

Partitioning in Hive

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

As we know that Hadoop is used to handle the huge amount of data, it is always required to use the best approach to deal with it. The partitioning in Hive is the best example of it.

Let's assume we have a data of 10 million students studying in an institute. Now, we have to fetch the students of a particular course. If we use a traditional approach, we have to go through the entire data. This leads to performance degradation. In such a case, we can adopt the better approach i.e., partitioning in Hive and divide the data among the different datasets based on particular columns.

The partitioning in Hive can be executed in two ways -

- Static partitioning
- Dynamic partitioning

Static Partitioning

In static or manual partitioning, it is required to pass the values of partitioned columns manually while loading the data into the table. Hence, the data file doesn't contain the partitioned columns.

Example of Static Partitioning

- First, select the database in which we want to create a table.

```
hive> use test;
```

- Create the table and provide the partitioned columns by using the following command: -

```
hive> create table student (id int, name string, age int, institute string)
```

```
partitioned by (course string)
```

```
row format delimited
```

fields terminated by ',';

- Let's retrieve the information associated with the table.

hive> describe student;

- Load the data into the table and pass the values of partition columns with it by using the following command: -

hive> load data local inpath '/home/codegyani/hive/student_details1' into table student partition(course= "java");

Here, we are partitioning the students of an institute based on courses.

- Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

hive> load data local inpath '/home/codegyani/hive/student_details2' into table student partition(course= "hadoop");

In the following screenshot, we can see that the table student is divided into two categories.

Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾							
Browse Directory							
<input type="text" value="/user/hive/warehouse/test.db/student"/>							<input type="button" value="Go!"/>
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 5:39:27 PM	0	0 B	course=hadoop
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 5:37:13 PM	0	0 B	course=java

- Let's retrieve the entire data of the table by using the following command: -

```
hive> select * from student;
```

Now, try to retrieve the data based on partitioned columns by using the following command: -

```
hive> select * from student where course="java";
```

In this case, we are not examining the entire data. Hence, this approach improves query response time.

- Let's also retrieve the data of another partitioned dataset by using the following command: -

```
hive> select * from student where course= "hadoop";
```

Dynamic Partitioning

In dynamic partitioning, the values of partitioned columns exist within the table. So, it is not required to pass the values of partitioned columns manually.

- First, select the database in which we want to create a table.

```
hive> use show;
```

Enable the dynamic partition by using the following commands: -

```
hive> set hive.exec.dynamic.partition=true;
```

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

- Create a dummy table to store the data.

```
hive> create table stud_demo(id int, name string, age int, institute string, course string)
```

```
row format delimited
```

```
fields terminated by ',';
```

- Now, load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/student_details' into table stud_demo;
```

- Create a partition table by using the following command: -

```
hive> create table student_part (id int, name string, age int, institute string)
partitioned by (course string)
row format delimited
fields terminated by ',';
```

Now, insert the data of dummy table into the partition table.

```
hive> insert into student_part
partition(course)
select id, name, age, institute, course
from stud_demo;
```

Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾							
Browse Directory							
<input type="text" value="/user/hive/warehouse/show.db/student_part"/>							<input type="button" value="Go!"/>
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 3:53:17 PM	0	0 B	course=__HIVE_DEFAULT_PARTITION_
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 3:53:15 PM	0	0 B	course=hadoop
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 3:53:16	0	0 B	course=java

- Let's retrieve the entire data of the table by using the following command: -

```
hive> select * from student_part;
```

Now, try to retrieve the data based on partitioned columns by using the following command: -

```
hive> select * from student_part where course= "java ";
```

In this case, we are not examining the entire data. Hence, this approach improves query response time.

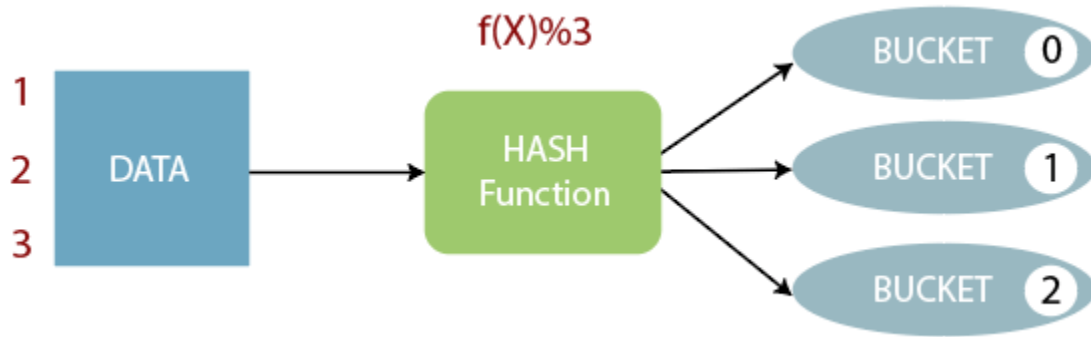
- Let's also retrieve the data of another partitioned dataset by using the following command: -

```
hive> select * from student_part where course= "hadoop";
```

Bucketing in Hive

The bucketing in Hive is a data organizing technique. It is similar to partitioning in Hive with an added functionality that it divides large datasets into more manageable parts known as buckets. So, we can use bucketing in Hive when the implementation of partitioning becomes difficult. However, we can also divide partitions further in buckets.

Working of Bucketing in Hive



- The concept of bucketing is based on the hashing technique.
- Here, modulus of current column value and the number of required buckets is calculated (let say, $F(x) \% 3$).
- Now, based on the resulted value, the data is stored into the corresponding bucket.

Example of Bucketing in Hive

- First, select the database in which we want to create a table.

hive> use showbucket;

Create a dummy table to store the data.

hive> create table emp_demo (Id int, Name string , Salary float)

row format delimited

fields terminated by ',' ;

- Now, load the data into the table.

hive> load data local inpath '/home/codegyani/hive/emp_details' into table emp_demo;

Enable the bucketing by using the following command: -

hive> set hive.enforce.bucketing = true;

- Create a bucketing table by using the following command: -

hive> create table emp_bucket(Id int, Name string , Salary float)

clustered by (Id) into 3 buckets

row format delimited

fields terminated by ',';

Now, insert the data of dummy table into the bucketed table.

```
hive> insert overwrite table emp_bucket
```

```
select * from emp_demo;
```

Here, we can see that the data is divided into three buckets.

Hadoop Overview Datanodes Snapshot Startup Progress Utilities								
Browse Directory								
/user/hive/warehouse/showbucket.db/emp_bucket								Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rwxr-xr-x	codegyani	supergroup	56 B	8/2/2019, 4:11:20 AM	1	128 MB	000000_0	
-rwxr-xr-x	codegyani	supergroup	53 B	8/2/2019, 4:11:20 AM	1	128 MB	000001_0	
-rwxr-xr-x	codegyani	supergroup	54 B	8/2/2019, 4:11:20 AM	1	128 MB	000002_0	

- Let's retrieve the data of bucket 0.

```
codegyani@ubuntu64server: ~  
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp_bucket/000000_0;  
\N,\N,\N  
\N,\N,\N  
6,"William",9000.0  
3,"Vishal",40000.0
```

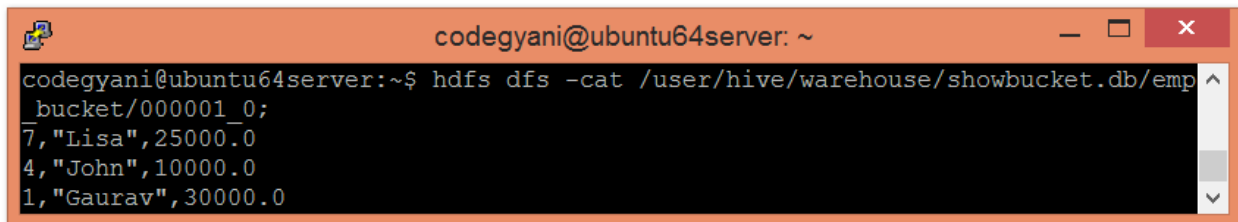
According to hash function :

$$6\%3=0$$

$$3\%3=0$$

So, these columns stored in bucket 0.

- Let's retrieve the data of bucket 1.



```
codegyani@ubuntu64server: ~  
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp  
bucket/000001_0;  
7, "Lisa", 25000.0  
4, "John", 10000.0  
1, "Gaurav", 30000.0
```

According to hash function :

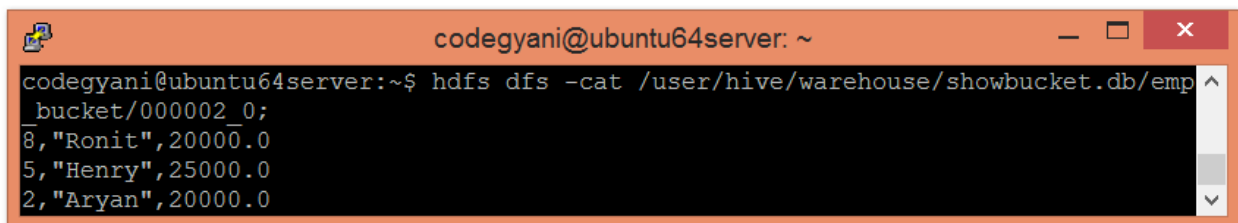
$$7\%3=1$$

$$4\%3=1$$

$$1\%3=1$$

So, these columns stored in bucket 1.

- Let's retrieve the data of bucket 2.



```
codegyani@ubuntu64server: ~  
codegyani@ubuntu64server:~$ hdfs dfs -cat /user/hive/warehouse/showbucket.db/emp  
bucket/000002_0;  
8, "Ronit", 20000.0  
5, "Henry", 25000.0  
2, "Aryan", 20000.0
```

According to hash function :

$$8\%3=2$$

$$5\%3=2$$

$$2\%3=2$$

So, these columns stored in bucket 2.

HiveQL - Operators

The HiveQL operators facilitate to perform various arithmetic and relational operations. Here, we are going to execute such type of operations on the records of the below table:

employee		
Id	Name	Salary
1	Gaurav	30000
2	Aryan	20000
3	Vishal	40000
4	John	10000
5	Henry	25000
6	William	9000
7	Lisa	25000
8	Ronit	20000

Example of Operators in Hive

Let's create a table and load the data into it by using the following steps: -

- o elect the database in which we want to create a table.

```
hive> use hql;
```

- o Create a hive table using the following command: -

```
hive> create table employee (Id int, Name string , Salary float)
```

```
row format delimited
```

```
fields terminated by ',' ;
```

- o Now, load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/emp_data' into table employee;
```

Let's fetch the loaded data by using the following command: -

```
hive> select * from employee;
```

Arithmetic Operators in Hive

In Hive, the arithmetic operator accepts any numeric type. The commonly used arithmetic operators are: -

Operators	Description
A + B	This is used to add A and B.
A - B	This is used to subtract B from A.
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.

Examples of Arithmetic Operator in Hive

- Let's see an example to increase the salary of each employee by 50.

```
hive> select id, name, salary + 50 from employee;
```

Let's see an example to decrease the salary of each employee by 50.

```
hive> select id, name, salary - 50 from employee;
```

Let's see an example to find out the 10% salary of each employee.

```
hive> select id, name, (salary * 10) /100 from employee;
```

Relational Operators in Hive

In Hive, the relational operators are generally used with clauses like Join and Having to compare the existing records. The commonly used relational operators are: -

Operator	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.

Examples of Relational Operator in Hive

- Let's see an example to fetch the details of the employee having salary >=25000.

hive> select * from employee where salary >= 25000;

Let's see an example to fetch the details of the employee having salary <25000.

hive> select * from employee where salary < 25000;

HiveQL - Functions

The Hive provides various in-built functions to perform mathematical and aggregate type operations. Here, we are going to execute such type of functions on the records of the below table:

employee_data

Id	Name	Salary
1	Gaurav	30000
2	Aryan	20000
3	Vishal	40000
4	John	10000
5	Henry	25000
6	William	9000
7	Lisa	25000
8	Ronit	20000

Example of Functions in Hive

Let's create a table and load the data into it by using the following steps: -

- Select the database in which we want to create a table.

```
hive> use hql;
```

- Create a hive table using the following command: -

```
hive> create table employee_data (Id int, Name string , Salary float)
```

```
row format delimited
```

```
fields terminated by ',' ;
```

- Now, load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/emp_details' into table employee_data;
```

- Let's fetch the loaded data by using the following command: -

```
hive> select * from employee_data;
```

Now, we discuss mathematical, aggregate and other in-built functions with the corresponding examples.

Mathematical Functions in Hive

The commonly used mathematical functions in the hive are: -

Return type	Functions	Description
BIGINT	round(num)	It returns the BIGINT for the rounded value of DOUBLE num.
BIGINT	floor(num)	It returns the largest BIGINT that is less than or equal to num.
BIGINT	ceil(num), ceiling(DOUBLE num)	It returns the smallest BIGINT that is greater than or equal to num.
DOUBLE	exp(num)	It returns exponential of num.
DOUBLE	ln(num)	It returns the natural logarithm of num.
DOUBLE	log10(num)	It returns the base-10 logarithm of num.
DOUBLE	sqrt(num)	It returns the square root of num.
DOUBLE	abs(num)	It returns the absolute value of num.
DOUBLE	sin(d)	It returns the sin of num, in radians.
DOUBLE	asin(d)	It returns the arcsin of num, in radians.
DOUBLE	cos(d)	It returns the cosine of num, in radians.

DOUBLE	acos(d)	It returns the arccosine of num, in radians.
DOUBLE	tan(d)	It returns the tangent of num, in radians.
DOUBLE	atan(d)	It returns the arctangent of num, in radians.

Example of Mathematical Functions in Hive

- Let's see an example to fetch the square root of each employee's salary.

hive> select Id, Name, sqrt(Salary) from employee_data ;

Aggregate Functions in Hive

In Hive, the aggregate function returns a single value resulting from computation over many rows. Let's see some commonly used 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 from it.

DOUBLE	max(col)	It compares the values and returns the maximum one form it.
--------	----------	---

Examples of Aggregate Functions in Hive

- Let's see an example to fetch the maximum salary of an employee.

hive> select max(Salary) from employee_data;

Let's see an example to fetch the minimum salary of an employee.

hive> select min(Salary) from employee_data;

Other built-in Functions in Hive

The following are some other commonly used in-built functions in the hive: -

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.
TRING	rtrim(str)	It returns the string by removing whitespaces from right-hand side.

Examples of other in-built Functions in Hive

- Let's see an example to fetch the name of each employee in uppercase.

```
select Id, upper(Name) from employee_data;
```

Let's see an example to fetch the name of each employee in lowercase.

```
select Id, lower(Name) from employee_data;
```

HiveQL - GROUP BY and HAVING Clause

The Hive Query Language provides GROUP BY and HAVING clauses that facilitate similar functionalities as in SQL. Here, we are going to execute these clauses on the records of the below table:

emp

Id	Name	Salary	Department
1	Gaurav	30000	Developer
2	Aryan	20000	Manager
3	Vishal	40000	Manager
4	John	10000	Trainer
5	Henry	25000	Developer
6	William	9000	Developer
7	Lisa	25000	Manager
8	Ronit	20000	Trainer

GROUP BY Clause

The **HQL Group By** clause is used to group the data from the multiple records based on one or more column. It is generally used in conjunction with the aggregate functions (like SUM, COUNT, MIN, MAX and AVG) to perform an aggregation over each group.

Example of GROUP BY Clause in Hive

Let's see an example to sum the salary of employees based on department.

- Select the database in which we want to create a table.

```
hive> use hiveql;
```

Now, create a table by using the following command:

```
hive> create table emp (Id int, Name string , Salary float, Department string)
```

```
row format delimited
```

fields terminated by ',' ;

Load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;
```

- Now, fetch the sum of employee salaries department wise by using the following command:

```
hive> select department, sum(salary) from emp group by department;
```

HAVING CLAUSE

The HQL **HAVING clause** is used with **GROUP BY** clause. Its purpose is to apply constraints on the group of data produced by GROUP BY clause. Thus, it always returns the data where the condition is **TRUE**.

Example of Having Clause in Hive

In this example, we fetch the sum of employee's salary based on department and apply the required constraints on that sum by using HAVING clause.

- Let's fetch the sum of employee's salary based on department having sum >= 35000 by using the following command:

```
hive> select department, sum(salary) from emp group by department having sum(salary)>=35000;
```

Here, we got the desired output.

HiveQL - ORDER BY and SORT BY Clause

By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order. Here, we are going to execute these clauses on the records of the below table:

emp

Id	Name	Salary	Department
1	Gaurav	30000	Developer
2	Aryan	20000	Manager
3	Vishal	40000	Manager
4	John	10000	Trainer
5	Henry	25000	Developer
6	William	9000	Developer
7	Lisa	25000	Manager
8	Ronit	20000	Trainer

HiveQL - ORDER BY Clause

In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time.

Example of ORDER BY Clause in Hive

Let's see an example to arrange the data in the sorted order by using ORDER BY clause.

- Select the database in which we want to create a table.

```
hive> use hiveql;
```

Now, create a table by using the following command:

```
hive> create table emp (Id int, Name string , Salary float, Department string)  
row format delimited
```

fields terminated by ',' ;

Load the data into the table.

```
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;
```

Now, fetch the data in the descending order by using the following command:

```
hive> select * from emp order by salary desc;
```

Here, we got the desired result.

HiveQL - SORT BY Clause

The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately. It may also give a partially ordered result.

Example of SORT BY Clause in Hive

In this example, we arrange the data in the sorted order by using SORT BY clause.

- Let's fetch the data in the descending order by using the following command:

```
hive> select * from emp sort by salary desc;
```

Here, we got the desired result.

HiveQL - JOIN

The HiveQL Join clause is used to combine the data of two or more tables based on a related column between them. The various type of HiveQL joins are: -

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

Here, we are going to execute the join clauses on the records of the following table:

employee

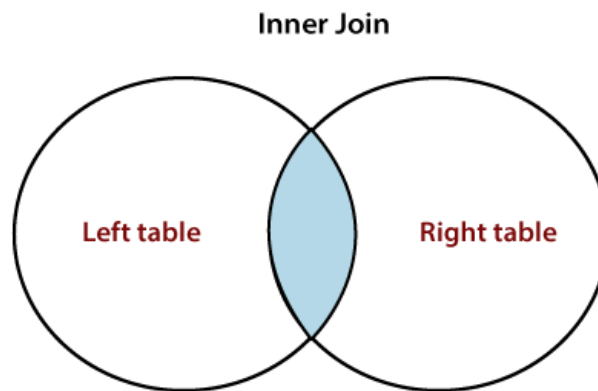
empid	empname	state
1	Gaurav	UP
2	Aryan	Punjab
3	Vishal	UP
4	John	Haryana
5	Henry	UP

employee_department

depid	department_name
2	IT
2	Trainer
3	Manager
4	Admin

Inner Join in HiveQL

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.



Example of Inner Join in Hive

In this example, we take two table employee and employee_department. The primary key (empid) of employee table represents the foreign key (depid) of employee_department table. Let's perform the inner join operation by using the following steps: -

- Select the database in which we want to create a table.

```
hive> use hiveql;
```

Now, create a table by using the following command:

```
hive> create table employee(empid int, empname string , state string)
row format delimited
fields terminated by ',' ;
```

Load the corresponding data into the table.

```
hive> load data local inpath '/home/codegyani/hive/employee' into table employee;
```

Now, create another table by using the following command:

```
hive> create table employee_department(depint int, department_name string)
row format delimited
```

fields terminated by ',' ;

Load the corresponding data into the table.

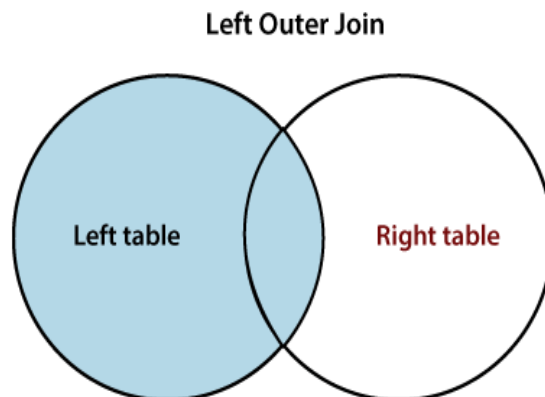
```
hive> load data local inpath '/home/codegyani/hive/employee_department' into table employee_department;
```

Now, perform the inner join operation by using the following command: -

```
hive> select e1.empname, e2.department_name from employee e1 join employee_department e2 on e1.empid= e2.depid;
```

Left Outer Join in HiveQL

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.



Example of Left Outer Join in Hive

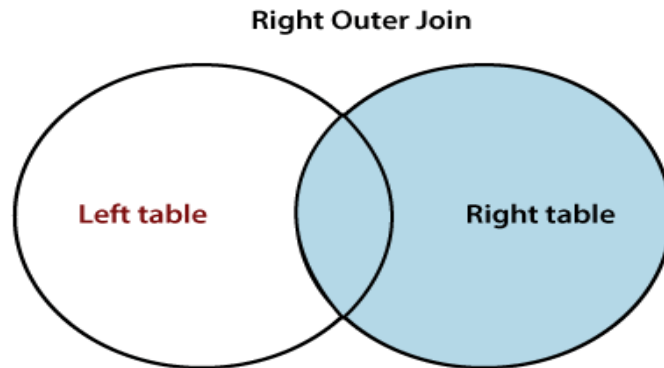
In this example, we perform the left outer join operation.

- Let's us execute the left outer join operation by using the following command: -

```
hive> select e1.empname, e2.department_name from employee e1 left outer join employee_department e2 on e1.empid= e2.depid;
```

Right Outer Join in HiveQL

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.



Example of Left Outer Join in Hive

In this example, we perform the left outer join operation.

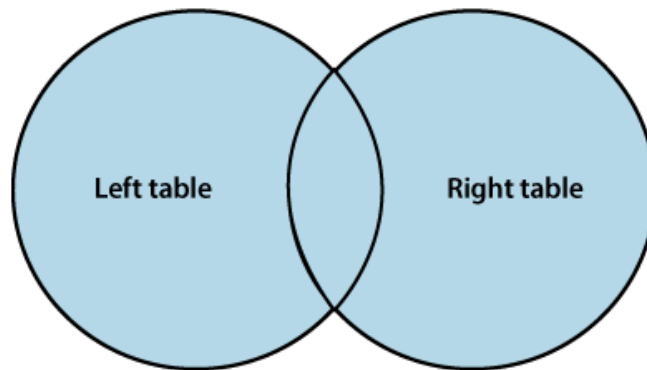
- Let's us execute the left outer join operation by using the following command: -

```
hive> select e1.empname, e2.department_name from employee e1 right outer join e  
mployee_department e2 on e1.empid= e2.depid;
```

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



Example of Full Outer Join in Hive

In this example, we perform the full outer join operation.

- Let's us execute the full outer join operation by using the following command: -

```
select e1.empname, e2.department_name from employee e1 full outer join employee  
e_department e2 on e1.empid= e2.depid;
```