

# Base de Datos

Una base de datos es una colección de datos que están organizados en una manera que facilita el acceso sencillo, como la administración y actualización eficiente.

Una base de datos está constituida por tablas que almacenan información relevante.

Por ejemplo, en PedidosYa utilizamos varias base de datos, cada una de ellas contiene mucha información como usuarios, órdenes, restaurantes, etc.

# Tablas de Bases de Datos

Una tabla almacena y despliega datos en un formato estructurado que consiste en **columnas** y **filas**, las cuales son similares a aquellas vistas en hojas de cálculo de Excel.

Las Bases de Datos usualmente contienen múltiples tablas, cada una designada para un propósito en específico. Por ejemplo, tener una tabla con los usuarios y otra con las direcciones de los mismos.

# Claves Primarias

Una clave primaria es un campo en la tabla que identifica de forma única los registros de la tabla.

Las principales características de una clave primaria son:

- Debe tener un **valor único** para cada fila.
- No puede contener valores **NULL**.
- Una tabla está limitada a **UNA** clave primaria.

Por ejemplo, nuestra tabla contiene un registro para cada usuario. El número de ID único sería una buena elección para una clave primaria en la tabla, ya que existen personas con el mismo nombre por lo tanto el nombre dejaría de ser una clave única (primaria). Nuestra ID sería un autoincremental, que cada vez que se registra un nuevo usuario se le asigna un nuevo identificador.

## Introducción a Bases de Datos

# ¿Qué es SQL?

Una vez que entienda lo que es una base de datos, entender SQL es sencillo. SQL significa **S**tructured **Q**uery **L**anguage (Lenguaje de Consulta Estructurada)

**SQL** es utilizado para acceder y manipular una **base de datos**.

**HUE** como muchos otros (MySQL, Redshift, SQL Server, etc), es un programa que entiende SQL.

En SQL se puede:

- insertar, actualizar o borrar registros en una base de datos.
- crear nuevas base de datos, tablas, procedimientos almacenados, vistas.
- recuperar datos de una base de datos, etc.

Nota: SQL es un estándar ANSI(Instituto Nacional Estadounidenses de Estándares), pero hay diferentes versiones del lenguaje SQL. La mayoría de los programas de base de datos SQL, tienen sus propias extensiones propietarias adicionales al estándar SQL, pero todos ellos soportan los comandos principales.

## Declaraciones SQL

# SELECT

La declaración SELECT es utilizada para seleccionar datos de una base de datos. El resultado es almacenado en una tabla resultante, la cual es llamada el conjunto resultante.

Una consulta puede recuperar información de columnas seleccionadas o de todas las columnas en una tabla. Para crear una simple declaración SELECT, especifica el(los) nombre(s) de la(s) columna(s) que necesitas de la tabla.

La sintaxis de la declaración SQL SELECT es:

```
SELECT column_list  
FROM table_name
```

- `column_list` incluye una o más columnas de las cuales los datos son recuperados
- `table_name` es el nombre de la tabla de la cual la información es recuperada

## Múltiples consultas

SQL permite ejecutar múltiples consultas o comandos en la misma línea.

La siguiente declaración SQL selecciona las columnas `user_name` & `user_email` de la tabla `user`.

```
SELECT user_name FROM datalake.users;  
SELECT user_email FROM datalake.users;
```

# Sensibilidad a mayúsculas

SQL es **insensible** a mayúscula. Las siguientes declaraciones son equivalente y producirán el mismo resultado

```
select user_name from datalake.users;  
SELECT user_name FROM datalake.users;  
sELECT user_name FRom datalake.users;
```

## Reglas de sintaxis de SQL

# Reglas de sintaxis

Una única declaración SQL puede ser colocada en una o más líneas de texto. Adicionalmente, múltiples declaraciones SQL. Puede ser combinada en una sola línea de texto.

Los espacios en blanco y las líneas múltiples son ignorados en SQL.  
Por ejemplo, la siguiente consulta es absolutamente correcta.

```
SELECT      user_name  
  
FROM datalake.users;
```

Sin embargo, es recomendable evitar espacio blanco innecesarios y líneas (dado que facilita la lectura de la misma).



Seleccionado múltiples columnas

## Selección más de una columna

Como fue mencionado anteriormente, la declaración SQL SELECT recupera los registros de tablas en tu base de datos SQL.

Puede seleccionar múltiples columnas de tablas de una vez.

Solo listar los nombres de las columnas, separados por comas:

```
SELECT user_name, user_last_name, user_email  
FROM datalake.users;
```

Seleccionado múltiples columnas

## Seleccionando todas las columnas

Para recuperar toda la información contenida en un tu tabla, coloca un signo de asterisco (\*) después del comando SELECT, en lugar de escribir cada nombre de columna por separado.

La siguiente delación SQL selecciona todas las columnas de la tabla user.

```
SELECT * FROM dataLake.users;
```

**Nota:** Deberíamos limitar el uso del **select \*** dado que cuanto más columnas traigamos más costoso (demora más) es obtener la información.

## Palabras Claves

# DISTINCT

Una tabla, en algunas situaciones, puede contener registros duplicados. Ejemplo la tabla de users, puede tener varios nombres iguales. Y si quisiéramos listar los nombres puede que tenga mucho más sentido retornar sólo registros únicos (nombres), en lugar de recuperar los duplicados.

La palabra clave DISTINCT en SQL es utilizada en conjunto con SELECT para eliminar todos los registros duplicados y retornar solo los registros únicos.

La sintaxis básica de DISTINCT es como se muestra a continuación:

```
SELECT DISTINCT user_name  
FROM datalake.users;
```

## Palabras Claves

# LIMIT

Por defecto, todos los resultados que satisfacen las condiciones especificadas en una declaración SQL son retornados. Sin embargo, algunas veces necesitamos recuperar un subconjunto de registros. En MySQL, estos se logra utilizando la palabra clave LIMIT.

La sintaxis para LIMIT es como sigue:

```
SELECT user_id, user_name, user_last_name  
FROM datalake.users LIMIT 5;
```

También puedes tomar un conjunto de registros de un desplazamiento particular.

En el siguiente ejemplo, hemos tomado cinco registros, comenzando desde la quinta posición.

```
SELECT user_id, user_name, user_last_name  
FROM datalake.users LIMIT 5, 10;
```

## Ordenando resultados

# ORDER BY

ORDER BY es utilizado con SELECT para ordenar los datos recuperados.

Los siguientes ejemplos ordenan nuestra tabla **user** por la columna **first\_name**.

```
SELECT user_id, user_name, user_email FROM datalake.users
WHERE user_name <> ''
ORDER BY user_name;
```

**Nota:** En esta consulta buscamos únicamente aquellos usuarios que tiene nombres distinto a vacío. Más adelante se explicará el uso del where.

Como pueden ver, las filas son ordenadas alfabéticamente por la columna **user\_email**.

Por defecto se ordena ASCENDENTE, pero en caso de querer hacerlo DESCENDENTE debemos poner **ORDER BY first\_name DESC**.

Ordenando resultados

## Ordenar múltiples columnas

ORDER BY puede ordenar múltiples columnas los datos recuperados. Cuando utilizamos ORDER BY con más de una columna, debemos separar la lista de columna posterior de ORDER BY con **comas**.

Ejemplo:

```
SELECT user_id, user_name, user_last_name FROM datalake.users
WHERE user_name <> ''
AND user_last_name <> ''
ORDER BY user_name, user_last_name DESC;
```

## Declaraciones

# La declaración WHERE

La cláusula WHERE es utilizada para extraer sólo esos registros que cumplen con un criterio específico.

La sintaxis para la cláusula WHERE:

```
SELECT column_list  
FROM table_name  
WHERE condition;
```

En la tabla anterior, para seleccionar un registro específico:

```
SELECT user_id, user_name, user_email  
FROM datalake.users  
WHERE user_id = 7;
```

# Operadores SQL

Los operadores de comparación y los Operadores Lógicos son utilizados en la cláusula **WHERE** para filtrar los datos a ser seleccionados.

Los siguientes operadores de comparación pueden ser utilizados en la cláusula **WHERE**

Operador	Descripción
=	Igual
<> o !=	No igual
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
BETWEEN	Dentro de un rango inclusivo



Declaraciones

# Operadores SQL

Por ejemplo, podemos mostrar todos los nombres listados en nuestra tabla “users” con la excepción de aquel con ID igual a 5.

```
SELECT user_id, user_name, user_email  
FROM datalake.users  
WHERE user_id <> 5;
```

## Declaraciones

# El operador BETWEEN

El operador BETWEEN selecciona valores dentro de un rango. El primer valor debe ser el límite menor y el segundo valor es el límite superior.

La sintaxis para la cláusula BETWEEN es como sigue:

```
SELECT column_name  
FROM table_name  
WHERE column_name BETWEEN value1 AND value2;
```

La siguiente declaración SQL selecciona todos los registros con los ID que están entre 3 y 7:

```
SELECT user_id, user_name, user_email  
FROM datalake.users  
WHERE user_id BETWEEN 3 AND 7;
```

## Declaraciones

# Valores de texto

Cuando trabajamos con columnas de texto, rodea cualquier texto que aparezca en la declaración **comillas simples** (').

La siguiente declaración SQL selecciona todos los registros en los cuales su nombre (columna **user\_name**) es igual a 'Guillermo'

```
SELECT user_id, user_name, user_email  
FROM datalake.users  
WHERE user_name = 'Guillermo';
```

Filtrado con AND, OR

## Operadores Lógicos

Los operadores lógicos pueden ser utilizados para combinar dos valores Booleanos y retornar un resultado true(verdadero), false(falso) o nulo(null).

Los siguientes operadores pueden ser utilizados:

Operador	Descripción
AND	TRUE si ambas expresiones son TRUE
OR	TRUE si alguna expresión es TRUE
IN	TRUE si el operando es igual a uno de una lista de expresiones
NOT	Retorna TRUE si la expresión no es TRUE

## Filtrado con AND, OR

# AND

Cuando recuperamos datos utilizando una declaración SELECT, utiliza operadores lógicos en la cláusula WHERE para combinar múltiples condiciones.

Si tú quieres seleccionar filas que satisfacen todas las condiciones dadas, utiliza operador lógico, AND.

Para encontrar los usuarios de la tabla “users” que hayan generado su primera orden en el mes de enero del 2019, arma una consulta como se muestra aquí:

```
SELECT user_id, user_name, user_email, user_first_order_date_id
FROM datalake.users
WHERE user_first_order_date_id >= 20190101
AND user_first_order_date_id < 20190201;
```

## Filtrado con AND, OR

# OR

Si quieres seleccionar filas que satisfacen al menos una de las condiciones dadas, puedes utilizar el operador lógico OR.

La siguiente tabla describe cómo el operador lógico OR funciona:

Condición1	Condición2	Resultado
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

## Filtrado con AND, OR

# OR

Por ejemplo, si quieres encontrar registros de la tabla “users” cuya país registrado en la columna “user\_country\_id” es igual a “Uruguay” o “Argentina”, la consulta se vería así:

```
SELECT user_id, user_name, user_email, user_country_id
FROM datalake.users
WHERE user_country_id = 1
OR user_country_id = 3;
```

**Nota:** user\_country\_id (1: Uruguay, 2: Chile, 3: Argentina...), para ver todos los country\_id podemos consultar la tabla datalake.countries.

Filtrado con AND, OR

## Combinando AND y OR

Las condiciones SQL AND y OR pueden ser combinadas para validar múltiples condiciones en una consulta. Estos dos operadores son llamados **operadores en conjunto**.

Cuando combinamos estas condiciones, es importante utilizar **paréntesis**, para que el orden para evaluar cada condición sea conocido.

La siguiente declaración selecciona todos los registros de “users” que tienen primera orden en enero 2019 en la columna “user\_first\_order\_date\_id” **AND** con país Uruguay **OR** Argentina en la columna “user\_country\_id”:

```
SELECT user_id, user_name, user_email, user_first_order_date_id, user_country_id
FROM datalake.users
WHERE user_first_order_date_id >= 20190101
AND user_first_order_date_id < 20190201
AND (user_country_id = 1
OR user_country_id = 3);
```



Declaraciones IN, NOT IN

## El operador IN

El operador IN es utilizado cuando quieres comparar una columna con más de un valor.

Por ejemplo, puedes necesitar seleccionar todos los registros de “users” países “Uruguay”, “Argentina” y “Chile” en la columna “user\_country\_id”.

Con la condición OR, tu SQL se vería así:

```
SELECT user_id, user_name, user_email, user_country_id FROM datalake.users
WHERE user_country_id = 1
OR user_country_id = 2
OR user_country_id = 3
```

Puedes lograr el mismo resultado con una simple condición IN, en lugar de múltiples condiciones OR:

```
SELECT user_id, user_name, user_email, user_country_id FROM datalake.users
WHERE user_country_id IN (1,2,3)
```

Declaraciones IN, NOT IN

## El operador NOT IN

El operador NOT IN te permite excluir una lista de valores específicos del conjunto de resultados.

Si añadimos la palabra clave NOT antes de IN en nuestra consulta anterior, los registros de la tabla “users” con países “Uruguay”, “Argentina” y “Chile” en la columna “user\_country\_id” serán excluidos.

```
SELECT user_id, user_name, user_email, user_country_id
FROM datalake.users
WHERE user_country_id NOT IN (1,2,3)
```

# La función CONCAT

La función CONCAT es utilizada para concatenar dos o más valores de texto y retorna la cadena de texto concatenada.

Vamos a concatenar la columna **user\_name** con la columna **user\_last\_name**, separándolas con una coma:

```
SELECT CONCAT(user_name , ', ', user_last_name)
FROM datalake.users;
```

El resultado obtenidos es:



```
1 SELECT concat(user_name, ', ', user_last_name)|
2 FROM users
3 limit 100
```

Historial de consultas	Consultas guardadas	Creador de consultas	Resultados (100+)
<b>_c0</b>			
1	Andrea, Chocron		
2	Bettina, Rocca		
3	Cece, Da Silva		
4	CÃ©sar, Sagues		

## Columnas personalizadas

# La palabra clave AS

Los resultados de una concatenación resultan en una nueva columna. El nombre predeterminado de la columna será el de la función CONCAT.

Puedes asignar un nombre personalizado a la columna resultante utilizando la palabra clave.

**AS:**

```
SELECT CONCAT(user_name , ' , ' , user_last_name) AS new_column  
FROM datalake.users;
```

Y cuando ejecutes la consulta,  
el nombre de la columna aparecerá  
modificado.

```
1 SELECT concat(user_name, ' , ' , user_last_name) as new_column  
2 FROM users  
3 limit 100
```

Historial de consultas 🔍 📄

Consultas guardadas 🔍

Creador de consultas

Resultados (100+) 🔍 ↗

new\_column




1	Andrea, Chocron
2	Bettina, Rocca
3	Cece, Da Silva

## Columnas personalizadas

# Operadores aritméticos

Los operadores aritméticos ejecutan operaciones aritméticas sobre operandos numéricos. Los operadores aritméticos incluyen adición (+), sustracción (-), multiplicación (\*) y división (/).

La siguiente tabla orders muestra el nombre del restaurante y el valor de la orden.

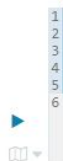


	order_id	restaurant_name	total_amount	discount_amount
1	972	Sushi Pop	327.2	81.8
2	422	Pancho Villa	329.4	36.6
3	222	Sushi Pop	216	54
4	641	Tele Pizza Paulistana Asa Norte	32.76	3.64
5	550	Charito Pizza Villa Urquiza	136.8	15.2
6	684	El Noble Microcentro	121.6	30.4
7	1901	Otaku Sushi Belgrano	136.5	58.5
8	456	Pan y Pizza	422.5	67.5
9	183	Los Gigantes de la Pizza II	94.5	10.5
10	244	Empanadas Gourmet Flores	83.3	14.7

# Operadores aritméticos

En el siguiente ejemplo quiero calcular cuál era el monto total sin el descuento.

```
SELECT order_id, restaurant_name, (total_amount + discount_amount) as amount_no_discount
FROM datalake.orders
WHERE state_id = 1
AND discount_amount > 0
LIMIT 100;
```



**Nota:** state\_id = 1 (órdenes confirmada)  
y discount\_amount > 0 para obtener  
aquellas órdenes que realmente se le  
aplicó descuento.

Historial de consultas 🔍 📄 Consultas guardadas 🔍 Creador de consultas

Resultados (100+) 🔍 📄

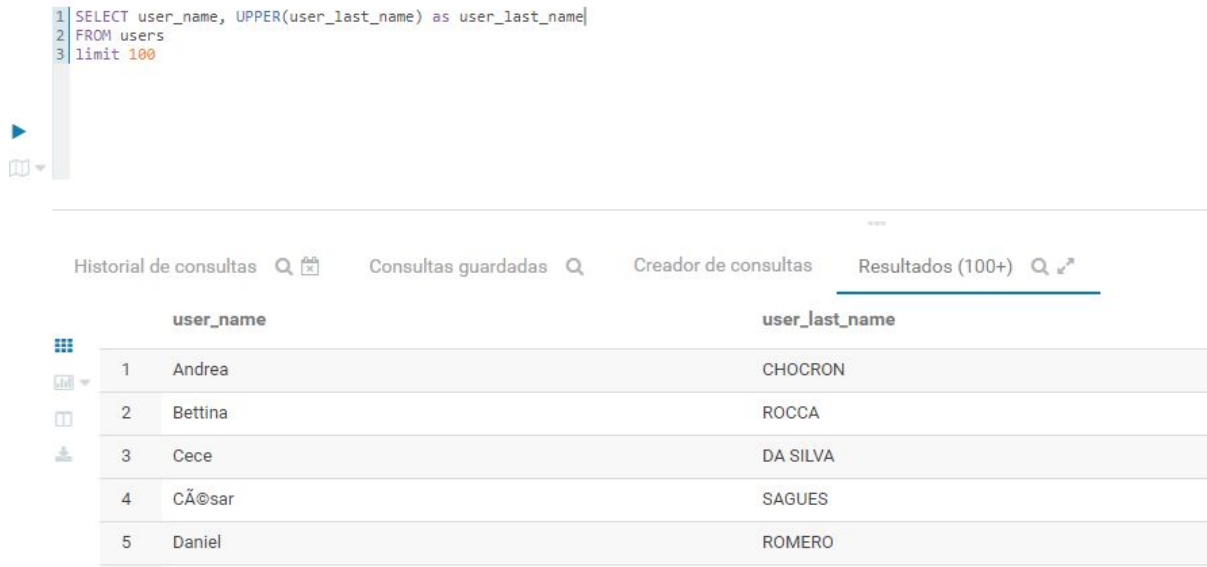
	order_id	restaurant_name	amount_no_discount
1	972	Sushi Pop	409
2	422	Pancho Villa	366
3	222	Sushi Pop	270
4	541	Tele Pizza Paulistana Asa Norte	36.4
5	550	Charito Pizza Villa Urquiza	152
6	1684	El Noble Microcentro	152

## Funciones

# La función UPPER

La función UPPER convierte todas las letras en la cadena de texto especificada a mayúsculas y la función LOWER en minúsculas.

```
SELECT user_name, UPPER(user_last_name) as user_last_name  
FROM datalake.users;
```



```
1 SELECT user_name, UPPER(user_last_name) as user_last_name|  
2 FROM users  
3 limit 100
```

Historial de consultas 🔍 📅 Consultas guardadas 🔍 Creador de consultas Resultados (100+) 🔍 📄

	user_name	user_last_name
1	Andrea	CHOCRON
2	Bettina	ROCCA
3	Cece	DA SILVA
4	CÃ©sar	SAGUES
5	Daniel	ROMERO

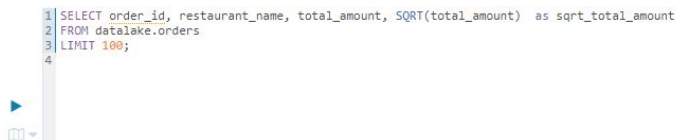
## Funciones

# SQRT

La función SQRT retorna la raíz cuadrada de un valor dado en el argumento.

Vamos a calcular la raíz cuadrada de cada monto en la columna “total\_amount”.

```
SELECT order_id, restaurant_name, total_amount, SQRT(total_amount) as sqrt_total_amount  
FROM datalake.orders  
LIMIT 100;
```



```
1 SELECT order_id, restaurant_name, total_amount, SQRT(total_amount) as sqrt_total_amount  
2 FROM datalake.orders  
3 LIMIT 100;  
4
```



	order_id	restaurant_name	total_amount	sqrt_total_amount
1	57	San Telmo Centro	85	9.219544457292887
2	51	San Telmo Centro	110	10.488088481701515
3	52	San Telmo Centro	110	10.488088481701515
4	72	Juve Chivitos	100	10



## Funciones

# AVG

De manera similar, la función AVG retorna el valor promedio de una columna numérica:

```
SELECT AVG(total_amount)
FROM datalake.orders;
```

## Funciones

# SUM

La función SUM es utilizada para calcular la suma de los valores de una columna.

Por ejemplo, para obtener la suma de todas las órdenes en la tabla orders, nuestra consulta SQL se vería así:

```
SELECT SUM(total_amount) as total  
FROM datalake.orders;
```

## Funciones

# COUNT

La función COUNT devuelve el número de filas que coinciden con un criterio específico.

Por ejemplo, para obtener la cantidad de todas las órdenes confirmadas, nuestra consulta SQL se vería así:

```
SELECT COUNT(order_id)
FROM datalake.orders;
WHERE state_id = 1;
```

## Funciones

# COUNT

El **DISTINCT** además se puede utilizar dentro de las funciones (y no necesariamente el COUNT). La misma sirve para contabilizar los valores únicos.

Por ejemplo, si quiero saber cual es la cantidad de usuarios diferentes que tuvieron órdenes confirmadas, la consulta sería:

```
SELECT COUNT(DISTINCT user_id)
FROM datalake.orders;
WHERE state_id = 1;
```

## Subconsultas

# Subconsultas

Consideramos el siguiente ejemplo. Pudiéramos necesitar la lista de todos los restaurantes cuyos montos de las órdenes sean mayores que el promedio en Uruguay para el mes Enero 2019.

Primero, calculamos el promedio.

```
SELECT AVG(total_amount)
FROM datalake.orders
WHERE registered_date >= '2019-01-01'
AND registered_date < '2019-02-01'
AND state_id = 1
AND country_id = 1
```

## Subconsultas

# Subconsultas

Como ya conocemos el promedio, podemos utilizar un simple WHERE para listar los únicos restaurantes que tienen órdenes con órdenes mayores del promedio.

```
SELECT DISTINCT restaurant_id, restaurant_name
FROM datalake.orders
WHERE registered_date >= '2019-01-01'
AND registered_date < '2019-02-01'
AND state_id = 1
AND country_id = 1
AND total_amount > 300;
```

**Nota:** El número manejado en el ejemplo es inventado.

```
1 SELECT DISTINCT restaurant_id, restaurant_name
2 FROM datalake.orders
3 WHERE registered_date >= '2019-01-01'
4 AND registered_date < '2019-02-01'
5 AND state_id = 1
6 AND country_id = 1
7 AND total_amount > 300
8 LIMIT 100;
9
```

Historial de consultas		Consultas guardadas	Creador de consultas	Resultados (100+)
restaurant_id		restaurant_name		
1	54	Alto Palermo Aguada		
2	107	Minie Foods		
3	108	La Taberna Del Diablo Pocitos		
4	512	Bamboo		

## Subconsultas

# Subconsultas

Una sola subconsulta retornará el mismo resultado de forma más sencilla.

```
SELECT DISTINCT restaurant_id, restaurant_name
FROM datalake.orders
WHERE registered_date >= '2019-01-01'
AND registered_date < '2019-02-01'
AND state_id = 1
AND country_id = 1
AND total_amount > (
    SELECT AVG(total_amount)
    FROM datalake.orders
    WHERE registered_date >= '2019-01-01'
    AND registered_date < '2019-02-01'
    AND state_id = 1
    AND country_id = 1
)
```

**Nota:** Las subconsultas nos permiten obtener resultados con condiciones **no** estáticas.

Combinando tablas

# Combinando tablas

Todas las consultas mostradas hasta ahora han seleccionado los valores de una sola tabla a la vez.

Una de las características más beneficiosas de SQL es la habilidad de combinar datos de dos o más tablas.

En las dos tablas que siguen, la tabla llamada users almacena información acerca de los usuarios.

La tabla orders almacena información acerca de órdenes individuales con su monto correspondiente:

**Nota:** En SQL, “combinar tablas” significa combinar datos de dos o más tablas. La combinación de tablas crea una tabla temporal que muestra la data de las tablas combinadas.



## Combinando tablas

# Combinando tablas

En lugar de almacenar el nombre del comprador en ambas tablas, la tabla **orders** contiene una referencia al **ID del usuario** que aparece en la tabla “**users**”. Este enfoque es más eficiente, al contrario de almacenar el mismo valor de texto en ambas tablas. Para poder seleccionar los datos correspondiente de ambas tablas, necesitaremos **combinar** ambas con esa condición.

## Combinando tablas

# Combinando tablas

Para combinar ambas tablas, especifica ambas como una lista separada por comas en la cláusula FROM:

```
SELECT users.id, users.name, orders.registred_date, orders.total_amount  
FROM datalake.users, datalake.orders  
WHERE users.id = orders.user_id
```

Fíjate que la cláusula WHERE “combina” las tablas con la condición de que el ID de la tabla users debe ser igual al user\_id de la tabla **orders**.

**Nota:** Más adelante encontraremos otra forma equivalente de combinar tablas, llamado “*inner join*”.

Tipo de combinaciones

## Nombres personalizados

Los nombres personalizados también pueden ser utilizados para las tablas. Pueden acortar las declaraciones dándoles “apodos” a las tablas.

```
SELECT u.user_id, u.user_name, o.registred_date, o.total_amount  
FROM datalake.users AS u, datalake.orders AS o  
WHERE u.user_id = o.user_id
```

Tipos de combinaciones

# Tipos de combinaciones

Los siguientes son los tipos de COMBINACIONES que pueden ser utilizados en HUE (y otros motores):

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

## Tipos de combinaciones

# INNER JOIN

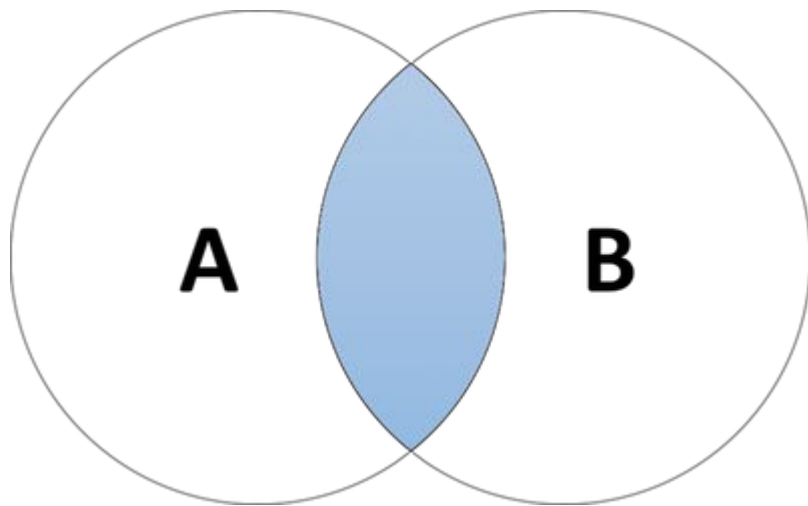
INNER JOIN es equivalente a JOIN retorna las filas cuando hay una coincidencias entre las tablas.

### Sintaxis:

```
SELECT column_name(s)
FROM table1 as A
INNER JOIN table2 as B
ON B.column_name = A.column_name;
```

Observemos en la palabra clave **ON**, para especificar la condición.

La siguiente imagen muestra cómo funciona el INNER JOIN:



## Tipos de combinaciones

# LEFT JOIN

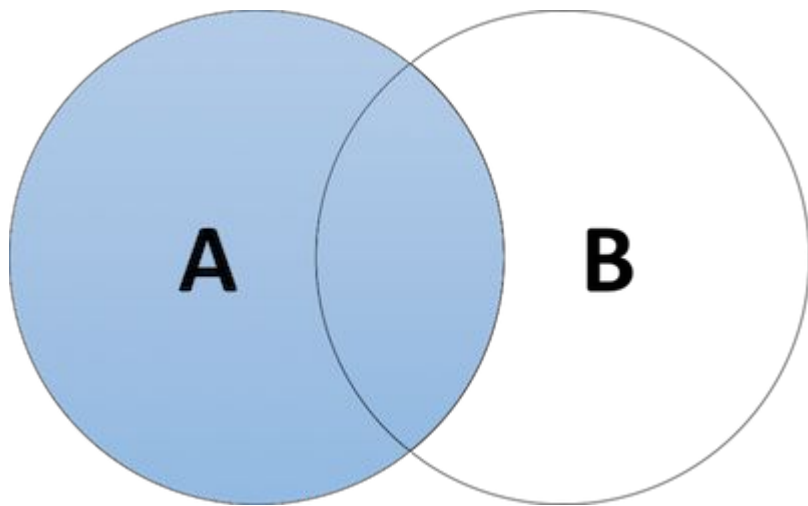
La combinación de LEFT JOIN retorna todas las filas de la tabla izquierda, así no haya coincidencias en la tabla derecha.

Esto significa que si no hay coincidencia para la cláusula ON en la tabla de la derecha, aún así la combinación retornará las filas de la primera tabla en el resultado.

### Sintaxis:

```
SELECT column_name(s)
FROM table1 as A
LEFT JOIN table2 as B
ON B.column_name = A.column_name;
```

La siguiente imagen muestra cómo funciona LEFT JOIN:

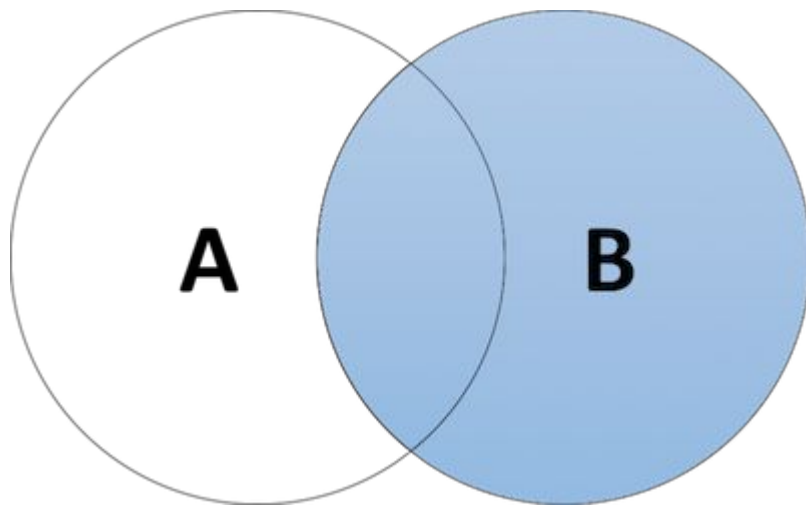


# RIGHT JOIN

La combinación RIGHT JOIN retorna todas las filas de la tabla derecha, así no haya coincidencias en la tabla izquierda.

## Sintaxis:

```
SELECT column_name(s)
FROM table1 as A
RIGHT JOIN table2 as B
ON B.column_name = A.column_name;
```



## Operaciones de agrupación

# UNION

Ocasionalmente, es necesario anexar datos de múltiples tablas de un conjunto de datos exhaustivos. Esto puede ser para tablas con datos similares dentro de la misma base de datos.

Para lograr esto, utiliza los operadores UNION y UNION ALL

**UNION** combina varios conjuntos de datos en un sólo conjunto de datos y remueve cualquier duplicado.

**UNION ALL** combina varios conjuntos de datos en un sólo conjunto de datos, pero no remueve filas duplicadas.

Es necesario que la cantidad de columnas y sus nombres coincidan en ambos. En caso de ser diferente, pero sí componen la misma información podríamos utilizar el alias (as) para “renombrar” esta columna.



## Operaciones de agrupación

# UNION

PedidosYa				ClickDelivery			
Nombre	Apellido	País	Plataforma	Nombre	Apellido	País	Plataforma
Guillermo	Hernández	Uruguay	PedidosYa	Camilo	Caceres	Colombia	ClickDelivery
Lorenzo	Rodriguez	Uruguay	PedidosYa	Andrés	Ramírez	Colombia	ClickDelivery

Ejemplo: Tenemos dos tablas de usuario en dos esquemas diferentes (PedidosYa & ClickDelivery).

Para visualizar los datos:

```
SELECT nombre, apellido, pais, plataforma  
FROM pedidosya.users
```

**UNION**

```
SELECT nombre, apellido, pais, plataforma  
FROM clickdelivery.users
```

PedidosYa UNION ClickDelivery			
Nombre	Apellido	País	Plataforma
Guillermo	Hernández	Uruguay	PedidosYa
Lorenzo	Rodriguez	Uruguay	PedidosYa
Camilo	Caceres	Colombia	ClickDelivery
Andrés	Ramírez	Colombia	ClickDelivery

## Operaciones de agrupación

# GROUP BY

La instrucción GROUP BY se usa a menudo con funciones agregadas (COUNT, MAX, MIN, SUM, AVG) para agrupar el conjunto de resultados por una o más columnas.

Por ejemplo si quisiéramos saber cuantas órdenes confirmadas se realizaron en cada mercado en un período determinado.

La consulta sería la siguiente:

```
SELECT o.country_id, c.country_name, count(o.order_id) as qty
FROM datalake.orders o
INNER JOIN datalake.countries c
ON c.country_id = o.country_id
WHERE o.registered_date >= '2019-01-01'
AND o.registered_date < '2019-02-01'
AND o.state_id = 1
GROUP BY o.country_id, c.country_name
```



o.country_id		c.country_name	qty
1	2	Chile	2
2	3	Argentina	3
3	17	Bolivia	17
4	11	Panamá	11
5	1	Uruguay	1
6	15	Paraguay	15

**Nota:** En esta oportunidad, aproveche a hacer un inner join para obtener el nombre del país.