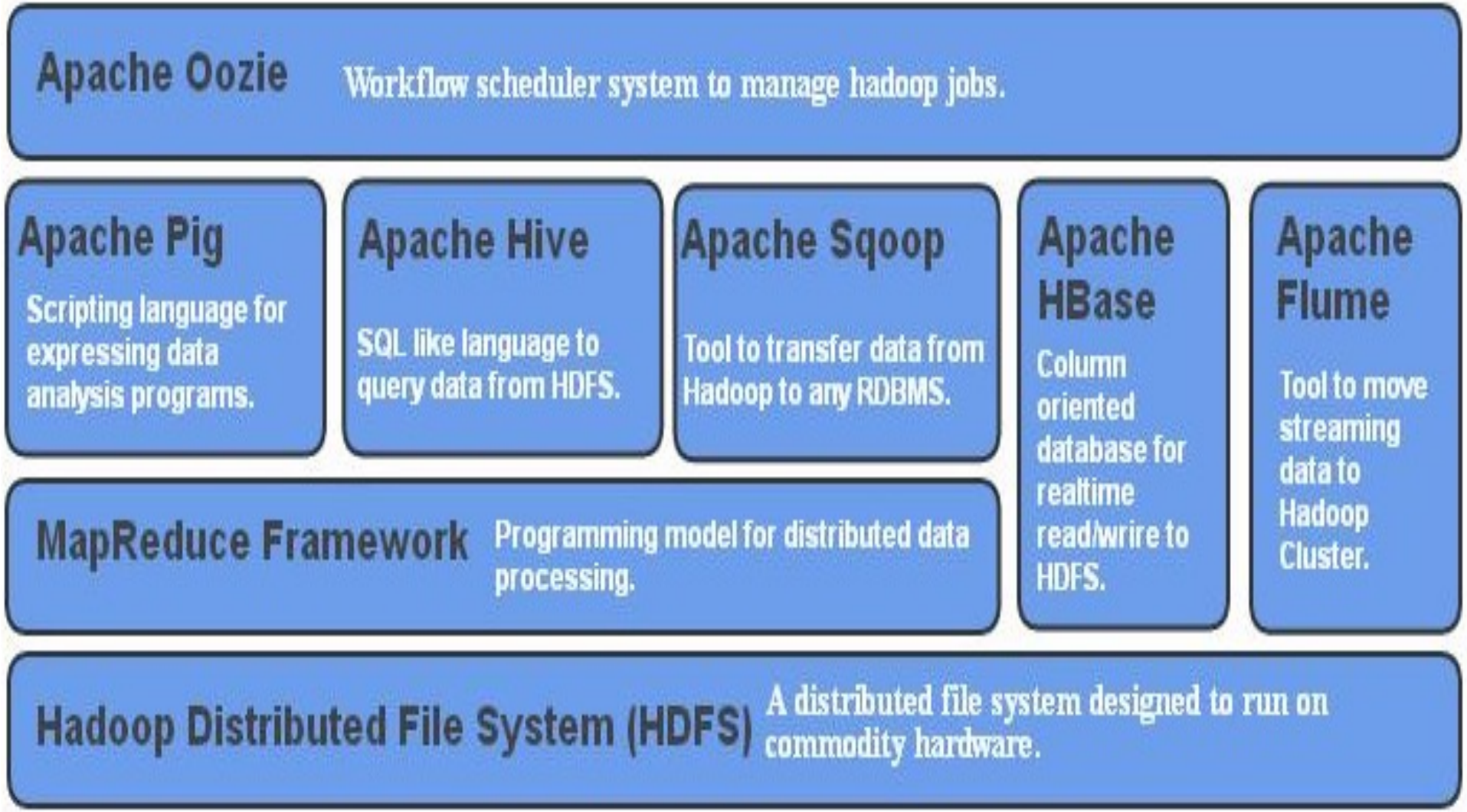
Hive

Contents

Hadoop Ecosystem



What is Hive?

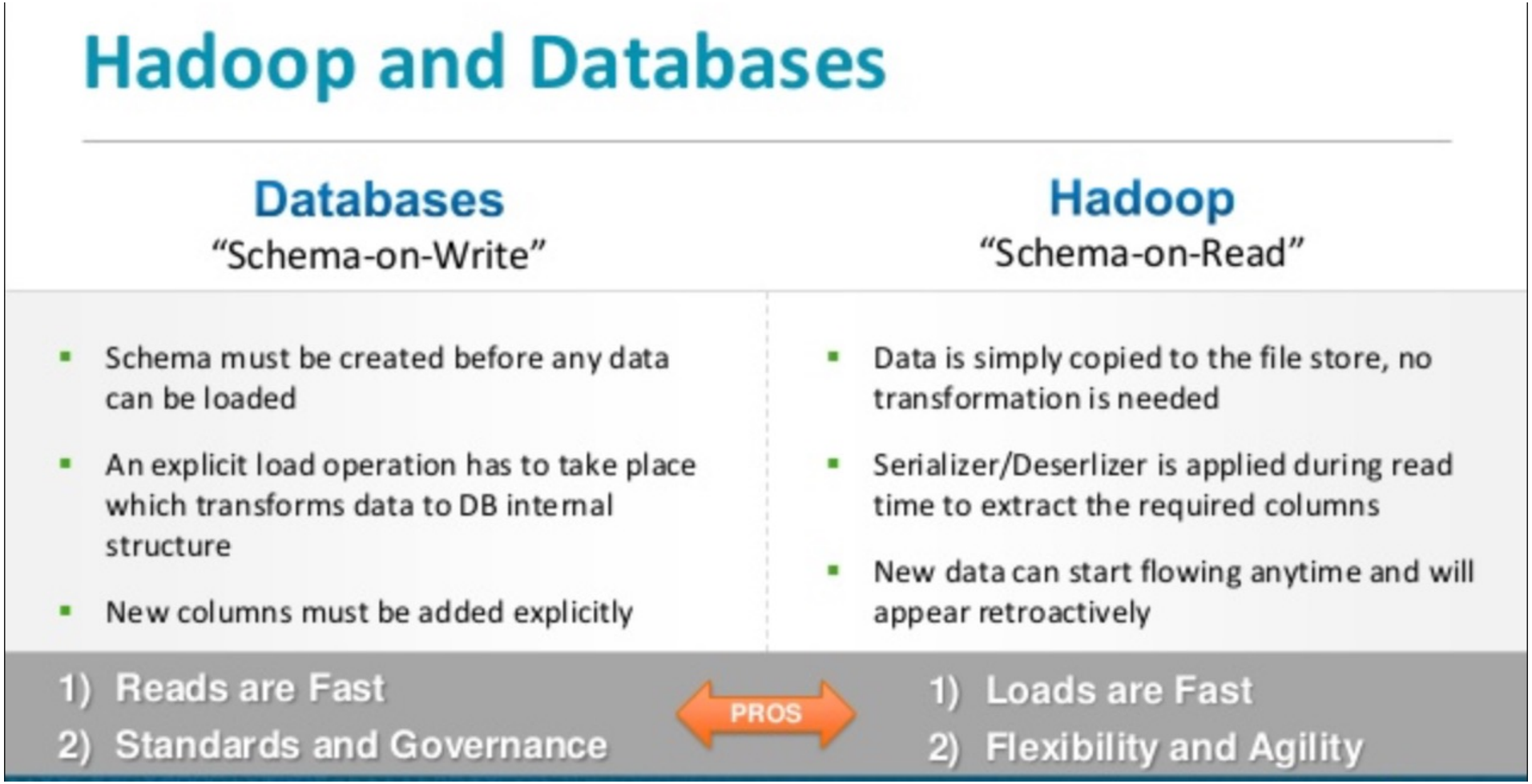
The **Apache Hive** data warehouse software facilitates querying and managing large datasets residing in distributed storage. Built on top of Apache Hadoop , it provides

- Tools to enable easy data extract/transform/load (ETL) .
- A mechanism to impose structure on a variety of data formats.
- Access to files stored either directly in Apache HDFS or in other data storage systems such as Apache Hbase.
- Query execution via MapReduce.
- Provides SQL like query language – HIVE QL. Allows to plug in custom mappers /reducers in queries and also allows UDFs.
- Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data.

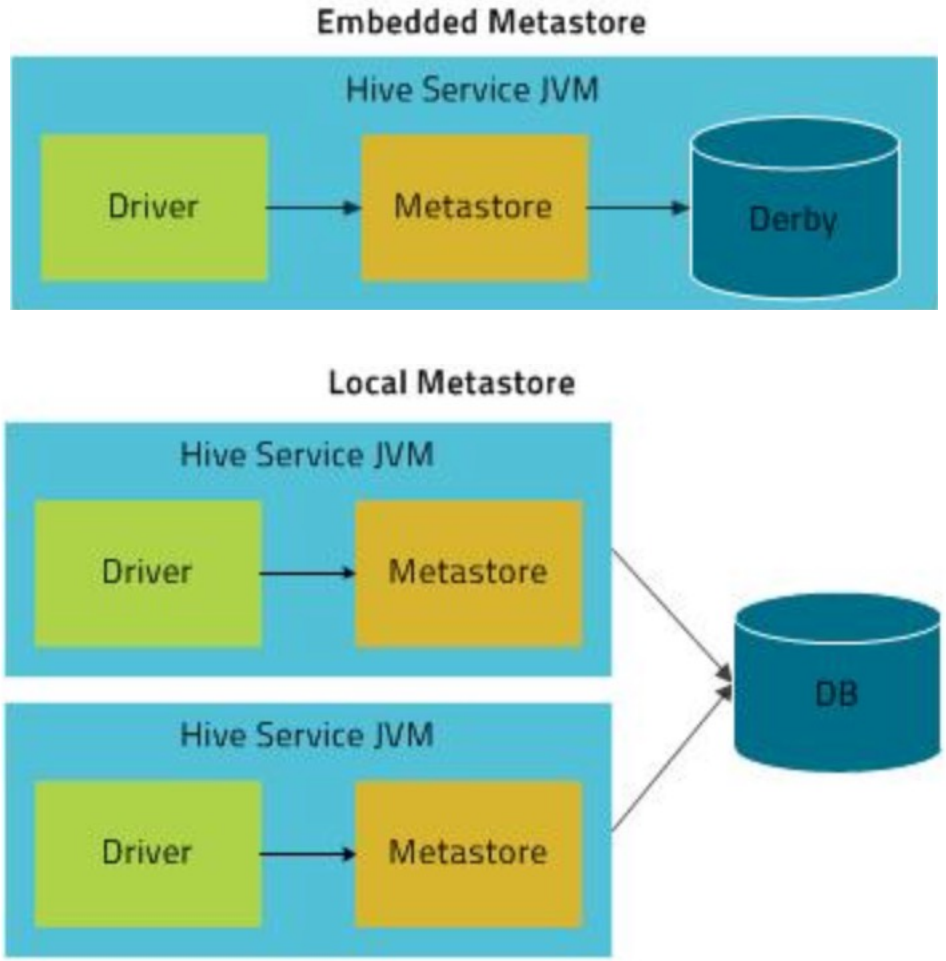
What Hive is not?

- Latency for Hive queries is generally very high (minutes) even when data sets involved are very small (say a few hundred megabytes). As a result it cannot be compared with systems such as RDBMS.
- Hive is not designed for online transaction processing and does not offer real-time queries and row level updates. It is best used for batch jobs over large sets of immutable data (like web logs). Table update is achieved by transforming data into new table.
- Unlike RDBMS table, More than one schema can be applied to same data.
- Hive doesn’t define clear semantics for concurrent access to tables, which means applications need to build their own application-level concurrency or locking mechanism. The Hive team is actively working on improvements in all these areas.
- Hive schema is not a ‘schema on write’. Does not verify the data when it is loaded. It is a ‘schema on read’ which verifies data on SQL query.

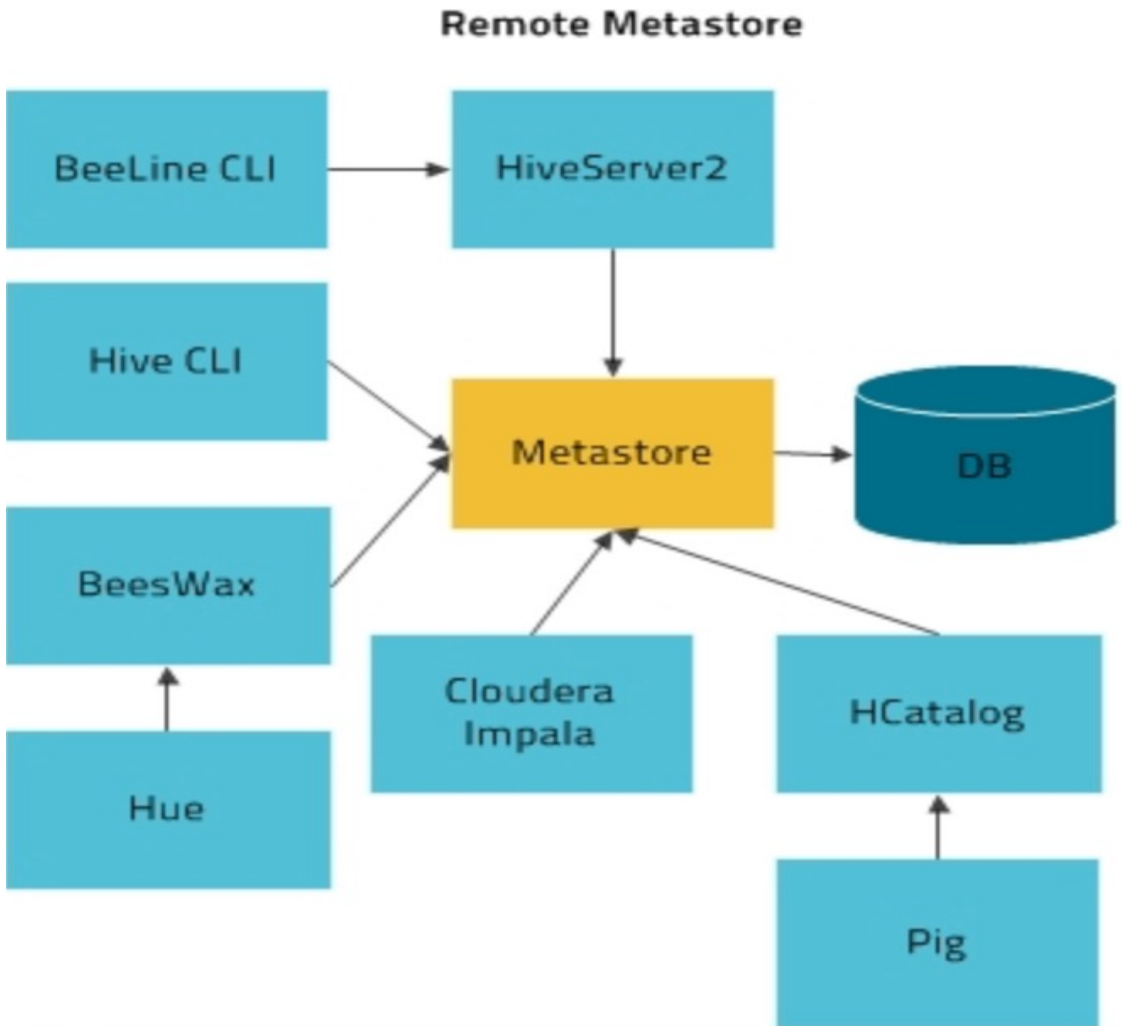
Schema on Read Vs Schema on Write



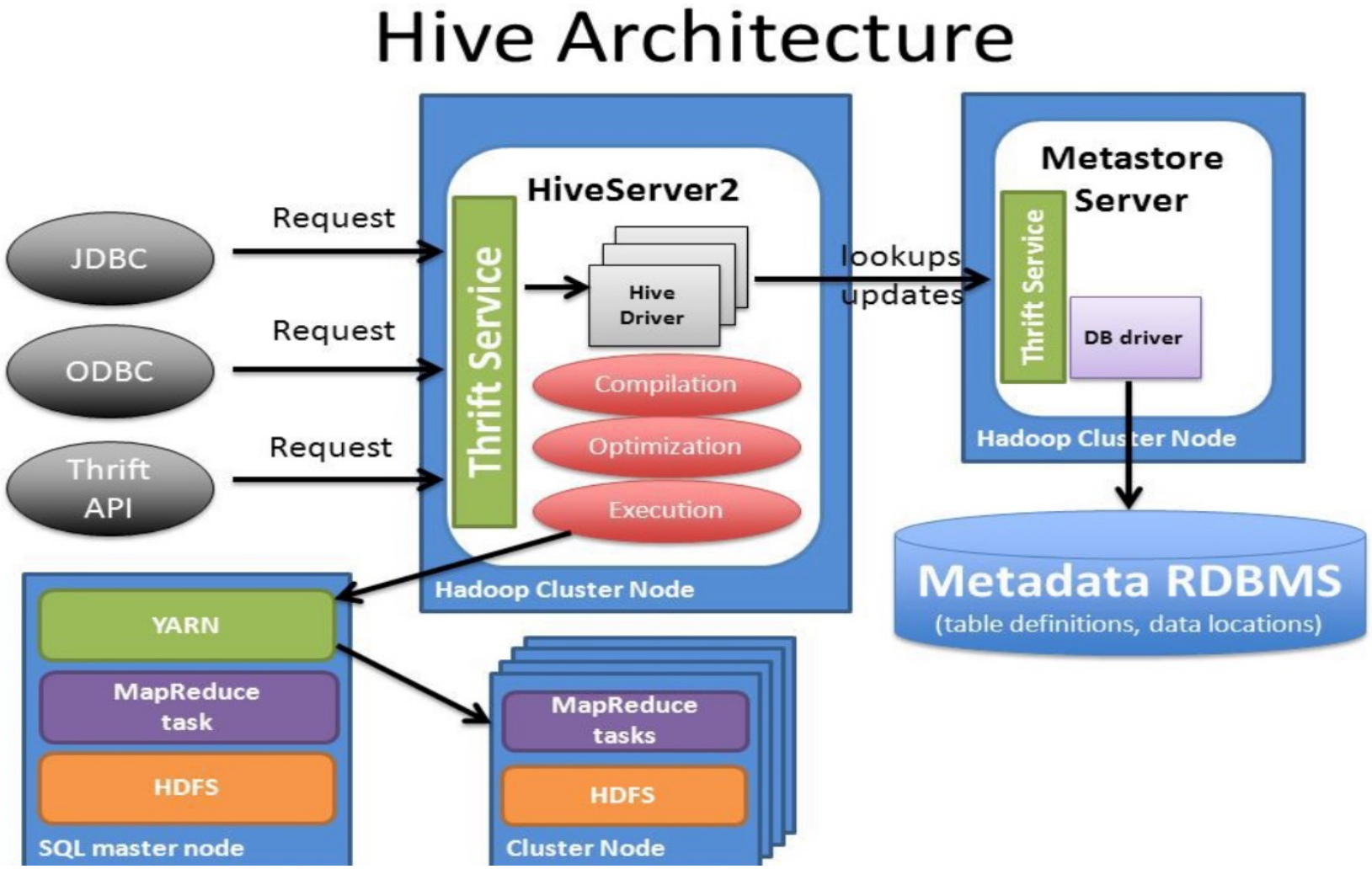
Hive Modes


Embedded Metastore
Hive Service JVM
Driver
Metastore
Derby
Local Metastore
Hive Service JVM
Driver
Metastore
Hive Service JVM
Driver
Metastore
DB

Hive Modes


Remote Metastore
BeeLine CLI
HiveServer2
Hive CLI
Metastore
DB
BeesWax
Cloudera Impala
HCatalog
Hue
Pig

Hive Architecture



Hive Architecture

- **Metastore**: Stores system catalog
- **Driver**: manages life cycle of HiveQL query as it moves thru' HIVE; also manages session handle and session statistics
- **Query compiler**: Compiles HiveQL into a directed acyclic graph of map/reduce tasks
- **Execution engines**: The component executes the tasks in proper dependency order; interacts with Hadoop
- **HiveServer2**: provides Thrift interface and JDBC/ODBC for integrating other applications.
- **Client components**: Beeline CLI/shell, web interface(HUE), jdbc/odbc interface

Hive Data Organized

- Databases – Namespace that separates tables.
- Tables - Homogeneous units of data with same schema.
- Partitions - A way of dividing a table into coarse-grained parts based on the value of a partition column, such as date.
- Buckets - Tables or partitions may further be subdivided into buckets, to give extra structure to the data that may be used for more efficient queries.

Hive QL

- Hive query language provides the basic SQL like operations like
- Ability to filter rows from a table using a where clause.
- Ability to select certain columns from the table using a select clause.
- Ability to do equi-joins between two tables.
- Ability to evaluate aggregations on multiple "group by" columns for the data stored in a table.
- Ability to store the results of a query into another table.
- Ability to store the results of a query in a hadoop dfs directory.
- Ability to manage tables and partitions (create, drop and alter).

SQL Vs Hive QL

Feature	SQL	HiveQL
Updates	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (populates whole table or partition)
Transactions	Supported	Not supported
Indexes	Supported	Not supported
Latency	Sub-second	Minutes
Data types	Integral, floating point, fixed point, text and binary strings, temporal	Integral, floating point, boolean, string, array, map, struct
Functions	Hundreds of built-in functions	Dozens of built-in functions
Multitable inserts	Not supported	Supported
Create table as select	Not valid SQL-92, but found in some databases	Supported
Select	SQL-92	Single table or view in the FROM clause. SORT BY for partial ordering. LIMIT to limit number of rows returned.

Feature	SQL	HiveQL
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins. SQL-92 syntax, with hinting.
Subqueries	In any clause. Correlated or noncorrelated.	Only in the FROM clause. Correlated subqueries not supported
Views	Updatable. Materialized or nonmaterialized.	Read-only. Materialized views not supported
Extension points	User-defined functions. Stored procedures.	User-defined functions. MapReduce scripts.

Data Types

Category	Type	Description	Literal examples
Primitive	TINYINT	1-byte (8-bit) signed integer, from -128 to 127	1
	SMALLINT	2-byte (16-bit) signed integer, from -32,768 to 32,767	1
	INT	4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647	1
	BIGINT	8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	1
	FLOAT	4-byte (32-bit) single-precision floating-point number	1.0
	DOUBLE	8-byte (64-bit) double-precision floating-point number	1.0
	BOOLEAN	true/false value	TRUE
	STRING	Character string	'a', "a"
	BINARY	Byte array	Not supported
	TIMESTAMP	Timestamp with nanosecond precision	1325502245000, '2012-01-02 03:04:05.123456789'

Category	Type	Description	Literal examples
Complex	ARRAY	An ordered collection of fields. The fields must all be of the same type.	array(1, 2)[a]
	MAP	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.	map('a', 1, 'b', 2)
	STRUCT	A collection of named fields. The fields may be of different types.	struct('a', 1, 1.0)[b]

Data Storage

- Warehouse directory in HDFS
  - e.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - e.g., /user/hive/warehouse/movies
  - Partitions, buckets form subdirectories of tables.
- Actual data stored in flat files(By Default)
  - Control char-delimited text.

Hive Shell – Hive CLI/beeline

Hive CLI

$ hive

hive>

Note:**set hive.cli.print.current.db=true** to display the current database

Or set it in hive-site.xml

```
<property>
        <name>hive.cli.print.current.db</name>
        <value>true</value>
 </property>
```

Hive Shell – Hive CLI/beeline

Beeline

$beeline

beeline>!connect jdbc:hive2://localhost:10000/default bigdata bigdata

Connected to: Apache Hive (version 1.1.0-cdh5.7.0)

Driver: Hive JDBC (version 1.1.0-cdh5.7.0)

Transaction isolation: TRANSACTION_REPEATABLE_READ

0: jdbc:hive2://localhost:10000/default>select * from movies;

0: jdbc:hive2://localhost:10000/default>show tables;

0: jdbc:hive2://localhost:10000/default>describe movies;

0: jdbc:hive2://localhost:10000/default>SET <key>=<value>;

Hive Command Line Options

Example of running Query from command line

```
$beeline –u jdbc:hive2://localhost:10000/default -e 'select movieid,title from movies'
```

Example of dumping data out from a query into a file using silent mode

```
$beeline –u jdbc:hive2://localhost:10000/default –silent true -e 'select movieid,title from movies' > a.txt
```

Example of running a script non-interactively

```
$beeline –u jdbc:hive2://localhost:10000/default -f /home/my/hive-script.sql
```

Example of running an initialization script before entering interactive mode

```
$beeline –u jdbc:hive2://localhost:10000/default -i /home/my/hive-init.sql
```

Hive Command Line Options

Using a different key-value from beeline

```
$beeline -hiveconf <key>=<value>
Ex:
$beeline -hiveconf yarn.resourcemanager.address=localhost:8032
```

Using SET command

```
$beeline SET
```

Precedence of properties

- The Hive SET command
- The command-line –hiveconf option
- hive-site.xml and the Hadoop site files(core-site.xml,hdfs-site.xml,mapred-site.xml, and yarn-site.xml)
- The Hive defaults and the Hadoop default files(core-default.xml,hdfs-default.xml, mapred-default.xml, and yarn-default.xml)

Databases

- The Hive concept of a database is essentially just a *catalog* or *namespace* of tables.
- They are very useful for larger clusters with multiple teams and users, as a way of avoiding table name collisions.
- It's also common to use databases to organize production tables into logical groups.
- If you don't specify a database, the "**default"** database is used.

Examples:

hive> **CREATE DATABASE** moviesdb;

hive> **CREATE DATABASE** IF **NOT EXISTS** moviesdb;

hive> **SHOW Databases**;

hive> **SHOW Databases like "m*"**;

- Hive creates directories for each database with .db extension under ***/user/hive/warehouse/moviesdb.db***

Note: To show the database in hive shell, execute the following command in hive shell

set hive.cli.print.current.db=true

Exercise – Logging in to Hive CL/beeline

Managed Tables

- If data is available as a flat file at source, hive tables can be created manually

```
CREATE TABLE users(userid int , gender string, age int, occupation int, zipcode int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ':';
To skip header : TBLPROPERTIES ("skip.header.line.count"="1");
```

- Data can be loaded in to a table using the LOAD statement in hive

```
LOAD DATA LOCAL INPATH 'users.dat' OVERWRITE INTO TABLE users;

Select * from users;
Note: By default no headers will be shown.
Set set hive.cli.print.header=true; to show the headers
```

External Tables

- Use external tables when you don't own the data and you do not want hive to manage the data.
- Developer can control the creation and deletion of data by specifying a location at the time of creation of the table.
- Use “External” keyword in the table definition to create an EXTERNAL Hive table.

```
CREATE EXTERNAL TABLE userratings (userid int,movieid int,rating int,createtimestamp int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ':' LOCATION '/user/bigdata/userratings’
```

- With the EXTERNAL keyword, Hive knows that it is not managing the data, so it doesn’t move it to it’s warehouse directory.
- Hive doesn’t even check whether the external location exists at the time it is defined.
- You can create the data lazily after creating the table.

External Tables

Loading data into an EXTERNAL table

```
LOAD DATA LOCAL INPATH ‘ratings.dat ' INTO TABLE userratings;
```

The difference between the two types of table is seen in the LOAD and DROP semantics.

	Managed Table	External Table
Create / Load	Data will be Warehouse directory	Data will be at external location. Does not even check whether external location exists.
Drop	Metadata and Data is deleted	Metadata is deleted and data is untouched.

Managed or External Tables

- As a rule of thumb, if you are doing all your processing with Hive, then use managed tables.
- If you wish to use Hive and other tools on the same dataset, then use external tables.
- A common pattern is to use an external table to access an initial dataset stored in HDFS (created by another process), then use a Hive transform to move the data into a managed Hive table.
- Another reason for using external tables is when you wish to associate multiple schemas with the same dataset.

Lab Exercise – Managed and External Tables

Assignment 1 – Facebook Data analysis

Create an **external** table for the face book data provided as a CSV file and answer the following queries

Task 1: Create a hive table by looking at the fields and determine the schema

Task 2: Import the data in to hive/hdfs

Task 3: Run some sample queries against the data

Query 1: Find out the total no of users in the dataset
Query 2: Find out the no of users above the age of 25
Query 3: Do male face book users tend to have more friends or female users
Query 4: How many likes do young people receive on face book as opposed to old people
Query 5: Find out the count of face book users for each birthday month
Query 6: Do young members use mobile phones or computers for face book browsing
Query 7: Do adult members use mobile phones or computers for face book browsing

Joins

One of the nice things about using Hive, rather than raw MapReduce, Joins are made very simple.

INNER JOIN:

hive> **SELECT * FROM sales;**
Joe 2
Hank 4
Ali 0
Eve 3
Hank 2

hive> **SELECT * FROM things;**
2 Tie
4 Coat
3 Hat
1 Scarf

We can perform an inner join on the two tables as follows:

hive> **SELECT sales.*, things.***
> **FROM sales JOIN things ON (sales.id = things.id);**
Joe 2 2 Tie
Hank 2 2 Tie
Eve 3 3 Hat
Hank 4 4 Coat

Outer Join

Left Outer Join:

```
hive> SELECT sales.*, things.*
> FROM sales LEFT OUTER JOIN things ON
(sales.id = things.id);
Ali    0    NULL    NULL
Joe    2    2    Tie
Hank    2    2    Tie
Eve    3    3    Hat
Hank    4    4    Coat
```

```
hive> SELECT sales.*, things.*
> FROM sales FULL OUTER JOIN things ON (sales.id =
things.id);
Ali    0    NULL    NULL
NULL    NULL    1    Scarf
Joe    2    2    Tie
Hank    2    2    Tie
Eve    3    3    Hat
Hank    4    4    Coat
```

Right Outer Join:

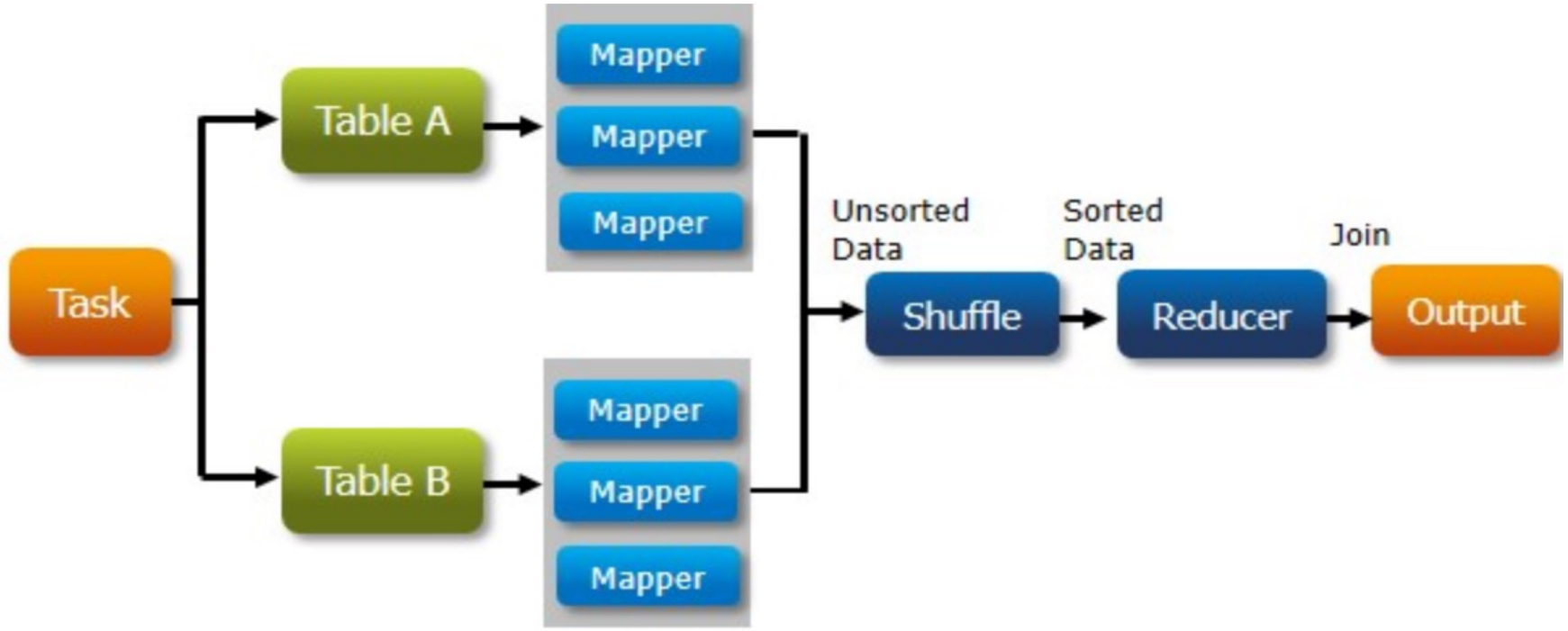
```
hive> SELECT sales.*, things.*
   > FROM sales RIGHT OUTER JOIN things ON
(sales.id = things.id);
NULL    NULL    1    Scarf
Joe    2    2    Tie
Hank    2    2    Tie
Eve    3    3    Hat
Hank    4    4    Coat
```

```
hive> SELECT *
   > FROM things LEFT SEMI JOIN sales ON
(sales.id = things.id);   same as

SELECT *
FROM things
WHERE things.id IN (SELECT id from sales);
```

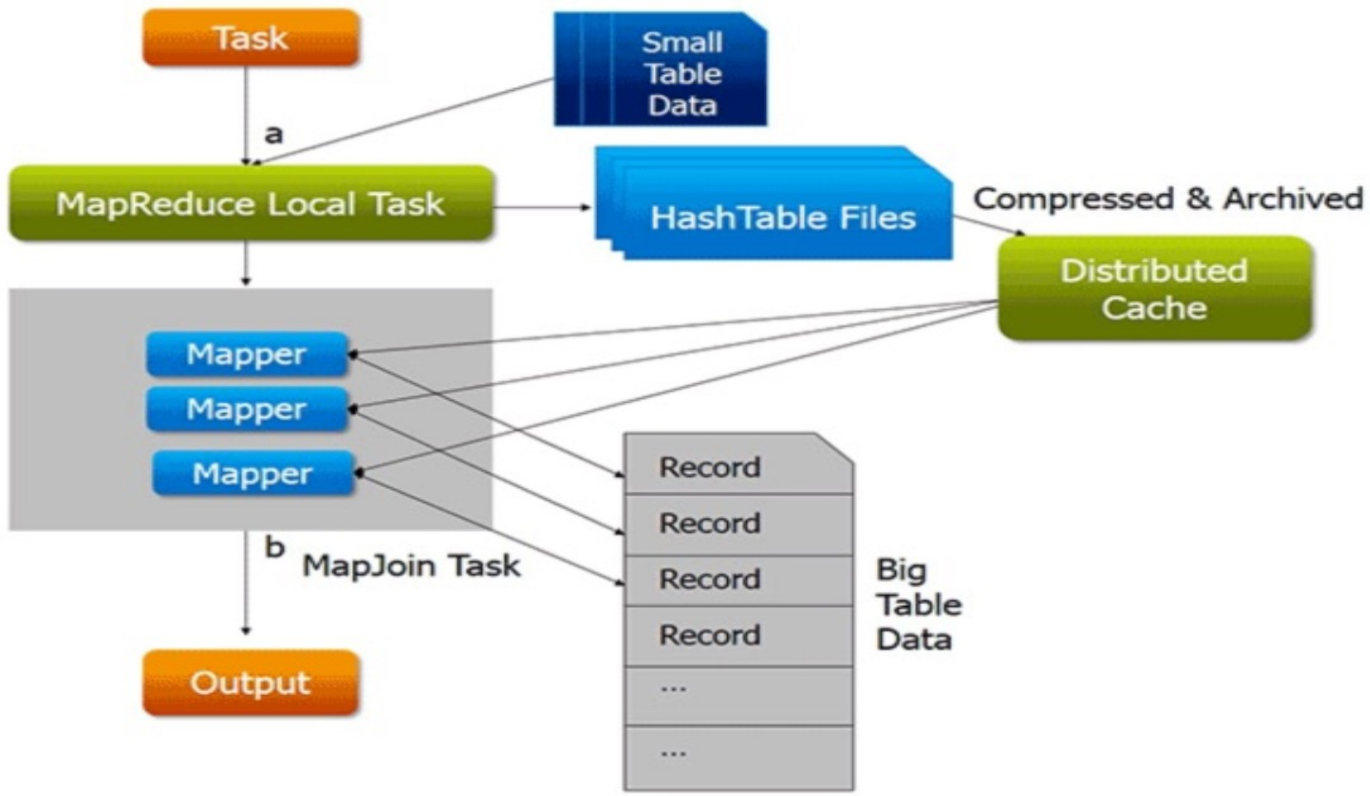
Map-side Vs Reduce-side join

A Reduce-side join is converted to a map-reduce program and the actual join happens at the reduce side.



Map-side Vs Reduce-side join

```
SELECT /*+ MAPJOIN(movies) */ userratings.userid,movies.movieid,movies.title,userratings.rating
FROM userratings JOIN movies ON (userratings.movieid = movies.movieid);
```



Lab Exercise – Joins on movies dataset

Loading Data between tables

Create Table As Select (CTAS)
Command: Create table target_table as select * from source_table
(Note: This command creates a new Table (target_table))

Ex: create table users_copy as select * from users;

Insert Overwrite: Populate data from one table to another **existing** table

INSERT OVERWRITE TABLE <target_table>
SELECT col1, col2
  FROM <source_table>;

OVERWRITE Keyword – replaces the contents in HDFS directory

Insert overwrite table users_copy select * from users;

Loading Data into Local/Hdfs locations

In certain situations you would want to write the output into a local file so that you could load it into an excel spreadsheet. This can be accomplished with the following command:

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'
SELECT pv_gender_sum.*
FROM pv_gender_sum;
```

Lab Exercise– Insert Overwrite / CTAS

Assignment 2 – Drivers data Analysis

2 Tables are provided

Drivers contains Driver info

Timesheets contains hours and miles logged by each driver (id)

Problem statement: **How many total hours and miles logged by each driver (print the name of each driver)?**

Assignment 3 – Orders data Analysis

Problem Statement: What is the daily product revenue for
CLOSED or COMPLETE orders?

Storage Formats

Row Format: How the row and fields are stored and controlled by *SerDe*
Deserializer: A Query - deserialize a row of data from the bytes in the file to objects
Serializer: A INSERT - serialize Hive's internal representation of a row of data into the bytes

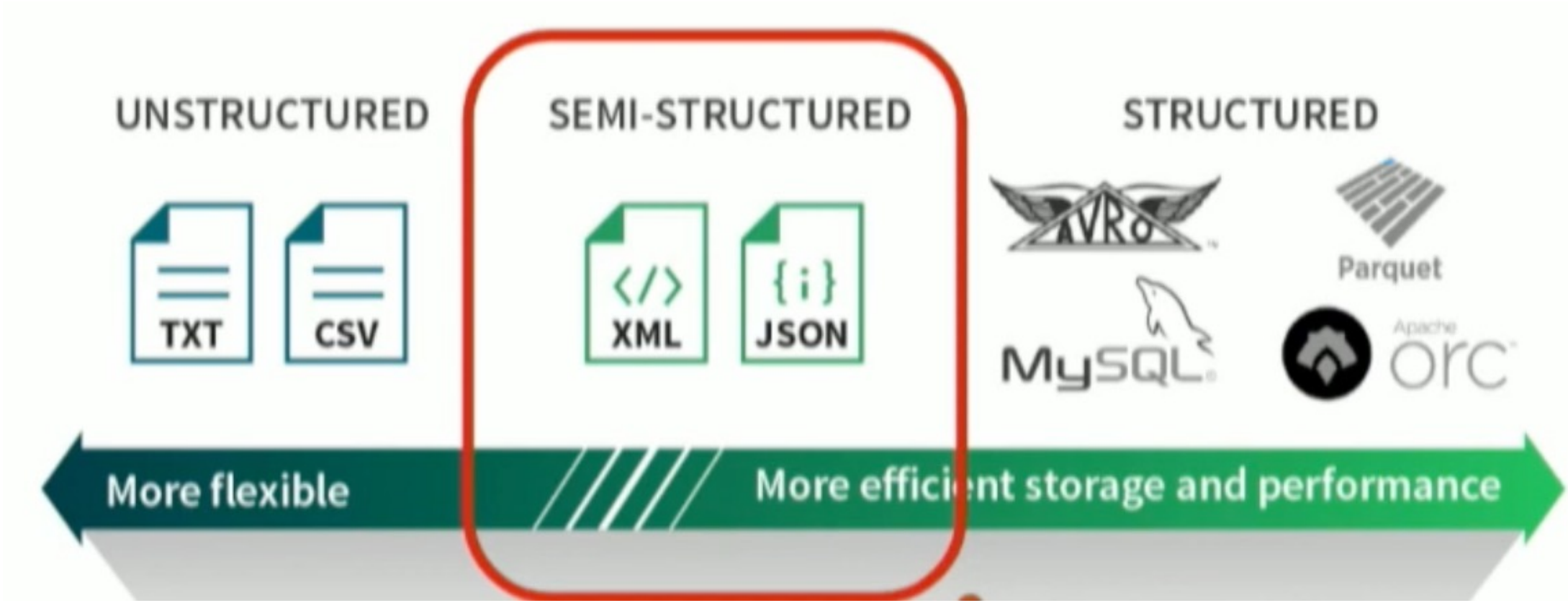
File Format: The file format dictates the container format for fields in a row. The simplest format is a plain text file, but there are row-oriented and column-oriented binary formats available,too.

Storage Formats
- RCFile
- ORC
- Parquet
- Avro

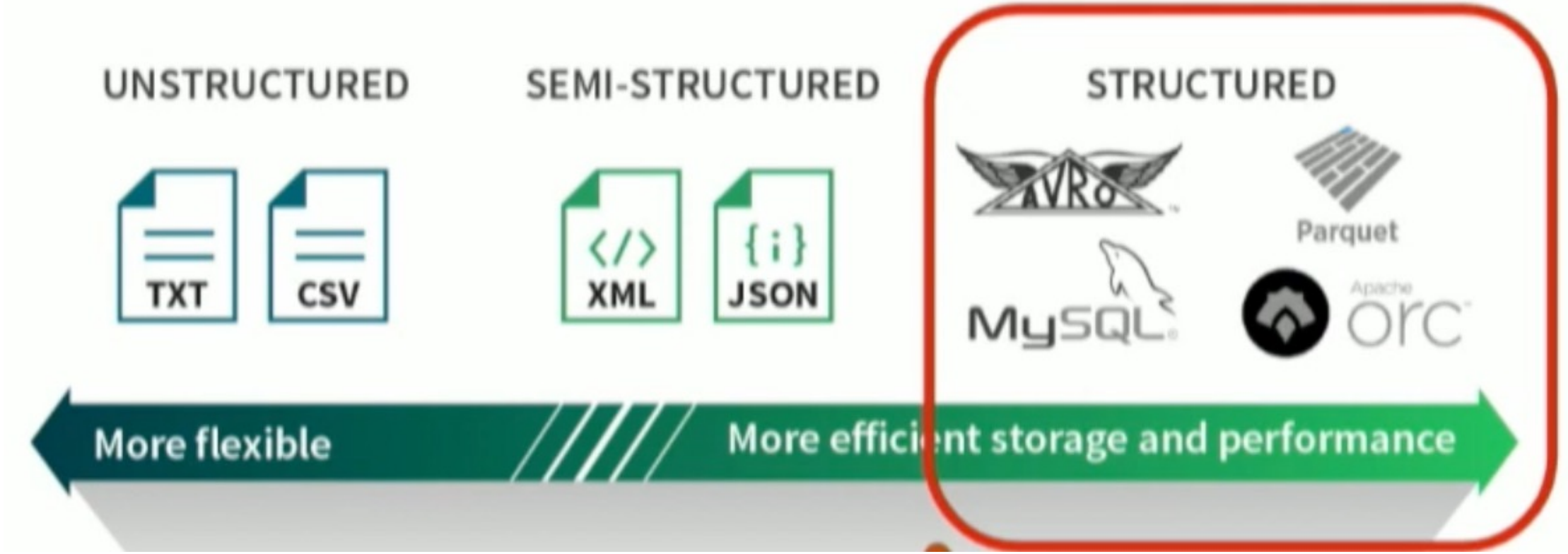
Compression
- Snappy
- Zlib
- Native gzip compression

Storage Formats

Storage Formats



Storage Formats



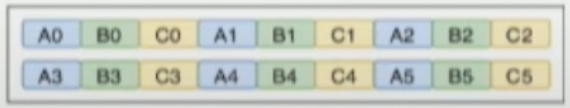
Storage Formats

Physical storage layout models

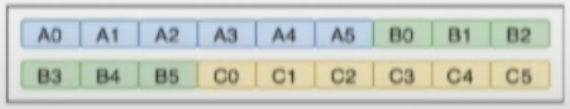
Logical

	Col A	Col B	Col C
Row 0	A0	B0	C0
Row 1	A1	B1	C1
Row 2	A2	B2	C2
Row 3	A3	B3	C3
Row 4	A4	B4	C4
Row 5	A5	B5	C5

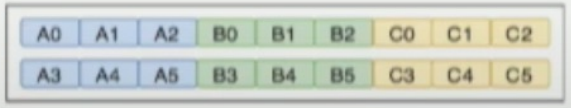
Physical



Row-wise



Columnar



Hybrid

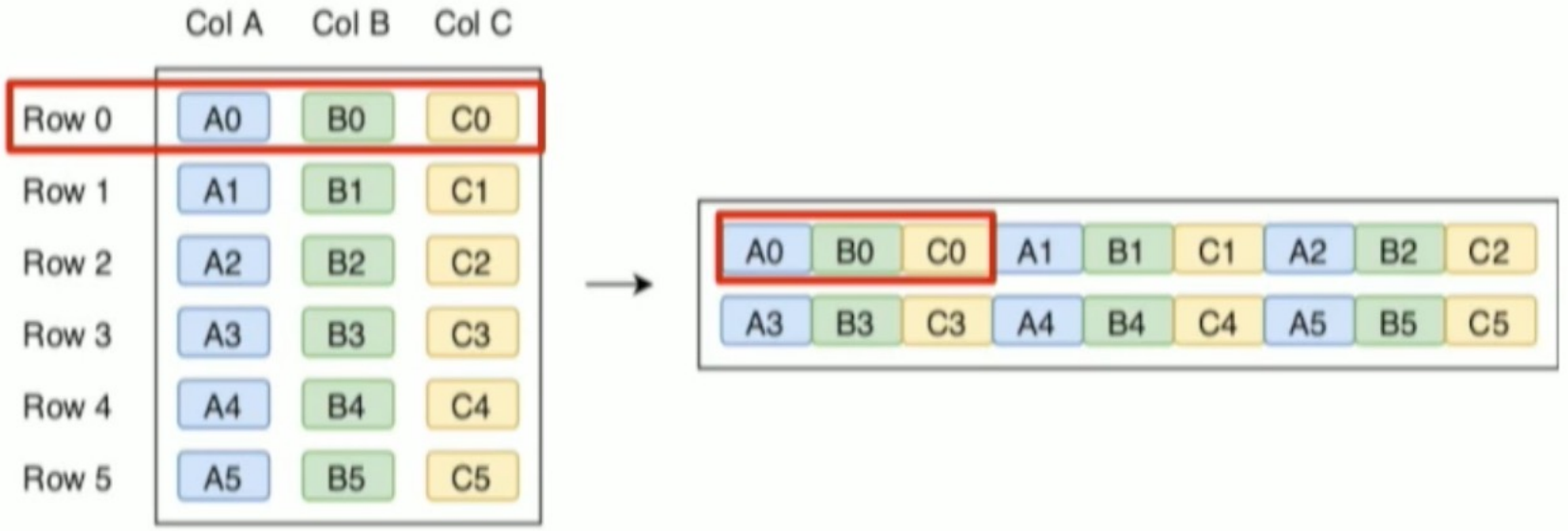
Storage Formats

- **Different Work loads**
  - **OLTP** :
    - Online Transaction Processing
    - Lots of small operation involving whole rows
  - **OLAP**:
    - Online Analytical Processing
    - A few large operations involving subset of all columns

- I/O is expensive(Memory, Disk , Network

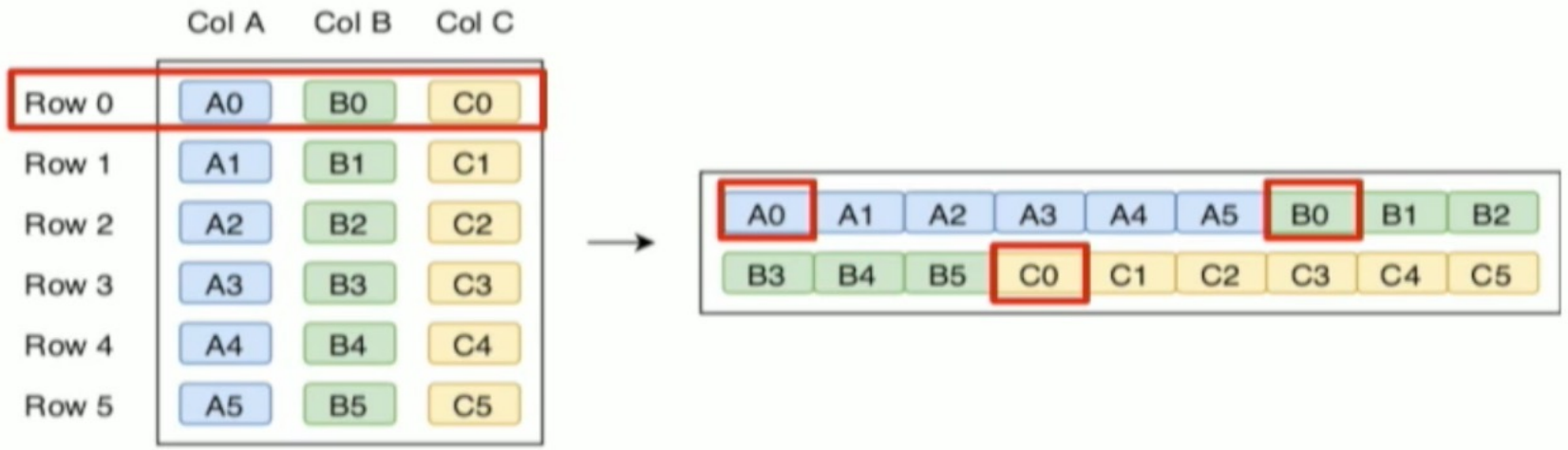
Storage Formats

Row-wise



- Horizontal partitioning
- OLTP ✓, OLAP ✗

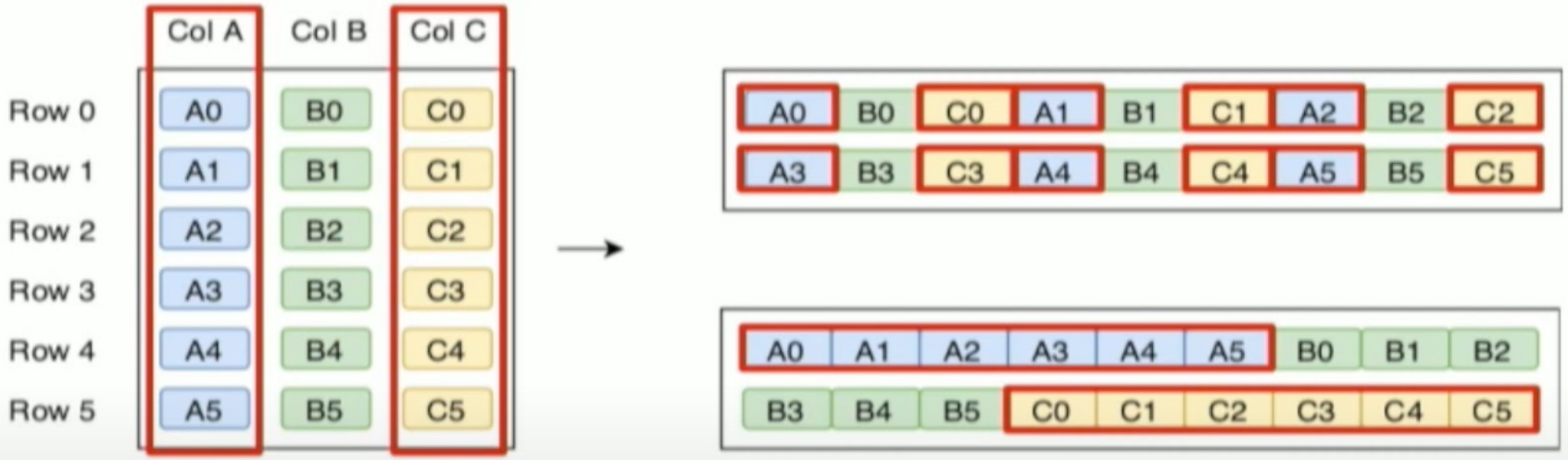
Columnar



- Vertical partitioning
- OLTP ✖, OLAP ✔
  - Free projection pushdown
  - Compression opportunities

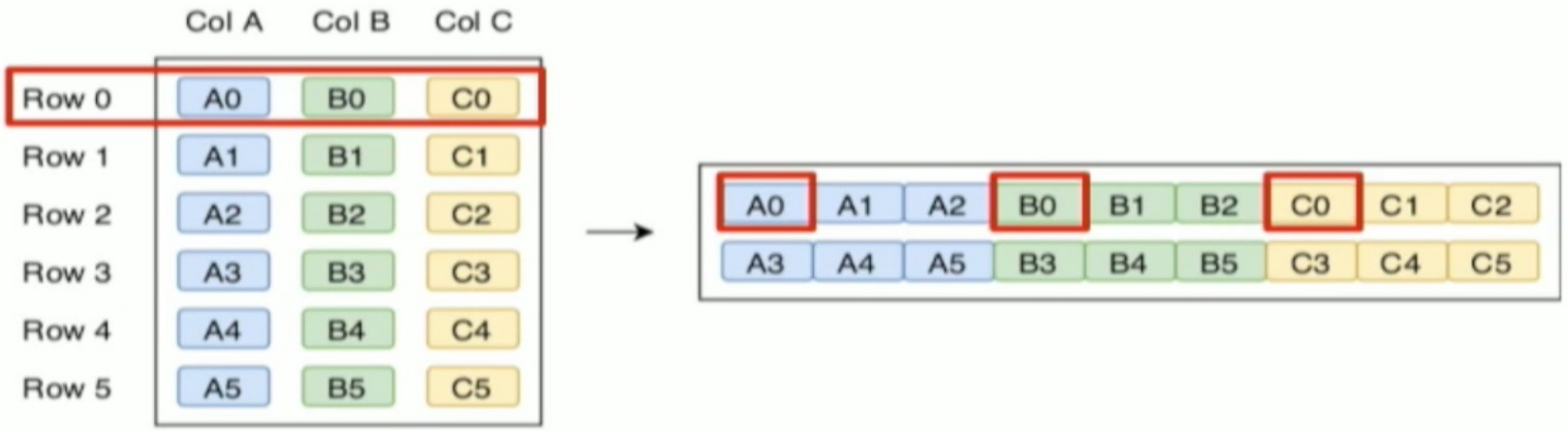
Storage Formats

Row-wise vs Columnar



Storage Formats

Hybrid



- Horizontal & vertical partitioning
- Used by Parquet & ORC
- Best of both worlds

Lab Exercise : Parquet/ORC format

Partitions

```
CREATE TABLE logs (ip String, createTime STRING,info String)
PARTITIONED BY (dt STRING, country STRING) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ‘,’;
```

```
LOAD DATA INPATH ‘file1'
INTO TABLE logs
PARTITION (dt='2012-01-01', country='GB');
```

```
/user/hive/warehouse/logs/dt=2012-01-01/country=GB/file1
                                                  /file2
                                        /country=US/file3
                              /dt=2012-01-02/country=GB/file4
                                                  /country=US/file5
                                                              /file6
```

```
SELECT ip,createTime, dt, info
FROM logs
WHERE country='GB';
```

This query will only scan file1, file2 and file4.

Dynamic Partition

Dynamic partition properties

```
Set hive.exec.dynamic.partition=true;
Set hive.exec.dynamic.partition.mode=nonstrict;
Set hive.exec.max.dynamic.partitions.pernode=<value>;
```

Inserting data into a Dynamic partition

```
Insert overwrite table LogAnalysis_partition partition(dt,country) select
ip,createTime,info,dt,country from Logs limit 50000;
```

Altering partitions

You can alter the partition table anytime to add/remove partitions

Alter Syntax Add partition

Alter table <tableName> add partition()

Set the location of the partition if data is added from outside
Alter table <tableName> partition(userid=3) set location
“hdfs://localhost:8020/user/hive/warehouse/<tableName>/userid=3

Alter Syntax Drop partition

ALTER TABLE some_table DROP IF EXISTS PARTITION(year = 2012);

Inserting data into partitions

- Insert into Hive partitioned Table using Values clause
- Inserting data into Hive Partition Table using SELECT clause
- Named insert data into Hive Partition Table

- Method 1:
  - INSERT INTO <Table name> PARTITION(id=1) VALUES (1, 'RRR');
- Method 2:
  - INSERT INTO <Table name> PARTITION(id) SELECT * FROM(SELECT 1 as id, 'abc' as name, '12/03/2003' as dob) dual;
- Method 3:
  - INSERT INTO insert_partition_demo PARTITION(dept=1) VALUES (1, 'abc');

Repairing of Partitions

- If the Partition directories/data are loaded in to a Partitioned table other than insert or load commands
  - You have to manually create the partitions and set the location by using alter command
  - Alternately, you can also use the msck command to repair the whole partition table.
    - MSCK REPAIR table <table name>

Lab Exercise - Partitioning

Buckets

Why Buckets?

- The first is to enable more efficient queries. Bucketing imposes extra structure on the table, which Hive can take advantage of when performing certain queries. In particular, a join of two tables that are bucketed on the same columns—which include the join columns.
- The second reason to bucket a table is to make sampling more efficient. When working with large datasets, it is very convenient to try out queries on a fraction of your dataset while you are in the process of developing or refining them.

```
CREATE TABLE Logs_Bucket(country String,ip String,dt String
,createTime String, line String)
   COMMENT 'This is the Bucketed table'
CLUSTERED BY(country) SORTED BY(ip) INTO 32 BUCKETS    ROW
FORMAT DELIMITED
         FIELDS TERMINATED BY ‘,’;
```

Within each bucket the
data is sorted in increasing order of ip. Such an organization allows the user to do efficient sampling on
the clustered column - in this case ip.

```
hive> SELECT * FROM Logs_Bucket
    > TABLESAMPLE(BUCKET 1 OUT OF 32 ON country);
```

Lab Exercise - Bucketing

Lab Exercise : JSON SerDe

Case.. When..Then in Hive

When we want to apply conditions on data in select clause we can use the Case… When.. Then clause

Syntax:

```
select studentid,name,marks,
   CASE
            WHEN  marks>70 THEN 'Pass'
            WHEN marks<70 THEN 'Fail'
      ELSE "No Marks"
      END as result
from marks;
```

Multi table insert

When we want to insert data from a single table in to multiple tables, we can use multi-table insert

Syntax:

```
FROM users
    insert overwrite table userslt3000 select * where userid<3000
    insert overwrite table usersgt3000 select * where userid>3000
```

Note: The destination tables should already have been present.

UDFs

Sometimes the query you want to write can't be expressed easily (or at all) using the built-in functions that Hive provides. By writing a user-defined function (UDF), Hive makes it easy to plug in your own processing code and invoke it from a Hive query.

UDFs have to be written in Java, the language that Hive itself is written in.

There are three types of UDF in Hive: (regular) UDFs, UDAFs (user-defined aggregate functions), and UDTFs (user-defined table-generating functions). They differ in the numbers of rows that they accept as input and produce as output:

- A UDF operates on a single row and produces a single row as its output. Most functions, such as mathematical functions and string functions, are of this type.
- A UDAF works on multiple input rows and creates a single output row. Aggregate
- functions include such functions as COUNT and MAX.
- A UDTF operates on a single row and produces multiple rows—a table—as output.

UDFs

A UDF for stripping characters from the ends of strings

```
package com.hadoopbook.hive;
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
public class Strip extends UDF {
 private Text result = new Text();

 public Text evaluate(Text str) {
  if (str == null) {
   return null;
  }
 result.set(StringUtils.strip(str.toString()));
  return result;
 }

 public Text evaluate(Text str, String stripChars) {
  if (str == null) {
   return null;
  }
  result.set(StringUtils.strip(str.toString(), stripChars));
  return result;
 }
}
```

hive> **SELECT strip(' bee ') FROM dummy;**
bee

Notice that the UDF's name is not case-sensitive:
hive> **SELECT STRIP(' bee ') FROM dummy;**
bee

Lab Exercise - UDFs

Altering/Dropping tables

ALTER TABLE source RENAME TO target;

In addition to updating the table metadata, ALTER TABLE moves the underlying table directory so that it reflects the new name. In the current example, /user/hive/warehouse/ source is renamed to /user/hive/warehouse/target. (An external table's underlying

ALTER TABLE target ADD COLUMNS (col3 STRING);

The new column col3 is added after the existing (nonpartition) columns.

DROP TABLE – Deletes data and metadata.

If you want to delete all the data in a table, but keep the table definition (like DELETE or TRUNCATE in MySQL), then you can simply delete the data files. For example:

hive> **dfs -rmr /user/hive/warehouse/my_table;**

Views

A view is a sort of “virtual table” that is defined by a SELECT statement. Views can be used to present data to users in a different way to the way it is actually stored on disk.

```
CREATE VIEW valid_records
AS
SELECT *
FROM records2
WHERE temperature != 9999
  AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9);
```

Transactions in Hive

To allow Transactions in hive set the following properties in hive shell or hive-site.xml

```
set hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
set hive.support.concurrency=true;
set hive.enforce.bucketing=true;
set hive.compactor.initiator.on=true;
set hive.compactor.worker.threads=2;
```

Syntax:

```
create external table hive_transactions(empid int, name string) clustered by (empid) into 4 buckets stored
as ORC TBLPROPERTIES ('transactional'='true');
```

```
insert into hive_transactions values(1,"raju");
insert into hive_transactions values(2,"John");
```

```
update hive_transactions1 set name=”Rakesh” where empid=1;
```

Hive Performance statistics

- **Hive Analyze table command – Table statistics**
  - Hive uses cost based optimizer.
  - Statistics serve as the input to the cost functions of the Hive optimizer so that it can compare different plans and choose best among them.
  - Hive uses the statistics such as number of rows in tables or table partition to generate an optimal query plan.
  - Other than optimizer, hive uses mentioned statistics in many other ways.
- **Uses of Hive Table or Partition Statistics**
  - There are many ways statistics can be useful.
  - Hive cost based optimizer uses the statistics to generate an optimal query plan.
  - Users can quickly get the answers for some of their queries by only querying stored statistics rather than firing long-running execution plans.

Hive Performance statistics

What Statistics are collected?

- Number of rows
- Number of files
- Size in Bytes
- Number of partition if the table is partitioned

Hive ANALYZE TABLE Command Syntax

ANALYZE TABLE [db_name.]tablename [PARTITION(partcol1[=val1], partcol2[=val2], ...)]
COMPUTE STATISTICS [FOR COLUMNS]

Note: If you donot want to gather or compute statistics set the following property to false
set hive.stats.autogather=false;

Hive Performance statistics

- **Hive Explain Command**
  - Latest version of Hive uses Cost Based Optimizer (CBO) to increase Hive query performance
  - Hive uses a cost-based optimizer to determine the best method for scan and join operations, join order, and aggregate operations.
  - You can use the Apache **Hive EXPLAIN command** to display the actual execution plan that Hive query engine generates and uses while executing any query in the Hadoop ecosystem.

Hive Performance statistics

Apache Hive Cost Based Optimizer

- Latest version of Apache Hive uses the cost based optimizer to determine the best methods for the query to be executed in the Hadoop ecosystem.
- Optimizer uses the statistics to determine the optimal and best execution plan for hive queries that involves complex logic and multiple table joins.
- The statistics include the following:
- The number of rows
- It also includes the number files in the Hadoop directory
- File size in bytes

Apache Hive EXPLAIN command Syntax

EXPLAIN [EXTENDED|DEPENDENCY|AUTHORIZATION] query;

Hive Performance statistics

TableScan	If one or more tables are small enough to fit in memory, the mapper scans the large table and do the joins. This table scan operation performs large table scan.
Select	Identifies the Select operators in the given query. This operator projects only columns that are given in the select clause of query. Query may have multiple select based on the complexity of the query.
GroupBy	This feature identifies grouping on records during computations. Feature may also vary on the complexity of the query.
Map	This is the map step that Hive uses to execute query. The map phase reads the tables and output the join key-value pairs into an intermediate file.
Reduce	This is the reduce phase that Hive uses. The reduce gets the sorted data and does the join.
MapJoin	If there are multiple tables joins that involve small and big table MapJoin phase is used. Small table (dimension table) joins big table (fact table). It is very fast since it saves shuffle and reduce stage.
Filter	This operator identifies the filter conditions such as WHERE clause.
Broadcast	This operator identifies the broadcast phase. If two tables are joined together, if joining table is small, Hive sends copy of table to all nodes to make tables as collocated.
Reducer	Reduces phases combines the results from reduce phase
Drop	This operator identifies drop table clause.
Create	This operator identifies any create table or create table as clause in the query.
Alter	This operator identifies any alter table clause in the query.

Question?