Apuntes HIVE

**Introducción a Apache Hive:**

Apache Hive es un sistema de almacenamiento de datos distribuido y tolerante a fallos diseñado para trabajar con grandes volúmenes de datos (Big Data). Actúa como una interfaz familiar para los usuarios, proporcionando un lenguaje de consulta similar a SQL (HiveQL) para interactuar con datos almacenado

1. Sentencia a Hive en HiveQL (HQL), es muy parecido al SQL "Calcular el total de ventas en el año 2025".

SELECT product, SUM(quantity) AS total\_quantity

FROM sales

WHERE year = 2025

GROUP BY product

1. Hive traduce la sentencia y la transforma en una serie de tareas más pequeñas y sencillas que puede entender Hadoop, el sistema donde están almacenados tus datos. Estas tareas se llaman MapReduce.
2. Hadoop divide los datos en pedazos más pequeños y asigna cada tarea a un nodo diferente en el cluster. Cada trabajador realiza su parte del trabajo y luego envía los resultados a Hive.
3. Hive toma los resultados de todos los nodos y los combina para dar una respuesta a la secuencia inicial en Hadoop.

**Tipos de datos**

Los tipos de datos en Hive se clasifican en **cuatro tipos**:

**1. Tipos de columnas:** El tipo de columna se utiliza como tipos de datos de columna de Hive. Son los siguientes:

* **Enteros**: Los datos de tipo entero se pueden especificar utilizando tipos de datos integrales:
* INT: Entero (4 Byte)
* BIGINT: Cuando el rango de datos excede al INT (8 Byte)
* SMALLINT: Cuando es un INT pequeño (2 Bytes)
* TINYINT: Es más pequeño que SMALLINT. (1 Byte)
* **String**: Los tipos de datos de tipo cadena se pueden especificar usando comillas simples (' ') o comillas dobles (" "). Contiene dos tipos de datos: VARCHAR y CHAR. Hive sigue el escape de los tipos C caracteres.
* **Timestamp**: Admite la marca de tiempo UNIX tradicional con precisión de nanosegundos opcional. Admite el formato java.sql.Timestamp:
  + - “YYYY-MM-DD HH:MM:SS.ffffffffff”
    - “aaaa-mm-dd hh:mm:ss.ffffffffff”.
* **Fechas:** Los valores de FECHA se describen en formato de año/mes/día en la forma {{YYYY—MM--DD}}.
* **Union:** Union es una colección de tipos de datos heterogéneos. Puedes crear una instancia usando crear unión.
  + UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}

{1:2.0}

{2:["three","four"]}

{3:{"a":5,"b":"five"}}

{2:["six","seven"]}

{3:{"a":8,"b":"eight"}}

{0:9}

{1:10.0}

* **Decimales**: El tipo DECIMAL en Hive es igual que el formato Big Decimal de Java. Se utiliza para que representa una precisión arbitraria inmutable.

Sitaxis: DECIMAL(precision, scale)

decimal(10,0)

**2. Literales:**

* **Tipos de punto flotante:** Los tipos de punto flotante no son más que números con puntos decimales. Generalmente, esto tipo de datos se compone de tipo de datos DOUBLE.
* **Tipo decimal:** Los datos de tipo decimal no son más que un valor de punto flotante con un rango más alto que Tipo de dato DOBLE. El rango de tipo decimal es aproximadamente -10-308 to 10308

**3. Valores nulos:** Es la falta de valor, se especifica como NULL.

**4. Tipos complejos:**

* **Array**: Las matrices en Hive se usan de la misma manera que en Java.

ARRAY<tipo\_datos>

* **Mapas**: Los mapas en Hive son similares a Java Maps.

MAP<tipo\_primitivo, tipo\_datos>

* **Estructuras**: Structs en Hive es similar al uso de datos complejos con comentarios.

STRUCT<col\_name : data\_type [COMMENT col\_comment], ...>

Creación y eliminación de BBDD

**Creación de una Base de Datos**

En Apache Hive, una base de datos es un contenedor lógico para tablas y otros objetos. La sintaxis básica es la siguiente:

CREATE DATABASE nombre\_de\_la\_base\_de\_datos;

CREATE DATABASE mi\_base\_de\_datos;

Si se desea asegurar de que la base de datos solo se cree si no existe previamente:

CREATE DATABASE IF NOT EXISTS mi\_base\_de\_datos;

Para especificar una ubicación en HDFS para almacenar la base de datos:

CREATE DATABASE mi\_base\_de\_datos LOCATION '/user/hive/warehouse/mi\_base\_de\_datos';

**Listar Bases de Datos**

Para ver las bases de datos disponibles en el sistema:

SHOW DATABASES;

Para filtrar por un patrón específico:

SHOW DATABASES LIKE 'mi\_\*';

**Seleccionar una Base de Datos**

Antes de trabajar con tablas dentro de una base de datos específica, se debe seleccionar dicha bbdd:

USE mi\_base\_de\_datos;

**Eliminación de una Base de Datos**

La sintaxis básica es:

DROP DATABASE nombre\_de\_la\_base\_de\_datos;

DROP DATABASE mi\_base\_de\_datos;

Si la base de datos contiene tablas u otros objetos, el comando anterior fallará a menos que se use la opción `CASCADE`, que eliminará la base de datos junto con todo su contenido:

DROP DATABASE mi\_base\_de\_datos CASCADE;

Para eliminar la base de datos en caso de que exista, sin que cause un error:

DROP DATABASE IF EXISTS mi\_base\_de\_datos;

Creación, modificación y eliminación de tablas

**Creación de una Tabla**

Para crear una tabla en Hive, se utiliza el comando CREATE TABLE. La sintaxis básica es:

CREATE TABLE nombre\_de\_la\_tabla (

columna1 TIPO,

columna2 TIPO,

...

)

STORED AS formato;

Ejemplo:

CREATE TABLE empleados (

id INT,

nombre STRING,

edad INT,

salario FLOAT

)

STORED AS TEXTFILE;

Si deseas asegurarte de que la tabla solo se cree si no existe, usa:

CREATE TABLE IF NOT EXISTS empleados (

id INT,

nombre STRING,

edad INT,

salario FLOAT

)

STORED AS TEXTFILE;

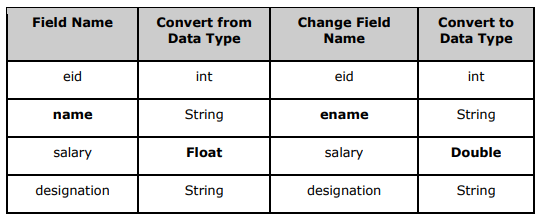
**Modificación de una Tabla**

Puedes modificar una tabla en Hive utilizando el comando ALTER TABLE. Algunas operaciones comunes incluyen:

Añadir una nueva columna:

ALTER TABLE empleados ADD COLUMNS (direccion STRING COMMENT ‘xxx’);

La siguiente tabla contiene los campos de la tabla de empleados y muestra los campos a cambiar (en negrita).



Las siguientes consultas cambian el nombre de la columna y el tipo de datos de la columna:

ALTER TABLE employee CHANGE name ename String;

ALTER TABLE employee CHANGE salary salary Double;

Para **remplazar**:

La siguiente consulta elimina todas las columnas de la tabla de empleados y reemplaza:

ALTER TABLE employee REPLACE COLUMNS (

eid INT empid Int,

ename STRING name String);

**Eliminación de una Tabla**

Para eliminar una tabla en Hive, usa el comando DROP TABLE:

DROP TABLE nombre\_de\_la\_tabla;

Ejemplo:

DROP TABLE empleados;

Si solo deseas eliminar la tabla si existe, sin causar un error si no está presente, usa:

DROP TABLE IF EXISTS empleados;

Particiones

Las particiones en Hive permiten organizar y mejorar el rendimiento de las consultas al dividir una tabla en segmentos más pequeños.

Por ejemplo, si tenemos una **tabla empleados** con los campos **id, nombre, departamento y año\_ingreso**, podemos particionarla por **año\_ingreso** para mejorar la eficiencia de las consultas:

CREATE TABLE empleados (

id INT,

nombre STRING,

departamento STRING

)

PARTITIONED BY (año\_ingreso INT)

STORED AS TEXTFILE;

Los datos se almacenarán en directorios separados dentro del sistema de archivos:

/empleados/año\_ingreso=2012/part-00000

/empleados/año\_ingreso=2013/part-00001

**Añadir Particiones:**

Las particiones deben agregarse manualmente utilizando ALTER TABLE:

ALTER TABLE empleados ADD PARTITION (año\_ingreso=2013) LOCATION '/empleados/2013';

Para cambiar el nombre de la partición:

ALTER TABLE empleados PARTITION (año\_ingreso=2013) RENAME TO PARTITION (año\_ingreso=2014);

**Listar Particiones**

Para ver las particiones de una tabla:

SHOW PARTITIONS empleados;

**Eliminar Particiones**

Para eliminar una partición específica:

ALTER TABLE empleados DROP PARTITION (año\_ingreso=2013);

HQL

Sintaxis:

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list]

[HAVING having\_condition]

[ORDER BY col\_list]]

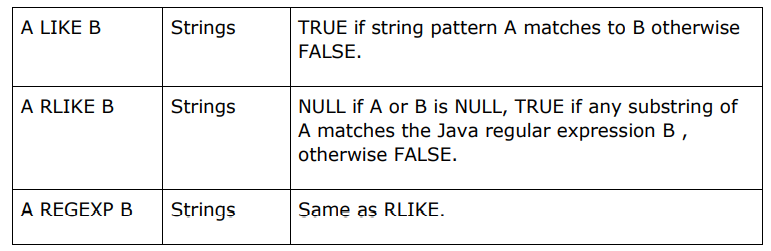
[LIMIT number];

Operadores

Existen los siguientes tipos:

* Operadores relacionales
* Operadores aritméticos
* Operadores lógicos
* Operadores complejos

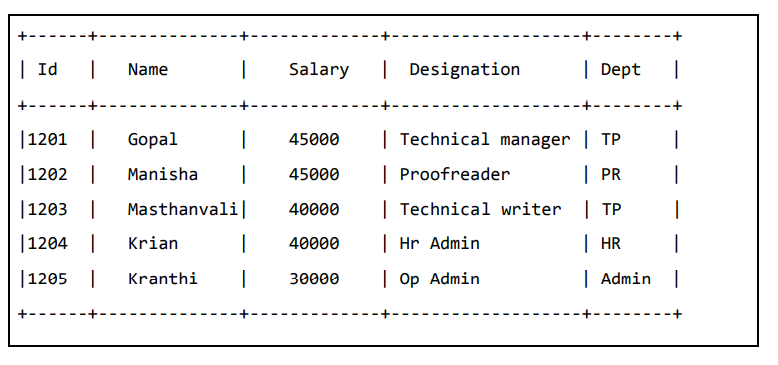
**Operadores relacionales:**

****

Ejemplo:

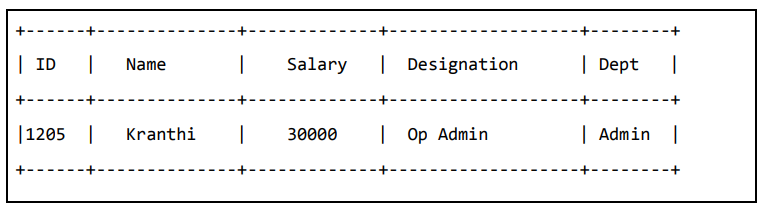
Supongamos que la tabla de empleados se compone de campos denominados Id, Name, Salary, Designation y Dept como se muestra a continuación.

Generar una consulta para recuperar al empleado detalles cuyo Id es 1205.

La query sería:

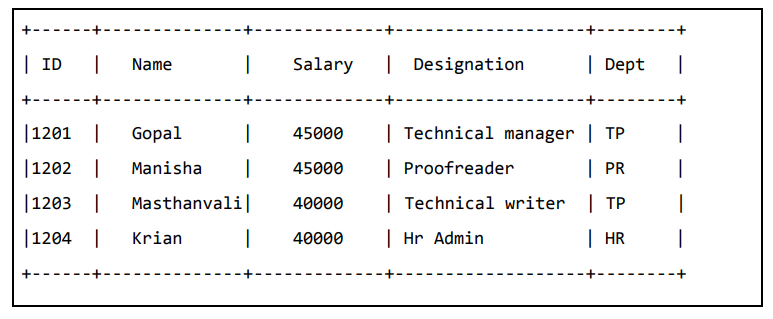
SELECT \* FROM employee WHERE Id=1205;

La consulta devolvería:

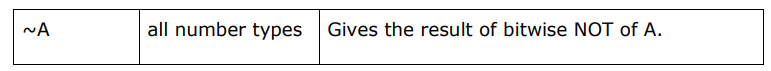
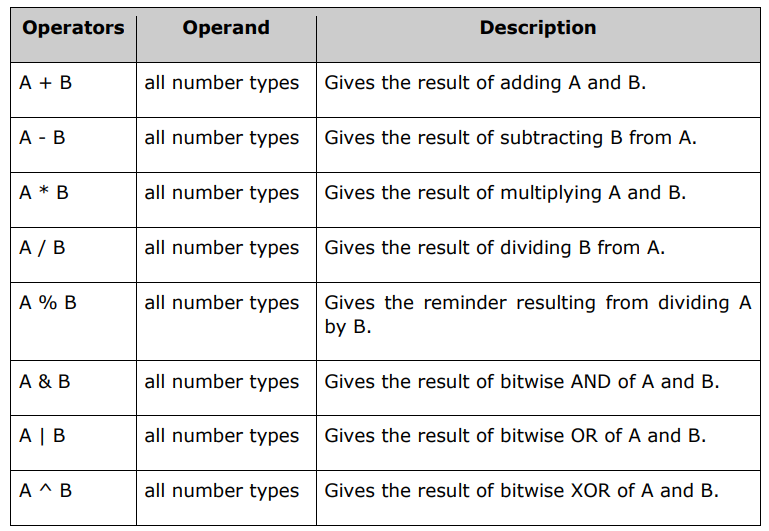


Buscar los empleados que tengan un salario superior a 40000:

SELECT \* FROM employee WHERE Salary>=40000;

**Operadores aritméticos:**

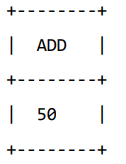
Los operadores aritméticos admiten varias operaciones aritméticas y todas ellas tipos numéricos. Los operadores aritméticos disponibles en Hive son los siguientes:



Ejemplo: Sumar 20 y 30

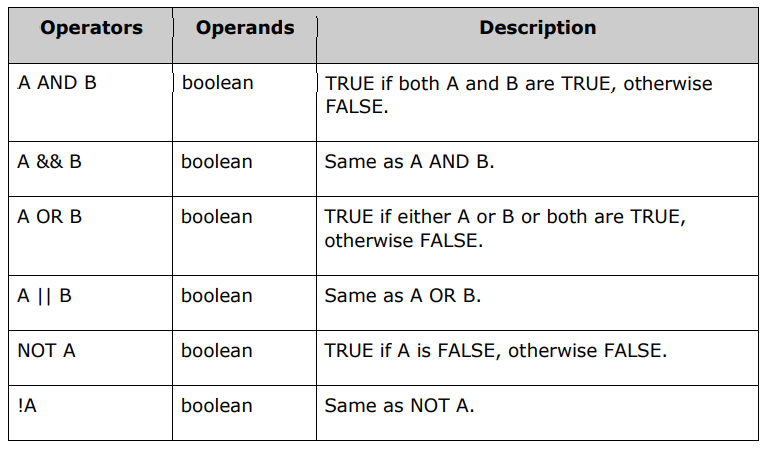
SELECT 20+30 ADD FROM temp;

El resultado:



**Operadores lógicos:**

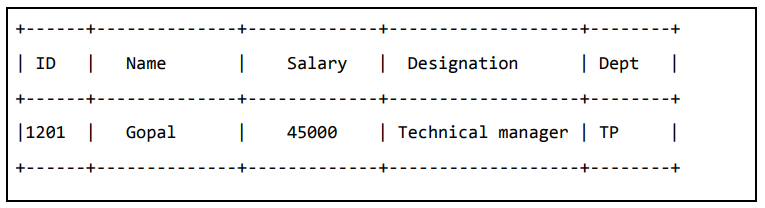
Los operadores son expresiones lógicas. Todos ellos devuelven VERDADERO o FALSO.



Ejemplo:

Consultar todos los empleados que pertenezcan al departamento TP y tenga un salario de más de 40000.

SELECT \* FROM employee WHERE Salary > 40000 && Dept = TP;

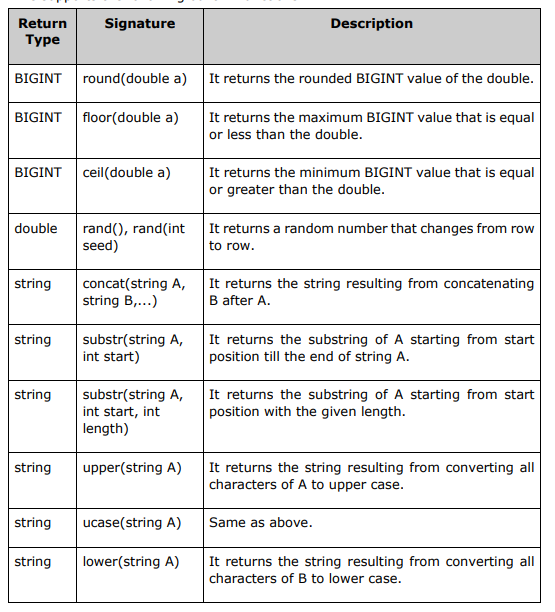
****

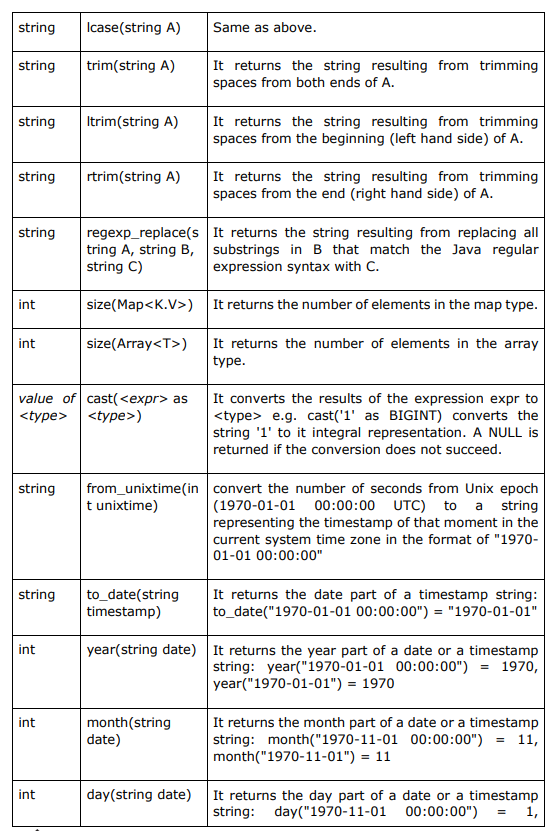
**Operadores complejos:**

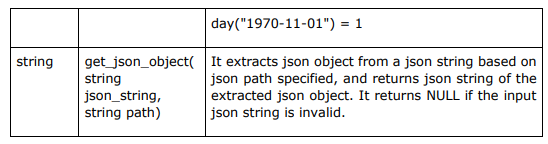
Los operadores complejos se refieren a los Arrays, Mapas y Estructuras

Funciones

Las funciones de Hive son muy similares a las de SQL:

****

****

****

Ejemplo:

round() function

SELECT round(2.6) from temp;

resultado: 3.0

floor() function

SELECT floor(2.6) from temp;

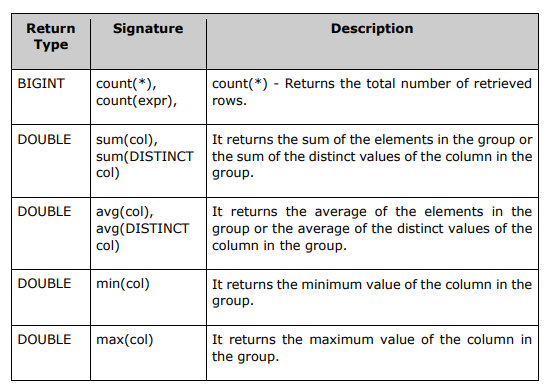
resultado: 2.0

ceil() function

SELECT ceil(2.6) from temp;

resultado: 3.0

Otras funciones:



Vistas e índices

El uso de la vista en Hive es la misma que la de la vista en SQL. Es un concepto RDBMS estándar. Podemos ejecutar todas las operaciones DML en una vista.

**Creación de vistas**

Sintaxis:

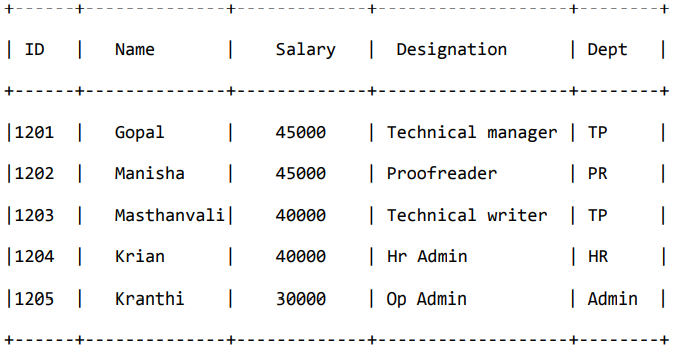
CREATE VIEW [IF NOT EXISTS] view\_name

[(column\_name [COMMENT column\_comment], ...) ]

[COMMENT table\_comment] AS SELECT ...

Ejemplo:

Supongamos una tabla empleados con los siguientes campos Id, Nombre, Salario, Designación y Dept. Crear una vista llamada emp\_30000 que contenga todos los empleados que ganen un salario de más de 30000.



CREATE VIEW emp\_30000 AS

SELECT \* FROM employee WHERE salary>30000;

**Borrado de vistas**

Sintaxis:

DROP VIEW view\_name

Ejemplo:

Eliminar la vista llamada emp\_30000

DROP VIEW emp\_30000;

**Creación de índices**

Un índice no es más que un puntero de una columna de una tabla.

Sintaxis:

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...) AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property\_name=property\_value, ...)]

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[ [ ROW FORMAT ...] STORED AS ... | STORED BY ... ]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

Ejemplo:

Crear un índice en la tabla empleados llamado index\_salary en la columna de salarios de la tabla.

CREATE INDEX inedx\_salary ON TABLE employee(salary)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';

**Eliminar índices**

Sintaxis:

DROP INDEX ON

Ejemplo:

DROP INDEX index\_salary ON employee;

HQL

Joins

JOINS es una cláusula que se usa para obtener datos de más de una tabla.

Sintaxis:

join\_table:

table\_reference JOIN table\_factor [join\_condition]

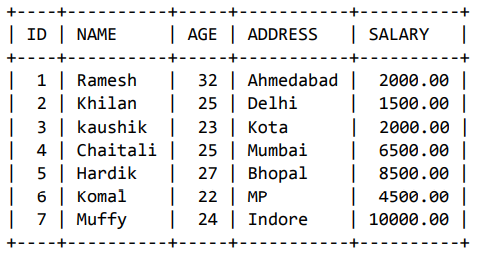
| table\_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table\_reference join\_condition

| table\_reference LEFT SEMI JOIN table\_reference join\_condition

| table\_reference CROSS JOIN table\_reference [join\_condition]

Ejemplo:

Usaremos las siguientes dos tablas: Customers y orders

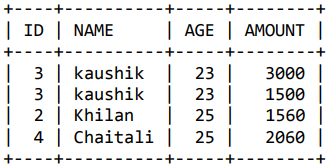
**JOIN**

La sentencia join enlaza las dos o más tablas. En este caso, saca los datos de customers de aquellos ids que existan en la tabla orders

SELECT c.ID, c.NAME, c.AGE, o.AMOUNT

FROM CUSTOMERS c JOIN ORDERS o > ON (c.ID = o.CUSTOMER\_ID);

Resultado:



**LEFT OUTER JOIN**

HiveQL LEFT OUTER JOIN devuelve todas las filas de la tabla de la izquierda, incluso si hay

no hay coincidencias en la tabla de la derecha. Esto significa que si la cláusula ON coincide con 0 (cero) registros en la tabla de la derecha, JOIN aún devuelve una fila en el resultado, pero con NULL en cada columna de la tabla de la derecha.

A LEFT JOIN devuelve todos los valores de la tabla de la izquierda, más los valores coincidentes de la tabla de la derecha, o NULL en caso de que no coincida.

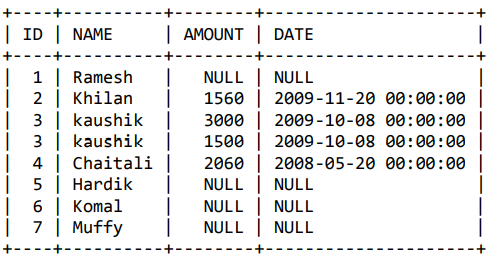
La siguiente consulta demuestra la LEFT OUTER JOIN entre CUSTOMERS y tablas de ORDEN:

SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

LEFT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER\_ID);

Resultado:



**RIGHT OUTER JOIN**

Es lo mismo que en LEFT OUTER JOIN con la diferencia que la que manda es la tabla de derecha. Es decir, devuelve todos los valores de la tabla de la derecha, más los valores coincidentes de la tabla de la izquierda, o NULL en caso de que no coincida.

La siguiente consulta demuestra la RIGHT OUTER JOIN entre CUSTOMERS y tablas de ORDEN:

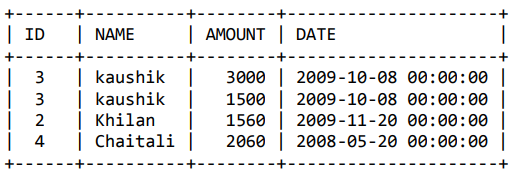
SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

RIGHT OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER\_ID);

Resultado:



**FULL OUTER JOIN**

HiveQL FULL OUTER JOIN combina los registros de la tabla de la izquierda y de la

tablas de la derecha que cumpla la condición JOIN. La tabla unida contiene todos los

registros de ambas tablas, y si no existe el dato lo completa con el valor NULL.

SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

FULL OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER\_ID);

Resultado:

