

SNO	Experiments
1	Installation of Cloudera and Virtual Box
2	DDL commands - (creating database, table, alter, drop)
3	Queries to sort
4	Aggregate functions in hive
5	Operators in Hive
6	Joins in Hive
7	Partitioning in Hive
8	Bucketing in Hive

1. Installation of Cloudera and Virtual Box

AIM:

To install Cloudera on a virtual environment using VirtualBox, providing a platform for managing and analyzing big data.

Pre-requires:

1. VirtualBox: <https://www.virtualbox.org>
2. Cloudera: <https://community.cloudera.com/t5/Support-Questions/Cloudera-Quickstart-VM-Download/m-p/291225>
3. Cloudera Direct download link:
https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.12.0-0-virtualbox.zip

Installation of VirtualBox:

1. Visit the [VirtualBox official website](#).
2. Download the VirtualBox installer for your operating system (Windows, macOS, Linux).
3. Run the installer and follow the on-screen instructions.
 - Select installation path and default options.
4. Once installed, open VirtualBox to verify the installation.

Installation of Cloudera:

1. Visit the Cloudera QuickStart VM download page.
2. Choose the version that matches your VirtualBox environment and download the `.ovf` file (VM image).
3. Save the downloaded file to a directory on your system.

Setting Cloudera in VirtualBox:

1. Open VirtualBox and click on `File > Import Appliance`.
2. Browse to the Cloudera `.ovf` file you downloaded and select it.
3. Click `Next` and review the VM settings.
4. Adjust any settings if necessary (like memory and CPU), and click `Import`.
5. Once the VM is imported, select it and click `Start`.

6. The Cloudera VM will boot up, and you can start using the Hadoop ecosystem.
7. CentOS does not require a password for login by default. To verify if Hive is installed correctly, open the terminal and type `hive`. If an error message appears, it indicates an issue with the Cloudera installation. Otherwise, if no error is displayed, the Hive database is functioning correctly.

2. DDL commands - (creating database, table, alter, drop)

AIM:

To create, alter and drop, database, tables, views and indexes.

ALGORITHM:

DDL:

DDL (Data Definition Language) refers to SQL commands used to define or modify database structures. Common DDL commands include `CREATE`, `ALTER`, and `DROP`, which are used to create tables, alter schema, and delete objects in a database. DDL commands directly affect the structure of the database, not the data.

1 - Create the Database

syntax:

```
CREATE DATABASE <database-name>
```

```
CREATE DATABASE student_details
```

this command will help to create the database with desired name.

2 - Use the Database

syntax:

```
USE <database-name>
```

```
USE student_details
```

once when we created the database, we have to select the database, then only we can use to create the database.

3 - Create the table

syntax:

```
CREATE TABLE <table-name> (  
column-name datatype,  
column-name datatype,  
.  
.  
.  
column-name datatype  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ',';
```

```
CREATE TABLE student(  
name string,  
age int,  
address string  
)  
ROW FORAMT DELIMITED  
FIELDS TERMINATED BY ',';
```

these above command will help to create the table, with desired column name and datatype.

4 - Inserting the data

syntax:

```
load data local inpath '<absolute-file-path>/<file-name>' into table  
<table-name>;
```

```
load data local inpath  
'/home/cloudera/Desktop/Arulkumaran/std_details.csv' into table student;
```

unlike the other database, in `hive` we have to load the data file, using the above command, and thus the data can be loaded into the table.

5 - Selecting the data

syntax:

```
SELECT * from <table-name>;
```

```
SELECT * from student;
```

to retrieve the meaningful insight from the table, we can use the select command, select command is used to fetch the data from the table.

name	age	address
Arulkumaran	21	Kanchipuram
lisa	23	Thailand
Jennie	20	Korea
Jisso	22	Korea
Rose	23	New zealand

6 - Altering the database

Rename

```
ALTER TABLE <table-name> RENAME to <new-table-name>;
```

```
ALTER TABLE student RENAME to student_details;
```

Add Columns

```
ALTER TABLE <table-name> ADD COLUMNS (  
  <column-name> datatype  
  .  
  .  
  <column-name> datatype  
);
```

```
ALTER TABLE student_details ADD COLUMNS(regno int);
```

Drop column name

```
ALTER TABLE <table-name> DROP <column-name>;
```

```
ALTER TABLE student_details DROP regno;
```

Change Column Name

```
ALTER TABLE <table-name> CHANGE <old-column-name> <new-column-name>;
```

```
ALTER TABLE student_details CHANGE name names;
```

these we the various `alter` that we use in hive, to change the column name, add the new column or rename the table or delete the column, in simple alter command used to change the schema structure of table.

7 - Views

Views in SQL are virtual tables created using the `SELECT` statement to present data from one or more tables. They allow users to simplify complex queries, restrict data access, or present data in a customized way without storing it physically. Views can be queried like regular tables but don't hold any data themselves.

syntax:

```
CREATE VIEW <view-name> as SELECT * from <table-name>  
WHERE <condition>;
```

```
CREATE VIEW lisa_details as SELECT * from student_details  
WHERE names="lisa";
```

8 - Index

An index in SQL is a database object that improves the speed of data retrieval by creating a reference to data in a table based on specified columns. It functions like a book index, allowing faster searches without scanning the entire table. However, indexes can slow down data modification operations like `INSERT` or `UPDATE`.

syntax:

```
CREATE INDEX <index-name> ON TABLE <table-name>;
```

```
CREATE INDEX student_names ON TABLE student_details(names);
```

9 - Drop

The `DROP` command in SQL is used to permanently delete database objects like tables, views, or indexes. Once executed, the object and all its data are removed from the database

without the possibility of recovery. It's an irreversible operation, so it should be used with caution.

drop view

```
DROP VIEW <view-name>;
```

```
DROP VIEW lisa_details;
```

drop index

```
DROP INDEX <index-name> ON <table-name>;
```

```
DROP INDEX student_names ON student_details;
```

drop table

```
DROP TABLE <table-name>
```

```
DROP TABLE student_details;
```

drop database

```
DROP DATABASE <database-name> CASCADE;
```

```
DROP DATABASE student_details CASCADE;
```


3. Queries to Sort

AIM:

to sort the hive tables based on ORDER BY, SORT BY, and CLUSTER BY clause.

ALGORITHM:

1. ORDER BY

Globally sorts the entire dataset across `multiple reducers`. It ensures a total ordering of results. This can be **slow** because all data is passed through a single reducer.

Name	rank	Age
Lisa	1	21
Jennie	2	20
Jisso	3	22
Rose	4	21

```
SELECT * from <table-name> ORDER BY <column-name> <specifier>;
```

```
SELECT * from kpop ORDER BY rank desc;
```

this command will sort the table by rank column in descending order.

Name	rank	Age
Rose	4	21
Jisso	3	22
Jennie	2	20
Lisa	1	21

2. SORT BY

Partially sorts the data within `each reducer`, not globally. It is **faster** than ORDER BY but doesn't guarantee total ordering.

```
SELECT * from <table-name> SORT BY <table-name> <specifier>;
```

```
SELECT * from kpop SORT BY Age asc;
```

this will sort based on Age column in ascending order.

Name	rank	Age
Jennie	2	20
Lisa	1	21
Rose	4	21
Jisso	3	22

3. CLUSTER BY

Distributes rows into different reducers based on the column, and within each reducer, it sorts the data. Used for *bucketing*.

```
SELECT * from <table-name> CLUSTER BY <table-name> <specifier>;
```

```
SELECT * from kpop CLUSTER BY Age asec;
```

this will sort based on Age column in ascending order.

Name	rank	Age
Jennie	2	20
Lisa	1	21
Rose	4	21
Jisso	3	22

4. Aggregate Functions in Hive

AIM:

to perform hive operations based on aggregate functions.

Algorithm:

Aggregate functions in Hive are used to perform calculations on multiple rows of data and return a single result. These functions, such as `COUNT()`, `SUM()`, `AVG()`, `MAX()`, and `MIN()`, are commonly used in combination with `GROUP BY` clauses to summarize or analyze large datasets. They help in tasks like counting rows, summing values, finding averages, and determining the highest or lowest values within a dataset.

1. Hive Functions

Return Type	Functions	Descriptions
BIGINT	<code>round(num)</code>	It returns the BIGINT for the rounded value of DOUBLE num.
BIGINT	<code>floor(num)</code>	It returns the largest BIGINT that is less than or equal to num.
BIGINT	<code>ceil(num)</code> <code>ceiling(DOUBLE num)</code>	It returns the smallest BIGINT that is greater than or equal to num.
DOUBLE	<code>exp(num)</code>	It returns exponential of num.
DOUBLE	<code>ln(num)</code>	It returns the natural logarithm of num.
DOUBLE	<code>log10(num)</code>	It returns the base-10 logarithm of num.
DOUBLE	<code>sqrt(num)</code>	It returns the square root of num.
DOUBLE	<code>abs(num)</code>	It returns the absolute value of num.
DOUBLE	<code>sin(d)</code>	It returns the sin of num, in radians.
DOUBLE	<code>asin(d)</code>	It returns the arcsin of num, in radians.
DOUBLE	<code>cos(d)</code>	It returns the cosine of num, in radians.
DOUBLE	<code>acos(d)</code>	It returns the arccosine of num, in radians.
DOUBLE	<code>tan(d)</code>	It returns the tangent of num, in radians.
DOUBLE	<code>atan(d)</code>	It returns the arctangent of num, in radians.

Example

Name	Salary	Age
Lisa	90000.4023	21
Jennie	85000.4313	20
Rose	75000.4323	22
Jisso	86000.5324	21

```
SELECT Name, floor(Salary) from kpop;
```

this command will round the salary column in closed to integer part.

Name	Salary
Lisa	90000
Jennie	85000
Rose	75000
Jisso	86000

2. Hive Aggregate Functions

Return type	Operator	Description
BIGINT	count(*)	It returns the count of the number of rows present in the file.
DOUBLE	sum(col)	It returns the sum of values.
DOUBLE	sum(DISTINCT col)	It returns the sum of distinct values.
DOUBLE	avg(col)	It returns the average of values.
DOUBLE	avg(DISTINCT col)	It returns the average of distinct values.
DOUBLE	min(col)	It compares the values and returns the minimum one form it.
DOUBLE	max(col)	It compares the values and returns the maximum one form it.

Example

Name	Salary	Age
Lisa	90000	21
Jennie	85000	20
Rose	75000	22
Jisso	86000	21

```
SELECT Name from kpop WHERE Age=max(Age);
```

this command will retrieve the name who has the maximin age.

Name
Rose

3. Hive Build-in Functions

Return type	Operator	Description
INT	length(str)	It returns the length of the string.
STRING	reverse(str)	It returns the string in reverse order.
STRING	concat(str1, str2, ...)	It returns the concatenation of two or more strings.
STRING	substr(str, start_index)	It returns the substring from the string based on the provided starting index.
STRING	substr(str, int start, int length)	It returns the substring from the string based on the provided starting index and length.
STRING	upper(str)	It returns the string in uppercase.
STRING	lower(str)	It returns the string in lowercase.
STRING	trim(str)	It returns the string by removing whitespaces from both the ends.
STRING	ltrim(str)	It returns the string by removing whitespaces from left-hand side.
STRING	rtrim(str)	It returns the string by removing whitespaces from right-hand side.

Example

Name	Salary	Age
Lisa	90000	21
Jennie	85000	20
Rose	75000	22
Jisso	86000	21

```
SELECT upper(Name), Salary from kpop;
```

this produces output Name and Salary column, but content in name column in Capitalized.

Name	Salary
LISA	90000
JENNIE	85000
ROSE	75000
JISSO	86000

5. Operators in Hive

AIM:

To perform Hive Operations, like Arithmetic, Relation operators and Hive Functions.

ALGORITHM:

1. Hive Arithmetic Operations

The arithmetic operator accepts any numeric type. The commonly used arithmetic operators

Operators	Description
A + B	This is used to add A and B.
A - B	This is used to Subtract A and B.
A * B	This is used to Multiply A and B.
A / B	This is used to divide A and B and returns the quotient of the operands.
A % B	This returns the remainder of A / B.
A B	This is used to determine the bitwise OR of A and B.
A & B	This is used to determine the bitwise AND of A and B.
A ^ B	This is used to determine the bitwise XOR of A and B.
~A	This is used to determine the bitwise NOT of A.

Example

consider this table

Name	Salary	Age
Lisa	90000	21
Jennie	85000	20
Rose	75000	22
Jisso	86000	21

```
SELECT Name, Salary + 10000 from kpop;
```

this above command will increment 10000 in salary to all members from the table.

Name	Salary
Lisa	100000

Name	Salary
Jennie	95000
Rose	85000
Jisso	96000

2. Hive Relational Operators

Operators	Description
A = B	It returns true if A equals B, otherwise false.
A <> B A != B	It returns null if A or B is null; true if A is not equal to B, otherwise false.
A < B	It returns null if A or B is null; true if A is less than B, otherwise false.
A > B	It returns null if A or B is null; true if A is greater than B, otherwise false.
A <= B	It returns null if A or B is null; true if A is less than or equal to B, otherwise false.
A >= B	It returns null if A or B is null; true if A is greater than or equal to B, otherwise false.
A IS NULL	It returns true if A evaluates to null, otherwise false.
A IS NOT NULL	It returns false if A evaluates to null, otherwise true.

Example

Name	Salary	Age
Lisa	90000	21
Jennie	85000	20
Rose	75000	22
Jisso	86000	21

```
SELECT Name from kpop WHERE Age <= 21;
```

this command will retrieve the Name from table whose age is lesser then 21.

Name
Lisa
Jennie
Jisso

6. Hive Joins

AIM:

To perform Hive inner join, left, right, full outer join operations.

ALGORITHM:

Example Tables

ani_char_rank

Rank	Character_Name
1	Kiyotaka Ayanokoji
2	Jack the Ripper
2	Hates
2	Poseidon
3	Light Yagami
3	Missa amane
4	Itachi Uchiha
4	Madara Uchiha

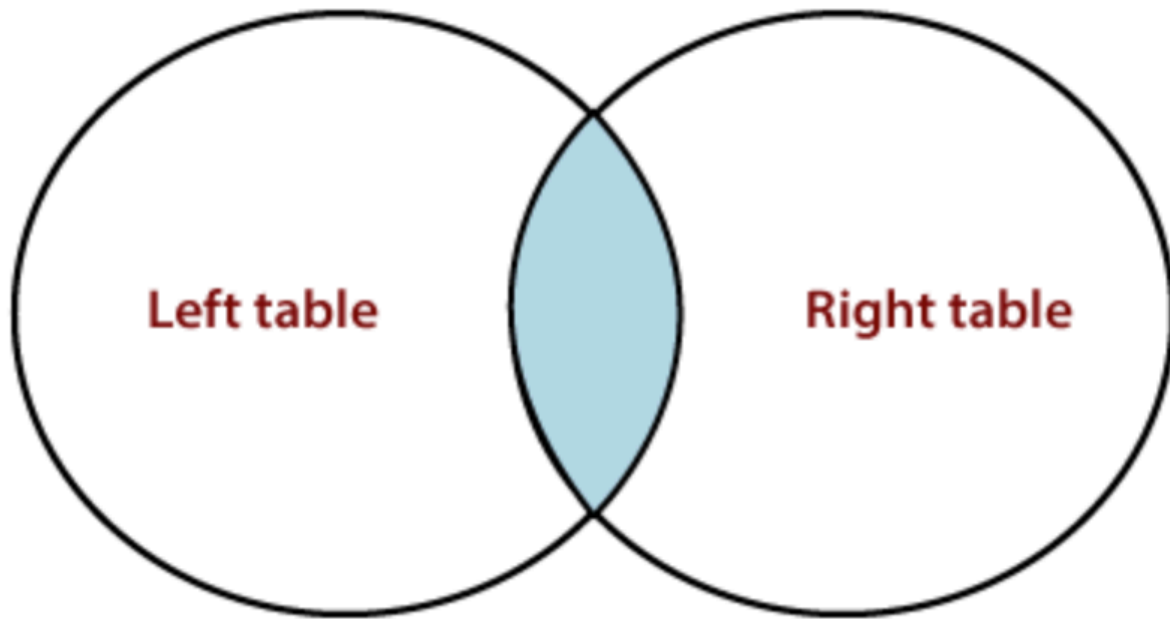
ani_rank

Anime_Rank	Anime_Name
1	Classroom of the Elite
2	Records of Ragnarok
3	Death Note
5	Death parade

1. Inner Join

The HiveQL inner join is used to return the rows of multiple tables where the join condition satisfies. In other words, the join criteria find the match records in every table being joined.

Inner Join



```
SELECT acr.Character_Name,ar.Anime_Name from ani_char_rank acr JOIN  
ani_rank ar ON acr.Rank=ar.Anime_Rank;
```

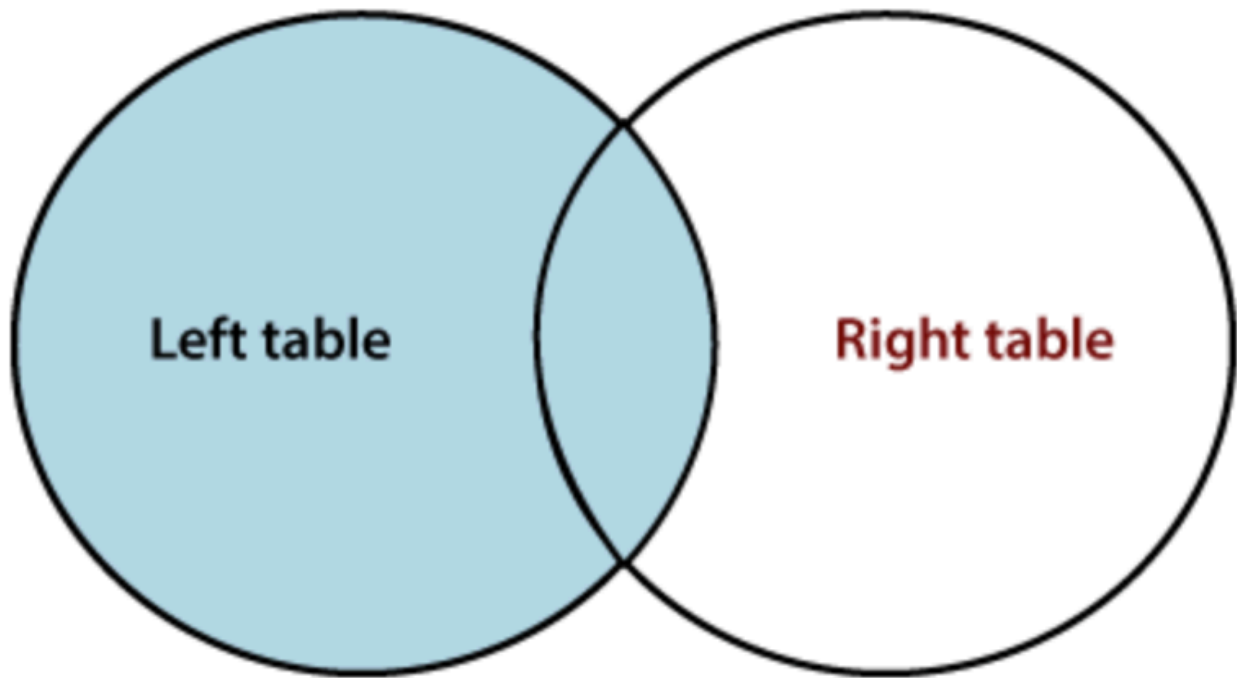
The SQL query retrieves character names and anime names by joining two tables, `ani_char_rank` and `ani_rank`. It matches the character's rank from `ani_char_rank` with the corresponding anime rank in `ani_rank` using the `Rank` field. The `SELECT` statement extracts the `Character_Name` from the first table and the `Anime_Name` from the second table. This join allows for combining related data from both tables where the ranks are the same, providing a clear association between anime characters and their respective shows.

Character_Name	Anime_Name
Kiyotaka Ayanokoji	Classroom of the Elite
Jack the Ripper	Records of Ragnarok
Hates	Records of Ragnarok
Poseidon	Records of Ragnarok
Light Yagami	Death Note
Missa amane	Death Note

2. Left Outer Join

The HiveQL left outer join returns all the records from the left (first) table and only that records from the right (second) table where join criteria find the match.

Left Outer Join



```
SELECT acr.Character_Name, ar.Anime_Name from ani_char_rank acr LEFT OUTER  
JOIN ani_rank ar ON acr.Rank=ar.Anime_Rank;
```

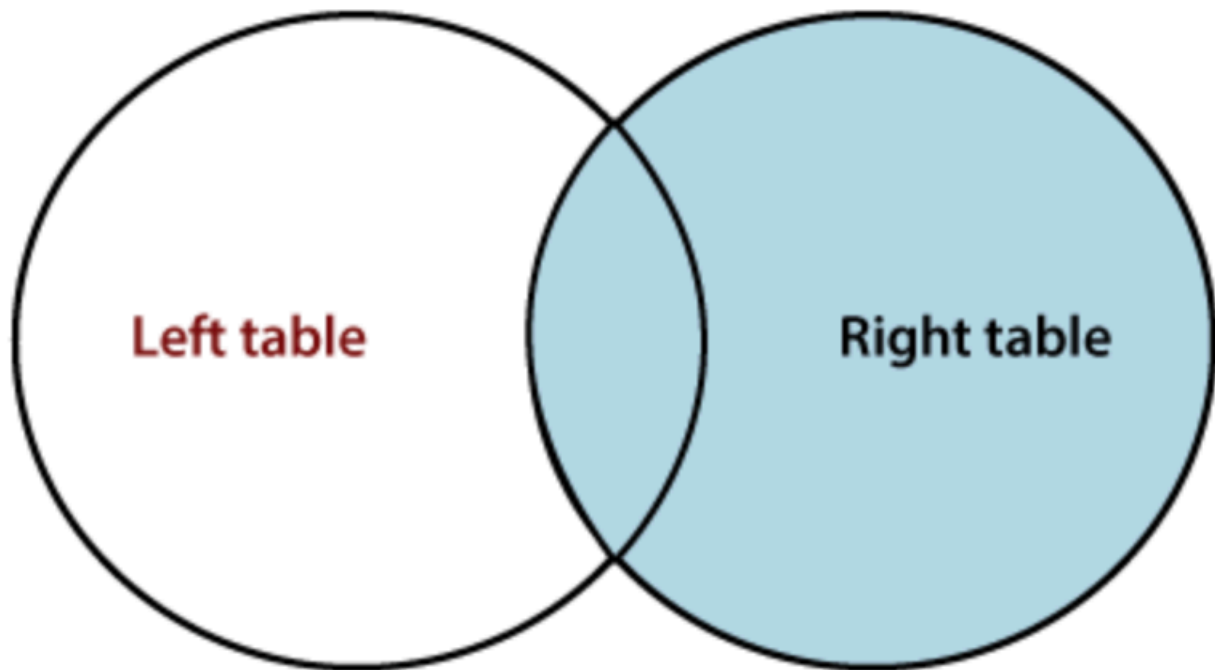
This SQL query retrieves character names from `ani_char_rank` and anime names from `ani_rank` using a left outer join. It returns all characters from `ani_char_rank`, even if there is no matching rank in `ani_rank`, showing `NULL` for the anime name when no match is found. The join is based on matching the `Rank` from `ani_char_rank` with `Anime_Rank` from `ani_rank`.

Character_Name	Anime_Name
Kiyotaka Ayanokoji	Classroom of the Elite
Jack the Ripper	Records of Ragnarok
Hates	Records of Ragnarok
Poseidon	Records of Ragnarok
Light Yagami	Death Note
Missa amane	Death Note
Itachi Uchiha	NULL
Madara Uchiha	NULL

3. Right Outer Join

The HiveQL right outer join returns all the records from the right (second) table and only that records from the left (first) table where join criteria find the match.

Right Outer Join



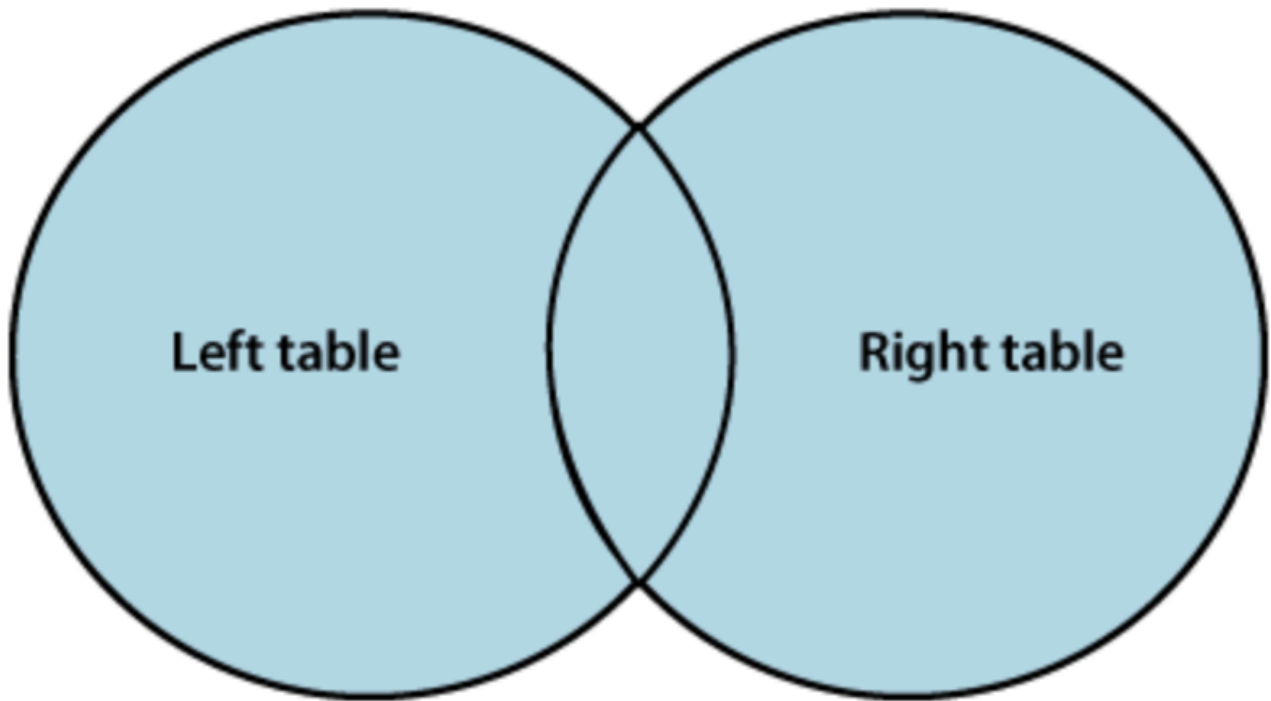
```
SELECT acr.Character_Name, ar.Anime_Name from ani_char_rank acr RIGHT  
OUTER JOIN ani_rank ar ON acr.Rank=ar.Anime_Rank;
```

Character_Name	Anime_Name
Kiyotaka Ayanokoji	Classroom of the Elite
Jack the Ripper	Records of Ragnarok
Hates	Records of Ragnarok
Poseidon	Records of Ragnarok
Light Yagami	Death Note
Missa amane	Death Note
NULL	NULL

4. Full Outer Join

The HiveQL full outer join returns all the records from both the tables. It assigns Null for missing records in either table

Full Outer Join



```
SELECT acr.Character_Name, ar.Anime_Name from ani_char_rank acr FULL OUTER  
JOIN ani_rank ar ON acr.Rank=ar.Anime_Rank;
```

This SQL query retrieves anime names from `ani_rank` and character names from `ani_char_rank` using a right outer join. It returns all anime names from `ani_rank`, even if there is no matching character rank in `ani_char_rank`, showing `NULL` for the character name when no match is found. The join is based on matching the `Rank` from `ani_char_rank` with `Anime_Rank` from `ani_rank`.

Character_Name	Anime_Name
Kiyotaka Ayanokoji	Classroom of the Elite
Jack the Ripper	Records of Ragnarok
Hates	Records of Ragnarok
Poseidon	Records of Ragnarok
Light Yagami	Death Note
Missa amane	Death Note
NULL	Death Parade
Itachi Uchiha	NULL
Madara Uchiha	NULL

7. Partitioning in Hive

AIM:

To perform Partitioning in HiveQL

ALGORITHM:

Partitioning in HiveQL refers to dividing a large dataset into smaller, more manageable parts (partitions) based on the values of one or more columns. It improves query performance by scanning only the relevant partitions instead of the entire dataset.

```
CREATE TABLE anime (sno INT, character_name STRING, power STRING)
PARTITIONED BY (anime_name STRING);
```

consider this 3 tables

jujutsukaisen.csv

sno	character_name	power
1	Ryoman Sukuna	melovalent strine
2	Gojo Saturo	Infinity void
3	Toji Fushigaro	Will Power GYM

natuto.csv

sno	character_name	power
1	Itachi Uchiha	Genjutsu
2	Jiraya	toad seige
3	Madara Uchiha	Rinnegan

codebreakers.csv

sno	character_name	power
1	Ogami	Fire
2	Toki	Magnetism
3	Hitomi	Electric Shock

we need to save these tables from seperate directory.

```
load data local inpath '/home/cloudera/Desktop/anime/jujutsu_kaisen.csv'
into table anime partition (anime_name="jujutsu kaisen");

load data local inpath '/home/cloudera/Desktop/anime/naruto.csv' into
table anime partition (anime_name="naruto");

load data local inpath '/home/cloudera/Desktop/anime/codebreakers.csv'
into table anime partition (anime_name="code breakers");
```

```
select * from anime;
```

sno	character_name	power	anime_name
1	Ryoman Sukuna	melovalent strine	jujutsu kaisen
2	Gojo Saturo	Infinity void	jujutsu kaisen
3	Toji Fushigaro	Will Power GYM	jujutsu kaisen
1	Itachi Uchiha	Genjutsu	naruto
2	Jiraya	toad seige	naruto
3	Madara Uchiha	Rinnegan	naruto
1	Ogami	Fire	code breakers
2	Toki	Magnetism	code breakers
3	Hitomi	Electric Shock	code breakers

```
select * from anime where anime_name="jujutsu kaisen";
```

sno	character_name	power	anime_name
1	Ryoman Sukuna	melovalent strine	jujutsu kaisen
2	Gojo Saturo	Infinity void	jujutsu kaisen
3	Toji Fushigaro	Will Power GYM	jujutsu kaisen

these **anime_name** are saved in separate directory for fast retrieval of tables.

8. Bucketing in Hive

AIM:

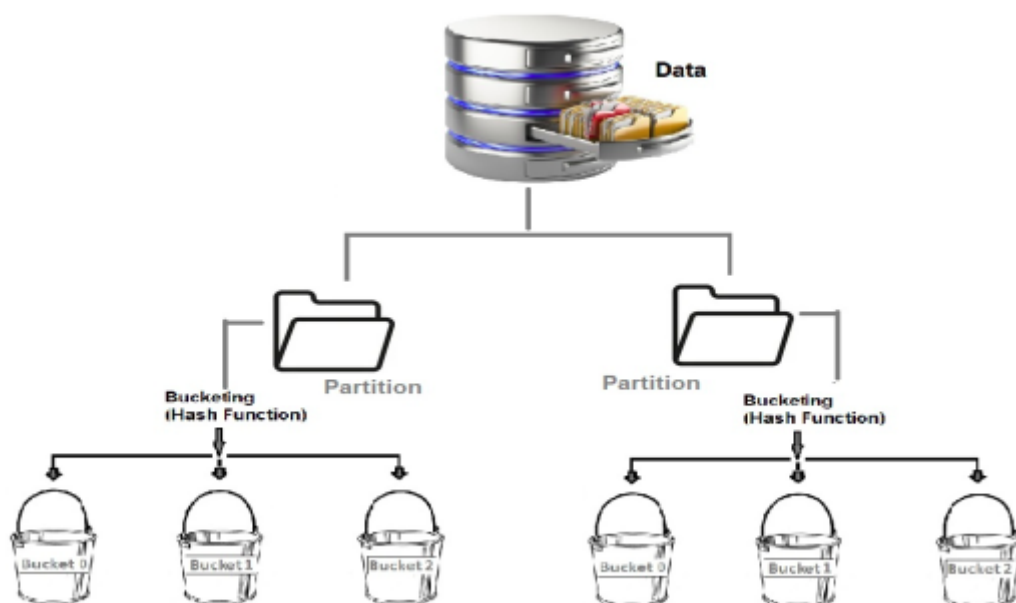
To perform Bucketing concept in Hive

ALGORITHM:

Bucketing in Hive is a technique used to further divide data within partitions into more manageable chunks (buckets) based on the hash of a column. It improves query efficiency, especially for operations like joins and sampling.

Bucketing distributes data across a fixed number of buckets based on the hash function of a specific column.

It optimizes joins and grouping operations by ensuring evenly distributed data within buckets. The column used for bucketing determines how data is split.



Example

consider this table

favanime.csv

rank	anime names
1	Classroom of the Elite
2	Death Note
3	Death Parade
2	Jujutsu Kaisen

rank	anime names
3	Records of Ragnarok
2	Ragna Crimson
1	Monster
4	Chainsaw man
4	Spy x family

```
CREATE TABLE fav_anime(
  rank INT,
  anime_name STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/anime/favanime.csv' INTO
TABLE fav_anime;
```

```
CREATE TABLE fav_anime_bucket(
  rank INT,
  anime_name STRING
)
CLUSTER BY (rank) into 3 buckets
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
INSERT OVERWRITE TABLE fav_anime_bucket SELECT * from fav_anime;
```

this above command will select every rows from ***fav_anime*** table and replace to ***fav_anime_bucket*** based on hashing algorithm with 3 buckets,

```
1 % 3 = 1
```


```
2 % 3 = 2
```



```
3 % 3 = 0
```






```
4 % 3 = 1
```

based on output 0, 1, 2 will make the decision, in which bucket the now is need to placed.

File Edit View History Bookmarks Tools Help

HDFS:/user/hive/warehouse/em... 

 localhost.localdomain:50075/browseDirectory.jsp?dir=%2Fuser%2Fhive%2Fwarehouse%2Fem... 

Most Visited  Cloudera  Cloudera Manager  Hue  HDFS NameNode  Hadoop JobTracker