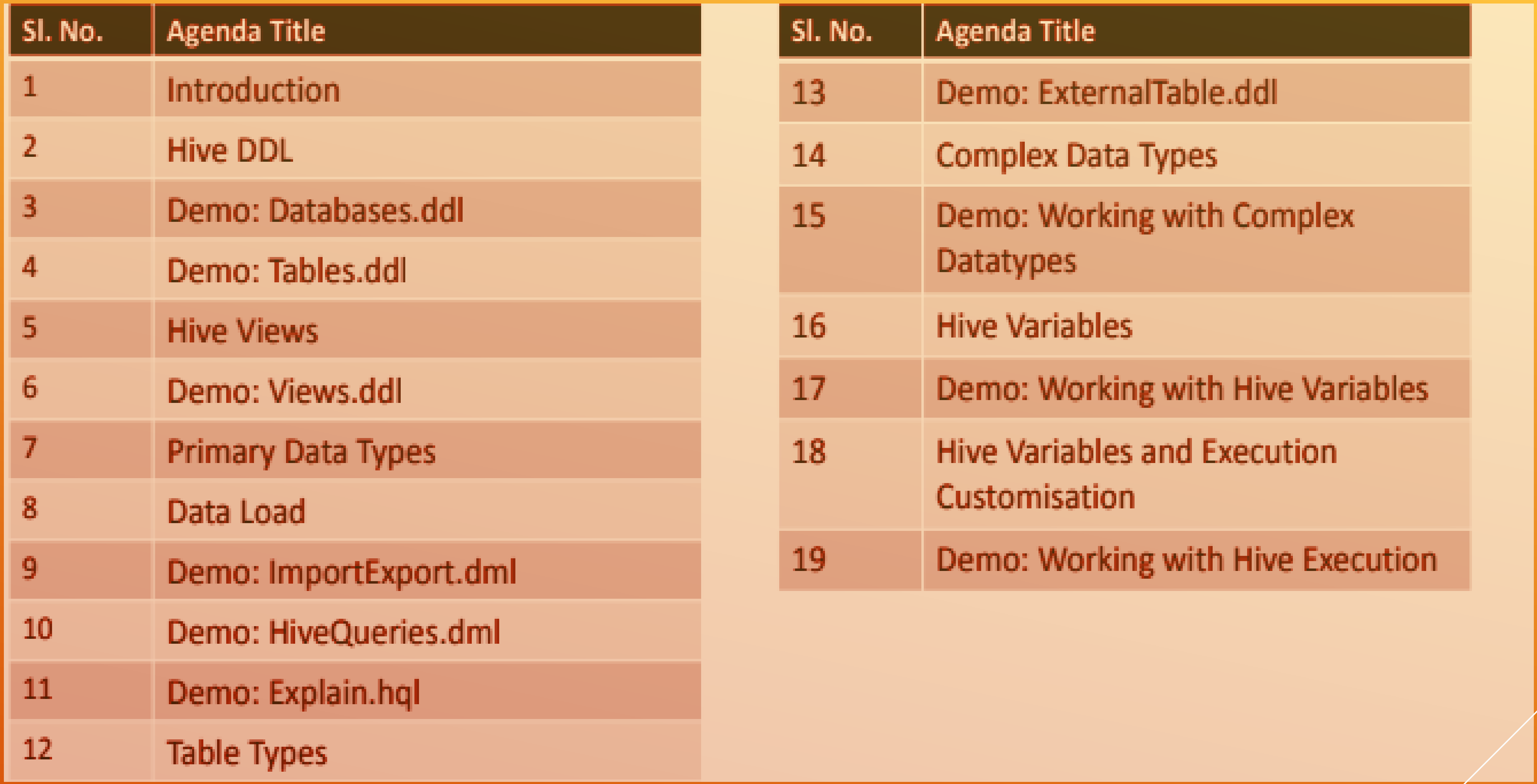
Hive & SQL

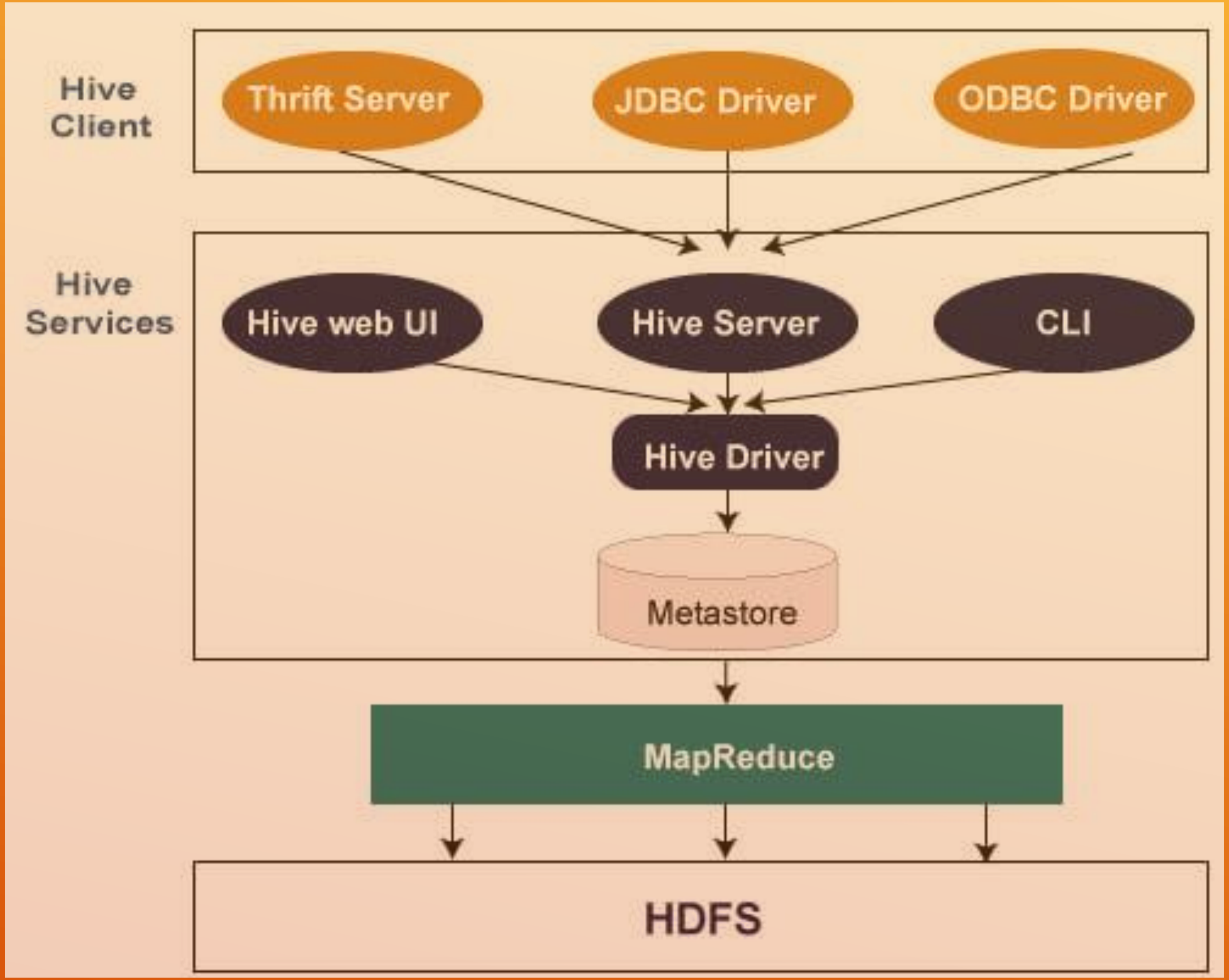


Sl. No.	Agenda Title
1	Introduction
2	Hive DDL
3	Demo: Databases.ddl
4	Demo: Tables.ddl
5	Hive Views
6	Demo: Views.ddl
7	Primary Data Types
8	Data Load
9	Demo: ImportExport.dml
10	Demo: HiveQueries.dml
11	Demo: Explain.hql
12	Table Types
13	Demo: ExternalTable.ddl
14	Complex Data Types
15	Demo: Working with Complex Datatypes
16	Hive Variables
17	Demo: Working with Hive Variables
18	Hive Variables and Execution Customisation
19	Demo: Working with Hive Execution

INTRODUCTION

- A data warehouse solution built on top of Hadoop - by Facebook.
- An essential tool in the Hadoop ecosystem that provides an SQL (Structured QueryLanguage) dialect (called as Hive Query Language) for querying data stored in the Hadoop Distributed Filesystem (HDFS).
- Most data warehouse applications are implemented using relational databases that use SQL as the query language. Hive lowers the barrier for moving these applications to Hadoop. People who know SQL can learn Hive easily.
- Automatically uses HDFS for storage, but stores all the meta information about database and table in metadata DB locally to Hive.
- Hive is most suited for data warehouse applications, where relatively static data is analyzed, fast response times are not required, and when the data is not changing rapidly.

HIVE ARCHITECTURE



HIVE DDL

- **DATABASES**
  - 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.
  - The simplest syntax for creating a database is shown in the following example:
- **CREATE DATABASE ineuron_db;**
  - Hive will throw an error if ineuron_db already exists.
  - You can suppress these warnings with this variation:
- **CREATE DATABASE IF NOT EXISTS ineuron_db;**
- **SHOW DATABASES LIKE 'a.*';**

HIVE DDL (CONTD.)

DATABASES

- Tables in that database will be stored in subdirectories of the database directory.
- The exception is tables in the default database, which doesn't have its own directory.
- The database directory is created under a top-level directory specified by the property
  - hive.metastore.warehouse.dir
  - set hive.metastore.warehouse.dir;
- You can override this default location for the new directory as shown:
- CREATE DATABASE ineuron_db
- LOCATION '/user/ineuron/mydb';

HIVE DDL (CONTD.)

DATABASES

- DESCRIBE DATABASE ineuron_db;
- The USE command sets a database as your working database, analogous to changing working directories in a filesystem.
  - USE ineuron_db;
  - set hive.cli.print.current.db=true;
  - DROP DATABASE IF EXISTS ineuron_db CASCADE;

HIVE DDL (CONTD.)

- The IF EXISTS is optional and suppresses warnings if ineuron_db doesn’t exist.
- By default, Hive won’t permit you to drop a database if it contains tables.
- You can either drop the tables first or append the CASCADE keyword to the command,which will cause the Hive to drop the tables in the database first:
- DROP DATABASE IF EXISTS ineuron_db CASCADE;
- When a database is dropped, its directory is also dropped.

HIVE DDL (CONTD.)

TABLES

- The CREATE TABLE statement follows SQL conventions, but Hive's version offers significant extensions to support a wide range of flexibility where the data files for tables are stored, the formats used, etc.

```
create table if not exists emp_details
(
        emp_name string,
        unit string,
        exp int,
        location string
)
row format delimited
fields terminated by ',';
```

- DROP TABLE IF EXISTS emp_details;

HIVE DDL (CONTD.)

TABLES

- Most table properties can be altered with ALTER TABLE statements, which change metadata about the table but not the data itself.

Renaming a Table

- Use this statement to rename the table emp_details to employee_details:

```
ALTER TABLE emp_details RENAME TO employee_details;
```

HIVE DDL (CONTD.)

Changing Columns

- You can rename a column, change its position, type, or comment:

```
ALTER TABLE emp_details
CHANGE COLUMN emp_name emp_name STRING
COMMENT 'Employee Name'
AFTER unit;
```

- You have to specify the old name, a new name, and the type, even if the name or type is not changing.

HIVE VIEWS

- A common use case for views is restricting the result rows based on the value of one or more columns.
- When a query becomes long or complicated, a view may be used to hide the complexity by dividing the query into smaller, more manageable pieces; similar to writing a function in a programming language or the concept of layered design in software.

```
CREATE VIEW joined_view AS
    SELECT * FROM people JOIN cart
    ON (cart.people_id=people.id)
WHERE firstname='john';
```

HIVE VIEWS (CONTD.)

- As part of Hive’s query optimization, the clauses of both the query and view may be combined together into a single actual query.
- The conceptual view still applies when the view and a query that uses it both contain an ORDER BY clause or a LIMIT clause.
- The view’s clauses are evaluated before the using query’s clauses.
- For example, if the view has a LIMIT 100 clause and the query has a LIMIT 200 clause, you’ll get at most 100 results.
- While defining a view doesn’t “materialize” any data, the view is frozen to any subsequent changes to any tables and columns that the view uses.
- Hence, a query using a view can fail if the referenced tables or columns no longer exist.

QUIZ

Q. When is it suggested to use a View?

1. When a query is composed of many inner and complex queries.
2. When we want to fetch specific records or columns from a table.
3. When we want to know table details along with query result.
4. When a table stores temporary data.

ANSWERS

Q.When is it suggested to use a View?

1. When a query is composed of many inner and complex queries.
2. When we want to fetch specific records or columns from a table.
3. When we want to know table details along with query result.
4. When a table stores temporary data.

PRIMARY DATA TYPES

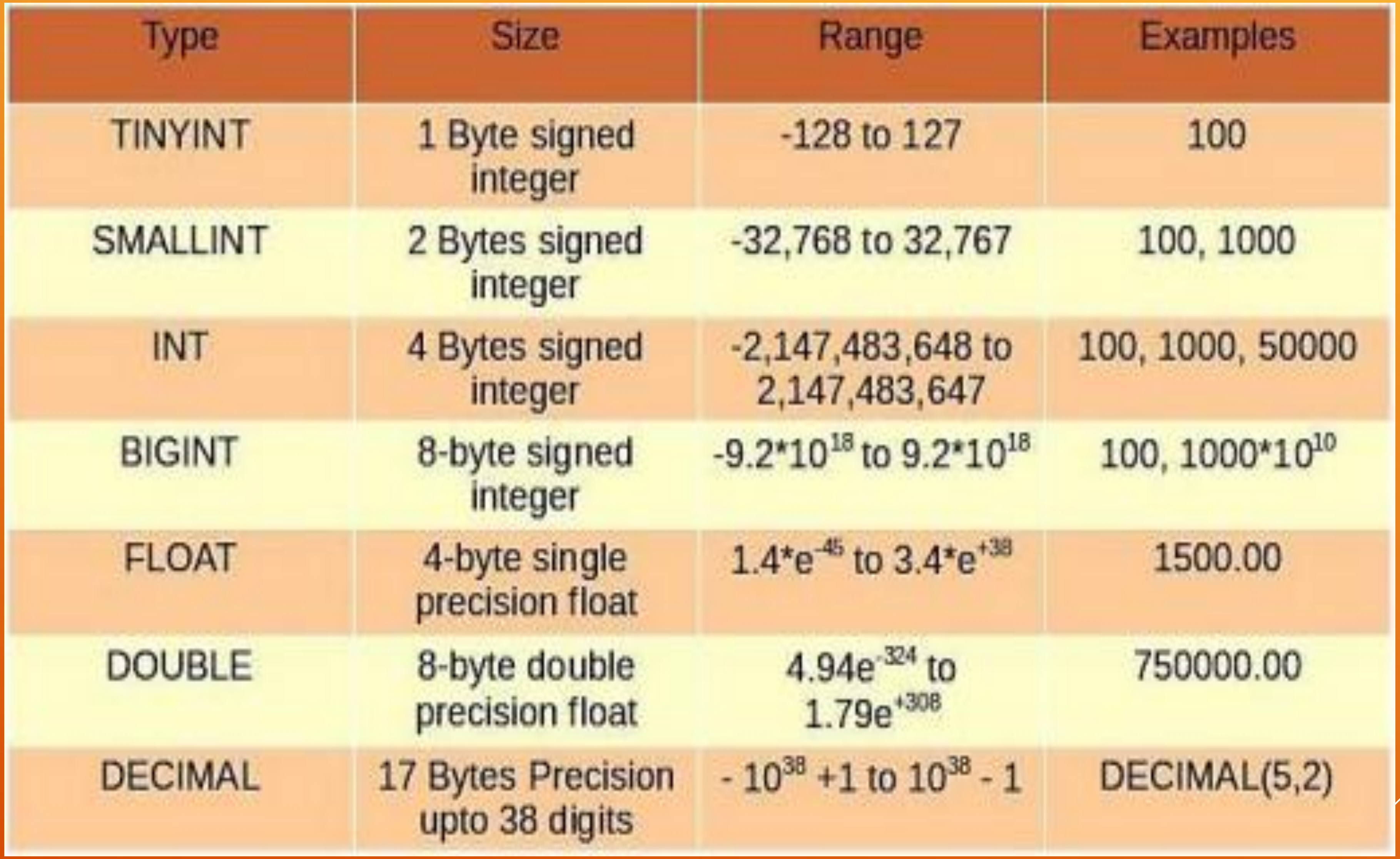
- Primary Data Types are further classified into four categories. They are:

  1. Numeric Types
  2. String Types
  3. Date/Time Types
  4. Miscellaneous Types

Numeric Data Types

- Integral types are – TINYINT, SMALLINT, INT & BIGINT- they store integer values
- Equivalent to Java's datatype i.e. byte , short , int , and long primitive types
- Floating types are – FLOAT, DOUBLE & DECIMAL.
- Equivalent to Java's float and double , and SQL's Decimal respectively.
- DECIMAL(5,3) represents total of 5 digits, out of which 3 are decimal digits. E.g. 13.345

PRIMARY DATA TYPES (CONTD.)
PRIMITIVE NUMERIC DATATYPE

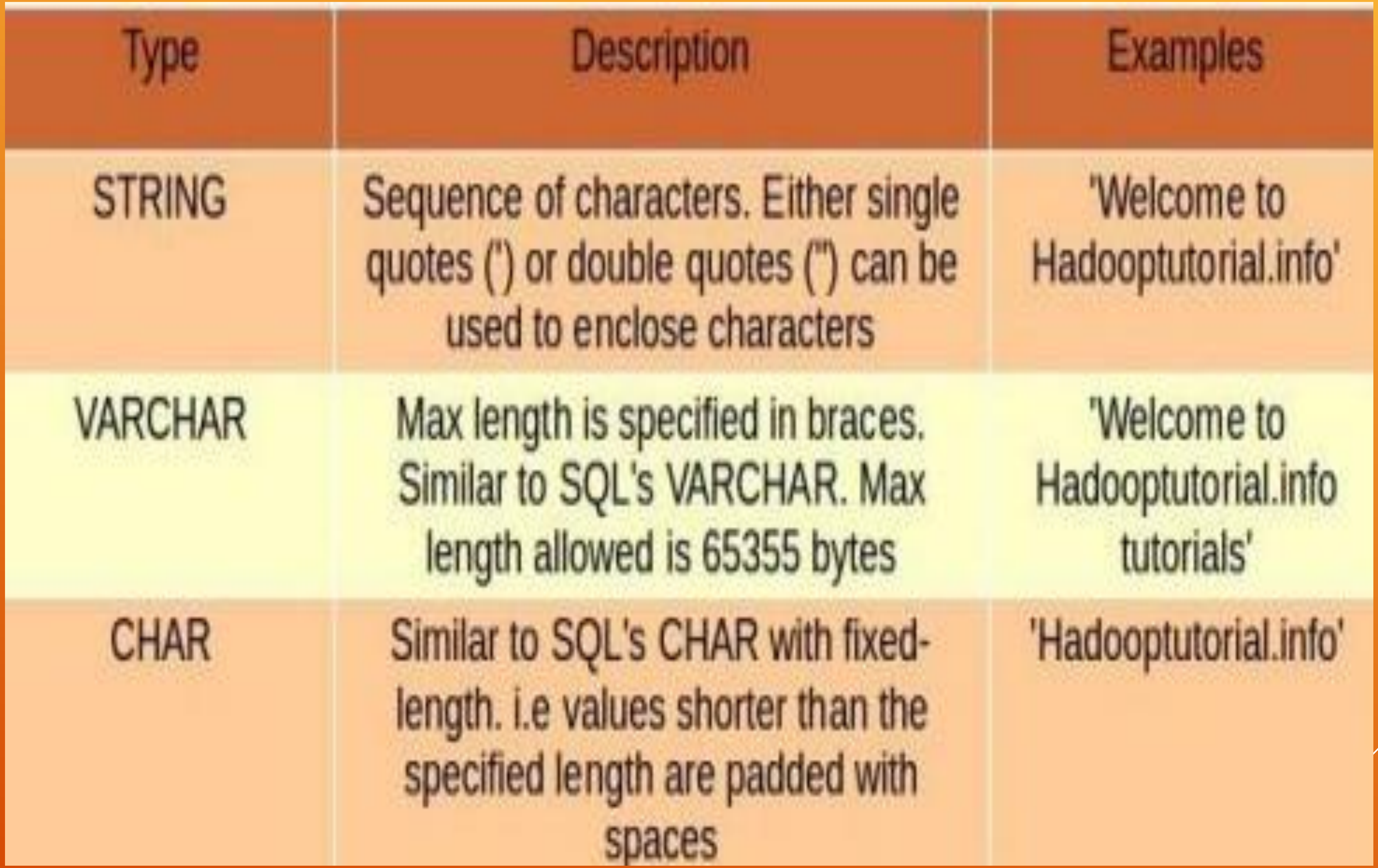


Type	Size	Range	Examples
TINYINT	1 Byte signed integer	-128 to 127	100
SMALLINT	2 Bytes signed integer	-32,768 to 32,767	100, 1000
INT	4 Bytes signed integer	-2,147,483,648 to 2,147,483,647	100, 1000, 50000
BIGINT	8-byte signed integer	$-9.2*10^{18}$ to $9.2*10^{18}$	$100, 1000*10^{10}$
FLOAT	4-byte single precision float	$1.4*e^{-45}$ to $3.4*e^{+38}$	1500.00
DOUBLE	8-byte double precision float	$4.94e^{-324}$ to $1.79e^{+308}$	750000.00
DECIMAL	17 Bytes Precision upto 38 digits	$-10^{38}+1$ to $10^{38}-1$	DECIMAL(5,2)

iNeuron

PRIMARY DATA TYPES (CONTD.)

PRIMITIVESTRING DATATYPE



Type	Description	Examples
STRING	Sequence of characters. Either single quotes (') or double quotes (") can be used to enclose characters	'Welcome to Hadooptutorial.info'
VARCHAR	Max length is specified in braces. Similar to SQL's VARCHAR. Max length allowed is 65355 bytes	'Welcome to Hadooptutorial.info tutorials'
CHAR	Similar to SQL's CHAR with fixed-length. i.e values shorter than the specified length are padded with spaces	'Hadooptutorial.info'

PRIMARY DATA TYPES (CONTD.)

Date/Time Types

- Hive provides DATE and TIMESTAMP data types in traditional (UNIX time stamp).
- UNIX time stamp format for date/time related fields in hive.
- DATE values are represented in the form YYYY-MM-DD. Example: DATE ‘2014-12-07’.
- Date ranges allowed are 0000-01-01 to 9999-12-31.
- TIMESTAMP use the format yyyy-mm-dd hh:mm:ss[.f...].
- We can also cast the String, Time-stamp values to Date format if they match format.

PRIMARY DATA TYPES (CONTD.)

Miscellaneous Types

- Hive supports two more primitive data types, BOOLEAN and BINARY. Similar to Java’s Boolean, BOOLEAN in hive stores true or false values only.
- BINARY is an array of Bytes and similar to VARBINARY in many RDBMSs

DATA LOAD

- Load operations are currently pure copy/move operations that move datafiles into location corresponding to Hive tables.
- create table if not exists saurav_tbl_csv

```
(
id string,
name string,
marks int
)
```

- row format delimited
- fields terminated by ','
- STORED AS TEXTFILE;
- LOAD DATA LOCAL INPATH '/tmp/saurav.csv' INTO TABLE saurav_tbl_csv;
- Filepath can refer to a file (in which case Hive will move the file into the table) or it can be a directory (in which case Hive will move all the files within that directory into the table).
- If the keyword LOCAL is specified, then the load command will look for filepath in the local file system.
- If a relative path is specified, it will be interpreted relative to the user's current working directory.
- The load command will try to copy all the files addressed by filepath to the target filesystem.

DATA LOAD (CONTD.)

- If the OVERWRITE keyword is used then the contents of the target table (or partition) will be deleted and replaced by the files referred to by filepath, otherwise the files referred by filepath will be added to the table.
- Query results can be inserted into filesystem directories.
- INSERT OVERWRITE [LOCAL] DIRECTORY directory1
- [ROW FORMAT row_format] [STORED AS file_format] (Note: Only available starting with

  Hive 0.11.0)

  SELECT ... FROM ...

QUIZ

Q. Identify the correct options

1. The SQL-like queries in Hive are translated to MapReduce jobs.
2. We can not access hive tables without Metastore.
3. A table with less records will have lesser entries in Metastore as compared to tables with larger records.
4. EXPLAIN gives the Abstract Syntax Tree of execution and then executes the MR job.

ANSWER

Q. Identify the correct options

1. The SQL-like queries in Hive are translated to MapReduce jobs.

2. We can not access hive tables without Metastore.

3. A table with less records will have lesser entries in Metastore as compared to tables with larger records.

4. EXPLAIN gives the Abstract Syntax Tree of execution and then executes the MR job.

TABLE TYPES

- When you drop an internal table, it drops the data, and it also drops the metadata.
- When you drop an external table, it only drops the meta data.
- On dropping the external table, the data does not get deleted from HDFS.
- Thus it is evident that the external table is just a pointer on HDFS data.

TABLE TYPES (CONTD.)

Use EXTERNAL tables when:

● The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.

●Data needs to remain in the underlying location even after a DROP TABLE. This can applyif you are pointing multiple schemas (tables or views) at a single data set or if you areiterating through various possible schemas.

● Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.

●You are not creating table based on existing table (AS SELECT).

Use INTERNAL tables when:

● The data is temporary.

● You want Hive to completely manage the lifecycle of the table and data.

Demo: ExternalTable.ddl

COMPLEX DATA TYPES

- Complex Types can be built up from primitive types and other composite types.
- Data type of the fields in the collection are specified using an angled bracket notation.
- Currently Hive supports four complex data types. They are:

ARRAY

- ARRAY<data_type>
- An Ordered sequences of similar type elements that are indexable using
- zero-based integers.
- It is similar to arrays in Java.
- Example – array (‘1’, ‘bala’, ‘praveen’);
- Second element is accessed with array[1].

COMPLEX DATA TYPES (CONTD.)

MAP

- MAP<primitive_type, data_type>
- Collection of key-value pairs.
- Fields are accessed using array notation of keys (e.g., [‘key’]).

STRUCT

- STRUCT<col_name : data_type [COMMENT col_comment], ...>
- It is similar to STRUCT in C language.
- It is a record type which encapsulates a set of named fields that can be any primitive data type.
- Elements in STRUCT type are accessed using the DOT (.) notation.
- Example – For a column c of type STRUCT {a INT; b INT} the a field is accessed by the expression c.a

COMPLEX DATA TYPES (CONTD.)

UNIONTYPE

- UNIONTYPE<data_type, data_type, ...>
- It is similar to Unions in C.
- At any point of time, an Union Type can hold any one (exactly one) data type from its specified data types.

QUIZ

Q. Identify the correct options

1. We can not insert data in EXTERNAL tables.- False
2. We can store integers in STRING array, in which data will be implicitly converted.-True
3. We can not have a single Hive tables with different complex types.-False
4. Struct can contain different data types referred by different names.-True

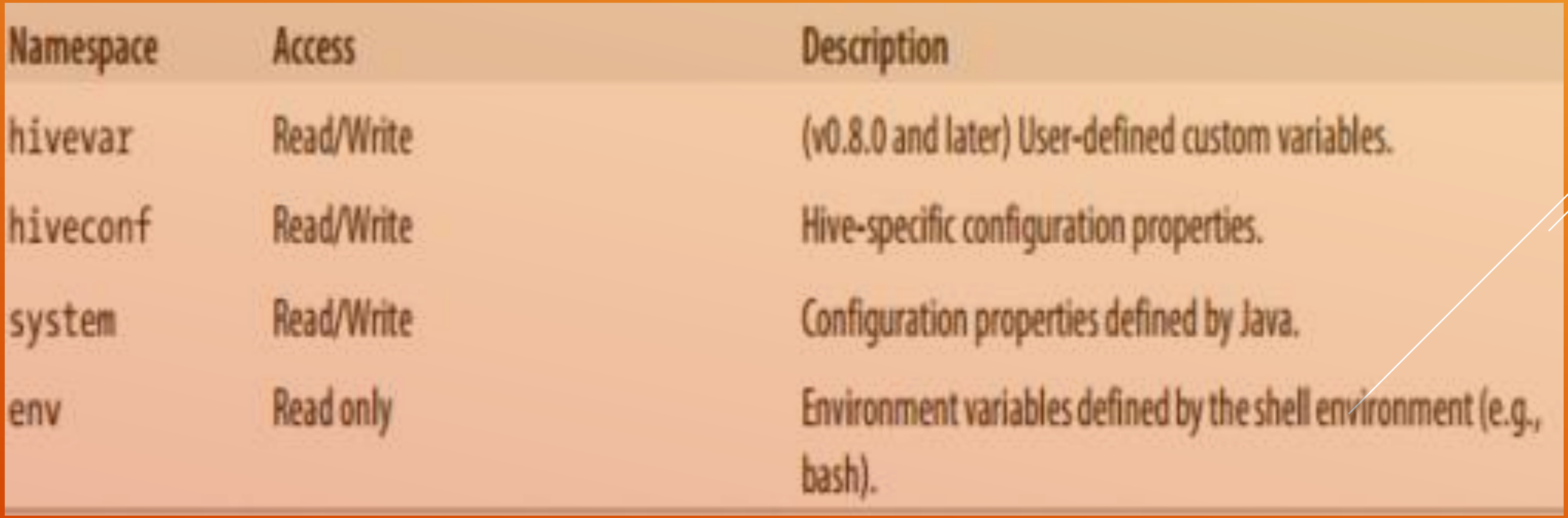
ANSWER

Q.Identify the correct options

1. We can not insert data in EXTERNAL tables.
2. We can store integers in STRING array, in which data will be implicitly converted.
3. We can not have a single Hive tables with different complex types.
4. Struct can contain different data types referred by different names.

HIVE VARIABLES

- You can define variables on the command line or inside script so that you can reference in Hive scripts to customize execution.
- Inside the CLI, variables are displayed and changed using the SET command.
- Hive's variables are internally stored as Java Strings.
- You can reference variables in queries; Hive replaces the reference with the variable's value before sending the query to the query processor.



Namespace	Access	Description
hivevar	Read/Write	(v0.8.0 and later) User-defined custom variables.
hiveconf	Read/Write	Hive-specific configuration properties.
system	Read/Write	Configuration properties defined by Java.
env	Read only	Environment variables defined by the shell environment (e.g., bash).

HIVE VARIABLES AND EXECUTION CUSTOMIZATION

- --hiveconf option is used for all properties that configure Hive behavior.
- Unlike hivevar variables, you have to use the system: or env: prefix with system properties and environment variables.
- -e option is used to execute query in a single shot.
- Adding the –S, starts execution in silent mode.
- Hive can execute one or more queries that were saved to a file using the -f file argument
- In $HOME/.hiverc file, one can specify a file of commands for the CLI to run as it starts, before showing you the prompt. Hive automatically looks for a file named .hiverc in your HOME directory and runs the commands it contains, if any.
- These files are convenient for commands that you run frequently, such as setting system or adding Java archives (JAR files) of custom Hive extensions to Hadoop’s distributed cache.

SORT BY VS ORDER BY VS CLUSTER BY VS DISTRIBUTE BY IN HIVE

Sort By:-

- Hive uses sort by to sort the data based on the data type of the column to be used for sorting per reducer i.e. overall sorting of output is not maintained. e.g. if column is of numeric type the data will be sorted per reducer in numeric order.

ORDER BY:-

- Very similar to ORDER BY of SQL. The overall sorting is maintained in case of order by and output is produced in single reducer. Hence, we need to use limit clause so that reducer is not overloaded.

DISTRIBUTE BY:-

- Hive uses the columns in *Distribute By* to distribute the rows among reducers. All rows with the same *Distribute By* columns will go to the same reducer. However,*Distribute By* does not guarantee clustering or sorting properties on the distributed keys.

CLUSTER BY:-

- Cluster By is a short-cut for both Distribute By and Sort By.

PARTITIONING

- Hive organizes tables horizontally into partitions.
- It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, department etc.
- Using partition, it is easy to query a portion of the data.
- Partitioning can be done based on more than column which will impose multi-dimensional structure on directory storage.
- In Hive, partitioning is supported for both managed and external tables.
- The partition statement lets Hive alter the way it manages the underlying structures of the table's data directory.
- In case of partitioned tables, subdirectories are created under the table's data directory for each unique value of a partition column.
- When a partitioned table is queried with one or both partition columns in criteria or in the WHERE clause, what Hive effectively does is partition elimination by scanning only those data directories that are needed.
- If no partitioned columns are used, then all the directories are scanned (full table scan) and partitioning will not have any effect.

CLASSIFICATION OF PARTITIONING

- Static partitioning
- Dynamic Partitioning

When to use static partitioning

- Static partitioning needs to be applied when we know data (supposed to be inserted) belongs to which partition.

When to use dynamic partitioning

- In static partitioning, every partitioning needs to be backed with individual hive statement which is not feasible for large number of partitions as it will require writing of lot of hive statements.
- In that scenario dynamic partitioning is suggested as we can create as many number of partitions with single hive statement.

BUCKETING

- Bucketing concept is based on (hashing function on the bucketed column) mod (by total number of buckets). The hash_function depends on the type of the bucketing column.
- Records with the same bucketed column will always be stored in the same bucket.
- We use CLUSTERED BY clause to divide the table into buckets.
- Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based.
- Bucketing can be done along with Partitioning on Hive tables and even without partitioning.
- Bucketed tables will create almost equally distributed data file parts, unless there is skew in data.
- Bucketing is enabled by setting hive.enforce.bucketing= true;

Advantages

- Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.
- As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables.
- Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient, since the join of each bucket becomes an efficient merge-sort.

BUCKETING VS PARTITIONING

- 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 that the same set of data is always written in same bucket.
- Bucketing helps a lot in joining of columns.
- Hive Bucket is nothing but another technique of decomposing data or decreasing the data into more manageable parts or equal parts.
- Partitioning a table stores data in sub-directories categorized by table location, which allows Hive to exclude unnecessary data from queries without reading all the data every time a new query is made.
- Hive does support Dynamic Partitioning (DP) where column values are only known at EXECUTION TIME. To enable Dynamic Partitioning :
- SET hive.exec.dynamic.partition =true;
- Another situation we want to protect against dynamic partition insert is that the user may accidentally specify all partitions to be dynamic partitions without specifying one static partition, while the original intention is to just overwrite the sub-partitions of one root partition.
- SET hive.exec.dynamic.partition.mode =strict;
- To enable bucketing:
- SET hive.enforce.bucketing =true;

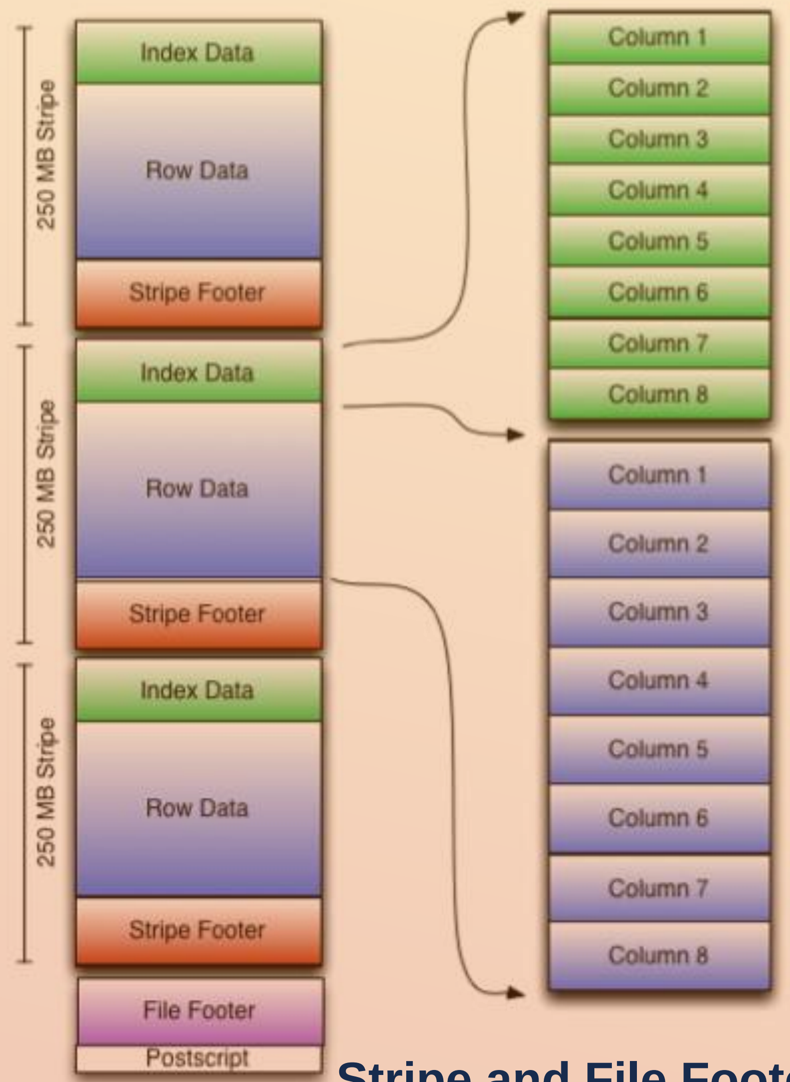
ORC FORMAT

- ORC stands for Optimized Row Columnar which means it can store data in an optimized way than the other file formats. ORC reduces the size of the original data up to 75%(eg: 100GB file will become 25GB). As a result the speed of data processing also increases. ORC shows better performance than Text, Sequence and RC file formats.
  An ORC file contains rows data in groups called as Stripes along with a file footer. ORC format improves the performance when Hive is processing the data.
- An ORC file contains groups of row data called **stripes**, along with auxiliary information in a **file footer**. At the end of the file a **postscript** holds compression parameters and the size of the compressed footer.
- The default stripe size is 250 MB. Large stripe sizes enable large, efficient reads from HDFS.
- The file footer contains a list of stripes in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum.
- This diagram illustrates the ORC file structure:

PARQUET FORMAT

- Parquet is an open source file format available to any project in the Hadoop ecosystem. Apache Parquet is designed for efficient as well as performant flat columnar storage format of data compared to row based files like CSV or TSV files.
- Columnar storage like Apache Parquet is designed to bring efficiency compared to row-based files like CSV. When querying, columnar storage you can skip over the non-relevant data very quickly. As a result, aggregation queries are less time consuming compared to row-oriented databases. This way of storage has translated into hardware savings and minimized latency for accessing data.
- Compression ratio is 60-70%.

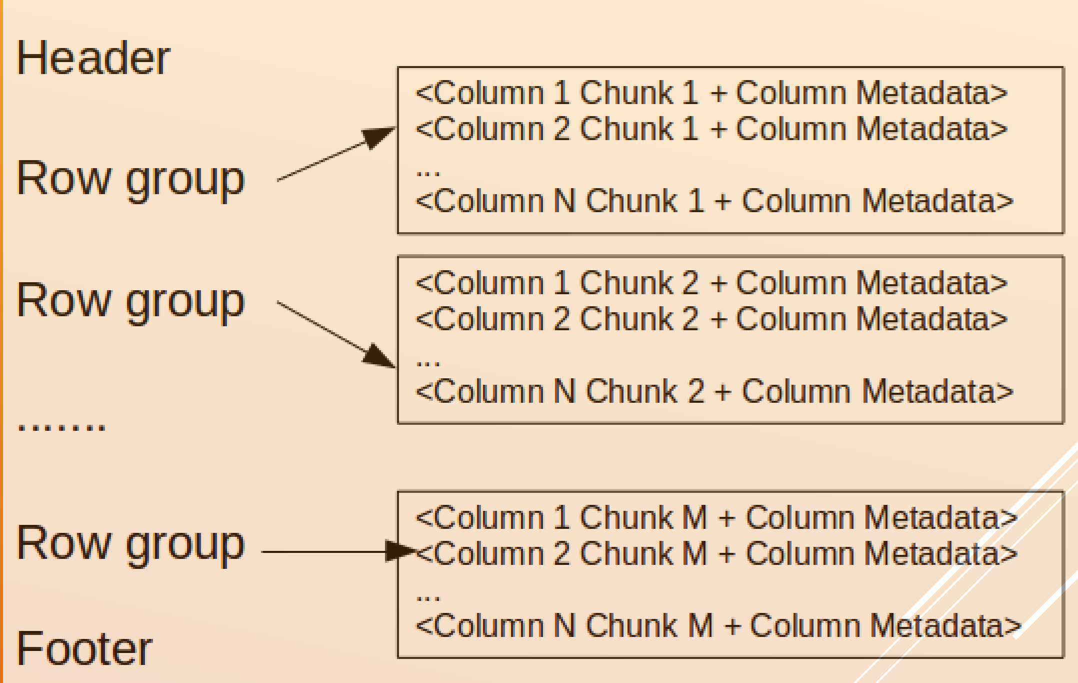
ORC



Index data includes min and max values for each column and the row positions within each column

Raw Data the row values

PARQUET



Header
Row group
Row group
....
Row group
Footer
<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>
<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>
<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>

Stripe and File Footer contains the list of stripes(Files footer) in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum at respective levels.

AVRO FORMAT

- -.avro
- -Compression ratio 50-55%
- -Schema is stored separately in avsc (avro schema) file
- -Schema evolution advantage – Even if one column data doesn’t appear automatically hive will default it to some value.
- -The default value is specified in the avsc file.

  Name , address, phone

- 1,blr,99168
- 2,Noida, 9999
- 3,nocityfound,9777

JOINS AND TYPES

Type	Approach	Pros	Cons
Shuffle Join	Join keys are shuffled using map/reduce and joins performed join side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
Broadcast Join	Small tables are loaded into memory in all nodes, mapper scans through the large table and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
Sort-Merge-Bucket Join	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be sorted and bucketed ahead of time.

Shuffle Joins – the default

customer

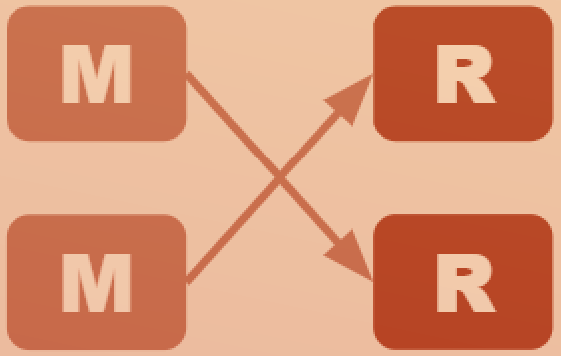
first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

```
SELECT * FROM customer join order ON customer.id = order.cid;
```

{ id: 11911, { first: Nick, last: Toner }}
{ id: 11914, { first: Rodger, last: Clayton }}
…
{ cid: 4150, { price: 10.50, quantity: 3 }}
{ cid: 11914, { price: 12.25, quantity: 27 }}
…

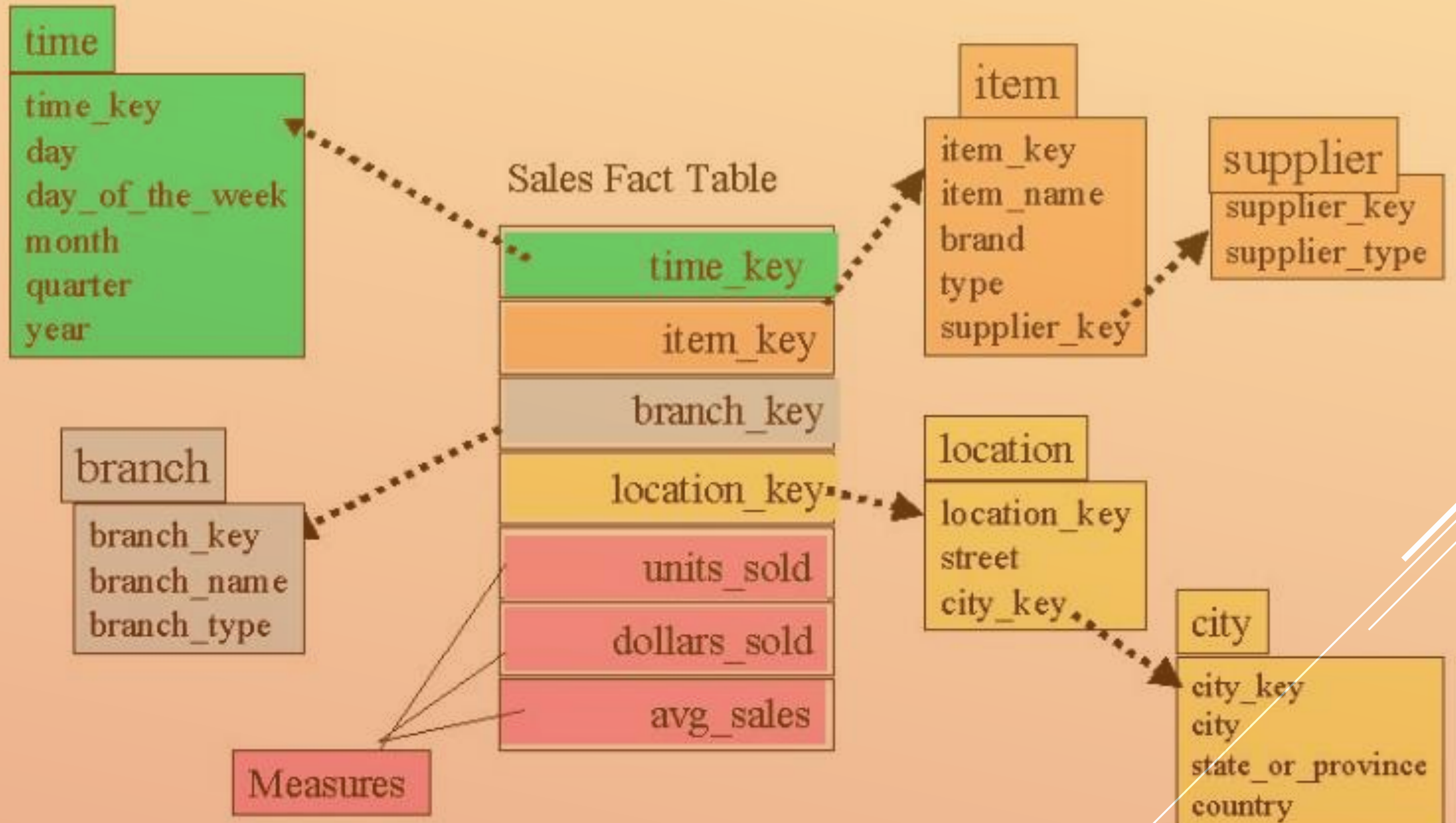


{ id: 11911, { first: Nick, last: Toner }}
{ cid: 4150, { price: 10.50, quantity: 3 }}
…
{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 }}

Identical keys shuffled to the same reducer. Join done reduce-side.
Expensive from a network utilization standpoint.

Broadcast Join (aka Map-side Join)

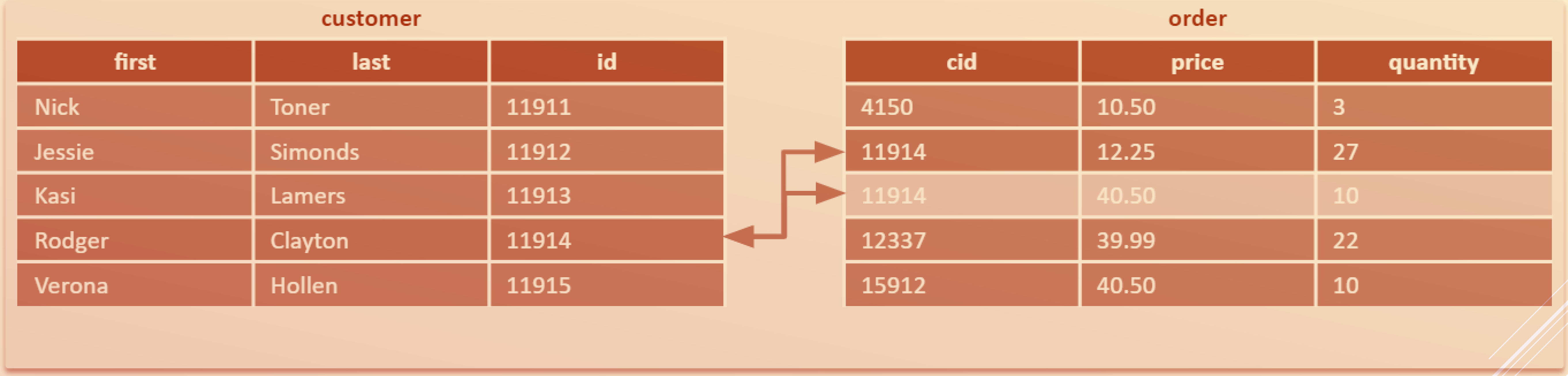
- Star schemas (e.g. dimension tables)
- Good when table is small enough to fit in RAM



USING BROADCAST JOIN

- Set hive.auto.convert.join = true
- HIVE then automatically uses broadcast join, if possible
  - Small tables held in memory by all nodes
  - Single pass through the large table
- Used for star-schema type joins common in Data warehousing use-cases
- hive.auto.convert.join.noconditionaltask.size (by default 10 MB) determines data size for automatic conversion to broadcast join:
  - Default 10MB is too low (check your default)
  - Recommended: 256MB for 4GB container

Sort-Merge-Bucket join: When both are too large for memory



customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
11914	40.50	10
12337	39.99	22
15912	40.50	10

```
CREATE TABLE order (cid int, price float, quantity int)
CLUSTERED BY(cid) SORTED BY(cid) INTO 32 BUCKETS;

CREATE TABLE customer (id int, first string, last string)
CLUSTERED BY(id) SORTED BY(id) INTO 32 BUCKETS;

SELECT * FROM customer join order ON customer.id = order.cid;
```

OPTIMIZATIONS IN HIVE

- As join requires data to be shuffled across nodes, use filtering and projection as early as possible to reduce data before join.

TUNE CONFIGURATIONS

Compress map/reduce output

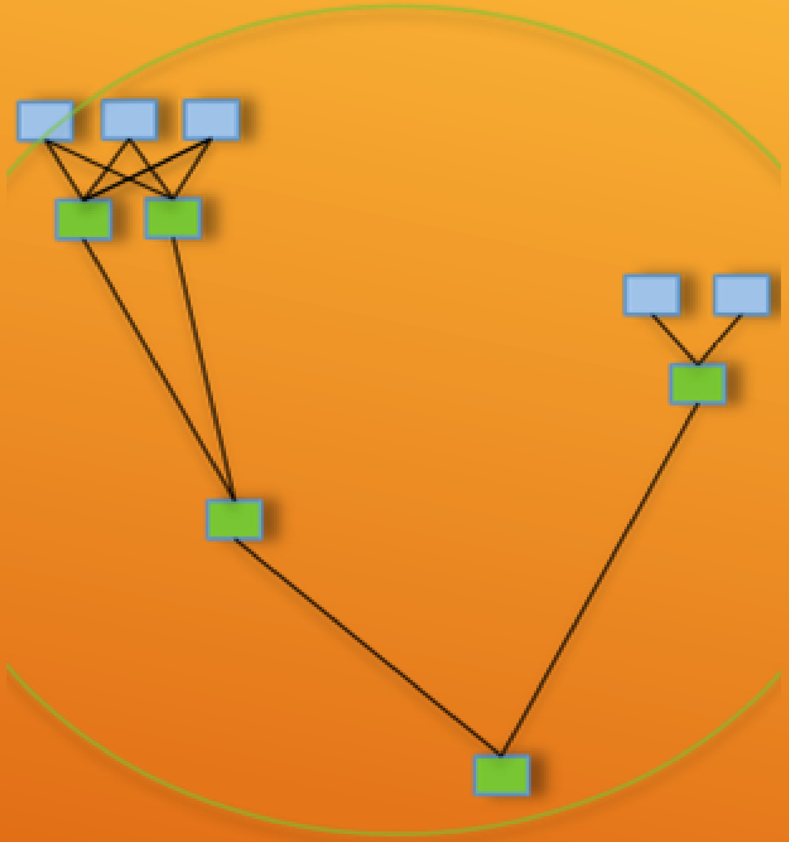
- SET mapred.compress.map.output =true;
- SET mapred.output.compress =true;

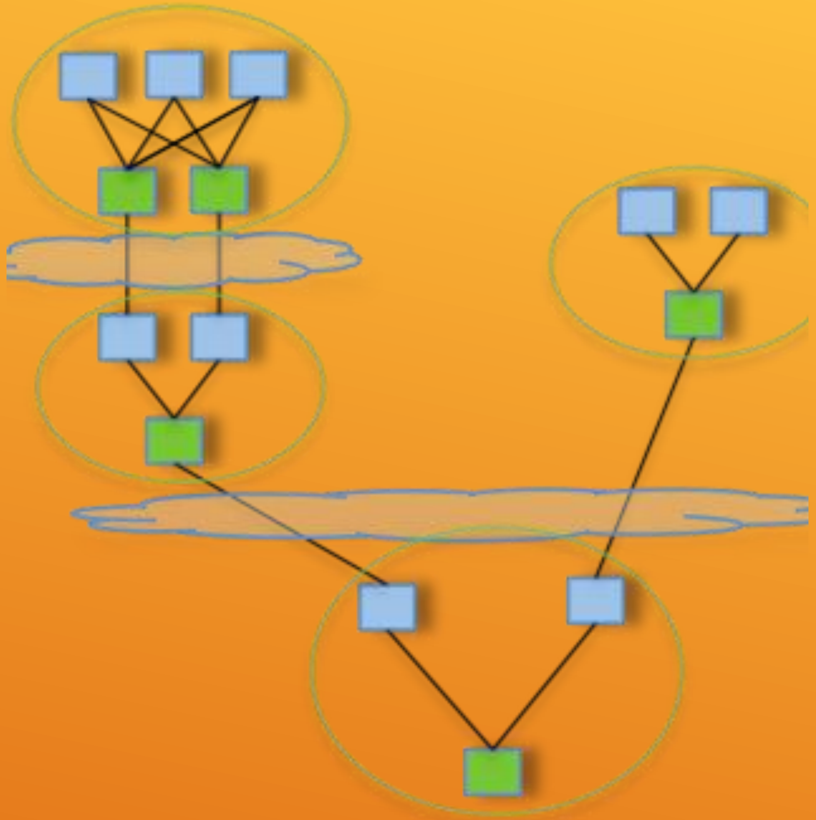
Parallel execution

- Applies to MapReduce jobs that can run in parallel, for example jobs processing different source tables before a join.
- SET hive.exec.parallel =true;

Vectorization

- Vectorized query execution is a Hive feature that greatly reduces the CPU usage for typical query operations like scans, filters, aggregates, and joins
- Vectorized query execution streamlines operations by processing a block of 1024 rows at a time (instead of 1 row at a time)
- ONLY works with ORCFiles
- SET hive.vectorized.execution.enabled = true;
- SET hive.vectorized.execution.reduce.enabled=true;


Pig/Hive - Tez


Pig/Hive - MR

OPTIMIZATIONS IN HIVE

- **USE TEZ**
- With Hadoop2 and Tez , the cost of job submission and scheduling is minimized.
- Also Tez does not restrict the job to be only Map followed by Reduce; this implies that all the query execution can be done in a single job without having to cross job boundaries.
- Creates a Directed Acyclic Graph to execute in shortest path possible.

HIVE CBO

Cost based optimization using Apache Calcite project reduces the shuffle and does below optimizations-

- How to order Join

- What algorithm to use for a given Join

- Should the intermediate result be persisted or should it be recomputed

- The degree of parallelism at any operator (specifically number of reducers to use).

- Semi Join selection

HIVE COST-BASED OPTIMIZATION (CBO)

- ***Cost-Based Optimization*** (CBO) engine uses statistics within Hive tables to produce optimal query plans
- Two types of stats used for optimization:
  - Table stats
  - Column stats
- Uses an open-source framework called ***Calcite*** *(formerly Optiq)*
- Stats are collected at the table level automatically when: set hive.stats.autogather=true;
- If you have an existing table without stats collected:
- CBO uses statistics about Hive tables, table partitions, and columns within a table to produce good query execution plans.
  - More efficient query plans better utilize cluster resources and improve query latency.
  - CBO is most useful for complex queries containing multiple JOIN statements and for queries on very large tables.
  - CBO uses column level statistics to come up with better query plans.
- Below parameters needs to be set to enable CBO:
- set hive.cbo.enable=true;
- set hive.compute.query.using.stats=true;
- set hive.stats.fetch.column.stats=true;
- set hive.stats.fetch.partition.stats=true;

HIVE ACID

- **New type of table that supports Insert/Update/Delete/Merge SQL operations (SQL**
- **2011)**
- **Concept of ACID transaction (Atomic, Consistent, Isolated, Durable)**
    - **CREATE TABLE T(a int, b int)**
    - **CLUSTERED BY (b) INTO 8 BUCKETS STORED AS ORC**
    - **TBLPROPERTIES ('transactional'='true');**
- **Not all tables support transactional semantics**
- **Non ACID tables must be re-created**
- **Structure limitations:**
    - Table must be bucketed
    - Table cannot be sorted
    - Requires ORC File

DELETE FROM hello_acid WHERE key = 2;
UPDATE hello_acid SET value = 10 WHERE key = 3

Syntax : Merge Statement – SQL Standard 2011

Target : table being modified

ID	State	County	Value
1	CA	LA	19.0
2	MA	Norfolk	15.0
7	MA	Suffolk	50.15
16	CA	Orange	9.1

Source : table from which modification are read

ID	State	Value
1		20.0
7		80.0
100	NH	6.0

```
MERGE INTO TARGET T
        USING SOURCE S ON T.ID=S.ID
                WHEN MATCHED THEN
        UPDATE SET T.Value=S.Value
        WHEN NOT MATCHED
                INSERT (ID,State,Value)
        VALUES(S.ID, S.State, S.Value)
```

ID	State	County	Value
1	CA	LA	20.0
2	MA	Norfolk	15.0
7	MA	Suffolk	80.0
16	CA	Orange	9.1
100	NH	null	6.0

SOLVE SMALL FILES PROBLEM

- Sometimes there are 100s of small files of e.g. 100 KB which is very less than the block size
- Unnecessary blocks will be occupied
- Unnecessary shuffle and sort will happen
- To avoid this we have to merge the files in a table on a periodic basis
- Run query **ALTER TABLE TABLENAME CONCATENATE**;
- This query will merge all the small files into 256 MB(Block size) file each.