Apache Hive

Apache Hive

- Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis.
- Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.
- Initial release date: 1 October 2010
- License: Apache License 2.0
- Preview release: 4.0.0-alpha-2 / November 16, 2022
- Stable release: 3.1.3 / April 8, 2022
- Programming language: Java
- Developer: Meta, Apache Software Foundation

Apache Hive

- Apache Hive is a data ware house system for Hadoop that runs SQL like queries called HQL (Hive query language) which gets internally converted to map reduce jobs.
- Hive was developed by Facebook.
- It supports Data definition Language, Data Manipulation Language and user defined functions.
- Our Hive tutorial includes all topics of Apache Hive with Hive Installation, Hive Data Types, Hive Table partitioning, Hive DDL commands, Hive DML commands, Hive sort by vs order by, Hive Joining tables etc.

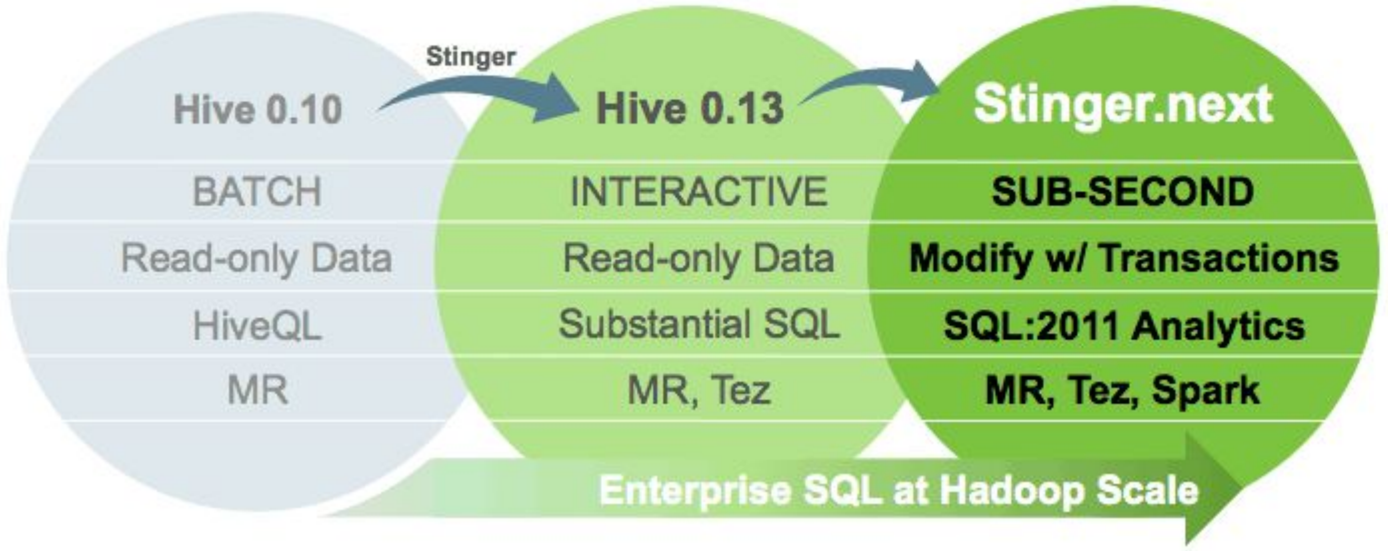
What is HIVE ?

- Data warehousing package built on top of Hadoop
- Used for data analysis
- Targeted towards users comfortable with SQL
- Similar to SQL the query language is HiveQL
- No need learn Java and Hadoop APIs
- Developed by Facebook and made open source
- For managing and querying structured data

Hadoop History

- **Jan 2006 – Doug Cutting joins Yahoo**
- Feb 2006 – Hadoop splits out of Nutch and Yahoo starts using it.
- **Dec 2006 – Yahoo creating 100-node Webmap with Hadoop**
- Apr 2007 – Yahoo on 1000-node cluster
- Jan 2008 – Hadoop made a top-level Apache project
- **Dec 2007 – Yahoo creating 1000-node Webmap with Hadoop**
- Sep 2008 – Hive added to Hadoop as a contrib project

Recent releases of Hive



Features of Hive

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

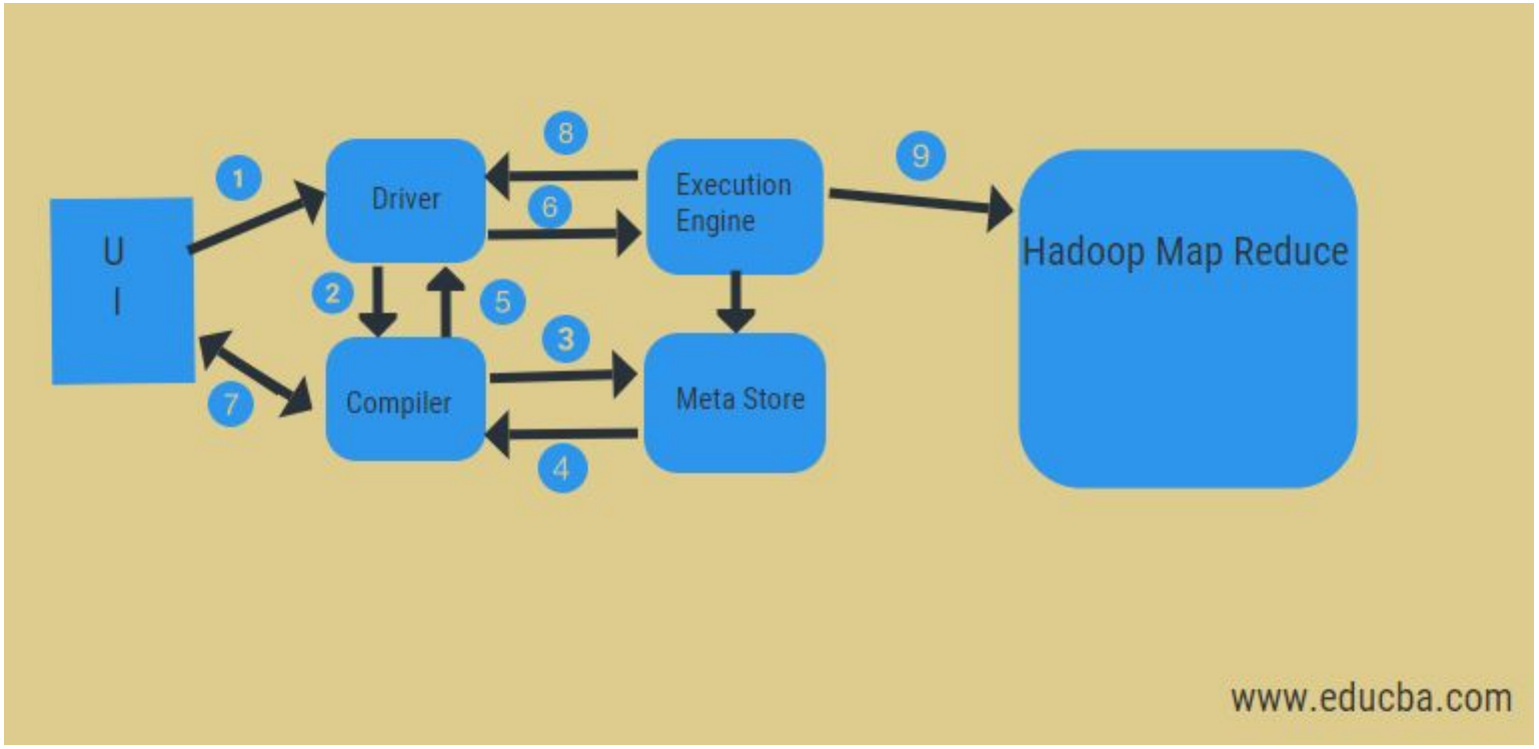
Limitations of Hive

- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

Hive integration and workflow

- **The steps include:**
- execute the Query from UI.
- get a plan from the driver tasks DAG stages.
- get metadata request from the meta store.
- send metadata from the compiler.
- sending the plan back to the driver.
- Execute plan in the execution engine.
- fetching results for the appropriate user query.
- sending results bi-directionally.

Hive integration and workflow



Hive Metastore

- Stores Hive metadata
- Default metastore database uses Apache Derby
- Various configurations:
  - Embedded (in-process metastore, in-process database)
    - Mainly for unit tests
  - Local (in-process metastore, out-of-process database)
    - Each Hive client connects to the metastore directly
  - Remote (out-of-process metastore, out-of-process database)
    - Each Hive client connects to a metastore server, which connects to the metadata database itself

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

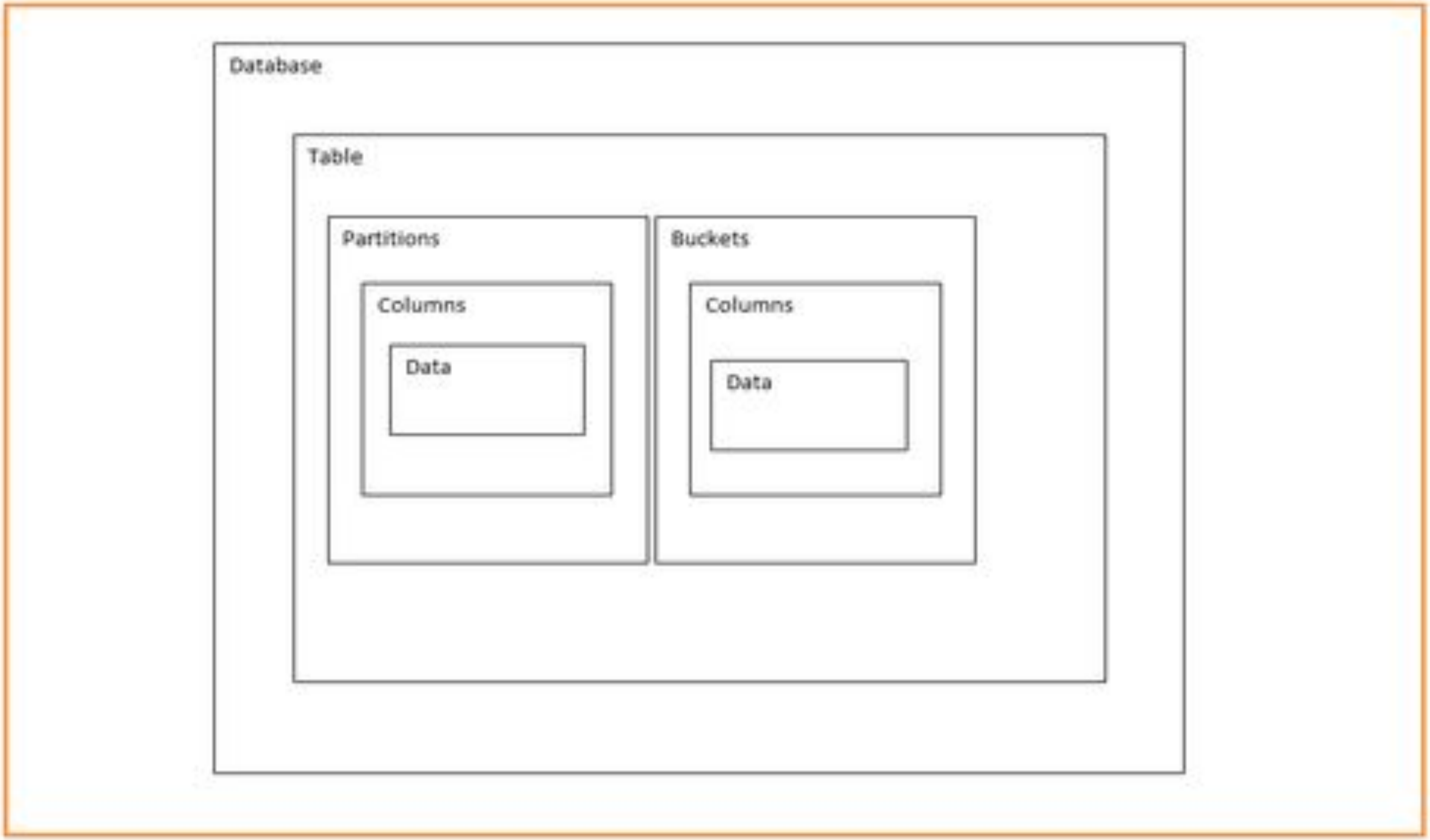
Hive Schemas

- 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

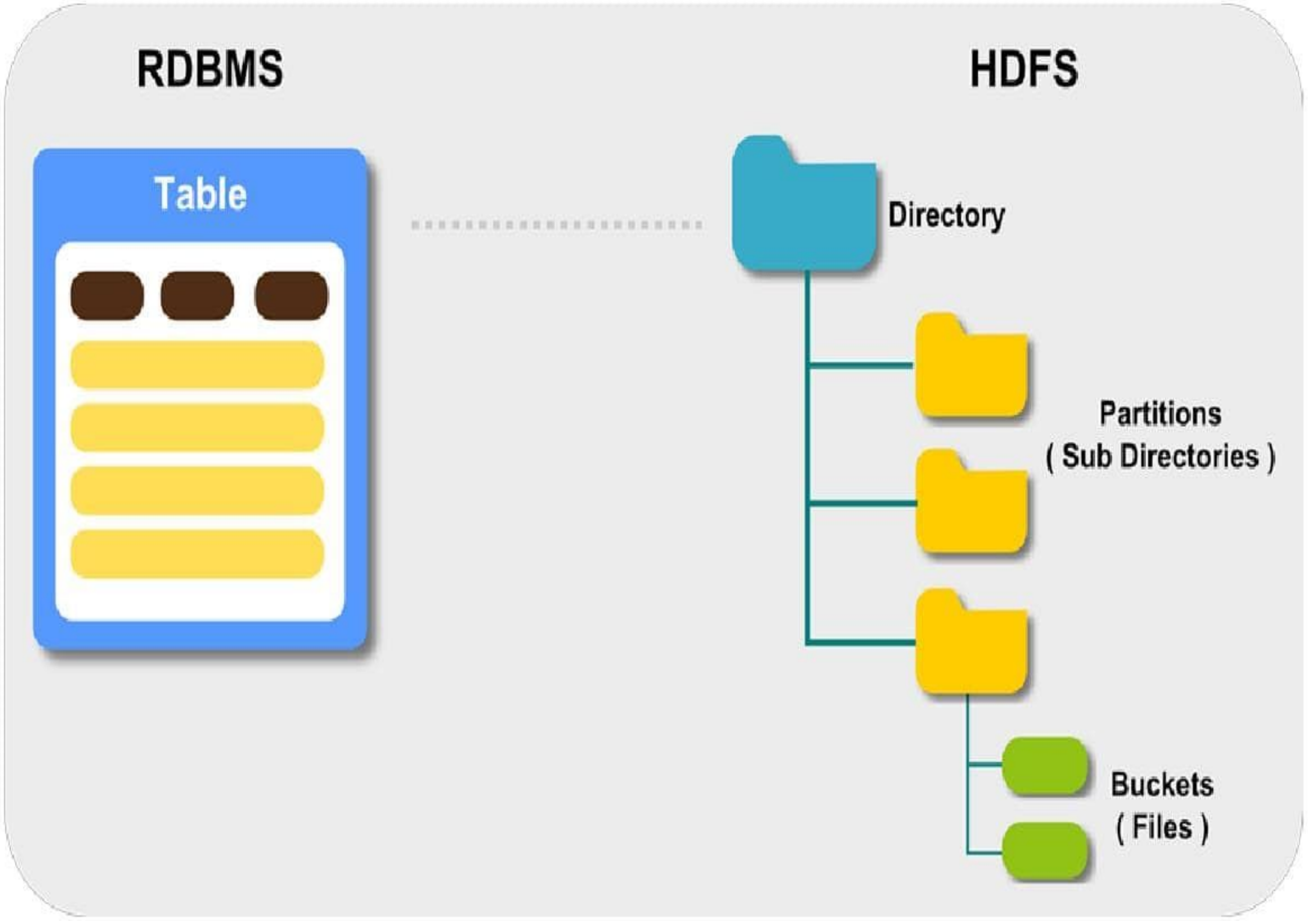
Hive Data Units

- Hive data is organized into: Databases: Namespaces function to avoid naming conflicts for tables, views, partitions, columns, and so on.
- Databases can also be used to enforce security for a user or group of users.
- Tables: Homogeneous units of data which have the same schema.

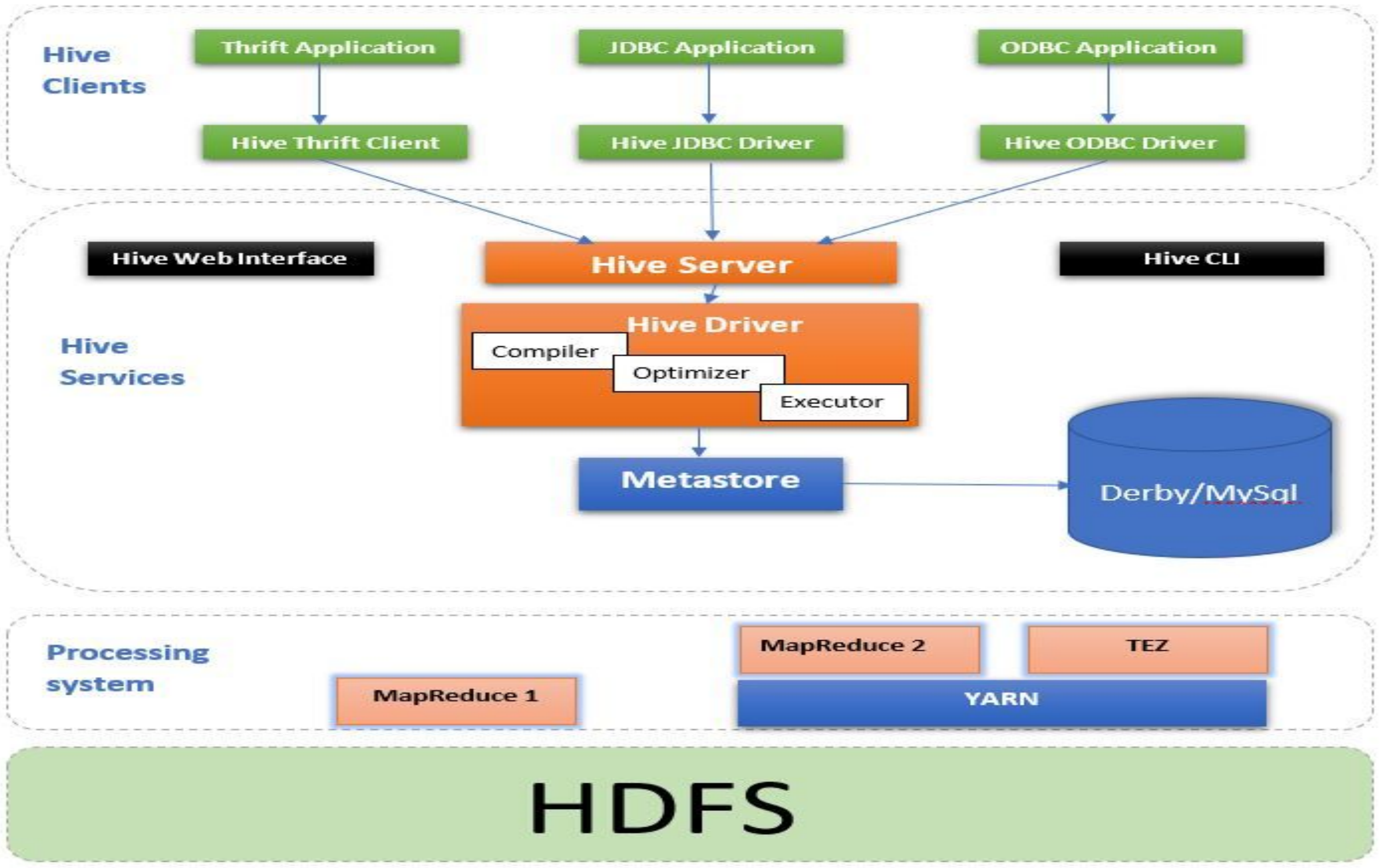
Hive Data Units



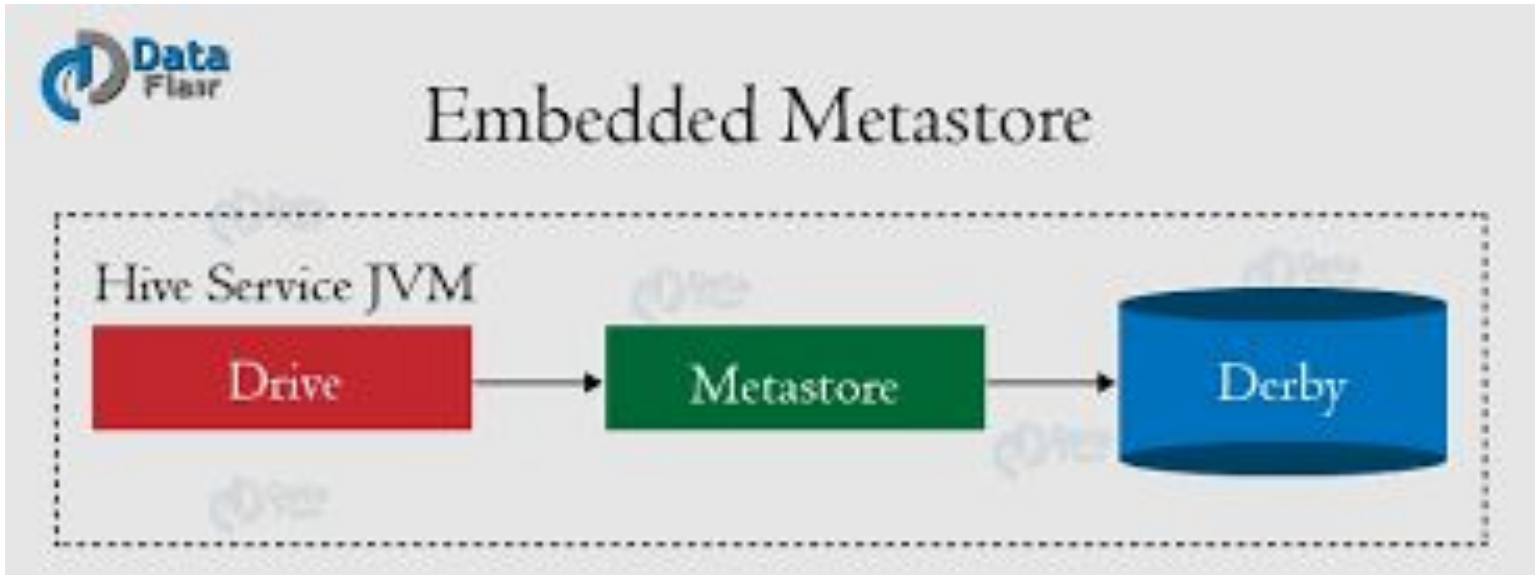
Hive Data Model



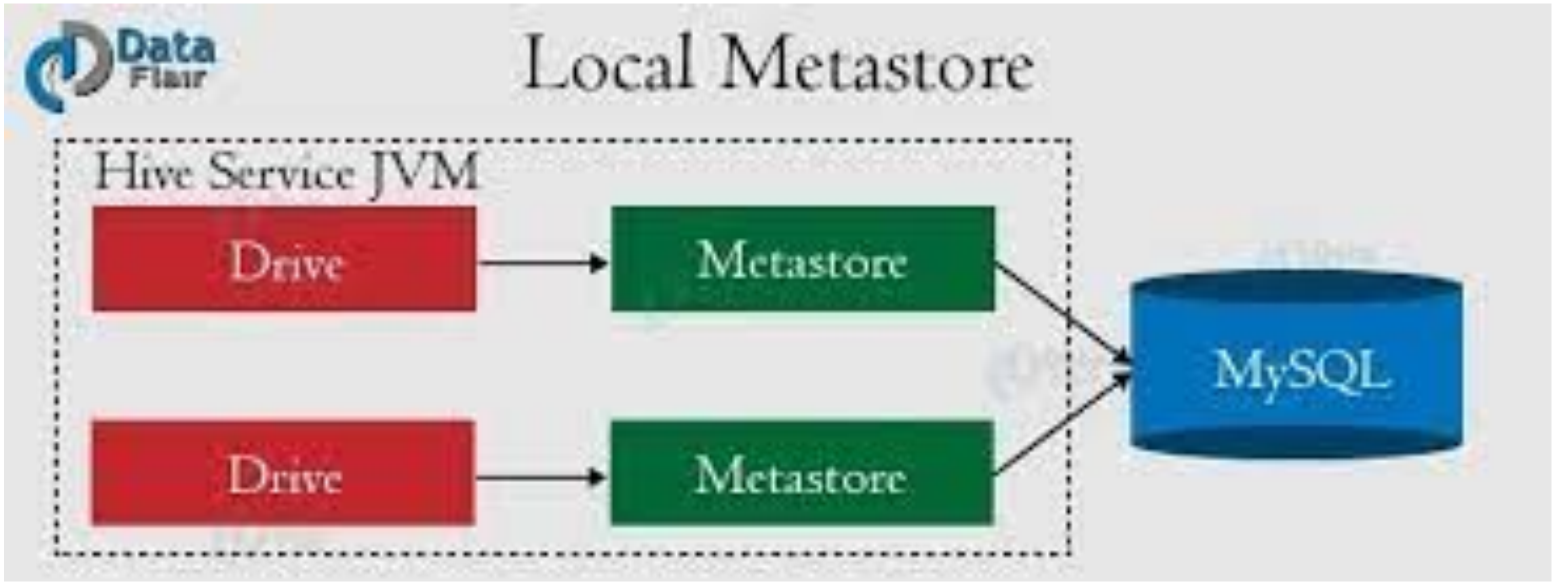
Hive Architecture



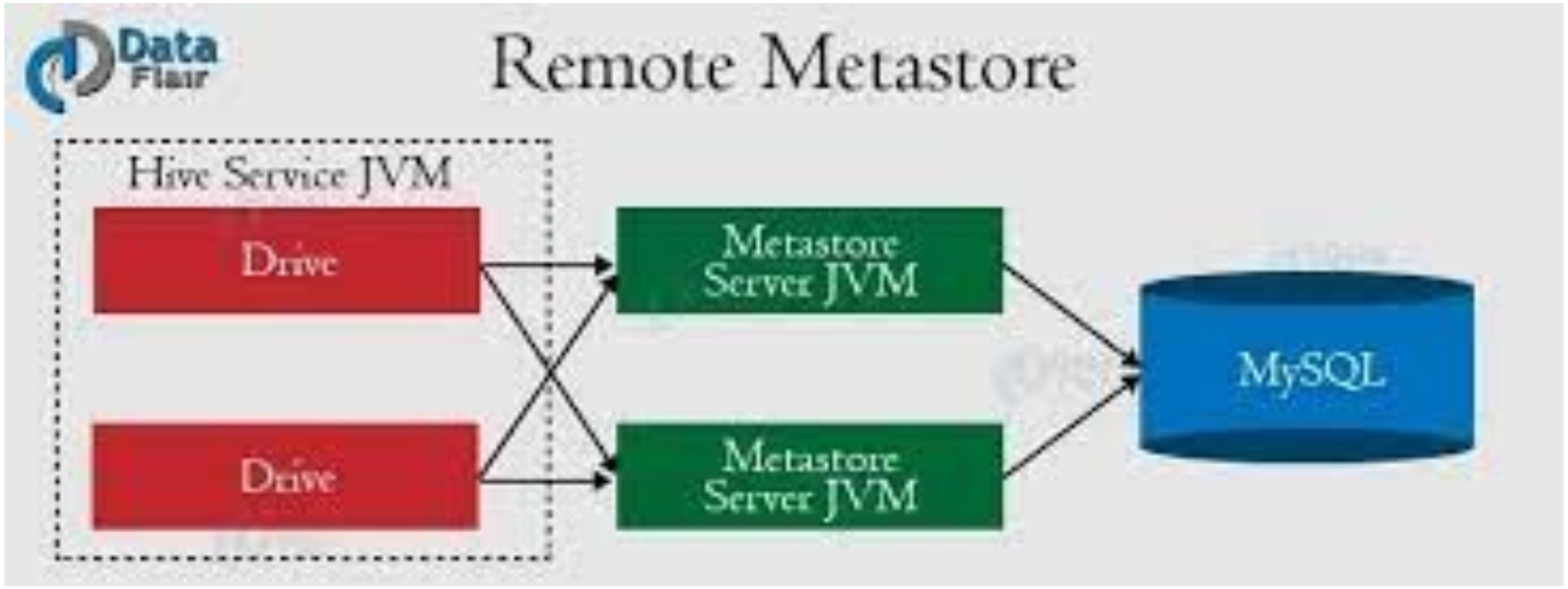
Embedded Metastore



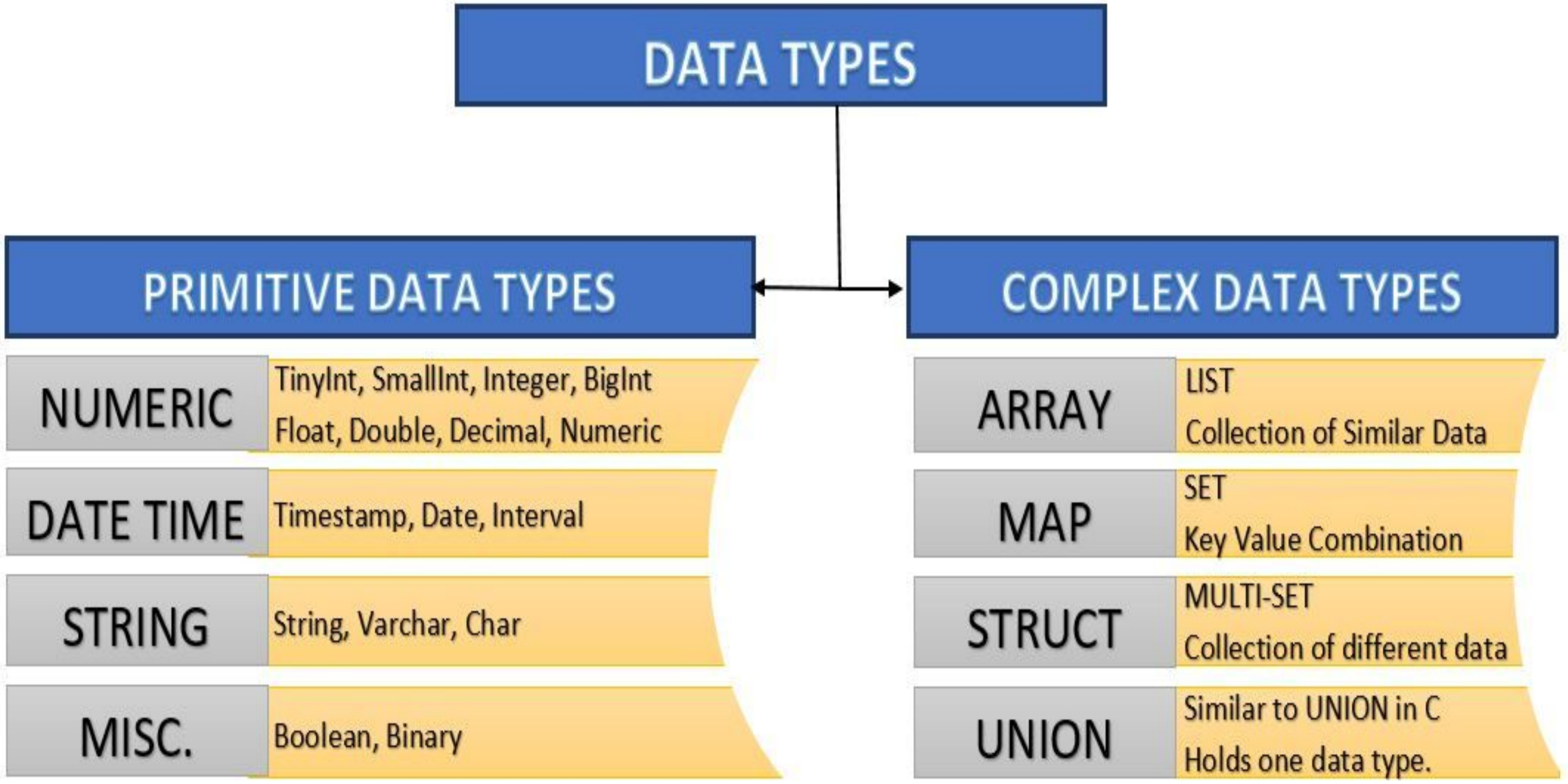
Local Metastore



Remote Metastore



Hive Data Types – Primitive & Collection



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'

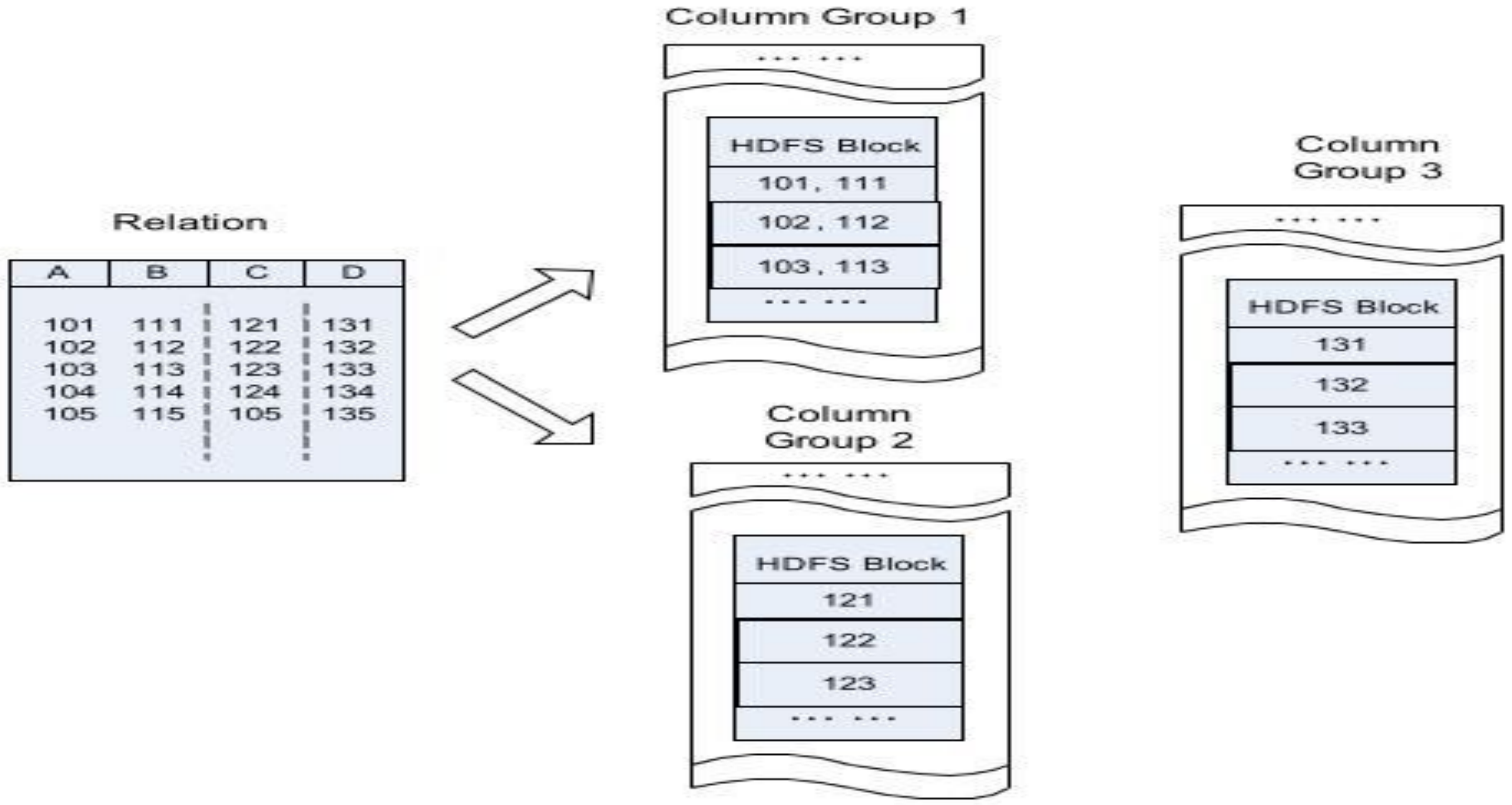
Hive File Format

- Hive facilitates managing large data sets supporting multiple data formats.
- Hive supports several file formats:
  - Text File, Sequential File and RCFile.
- By Default its **TextFileFormat** for hive unless you use serdes to change it.

Text File, Sequential File and RCFile

- Text File :
- Hive Textfile format supports comma-, tab-, and space-separated values, as well as data specified in JSON notation.
- The DELIMITED FIELDS TERMINATED BY subclause identifies the field delimiter within a data record (line).
- The sales_info table field delimiter is a comma (,).
- Sequential File :
- It is a flat file that store binary key-value pairs.
- It includes compression support which reduce the CPU, I/O requirements.
- RCFiles(Record Columnar File) :
- RCFile stores columns of a table in a record columnar way.

RCFile Example



HiveQL

- HiveQL / HQL provides the basic SQL-like operations:
  - Select columns using SELECT
  - Filter rows using WHERE
  - JOIN between tables
  - Evaluate aggregates using GROUP BY
  - Store query results into another table
  - Download results to a local directory (i.e., export from HDFS)
  - Manage tables and queries with CREATE, DROP, and ALTER

HiveQL Limitations

- HQL only supports equi-joins, outer joins, left semi-joins
- Because it is only a shell for mapreduce, complex queries can be hard to optimise
- Missing large parts of full SQL specification:
  - HAVING clause in SELECT
  - Correlated sub-queries
  - Sub-queries outside FROM clauses
  - Updatable or materialized views
  - Stored procedures

HIVE Query Language(HQL)

- HQL provide basic SQL like operations.
  - Create and manage tables and partitions.
  - Support various Relational, Arithmetic and Logical Operations.
  - Evaluate functions.
  - Download the contents of a table to a local directory or result of queries to HDFS directory.
  - 1. DDL 2. DML 3. Starting Hive Shell
    4. Database 5. Tables 6. Partitions 7. Bucketing
    8. Views 9. Sub-Query 10. Joins
    11. Aggregation 12. Group By and Having

Data Hierarchy

- Hive is organised hierarchically into:
  - Databases: namespaces that separate tables and other objects
  - Tables: homogeneous units of data with the same schema
    - Analogous to tables in an RDBMS
  - Partitions: determine how the data is stored
    - Allow efficient access to subsets of the data
  - Buckets/clusters
    - For subsampling within a partition
    - Join optimization

Hive Data Model Contd.

- Tables

- Analogous to relational tables
- Each table has a corresponding directory in HDFS
- Data serialized and stored as files within that directory
- Hive has default serialization built in which supports compression and lazy deserialization
- Users can specify custom serialization –deserialization schemes (**SerDe’s**)

Hive Data Model Contd.

- Partitions
  - Each table can be broken into partitions
  - Partitions determine distribution of data within subdirectories

Example -

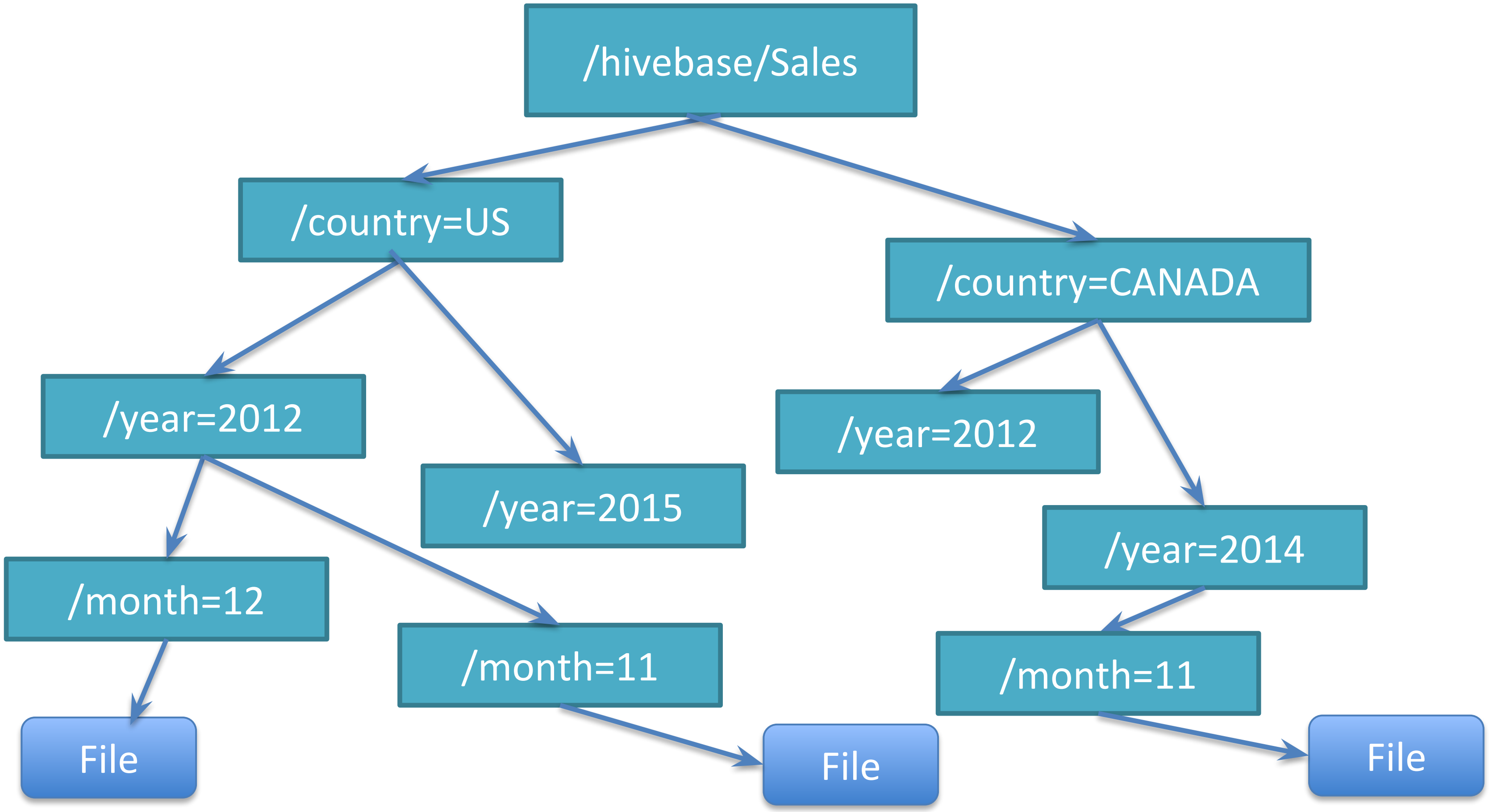
CREATE_TABLE Sales (sale_id INT, amount FLOAT)

PARTITIONED BY (country STRING, year INT, month INT)

So each partition will be split out into different folders like

Sales/country=US/year=2012/month=12

Hierarchy of Hive Partitions



Hive Data Model Contd.

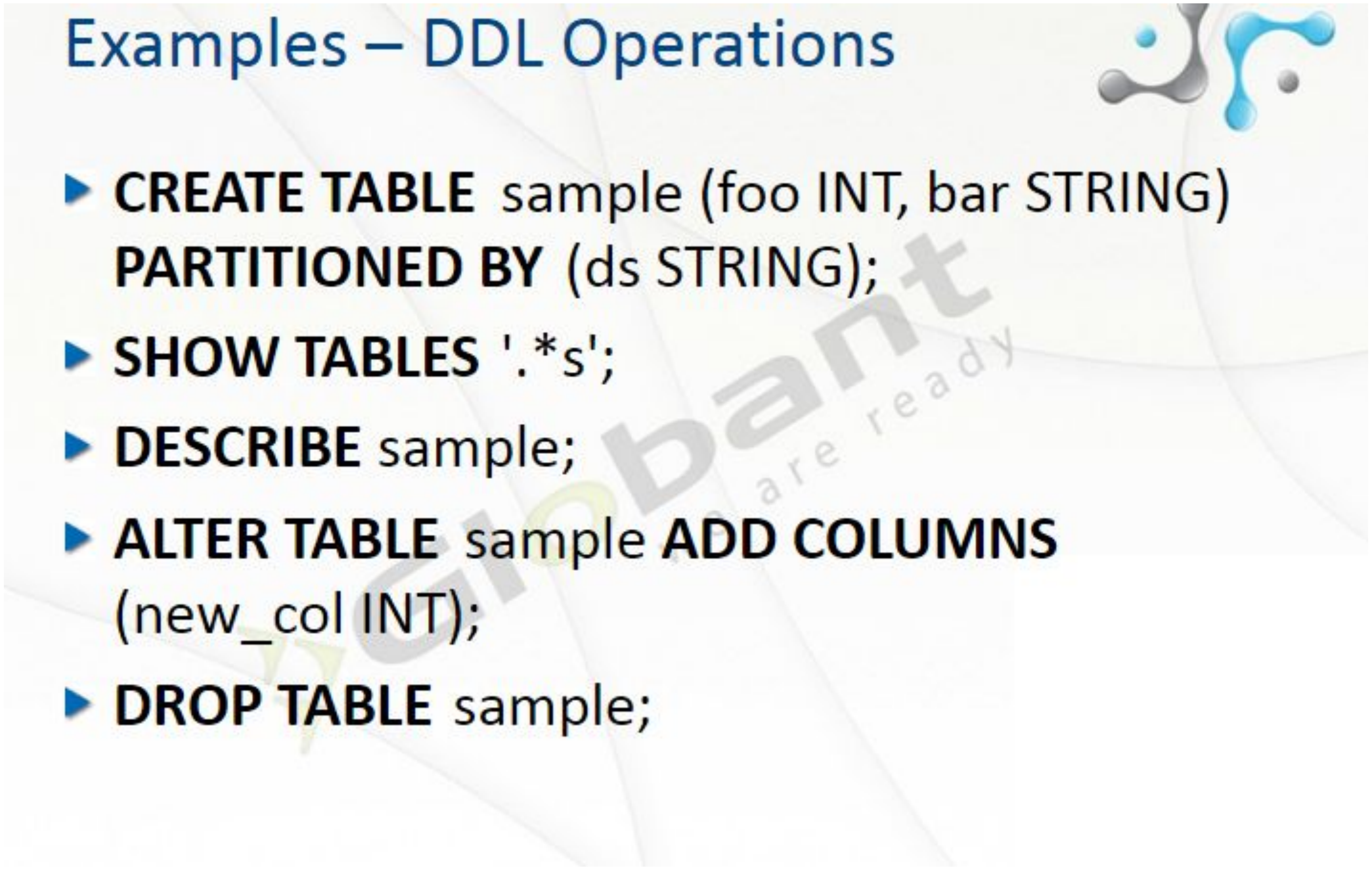
- Buckets

- Data in each partition divided into buckets
- Based on a hash function of the column
- **H(column) mod NumBuckets = bucket number**
- Each bucket is stored as a file in partition directory

Data Definition Language

- These statements are used to build and modify the tables and other objects in the database.
- Commands are,
  - Create /Drop / Alter Database
  - Create /Drop / Truncate table
  - Alter Table / Partition / Column
  - Create / Drop / Alter View
  - Create / Drop / Alter View
  - Show
  - Describe

HiveQL



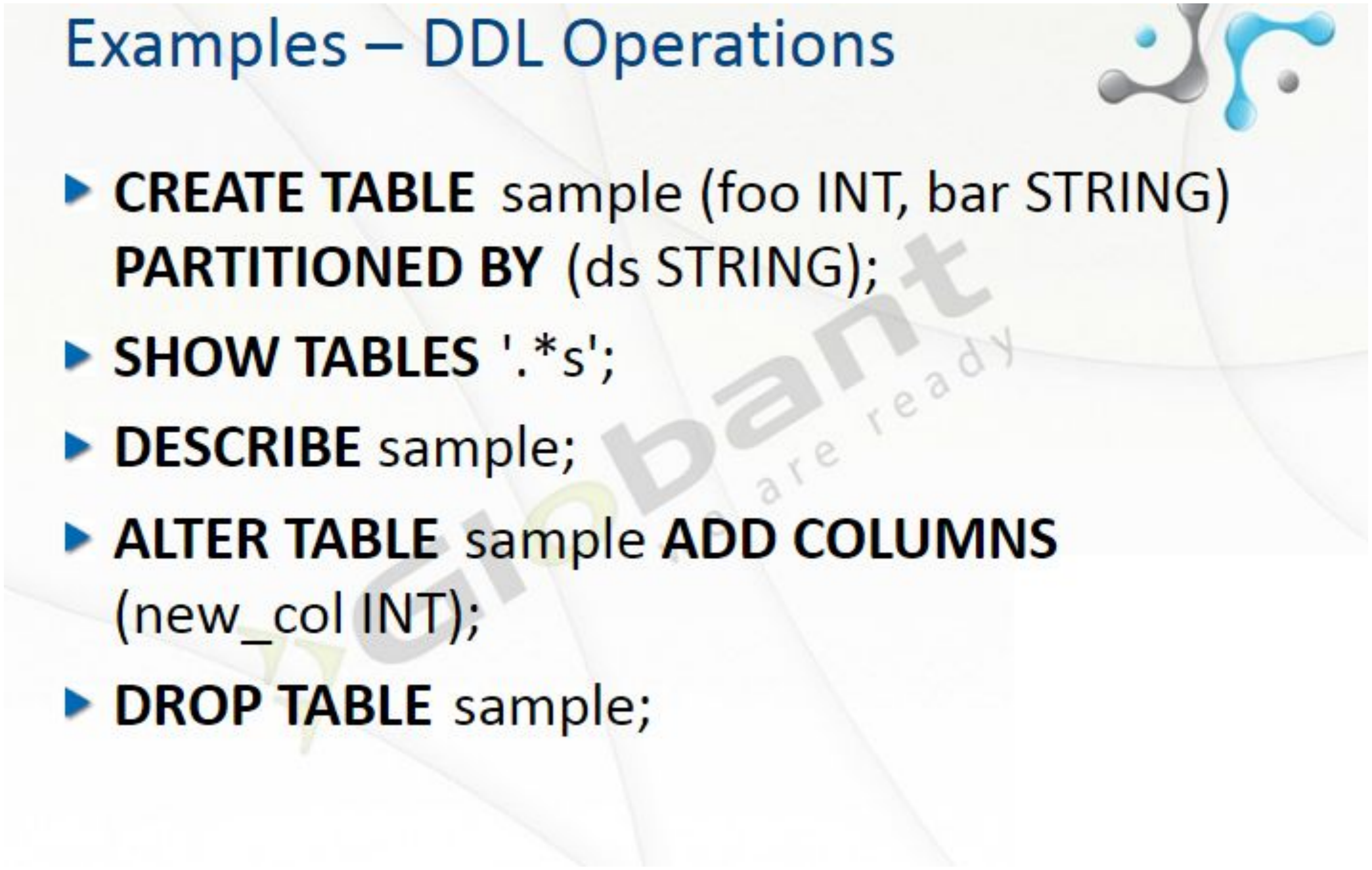
Examples – DDL Operations

- **CREATE TABLE** sample (foo INT, bar STRING) **PARTITIONED BY** (ds STRING);
- **SHOW TABLES** '.*s';
- **DESCRIBE** sample;
- **ALTER TABLE** sample **ADD COLUMNS** (new_col INT);
- **DROP TABLE** sample;

Data Manipulation Language

- DML statements are used to remove, store, modify, delete and update data to database.
- The DML commands are,
  - Loading files into table
  - Inserting data into Hive Tables from queries

HiveQL



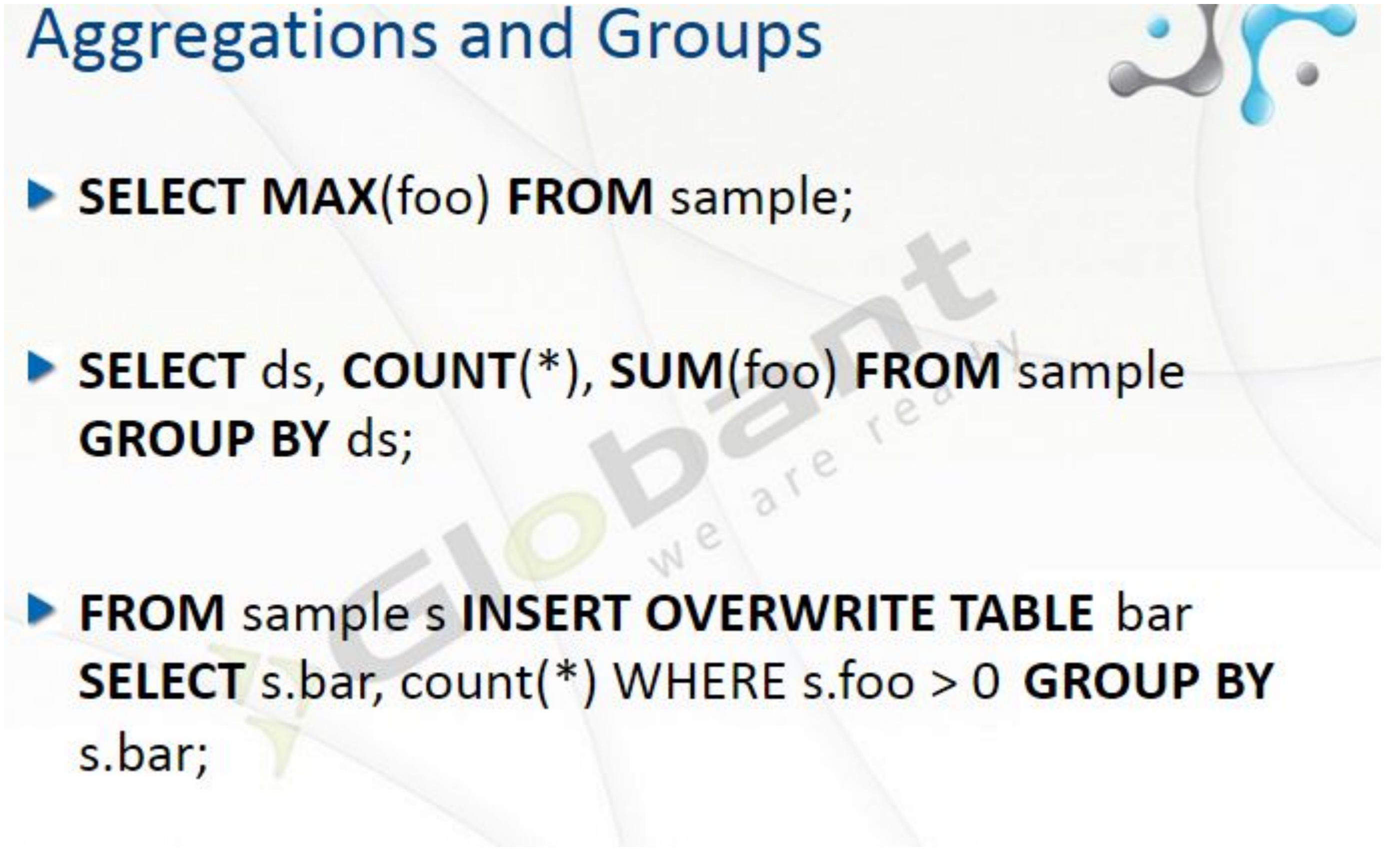
Examples – DDL Operations

- **CREATE TABLE** sample (foo INT, bar STRING) **PARTITIONED BY** (ds STRING);
- **SHOW TABLES** '.*s';
- **DESCRIBE** sample;
- **ALTER TABLE** sample **ADD COLUMNS** (new_col INT);
- **DROP TABLE** sample;

SELECTS and FILTERS

- **SELECT** foo **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE DIRECTORY** '/tmp/hdfs_out' **SELECT** * **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE LOCAL DIRECTORY** '/tmp/hive-sample-out' **SELECT** * **FROM** sample;

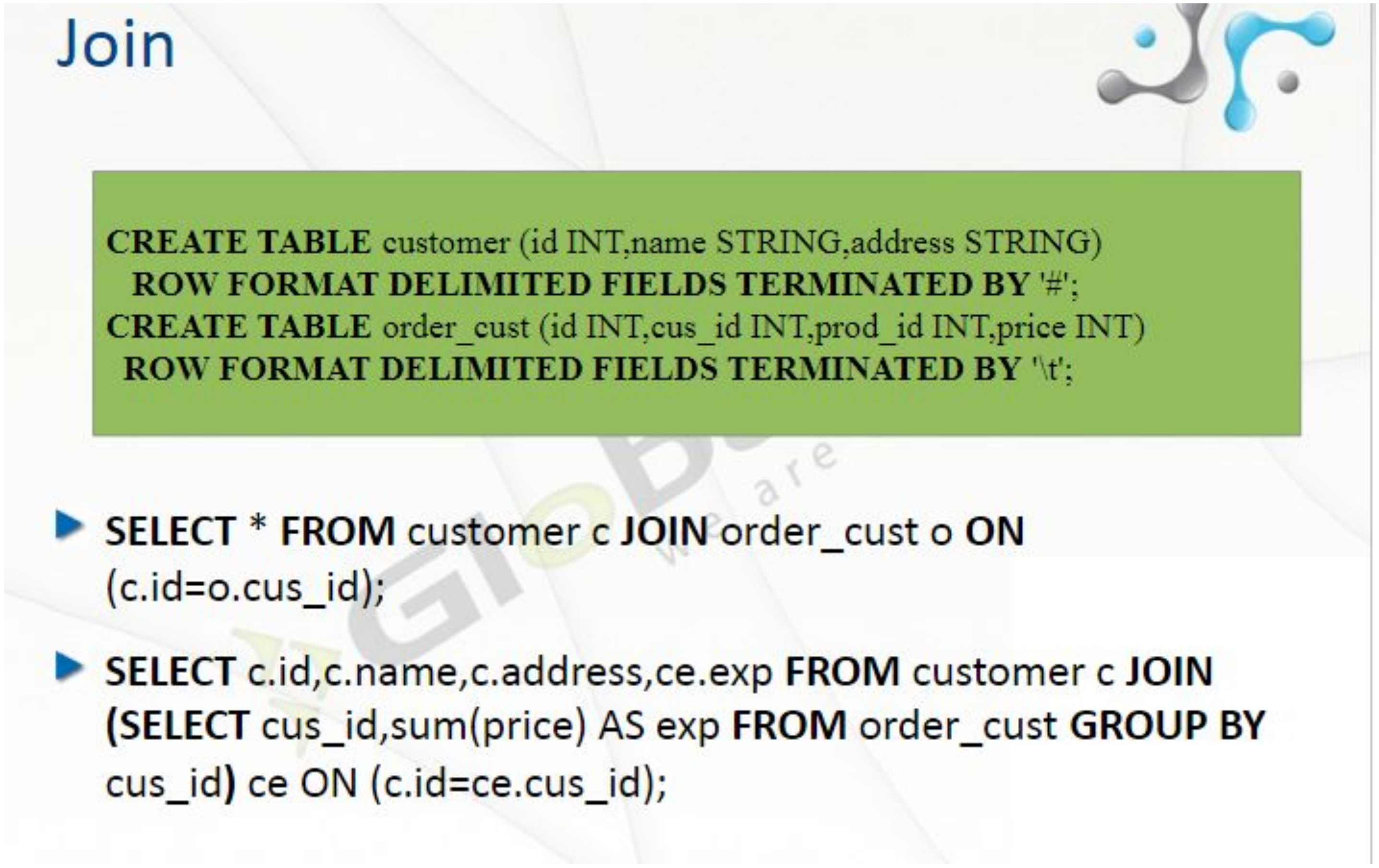
Aggregate Function



Aggregations and Groups

- **SELECT MAX**(foo) **FROM** sample;
- **SELECT** ds, **COUNT**(*), **SUM**(foo) **FROM** sample **GROUP BY** ds;
- **FROM** sample s **INSERT OVERWRITE TABLE** bar **SELECT** s.bar, count(*) WHERE s.foo > 0 **GROUP BY** s.bar;

Join

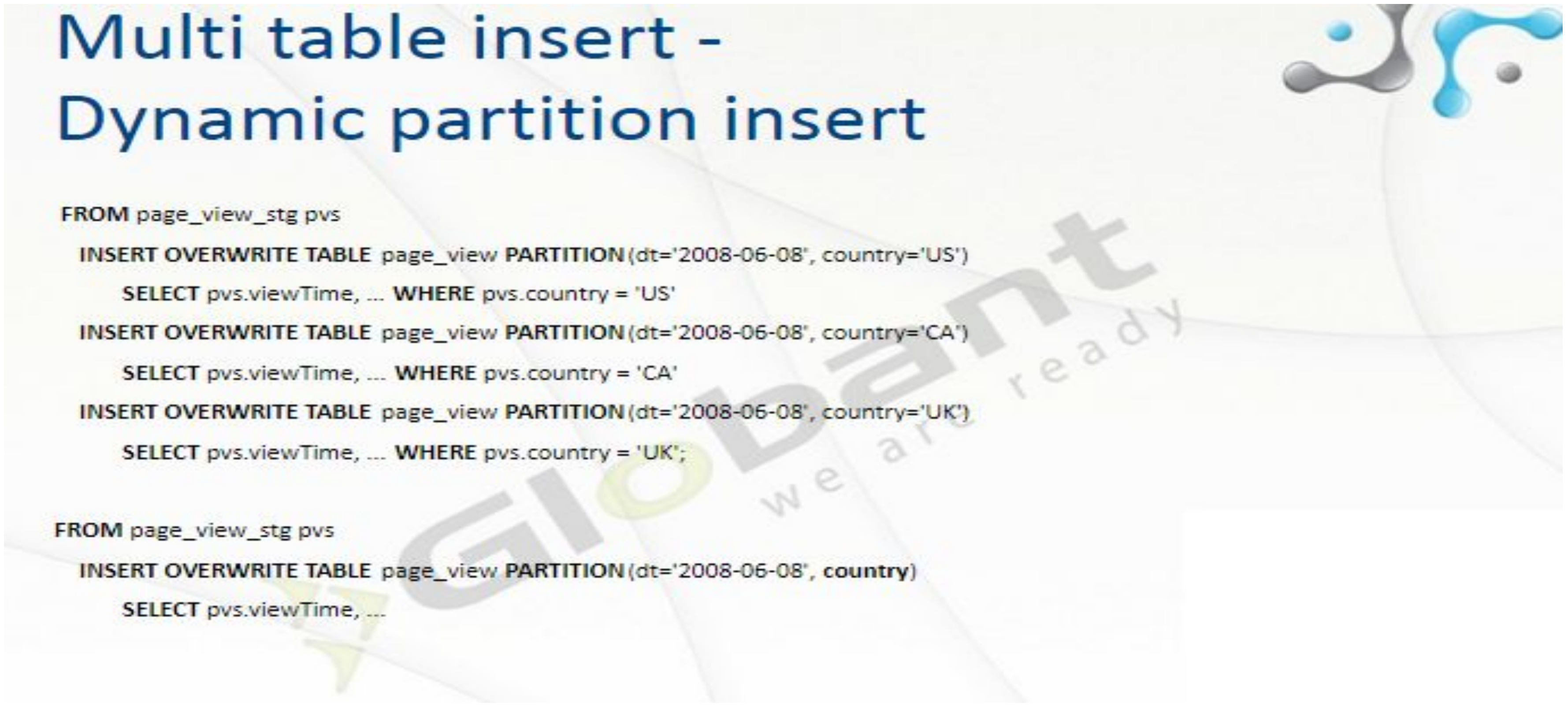


Join

```
CREATE TABLE customer (id INT,name STRING,address STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '#';
CREATE TABLE order_cust (id INT,cus_id INT,prod_id INT,price INT)
 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- **SELECT** * **FROM** customer c **JOIN** order_cust o **ON** (c.id=o.cus_id);
- **SELECT** c.id,c.name,c.address,ce.exp **FROM** customer c **JOIN** **(SELECT** cus_id,sum(price) AS exp **FROM** order_cust **GROUP BY** cus_id**)** ce ON (c.id=ce.cus_id);

Multi-



Multi table insert - Dynamic partition insert

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

FROM page_view_stg pvs
  INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country)
      SELECT pvs.viewTime, …
```

Tables – Managed and External

```
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;
```

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';
```

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

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

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

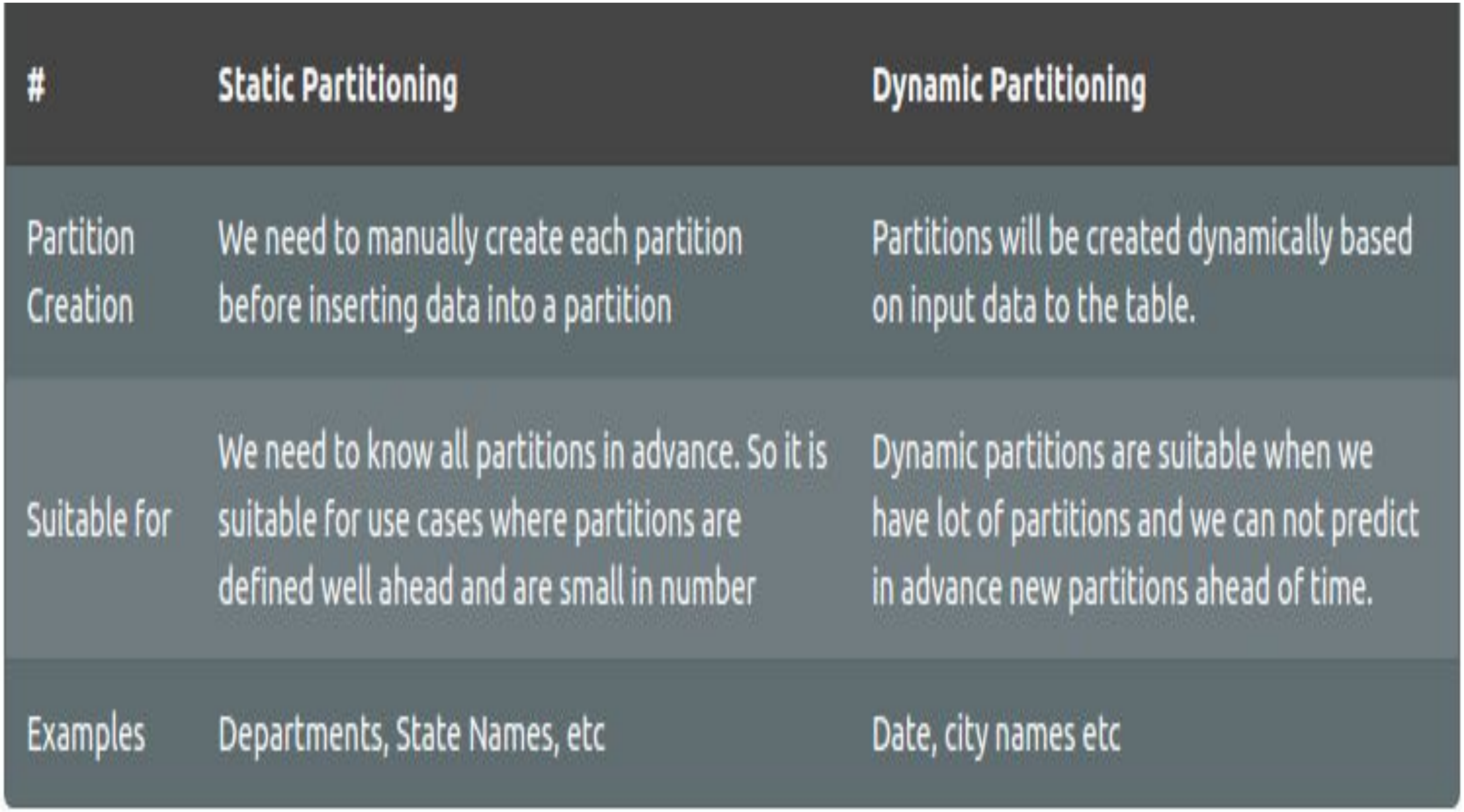
Browsing Tables And Partitions

Command	Comments
`SHOW TABLES;`	Show all the tables in the database
`SHOW TABLES 'page.*';`	Show tables matching the specification
`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

Types of Partitions

- Partition is classified into two,
  - Static and Dynamic
- Static Partition
  - Static Partition comprise columns whose values are known at compile time.
  - In static or manual partitioning, **it is required to pass the values of partitioned columns manually while loading the data into the table**.
  - Hence, the data file doesn't contain the partitioned columns. Example of Static Partitioning. First, select the database in which we want to create a table.
- Dynamic Partition
  - Dynamic Partition have columns whose values are known only at Execution time.
  - **Single insert to partition table** is known as a dynamic partition.
  - Usually, dynamic partition loads the data from the non-partitioned table.
  - Dynamic Partition takes more time in loading data compared to static partition.

Difference between Partitions



#	Static Partitioning	Dynamic Partitioning
Partition Creation	We need to manually create each partition before inserting data into a partition	Partitions will be created dynamically based on input data to the table.
Suitable for	We need to know all partitions in advance. So it is suitable for use cases where partitions are defined well ahead and are small in number	Dynamic partitions are suitable when we have lot of partitions and we can not predict in advance new partitions ahead of time.
Examples	Departments, State Names, etc	Date, city names etc

Create Partitions in Hive

- To create data partitioning in Hive following command is used-
  CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY (partition1 data_type, partition2 data_type,….);

Hive Data Partitioning Example

- The file name says file1 contains client data table:
- tab1/clientdata/file1
  id, name, dept, yoj
  1, sunny, SC, 2009
  2, animesh, HR, 2009
  3, sumeer, SC, 2010
  4, sarthak, TP, 2010[/php]
  Now, let us partition above data into two files using years
  tab1/clientdata/2009/file2
  1, sunny, SC, 2009
  2, animesh, HR, 2009
  tab1/clientdata/2010/file3
  3, sumeer, SC, 2010
  4, sarthak, TP, 2010

Hive Data Partitioning Example

- Now when we are retrieving the data from the table, only the data of the specified partition will be queried. Creating a partitioned table is as follows:
- CREATE TABLE table_tab1 (id INT, name STRING, dept STRING, yoj INT) PARTITIONED BY (year STRING);
  LOAD DATA LOCAL
  INPATH tab1'/clientdata/2009/file2'OVERWRITE INTO TABLE studentTab PARTITION (year='2009');
  LOAD DATA LOCAL
  INPATH tab1'/clientdata/2010/file3'OVERWRITE INTO TABLE studentTab PARTITION (year='2010');

Hadoop Streaming

- **Allow to write Map and Reduce functions in any languages**
  - Hadoop Map/Reduce only accepts Java

- **Example: Word Count**

  -
    ```
    hadoop streaming
    -input /user/zshao/articles
    -mapper ‘tr “ ” “\n”’
    -reducer ‘uniq -c‘
    -output /user/zshao/
    -numReduceTasks 32
    ```

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
      ```

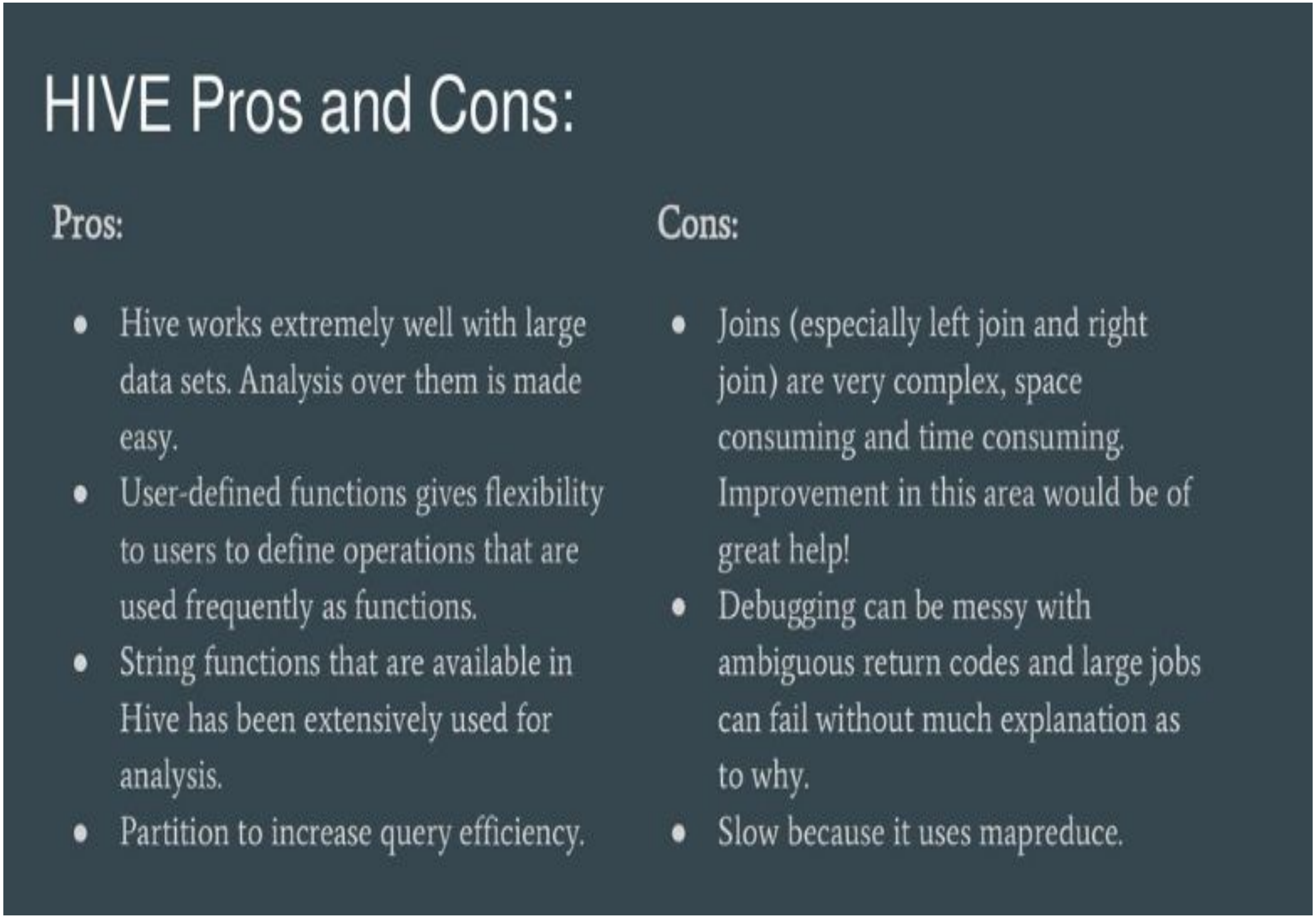
  - If multiple tables are joined, put the biggest table last and the reducer will stream the last table, buffer the others
  - Use left semi-joins to take the place of IN/EXISTS

```
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 Pros and Cons:

Pros:

- Hive works extremely well with large data sets. Analysis over them is made easy.
- User-defined functions gives flexibility to users to define operations that are used frequently as functions.
- String functions that are available in Hive has been extensively used for analysis.
- Partition to increase query efficiency.

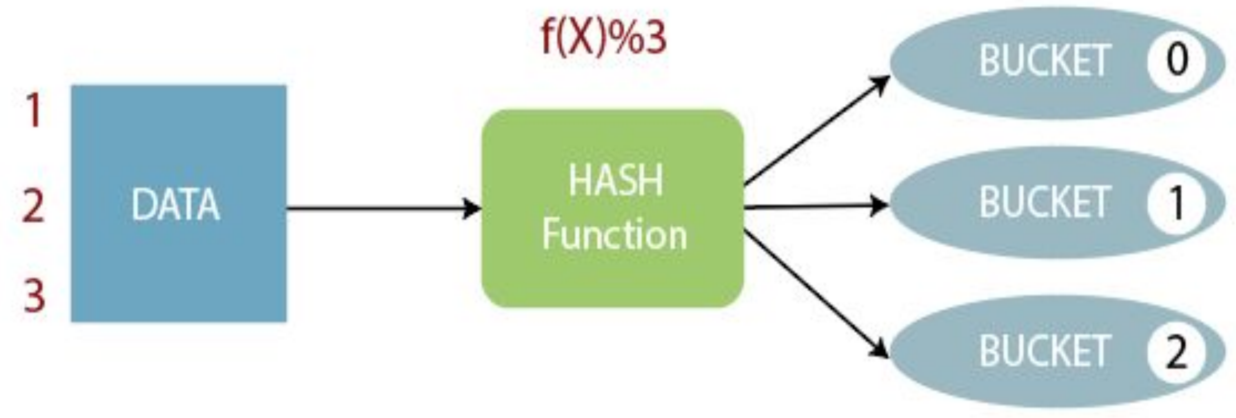
Cons:

- Joins (especially left join and right join) are very complex, space consuming and time consuming. Improvement in this area would be of great help!
- Debugging can be messy with ambiguous return codes and large jobs can fail without much explanation as to why.
- Slow because it uses mapreduce.

Bucketing

- Bucketing is a method in Hive which is used for organizing the data.
- It is a concept of separating data into ranges known as buckets.
- Bucketing in hives comes helpful when the use of partitioning becomes hard.
- A user can determine the range of a specific bucket by the hash value.
- **Partitioning helps in elimination of data, if used in WHERE clause, where as bucketing helps in organizing data in each partition into multiple files**, so as same set of data is always written in same bucket.

Working of Bucketing in Hive



- The concept of bucketing is based on the hashing technique.
- Here, modules of current column value and the number of required buckets is calculated (let say, F(x) % 3).
- Now, based on the resulted value, the data is stored into the corresponding bucket.

Example of Bucketing in Hive

- Create a bucketing table by using the following command: -
- hive**>** create table emp_bucket(Id int, Name string , Salary float)
- clustered by (Id) into 3 buckets
- row format delimited fields terminated by ',' ;

Views

- Views are **generated based on user requirements**.
- You can save any result set data as a view.
- The usage of view in Hive is same as that of the view in SQL.
- Creating a View :
- You can create a view at the time of executing a SELECT statement.
- The syntax is as follows:
- CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...)] [COMMENT table_comment] AS SELECT ...

Example

- hive> CREATE VIEW emp_30000 AS SELECT * FROM employee WHERE salary>30000;

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

Sub queries

- A Query present within a Query is known as a sub query. The main query will depend on the values returned by the subqueries.
- Subqueries can be classified into two types,
  – Subqueries in FROM clause
  – Subqueries in WHERE clause

Sub queries

- **Hive supports subqueries in FROM clauses and in WHERE clauses of SQL statements**.
- A subquery is a SQL expression that is evaluated and returns a result set. Then that result set is used to evaluate the parent query.
- The parent query is the outer query that contains the child subquery.
- A subquery is **a query that is nested inside a SELECT , INSERT , UPDATE , or DELETE statement, or inside another subquery**.
- A subquery can be used anywhere an expression is allowed.

Sub queries

- To get a particular value combined from two column values from different tables
- Dependency of one table values on other tables
- Comparative checking of one column values from other tables
- **Syntax:**
- Subquery in FROM clause SELECT <column names 1, 2…n>From (SubQuery) <TableName_Main > Subquery in WHERE clause SELECT <column names 1, 2…n> From<TableName_Main>WHERE col1 IN (SubQuery);

Example

- SELECT col1 FROM (SELECT a+b AS col1 FROM t1) t2
- Here t1 and t2 are table names.
- Here a and b are columns that are added in a subquery and assigned to col1.
- Col1 is the column value present in Main table.
- This column “col1” present in the subquery is equivalent to the main table query in column col1.

RCFiles Implementation

- RCFiles, short of Record Columnar File, are flat files consisting of binary key/value pairs, which shares much similarity with SequenceFile. RCFile stores columns of a table in a record columnar way.
- RCFile (Record Columnar File) is a data placement structure designed for MapReduce-based data warehouse systems.
- Hive added the RCFile format in version 0.6.0.
- It first partitions rows horizontally into row splits, and then it vertically partitions each row split in a columnar way.
- RCFile combines the advantages of both row-store and column-store to satisfy the need for fast data loading and query processing, efficient use of storage space, and adaptability to highly dynamic workload patterns.

SERDE

- Hive uses the SerDe interface for IO.
- The interface handles both serialization and deserialization and also interpreting the results of serialization as individual fields for processing.
- A SerDe allows Hive to read in data from a table, and write it back out to HDFS in any custom format.
- Contains the logic to convert unstructured data into records.
- Implemented using Java.
- Serializers are used at the time of writing.
- Deserializers are used at query time.

User-defined functions

- User Defined Functions, also known as UDF, **allow you to create custom functions to process records or groups of records**.
- Hive comes with a comprehensive library of functions.
- some specific cases for which UDFs are the solution.
- List of available Hive user-defined functions are,

Function Name	**Arguments**	**Return type**
datediff	end_date varchar(50), start_date varchar(50)	int
date_add	start_date varchar(50), daysToAdd int	varchar(50)
date_sub	start_date varchar(50), daysToSubtract int	varchar(50)
decode	binToDecode varbinary(50), charSet varchar(50)	varchar(50)