
hadoop
HIVE

APACHE HIVE

Objectives

In this module, we are going to look at the following topics:

- ✓ Understanding Hive
- ✓ Hive Architecture
- ✓ Hive Components
- ✓ Hive Data Types – Primitive & Complex
- ✓ Understand Hive Data Model
- ✓ Managed & External Tables
- ✓ Table Partitioning and Bucketing
- ✓ Basic HiveQL Operations

What is Hive ?

Hive is a distributed data warehousing infrastructure built on top of Hadoop

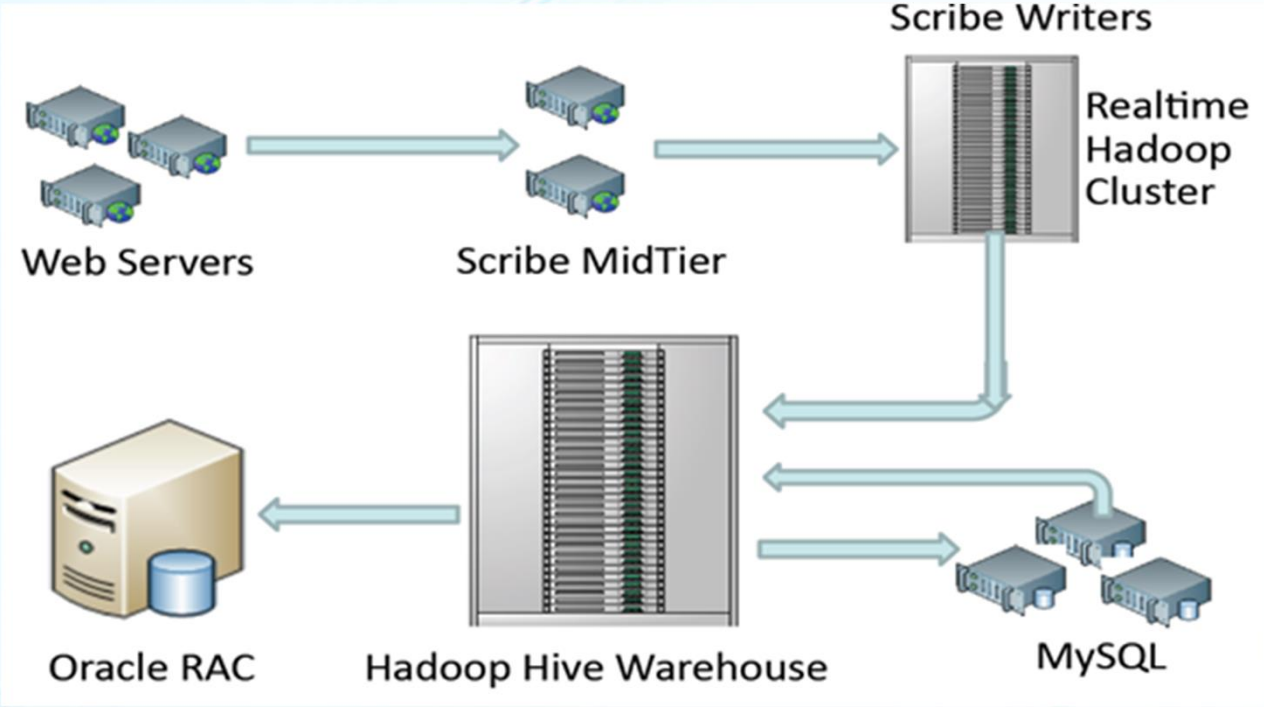
- Easy data summarization
- Analysis of large volume of data
- Ad-hoc Querying
- Targeted for Data Analysts

What is Hive ?

- Data Warehouse Infrastructure
- Used for Data Analysis
- Can leverage existing SQL expertise
- Provides a dialect of SQL called HQL
- An abstraction on top of MapReduce. So there is no need to learn Java or Hadoop APIs
- Allows programmers to plug-in custom functionality such as UDFs
- Facebook uses Hive to analyze several TBs of data every day.

Background

- Early Hive development work started at Facebook in 2007
- Later open-sourced to Apache under Hadoop Project
- Today, Hive is a top-level project at Apache



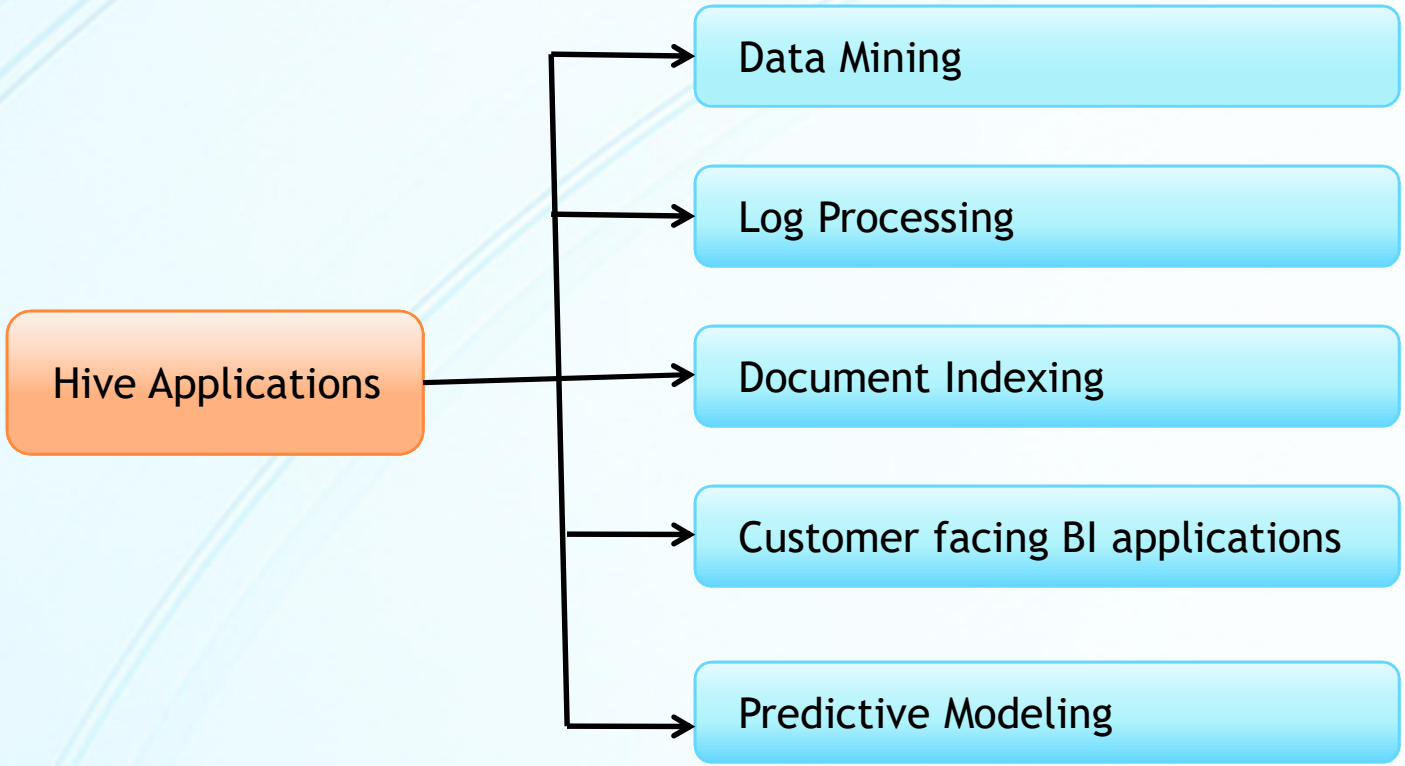
Advantages of Hive

- SQL like programming
- Schema Flexibility (Schema on read)
- Supports Table Partitioning and Bucketing
- Hive tables directly defined on HDFS
- Extensible Types, Formats and Functions
- JDBC / ODBC drivers

Limitations of Hive

- Not designed for OLTP
- Does not offer real-time queries and row-level updates
- Does not provide optimal latency for interactive data browsing
- Latency for Hive queries is generally high

Hive Applications



Hive Vs Pig

- PigLating is a data flow language

```
Ex: data = load 'data-file.txt';
    dump data;
```

- Pig is used by Programmers and Researchers

- Hive is uses an SQL like language

```
Ex: SELECT * FROM data_table;
```

- Hive is used by Data Analysts who are good in SQL

Hive Vs Pig

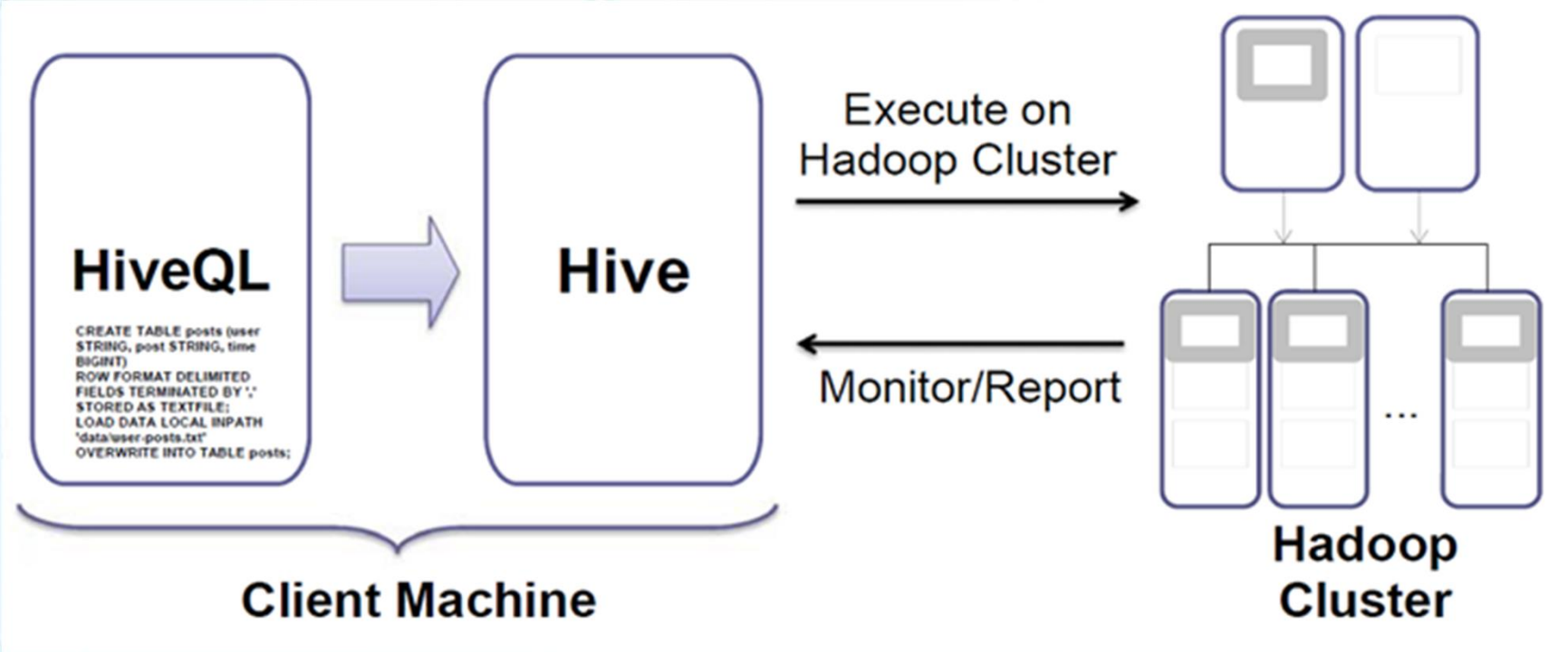
Feature	Pig	Hive
Language	PigLatin	Hive QL
Schema	Implicit	Explicit
Partitions	No	Yes
Custom SerDe	Yes	Yes
DFS Direct Access	Yes (Explicit)	Yes (Implicit)
Join, Order & Sort	Yes	Yes
Shell	Yes	Yes
UDFs	Yes	Yes
Web UI	No	Yes
JDBC / ODBC	No	Yes (limited)

Hive Vs Traditional RDBMS

- Hive validates schema on read
  - Does not verify data when loaded
  - Load operation is just a file copy or move
  - Very fast initial load
- DML operations such as single row inserts, updates and deletes were not supported until Hive 0.14.
- As of Hive version 0.14.0 - INSERT...VALUES, UPDATE, and DELETE are available with full ACID support.

Hive Architecture

Translates HiveQL statements into a set of MapReduce jobs which are then executed on a Hadoop Cluster



Hive Architecture

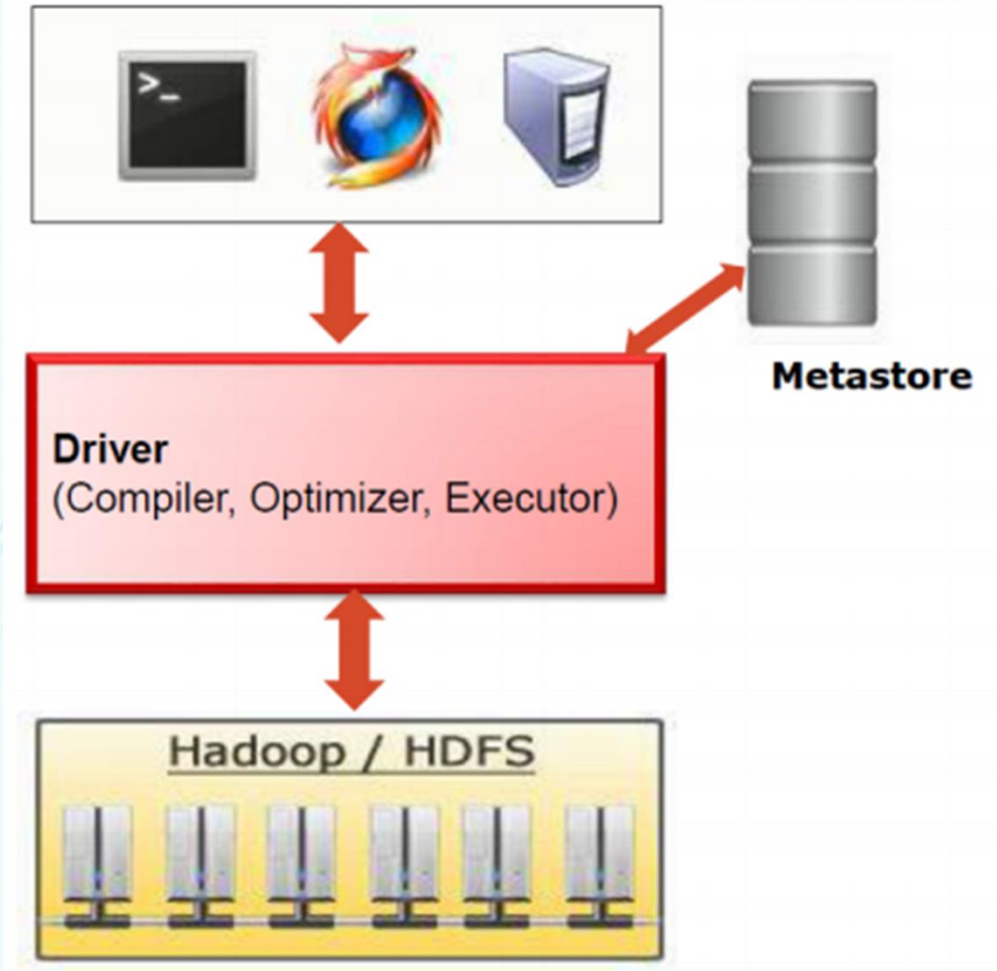
Interfaces

Driver

Compiler

Optimizer

Executor



Metastore

Components of Hive Architecture

Driver

Compiler

Executor

Metastore

Optimizer

Interfaces

Components - Driver

Driver

- Acts like a controller which receives the HiveQL statements.
- It starts the execution of the statement by creating sessions, and monitors the life cycle and progress of the execution.
- It stores the necessary metadata generated during the execution of a HiveQL statement.
- Driver also acts as a collection point of data or query results obtained after Reduce operation.

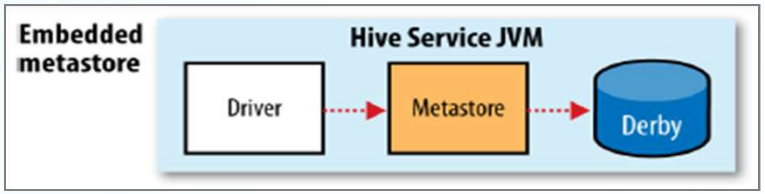
Components – Meta Store

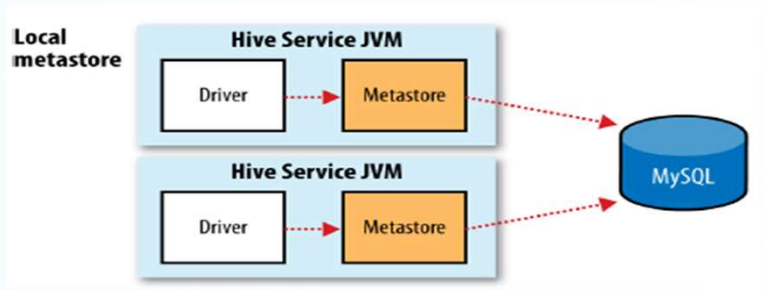
Stores system catalog and metadata about tables, columns, partitions etc.

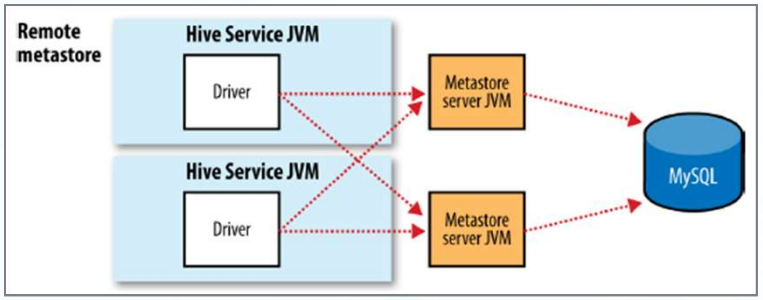
The data is stored in a traditional RDBMS format.

By default uses Apache Derby as Metastore. Can also use MySQL or other relational databases.

Metadata helps the driver to keep track of crucial data. Backup server regularly replicates the data.







Components - Compiler

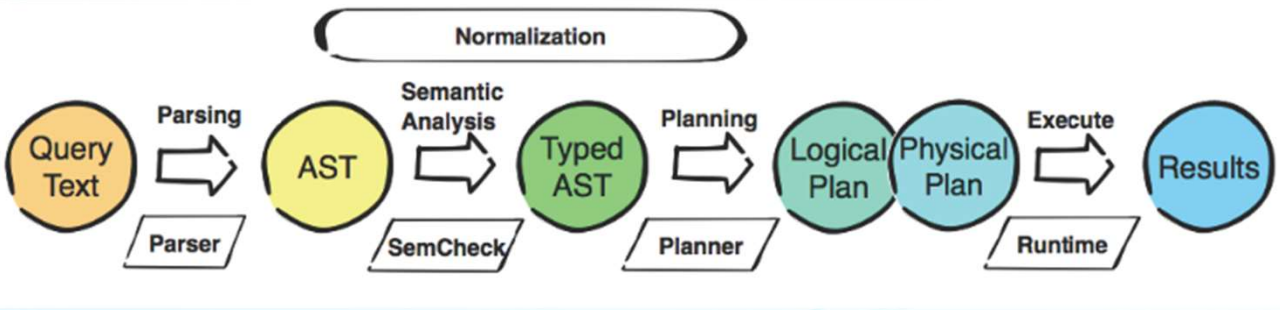
Compiler

Performs compilation of the HiveQL query, which converts the query to an execution plan.

The compiler converts the query to an abstract syntax tree (AST)

After checking for compatibility and compile time errors, it converts the AST to a directed acyclic graph (DAG)

The DAG divides operators to MapReduce stages and tasks based on the input query and data.



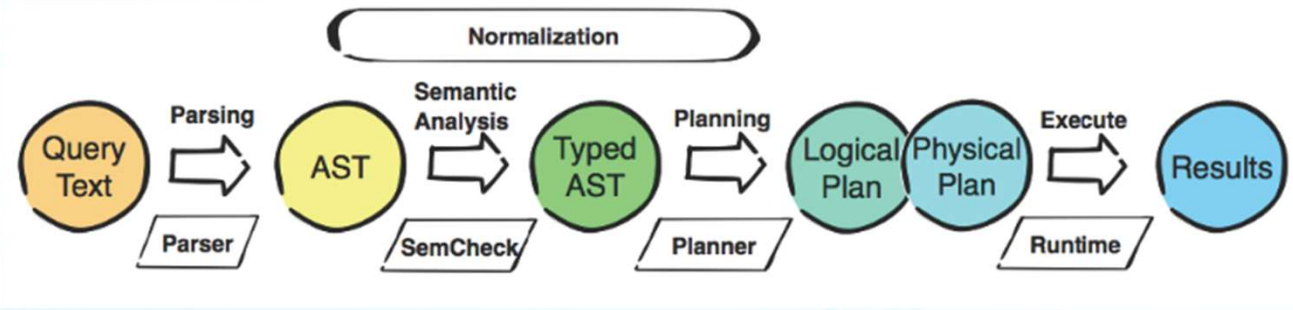
Components - Optimizer

Performs various transformations on the execution plan to get an optimized DAG.

Transformations can be aggregated together, such as converting a pipeline of joins to a single join, for better performance.

It can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance and scalability.

Optimizer



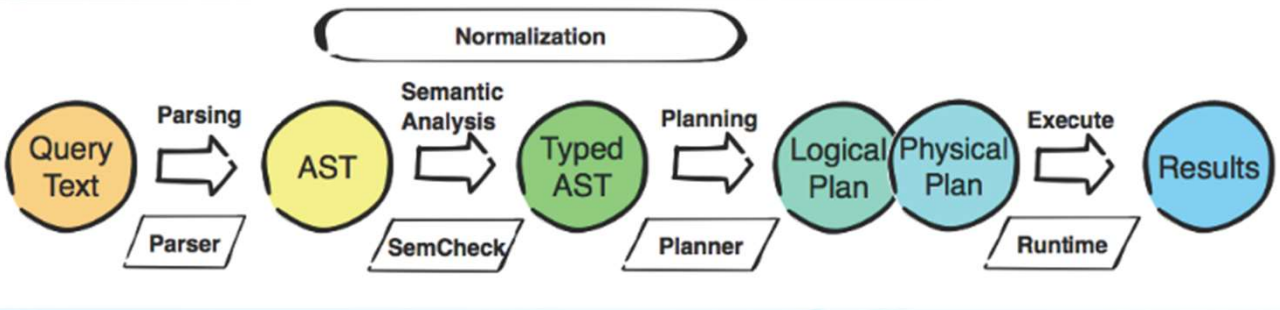
Components – Executor

Executor

After compilation and optimization, the executor executes the tasks.

It interacts with the processing daemons of Hadoop to schedule tasks to be run.

It takes care of pipelining the tasks by making sure that a task with dependency gets executed only if all other prerequisites are run.

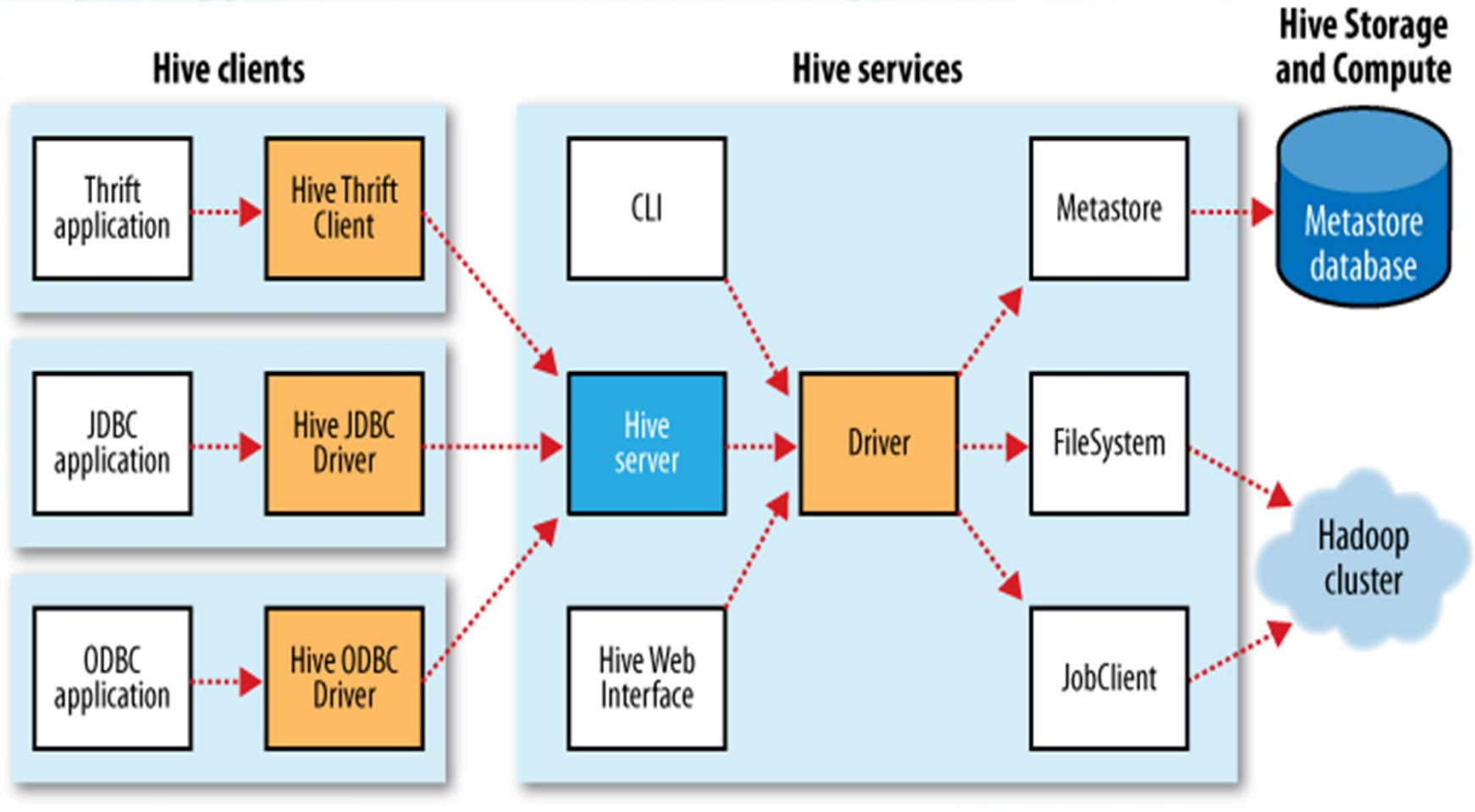


Components – Interfaces

- Command-line interface (CLI)
- JDBC / ODBC Drivers
- Thrift server

Interfaces

Hive Architecture



Hive Shell & HiveQL

The hive shell is the primary way that we will interact with hive

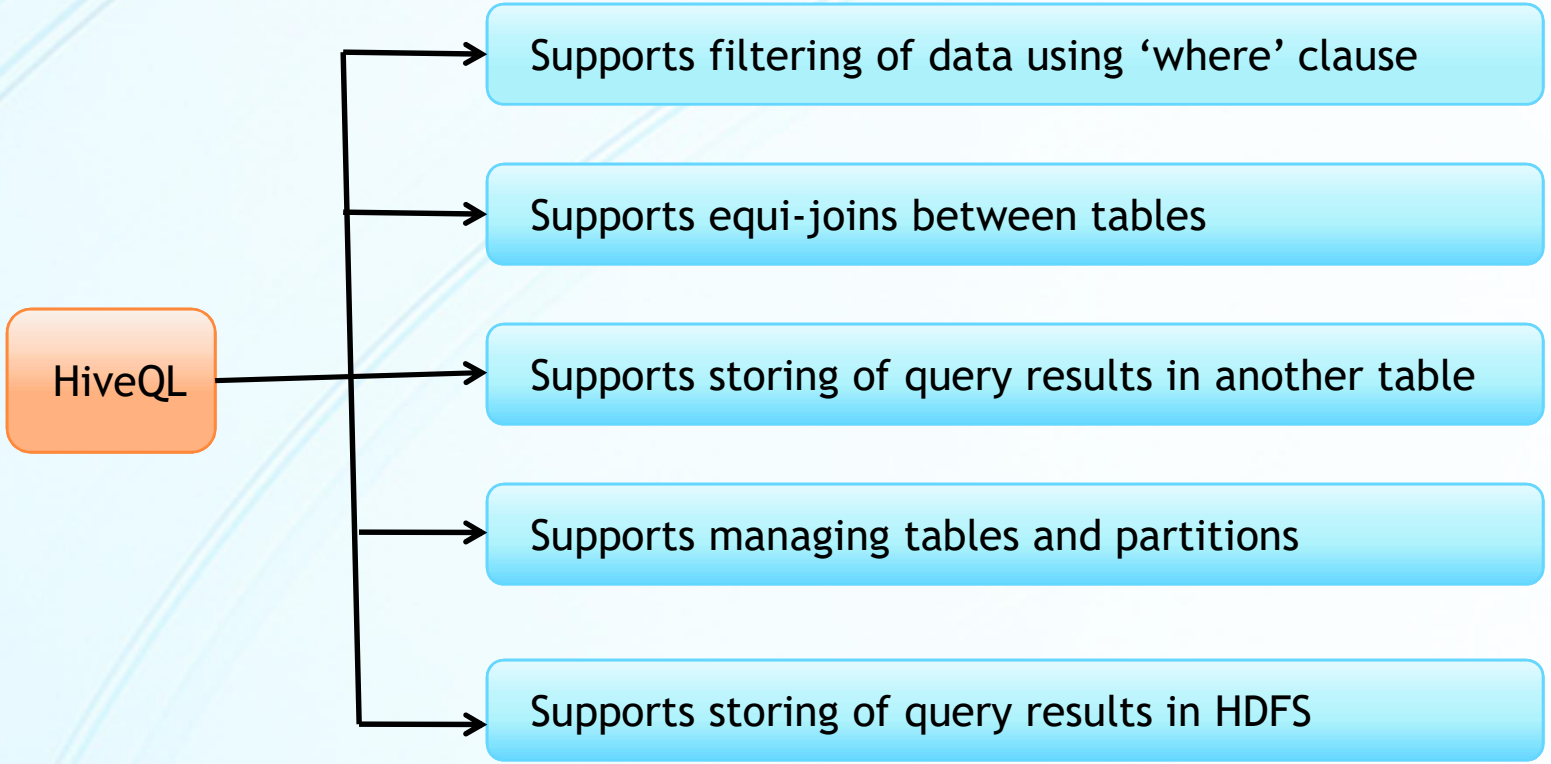
Hive CLI (old) & Beeline CLI (new)

HiveQL is the Hive's Query Language

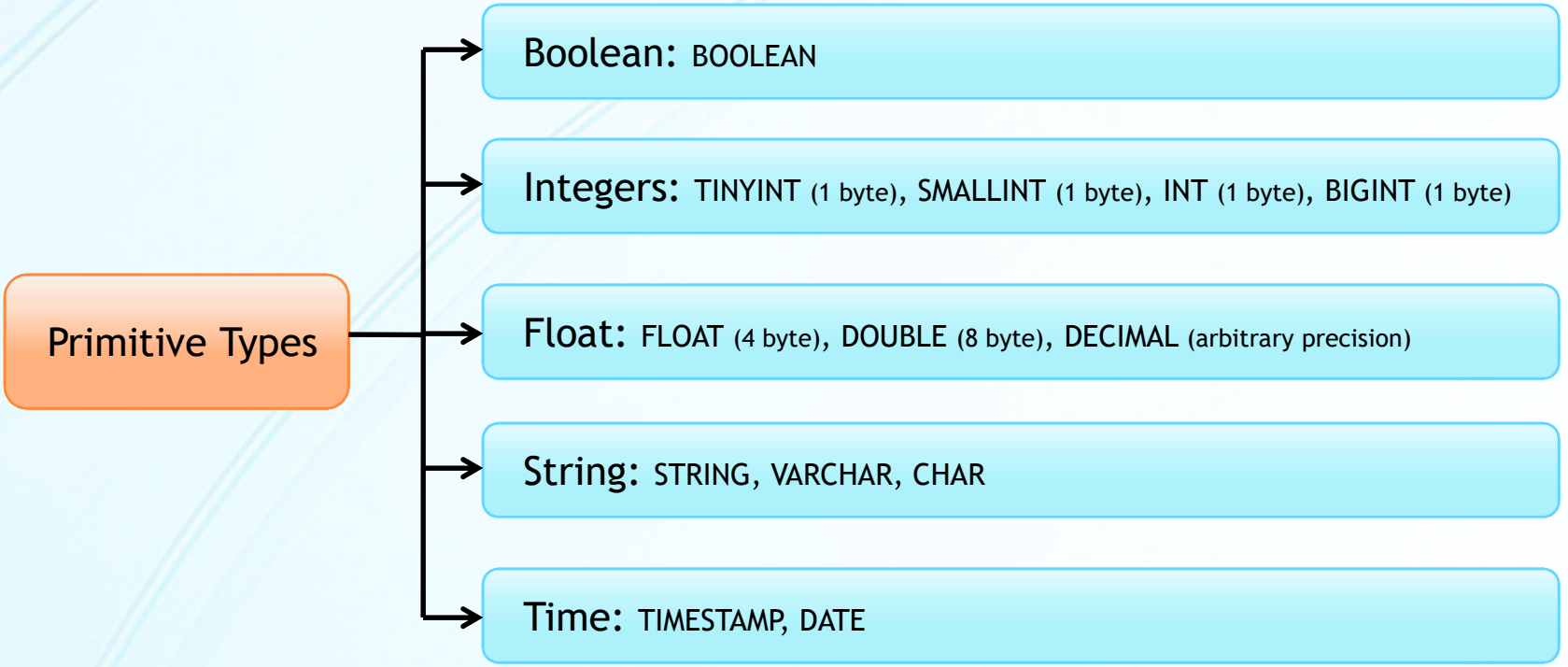
HiveQL is generally case sensitive

The hive shell can be run in non-interactive mode also.
The –f option runs the commands in the specified script file.

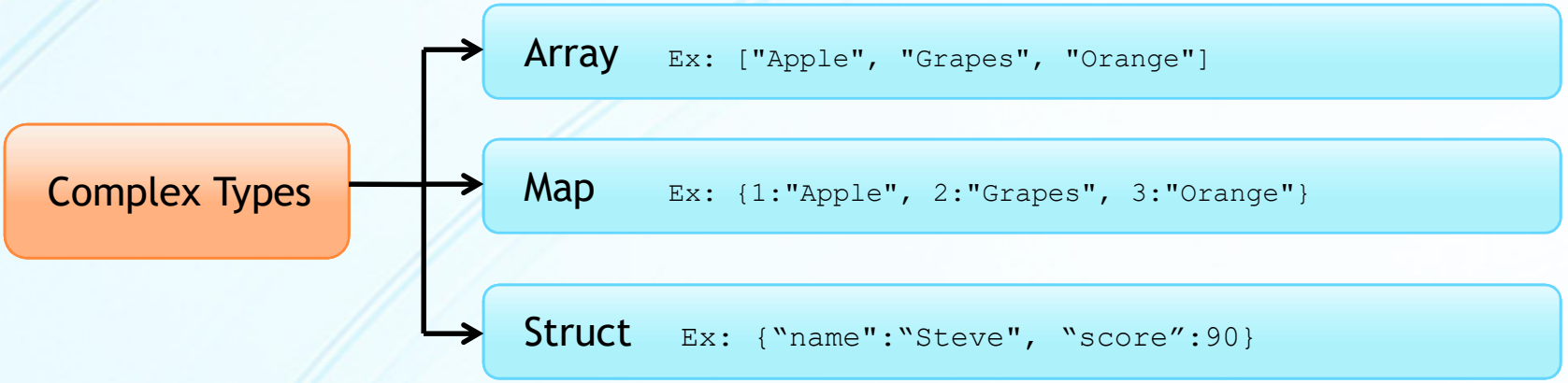
Hive Query Language



HiveQL Data Types



HiveQL Data Types



Hive Data Model

- Databases
- Tables
- Partitions
  - Grouping data based on some column(s)
  - Used for faster querying
- Buckets or Clusters
  - Partitions further divided into buckets based on some other column
  - Use for sampling of data

Let’s start working in Hive

Getting into the Hive Shell

Hive CLI

beeline shell

HiveServer2 supports a command shell Beeline that works with HiveServer2. It's a JDBC client that is based on the SQLLine CLI.

```
[cloudera@quickstart ~]$ beeline
beeline> !connect jdbc:hive2://quickstart:10000 cloudera cloudera org.apache.hive.jdbc.HiveDriver
0: jdbc:hive2://quickstart:10000> show databases;
+----------------+--+
| database_name  |
+----------------+--+
| default        |
+----------------+--+
1 row selected (0.623 seconds)
```

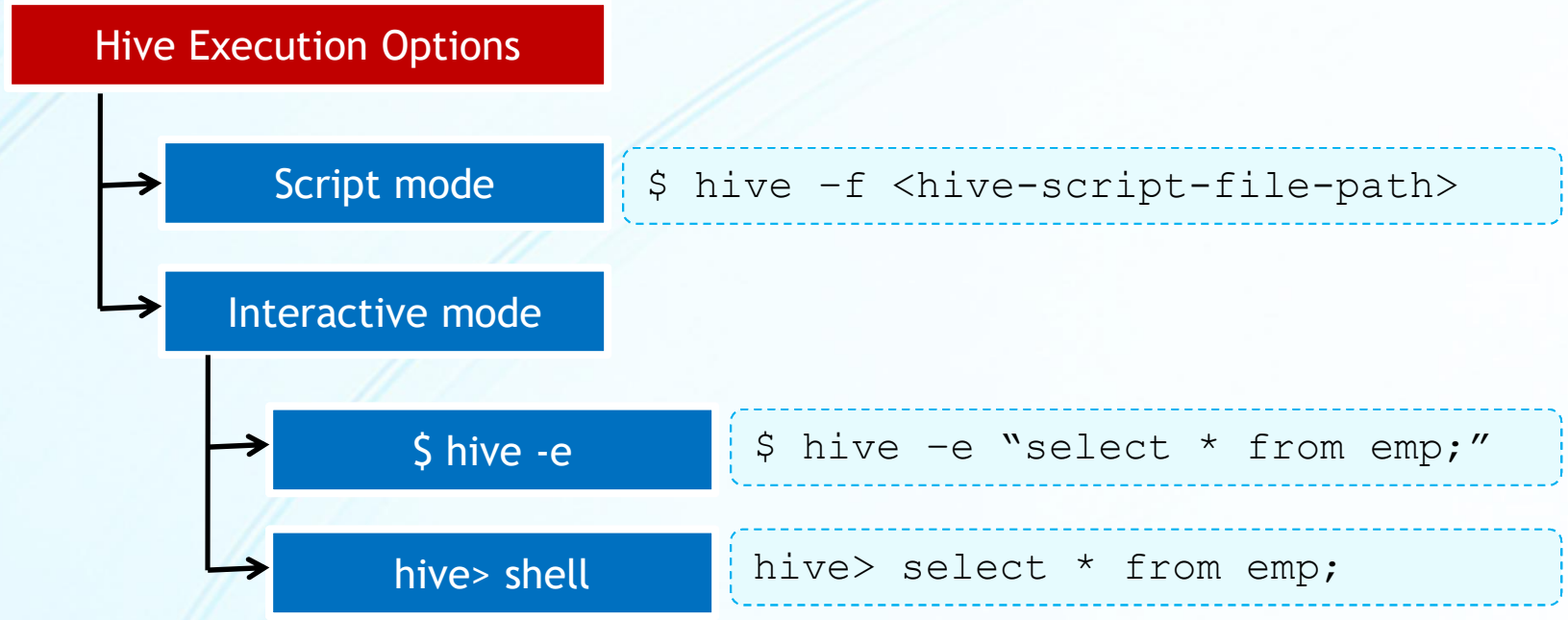
hive shell

```
[cloudera@quickstart ~]$ hive
hive>
```

Hive Shell vs. Beeline Shell

- 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 also uses the Hive JDBC driver but instead executes queries through HiveServer2, which allows multiple concurrent client connections and supports authentication.

Hive Execution Options



Create Database

```
hive> show databases;
hive> create database demodb;
hive> use demodb;
```

List existing databases

Create a new database

Use the created database

```
hive> show databases;
OK
default
Time taken: 5.206 seconds, Fetched: 1 row(s)
hive> create database demodb;
OK
Time taken: 0.333 seconds
hive> show databases;
OK
default
demodb
Time taken: 0.135 seconds, Fetched: 2 row(s)
hive> use demodb;
OK
Time taken: 0.101 seconds
```

Create Managed Table

```
CREATE TABLE IF NOT EXISTS emp
(empid INT, ename STRING, gender CHAR(1), age INT,
 country STRING, salary BIGINT, deptid TINYINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ","
LINES TERMINATED BY "\n"
STORED AS TEXTFILE;
```

Specifies how the file should be parsed

Specifies how the output file should be stored

```
hive> CREATE TABLE IF NOT EXISTS emp (
    > empid INT, ename STRING, gender CHAR(1), age INT, country STRING, salary BIGINT, deptid TINYINT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ","
    > LINES TERMINATED BY "\n"
    > STORED AS TEXTFILE;
OK
Time taken: 0.253 seconds
```

Supported Storage Formats

STORED AS TEXTFILE
STORED AS SEQUENCEFILE
STORED AS ORC
STORED AS PARQUET
STORED AS AVRO
STORED AS RCFILE
STORED AS JSONFILE

Load data into Table

```
LOAD DATA LOCAL INPATH 'datasets/emp.csv'
OVERWRITE INTO TABLE emp;
```

→ Load local (Linux) file into the table

→ Overwrite table data

```
LOAD DATA INPATH 'hdfs_file_path'
INTO TABLE tablename
```

→ Load HDFS file into the table

→ Append table data

```
hive> LOAD DATA LOCAL INPATH 'datasets/emp.csv'
    > OVERWRITE INTO TABLE emp;
Loading data to table demodb.emp
Table demodb.emp stats: [numFiles=1, numRows=0, totalSize=707, rawDataSize=0]
OK
Time taken: 2.492 seconds
```

- `LOCAL INPATH` option lets you load a file from local file system
- `INPATH` option lets you load a file from HDFS file system
- `OVERWRITE INTO` option will delete old data files before loading data
- `INTO` option will append data to the table i.e adds a new data file to the table

Describe Table

```
describe emp;
```

→ Describes the table schema

```
describe extended emp;
```

→ Provides a lot more info about the table

```
hive> describe emp;
OK
empid                   int
ename                   string
gender                  char(1)
age                     int
country                 string
salary                  bigint
deptid                  tinyint
Time taken: 0.397 seconds, Fetched: 7 row(s)
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/demodb.db/emp
Found 1 items
-rwxrwxrwx   1 cloudera supergroup        707 2018-12-04 04:52 /user/hive/warehouse/demodb.db/emp/emp.csv
```

Location of the data files of this table

```
Detailed Table Information      Table(tableName:emp, dbName:demodb, owner:cloudera, createTime:154392
7477, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:empid, type:int, com
ment:null), FieldSchema(name:ename, type:string, comment:null), FieldSchema(name:gender, type:char(1)
, comment:null), FieldSchema(name:age, type:int, comment:null), FieldSchema(name:country, type:string
, comment:null), FieldSchema(name:salary, type:bigint, comment:null), FieldSchema(name:deptid, type:t
inyint, comment:null)], location:hdfs://quickstart.cloudera:8020/user/hive/warehouse/demodb.db/emp, i
nputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIg
noreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializatio
nLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{line.delim=
, field.delim=,, serialization.format=,}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:Skew
edInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:
false), partitionKeys:[], parameters:{totalSize=707, numRows=0, rawDataSize=0, COLUMN_STATS_ACCURATE=
true, numFiles=1, transient_lastDdlTime=1543927974}, viewOriginalText:null, viewExpandedText:null, ta
bleType:MANAGED_TABLE)
Time taken: 0.238 seconds, Fetched: 10 row(s)
```

Querying data

```
SELECT * FROM emp WHERE age > 30;

SELECT eid, ename, age FROM emp LIMIT 10;
```

```
hive> SELECT * FROM emp LIMIT 5;
OK
1       Steve   M       60      USA     1000000 1
2       Peter   M       35      USA     1200000 1
3       Winsten M       50      UK      900000  2
4       Sony    F       45      UK      1000000 2
5       Pavan   M       25      India   1400000 2
Time taken: 1.487 seconds, Fetched: 5 row(s)
```

count

```
SELECT COUNT(*) FROM emp;
```

aggregation

```
SELECT COUNT(DISTINCT country) FROM emp;
```

grouping

```
SELECT country, COUNT(*), SUM(salary)
FROM emp GROUP BY country;
```

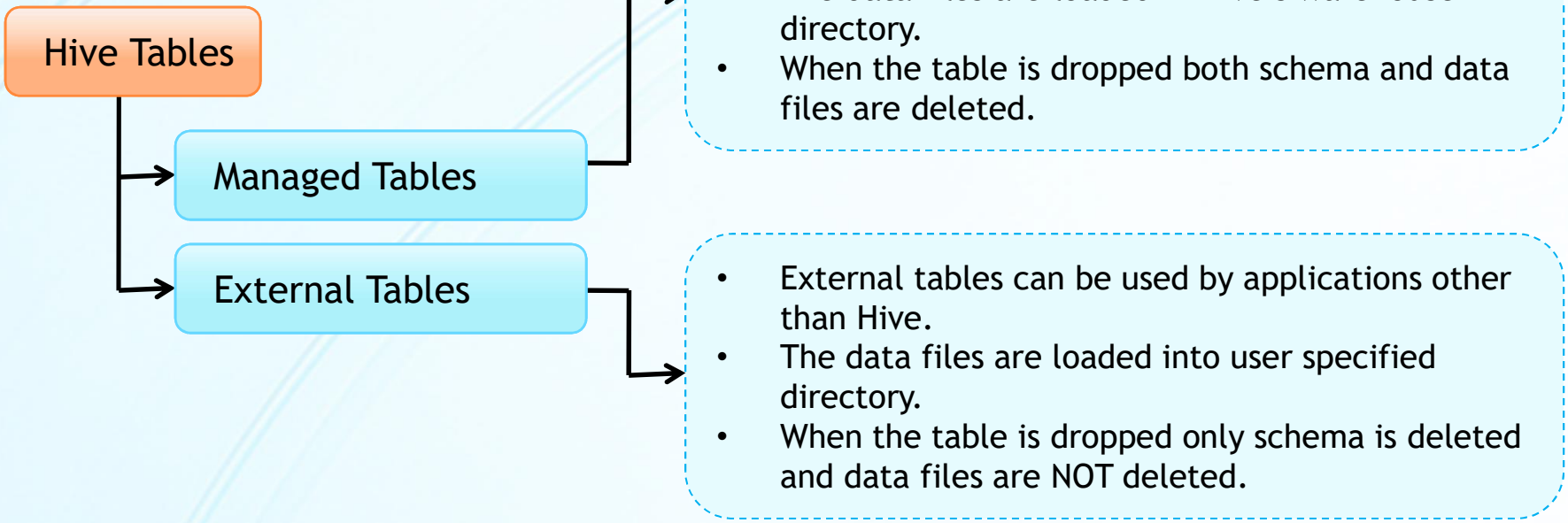
Drop the Table

```
DROP TABLE emp;
```

```
hive> DROP TABLE tmp;
OK
Time taken: 0.268 seconds
```

- When you drop a managed table :
  - The schema of the table is deleted from the metastore
  - The data files are deleted from the warehouse directory

External Tables



Creating External Table

```
CREATE EXTERNAL TABLE IF NOT EXISTS emp_ext
 (empid INT, ename STRING, gender CHAR(1), age INT,
  country STRING, salary BIGINT, deptid TINYINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ","
LINES TERMINATED BY "\n"
STORED AS TEXTFILE
LOCATION '/user/cloudera/hive';
```

```
hive> CREATE EXTERNAL TABLE emp_external
    > (empid INT, ename STRING, gender CHAR(1), age INT,
    >  country STRING, salary BIGINT, deptid TINYINT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ","
    > LINES TERMINATED BY "\n"
    > STORED AS TEXTFILE
    > LOCATION '/user/cloudera/hive/tables';
OK
Time taken: 0.13 seconds
```

Hive will copy all the data files that are loaded into the table into **'/user/cloudera/hive'** directory

```
Detailed Table Information      Table(tableName:emp_external, dbName:demodb, owner:cloudera,
, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:empid, type:int, comment:null), F
ent:null), FieldSchema(name:gender, type:char(1), comment:null), FieldSchema(name:age, type:
ntry, type:string, comment:null), FieldSchema(name:salary, type:bigint, comment:null), Field
t:null)], location:hdfs://quickstart.cloudera:8020/user/cloudera/hive/tables, inputFormat:or
outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, n
ull, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{line.d
, field.delim=,, serialization.format=,}), bucketCols:[], sortCols:[], parameters:{}, skewed
dColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:
UE, numRows=-1, rawDataSize=-1, COLUMN_STATS_ACCURATE=false, numFiles=0, transient_lastDdlTi
iewExpandedText:null, tableType:EXTERNAL_TABLE)
```

Schema on Read

- Hive does not validate the schema compatibility of data when loading the data into the table.
- The validation happens only when you query the data.
- If any data is not compatible with the schema, then Hive will simply show **NULL** values for that data.
  - For example, if the data file contains STRING values for age (which is defined as INT), it will simply show NULL when you query. It won't complain when you load data into the table.

Partitions

In Hive, Partitions provide a way of dividing a table into coarse-grained parts based on the value of a partition column, such as a date or country.

- Using partitions can make it faster to run queries on segments of the data.
- A table may be partitioned by more than one column such as by date and then by country.
- Partition keys determine how the data is stored.
- Each unique value of the partition keys defines the partition of the table

```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

Creating Partitioned Table

```
CREATE TABLE emp_part
(empid INT,ename STRING,gender CHAR(1),age INT,salary BIGINT,deptid TINYINT)
PARTITIONED BY (country STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
STORED AS TEXTFILE;
```

```
hive> CREATE TABLE emp_part
    > (empid INT, ename STRING, gender CHAR(1), age INT, salary BIGINT, deptid TINYINT)
    > PARTITIONED BY (country STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ","
    > STORED AS TEXTFILE;
OK
Time taken: 0.385 seconds
```

```
hive> describe emp_part;
OK
empid                   int
ename                   string
gender                  char(1)
age                     int
salary                  bigint
deptid                  tinyint
country                 string

# Partition Information
# col_name    data_type    comment

country       string
```

```
hive> show partitions emp_part;
OK
country=india
country=uk
country=usa
Time taken: 0.501 seconds, Fetched: 3 row(s)
```

Loading data into Partitioned Table

```
LOAD DATA LOCAL INPATH '/home/cloudera/datasets/emp_india.csv'
OVERWRITE INTO TABLE emp_part PARTITION (country = 'india');
```

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/datasets/emp_india.csv'
    > OVERWRITE INTO TABLE emp_part PARTITION (country = 'india');
Loading data to table demodb.emp_part partition (country=india)
Partition demodb.emp_part{country=india} stats: [numFiles=1, numRows=0, totalSize=233, rawDataSize=0]
OK
Time taken: 1.605 seconds
```

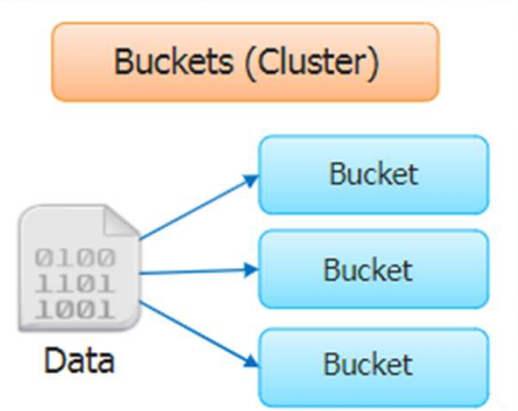
```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/demodb.db/emp_part
Found 3 items
drwxrwxrwx   - cloudera supergroup          0 2018-12-05 23:17 /user/hive/warehouse/demodb.db/emp_part/country=india
drwxrwxrwx   - cloudera supergroup          0 2018-12-05 23:28 /user/hive/warehouse/demodb.db/emp_part/country=uk
drwxrwxrwx   - cloudera supergroup          0 2018-12-05 23:28 /user/hive/warehouse/demodb.db/emp_part/country=usa
```

- All LOAD & INSERT statements must specify the value for the partitioned column.
- Each data file is loaded into a separate partition.
- Partitions are physically stored under different directories
- Query table partitions data using "`show partitions <table-name>`" command

Buckets

In Hive, Buckets provide an extra structure to the data to improve query efficiency.

- Breaks data into a set of buckets based on the hash function of the bucketed column.
- Buckets provide a mechanism to query and examine random samples of data.
- Join of two tables that are bucketed on the same columns—which include the join columns—can be efficiently implemented as a map-side join.



Bucketing is not enforced by default. To enforce bucketing, do the following:

```
hive> set hive.enforce.bucketing = true; OR
hive> set mapred.reduce.tasks = 25;
```

Working with Bucketed Table

Create a bucketed table

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid)
INTO 3 BUCKETS;
```

Working with Bucketed Table

Create a bucketed table

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid)
INTO 3 BUCKETS;
```

Create a bucketed table with sorting

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid) SORTED BY (empid ASC)
INTO 3 BUCKETS;
```

Working with Bucketed Table

Create a bucketed table

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid)
INTO 3 BUCKETS;
```

Create a bucketed table with sorting

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid) SORTED BY (empid ASC)
INTO 3 BUCKETS;
```

Enforce bucketing

```
set hive.enforce.bucketing = true;
```

Working with Bucketed Table

Create a bucketed table

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid)
INTO 3 BUCKETS;
```

Create a bucketed table with sorting

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid) SORTED BY (empid ASC)
INTO 3 BUCKETS;
```

Enforce bucketing

```
set hive.enforce.bucketing = true;
```

Insert data into bucketed table

```
INSERT OVERWRITE TABLE emp_bucket
SELECT empid, ename, gender, age FROM emp;
```

Working with Bucketed Table

Create a bucketed table

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid)
INTO 3 BUCKETS;
```

Create a bucketed table with sorting

```
CREATE TABLE emp_bucket
(empid INT, ename STRING, gender CHAR(1), age INT)
CLUSTERED BY (empid) SORTED BY (empid ASC)
INTO 3 BUCKETS;
```

Enforce bucketing

```
set hive.enforce.bucketing = true;
```

Insert data into bucketed table

```
INSERT OVERWRITE TABLE emp_bucket
SELECT empid, ename, gender, age FROM emp;
```

Separate file per bucket is created

```
hive> dfs -ls /user/hive/warehouse/demodb.db/emp_bucket;
Found 3 items
-rwxrwxrwx   1 cloudera supergroup        113 2018-12-06 03:12 /user/hive/warehouse/demodb.db/emp_bucket/000000_0
-rwxrwxrwx   1 cloudera supergroup        122 2018-12-06 03:12 /user/hive/warehouse/demodb.db/emp_bucket/000001_0
-rwxrwxrwx   1 cloudera supergroup        111 2018-12-06 03:12 /user/hive/warehouse/demodb.db/emp_bucket/000002_0
```

Sampling Data from Bucketed Tables

```
SELECT * FROM emp_bucket TABLESAMPLE (BUCKET 1 OUT OF 3 ON empid)
```

When we query a bucketed table as shown above, it will retrieve the data from the 1st of the 3 buckets i.e. approximately one-third of the records are sampled.

```
SELECT * FROM emp_bucket TABLESAMPLE (BUCKET 1 OUT OF 2 ON empid)
```

It's possible to sample a number of buckets by specifying a different proportion (which need not be an exact multiple of the number of buckets, as sampling is not intended to be a precise operation). Above example will fetch about half the buckets.

Block Sampling

- Block sampling refers to sampling x number of blocks based on some sampling parameter (available starting with Hive 0.8)
- We do it in HDFS block level so that the sampling granularity is block size. For example, if block size is 256MB, even if n% of input size is only 100MB, you get 256MB of data.

```
SELECT * FROM source TABLESAMPLE(0.1 PERCENT) s;
```

The above query samples 0.1% of the dataset (but a minimum of one block)

```
SELECT * FROM source TABLESAMPLE(100M) s;
```

The above query samples 100MB of the dataset (but a minimum of one block)

Other Sampling Options

- Hive also supports limiting input by row count basis.
- The row count given by user is applied to each split. So total row count can vary by number of input splits.

```
SELECT * FROM source TABLESAMPLE(10 ROWS);
```

The above query will take the first 10 rows from each input split

Managing Outputs

Inserting output into another table:

```
INSERT OVERWRITE TABLE emp2 SELECT * FROM emp
```

Inserting output into local file:

```
INSERT OVERWRITE LOCAL DIRECTORY 'emp2' SELECT * FROM emp
```

Inserting output into HDFS:

```
INSERT OVERWRITE DIRECTORY 'emp2' SELECT * FROM emp
```

Joins

Hive supports the classic SQL JOIN statement, but only *equi-joins* are supported.

- Joins
  - Inner Join
  - Outer Join
  - Semi Join using IN

```
SELECT a.* FROM a JOIN b ON (a.id = b.id)
```

```
SELECT a.* FROM a JOIN b ON
(a.id = b.id AND a.department = b.department)
```

Joins

Hive supports the classic SQL JOIN statement, but only *equi-joins* are supported.

- Joins
  - Inner Join
  - Outer Join
    - Left Outer Join
    - Right Outer Join
    - Full Join
  - Semi Join using IN

```
SELECT a.* FROM a JOIN b ON (a.id = b.id)
```

```
SELECT a.* FROM a JOIN b ON
(a.id = b.id AND a.department = b.department)
```

```
SELECT a.* FROM a LEFT OUTER JOIN b ON (a.id = b.id)
```

```
SELECT a.* FROM a RIGHT OUTER JOIN b ON(a.id = b.id)
```

```
SELECT a.* FROM a FULL OUTER JOIN b ON (a.id = b.id)
```

Joins

Hive supports the classic SQL JOIN statement, but only *equi-joins* are supported.

Joins

More than two tables can be joined in the same query

```
SELECT a.val, b.val, c.val FROM a JOIN b ON
(a.key = b.key1) JOIN c ON (c.key = b.key2)
```

Semi Join - IN/NOT IN/EXISTS/NOT EXISTS operators are supported

```
SELECT a.key, a.val FROM a
LEFT SEMI JOIN b ON (a.key = b.key)
```

```
SELECT a.key, a.value
FROM a WHERE a.key in (SELECT b.key FROM B);
```

Subquery

- Subquery in the FROM Clause:

```
SELECT col FROM (SELECT a AS col FROM t1) t2
```

- The subquery has to be given a name because every table in a FROM clause must have a name.
- Columns in the subquery select list must have unique names.
- The columns in the subquery select list are available in the outer query just like columns of a table.
- The subquery can also be a query expression with UNION. Hive supports arbitrary levels of subqueries.

Subquery

- Subquery in the FROM Clause:

```
SELECT col FROM (SELECT a AS col FROM t1) t2
```

- The subquery has to be given a name because every table in a FROM clause must have a name.
- Columns in the subquery select list must have unique names.
- The columns in the subquery select list are available in the outer query just like columns of a table.
- The subquery can also be a query expression with UNION. Hive supports arbitrary levels of subqueries.

- Subquery in the WHERE Clause:

```
SELECT * FROM emp WHERE emp.deptid IN (SELECT deptid FROM dept);
```

- These subqueries are only supported on the right-hand side of an expression.
- IN/NOT IN subqueries may only select a single column.
- EXISTS/NOT EXISTS must have one or more correlated predicates.
- References to the parent query are only supported in the WHERE clause of the subquery.

SORTING

ORDER BY

```
SELECT * FROM emp ORDER BY ename;
```

- ORDER BY performs a parallel total sort of the input.

SORTING

ORDER BY

```
SELECT * FROM emp ORDER BY ename;
```

- ORDER BY performs a parallel total sort of the input.

SORT BY

```
SELECT * FROM emp SORT BY ename;
```

- SORT BY produces a sorted file per reducer

SORTING

ORDER BY

```
SELECT * FROM emp ORDER BY ename;
```

- ORDER BY performs a parallel total sort of the input.

SORT BY

```
SELECT * FROM emp SORT BY ename;
```

- SORT BY produces a sorted file per reducer

DISTRIBUTE BY

```
SELECT * FROM emp DISTRIBUTE BY country
SORT BY country ASC, dept DESC;
```

- DISTRIBUTE BY controls which reducer a particular row goes to, so you can perform some subsequent aggregation. In the ex., all rows of a specific country end up in one reducer.

SORTING

ORDER BY

```
SELECT * FROM emp ORDER BY ename;
```

- ORDER BY performs a parallel total sort of the input.

SORT BY

```
SELECT * FROM emp SORT BY ename;
```

- SORT BY produces a sorted file per reducer

DISTRIBUTE BY

```
SELECT * FROM emp DISTRIBUTE BY country
SORT BY country ASC, dept DESC;
```

- DISTRIBUTE BY controls which reducer a particular row goes to, so you can perform some subsequent aggregation. In the ex., all rows of a specific country end up in one reducer.

CLUSTER BY

```
SELECT * FROM emp CLUSTUR BY country;
```

- If the columns for SORT BY and DISTRIBUTE BY are the same, you can use CLUSTER BY as a shorthand for specifying both.

GROUP BY

The **GROUP BY** clause is used to group all the records in a result set using a particular collection column. It is used to query a group of records.

```
SELECT deptid, COUNT(*) FROM emp
GROUP BY deptid;
```

Import Data : INSERT..SELECT

You can also populate a table with data from another Hive table using an **INSERT .. SELECT** statement.

INSERT INTO

```
INSERT INTO emp2 SELECT * FROM emp WHERE deptid=1;
INSERT INTO emp2 SELECT * FROM emp WHERE deptid=2;
```

Separate data files are created for each insert

```
[cloudera@quickstart]$ hadoop fs -ls /user/hive/warehouse/demodb.db/emp2/
Found 3 items
-rwxrwxrwx   1 cloudera supergroup          0 2018-12-07 00:11 /user/hive/warehouse/demodb.db/emp2/000000_0
-rwxrwxrwx   1 cloudera supergroup        707 2018-12-07 00:12 /user/hive/warehouse/demodb.db/emp2/000000_0_copy_1
-rwxrwxrwx   1 cloudera supergroup        224 2018-12-07 00:14 /user/hive/warehouse/demodb.db/emp2/000000_0_copy_2
```

Import Data: INSERT..SELECT

You can also populate a table with data from another Hive table using an **INSERT .. SELECT** statement.

INSERT INTO

```
INSERT INTO emp2 SELECT * FROM emp WHERE deptid=1;
INSERT INTO emp2 SELECT * FROM emp WHERE deptid=2;
```

Separate data files are created for each insert

```
[cloudera@quickstart]$ hadoop fs -ls /user/hive/warehouse/demodb.db/emp2/
Found 3 items
-rwxrwxrwx   1 cloudera supergroup          0 2018-12-07 00:11 /user/hive/warehouse/demodb.db/emp2/000000_0
-rwxrwxrwx   1 cloudera supergroup        707 2018-12-07 00:12 /user/hive/warehouse/demodb.db/emp2/000000_0_copy_1
-rwxrwxrwx   1 cloudera supergroup        224 2018-12-07 00:14 /user/hive/warehouse/demodb.db/emp2/000000_0_copy_2
```

INSERT OVERWRITE

```
INSERT OVERWRITE TABLE emp2 SELECT * FROM emp;
```

Overwrites existing data

```
[cloudera@quickstart]$ hadoop fs -ls /user/hive/warehouse/demodb.db/emp2/
Found 1 items
-rwxrwxrwx   1 cloudera supergroup        707 2018-12-07 00:22 /user/hive/warehouse/demodb.db/emp2/000000_0
```

Dynamic Partitioning

You can specify the partition dynamically by determining the partition value from the SELECT statement. This is known as a *dynamic partition insert*.

```
INSERT INTO emp_part
PARTITION (country)
SELECT empid, ename, gender, age, salary, deptid, country
FROM emp;
```

To enable dynamic partitioning you have to set the following:

```
set hive.exec.dynamic.partition.mode=nonstrict
```

Multi-table Insert

HQL supports multi-table insert, where in, in a single insert statement, you can insert data into multiple destination tables from a single source table.

```
FROM records2
    INSERT OVERWRITE TABLE stations_by_year
        SELECT year, COUNT(DISTINCT station)
        GROUP BY year
    INSERT OVERWRITE TABLE records_by_year
        SELECT year, COUNT(1)
        GROUP BY year
    INSERT OVERWRITE TABLE good_records_by_year
        SELECT year, COUNT(1)
        WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
        GROUP BY year;
```

In the above statement, there is a single source table (records2), but three tables to hold the results from three different queries over the source.

CREATE TABLE..AS SELECT

CTAS construct allows us to store the output of a Hive query in a new table.

```
CREATE TABLE emp_dept_1
AS
SELECT * FROM emp WHERE deptid = 1
```

- The new table's column definitions are derived from the columns retrieved by the SELECT clause.
- A CTAS operation is atomic, so if the SELECT query fails for some reason, the table is not created.

ALTER Tables

Rename a table

```
ALTER TABLE emp RENAME TO employee;
```

- Updates the table metadata.
- Moves the underlying table directory so that it reflects the new name.

ALTER Tables

Rename a table

```
ALTER TABLE emp RENAME TO employee;
```

- Updates the table metadata.
- Moves the underlying table directory so that it reflects the new name.

Add Column

```
ALTER TABLE emp ADD COLUMNS (email STRING)
```

- The new column *email* is added after the existing (non-partition) columns.
- The data-files are not updated, so queries will return `null` for all values of *email*.
- It is more common to create a new table that defines new columns and populates them using a SELECT statement (instead of following this approach).

ALTER Tables

Rename a table

```
ALTER TABLE emp RENAME TO employee;
```

- Updates the table metadata.
- Moves the underlying table directory so that it reflects the new name.

Add Column

```
ALTER TABLE emp ADD COLUMNS (email STRING)
```

- The new column *email* is added after the existing (non-partition) columns.
- The data-files are not updated, so queries will return `null` for all values of *email*.
- It is more common to create a new table that defines new columns and populates them using a SELECT statement (instead of following this approach).

Drop Column

```
ALTER TABLE emp DROP COLUMN email
```

ALTER Tables

Rename a table

```
ALTER TABLE emp RENAME TO employee;
```

- Updates the table metadata.
- Moves the underlying table directory so that it reflects the new name.

Add Column

```
ALTER TABLE emp ADD COLUMNS (email STRING)
```

- The new column *email* is added after the existing (non-partition) columns.
- The data-files are not updated, so queries will return `null` for all values of *email*.
- It is more common to create a new table that defines new columns and populates them using a SELECT statement (instead of following this approach).

Drop Column

```
ALTER TABLE emp DROP COLUMN email
```

Change Col. Name

```
ALTER TABLE emp CHANGE email email2 STRING
```

DROP & TRUNCATE

Drop a table

```
DROP TABLE emp;
```

Truncate a table

```
TRUNCATE TABLE emp;
```

Create empty table

```
CREATE TABLE empnew LIKE emp;
```

Views

A view is a sort of ‘virtual table’ that is defined by a SELECT statement

Create

```
CREATE VIEW IF NOT EXISTS vw_emp_dept_1
AS SELECT * FROM emp WHERE deptid = 1;
```

Query

```
SELECT * FROM vw_emp_dept_1;
```

Drop

```
DROP VIEW vw_emp_dept_1;
```

Indexes

- An Index is nothing but a pointer on a particular column of a table.
- Hive (prior to versions 3.0) supports compact and bitmap indexes.

Create

```
CREATE INDEX emp_index ON TABLE emp (deptid) AS 'COMPACT'
WITH DEFERRED REBUILD;
```

Drop

```
DROP INDEX indx_emp_deptid ON emp;
```

Indexing Is Removed since 3.0

There are alternate options which might work similarily to indexing:

- Materialized views with automatic rewriting can result in very similar results. Hive 2.3.0 adds support for materialzed views.
- Using columnar file formats (Parquet, ORC) – they can do selective scanning; they may even skip entire files/blocks.

⚠ Indexing has been **removed** in version 3.0 (HIVE-18448).

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

THANK YOU