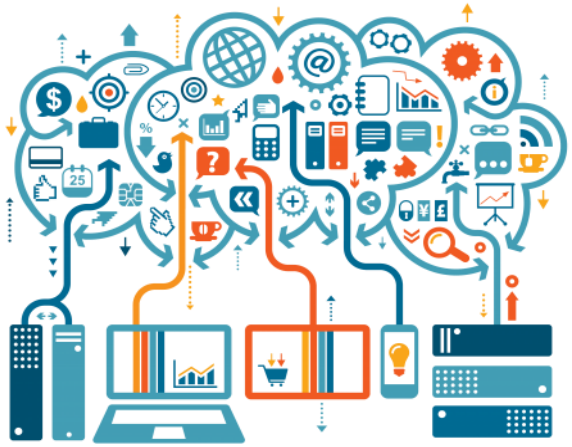
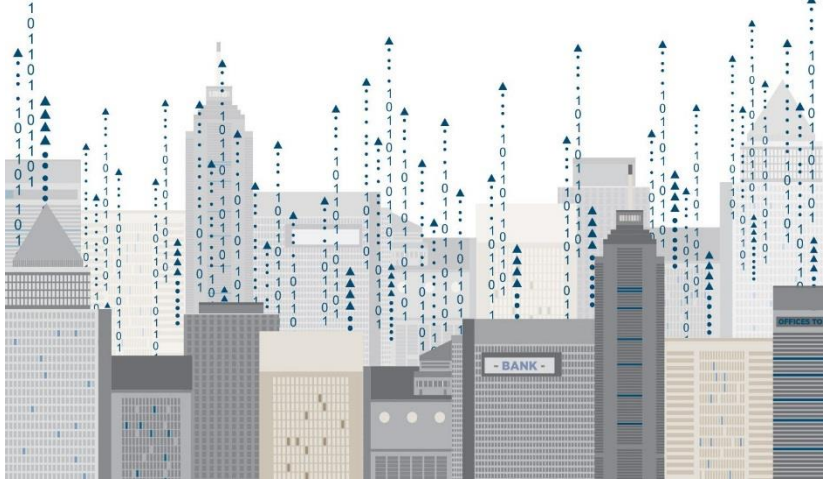




Big Data Analytics

Fall 2025

Lecture 10

Data Analysts with Hadoop

Challenges of MapReduce

- Requires complex, verbose Java (python using PySpark these days)
- Complex: Understand distributed computing
- Imperative programming: define step-by-step procedures
- No support for SQL (motivation for hive)
- Optimized for batch processing – latency issues
- Job execution stack too long for simple queries (remember uber?)
- No schema management
- No connectivity with BI tools (can connect through Hive, Presto)
- Maintenance burden

… Enter Hive!

The Base of Hive

- Data warehouse infra on top of Hadoop
- Enables data summarization
- Enables ad-hoc queries - select * type queries :)
- Initially developed by Facebook.
- Hive stores data in HDFS
- Supports SQL like Query Language : HiveQL
- HQL statements are broken down by the Hive service into MapReduce jobs
  - For MR execution engine only
  - Tez engine (DAG-based), Spark engine (Spark-based) and Long Live and Process (LLAP) (in-memory caching)

Engine	Description	Enabled by
MapReduce	Original, stable, slow	Default in older Hadoop
Apache Tez	DAG-based engine — faster, optimized	hive.execution.engine=tez
Apache Spark	Uses Spark as execution engine	hive.execution.engine=spark
LLAP (Live Long and Process)	In-memory caching for low-latency	Part of Hive-on-Tez

Step	Description
1. Parse	Hive parses HQL into an Abstract Syntax Tree (AST).
2. Semantic Analysis	Validates schema & columns via Metastore.
3. Logical Plan	Builds a logical operator tree (select, filter, join, etc.).
4. Optimization	Applies rule-based or cost-based optimizations.
5. Physical Plan	Converts logical plan to physical plan → MapReduce DAG / Tez DAG / Spark DAG.
6. Execution	Submitted to YARN for distributed execution.

Organization

- Data in Hive is organized into Tables
- Work with data inside HDFS
- Tables:
  - Data: Files in HDFS
  - Schema: In the form of metadata stored in RDBMS (logical layer)
  - Have a corresponding HDFS directory
  - Data in a table is serialized
- Supports primitive types and also nestable collections: Array and Map(Key Value pair)

Data Model

- Tables
  - Each table has a corresponding HDFS directory
  - Hive provides built-in serialization formats (lazy serialization – delayed until needed)
- Partitions: Each table can have 1 or more horizontal partitions
  - Example: Table T in the directory : /wh/T.
  - If T is partitioned on columns ds = ‘20090101’, and ctry = ‘US’, will be stored /wh/T/ds=20090101/ctry=US.
- Buckets: Data in each partition may be divided into buckets based on the hash of a column in the table
  - Each bucket is stored as a file in the partition directory

```
CREATE TABLE ecommerce_sales (
  order_id STRING,
  category STRING,
  product_id STRING,
  price FLOAT,
  payment_type STRING
)
PARTITIONED BY (country STRING, order_date DATE)
STORED AS PARQUET;
```

```
/warehouse/ecommerce_sales/country=Pakistan/order_date=2025-11-01/
/warehouse/ecommerce_sales/country=Pakistan/order_date=2025-11-02/
/warehouse/ecommerce_sales/country=USA/order_date=2025-11-02/
```

```
SELECT SUM(price)
FROM ecommerce_sales
WHERE country = 'Pakistan' AND order_date = '2025-11-02';
```

Only 1 partition will be read – no need to scan the whole dataset

```
CREATE TABLE ecommerce_sales_bucketed (
  order_id STRING,
  customer_id STRING,
  category STRING,
  product_id STRING,
  price FLOAT,
  payment_type STRING
)
PARTITIONED BY (country STRING)
CLUSTERED BY (customer_id) INTO 8 BUCKETS
STORED AS ORC;
```

All records for the same cst always go to the same bucket file (=hash(cid) mod 8)

A single cst is completely in 1 bucket – not spread out.

Hive assigns **each bucket to one reducer only**

```
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000000_0
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000001_0
...
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000007_0
```

Each bucket file corresponds to a hash(cid) mod 8

```
SELECT customer_id, COUNT(order_id)
FROM ecommerce_sales_bucketed
WHERE country = 'Pakistan'
GROUP BY customer_id;
```

Data pre-hashed by cid, so Hive parallelizes aggregations and joins using bucket files.

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
             page_url STRING, referrer_url STRING,
          friends ARRAY<BIGINT>, properties MAP<STRING, STRING>
          ip STRING COMMENT 'IP Address of the User')
          COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '1'
        COLLECTION ITEMS TERMINATED BY '2'
        MAP KEYS TERMINATED BY '3'
STORED AS SEQUENCEFILE;
```

Hive Key Principles

Enables users to query data as if they were using a relational database - much simpler than writing raw MapReduce code

Hive Key Principles

Facilitates large-scale data storage, querying, and analysis

Hive Key Principles

External/Internal Tables, Schema-on-read, Join/Group-by, UDF, Partitioning, Bucketing, Direct load, Spark integration,

Hive Key Principles

Going beyond HiveQL

TRANSFORM: incorporate custom **mapper and reducer scripts** directly into Hive queries.

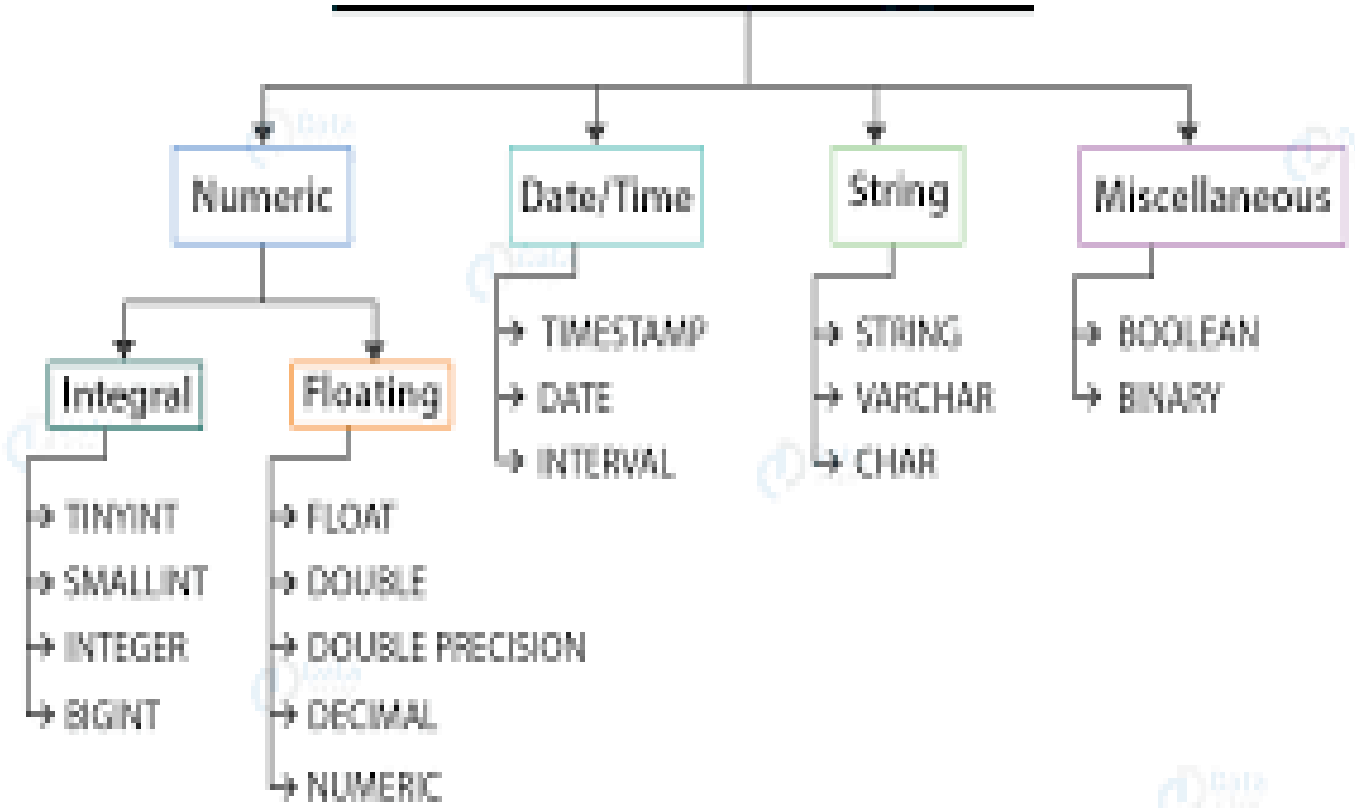
Hive is Not…

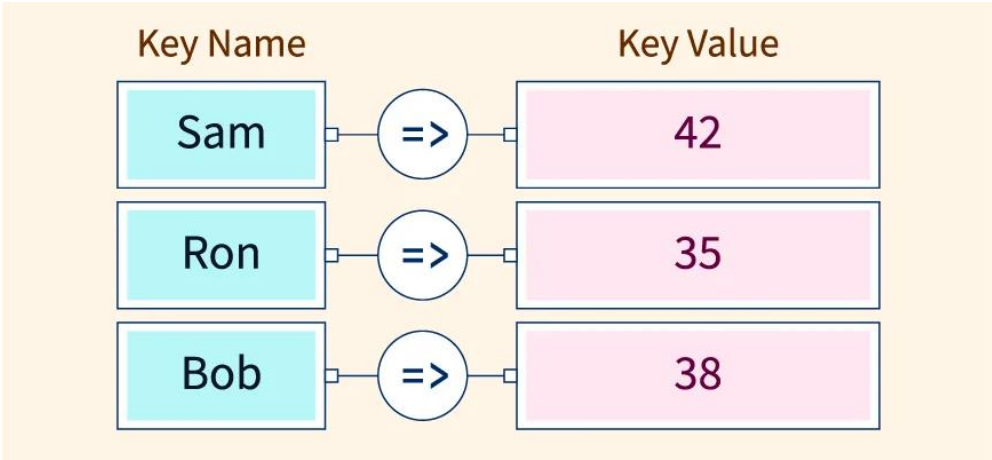
- Hive designed for batch processing of large datasets
- Not an OLTP or real-time system
- Latency (Time to start & finish one query) is high compared to RDBMS
- Throughput (Data processed per batch) is high compared to RDBMS
- Even when dealing with relatively small data (<100 MB)

I Am, I Am Not

Nuts and Bolts

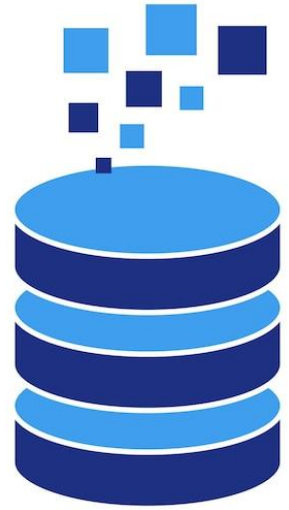
- Supports primitive types, and Associative Arrays (maps, dictionaries, hash tables) Lists, Struct.





Nuts and Bolts

- HQL supports DDL
  - Define or modify the structure - Create, alter, drop, truncate
  - Automatically commits
- HQL supports DML
  - Don’t change the database schema but rather modify the actual data within tables.
  - Insert, update, delete, select
- HQL has limited equality and join predicates (inner, left and right outer, full) - and has no inserts on existing tables.
- But improving in newer versions



Requirement	Description
Format	ORC or Parquet recommended
Bucketing	Required for full ACID tables
Property	'transactional'='true'
Execution engine	Tez or Spark (not MR)
HiveServer2 + Metastore	Must be configured with concurrency control

:)

Operation	Supported?	Notes
INSERT INTO	Yes	Appends new files to table (always supported)
INSERT OVERWRITE	Yes	Replaces table or partition data
UPDATE / DELETE	Yes (only in ACID tables)	Requires transactional setup
MERGE	Yes (ACID only)	For upsert operations

Serialize - Deserialize

- Convert data into a format suitable for storage or transmission - and reconstruct back to original form
- Purpose of Ser: persist the data's state
- Purpose of DeSer: reconstruct original state for application usage
- Allows complex data structures (lists, dictionaries) to be saved as a single entity
- Considerations:
  - Overhead
  - Data Integrity (for non-ACID tables)

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

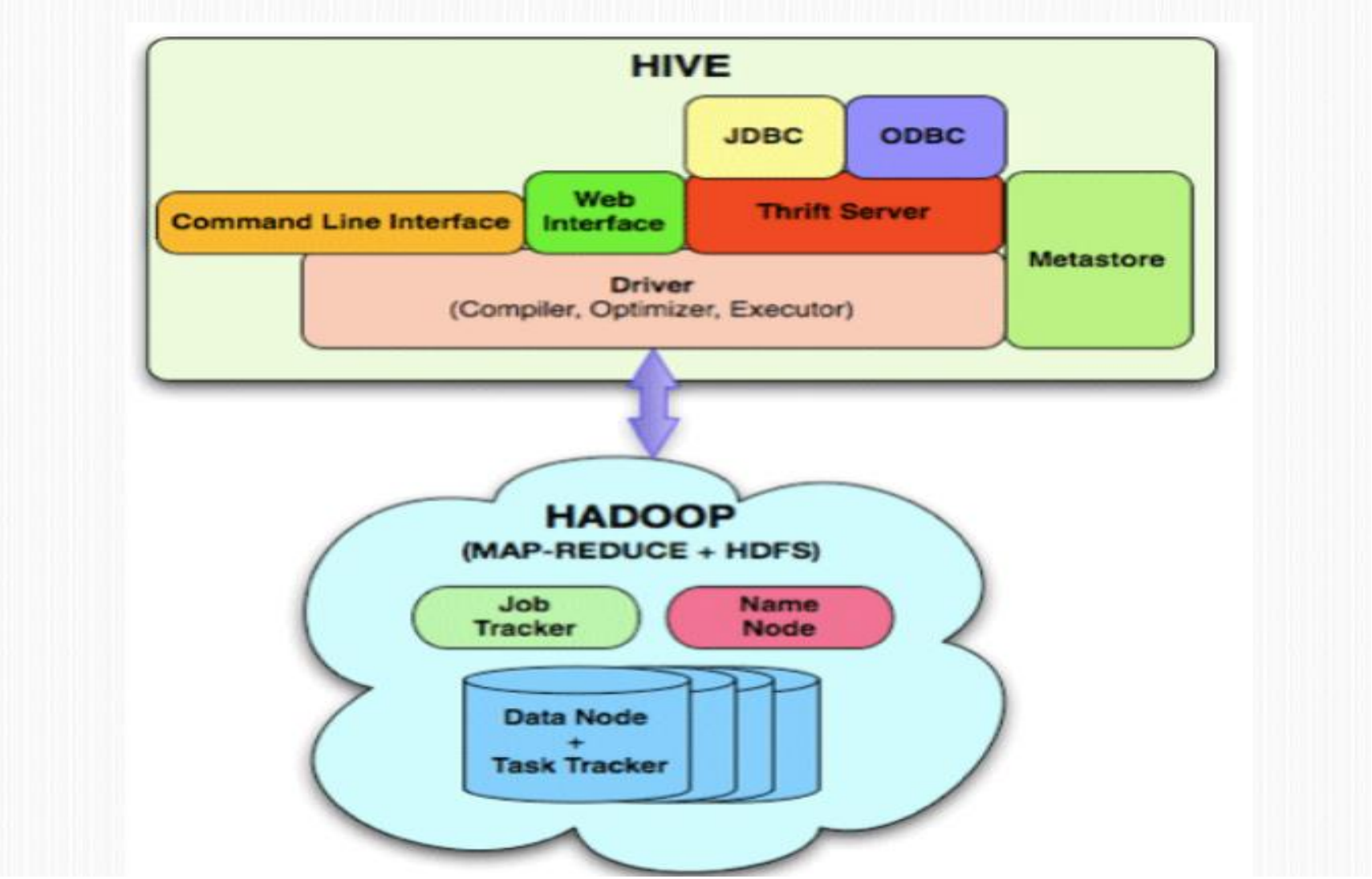
Serialize - Deserialize

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return f"Person(name={self.name}, age={self.age})"

# Create an instance of Person
person = Person("John Doe", 30)
print("Original object:", person)
```

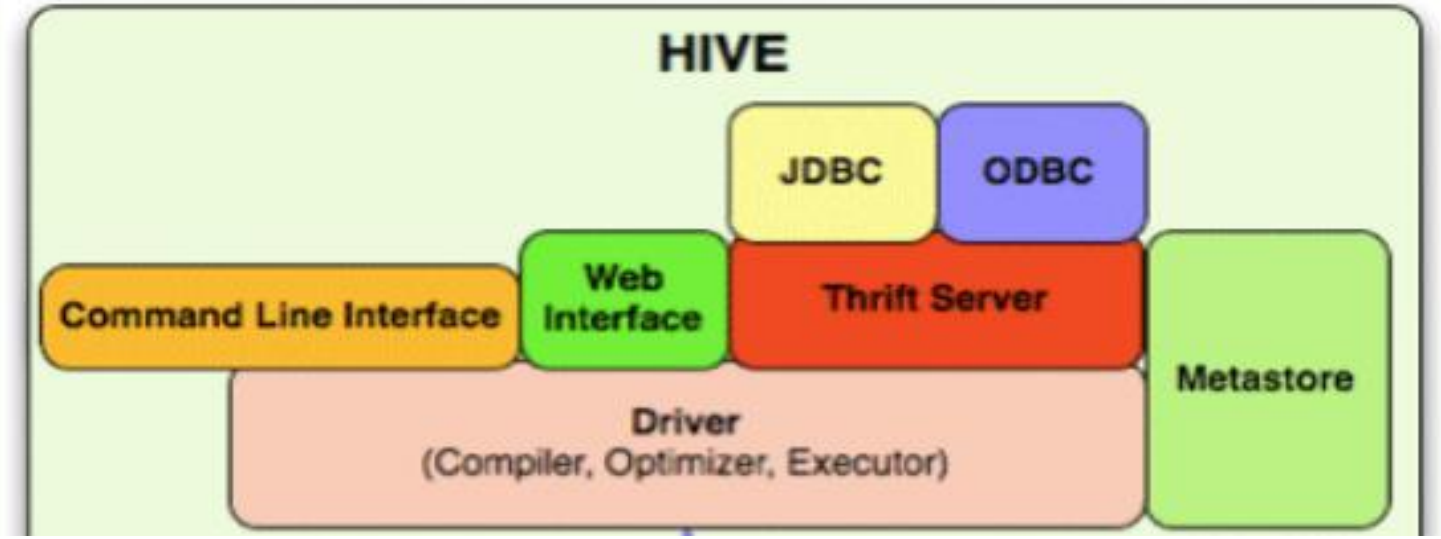
```
# Serialize the object
serialized_person = pickle.dumps(person)

# Show the serialized binary data
print("Serialized object (binary):", serialized_person)


# Deserialize the object
deserialized_person = pickle.loads(serialized_person)

# Show the deserialized object
print("Deserialized object:", deserialized_person)
```

```
Original object: Person(name=John Doe, age=30)

Serialized object (binary): b'\x80\x04\x95...'

Deserialized object: Person(name=John Doe, age=30)
```

Hive Architecture

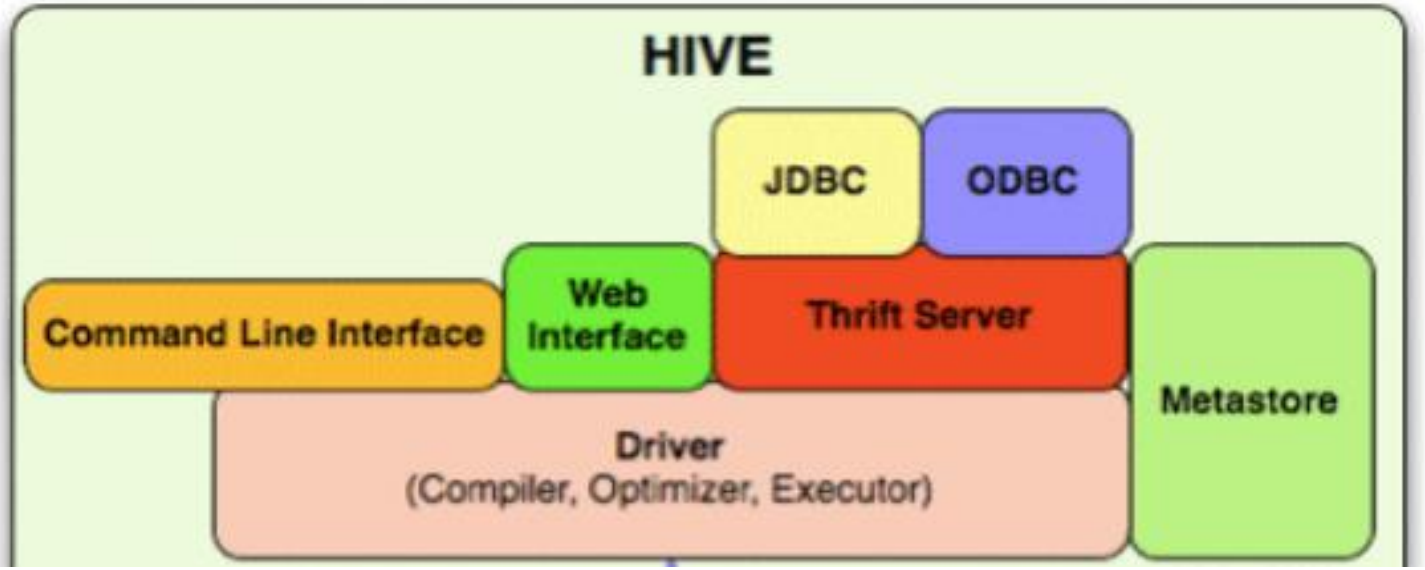


Hive Architecture



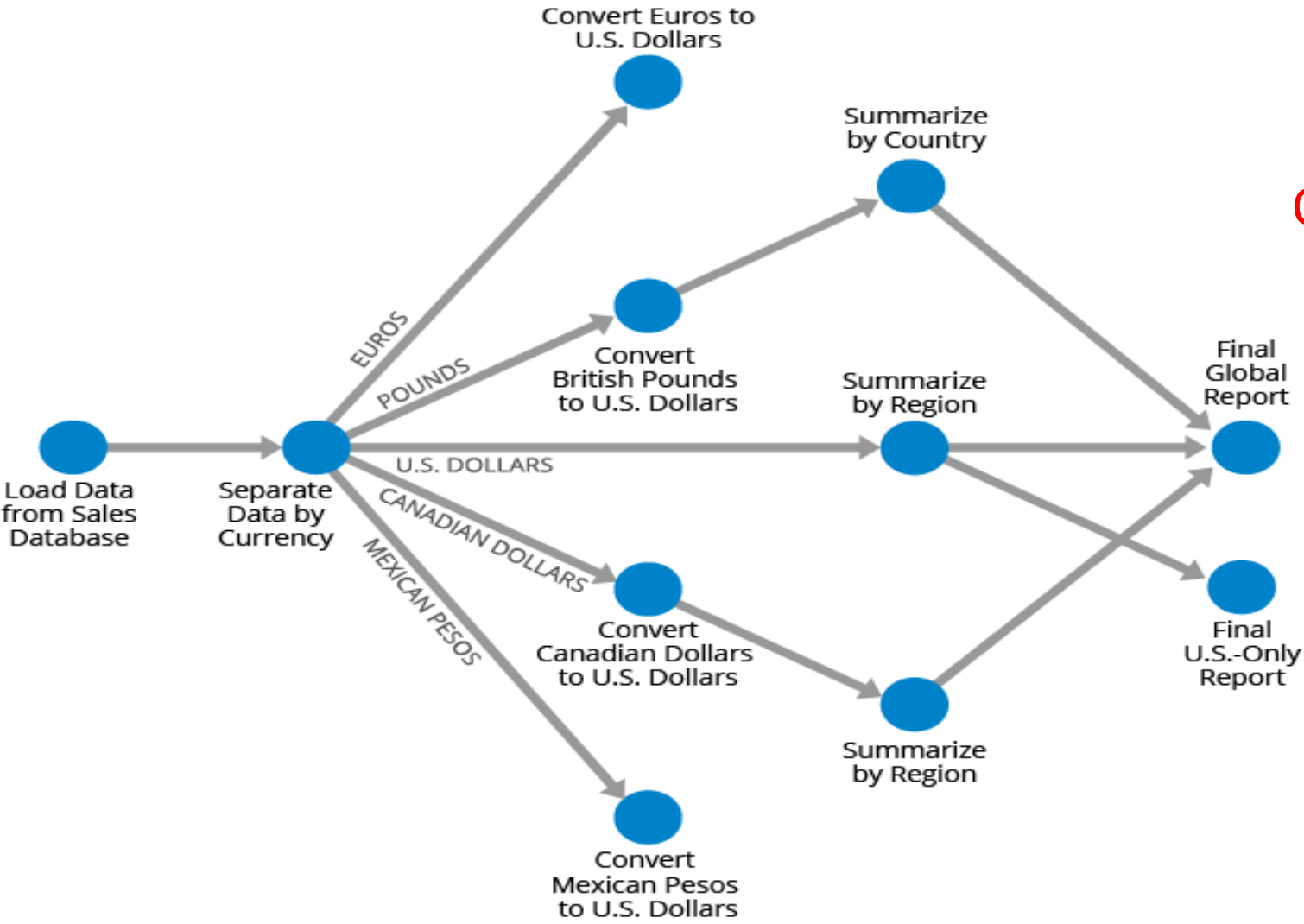
- External Interfaces:
  - Web UI : Management
  - Hive CLI : Run Queries, Browse Tables etc
  - API : JDBC, ODBC
- Metastore: System catalog - contains metadata about Hive tables
- Driver: Manages life cycle of a HiveQL statement during compilation, optimization and execution

Hive Architecture



- Compiler : translates HiveQL statement into a plan which consists of a DAG of map-reduce jobs – Why DAG?
- Database: is a namespace for tables (meaning?)
- Table: Metadata for table contains list of columns and their types, owner, storage and SerDe information - Also contains any user supplied key and value data.
- Partition: Each partition can have it own columns and SerDe and storage information.

Why DAG for queries?



Dependency Management

Optimize – eliminate unnecessary steps

Parallel execution

Fault tolerance – run only the dependent nodes when a task fails

Effective resource allocation – as we know dependencies

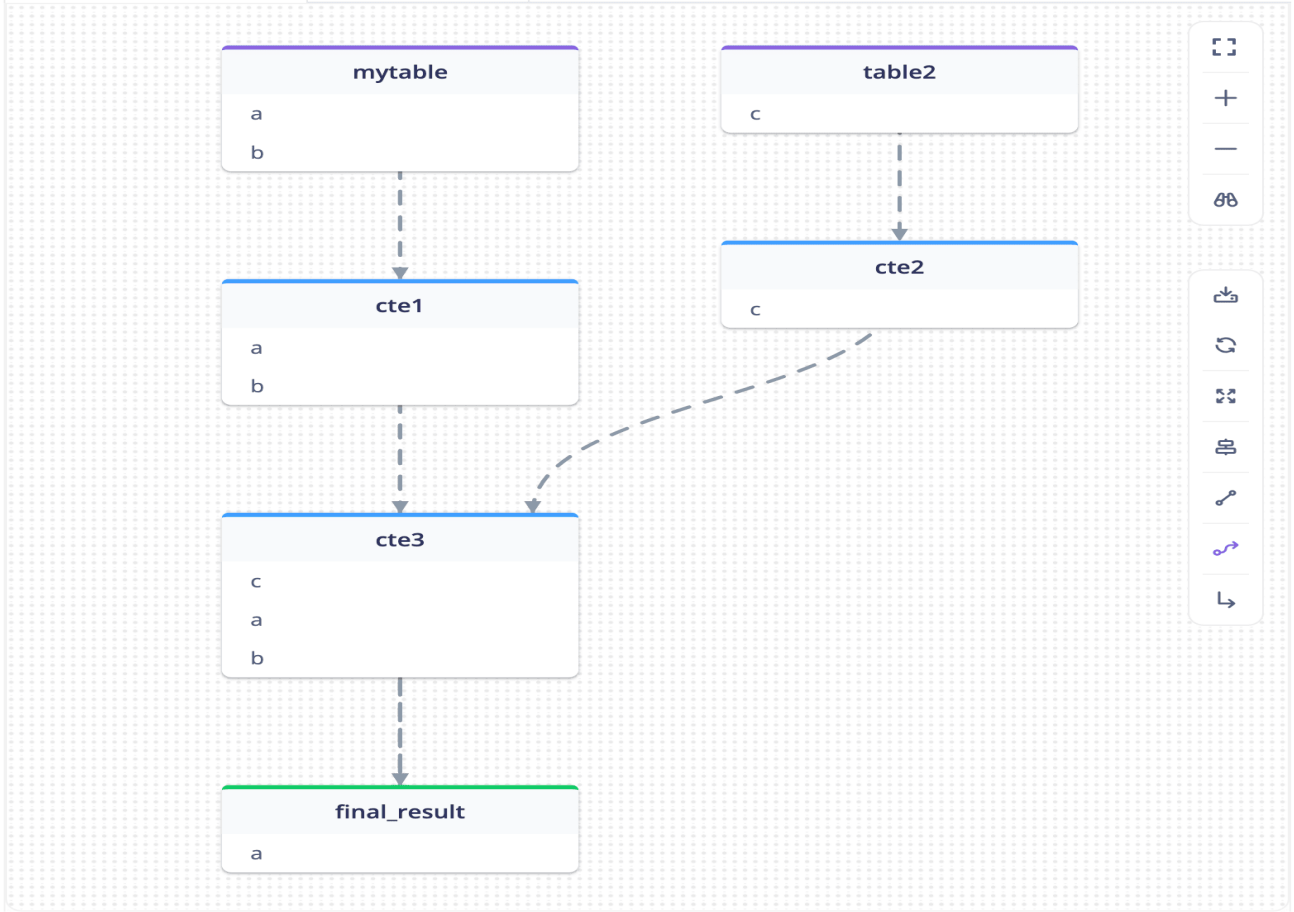
Multi CTEs - DAG in SQL

mysql | Properties | Open in a new page | Share

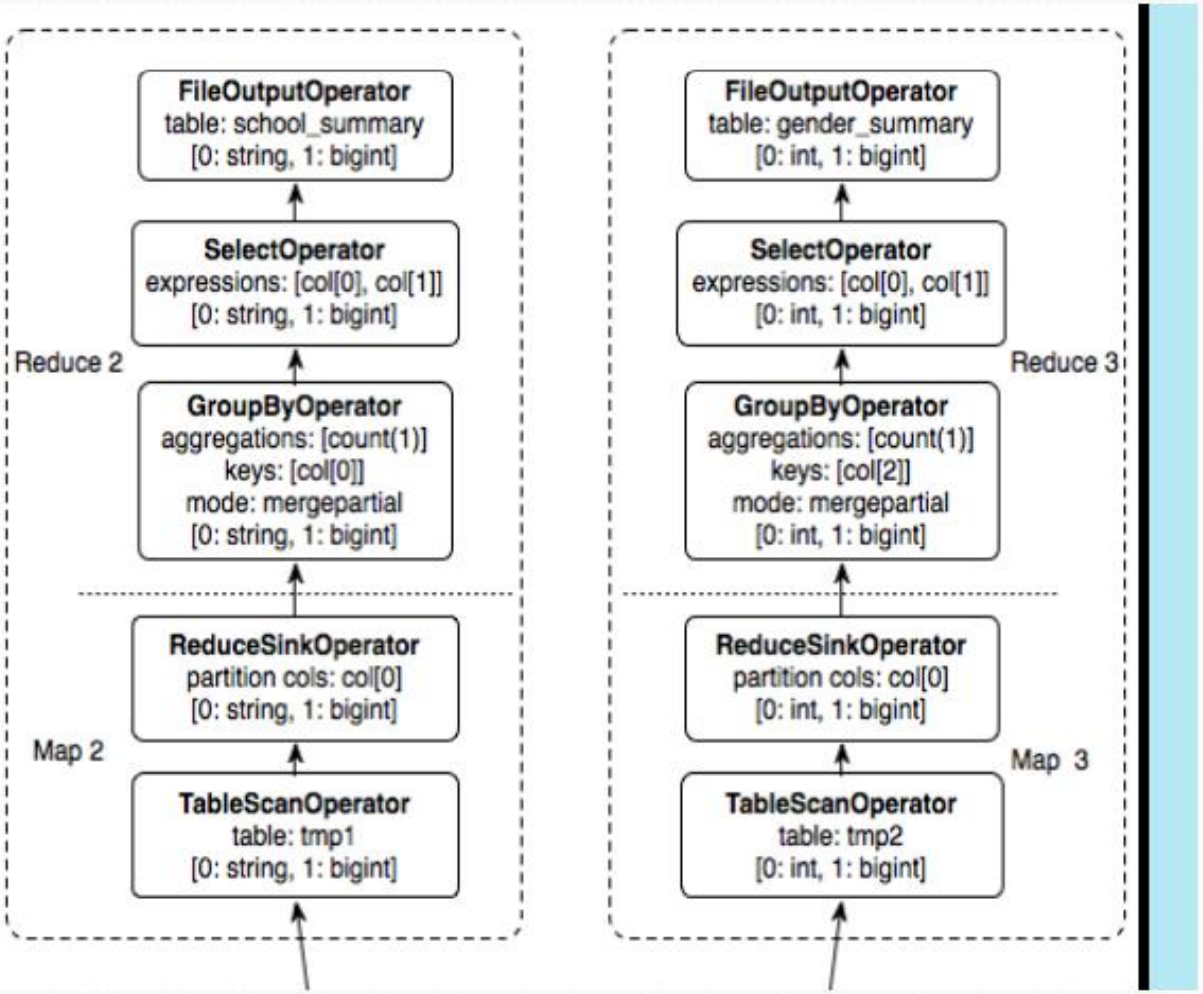
Last edit was made 0 seconds ago · by [redacted]

```sql
WITH
cte1 AS (
    SELECT
        a,
        b
    FROM mytable
),

cte2 AS (
    SELECT c
    FROM table2
),

cte3 AS (
    SELECT
        ct1.a,
        cte2.c
    FROM cte1
    INNER JOIN cte2 ON cte1.b = cte2.c
),

final_result AS (
    SELECT
        cte1.a,
        cte3.c
    FROM cte1
    INNER JOIN cte3 ON cte1.a = cte3.a
)

SELECT *
FROM final_result;
```

Table-level lineage | ER Diagram



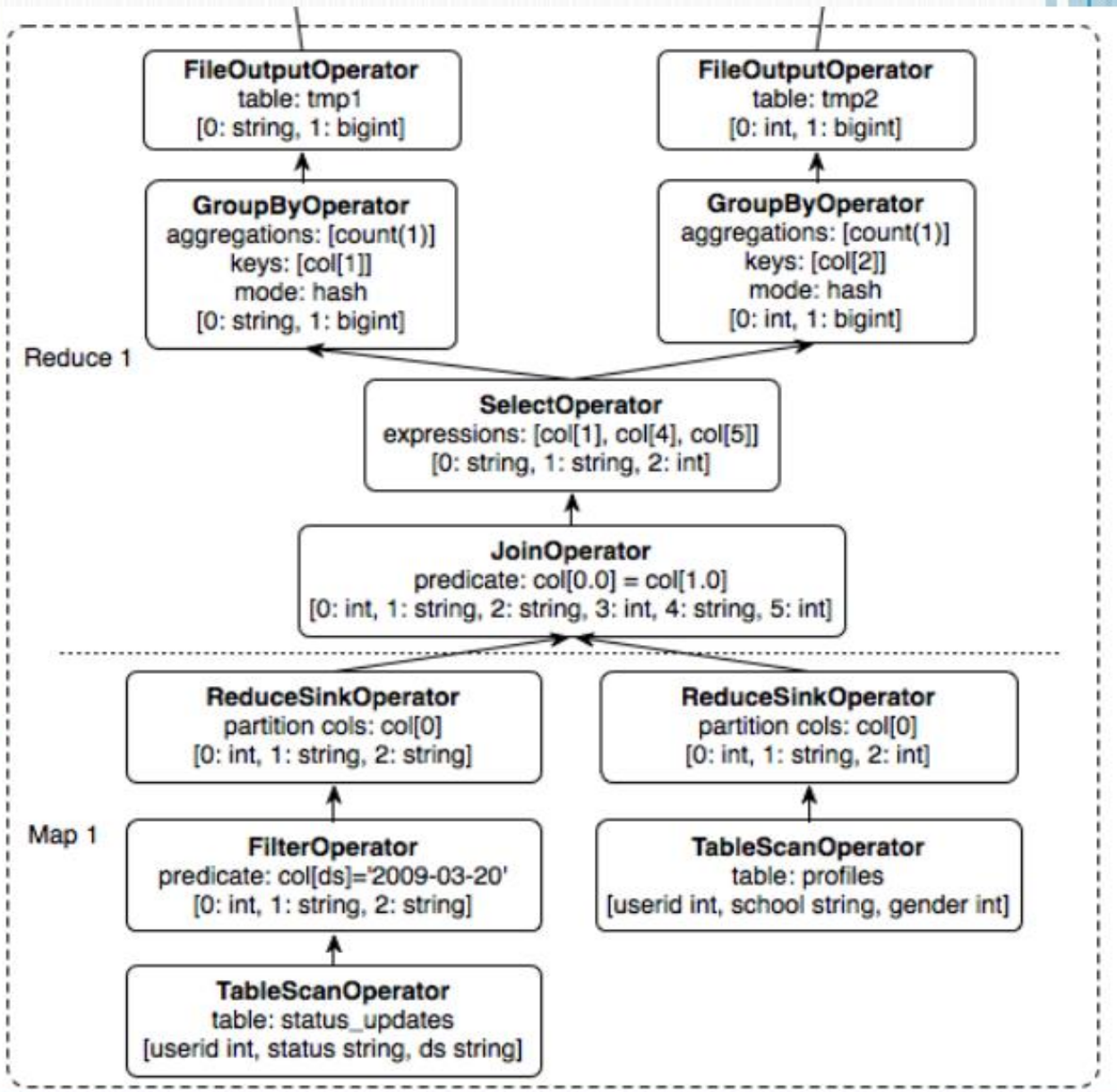
Doc
Comment
History
Snippet
Linter
Lineage



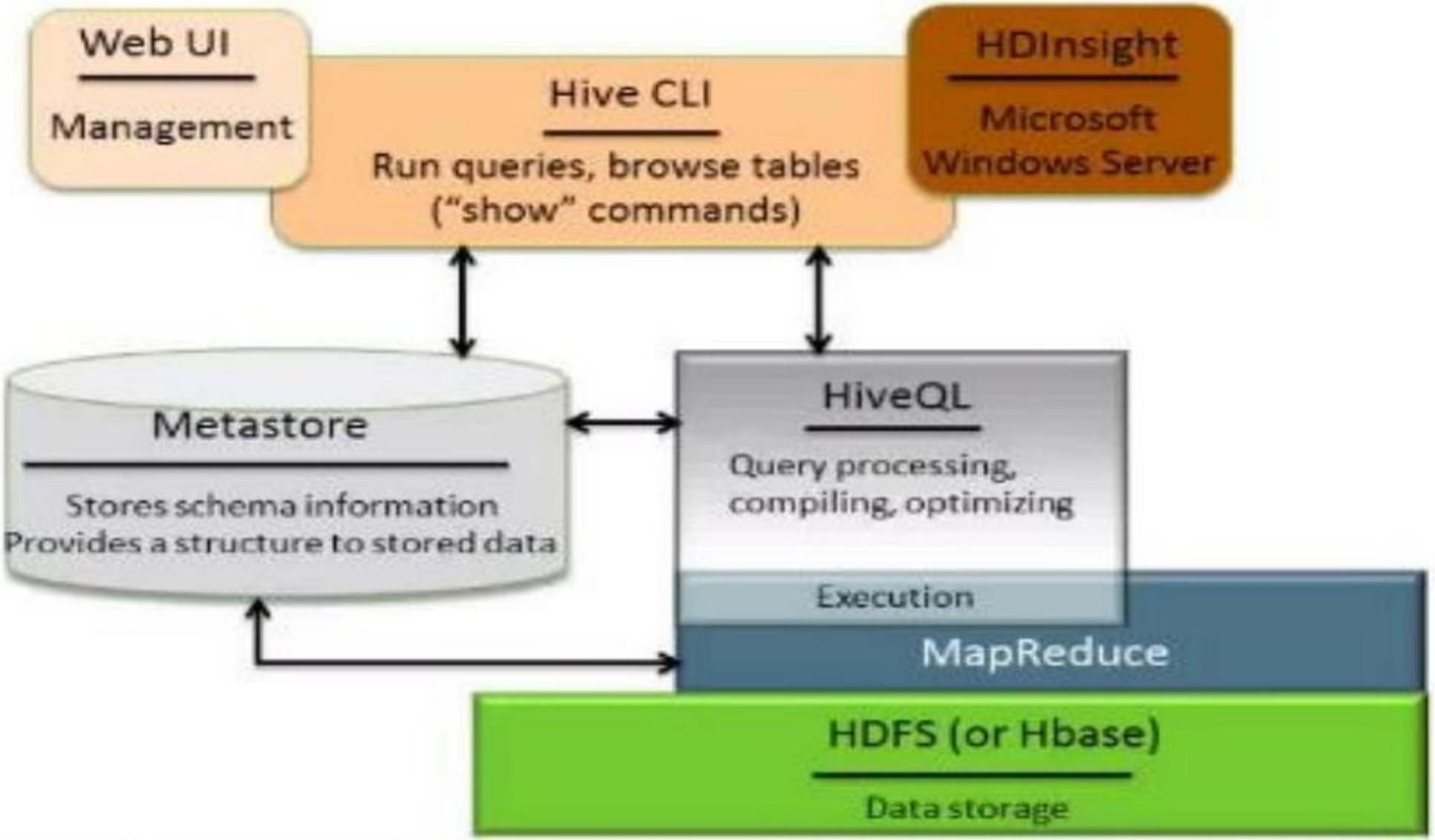
Top

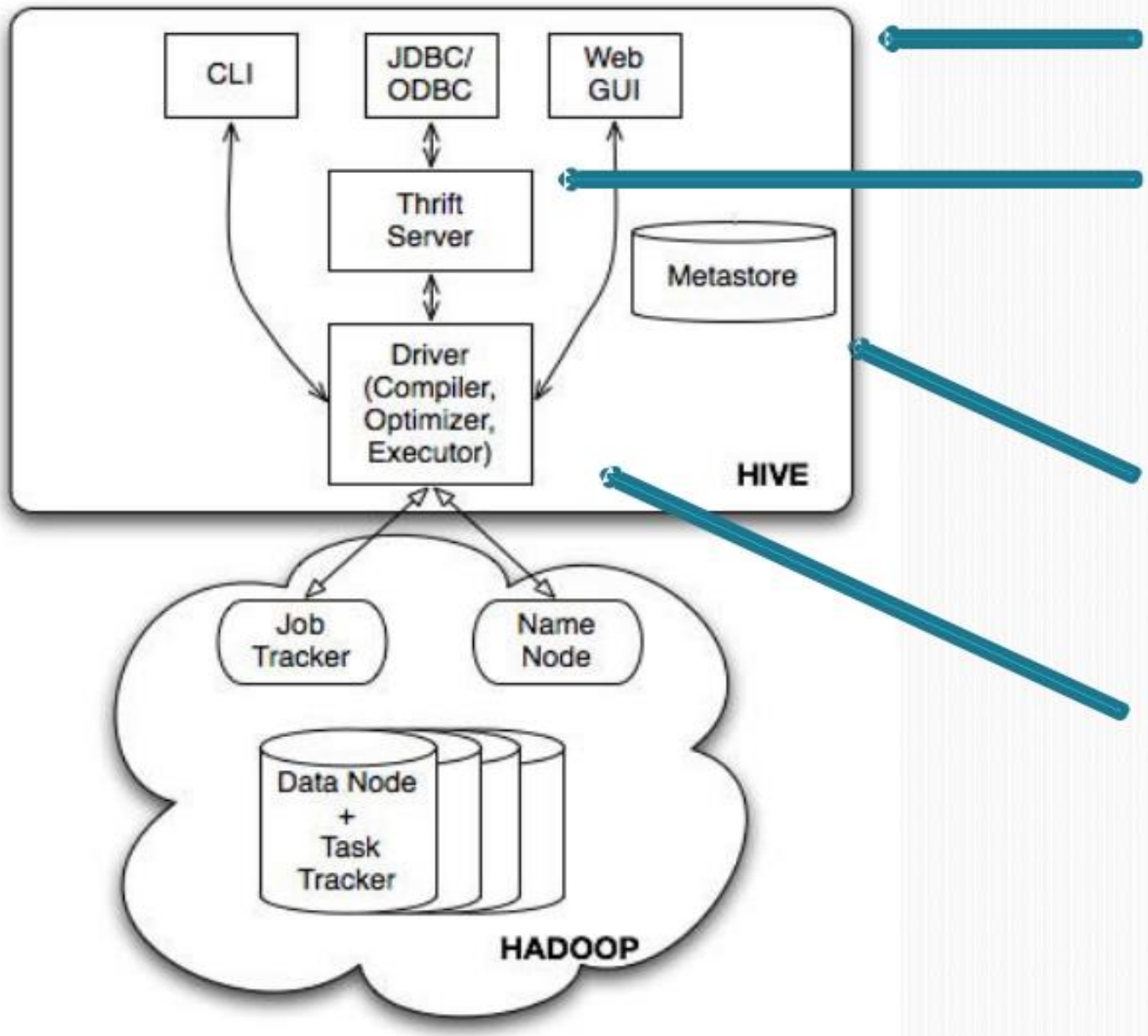
Figure 2: Query plan with 3 map-reduce jobs for multi-table insert query

Bottom



System for querying and managing structured data





External interface: Both user interface like command line (cli) and web UI

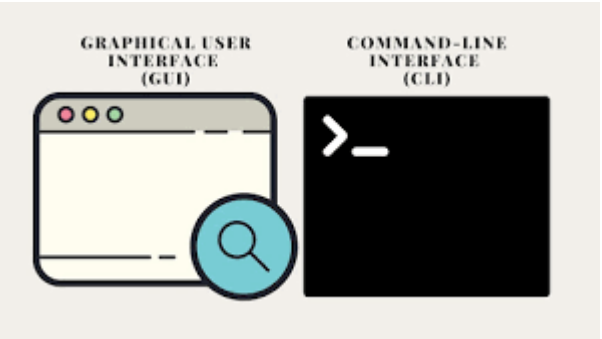
Thrift is a framework for cross-language services, where a server written in one language (like Java) can also support clients in other languages.

Metastore is the system catalog. All other components of Hive interact with metastore

The Driver manages the life cycle (statistics) of a HiveQL statement during compilation, optimization and execution

CLI

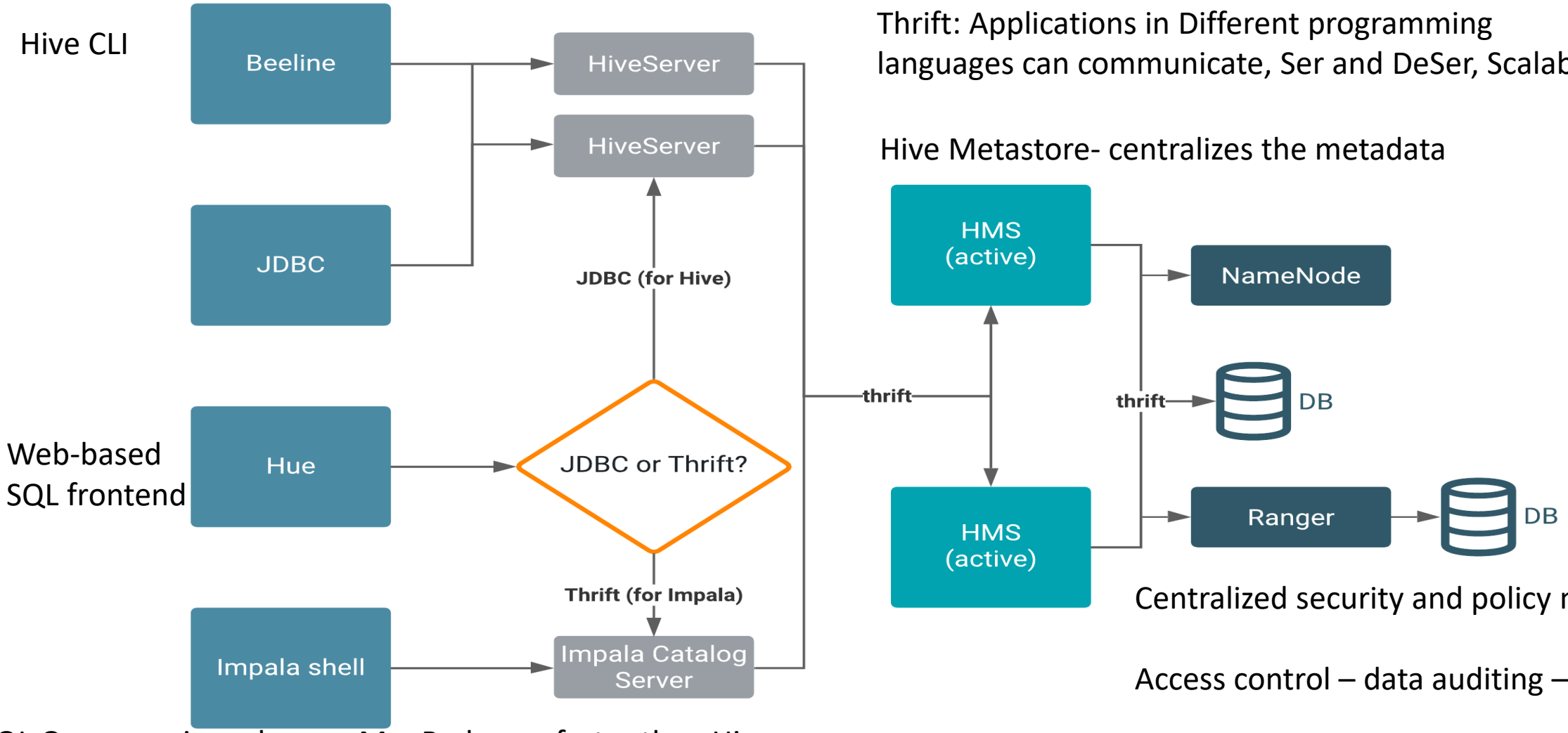
- There are several ways to interact with Hive, including some popular graphical user interface but CLI is sometimes preferable (Beeline)
- CLI allows creating, inspecting schema and query tables, etc.
- All commands and queries go to the Driver, which complies, optimizes and executes queries usually with MapReduce jobs.
- Hive communicates with Job Tracker to initiate the MapReduce job.



Metastore

- The system catalog which contains metadata about the tables stored in Hive
- This data is specified during table creation and reused very time the table is referenced in HiveQL
- Contains the following objects:
  - Database : the namespace for tables
  - Table: metadata for table contains list of columns and their types, owners, storage and SerDe information
  - Partition: each partition can have its own columns and SerDe and storage information

Hive Metastore (Cloudera)



Thrift: Applications in Different programming languages can communicate, Ser and DeSer, Scalable

Hive Metastore- centralizes the metadata

Centralized security and policy management

Access control – data auditing – data masking

SQL Query engine – bypass MapReduce – faster than Hive

Compile

- Converts DDL-DML to a plan.
- Parser: transforms a query to a parse tree representation
- Semantic analyzer: parse tree to internal (block-based) representation
- Plan generator: internal representation to a logical plan
- Optimizer: multiple passes over plan and rewrites
  - Combine multiple joins sharing join key into single multiway join - single MR!
  - Add repartition operators (redistribute data to balance workload)
  - Prune columns early (select req. columns as early as possible)
  - Push query predicates closer to the table scan operators (predicate pushdown – why?)

Compile

- Physical Plan generator: converts logical plan into physical plan = DAG of MR jobs

HiveQL

- SQL like language: HiveQL
- DDL : to create tables with specific serialization formats
- DML : load and insert to load data from external sources and insert query results into Hive tables
- Do not support updating and deleting rows in existing tables
- Supports Multi-Table insert
- Supports Select, Project, Join, Aggregate
- Supports Union all and Sub-queries in the From clause

HiveQL

- Can be extended with custom functions (UDFs)
- User Defined Transformation Function(UDTF) https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide+UDTF
- User Defined Aggregation Function (UDAF)
- Users can embed custom map-reduce scripts written in any language using a simple row-based streaming interface

- Running time example: Status Meme

When Facebook users update their status, the updates are logged into flat files in an NFS directory /logs/status_updates

status_updates(userid int,status string,ds string) using a load statement like below.

```
LOAD DATA LOCAL INPATH '/logs/status_updates'
INTO TABLE status_updates PARTITION (ds='2009-03-20')
```

Compute daily statistics on the frequency of status updates based on gender and school

```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

Advantages – Child's play

- Fast response
- Scalable and extensible
- Thrift
- Ser/De
- Flexible schema

Advantages – Child's play

- No row-level insert, update or delete operations – load it!
- Four file formats: TEXTFILE, SEQUENCEFILE, ORC and RCFILE.
- Example: 'NASDAQ_daily_prices_B.csv' a log file of stocks record of NASDAQ.
- exchange,stock_symbol,date,stock_price_open,stock_price_high,stock_price_low,stock_price_close,stock_volume,stock_price_adj_close
- NASDAQ,BBND,2010-02-08,2.92,2.98,2.86,2.96,483800,2.96
  NASDAQ,BBND,2010-02-05,2.85,2.94,2.79,2.93,884000,2.93
  NASDAQ,BBND,2010-02-04,2.83,2.88,2.78,2.83,1333300,2.83

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

- Create a database:

```
hive> CREATE DATABASE financials;
or
hive> CREATE DATABASE IF NOT EXISTS financials;
```

- Describe table:

```
hive> DESCRIBE DATABASE financials;
OK
Financials
        hdfs://localhost:54310/user/hive/warehouse/financials.db
```

- Use database:

```
hive> USE financials;
```

- Drop database:

```
hive> DROP DATABASE IF EXISTS financials;
```

Some Facts

- Use LOAD DATA to import data into a Hive table
- Hive>Load Data LOCAL INPATH '/home/sunny/EmployeeDetails.txt ' INTO TABLE Employee
- Use the word OVERWRITE to write over a file of the same name
- We can Load data from Local file system by using LOCAL keyword
- Inserting Data into new table by using SELECT statement
- For Example, INSERT OVERWRITE SELECT * FROM Employee

Operation	Command Syntax
See current tables	Hive>Show TABLES
Check the table name	Hive>Describe <Table_Name>
Change the table name	Hive>Alter Table <table_Name> Rename to mytab
Add a column	Hive> Alter Table <table_Name> ADD COLUMNS (MyID String)
Drop a partition	Hive>Alter Table <table_Name> DROP PARTITION (Age>70)

Some Facts

- WHERE Clause
- UNION All and DISTINCT
- GROUP BY and HAVING
- LIMIT Clause
- Hive Supports Sub-Queries but only in FROM Clause
- JOINS , ORDER BY, SORT BY

Output Data

- Output data produced by Hive is structured, typically stored in a relational database.
- For cluster, MySQL or similar relational database is required.
- The result tables then can be manipulated using HiveQL in the similar way of SQL to relational database.

```
hive> LOAD DATA LOCAL INPATH
'/Users/nqt289/Desktop/NASDAQ_daily_prices_B.csv'
    > OVERWRITE INTO TABLE stocks;
Copying data from
file:/Users/nqt289/Desktop/NASDAQ_daily_prices_B.csv
Copying file:
file:/Users/nqt289/Desktop/NASDAQ_daily_prices_B.csv
Loading data to table mydb.stocks
Deleted
hdfs://localhost:54310/Users/nqt289/Desktop/NASDAQ_
daily_prices_B.csv
OK
Time taken: 0.231 seconds
```

```
hive> SELECT * FROM STOCKS WHERE price_open='2.92';
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201403311509_0003, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201403311509_0003
Kill Command = /Users/nqt289/hadoop-0.20.2/bin/../bin/hadoop job  -Dmapred.job.tracker=localhost:54311 -kill job_201403311509_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-03-31 15:39:20,577 Stage-1 map = 0%,  reduce = 0%
2014-03-31 15:39:23,597 Stage-1 map = 100%,  reduce = 0%
2014-03-31 15:39:26,625 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201403311509_0003
MapReduce Jobs Launched:
Job 0: Map: 1   HDFS Read: 21998523 HDFS Write: 5166 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
NASDAQ  BBND    2010-02-08  2.92    2.98    2.86    2.96    483800
2.96
NASDAQ  BTFG    2009-12-21  2.92    2.92    2.75    2.79    15100
2.79
NASDAQ  BJCT    2004-04-21  2.92    2.98    2.9     2.98    3200
2.98
NASDAQ  BJCT    2004-04-20  2.92    3.0     2.92    2.95    27900
2.95
...
```

Primitive Data Types

Type	Comments
TINYINT, SMALLINT, INT, BIGINT	1, 2, 4 and 8-byte integers
BOOLEAN	TRUE/FALSE
FLOAT, DOUBLE	Single and double precision real numbers
STRING	Character string
TIMESTAMP	Unix-epoch offset *or* datetime string
DECIMAL	Arbitrary-precision decimal
BINARY	Opaque; ignore these bytes

Complex Data Types

Type	Comments
STRUCT	A collection of elements If S is of type STRUCT {a INT, b INT}: S.a returns element a
MAP	Key-value tuple If M is a map from 'group' to GID: M['group'] returns value of GID
ARRAY	Indexed list If A is an array of elements ['a','b','c']: A[0] returns 'a'

HiveQL Limitations

- Missing large parts of full SQL specification:
  - HAVING clause in SELECT (must be used after Group By – !directly)
  - Correlated sub-queries (use them with JOIN)
  - Updatable or materialized views (updatable views not possible)
  - Stored procedures (not allowed; rather use UDFs)

```
SELECT department, COUNT(employee_id) AS
employee_count
FROM employees
GROUP BY department
HAVING employee_count > 5;
```

```sql
SELECT employee_id, salary
FROM employees e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
);
```

```sql
SELECT e.employee_id, e.salary
FROM employees e
JOIN (
    SELECT department_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
) dept_avg
ON e.department_id = dept_avg.department_id
WHERE e.salary > dept_avg.avg_salary;
```

```
WITH dept_avg AS (
    SELECT department_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
)
SELECT e.employee_id, e.salary
FROM employees e
JOIN dept_avg
ON e.department_id = dept_avg.department_id
WHERE e.salary > dept_avg.avg_salary;
```

```
CREATE MATERIALIZED VIEW sales_summary
AS
SELECT product_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY product_id;
```

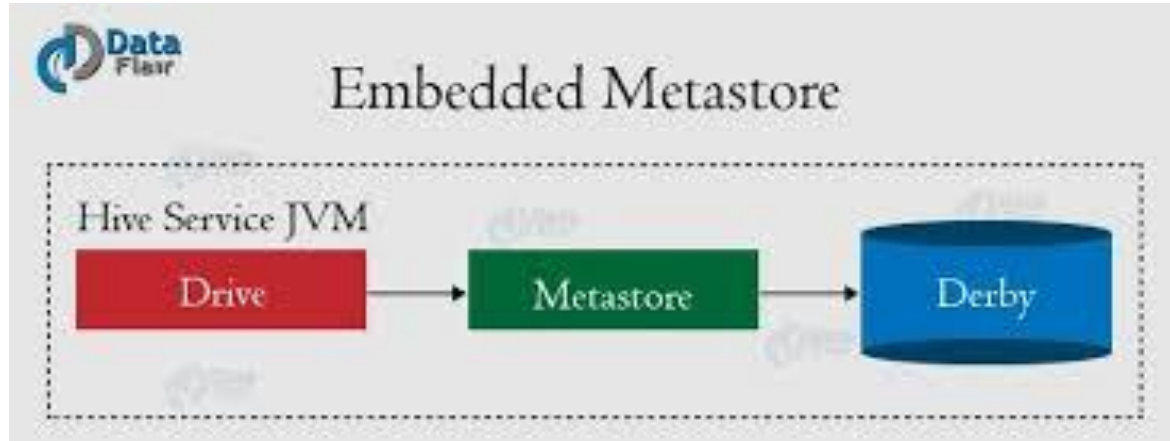
```
ALTER MATERIALIZED VIEW sales_summary REBUILD;
```

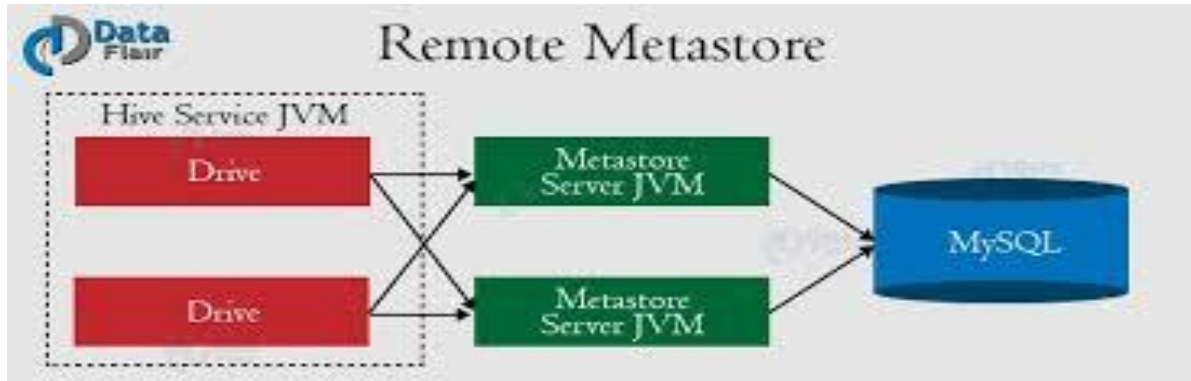
Hive Metastore

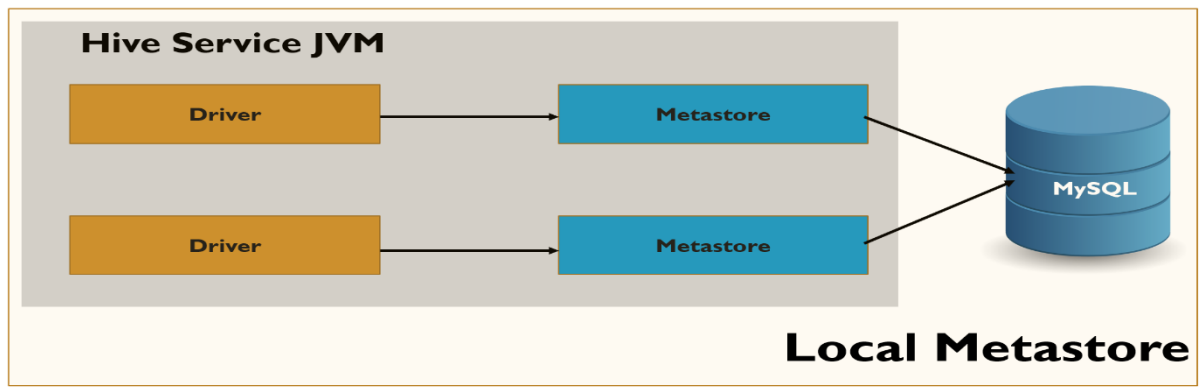
- Stores Hive metadata
- Default metastore database uses Apache Derby
- Default: **embedded, single-user database** in local or testing environments
- **Only one active connection** at a time
- For production, replace with Postgres, MySQL (configure hive-site.xml)

Hive Metastore

- Embedded: Metastore runs within the Hive service process – uses embedded Derby - for unit tests, development, single-user
- Local: Each Hive service has its own metastore process - client connects to the metastore directly – shared external db – for multi-user environment
- Remote: Metastore runs as a separate standalone service - Each Hive client connects to the metastore server, which connects to the metadata database itself.


Data Flair
Embedded Metastore
Hive Service JVM
Drive
Metastore
Derby


Data Flair
Remote Metastore
Hive Service JVM
Drive
Drive
Metastore Server JVM
Metastore Server JVM
MySQL

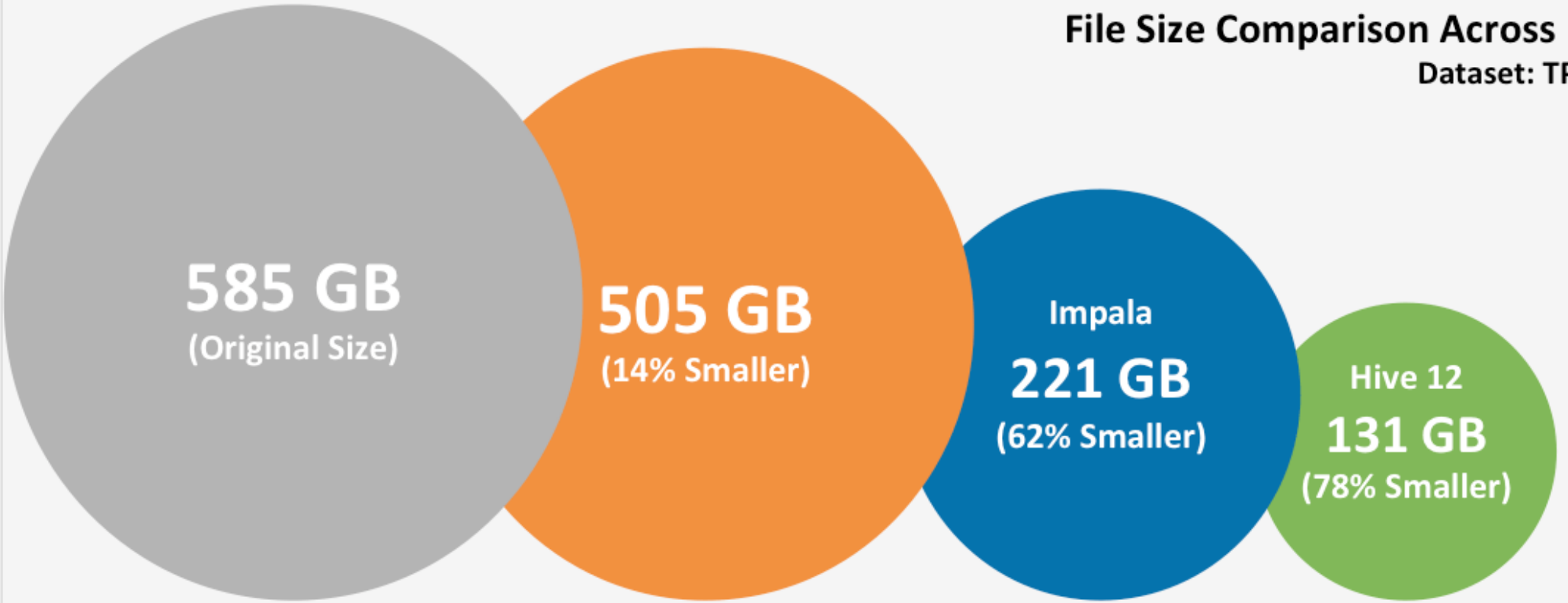

Hive Service JVM
Driver
Driver
Metastore
Metastore
MySQL
Local Metastore

Hive Warehouse

- Hive tables are stored in the Hive “warehouse”
  - Default HDFS location: /user/hive/warehouse
- Tables are stored as sub-directories in the warehouse directory
- Partitions are subdirectories of tables
- External tables are supported in Hive
- The actual data is stored in flat files

File Size Comparison Across Encoding Methods

Dataset: TPC-DS Scale 500 Dataset



Encoded with **Text**

Encoded with **RCFile**

Encoded with **Parquet**

Encoded with **ORCFile**

- Larger Block Sizes
- Columnar format arranges columns adjacent within the file for compression & fast access

Hive Schemas

- Hive is schema-on-read
  - Schema is only enforced when the data is read (at query time)
  - Allows greater flexibility: same data can be read using multiple schemas
- Contrast with an RDBMS, which is schema-on-write
  - Schema is enforced when the data is loaded
  - Speeds up queries at the expense of load times

Create Table Syntax

```
CREATE TABLE table_name
    (col1 data_type,
     col2 data_type,
     col3 data_type,
     col4 datatype )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   STORED AS format_type;
```

Simple Table

```
CREATE TABLE page_view
    (viewTime INT,
     userid BIGINT,
     page_url STRING,
     referrer_url STRING,
     ip STRING COMMENT 'IP Address of the User' )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE;
```

More Complex Table

```
CREATE TABLE employees  (
    (name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
                   city:STRING,
                   state:STRING,
                   zip:INT>)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  STORED AS TEXTFILE;
```

More About Tables

- CREATE TABLE
  - LOAD: file moved into Hive's data warehouse directory
  - DROP: both metadata and data deleted
- CREATE EXTERNAL TABLE
  - LOAD: no files moved
  - DROP: only metadata deleted
  - Use this when sharing with other Hadoop applications, or when you want to use multiple schemas on the same data

Managed vs External

- Managed: Hive owns and manages both the metadata and the actual data stored on HDFS.
- When Hive is the main tool for managing data and when you want Hive to control the lifecycle of both data and metadata

```
CREATE TABLE employees (id INT, name STRING)
STORED AS ORC;
```

/user/hive/warehouse/employees

External

- Hive manages only metadata while **actual data files are outside Hive's control**.
- The data can reside in HDFS
- Hive only maintains a reference to this data
- Dropping the table doesn’t delete the data

```
CREATE EXTERNAL TABLE employees (id INT, name STRING)
STORED AS ORC
LOCATION '/user/data/employees';
```

External Table

```
CREATE EXTERNAL TABLE page_view_stg
    (viewTime INT,
     userid BIGINT,
     page_url STRING,
     referrer_url STRING,
     ip STRING COMMENT 'IP Address of the User')
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE
   LOCATION '/user/staging/page_view';
```

Partitioning

- Can make some queries faster
- Divide data based on partition column
- Use PARTITION BY clause when creating table
- Use PARTITION clause when loading data
- SHOW PARTITIONS will show a table’s partitions

```
CREATE TABLE sales (
  id INT,
  product STRING,
  amount DOUBLE
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET;



INSERT INTO TABLE sales PARTITION (year=2024, month=10)
VALUES (1, 'ProductA', 250.0),
       (2, 'ProductB', 300.0);




SELECT * FROM sales
WHERE year = 2024 AND month = 10;
```

Bucketing

- Can speed up queries that involve sampling the data
  - Sampling works without bucketing, but Hive has to scan the entire dataset
- Use CLUSTERED BY when creating table
  - For sorted buckets, add SORTED BY
- To query a sample of your data, use TABLESAMPLE

```
CREATE TABLE employees (
  id INT,
  name STRING,
  department STRING
)
CLUSTERED BY (id) INTO 4 BUCKETS
STORED AS TEXTFILE;
```

```
INSERT INTO TABLE employees
VALUES (1, 'Alice', 'HR'),
       (2, 'Bob', 'Finance'),
       (3, 'Charlie', 'Engineering');
```

```
SELECT * FROM employees
WHERE department = 'Engineering';
```

Browsing Tables And Partitions

Command	Comments
`SHOW TABLES;`	Show all the tables in the database
`SHOW TABLES 'page.*';`	Show tables matching the specification (uses regex syntax)
`SHOW PARTITIONS page_view;`	Show the partitions of the page_view table
`DESCRIBE page_view;`	List columns of the table
`DESCRIBE EXTENDED page_view;`	More information on columns (useful only for debugging)
`DESCRIBE page_view PARTITION (ds='2008-10-31');`	List information about a partition

Loading Data

- Use LOAD DATA to load data from a file or directory
  - Will read from HDFS unless LOCAL keyword is specified
  - Will append data unless OVERWRITE specified
  - PARTITION required if destination table is partitioned

```
LOAD DATA LOCAL INPATH '/tmp/pv_2008-06-8_us.txt'
  OVERWRITE INTO TABLE page_view
  PARTITION (date='2008-06-08', country='US')
```

Inserting Data

- Use INSERT to load data from a Hive query
  - Will append data unless OVERWRITE specified
  - PARTITION required if destination table is partitioned

```
FROM page_view_stg pvs
  INSERT OVERWRITE TABLE page_view
  PARTITION (dt='2008-06-08', country='US')
  SELECT pvs.viewTime, pvs.userid, pvs.page_url,
       pvs.referrer_url
  WHERE pvs.country = 'US';
```

Inserting Data

- Normally only one partition can be inserted into with a single INSERT
- A multi-insert lets you insert into multiple partitions

```
FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view
PARTITION  ( dt='2008-06-08', country='US` )
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url WHERE pvs.country = 'US'
INSERT OVERWRITE TABLE page_view
PARTITION ( dt='2008-06-08', country='CA' )
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url WHERE pvs.country = 'CA'
INSERT OVERWRITE TABLE page_view
PARTITION ( dt='2008-06-08', country='UK' )
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url WHERE pvs.country = 'UK';
```

Inserting Data During Table Creation

- Use AS SELECT in the CREATE TABLE statement to populate a table as it is created

```
CREATE TABLE page_view AS
  SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url
  FROM page_view_stg pvs
  WHERE pvs.country = 'US';
```

Loading And Inserting Data: Summary

Use this	For this purpose
`LOAD`	Load data from a file or directory
`INSERT`	Load data from a query • One partition at a time • Use multiple INSERTs to insert into multiple partitions in the one query
`CREATE TABLE AS (CTAS)`	Insert data while creating a table
Add/modify external file	Load new data into external table

Sample Select Clauses

- Select from a single table

```
SELECT *
   FROM sales
   WHERE amount > 10 AND
            region = "US";
```

- Select from a partitioned table

```
SELECT page_views.*
 FROM page_views
 WHERE page_views.date >= '2008-03-01' AND
      page_views.date <= '2008-03-31'
```

Relational Operators

- ALL and DISTINCT
  - Specify whether duplicate rows should be returned
  - ALL is the default (all matching rows are returned)
  - DISTINCT removes duplicate rows from the result set
- WHERE
  - Filters by expression
  - Does not support IN, EXISTS or sub-queries in the WHERE clause
- LIMIT
  - Indicates the number of rows to be returned

Relational Operators

- GROUP BY
  - Group data by column values
  - Select statement can only include columns included in the GROUP BY clause
- ORDER BY / SORT BY
  - ORDER BY performs total ordering
    - Slow, poor performance
  - SORT BY performs partial ordering
    - Sorts output from each reducer

Advanced Hive Operations

- JOIN
  - If only one column in each table is used in the join, then only one MapReduce job will run
    - This results in 1 MapReduce job:

      ```
      SELECT * FROM a JOIN b ON a.key = b.key JOIN c ON b.key = c.key
      ```

    - This results in 2 MapReduce jobs:

      ```
      SELECT * FROM a JOIN b ON a.key = b.key JOIN c ON b.key2 = c.key
      ```

  - If multiple tables are joined, put the biggest table last and the reducer will stream the last table, buffer the others
  - Use left semi-joins to take the place of IN/EXISTS

    ```
    SELECT a.key, a.val FROM a LEFT SEMI JOIN b on a.key = b.key;
    ```

Advanced Hive Operations

- JOIN
  - Do not specify join conditions in the WHERE clause
    - Hive does not know how to optimise such queries
    - Will compute a full Cartesian product before filtering it
- Join Example

```
SELECT
  a.ymd, a.price_close, b.price_close
FROM stocks a
JOIN stocks b ON a.ymd = b.ymd
WHERE a.symbol = 'AAPL' AND
      b.symbol = 'IBM' AND
      a.ymd > '2010-01-01';
```

Hive Stinger

- MPP-style execution of Hive queries
- Available since Hive 0.13
- No MapReduce

References

- http://hive.apache.org

HIVE CHEAT SHEET

Hive Basics

Apache Hive

It is a data warehouse infrastructure based on Hadoop framework which is perfectly suitable for data summarization, analysis and querying. It uses an SQL like language called HQL (Hive query Language)

HQL: It is a query language used to write the custom map reduce framework in Hive to perform more sophisticated analysis of the data

Table: Table in hive is a table which contains logically stored data

Hive Interfaces:

- Hive interfaces includes WEB UI
- Hive command line
- HD insight (windows server)

Components of Hive

Meta store: Meta store is where the schemas of the Hive tables are stored, it stores the information about the tables and partitions that are in the warehouse.

SerDe: Serializer, Deserializer which gives instructions to hive on how to process records

Thrift

A thrift service is used to provide remote access from other processors

Meta Store

This is a service which stores the metadata information such as table schemas

Indexes

Indexes are created to the speedy access to columns in the database

Syntax: Create index <INDEX_NAME> on table <TABLE_NAME>

Hive Function Meta Commands

Show functions: Lists Hive functions and operators

Describe function [function name]: Displays short description of the particular function

Describe function extended [function name]: Displays extended description of the particular function

Hive Functions

- UDF(User defined Functions): It is a function that fetches one or more columns from a row as arguments and returns a single value
- UDTF(User defined Tabular Functions): This function is used to produce multiple columns or rows of output by taking zero or more inputs
- Macros: It is a function that uses other Hive functions
- User defined aggregate functions: A user defined function that takes multiple rows or columns and returns the aggregation of the data
- User defined table generating functions: A function which takes a column from single record and splitting it into multiple rows

Hive SELECT Command

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number]
;
```

- Select: Select is a projection operator in HiveQL, which scans the table specified by the FROM clause
- Where: Where is a condition which specifies what to filter
- Group by: It uses the list of columns, which specifies how to aggregate the records
- Cluster by, Distribute by, Sort by: Specifies the algorithm to sort, distribute and create cluster, and the order for sorting
- Limit: This specifies how many records to be retrieved

Hive Data Types

Integral data types:

- Tinyint
- Smallint
- Int
- Bigint

String types:

- VARCHAR-Length(1 to 65355)
- CHAR-Length(255)

Union type: It is a collection of heterogenous data types.

- Syntax: UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

Timestamp: It supports the traditional Unix timestamp with optional nanosecond precision

- Dates
- Decimals

Complex types:

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

Bucketing

It is a technique to decompose the datasets into more manageable parts

Partitioner

Partitioner controls the partitioning of keys of the intermediate map outputs, typically by a hash function which is same as the number of reduce tasks for a job

- Partitioning: It is used for distributing load horizontally. It is a way of dividing the tables into related parts based on values such as date, city, departments etc.

Hcatalog

It is a metadata and table management system for Hadoop platform which enables storage of data in any format.

Hive commands in HQL

Data Definition Language(DDL): It is used to build or modify tables and objects stored in a database. Some of the DDL commands are as follows:

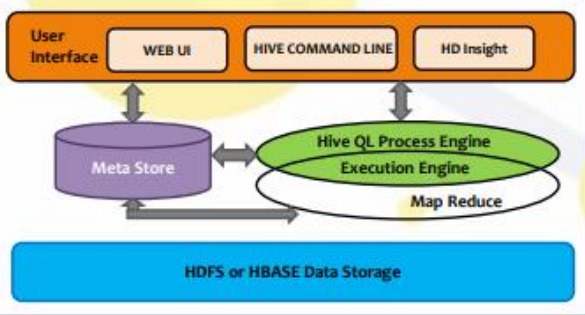
- To create database in Hive: create database<data base name>
- To list out the databases created in a Hive warehouse: show databases
- To use the database created: USE <data base name>
- To describe the associated database in metadata: describe<data base name>
- To alter the database created: alter<data base name>

Data Manipulation Language(DML): These statements are used to retrieve, store, modify, delete, insert and update data in a database

- Inserting data in a database: The Load function is used to move the data into a particular Hive table.

  LOAD data <LOCAL> inpath <file path> into table [tablename]
- Drop table: The drop table statements deletes the data and metadata from the table: drop table<table name>
- Aggregation: It is used to count different categories from the table :

  Select count (DISTINCT category) from tablename;
- Grouping: Group command is used to group the result set, where the result of one table is stored in the other: Select <category>, sum(amount) from <txt records> group by <category>
- To exit from the Hive shell: Use the command quit



Operations - Performed on Hive

Function	HQL Query
To retrieve information	SELECT from_columns FROM table WHERE conditions;
To select all values	SELECT * FROM table;
To select a particular category values	SELECT * FROM table WHERE rec_name = "value";
To select for multiple criteria	SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";
For selecting specific columns	SELECT column_name FROM table;
To retrieve unique output records	SELECT DISTINCT column_name FROM table;
For sorting	SELECT col1, col2 FROM table ORDER BY col2;
For sorting backwards	SELECT col1, col2 FROM table ORDER BY col2 DESC;
For counting rows from the table	SELECT COUNT(*) FROM table;
For grouping along with counting	SELECT owner, COUNT(*) FROM table GROUP BY owner;
For selecting maximum values	SELECT owner, COUNT(*) FROM table GROUP BY owner;
Selecting from multiple tables and joining	SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name);

Command Line Statements

Function	Hive Commands
To run the query	hive -e 'select a.col from tab1 a'
To run a query in a silent mode	hive -S -e 'select a.col from tab1 a'
To select hive configuration variables	hive -e 'select a.col from tab1 a' –hiveconf hive.root.logger=DEBUG,console
To use the initialization script	hive -i initialize.sql
To run the non-interactive script	hive -f script.sql
To run script inside the shell	source file_name
To run the list command	dfs –ls /user
To run ls (bash command) from the shell	!ls
To set configuration variables	set mapred.reduce.tasks=32
Tab auto completion	set hive.<TAB>
To display all variables starting with hive	set
To revert all variables	reset
To add jar files to distributed cache	add jar jar_path
To display all the jars in the distributed cache	list jars
To delete jars from the distributed cache	delete jar jar_name

Metadata Functions and Query

Function	Hive Commands
Selecting a database	USE database;
Listing databases	SHOW DATABASES;
listing table in a database	SHOW TABLES;
Describing format of a table	DESCRIBE (FORMATTED|EXTENDED) table;
Creating a database	CREATE DATABASE db_name;
Dropping a database	DROP DATABASE db_name (CASCADE);