N.p.: Manning Publications, 2014. Print. (Chapter About Hive)

Final Word

I want to thank you and congratulate you for downloading the book “The Definitive Guide to Programming Apache Hive".

This book contains proven steps and strategies on how to become a truly a great Apache Hive Programmer. You will be able to increase your profitability, areas of opportunity, and so much more by unleashing the skills you’ve learned

Here’s an inescapable fact: you will need to practice your way to mastery.

Because Hive is basically SQL on Hadoop, you would need to develop your SQL skills along the way.

If you do not develop your SQL skills, you will not master Hive. SQL is at the heart of Hive, and we hope this book has helped you get on the right path.

It’s time for you to become an amazing Big Data professional, who can crunch massive amounts of data with the help of the information compiled within this book.