# Apache Hive

BAS Academy

# Agenda

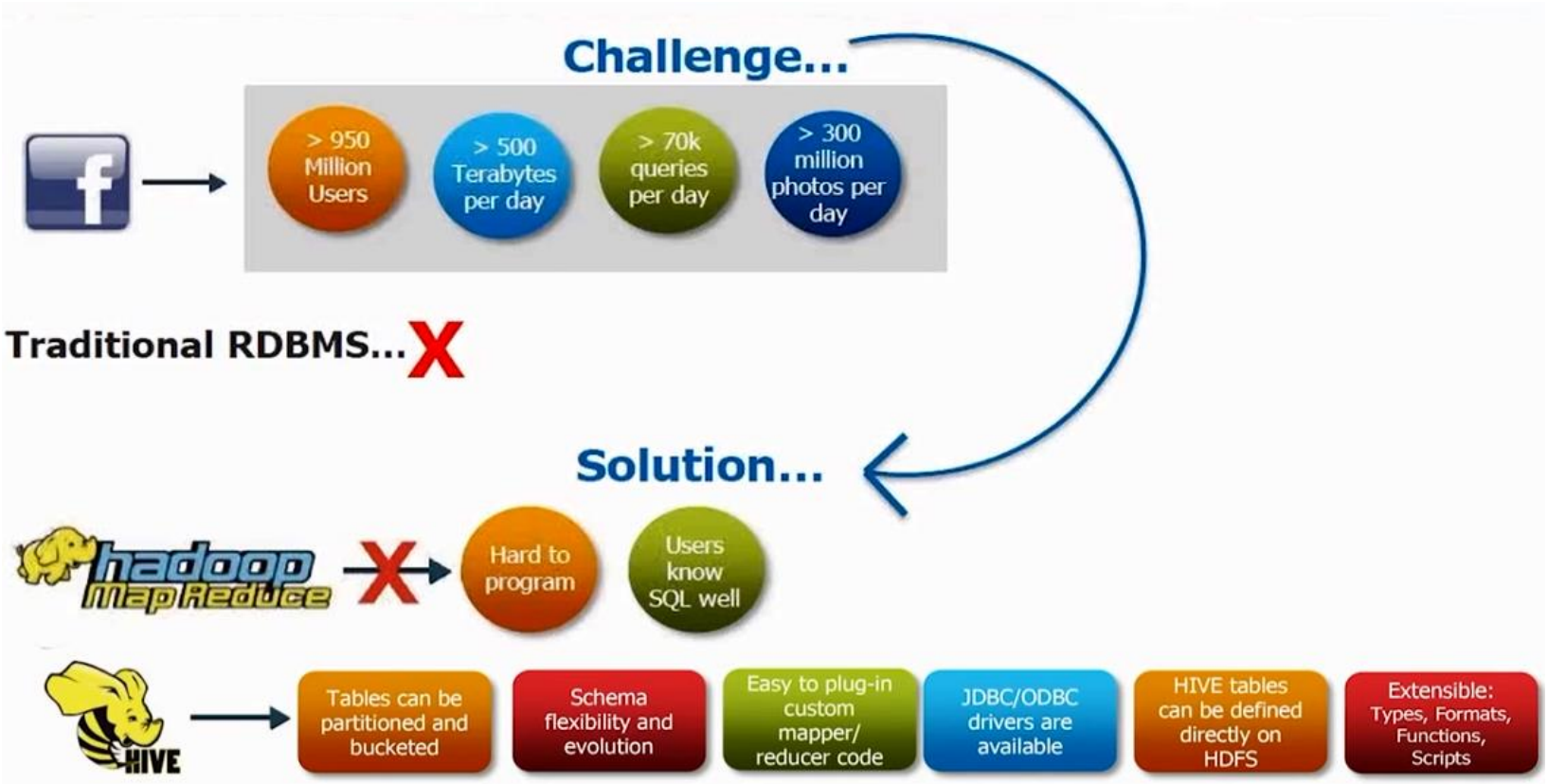
- About Features
- Hive Architecture
- Hive Operations
- Hive Operators and Functions
- Hive Advanced
- Hands On

# Hive Features

# What is Hive

- Data warehousing package built on top of Hadoop
- Used for Data Analysis
- SQL like scripting language called HiveQL
- Targeted to users comfortable with SQL
- Developed by Facebook  in 2007 and took over by Apache

# Hive Background



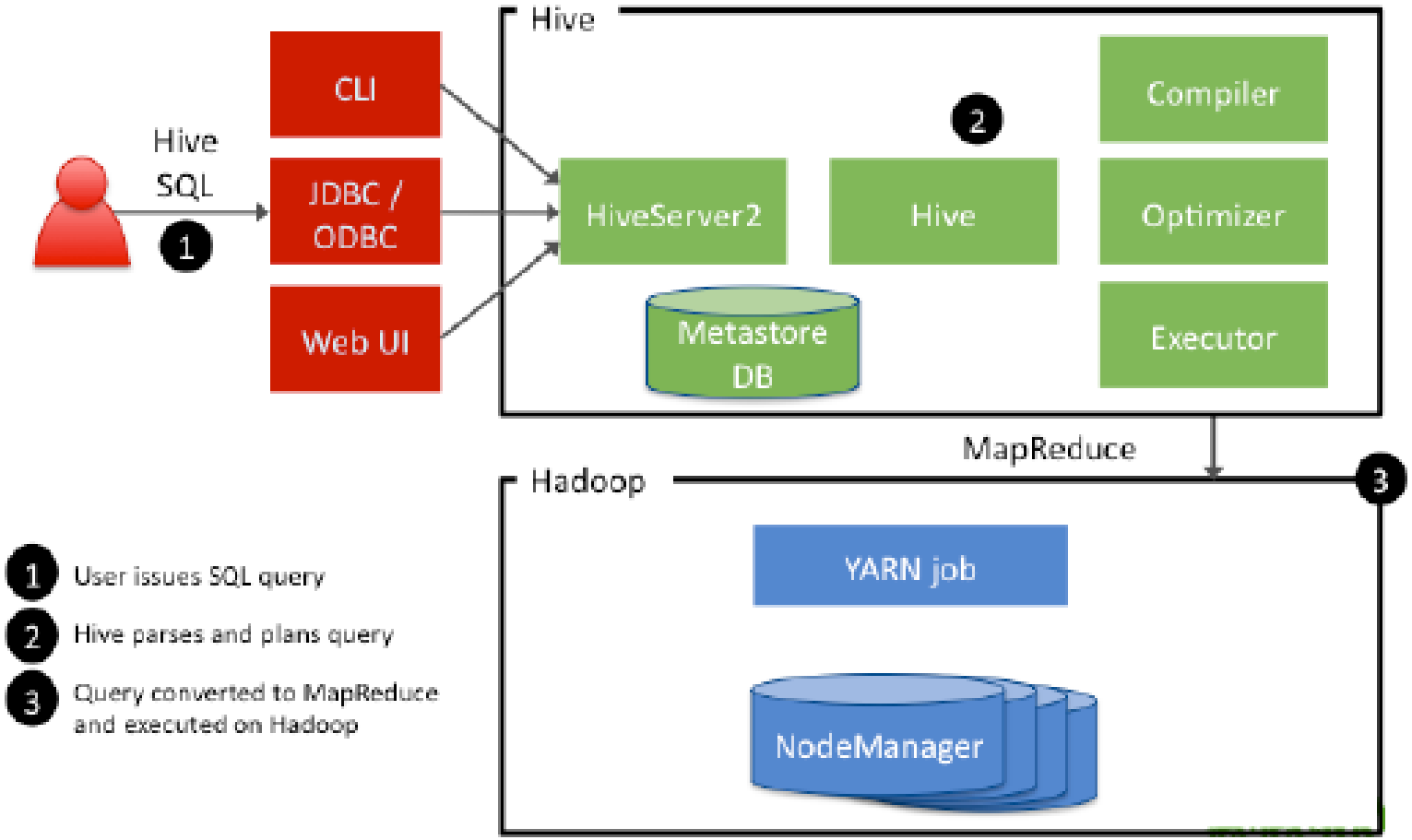

- Summarization: Daily or weekly aggregations of click counts
- Adhoc: How many group admins by country/state

# Hive Features

- Hive is not a database, but uses a database called metastore to store the tables
- Metadata has tables, databases, columns in a table, their data types, and HDFS mapping.
- It uses Derby DB by default as Metastore and it is configurable
- Hive table consists of schema stored in the metastore and process the data stored on HDFS
- Ability to bring structure to various data formats
- Hive converts HiveQL commands into MapReduce jobs.

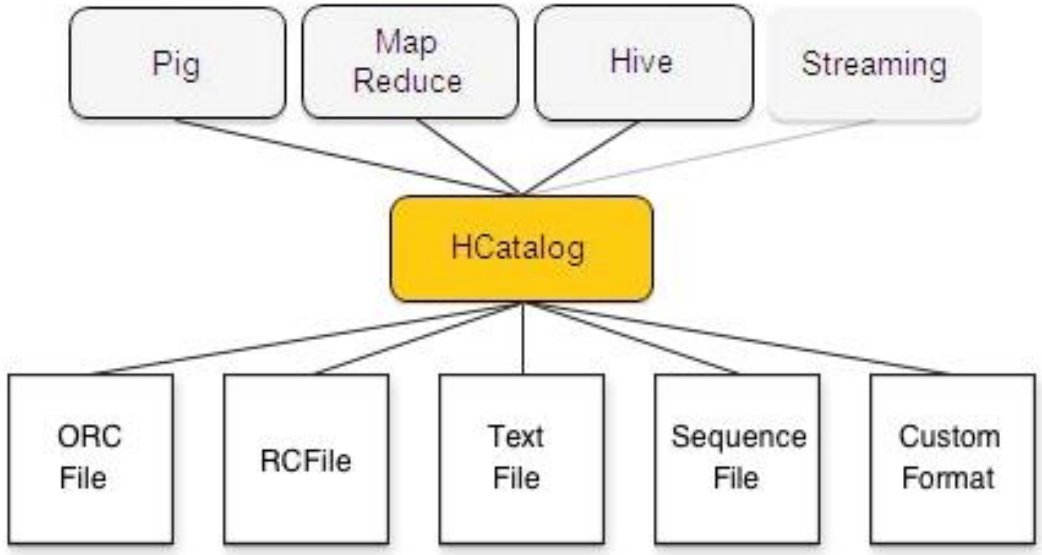
# Hive Architecture

# Hive Architecture



# HCatalog

- HCatalog is designed to easily store and share schemas for your big data.
- HCatalog store its schema information in Hive Metastore
- Allows users to share data and metadata across Hive, Pig, and MapReduce



# Hive Vs Pig

Pig and Hive work well together
and many businesses use both.


Hive is a good choice:

- when you want to query the data
- when you need an answer to specific questions
- if you are familiar with SQL

Pig is a good choice:

- for ETL (Extract -> Transform -> Load)
- for preparing data for easier analysis
- when you have a long series of steps to perform

- Pig is used to reformat unstructured data to define a structure to it
- Hive is used to query the data that has certain structure to it.

- Pig is used for programmers and researchers
- Hive is used for Analysts

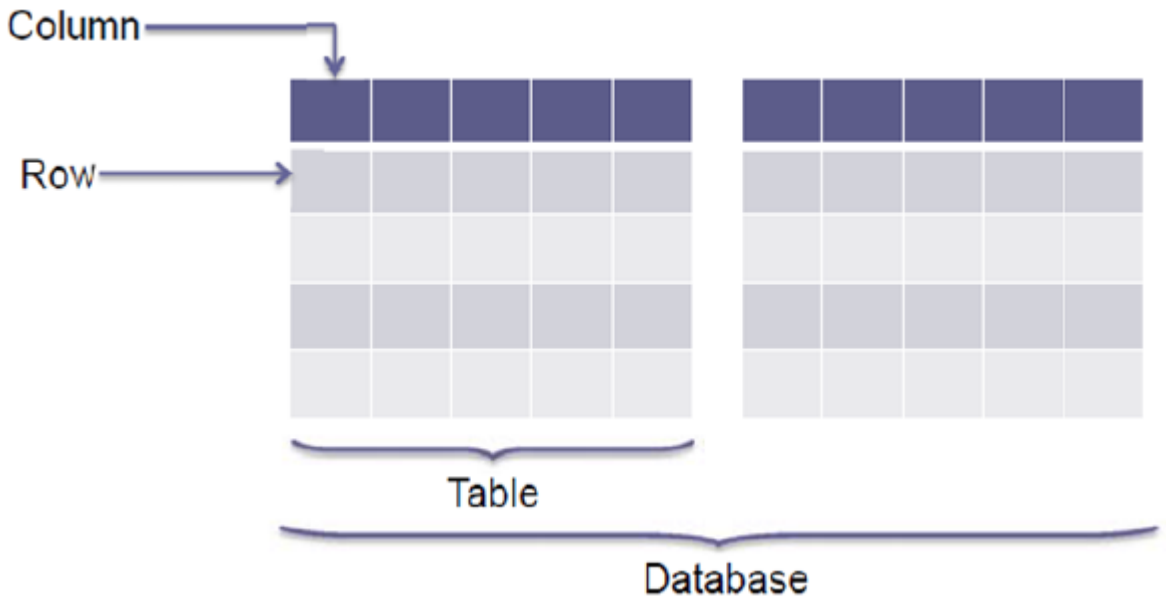
# Hive Limitations

- Hive is not for On Line Transaction Processing (OLTP)
- Hive does not provide low latency response
- Hive is designed for scalability and ease of use rather than low latency response
- The data in Hive is read only (no update can be done)

# Hive Concepts

- Hive table is used to add structure to the unstructured data in HDFS

- All HDFS and Unix commands can be used in Hive



# Hive Concepts – Data Types

## INTEGRAL DATA TYPES

- TINYINT (This TINYINT is equal to Java's BYTE data type)
- SMALLINT (This SMALLINT is equal to Java's SHORT data type)
- INT (This INT is equal to Java's INT data type)
- BIGINT (This BIGINTis equal to Java's LONG data type)

## FLOATING DATA TYPES

- FLOAT (This FLOAT is equal to Java's FLOAT data type )
- DOUBLE (This DOUBLE is equal to Java's DOUBLE data type)
- DECIMAL (This DECIMAL is equal to SQL's DECIMAL data type)

## STRING DATA TYPES

In Hive String Data Types are Mainly divided into 3 types there are mentioned below

- STRING
- VARCHAR
- CHAR

## MISCELLANEOUS TYPES

Hive Supports 2 more primitive Data types

- BOOLEAN
- BINARY

## DATE/TIME DATA TYPES

Date/Time Data types are mainly Divide into 2 types

- DATE
- TIMESTAMP

# Complex Data Types

There are three complex types in hive

**Array:**

- It is an ordered collection of elements of same type.
- These elements are accessed by using an index.
- Ex: Country, containing a list of elements [‘US’, ’UK’, ‘FR’], the element “US” in the array can be accessed by specifying Country[1].

**Map:**

- It is an unordered collection of key-value pairs.
- The elements are accessed by using the keys
- Ex: pass_list, containing the “user name” as key and “password” as value, the password of the user can be accessed by specifying pass_list[‘username’]

**Struct:**

- It is a collection of elements of different types.
- The elements can be accessed by using the dot notation.
- Ex: Employee ,containing a list of elements [‘Name’, ’Age’, ‘Address’], the element age of the employee can be retrieved as specifying employee.age

# Hive Operations

# Hive Operations- Creating Table

Create Table

- To create a table in Hive.

**Syntax**

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]

Row Format

- Use DELIMITED clause to read delimited files
- Use SERDE clause to read binary files

- An error is thrown if a table or view with the same name already exists. You can use IF NOT EXISTS to skip the error.

# Hive File Formats

Table names and column names are case insensitive

Use STORED AS

- TEXTFILE if the data needs to be stored as plain text files.
- SEQUENCEFILE if the data needs to be compressed (Serialized key/value pairs)
- RCFILE (Record Columnar file) that organizes data by columns
- ORCFILE optimized row columnar format to improves the efficiency of Hive

# Creating Table - Example

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 10.606 seconds
```

1st line: creates a table with 3 columns
2nd and 3rd line: how the underlying file should be parsed
4th line: how to store data

Statements must end with a semicolon and can span multiple rows

```
hive> show tables;
OK
posts
Time taken: 0.221 seconds
```

Display all of the tables

Result is displayed between "OK" and "Time taken..."

```
hive> describe posts;
OK
user    string
post    string
time    bigint
Time taken: 0.212 seconds
```

Display schema for `posts` table

# External Table

Use EXTERNAL tables when:

- The data is also used outside of Hive.
- Data needs to remain in the underlying location even after a DROP TABLE

Location:

Default: /apps/hive/warehouse

Any other location in HDFS can be defined by using LOCATION clause

Ex: CREATE EXTERNAL TABLE salaries ( gender string, age int, salary double, zip int )

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/user/train/salaries/';

# Hive Operations– Loading Table

Load Table

- The data for a Hive table resides in HDFS. To associate data with a table, use the LOAD DATA command.

There are two ways to load data:

- local file system
- Hadoop file system

**Syntax**

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

Ex: LOAD DATA LOCAL INPATH '/tmp/employees.csv' OVERWRITE INTO TABLE employee;

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

# Loading Data Options

- Several options to start using data in Hive
  - Load data from HDFS location

```
hive> LOAD DATA INPATH '/training/hive/user-posts.txt'
    > OVERWRITE INTO TABLE posts;
```

    - ✓ File is moved from the provided location to /user/hive/warehouse/ (or configured location)
  - Load data from Local file system

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'
     > OVERWRITE INTO TABLE posts;
```

    - ✓ File is copied from the provided location to /user/hive/warehouse/ (or configured location)
  - Utilize an existing location on HDFS
    - ✓ Just point to an existing location when creating a table

# Create Table and Load HDFS Data

- Create will create table directory in user/hive/warehouse
- Load will move the file from hdfs location to warehouse directory. So the file will not be available in hdfs location
- Drop will delete the table schema and the warehouse table directory.
- After drop, the file will not be available in hdfs location and warehouse location

# Create Table and Load Local Data

- Create will create table directory in user/hive/warehouse
- Load will copy the file from local location to warehouse directory. So the file will be available in local system
- Drop will delete the table schema and the warehouse table directory. But the file in local system will still be there.
- After drop, the file will be available in local system but not in warehouse location

# Create External Table and Load HDFS Data

- Create will create table directory in user/hive/warehouse  location or any mentioned location
- Load will move the file from hdfs location to warehouse/mentioned directory. So the file will not be available in hdfs location
- Drop will delete the table schema but the file in warehouse/ mentioned directory will still be there
- After drop, the file will not be available in hdfs location but will be available in warehouse/mentioned location

# Create External Table and Load Local Data

- Create will create table directory in user/hive/warehouse location or any mentioned location
- Load will copy the file from local location to warehouse/mentioned directory. So the file will be available in local system
- Drop will delete the table schema but the file in warehouse/mentioned location will still be available. Also, file is available in local system
- After drop, the file will be available in local system and in warehouse /mentioned location

# Schema Violations

- What would happen if we try to insert data that does not comply with the pre-defined schema?

```
hive> !cat data/user-posts-inconsistentFormat.txt;
user1,Funny Story,1343182026191
user2,Cool Deal,2012-01-05
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

hive> describe posts;
OK
user  string
post  string
time  bigint
Time taken: 0.289 seconds
```

Third Column 'post' is of type bigint; will not be able to convert '2012-01-05' value

# Schema Violations

```
hive> LOAD DATA LOCAL INPATH
        > 'data/user-posts-inconsistentFormat.txt'
    > OVERWRITE INTO TABLE posts;
OK
Time taken: 0.612 seconds

hive> select * from posts;
OK
user1  Funny Story   1343182026191
user2  Cool Deal     NULL
user4  Interesting Post    1343182154633
user5  Yet Another Blog    13431839394
Time taken: 0.136 seconds
hive>
```

null is set for any value that violates pre-defined schema

# Hive Operations – Alter Table

Alter Table

- It is used to alter a table in Hive.

**Syntax**

- ALTER TABLE name RENAME TO new_name
- ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
- ALTER TABLE name CHANGE column_name new_name new_type
- ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

# Hive Operations – Select from Table

Select Table

- To display the records from table

**Syntax**

- SELECT * FROM TABLE WHERE condition LIMIT value;

Ex:

FROM customers

SELECT firstName, lastName, address, zip

WHERE orderID > 0

ORDER BY zip;

The FROM clause in Hive can appear before or after the SELECT clause.

# Hive Operations – Drop Table

Drop Table

- When you drop a table from Hive Metastore, it removes the table/column data and their metadata.

**Syntax**

- DROP TABLE [IF EXISTS] table_name;

Ex:

DROP TABLE posts;

# Hive Operators and Functions

# Hive – Built in Operators

## Relational Operators

These operators are used to compare two operands. The following table describes the relational operators available in Hive:

| Operator | Operand | Description |
|---|---|---|
| A = B | all primitive types | TRUE if expression A is equivalent to expression B otherwise FALSE. |
| A != B | all primitive | TRUE if expression A is not equivalent to |

## Arithmetic Operators

These operators support various common arithmetic operations on the operands. All of them return number types. The following table describes the arithmetic operators available in Hive:

| Operators | Operand | Description |
|---|---|---|
| A + B | all number types | Gives the result of adding A and B. |
| A - B | all number types | Gives the result of subtracting B from A. |

## Logical Operators

The operators are logical expressions. All of them return either TRUE or FALSE.

| Operators | Operands | Description |
|---|---|---|
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE. |
| A && B | boolean | Same as A AND B. |

## Complex Operators

These operators provide an expression to access the elements of Complex Types.

| Operator | Operand | Description |
|---|---|---|
| A[n] | A is an Array and n is an int | It returns the nth element in the array A. The first element has index 0. |
| M[key] | M is a Map<K, V> and key has type K | It returns the value corresponding to the key in the map. |
| S.x | S is a struct | It returns the x field of S. |

# Hive – Built in Functions

- SHOW FUNCTIONS

Lists Hive functions and operators

- DESCRIBE FUNCTION [function name]

Displays short description of the function

- DESCRIBE FUNCTION EXTENDED [function name]

Access extended description of the function

Hive supports following build in functions:

- Mathematical Functions
- Type Conversion Functions
- Date Functions
- Conditional Functions
- String Functions

# Hive – Built in Functions

## Mathematical Functions

| Return Type | Name(Signature) | Description |
|---|---|---|
| double | round(double a) | Returns the rounded BIGINT value of the double |
| double | round(double a, int d) | Returns the double rounded to d decimal places |
| bigint | floor(double a) | Returns the maximum BIGINT value that is equal or less than the double |

## Type Conversion Functions

| Return Type | Name (Signature) | Description |
|---|---|---|
| binary | binary (string\|binary) | Casts the parameter into a binary |
| Expected "=" to follow "type" | cast(expr as <type>) | Converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) will convert the string '1' to it integral representation. A null is returned if the conversion does not succeed. |

## String Functions

| Return Type | Name(Signature) | Description |
|---|---|---|
| int | ascii(string str) | Returns the numeric value of the first character of str |
| string | concat(string\|binary A, string\|binary B...) | Returns the string or bytes resulting from concatenating the strings or bytes passed in as parameters in order. e.g. concat ('foo', 'bar') results in 'foobar'. Note that this function can take any number of input strings. |

## Date Functions

| Return Type | Name(Signature) | Description |
|---|---|---|
| string | from_unixtime(bigint unixtime[, string format]) | Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00" |
| bigint | unix_timestamp() | Gets current time stamp using the default time zone. |
| bigint | unix_timestamp(string date) | Converts time string in format `yyyy-MM-dd HH:mm:ss` to Unix time stamp, return 0 if fail: unix_timestamp('2009-03-20 11:30:01') = 1237573801 |

## Conditional Functions

| Return Type | Name(Signature) | Description |
|---|---|---|
| T | if(boolean testCondition, T valueTrue, T valueFalseOrNull) | Return valueTrue when testCondition is true, returns valueFalseOrNull otherwise |
| T | COALESCE(T v1, T v2, ...) | Return the first v that is not NULL, or NULL if all v's are NULL |
| T | CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END | When a = b, returns c; when a = d, return e; else return f |
| T | CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END | When a = true, returns b; when c = true, return d; else return e |

# Hive Advanced

# Insert Statement

- INSERT OVERWRITE will overwrite any existing data in the table.

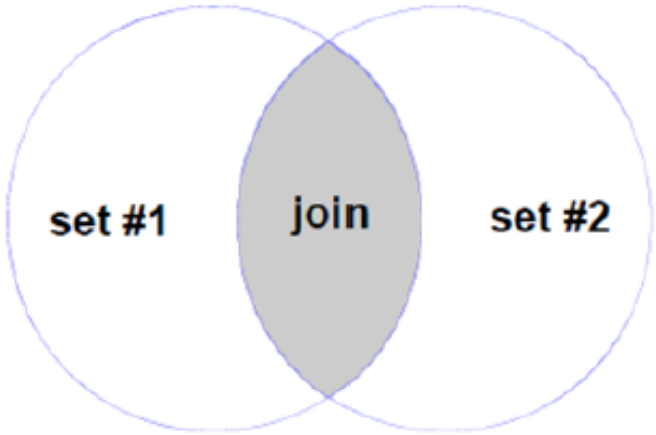
```
INSERT OVERWRITE TABLE likes_new
SELECT * FROM  likes WHERE liekr>10;
```

- INSERT INTO will append to the table, keeping the existing data intact.

```
INSERT INTO TABLE likes_new
SELECT * FROM likes LIMIT 2;
```

# Hive Joins

- Can Join multiple tables
- Hive supports
  - Inner, Outer, Full Join
- Default Join is Inner Join
  - Rows are joined where the keys match
  - Rows that do not have matches are not included in the result



# Hive - Simple Inner Join

- Let say we have 2 tables posts and likes

```
hive> select * from posts limit 10;
OK
user1    Funny Story       1343182026191
user2    Cool Deal         1343182133839
user4    Interesting Post  1343182154633
user5    Yet Another Blog  1343183939434
Time taken: 0.108 seconds
hive> select * from likes limit 10;
OK
user1    12      1343182026191
user2    7       1343182139394
user3    0       1343182154633
user4    50      1343182147364
Time taken: 0.103 seconds
hive> CREATE TABLE posts_likes (user STRING, post STRING, likes_count INT);
OK
Time taken: 0.06 seconds
```

We want to join these 2 data-sets and produce a single table that contains user, post and count of likes

# Simple Inner Join...

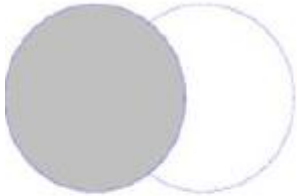
```
hive> INSERT OVERWRITE TABLE posts_likes
    > SELECT p.user, p.post, l.count
    > FROM posts p JOIN likes l ON (p.user = l.user);
OK
Time taken: 17.901 seconds
```

Two tables are joined based on user column; 3 columns are selected and stored in posts_likes table

```
hive> select * from posts_likes limit 10;
OK
user1 Funny Story       12
user2 Cool Deal         7
user4 Interesting Post  50
Time taken: 0.082 seconds
hive>
```

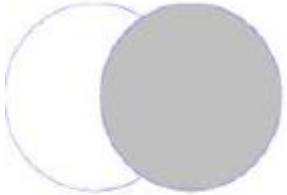
# Hive - Outer Join

- Rows which will not join with the 'other' table are still included in the result



**Left Outer**

- Row from the first table are included whether they have a match or not. Columns from the unmatched (second) table are set to null.



**Right Outer**

- The opposite of Left Outer Join: Rows from the second table are included no matter what. Columns from the unmatched (first) table are set to null.

**Full Outer**

- Rows from both sides are included. For unmatched rows the columns from the 'other' table are set to null.

# Left Outer Join

```
SELECT p.*, l.*
FROM posts p LEFT OUTER JOIN likes l ON (p.user = l.user)
limit 10;
```

```
OK
user1   Funny Story     1343182026191   user1   12      1343182026191
user2   Cool Deal       1343182133839   user2   7       1343182139394
user4   Interesting Post        1343182154633   user4   50
        1343182147364
user5   Yet Another Blog        13431839394     NULL    NULL    NULL
Time taken: 28.317 seconds
```

# Right Outer Join

```
SELECT p.*, l.*
FROM posts p RIGHT OUTER JOIN likes l ON (p.user = l.user)
limit 10;
```

```
OK
user1   Funny Story     1343182026191   user1   12      1343182026191
user2   Cool Deal       1343182133839   user2   7       1343182139394
NULL    NULL    NULL    user3   0       1343182154633
user4   Interesting Post        1343182154633   user4   50
        1343182147364
Time taken: 24.775 seconds
```

# Full Outer Join

```
SELECT p.*, l.*
FROM posts p FULL OUTER JOIN likes l ON (p.user = l.user)
limit 10;
```

```
OK
user1   Funny Story     1343182026191   user1   12      1343182026191
user2   Cool Deal       1343182133839   user2   7       1343182139394
NULL    NULL    NULL    user3   0       1343182154633
user4   Interesting Post        1343182154633   user4   50
        1343182147364
user5   Yet Another Blog        13431839394     NULL    NULL    NULL
Time taken: 24.229 seconds
```

# Hive Partitions

- Underlying data in files will be partitioned by a specified column in the table.
- The values of partition columns divide a table into segments
- It improves the performance as the data is separated into files
- Each partition can be ignored at run time
- Partition can done on multiple columns

```
create table employees (id int, name string, salary double)
partitioned by (dept string);
```

This will result in each department having its own subfolder in the underlying `warehouse` folder for the table:

```
/apps/hive/warehouse/employees
      /dept=hr/
      /dept=support/
      /dept=engineering/
      /dept=training/
```

# Hive – Create Partition Table

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
    > PARTITIONED BY(country STRING)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.116 seconds

hive> describe posts;
OK
user    string
post    string
time    bigint
countrystring
Time taken: 0.111 seconds

hive> show partitions posts;
OK
Time taken: 0.102 seconds
hive>
```

Partition table based on the value of a country.

There is no difference in schema between "partition" columns and "data" columns

# Hive – Load Partition Table

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
    > OVERWRITE INTO TABLE posts;
FAILED: Error in semantic analysis: Need to specify partition
columns because the destination table is partitioned
```

Since the posts table was defined to be partitioned any insert statement must specify the partition

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
    > OVERWRITE INTO TABLE posts PARTITION(country='US');
OK
Time taken: 0.225 seconds

hive> LOAD DATA LOCAL INPATH 'data/user-posts-AUSTRALIA.txt'
    > OVERWRITE INTO TABLE posts PARTITION(country='AUSTRALIA');
OK
Time taken: 0.236 seconds
hive>
```

Each file is loaded into separate partition; data is separated by country

# Partitioned Table Storage

- Partitions are physically stored under separate directories

```
hive> show partitions posts;
OK
country=AUSTRALIA
country=US
Time taken: 0.095 seconds
hive> exit;

$ hdfs dfs -ls -R /user/hive/warehouse/posts
   /user/hive/warehouse/posts/country=AUSTRALIA
   /user/hive/warehouse/posts/country=AUSTRALIA/user-posts-AUSTRALIA.txt
   /user/hive/warehouse/posts/country=US
   /user/hive/warehouse/posts/country=US/user-posts-US.txt
```

There is a directory for each partition value

# Querying Partitioned Table

- There is no difference in Syntax
- When partitioned column is specified in the where clause entire directories/partitions could be ignored

Only "COUNTRY=US" partition will be queried,
"COUNTRY=AUSTRALIA" partition will be ignored

```
hive> select * from posts where country='US' limit 10;
OK
user1 Funny Story 1343182026191     US
user2 Cool Deal   1343182133839     US
user2 Great Interesting Note  13431821339485    US
user4 Interesting Post  1343182154633     US
user1 Humor is good     1343182039586     US
user2 Hi I am user #2   1343182133839     US
Time taken: 0.197 seconds
```

# Dynamic Partitions

- Used when the values for partition columns are known only during loading of the data into a Hive table
- Hive automatically takes care of updating the Hive metastore when using dynamic partitions.
- Table creation semantics are the same for both static and dynamic partitioning.
- In order to detect the values for partition columns automatically, partition columns must be specified at the end of the “SELECT” clause in the same order as they are specified in the “PARTITIONED BY” clause while creating the table.

Do the following settings. Dynamic partitioning is disabled by default

- SET hive.exec.dynamic.partition.mode=nonstrict;
- SET hive.exec.dynamic.partition=true;

# Example - Dynamic Partitions

```
CREATE TABLE partitioned_user(
    firstname VARCHAR(64),
    lastname  VARCHAR(64),
    address   STRING,
    city      VARCHAR(64),
    post      STRING,
    phone1    VARCHAR(64),
    phone2    STRING,
    email     STRING,
    web       STRING
    )
    PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))
    STORED AS SEQUENCEFILE;
```

```
INSERT INTO TABLE partitioned_user
    PARTITION (country, state)
        SELECT  firstname ,
        lastname  ,
        address   ,
    city      ,
        post      ,
        phone1    ,
        phone2    ,
        email     ,
        web       ,
        country   ,
    state
    FROM temp_user;
```

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
```

```
hive> SHOW PARTITIONS partitioned_user;
OK
country=AU/state=AC
country=AU/state=NS
country=AU/state=NT
country=AU/state=QL
country=AU/state=SA
country=AU/state=TA
country=AU/state=VI
country=AU/state=WA
country=CA/state=AB
country=CA/state=BC
country=CA/state=MB
```

# Hive - Buckets

- Bucketing decomposes data into more manageable or equal parts
- Buckets are created using the clustered by clause.
- It can be done with or without partitioning
- Each bucket is a file in the table directory

Ex: Following table has16 buckets that are clustered by the id column

CREATE TABLE employees (id int, name string, salary double)

CLUSTERED BY (id) INTO 16 BUCKETS;

The bucket number is determined by the expression or calculation based on average partition size and memory available

hash_function(bucketing_column) mod num_buckets.

# Hive – Skewed Tables

- Skew refers to one or more columns in a table that have values that appear very often (Heavy Skew).

```
CREATE TABLE Customers ( id int, username string, zip int )
SKEWED BY (zip) ON (57701, 57702)
STORED as DIRECTORIES;
```

- In the Customers table above, records with a zip of 57701or 57702 will be stored in separate files because the assumption is that there will be a large number of customers in those two ZIPcodes.

# Hive Scripts

- **Hive scripts** are used to execute a set of **Hive** commands collectively
- Reusability of code

```
create table product ( productid int, productname string, price float,
category string ) row format delimited fields terminated by ',';

describe product;

load data local inpath '/home/cloudera/input.txt' into table product;

select * from product;
```

- Execute the hive script using the following command:

Command: **hive -f /home/cloudera/sample.hql**

# Hive UDF

- Similar to Pig, Hive has the capability to use User Defined Function written in Java to perform computations

To invoke a UDF from within a Hive script, you need to:

- Register the `JAR` file that contains the UDF class and
- Define an alias for the function using the `CREATE TEMPORARY FUNCTION` command.

For example, the following Hive commands demonstrate how to invoke the `ComputeShipping` UDF defined above:

```
ADD JAR /myapp/lib/myhiveudfs.jar;
CREATE TEMPORARY FUNCTION ComputeShipping
       AS 'hiveudfs.ComputeShipping';
FROM orders SELECT address, description, ComputeShipping(zip, weight);
```

# Hive Tables Export to Files

- Hive table values can be exported to files

**Using Commandline:**

- $hive -e 'select * from myTable' > MyResultsFile.txt
- $hive -e 'select books from myTable' > /home/cloudera/file1.tsv –tab separated files
- Specify the property set hive.cli.print.header=true to export along with headers

**Using INSERT OVERWRITE:**

```
INSERT OVERWRITE LOCAL DIRECTORY '/home/cloudera/staging'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY  ','
SELECT * from  myTable;
```

# Hands On

# Hive – Hands On

- Source File
  - File Name: apat63_99.txt
  - File Content: PatentID, Year, Applicable Year, Country, State

Apat63_99.txt – data description

- PATENT - integer
- GYEAR - integer
- GDATE - integer
- APPYEAR - integer
- COUNTRY - string
- POSTATE - string
- ASSIGNEE - string
- ASSCODE - integer
- CLAIMS - integer
- NCLASS - integer
- CAT - integer
- SUBCAT - integer
- CMADE - integer
- CRECEIVE - integer
- RATIOCIT - integer
- GENERAL - integer
- ORIGINAL - integer
- FWDAPLAG - float
- BCKGTLAG - float
- SELFCTUB - float
- SELFCTLB - float
- SECDUPBD - float
- SECDLWBD - float

- Create the table structure
- Name the table as <name>_apat63_99
- Load data into table from apat63_99.txt
- Find the total no. of patents granted to US by state

# Thank You

Keerthiga Barathan