Session 2 – Hive

File Format

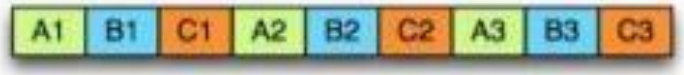
Columnar Format

- Better Compression as data is more homogeneous
- We can efficiently scan only a subset of columns while reading the data
- Works well for queries that only access a small subset of columns
- Format Ex: Optimized Row Columnar (ORC) and Parquet

To illustrate what columnar storage is all about, here is an example with three columns.

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

In a row-oriented storage, the data is laid out one row at a time as follows:



Whereas in a column-oriented storage, it is laid out one column at a time:

File Format

- Standard file format consists of Text files, CSV, XML or binary file types (such as images).
- SequenceFiles store data as binary key-value pairs
- SerDe is short for serializer/deserializer and refers to how records read in from a table (deserialized) and written back out to HDFS (serialized).

- Avro - write-heavy
  - Row-based storage format
  - Contains its own schema and schema evolution
  - Amenable to "full-table scans"
- Parquet - read-heavy
  - Columnar storage formats, sometimes used for final storage
  - Smaller disk-reads
  - Great for feature selection
- ORC - read-heavy
  - Defaults to Zlib compression
  - Mixed row-column, splittable

Compression

- Compression is important consideration for storing data in Hadoop for reducing storage requirements and improving data processing performance
- Splittability is a major consideration in choosing a compression format as well as file format
- Any compression format can be made splittable when used with container file formats like Avro, SequenceFiles

Format	Strengths	Weakness
Snappy	• Developed at Google for high compression speeds	• Doesn't offer the best compression sizes • Relatively slow in decompression • Non-Splittable
LZO	• Rapid compression • Balanced compression and decompression times	• Doesn't offer the best compression sizes • Non-Splittable - can be made splittable by adding additional indexing step • LZO's license requires separate install to be distributed within hadoop
Gzip	• Good compression performance • Reasonable speed • Smaller blocks with Gzip can lead to better performance	• Relatively slower in write speed than snappy and LZ4 • Non-Splittable
bzip2	• Excellent compression performance • Splittable	• bzip2 is about 10 times slower than Gzip
LZ4	• Quick Compression • Best results in decompression	• Non-Splittable

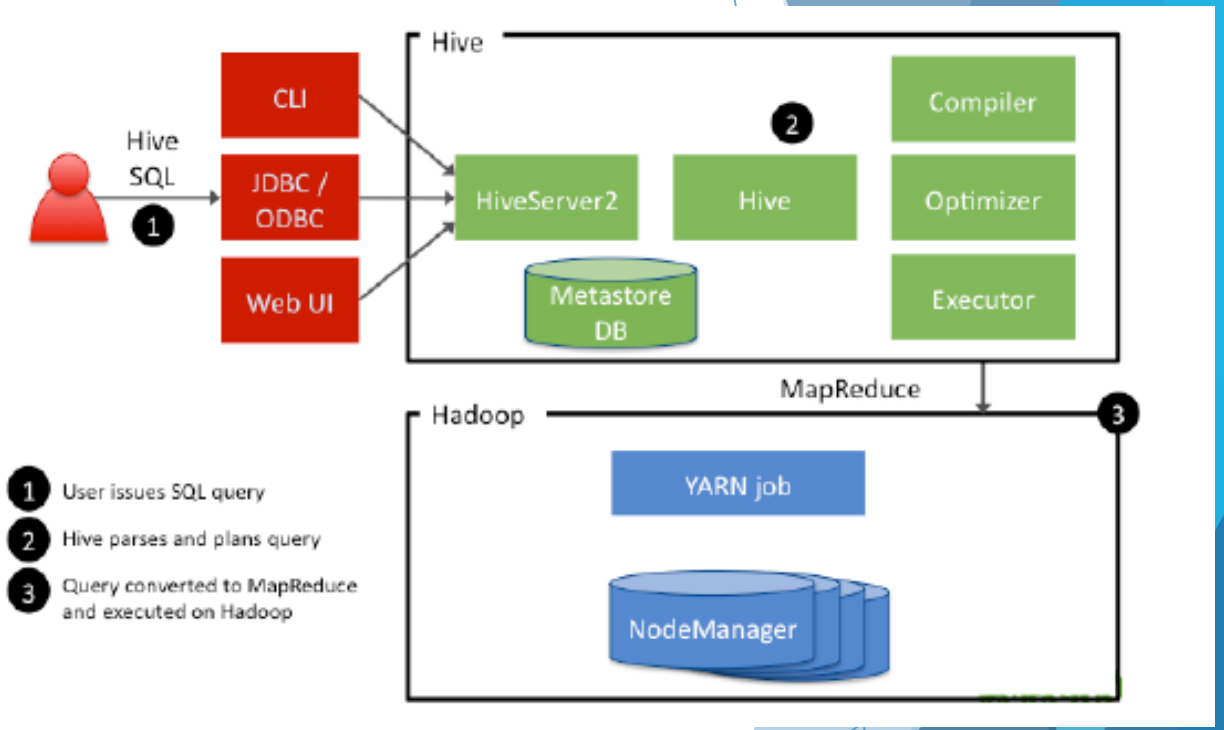
Hive

What is Hive

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

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 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

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

```
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

- 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.

TBLPROPERTIES

- TBLPROPERTIES contains the properties of a table

Ex:

- TBLPROPERTIES ("orc.compress"="ZLIB") or ("orc.compress"="SNAPPY") or ("orc.compress"="NONE") and other ORC properties
- TBLPROPERTIES("skip.header.line.count"="3","skip.footer.line.count"="1");
- TBLPROPERTIES ("comment"="table_comment");
- TBLPROPERTIES ("auto.purge"="true") or ("auto.purge"="false")

Compression Techniques

- hive>set hive.exec.compress.output = true;
- hive>set mapred.output.compression.codec= com.hadoop.compression.fourmc.FourMcCodec
- hive>set mapred.output.compression.codec= org.apache.hadoop.io.compress.GzipCodec
- hive>set mapred.output.compression.codec= com.hadoop.compression.lzo.LzopCodec
- hive>set mapred.output.compression.codec= org.apache.hadoop.io.compress.SnappyCodec
- hive>set mapred.output.compression.codec= org.apache.hadoop.io.compress.BZip2Codec
- hive>set mapred.output.compression.codec= org.apache.hadoop.io.compress.Lz4Codec

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: /user/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/';

Temporary Table

- A temporary table is a convenient way for an application to automatically manage intermediate data generated during a large or complex query execution
- Hive temporary tables are local to the user session
- Hive automatically deletes all temporary tables at the end of the Hive session in which they are created.
- The data in temporary tables is stored in the user's scratch directory rather than in the Hive warehouse directory.
- Usually, the location will be /tmp/hive/<user>/*.

```
hive> create temporary table t3(col1 int, col2 string);
OK
Time taken: 0.061 seconds
```

Hive Operations- Loading 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

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

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

- 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

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 DROP [COLUMN] column_name

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 – 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

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;
```

Simple Inner Join...

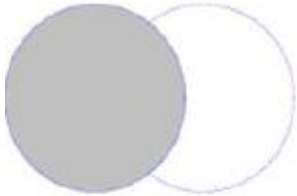
```
hive> INSERT OVERWRITE TABLE posts_likes
    > SELECT p.user, p.post, l.count
    > FROM posts p JOIN likes l ON (p.user = l.user);
OK
Time taken: 17.901 seconds
```

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>
```

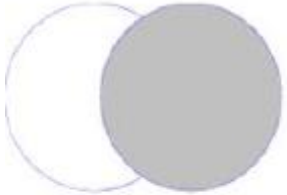
Hive - Outer Join

- 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
```

Left Semi Join

- Only returns the records from the left-hand table which present in Right side table
- The left semi join is used in place of the IN/EXISTS sub-query in Hive

```
hive> select * from klikes;
OK
user1   12      1343182026191
user2   7       1343182139394
user3   0       1343182154633
user4   50      1343182147364
Time taken: 0.253 seconds, Fetched: 4 row(s)
hive> select * from kposts;
OK
user1   Funny Story     1343182026191
user2   Cool Deal       1343182133839
user4   Interesting Post        1343182154633
user5   Yet Another Blog        13431839394
Time taken: 0.269 seconds, Fetched: 4 row(s)
```

```
hive> SELECT *
    > FROM kposts p LEFT SEMI JOIN klikes l ON (p.user = l.user);
Query ID = focus2_20200515085353_d268a9eb-d592-4b33-919c-c0277160a35f
Total jobs = 1
Execution log at: /tmp/focus2/focus2_20200515085353_d268a9eb-d592-4b33-919c
```

```
OK
user1   Funny Story     1343182026191
user2   Cool Deal       1343182133839
user4   Interesting Post        1343182154633
Time taken: 41.563 seconds, Fetched: 3 row(s)
```

Multi-Table/File Insert

- Run multiple queries within a single MapReduce job.
- You can gain a lot of performance by running two tasks at the same time instead of running two separate MapReduce jobs.

Consider the following simple Hive query that selects all White House visitors for the year 2013.

```
insert overwrite directory '2013_visitors' select * from wh_visits where
visit_year='2013' ;
```

Now suppose we have the following query on a different table named `congress`:

```
insert overwrite directory 'ca_congress' select * from congress where state='CA' ;
```

As expected, each query above requires a MapReduce job.

```
insert overwrite directory '2013_visitors' select * from wh_visits where
visit_year='2013'
```

No semi-colon

```
insert overwrite directory 'ca_congress' select * from congress where state='CA' ;
```

```
from visitors
INSERT OVERWRITE TABLE gender_sum
    SELECT visitors.gender, count_distinct(visitors.userid)
    GROUP BY visitors.gender

INSERT OVERWRITE DIRECTORY '/user/tmp/age_sum'
    SELECT visitors.age, count_distinct(visitors.userid)
    GROUP BY visitors.age;
```

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 Advanced

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 has 3 buckets that are clustered by the id column

CREATE TABLE employees (id int, name string, salary double)
CLUSTERED BY (id) INTO 3 BUCKETS;

Property:

- set hive.enforce.bucketing = true

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.

Example - Hive Buckets

```
create table emphive(   first_name      string,
                        last_name       string,
                        job_title       string,
                        department      string,
                        hire_date       string,
                        salary          int,
                        country         string,
                        state           string)
       clustered by (country) into 3 buckets
       row format delimited
       fields terminated by ',';_
```

```
from employees
        insert overwrite table emphive
        select first_name,last_name,job_title,department,
        hire_date,salary,country,state;_
```

```
[pkp@sandbox ~]$ hadoop fs -ls /apps/hive/warehouse/pkdb.db/emphive
Found 3 items
-rwx------   1 pkp hdfs       4602 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000000_0
-rwx------   1 pkp hdfs       2303 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000001_0
-rwx------   1 pkp hdfs         89 2013-11-25 09:40 /apps/hive/warehouse/pkdb.db
/emphive/000002_0
[pkp@sandbox ~]$ _
```

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);
```

Views

- A view in Hive is defined by a SELECT statement and allows you to treat the result of the query like a table.
- Define a view to reduce the complexity of a query.
- Define a view that contains only the columns and rows that the user needs

```
CREATE VIEW 2010_visitors AS
    SELECT fname, lname, time_of_arrival, info_comment
    FROM wh_visits
    WHERE
       cast(substring(time_of_arrival,6,4) AS int) >= 2010
    AND
       cast(substring(time_of_arrival,6,4) AS int) < 2011;
```

```
hive> describe 2010_visitors;
OK
fname                   string                  None
lname                   string                  None
time_of_arrival         string                  None
info_comment            string                  None
```

```
hive> show tables;
OK
2010_visitors
wh_visits
```

```
DROP VIEW 2010_visitors;
```

Sampling in Hive

- Sampling is selection of a subset of data from a large dataset to run queries and verify results
- We can run Hive queries on a sample of data using the TABLESAMPLE clause. Any column can be used for sampling the data.
- We need to provide the required sample size in the queries.

It has 2 types

- Sampling by Bucketing
- Block Sampling

Sampling by Bucketing

- We can run bucketing sampling queries on bucketed as well as non-bucketed tables. Also we can use bucketing columns as well as other columns for sampling.
- For a bucketed table we need not specify the column for sampling
- For non-bucketed table, whole table will be scanned and create the buckets as per the column(s) specified in the sampling query.

- TABLESAMPLE (BUCKET x OUT OF y [ON colname])

- Ex:

select * from Employee TABLESAMPLE(BUCKET 1 OUT OF 5);

Or

select * from Employee TABLESAMPLE(BUCKET 1 OUT OF 5 ON ID);

Block Sampling

- Block sampling allows Hive to select at least n% data from the whole dataset
- Sampling granularity is at the HDFS block size level.
- If HDFS block size is 64MB and n% of input size is only 10MB, then 64MB of data is fetched
- It can be done using percent or rows

- Ex: select * from Employee TABLESAMPLE(20 PERCENT);

or

- select * from Employee TABLESAMPLE(2 ROWS);

Hive Set Operators

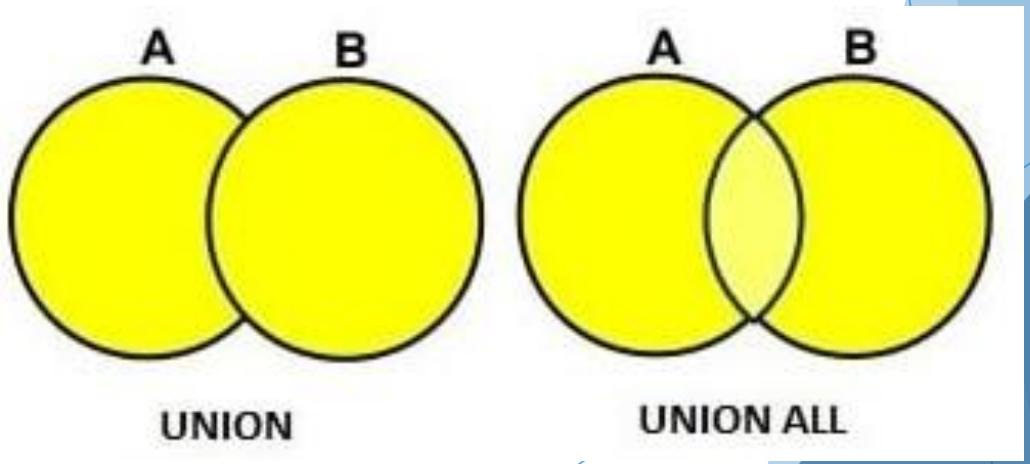
UNION is used to combine the result from multiple SELECT statements into a single result set.

Hadoop Hive supports following set operators.

- UNION
- UNION ALL

- A UNION ALL set operation does not remove duplicate rows from the result set.

```
SELECT av.uid AS uid
FROM action_video av
WHERE av.date = '2008-06-03'
UNION ALL
SELECT ac.uid AS uid
FROM action_comment ac
WHERE ac.date = '2008-06-03'
```



IN Operator

- Left Semi Join performs the same operation IN do in SQL.
- Both will fulfill the same purpose.

```
hive> select * from klikes;
OK
user1   12      1343182026191
user2   7       1343182139394
user3   0       1343182154633
user4   50      1343182147364
Time taken: 0.253 seconds, Fetched: 4 row(s)
hive> select * from kposts;
OK
user1   Funny Story     1343182026191
user2   Cool Deal       1343182133839
user4   Interesting Post        1343182154633
user5   Yet Another Blog        13431839394
Time taken: 0.269 seconds, Fetched: 4 row(s)
```

```
hive> SELECT *
    > FROM kposts
    > WHERE kposts.user IN (SELECT user FROM klikes);
```

```
user1   Funny Story     1343182026191
user2   Cool Deal       1343182133839
user4   Interesting Post        1343182154633
Time taken: 41.618 seconds, Fetched: 3 row(s)
hive>
```

Exists

- Apache Hive Correlated subquery is a query within a query that refer the columns from the outer query.
- The EXISTS operator is used to test for the existence of any record in a subquery.

```
SELECT col1, col2, ....
FROM table1 outer
WHERE col1 operator
        (SELECT col1, col2, ....
         FROM table2 inner
         where inner.col1 = outer.col1);
```

- Ex: Display Posts which got likes > 15

```
hive> SELECT p.user,p.post
    > FROM kposts p
    > WHERE EXISTS (SELECT l.user FROM klikes l WHERE p.user = l.user and l.like
r > 10);
```

```
hive> select * from klikes;
OK
user1   12      1343182026191
user2   7       1343182139394
user3   0       1343182154633
user4   50      1343182147364
Time taken: 0.253 seconds, Fetched: 4 row(s)
hive> select * from kposts;
OK
user1   Funny Story     1343182026191
user2   Cool Deal       1343182133839
user4   Interesting Post        1343182154633
user5   Yet Another Blog        13431839394
Time taken: 0.269 seconds, Fetched: 4 row(s)
```

```
OK
user1   Funny Story
user4   Interesting Post
Time taken: 42.581 seconds, Fetched: 2 row(s)
```

Lateral View Functions - Explode

- Hive offers functionality to bring nested data back into a relational view
- UDTF's (User defined Table-generating functions) like explode() or inline().
- These functions take a complex type field as a parameter and return multiple rows and columns, reflecting the same data in a relational view.
- Explode() takes in an array as input and outputs the elements of arrays as separate rows

```
Watches ["Red","Green"]
Clothes ["Blue","Green"]
Books   ["Blue","Green","Red"]
```

```
SELECT p.productname,colors.colorselection FROM default.products P
LATERAL VIEW EXPLODE(p.productcolors)colors as colorselection;
```

```
select explode(productcolors) as colors
from products;
```

```
Red
Green
Blue
Green
Blue
Green
Red
```

```
Watches Red
Watches Green
Clothes Blue
Clothes Green
Books   Blue
Books   Green
Books   Red
```

Inline Functions

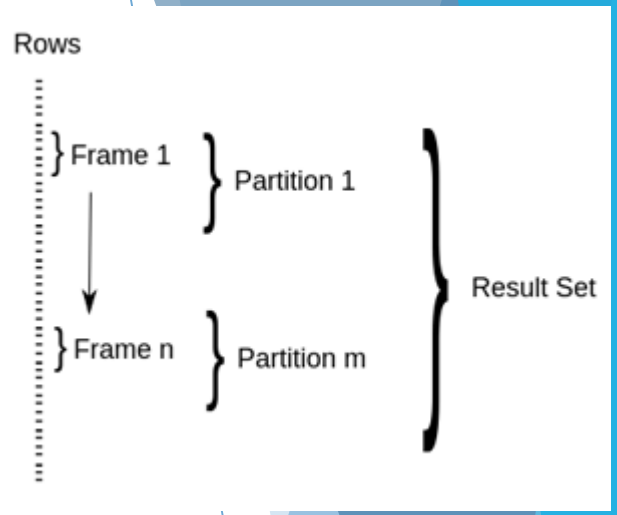
- You can use lateral view with inline function as well.
- Explodes an array of structs to multiple rows

```
SELECT lv.*
FROM   (SELECT 0) t
lateral VIEW
inline(array(struct('A',10,'AAA'),struct('B',20,
'BBB'),struct('B',300, 'CCC') )) lv AS col1, col2, col3;
```

```
+----------+----------+----------+--+
| lv.col1  | lv.col2  | lv.col3  |
+----------+----------+----------+--+
| A        | 10       | AAA      |
| B        | 20       | BBB      |
| B        | 300      | CCC      |
+----------+----------+----------+--+
```

Windowing Functions

- Windowing allows you to create a window on a set of data further allowing aggregation surrounding that data.
- The OVER clause allows you to define a window over which to perform a specific calculation.
- PARTITION clause divides result set into window partitions by one or more columns, and the rows within can be optionally sorted by one or more columns.
- Window frame selects rows from partition for window function to work on. There're two ways of defining frame in Hive, ROWS AND RANGE




```
(ROWS | RANGE) BETWEEN (UNBOUNDED | [num]) PRECEDING AND ([num] PRECEDING | CURRENT ROW | (UNBOUNDED
(ROWS | RANGE) BETWEEN CURRENT ROW AND (CURRENT ROW | (UNBOUNDED | [num]) FOLLOWING)
(ROWS | RANGE) BETWEEN [num] FOLLOWING AND (UNBOUNDED | [num]) FOLLOWING
```

- OVER with standard aggregates:
  - COUNT
  - SUM
  - MIN
  - MAX
  - AVG

Analytics functions

- RANK
- ROW_NUMBER
- DENSE_RANK
- CUME_DIST
- PERCENT_RANK
- NTILE

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;
```

ACID - Hive

- **ACID Transactions** – Turn this on to enable the creation of transactional tables globally.
- **Run Compactor** – The "Run Compactor" setting should always be set to true.

- The table must be clustered, be stored as ORCFile data, and have a table property that says "transactional" = "true".
- This type of table supports **UPDATE/DELETE AND MERGE** operation

```
CREATE TABLE hello_acid (key int, value int)
PARTITIONED BY (load_date date)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true');
```

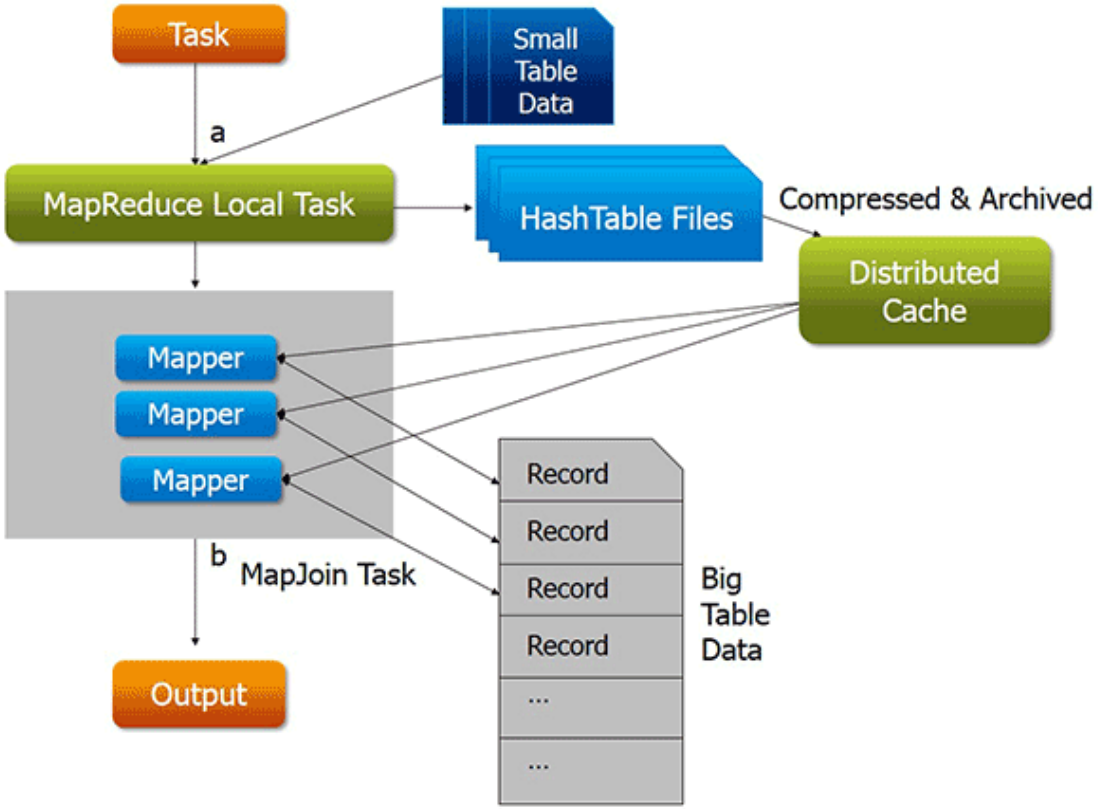
```
UPDATE students SET name = null WHERE gpa <= 1.0;
```

```
DELETE FROM students WHERE gpa <= 1.0;
```

```
merge into customer
using ( select * from new_customer_stage) sub
on sub.id = customer.id
when matched then update set name = sub.name, state = sub.new_state
when not matched then insert values (sub.id, sub.name, sub.state);
```

Map Join

- Map (Broadcast) Joins can be used to join a big table with a small table.
- Small table is stored in cache and is distributed (broadcast) to each mapper and then joined with the big table in the Ma phase
- Loading a small table into cache will save read time on each data node



- You can set the small file size by using the following property:

hive.mapjoin.smalltable.filesize=(default it will be 25MB)

- By setting the following property to true.

hive.auto.convert.join=true

- As it is a Map-side join, the number of reducers will be set to 0 automatically.

```
select /*+ MAPJOIN(a) */ a.* from passwords a, passwords2 b where a.col0=b.col0 ;
```

Explain

- EXPLAIN feature is used to learn how Hive translates queries into MapReduce jobs

```
hive> EXPLAIN SELECT SUM(number) FROM onecol;
```

```
hive> explain select * from twitpart;
OK
STAGE DEPENDENCIES:
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        TableScan
          alias: twitpart
          Statistics: Num rows: 24 Data size: 4082 Basic stats: COMPLETE Column
stats: NONE
          Select Operator
            expressions: tweetid (type: bigint), username (type: string), txt (t
ype: string), favc (type: bigint), retweet (type: string), retcount (type: bigin
t), followerscount (type: bigint), profilelocation (type: string)
            outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6,
_col7
            Statistics: Num rows: 24 Data size: 4082 Basic stats: COMPLETE Colum
n stats: NONE
            ListSink

Time taken: 1.754 seconds, Fetched: 17 row(s)
```

Sort By

- Hive uses SORT BY to sort the rows based on the given columns per reducer
- If there are more than one reducer, then the output per reducer will be sorted, but the order of total output is not guaranteed to be sorted.
- set mapred.reduce.tasks=2;
- select * from employee sort by salary;

employee.id	employee.name	employee.age	employee.salary	employee.department
10001	rajesh	29	50000	BIGDATA
1	aarti	28	55000	HR
2	sakshi	22	60000	HR
10003	dinesh	35	70000	BIGDATA
3	mahesh	25	25000	HR
10002	rahul	23	250000	BIGDATA

Rows in Red are sorted and rows in Blue are sorted among themselves. But the overall sorting order is not maintained. Both these set of rows were processed by different reducers

Order By

- Order by guarantees the total ordering of the output. Even if there are multiple reducers, the overall order of the output is maintained.
- set mapred.reduce.tasks=2;
- select * from employee order by salary;

employee.id	employee.name	employee.age	employee.salary	employee.department
3	mahesh	25	25000	HR
10001	rajesh	29	50000	BIGDATA
1	aarti	28	55000	HR
2	sakshi	22	60000	HR
10003	dinesh	35	70000	BIGDATA
10002	rahul	23	250000	BIGDATA

Overall order of sorting is maintained in the result by using Order By

Hands On

Thank You

Keerthiga Barathan