Tech Mahindra

Apache Hive

Bigdata

TLS

Topics to be covered

- Introduction to Hive
- Manipulating Data with Hive
- Partitioning and Bucketing
- Advanced Hive Features
- Hive Best Practices

TLS

Tech Mahindra

Introduction to Hive

TLS

Introduction to Hive

- What Is Hive?
- Benefits of Hive
- Getting Data into Hive
- The Hive Architecture
- Creating Hive Tables
- Loading Data into Hive
- Storing Query Results in HDFS
- Creating Different Databases

TLS

What is Hive?

- Hive is an open-source data warehousing solution built on top of Hadoop.
- Hive is a way to allow non-Java programmers access to the data stored in Hadoop clusters.
- Used by Data analysts, Statisticians, Scientists etc
- Hive supports queries expressed in a SQL-like declarative language - HiveQL, which are compiled into MapReduce jobs that are executed using Hadoop.
- HiveQL includes a type system with support for tables.
- HiveQL contains primitive types, collections like arrays & maps and nested compositions of the same.

What is Hive? (cont'd)

- In addition, HiveQL enables users to plug in custom map-reduce scripts into queries.
- Hive also includes a system catalog - Metastore – that contains schemas and statistics, which are useful in data exploration, query optimization and query compilation.
- The Hive implementation at Facebook contains tens of thousands of tables and stores over 700TB of data. It is being used extensively for both reporting and ad-hoc analyses by more than 200 users per month.

Benefits of Hive

- Hive is much easier to learn and use as its based on standard SQL.
- Writing HiveQL queries is much faster than writing the equivalent Java code.
- Many people already know SQL.
- Can rapidly start using Hive to query and manipulate data in the cluster.

TLS

Getting Data Into Hive

- To get data into Hive, first access Hive through the Hive Shell.

```
$ hive>
```

- Invoke the Hive shell from the command line by typing

```
$ hive
```

- The shell responds with its own command prompt

```
hive>
```

- A file containing HiveQL code can be executed using the -f option

```
$ hive  -f  myfile.hql
```

TLS

Getting Data Into Hive (cont'd)

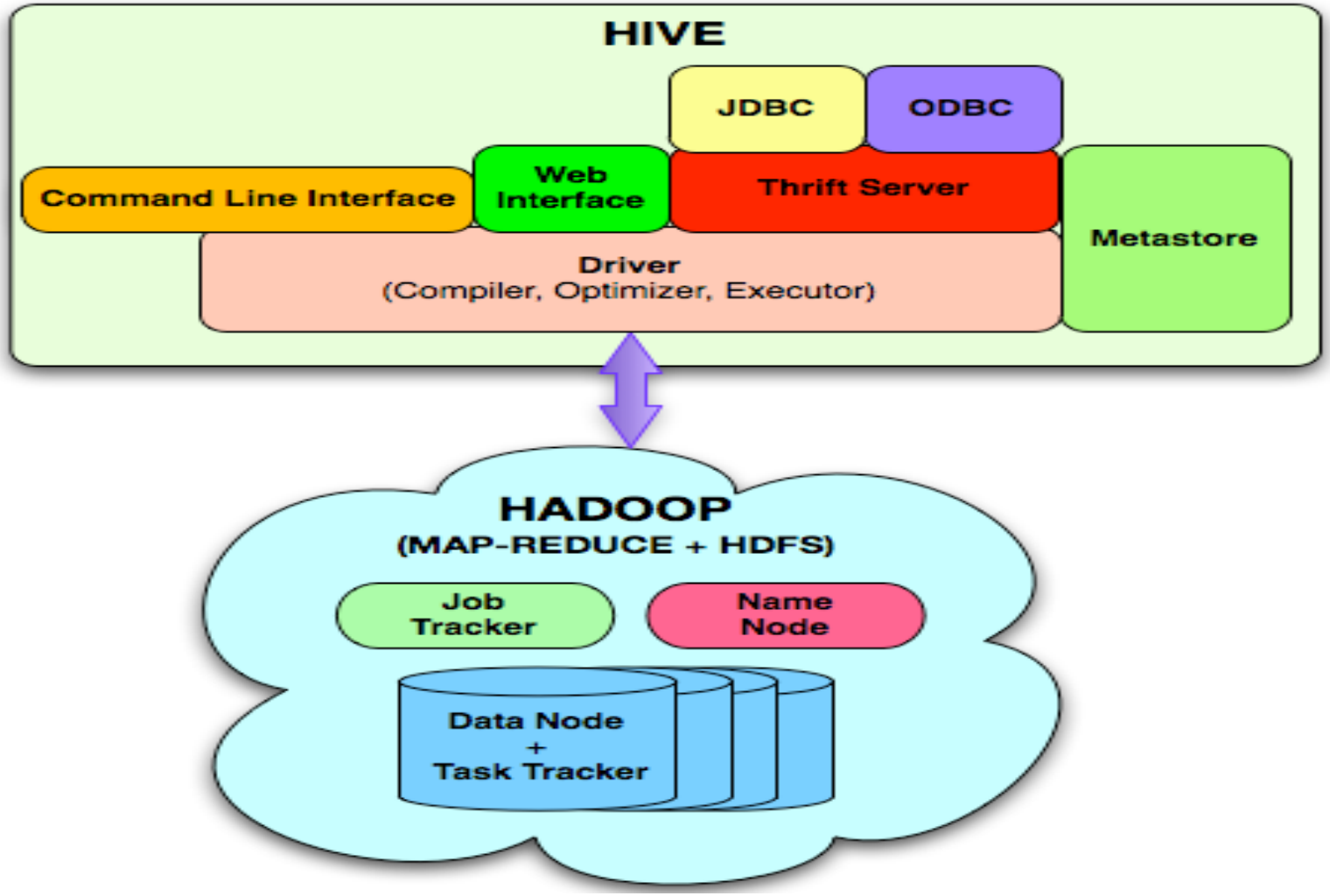
- HiveQL can be used directly from the command line using the -e option

```
$ hive -e 'select * from users`;
```

- As with most similar shells, HiveQL queries must be terminated with a Semicolon (;).

TLS

The Hive Architecture



TLS

The Hive Architecture (cont'd)

- Metastore: stores system catalog.
- Driver: manages the life cycle of HiveQL query as it moves thru' HIVE, also manages session handle and session statistics.
- Query compiler: Compiles HiveQL into a directed acyclic graph of map/reduce tasks.
- Execution engines: The component executes the tasks in proper dependency order; interacts with Hadoop.
- HiveServer: provides Thrift interface and JDBC/ODBC for integrating other applications.
- Client components: include - CLI, web interface, JDBC/ODBC interface, Extensibility interface include SerDe, User Defined Functions and User Defined Aggregate Function.

TLS

Creating a Table in Hive

- Example table definition:

```
CREATE TABLE movies (id INT, name STRING, year INT)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 STORED AS TEXTFILE
```

- ROW FORMAT DELIMITED
  - Tells Hive to expect one record per line.

- FIELDS TERMINATED BY . . .
  - Columns will be separated by the specified character.
  - Default separator is ‘Ctrl-A’.

TLS

Hive's Column Types

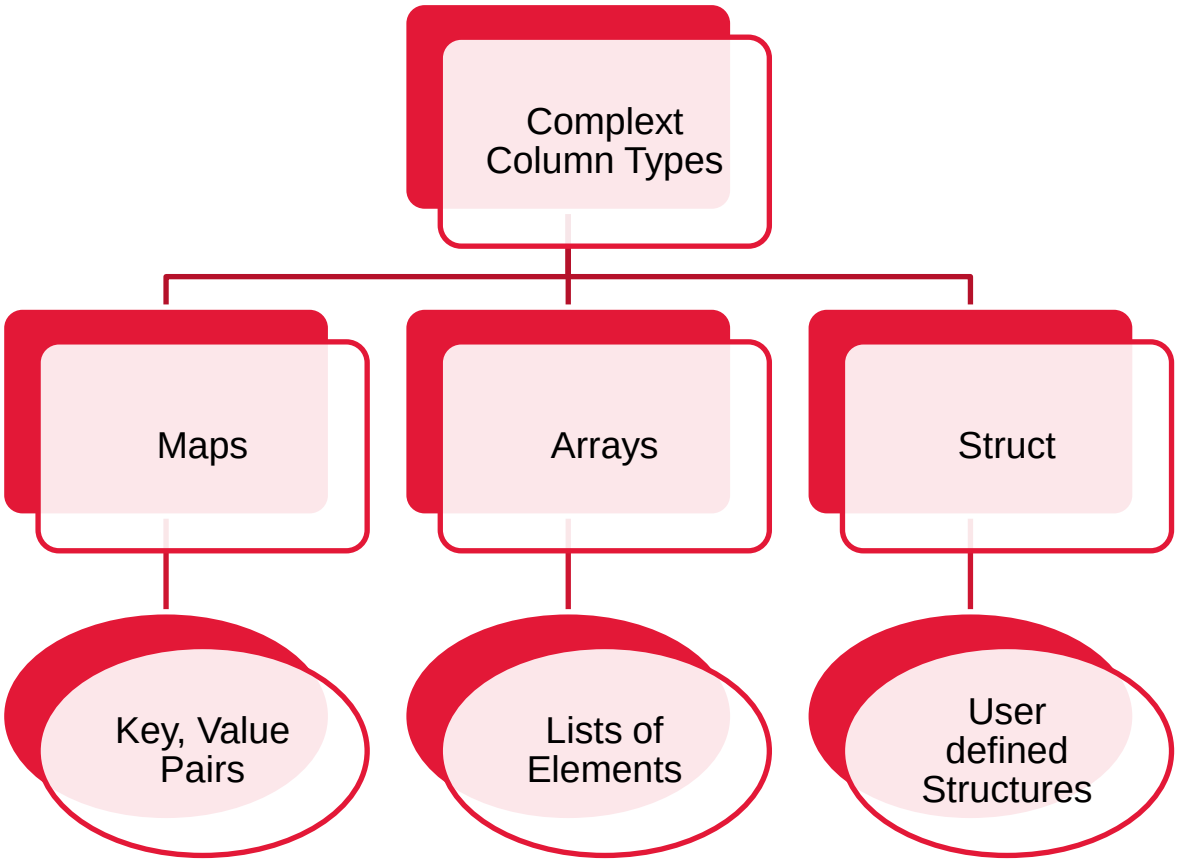
- Hive supports a number of different column types.
- Each type maps to a native data type in Java.

Type	Description
TINYINT	1 byte
SMALLINT	2 bytes
INT	4 bytes
BIGINT	8 bytes
FLOAT	Single precision
DOUBLE	Double precision
STRING	Sequence of characters
BOOLEAN	True/false
TIMESTAMP	YYYY/MM/DD HH:MM:SS.fffffffff

TLS

Hive’s Column Types (cont'd.)

- Hive has no binary column types like BLOB data type available in RDBMSs.
- Hive is designed to deal with textual data.



TLS

How Hive Stores Data

- By default, tables are stored in Hive's Warehouse HDFS directory
  - The contents of the files in this directory are considered to be the contents of the table.

hive> **CREATE TABLE** employees (name string, age int);

Now check if this table is stored as a directory in hive's warehouse in HDFS:

```
$ hadoop fs -ls /user/hive/warehouse/employees
drwxr-xr-r . . . /user/hive/warehouse/employees/data1.txt
```

TLS

How Hive Stores Data (cont'd)

Accessing table using hive shell:

```
hive> SELECT * FROM employees;
OK
Steve 32
John 28
Brian 35
```

Check the contents of table in HDFS using cat command :

```
$ hadoop fs -cat /user/hive/warehouse/employees/data1.txt
Steve 32
John 28
Brian 35
```

TLS

How Hive interprets Data?

- Hive layers a table definition onto a directory.
  - Doesn't verify the data when it is loaded, but rather when a
  - query is issued. This is called schema on read.

- The table definition describes the layout of the data files
  - Typically they are delimited files (using commas, tabs, or other characters).
  - Hive's default delimiter is the Control-A character.
  - This does not have to be the case if you use a custom Serializer/Deserializer.

- This table definition is saved in Hive's Metastore.

TLS

Hive's Metastore

- The Hive Metastore is held in a set of tables stored in an RDBMS.
  - Typically either Derby (the default) or MySQL.

- The data held in the Metastore includes:
  - Table definitions.
  - Table name, column names, column data types, etc.
  - Information on where the table data is stored in HDFS.
  - Row format of files in the table.
  - Storage format of the files in the table.
  - Determines which Input Format and Output Format the MapReduce jobs will use.

TLS

Submitting a Hive Query

- When a HiveQL query is submitted, the Hive interpreter on the client machine converts it into one or more MapReduce jobs.
- The MapReduce jobs are then submitted to the cluster by the interpreter.

TLS

External Table

- It is possible to create a table which points to an existing directory rather than having the directory automatically created under /user/hive/warehouse.
- This is known as an external table.

```
CREATE EXTERNAL TABLE movies (id INT, name STRING, year INT)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE
   LOCATION  ‘ /user/ian/movieFileDirectory ’
```

TLS

External Table (cont'd)

- External Tables
  - Creates a table and points to a specified directory in HDFS.
  - Hive does not use a default location for this table.
  - Very useful for existing data.
  - DROPing an External Table only removes metadata.

TLS

Table operations

- VIEW Table Definitions
  - DESCRIBE t (to see the columns of the table).
  - DESCRIBE EXTENDED t (to see detailed description of the table).
- DELETE the Table
  - DROP TABLE t (to delete the table).
- ALTER structure of existing tables
  - Add and remove partitions.
  - Rename a table/modify properties.
  - Modify Columns.
  - Change a table's location.

TLS

Table operations (cont'd)

- Examples
  - ALTER TABLE t ADD PARTITION (part_col='val')
  - ALTER TABLE t DROP PARTITION (part_col='val')
  - ALTER TABLE t RENAME TO x
  - ALTER TABLE t CHANGE old_name new_name new_type
  - ALTER TABLE t ADD COLUMNS (col_name, type, ...)
  - ALTER TABLE t SET LOCATION new_location (change location for external table)

TLS

Loading data into Hive

- Loading data into a Hive table involves moving the data files into the Hive table warehouse directory in HDFS

- This can be done by two ways
  - Can be done directly using hadoop fs command.

```
$ hadoop fs –mv  /depts/finance/salesdata   /user/hive/warehouse/sales/
```

  - Alternatively, use Hive's LOAD DATA INPATH command:

```
hive> LOAD DATA  INPATH  '/depts/finance/salesdata'
        INTO TABLE sales ;
```

TLS

Loading Data From Files (cont'd)

- Add the LOCAL keyword to load data from the local disk
  - Copies a local file (or directory) to the table's directory in HDFS

hive> **LOAD DATA LOCAL INPATH** '/home/bob/salesdata'

INTO TABLE sales;

Or Alternatively Hadoop fs **put** command

$ hadoop fs -put /home/bob/salesdata /user/hive/warehouse/sales

TLS

Loading Data From Files (cont'd)

- OVERWRITE
  - Direct update in hive tables is not possible in which case, load thenew updated file with updated records using OVERWRITE command

```
hive> LOAD DATA  INPATH '/depts/finance/salesdata'
        OVERWRITE  INTO TABLE sales;
```


Loading Data from a file to External Table

- Example
  - Define the External Table and point it to a directory within HDFS.

```
CREATE EXTERNAL TABLE p_view_stagq (hitTime INT,  userid STRING,
page_url STRING, comment STRING)
 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
 LOCATION '/user/data/staging/p_views';
```

  - Copy a local file into the specified HDFS directory.

```
$ hadoop fs -put  /tmp/pv_data.txt   /user/data/staging/p_views
```

TLS

Tech Mahindra

Hive Complex types

TLS

Complex types

- Hive suppots 3 complex types
  - Array
  - Map
  - Struct

TLS

Array type

- An array is a collection of individual atoms
- Each value in array has to be separated by a delimiter
- Elements are extracted using 0 based index

- **Syntax**:

```
CREATE TABLE <tablename>(
    column_definitions,
    columnname ARRAY<DATA TYPE>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY 'symbol'
COLLCTION ITEMS TERMINIATED BY 'symbol'
STORED AS file_type;
```

TLS

Array type example

```
CREATE TABLE array_tab(
    id          INT,
    name                STRING,
    phone               ARRAY<STRING>,
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
COLLECTION ITEMS TERMINATED BY ‘|’
STORED AS TEXTFILE;
```

Sample Text file

```
1,ramakumar,8544283421|8877727733
2,rajesh,3333333222|1122334455
3,latha,8822112233|9988776655
```

TLS

Data retrieval : Array type

- Data can be retrieved using below syntax

SELECT <columns>,array_column_name[index] FROM tablename

- Index starts from 0

- Example:

SELECT id,name,phone FROM array_tab;
SELECT id,name,phone[1] FROM array_tab;

TLS

Map type

- Map is a combination of (key,value) pair
- Value can be retrieved using key

- Syntax

```
CREATE TABLE <table_name>)
    column_definitions,
    award    MAP<key_data_type,value_data_type)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY 'symbol'
COLLECTION ITEMS TERMINATED BY 'symbol'
MAP KEYS TERMINATED BY 'symbol'
STORED AS file_type;
```

TLS

MAP Example

```
CREATE TABLE rewards(
    id          INT,
    name                STRING,
    award               MAP<STRING,INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
COLLECTION ITEMS TERMINATED BY ‘|’
MAP KEYS TERMINATED BY ‘#’
STORED AS TEXTFILE;
```

Sample Text file

```
1,rajesh,AOM#2002|POB#2002
2,radha,POB#2001|BRAVO#2009
3,rakesh,ACE#2013
```

TLS

Data retrival : MAP type

- Data can be retrieved using below syntax

SELECT <columns>,column_name[“key”] FROM tablename

- Index starts from 0
- Example:

SELECT * FROM rewards;
SELECT id,name,reward[“AOM”] FROM rewards;

STRUCT type

- A Structure is a collection of 2 or more elements placed in one column
- Each value in the structure is delimited

- **Syntax**

```
CREATE TABLE <table_name>(
    column_definitions,
    column_name STRUCT<elementname:data_type,
elementname:data_type>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY 'symbol'
COLLECTION ITEMS TERMINATED BY 'symbol'
STORED AS file_type;
```

TLS

Example Struct

```
CREATE TABLE employee(
    id          INT,
    name     STRUCT<fname:STRING,mname:STRING,lname:STRING>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
COLLECTION ITEMS TERMINATED BY ‘|’
STORED AS TEXTFILE;
```

Sample Text file

1,rama|kumar|prabhala
2,rajesh||maddali

TLS

Data retrieval : Struct type

- Data is retrieved using . (dot) operator
- Examples

```
SELECT * FROM employee;
SELECT id,name.fname FROM employee;
```

TLS

Tech Mahindra

DML Operations in Hive

TLS

Hive DML Operations

- Understanding ACID and Transactions in Hive
  - Hive support Transactional operations on Hive tables just like traditional RDBMS Systems
- Hive Supports DML Operations
  - Insert
  - Update
  - Delete

TLS

Configuration for Hive DML

- Configure below options in hive configuration file and restart hive server or issue below commands from hive prompt.

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

TLS

Create a table with ACID Properties

- Table updation is allowed only on tables that are created with transactional facility along with bucketing feature

```
CREATE TABLE <table_name>(
      Column list,
)
CLUSTERED BY (column_name)
INTO <number> BUCKETS
STORED AS ORCFILE
TBLPROPERTIES('transactional'='true');
```

TLS

Insert Operation on Hive Table

- Inserting Single Record into Hive Table
  - `INSERT INTO <table_name> VALUES(…)`
- Example
  - `INSERT INTO employee VALUES(1,'RAMA','10-JUN-1996','MALE')`
- Inserting Multiple Records into Hive Table
  - `INSERT INTO <table_name> VALUES (…),(…),…`
- Example
  - `INSERT INTO employee VALUES`

```
(1,'RAMA','10-JUN-1996','MALE'),
(2,'RADHA','12-MAY-1992,'FEMALE')
```

TLS

Update operation on Hive Table

- UPDATE <table_name> SET colname=value[,…] [WHERE <condition>

TLS

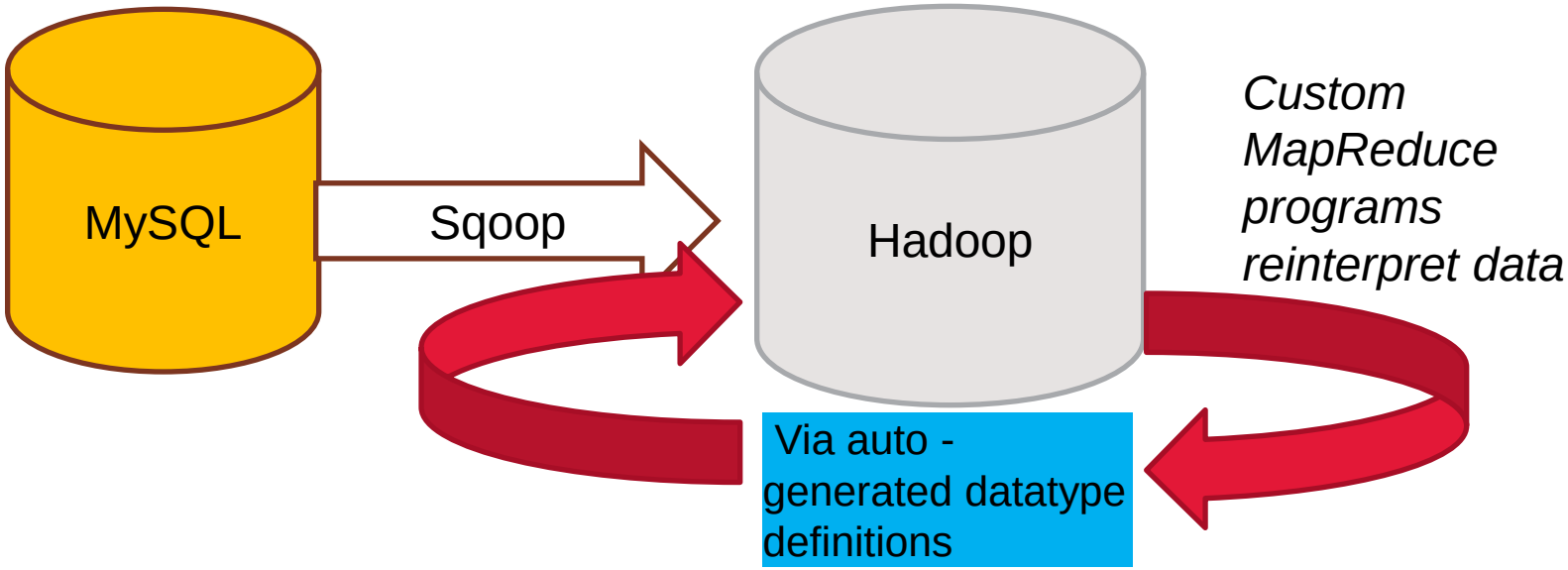
Delete operation on Hive Table

- DELETE FROM <table_name> [WHERE <condition>;

TLS

Sqoop – Import Data

- A frequent requirement is to load data from a table in an existing RDBMS.
- Sqoop, an open-source tool from Cloudera, helps to automate this process.



TLS

Sqoop Concepts

- The Sqoop program launches a Map-only MapReduce job to import the data.
- Makes multiple simultaneous connections to the RDBMS.
  - Default is four
  - Each connection imports a portion of the table
- Sqoop uses JDBC to connect to RDBMSs.
- Inspects the table and creates a mapping from the database column types to the corresponding Java data types.
- Can be configured to automatically create a Hive table from the imported data.

Sqoop Custom Connectors

- Connectors allow data transfer at much higher speeds than the standard JDBC interface.

- These connectors are not open-source but are provided free-of-charge by their respective OEMs (Original Equipment Manufacturers) like Netezza, Teradata, Oracle (via Quest software).

TLS

Sqoop Syntax

- Basic Sqoop syntax to import table t1 from database db, store as tab-separated files, and create a Hive table

```
sqoop import \
        --username user \
        --password pass \
        --connect jdbc:mysql://dbserver.example.com/db \
        --hive-import \
        --fields-terminated-by '\t' \
        --table t1
```

TLS

Storing Query Results in HDFS

- Writing Output into Local Filesystem or HDFS.
  - INSERT OVERWRITE DIRECTORY
- Writing Output to an existing Hive Table.
  - INSERT OVERWRITE
  - INSERT INTO

TLS

Writing Output an Existing Hive Table

- INSERT OVERWRITE
  - Copies data from one Hive table into another table
  - Overwrites contents of second table

```
INSERT OVERWRITE TABLE t2 SELECT * FROM t;
```

- INSERT INTO
  - Like INSERT OVERWRITE, it copies data from one table into another
  - However, it appends to (rather than overwrite) the contents of the second table
- INSERT INTO is only supported in Hive versions 0.8 and later

```
INSERT INTO TABLE t2 SELECT * FROM t;
```

TLS

Writing Output - Local file system or HDFS

- INSERT OVERWRITE DIRECTORY
  - Writes query results into HDFS
  - Can specify LOCAL to write to the local file system
  - Data written to the file system is serialized as text
  - Columns are separated by Ctrl-A characters
  - Rows are separated by newlines
  - Appropriate for extracting large amounts of data from Hive
- Example
  - Write out the entire contents of table ‘t’ to the local file system.

```
INSERT OVERWRITE LOCAL DIRECTORY '/path/file.dat'
        SELECT * FROM t;
```

TLS

Multiple Databases in Hive

- Earlier versions of Hive placed all tables in the same namespace.
- This is still the default.
- Recent versions support the creation of multiple databases.
  - Useful on clusters with multiple users
- Default database is named default.

TLS

Hands-On Exercise

Tech Mahindra

Manipulating Data with Hive

TLS

Data retrieval in Hive

- An Introduction to HiveQL
- Retrieving Data using the Select statement
- Joining Tables
- Basic Hive Functions
- More Advanced HiveQL Queries
- Statistics and Data Mining

TLS

An Introduction to HiveQL

- HiveQL is Hive's query language, a dialectic of SQL
  - Has many elements that are in SQL-92
  - Has some additional elements that are not in SQL-92
  - Supports JOINS, AGGREGATES,SUBQUERIES etc.
  - Does not support UPDATE or DELETE
- Includes Hive-specific extensions
  - Partitioning
  - Sampling
  - Complex data structures (Structs, Maps, Arrays)
  - User-defined functions
  - Multi-table inserts

TLS

Retrieving Data using select Statement

- Hive uses the SELECT statement to retrieve data.

SELECT expr, expr, … **FROM** tablename

- Expressions can be column names, function calls etc.
- FROM clause is required.
- Hive keywords are not case-sensitive.

Basic SELECT Syntax

- Tables can be aliased in Hive.
  - However, Hive only accepts the following syntax

```
SELECT expr, expr, … FROM tablename alias
```

  - Useful to get around a bug that won't allow a SELECT from a column that has the same name as the table name

```
SELECT alias.t  FROM t alias
```

  - Also note that the alternative AS syntax used by some RDBMSs is not supported in Hive

```
SELECT expr, expr, … FROM tablename AS alias
```

TLS

Limiting the rows returned

- Limit the rows returned from the query with the WHERE clause.

SELECT expr, expr, … **FROM** tablename **WHERE** condition

- Conditions can be any Boolean expression. Examples:
  - countryCode = 'USA‘
  - id > 100000
  - url LIKE '%.example.com’
  - col1 IS NULL
- Conditions can be combined using AND/OR.

TLS

Sorting the Rows returned

- HiveQL supports the ORDER BY expression.
- Can sort in descending order with ORDER BY expr DESC.

```
SELECT expr, expr, … FROM tablename
WHERE condition
ORDER  BY  expr;
```

- Often used with a LIMIT clause.
  - Limits the output to the first n rows

```
SELECT expr, expr, … FROM tablename
WHERE condition
ORDER  BY  expr
LIMIT n;
```

TLS

Sorting the Rows returned(Cont'd)

- HiveQL also supports the SORT BY expression
  - Sorts the rows before feeding to a reducer
  - Sort order is dependent upon the column type
  - i.e. if numeric then numeric order, if string then lexicographical (alphabetical) order.

```
SELECT expr, expr, … FROM tablename
WHERE condition
SORT  BY  expr;
```

TLS

SORT BY and ORDER BY

- SORT BY
  - May use multiple reducers for final output
  - Sorts the data per reducer
  - Only guarantees ordering of rows within reducers
  - May give partially ordered final results
- ORDER BY
  - Uses a single reducer to guarantee total order in output
  - Single reducer will take long to sort very large outputs
  - One reducer writes to one file in HDFS, this could fill the disk on a particular node
  - Use the LIMIT clause to minimize sort time

TLS

DISTRIBUTE BY

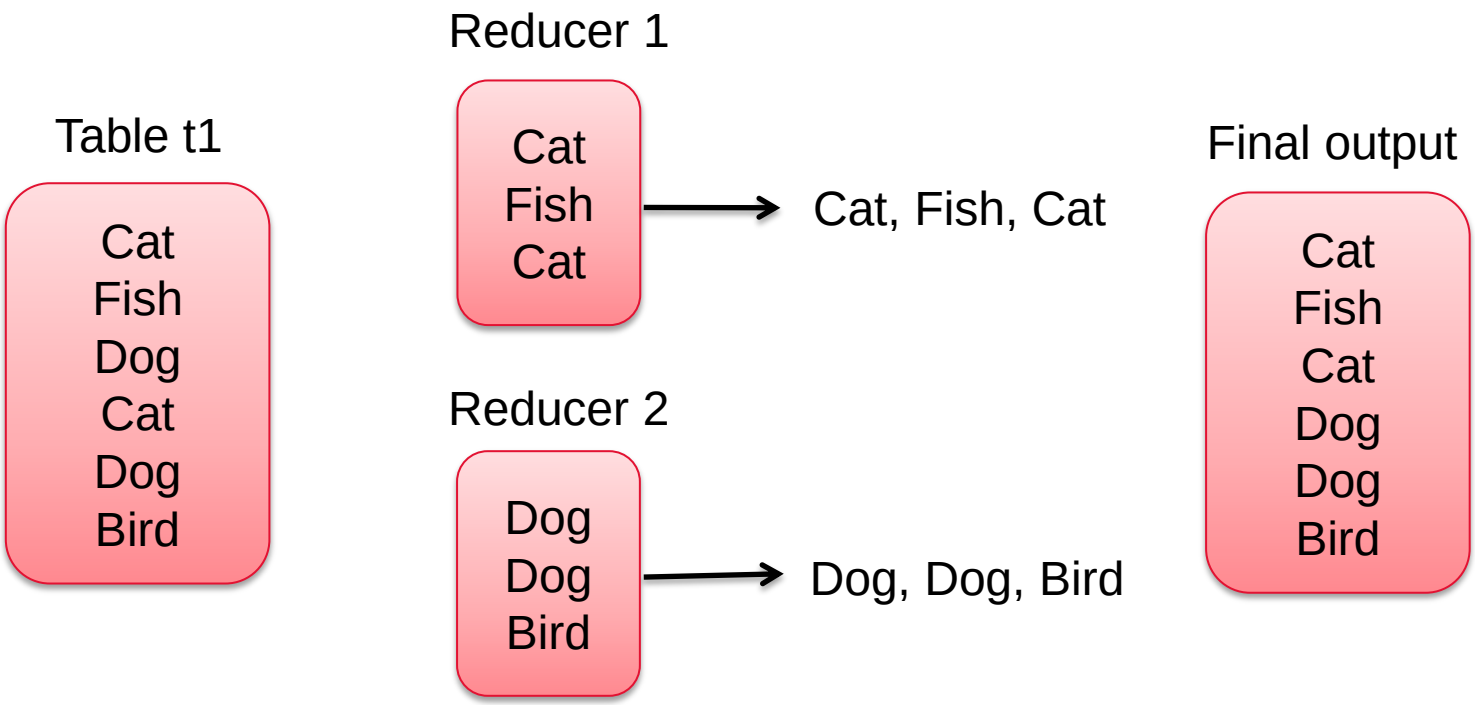
- DISTRIBUTE BY
  - Distributes the rows among reducers
  - Does not guarantee clustering or sorting properties
  - Useful if there is a need to partition and sort the output of the query for subsequent queries

TLS

DISTRIBUTE BY (cont'd)

- EXAMPLES

```
SELECT pets FROM t1 DISTRIBUTE BY pets;
```



CLUSTER BY

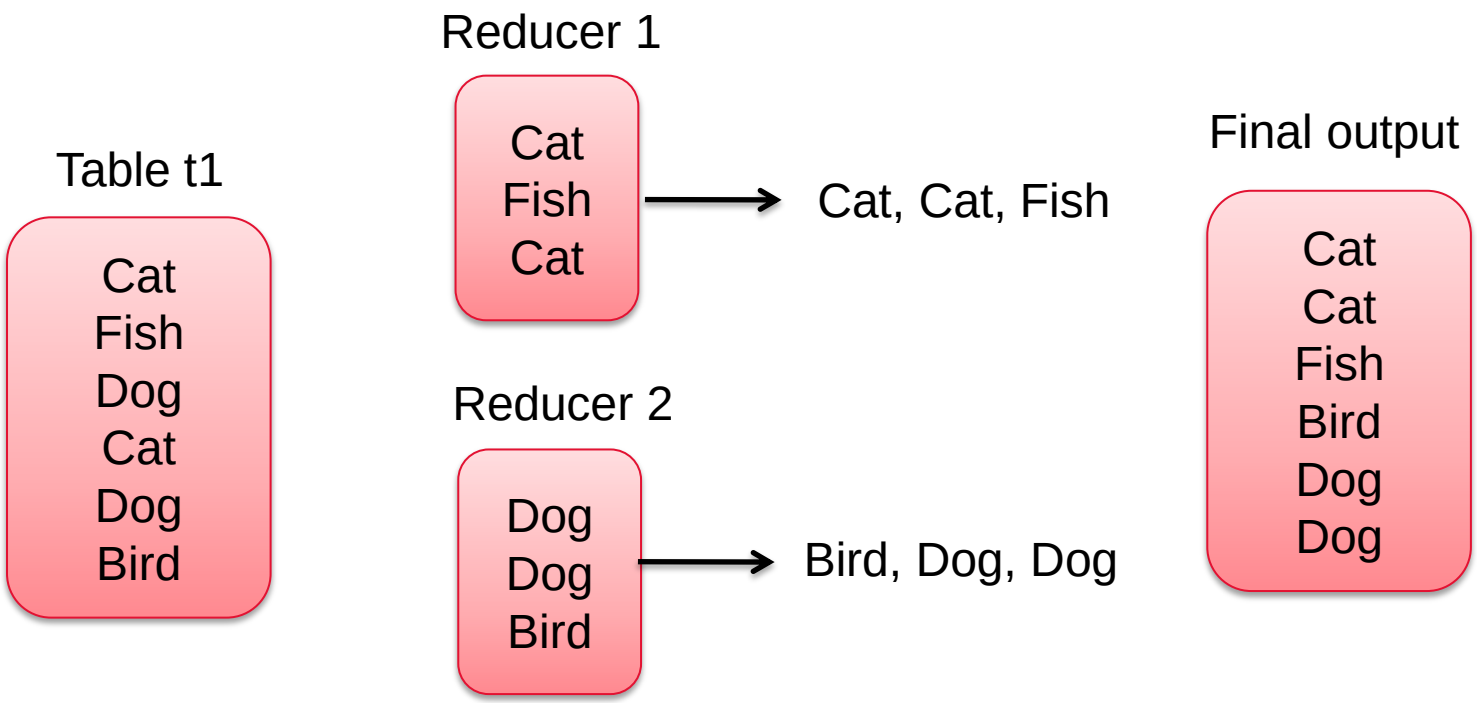
- CLUSTER BY
  - Combination of DISTRIBUTE BY and SORT BY
  - Rows with the same keys are distributed to the same reducer
  - Clustered in adjacent position and sorted per reducer

TLS

CLUSTER BY

- EXAMPLE

SELECT pets **FROM** t1 **CLUSTER BY** pets;



TLS

Grouping the rows returned

- The GROUP BY aggregate function is used to group the retrieved data by one or more columns

```
SELECT gender, count  (DISTINCT userid )
FROM users
GROUP  BY  gender;
```

- Also used with multiple aggregations

```
SELECT gender, count  (DISTINCT userid ), count(*)
FROM users
GROUP  BY  gender;
```

TLS

Tech Mahindra

Hands-On Exercise

TLS

Tech Mahindra

Joining Tables

TLS

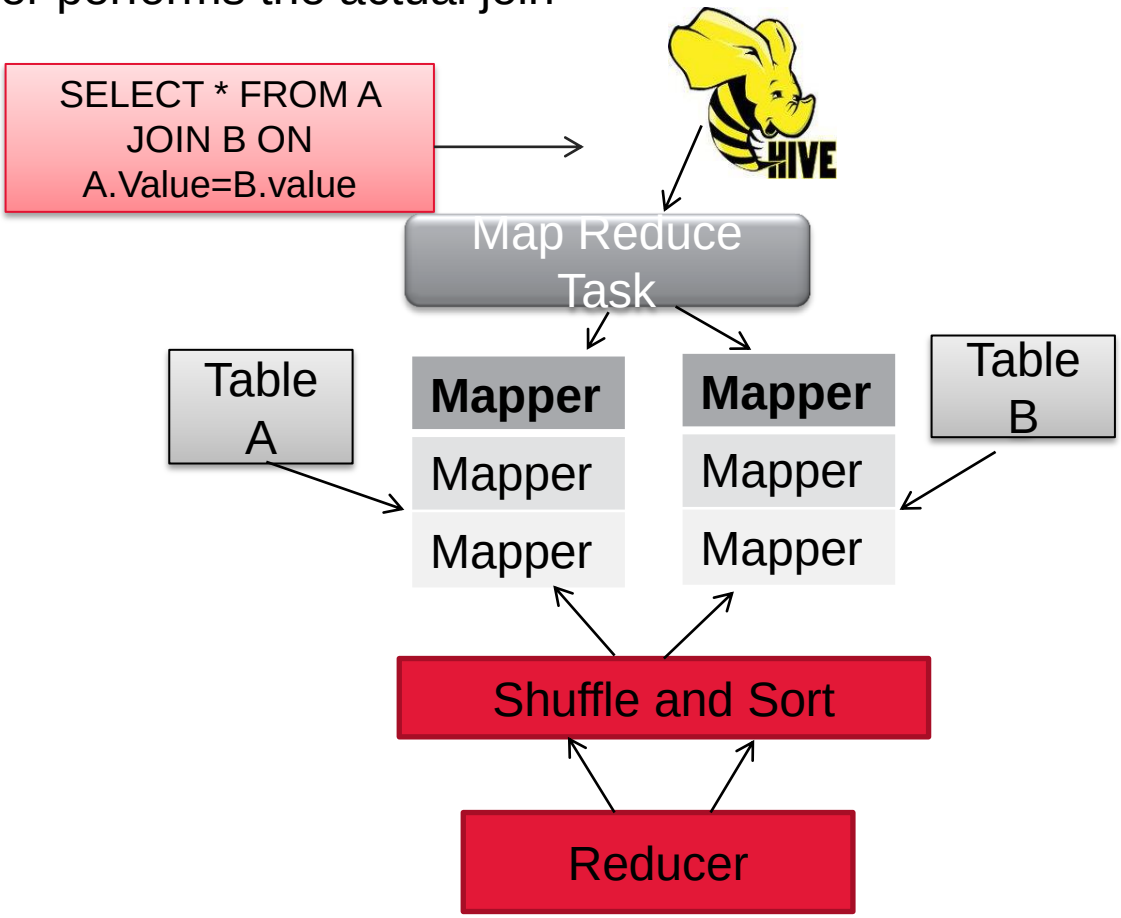
Joining Tables

- A frequent requirement is to join two or more tables together.
- Hive supports
  - Inner joins
  - Left Outer joins
  - Right Outer joins
  - Full Outer joins
  - Left semi joins
  - Returns rows from the first table when one or more matches are found
  - in the second table (e.g. list the departments in a table that have at
  - least one employee)

TLS

Default Join Behavior

- Reduce side join
  - Mapper reads the tables and emits the join key and join value
  - Reducer performs the actual join



TLS

Join Optimization

- Map-side join
  - Reads the smaller table to an in-memory hash table
  - Each mapper gets its own copy of the hash table
  - The join is performed in the mapper
  - No reducer or sort and shuffle phases are required

```
SELECT /*+mapjoin(B)*/ * FROM A JOIN B ON
A.Value = B.Value;
```

TLS

Map-side v/s Reduce-side Joins

- There are two fundamental ways to join dataset in MapReduce
  - Map-side join
  - Reducer-side join
- Map-side Joins
  - Function similar to an RDBMS joins
  - Scans the smaller table in memory for each record found in the larger table
  - Smaller table is shipped with the job, replicated and loaded into memory
  - Larger table can be filtered in the mapper
  - i.e. if there is no row match the row isn't sent across the network to the reducer

TLS

Map- side v/s Reduce- side Joins (cont'd)

- Pros
  - More efficient, but requires one dataset to be small enough to be held in the memory
- Cons
  - All data is shipped over the network to the reducers
  - Wasteful for inner joins where there are few matches between tables
  - Previous versions of Hive needed to be told if they could use Map-side join via a 'hint'
  - An extra clause in the HiveQL query
  - Largest version of Hive can work this out for themselves, based on the side of the input data
  - To enable this, set hive.auto.convert.join = true

TLS

Joining Tables: Syntax

- Hive uses the following syntax to join tables.

```
SELECT cols FROM  t1 JOIN t2 ON (condition) JOIN
 t3 ON (condition)
```

- JOIN can be replaced with LEFT OUTER JOIN,RIGHT OUTER JOIN,FULL OUTER JOIN or LEFT SEMI JOIN.

TLS

Join Examples

- Consider two tables:

Customer Table

custid	firstname	surname
1	Jane	Doe
2	Tom	Smith
3	Doug	Jones

Purchases Table

custid	order_date	cost
1	2010-01-15	42.99
2	2010-03-21	19.65
2	2010-04-01	1.00
1	2010-04-18	170.16

Inner Join

- An inner join returns all rows where the condition is satisfied.

```
SELECT * FROM Customer JOIN Purchases
ON (customer.custid = purchases.custid)
```

custid	firstname	surname	custid	order_date	cost
1	Jane	Doe	1	2010-01-15	42.99
1	Jane	Doe	1	2010-04-18	170.16
2	Tom	Smith	2	2010-03-21	19.65
2	Tom	Smith	2	2010-04-01	1.00

Outer join

- A left outer join returns all the rows in the first table even if there are no matches.

```
SELECT C.custid, firstname, surname, order_date
FROM Customer C LEFT OUTER JOIN Purchases P
ON (C.custid = P.custid)
```

custid	firstname	surname	order_date
1	Jane	Doe	2010-01-15
1	Jane	Doe	2010-04-18
2	Tom	Smith	2010-03-21
2	Tom	Smith	2010-04-01
3	Doug	Jones	**NULL**

TLS

Left Semi Joins

- A left semi join is an optimized IN/EXISTS sub-query.
  - Performs a map-side group by to reduce data flowing to reducers
  - Allows for an early exit if match in join

```
SELECT A.*
FROM  A  LEFT SEMI JOIN B
ON (A.KEY = B.KEY AND B.VALUE >100);
```

  - Equivalent IN/EXISTS subquery

```
SELECT A.*
FROM  A  WHERE A.KEY IN
( SELECT A.KEY = B.KEY AND B.VALUE >100);
```

TLS

Identifying Unmatched Records

- Outer joins are useful for finding unmatched records.
- Example: Find customers who have not made purchases.

```
SELECT C.custid, firstname, surname
FROM  Customer C LEFT SEMI JOIN Purchases P
ON (C.custid = P.custid) WHERE P.custid IS NULL;
```

custid	firstname	surname
3	Doug	Jones

TLS

Joining Multiple tables

- Syntax for joining multiple tables

```
SELECT  cols FROM
      t1 JOIN t2 ON (condition)
     JOIN t3 ON (condition)
```

TLS

Basic Hive Functions

Hive Functions

- Hive provides many built-in functions
  - Mathematical
  - Date
  - Conditional
  - String
  - Aggregate functions
- Hive also supports user-defined functions.

TLS

Numeric Functions

- Numeric functions include
  - round()
  - floor()
  - ceil()
  - rand()
  - exp()
  - log()
  - sqrt()
  - abs()
  - sin()
  - cos()
  - etc.

TLS

String Functions

- String functions include
  - length()
  - concat()
  - substr()
  - upper()
  - lower()
  - trim()
  - regexp_replace()
  - etc.

TLS

Date Functions

- Date functions include
  - unix_timestamp()
  - from_unixtime()
  - to_date()
  - year()
  - month()
  - day()
  - date_add()
  - date_sub()
  - datediff()
  - etc.

TLS

Aggregate Functions

- Aggregate functions include
  - sum()
  - avg()
  - min()
  - max()
  - stddev_pop()
  - stddev_sample()
  - percentile()
  - etc.

TLS

Multi-table insert

- Hive SELECT queries can take a long time.
- We sometime need to extract data to a multiple tables based on the same query.
- Hive provides an extension to SQL to do this
  - 'Multi-table insert'
  - Only requires the query to be run once

```
FROM  (SELECT ...)  alias
INSERT  OVERWRITE  TABLE  t1 ...
INSERT  OVERWRITE  TABLE  t2 ...
...
```

TLS

Multi-table insert: Example

- Multi-table insert syntax

```
FROM (SELECT * FROM movies WHERE documentary = 1 ) docs
INSERT OVERWRITE DIRECTORY 'docs_names'
SELECT movie_name
INSERT OVERWRITE DIRECTORY 'docs_count'
SELECT count(1);
```

TLS

Tech Mahindra

More advanced HiveQL features

TLS

Creating tables based on existing data

- Hive supports creating table based on SELECT statement.
  - Often known as ‘Create table as select’, or CTAS

```
CREATE TABLE newtable AS
SELECT col1, col2 FROM existing table
```

- Column definitions are derived from the existing table
- Column names are inherited from the existing names.
  - Use aliases in the SELECT statement to specify new names

TLS

Subqueries

- Hive supports sub-queries in the FROM clause of the SELECT statement.
  - It does not support correlated sub-queries

```
SELECT col FROM
(SELECT col_a + col_b AS col FROM t1 ) t2
```

- The sub-query must be given a name.
  - t2 in the example above
- Hive supports arbitrary levels of sub-queries

Views

- Hive supports Views.
  - Syntax

```
CREATE  VIEW  v (col_names)
AS  SELECT ...
```

- Column name list is optical.
  - Will use column names from the underlying table(s) if omitted
  - Views are not materialized
  - Content changes as the underlying tables change.
- Views cannot be modified directly
  - Cannot be used as the target for e.g., INSERT OVERWRITE

TLS

Views (cont'd)

- Views may contain ORDER BY and LIMIT clauses.
- When a View is used, the underlying query is executed to retrieve the View's contents.
- To remove a View

```
DROP VIEW v
```

TLS

UNION

- Hive supports UNIONS
  - Combine the results from multiple SELECT statements into a single result set
  - The number and names of columns returned by each select statement has to be the same
  - Only UNION ALL is supported
  - i.e. duplicates are not eliminated

select_statement UNION ALL select_statement
UNION ALL select_statement ...

TLS

Tech Mahindra

Statistics and Data Mining

TLS

N-gram Frequency Estimation

- What is an N-gram?
  - A contiguous sequence of n items from a given sequence of text
  - Used to find frequently occurring related items
    - NGRAMS()
    - Find the k most frequent n-grams from one or more sequences
    - CONTEXT_NGRAMS()
    - Expands ngrams() by specifying a ‘context’ string around which n-grams are to be estimated
  - Example
    - Return a list of the top 100 words that follow the phrase "I love" in a hypothetical database of Twitter tweets

TLS

N-gram Frequency Estimation (cont'd)

```
SELECT context_ngrams(sentences(lower(tweet)),
array("I","love", null), 100) FROM twitter;
```

- NGRAMS() Example Use Cases
  - Find important topics in text in conjunction with a stopword list
  - Find trending topics in text
  - Find frequency accessed URL sequences
- CONTEXT_NGRAMS() Example Use Cases
  - Extract marketing intelligence around certain words (e.g., " I love…")
  - Find frequently accessed URL sequences that sort or end at a particular URL
  - Pre-compute common search look ahead

TLS

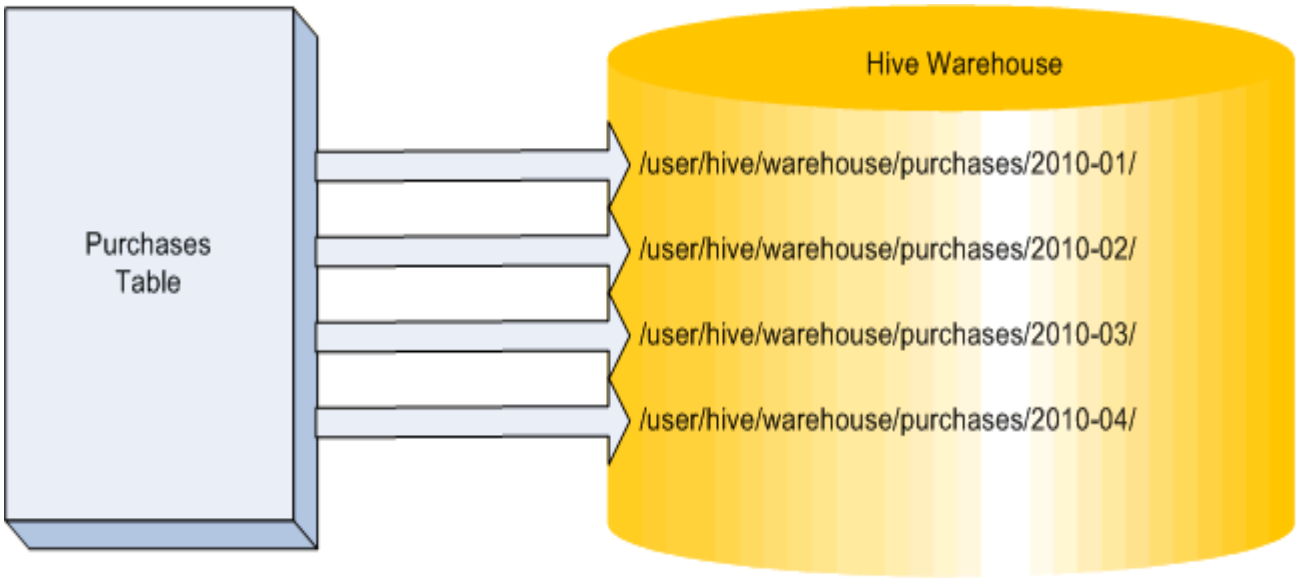
Tech Mahindra

Partitioning and Bucketing

TLS

What is Partitioning?

- Partitioning a dataset means splitting it into smaller portions based on the value in a column.
- This is some times known as ‘horizontal partitioning’.
- Hive partitions are stored in subdirectories of the table directory.



TLS

Why Partitioning?

- Partitioning allows hive to filter at the input path level.
  - Non-matching data is never read
- Splitting a table into partitions is useful if many queries refer to a specific column.
- For e.g. - A table containing log entries
  - Much of the analysis looks at individual days
  - By partitoning the data based on the date column, only a subset of the data needs to be scanned while executing the query
  - Since each partition is stored in a different directory, only the data in the relevant directories needs to be read
  - Any queries which do not specify the date will still work, however in such cases, the entire data set will have to be scanned

TLS

Creating a Partitioned table in Hive

- To create a partitioned table specify a column like this

```
CREATE TABLE t (column_spec)
PARTITIONED BY (column_name  datatype)
ROW FORMAT ...
```

- Example

```
CREATE TABLE logs (uid INT, action STRING)
PARTITIONED BY (dt STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY  '\t'
```

TLS

Creating a Partitioned table in Hive

- The partitioned column is displayed if you DESCRIBE the table

```
hive> DESCRIBE logs;
OK
uid INT
action STRING
dt STRING
```

- However, the partition is a ‘virtual column’
  - The data does not exist in your incoming data
  - Instead, the partition is specified when loading the data

TLS

Loading Data into Partitions

- To load data into a partition in the table, use

```
LOAD DATA INPATH '/path/to/table'
INTO TABLE t
PARTITION (col=val)
```

- Example:

```
LOAD DATA INPATH '/user/ian/mylogs'
INTO TABLE logs
PARTITION (dt='2012-01-15')
```

TLS

Querying a Partitioned Table

- When queries include the partition column in the WHERE clause, Hive only needs to process the relevant subdirectories not the entire data set.

- The filtering is done by the Hive interpreter, using data from Hive's Metastore.
  - Supports =, !=, <, <=, >, >=, and LIKE for strings

  - Conditions can be chained together with AND and OR

TLS

Dynamic Partition Inserts

- If your data already exists in a table, recent versions of Hive can dynamically insert the data into specific partitions for you.

- Syntax:

```
FROM employees
INSERT OVERWRITE TABLE emps_by_dept  PARTITION(dept)
SELECT first_name, last_name, address, phone, dept;
```

- Partitions are automatically created based on the value of the last column.
  - If the partition does not already exist, it will be created
  - If the partition does exist, it will be overwritten

TLS

Dynamic Partition Inserts

- Three Hive configuration properties exist to limit this:
- hive.exec.max.dynamic.partitions.pernode
  - Maximum number of dynamic partitions that can be created by any given Mapper or Reducer
  - Default 100
- hive.exec.max.dynamic.partitions
  - Total number of dynamic partitions that can be created by one.
  - HiveQL statement
  - Default 1000
- hive.exec.max.created.files
  - Maximum total files created by Mappers and Reducers
  - Default 100000

TLS

Sub-Partitions

- A table can have sub-partitions.
- Example

```
CREATE TABLE t (col_spec)
PARTITION(dt STRING, country STRING)
```

- Second partition will be a subdirectory of the first partition's directory.
  - E.g. create a partition year, a sub-partition month and another sub-partition for date
- Multiple partitions can be used during a multi-partition insert.
  - The dynamically defined partition must be the last partition in the list

TLS

Dropping/Adding Partitions

- Hive's ALTER TABLE statement provides the ability to drop or add a partition.
- Adding multiple partitions example

```
ALTER TABLE page_view ADD PARTITION (dt='2008-08-08',
country='us') location '/path/to/us/part080808';

ALTER TABLE page_view ADD PARTITION (dt='2008-08-09',
country='us') location '/path/to/us/part080809';
```

- Removing partition example

```
ALTER TABLE page_view DROP PARTITION
(dt='2008-08-08', country='us');
```

TLS

What is Bucketing?

- Bucketing data is similar to partitioning data.
- Data is split into buckets based on the hash value of the column of the incoming data.
- Intended to produce an even distribution of rows across n buckets.
- Useful for jobs which need to ‘sample’ data from the table.
  - Jobs which just work on a ‘random’ portion of the data

TLS

Creating a bucketed table

- Syntax to create a bucketed table

```
CREATE TABLE t (col_spec)
CLUSTERED BY (col) INTO n BUCKETS
```

TLS

Inserting data into the bucketed table

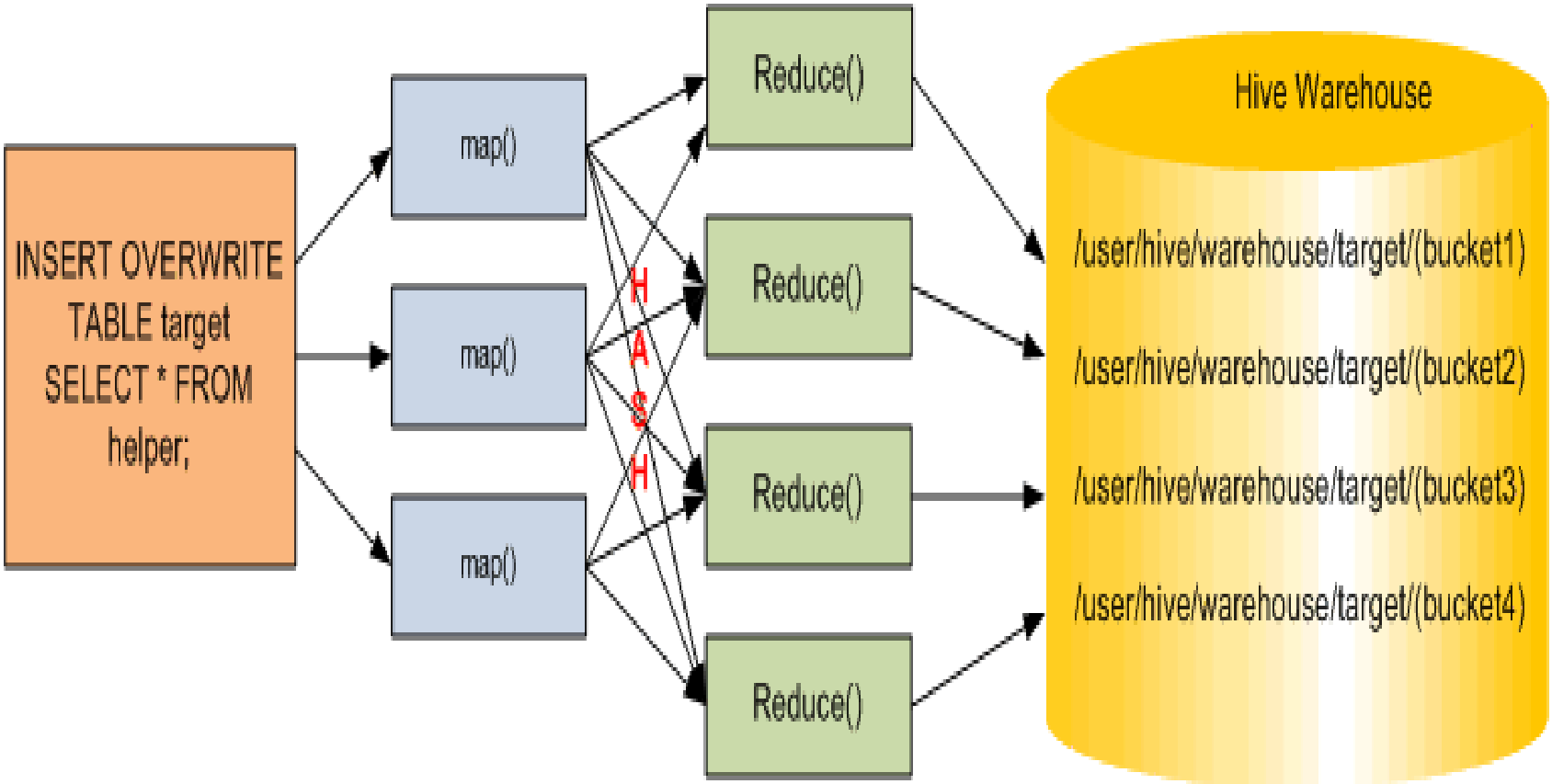
- To insert data into the bucketed column, first insert it into a ‘helper’ table (with no bucketing) and then fire the following commands

```
hive>SET mapred.reduce.tasks=(number-of-buckets)
hive>SET hive.enforce.bucketing=true;
hive>INSERT OVERWRITE TABLE target_bucketed_table
SELECT * FROM helper_table;
```

TLS

Inserting data into the bucketed table

- Hive implements bucketing by running mapReduce job on the data in the helper table.



TLS

Sampling data from a bucketed table

- To use the bucketed table, use the following syntax:

```
SELECT * FROM bucketed_table
TABLESAMPLE (bucket 1 OUT OF 4 ON col)
```

- If the table was bucketed into the 4 buckets on col, this would choose just the rows in bucket 1

- If the table was bucketed into the 16 buckets on col, this would choose the rows in bucket 1, 5, 9, 13

TLS

Tech Mahindra

Hands-On Exercise

TLS

Tech Mahindra

Advanced Hive Features

TLS

Advanced Hive Features

- Hive Variables
- The Hive CLI
- Hive and Thrift
- TRANSFORM
- Creating User-Defined Functions and SerDes
- Debugging and Troubleshooting Hive Queries

TLS

Hive Variables

- The Hive variable substitution mechanism was designed to avoid some of the code that was getting baked into the scripting language on top of Hive.
- Variables can be set in two ways
  - Through Hive shell
  - By invoking Hive from the command line
- Within the Hive shell

```
hive>SET var=val
```

- To use the variable's value in a HiveQL query

```
hive>SELECT * FROM tbl WHERE col=${hiveconf:var};
```

TLS

Hive Variables (cont'd)

- Invoke Hive from the command line.
- This makes repetitive operations easier, without modifying any HiveQL For example, imagine that the following queries in region.hql helping us to find all employees in the north region with this command

```
$ hive –hiveconf var=val

SELECT * FROM employees WHERE region =
                                        '${hiveconf:REGION}';

$ hive -hiveconf REGION=north -f region.hql
```

TLS

Hive Command Line Interface

- Usage: hive [-hiveconf x=y]* [<-i filename>]* [<-f filename>|<-e query-string>]
- -i <filename> Initialization Sql from file (executed automatically and silently before any other commands)
- -e Quoted query string' Sql from command line
- -f <filename> Sql from file
- -v Verbose mode (echo executed SQL to the console)
- -p <port> connect to Hive Server on port number
- -hiveconf x=y Use this to set hive/hadoop configuration variables
- -e and -f Cannot be specified together. In the absence of these options, interactive
- -i Can be used to execute multiple init scripts

TLS

What is Thrift?

- Thrift is a software framework.
  - Allows for scalable cross/language services development
  - complete stack for creating clients and servers
  - Build services that work seamlessly between languages
  - It is used as a remote procedure call (RPC) framework and was developed at Facebook for scalable cross-language services development.

- Allows use of other languages with Hive.
  - E.g., C++, Java, Python, PHP, Ruby, Perl, C#, JavaScript, etc.

TLS

Using TRANSFORM to Process Data Using External Scripts

- Hive allows you to transform data through external scripts or programs.These can be written in nearly any language

- This is done with HiveQL's TRANSFORM ... USING construct.
  - One or more fields are supplied as arguments to TRANSFORM()

  - The external script is identified by USING
    - It receives each record, processes it, and returns the result

- Use ADD FILE to distribute the script to slave nodes in the cluster.

```
hive> ADD FILE myscript.pl;
hive> SELECT TRANSFORM(*) USING 'myscript.pl'
                          FROM employees;
```

TLS

Hive TRANSFORM Example

- Let's look at a complete example of using TRANSFORM in Hive.
  - The perl script parses an e/mail address, determines to which country it corresponds, and then returns an appropriate greeting
  - Here's a sample of the input data

```
hive>SELECT name, email_address FROM employees;
Antoine    antoine@example.fr
Kai        kai@example.de
Pedro      pedro@example.mx
```

TLS

Hive TRANSFORM Example (cont'd)

- Here's the corresponding HiveQL code.

```
hive> ADD FILE greeting.pl;
hive> SELECT  TRANSFORM(name, email_address)
            USING   'greeting.pl' AS greeting FROM employees;
hive>SELECT name,email_address FROM employees;
Antoine    antoine@example.fr
Kai        kai@example.de
Pedro      pedro@example.mx
```

TLS

Hive TRANSFORM Example (cont'd)

- The perl script for this example is shown below.
  - A complete explanation of this script follows on the next few slides.

```
#!/usr/bin/env perl
greetings = ('de' => 'Hallo',
                'fr' => 'Bonjour',
                'mx' => 'Hola');
while (<STDIN>)
   {
          ($name, $email) = split /\t/;
          ($suffix) = $email =~ /\.([a-z]+)/;
          $greeting = $greetings{$suffix};
          $greeting = 'Hello' unless defined($greeting);
          print  $greeting $name\n ;
    }
```

TLS

Hive TRANSFORM Example (cont'd)

- Finally, let's see the result of the transformation

```
hive> ADD FILE greeting.pl;
hive> SELECT  TRANSFORM(name, email_address)
        USING 'greeting.pl' AS greeting
        FROM employees;

Bonjour  Antoine
Hallo  Kai
Hola  Pablo
```

TLS

Transform/MapReduce Typing

- Hive also supports the MAP and REDUCE keywords for external processing.
- Allows to embed custom mapper and reducer scripts
- The input and output formats are the same as with TRANSFORM
- Example usage

```
FROM (
        FROM pv_users
        MAP pv_users.userid, pv_users.date
        USING 'map_script'
        AS dt, uid
        CLUSTER BY dt) map_output
    INSERT OVERWRITE TABLE pv_users_reduced
        REDUCE map_output.dt, map_output.uid
        USING 'reduce_script'
        AS date, count;
```

TLS

Creating User/Defined Functions (UDFs)

- User-defined functions (UDFs) can be used in HiveQL queries in the same way as standard functions.

- UDFs are written in Java.
- Create a class which extends the UDF class

- Class should contain a method named evaluate

- Hadoop supports three forms of UDF
- Standard UDFs

- User/Defined Aggregate Functions (UDAFs)

- Take multiple input values, one per row, return a single value

- User/Defined Table Functions (UDTFs)
  - Take a single row as input, return multiple rows as output

TLS

Creating Custom UDFs

- First, create a new class that extends UDF, with one or more methods named evaluate.

- For example

```
package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
public final class Lower extends UDF {
                        public Text evaluate(final Text s)
                         {
                              if (s == null)
                                    {
                                        return null;
                                     }
                                    return new
Text(s.toString().toLowerCase());
                        }
          }
```

TLS

Deploying the Jar

- Compile the code to a jar and add to the Hive classpath. Then the jar will then be on the classpath for all jobs initiated from that session

```
hive> add jar my_jar.jar;
hive> list jars;
```

- The final step is to register the function.

```
hive> create temporary function my_lower as
                        'com.example.hive.udf.Lower';
```

TLS

Calling UDFs in Hive

- The UDF can be used in HiveQL

```
hive> SELECT  my_lower(title), sum(freq) FROM titles
            GROUP BY my_lower(title);
                          ...
Ended Job = job_200906231019_0006
OK
cmo 13.0
vp 7.0
```

TLS

Debugging Hive Queries

- Troubleshooting Hive queries can be difficult.
- The Hive interpreter turns the query into a set of MapReduce jobs
- Those jobs are run on the cluster
- It can be difficult to determine what part of the query caused the problem
- Hadoop's JobTracker Web UI displays the progress of MapReduce jobs on the cluster
- Accessed at http://job_tracker_address:50030

TLS

Thank You

Disclaimer

Tech Mahindra Limited, herein referred to as TechM provide a wide array of presentations and reports, with the contributions of various professionals. These presentations and reports are for informational purposes and private circulation only and do not constitute an offer to buy or sell any securities mentioned therein. They do not purport to be a complete description of the markets conditions or developments referred to in the material. While utmost care has been taken in preparing the above, we claim no responsibility for their accuracy. We shall not be liable for any direct or indirect losses arising from the use thereof and the viewers are requested to use the information contained herein at their own risk. These presentations and reports should not be reproduced, re-circulated, published in any media, website or otherwise, in any form or manner, in part or as a whole, without the express consent in writing of TechM or its subsidiaries. Any unauthorized use, disclosure or public dissemination of information contained herein is prohibited. Unless specifically noted, TechM is not responsible for the content of these presentations and/or the opinions of the presenters. Individual situations and local practices and standards may vary, so viewers and others utilizing information contained within a presentation are free to adopt differing standards and approaches as they see fit. You may not repackage or sell the presentation. Products and names mentioned in materials or presentations are the property of their respective owners and the mention of them does not constitute an endorsement by TechM. Information contained in a presentation hosted or promoted by TechM is provided “as is” without warranty of any kind, either expressed or implied, including any warranty of merchantability or fitness for a particular purpose. TechM assumes no liability or responsibility for the contents of a presentation or the opinions expressed by the presenters. All expressions of opinion are subject to change without notice.

TLS

