Clave primaria

Fuente: https://www.tutorialesprogramacionya.com

Una clave primaria es un campo (o varios) que identifica 1 solo registro (fila) en una tabla. Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.
Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.
Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre distinto.
Establecemos que un campo sea clave primaria al momento de creación de la tabla:

```
CREATE TABLE usuarios (
  nombre varchar(20),
  clave varchar(10),
  primary key(nombre)
 );
```

Para definir un campo como clave primaria agregamos "primary key" luego de la definición de todos los campos y entre paréntesis colocamos el nombre del campo que queremos como clave.
Si visualizamos la estructura de la tabla con "describe" vemos que el campo "nombre" es clave primaria y no acepta valores nulos(más adelante explicaremos esto detalladamente).
Ingresamos algunos registros:

```
INSERT INTO usuarios (nombre, clave)  VALUES ('Leonardo',Leo'),
                                             ('MarioPerez','Marito'),
                                             ('Marcelo','River'),
                                             ('Gustavo','River');
```

Si intentamos ingresar un valor para el campo clave que ya existe, aparece un mensaje de error indicando que el registro no se cargó pues el dato clave existe. Esto sucede porque los campos definidos como clave primaria no pueden repetirse.
Ingresamos un registro con un nombre de usuario repetido, por ejemplo:

```
INSERT INTO usuarios (nombre, clave) VALUES ('Gustavo','Boca');
```

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan.
Al establecer una clave primaria estamos indexando la tabla, es decir, creando un índice para dicha tabla; a este tema lo veremos más adelante.

Ejemplo:

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios (
  nombre varchar(20),
  clave varchar(10),
  PRIMARY KEY(nombre)
 );

DESCRIBE usuarios;

INSERT INTO usuarios (nombre, clave)  VALUES   ('Leonardo','Leo'),
                                               ('MarioPerez','Marito'),
                                               ('Marcelo','River'),
                                               ('Gustavo','River'),
                                               ('Gustavo','Boca');
```

Clave foránea

Un campo que se usa para establecer un "JOIN" con otra tabla en la cual es clave primaria, se denomina "clave ajena o foránea".
En el ejemplo de la librería en que utilizamos las tablas "libros" y "editoriales" con los campos:
libros: codigo (clave primaria), titulo, autor, codigo_editorial, precio, cantidad y **editoriales**: codigo (clave primaria), nombre.

el campo "codigo_editorial" de "libros" es una clave foránea, se emplea para enlazar la tabla "libros" con "editoriales" y es clave primaria en "editoriales" con el nombre "codigo".
Cuando alteramos una tabla, debemos tener cuidado con las claves foráneas. Si modificamos el tipo, longitud o atributos de una clave foránea, ésta puede quedar inhabilitada para hacer los enlaces.
Las claves foráneas y las claves primarias deben ser del mismo tipo para poder enlazarse. Si modificamos una, debemos modificar la otra para que los valores se correspondan.

Joins

Los JOINs en SQL sirven para combinar filas de dos o más tablas basándose en un campo común entre ellas, devolviendo por tanto datos de diferentes tablas. Un JOIN se produce cuando dos o más tablas se juntan en una sentencia SQL.
Los más importantes son los siguientes:

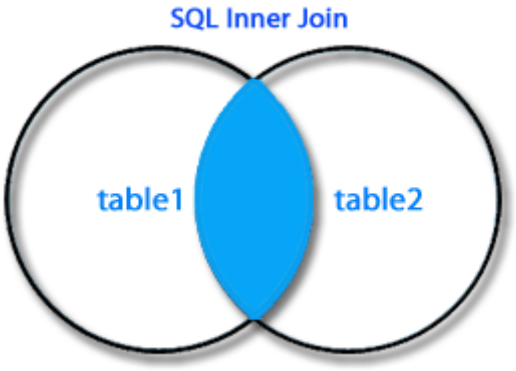
1. **INNER JOIN**: Devuelve todas las filas cuando hay al menos una coincidencia en ambas tablas.
2. **LEFT JOIN:** Devuelve todas las filas de la tabla de la izquierda, y las filas coincidentes de la tabla de la derecha.
3. **RIGHT JOIN**: Devuelve todas las filas de la tabla de la derecha, y las filas coincidentes de la tabla de la izquierda.

4. **OUTER JOIN**: Devuelve todas las filas de las dos tablas, la izquierda y la derecha. También se llama FULL OUTER JOIN.

1. INNER JOIN

INNER JOIN selecciona todas las filas de las dos columnas siempre y cuando haya una coincidencia entre las columnas en ambas tablas. Es el tipo de JOIN más común.
SELECT nombreColumna(s)
FROM tabla1
INNER JOIN tabla2
ON tabla1.nombreColumna=table2.nombreColumna;

Se ve más claro utilizando una imagen:



Vamos a verlo también con un ejemplo, mediante las tablas clientes y pedidos:
clientes:

id_cliente	nombre	contacto
1	Marco Lambert	456443552
2	Lydia Roderic	445332221
3	Ebbe Therese	488982635
4	Sofie Mariona	412436773

pedidos:

id_pedido	id_cliente	factura
233	4	160
234	2	48
235	3	64
236	4	92

Creación de Base de Datos de prueba:

```
CREATE TABLE clientes(
    id_cliente int unsigned AUTO_INCREMENT,
    nombre varchar(40) not null,
    contacto int unsigned not null,
    PRIMARY KEY (id_cliente)
);

CREATE TABLE pedidos(
  id_pedido int unsigned AUTO_INCREMENT,
  id_cliente int unsigned,
  factura int unsigned not null,
  PRIMARY KEY(id_pedido)
);

ALTER TABLE pedidos AUTO_INCREMENT=233;

INSERT INTO clientes (nombre,contacto) VALUES ('Marco Lambert',456443552),
                                              ('Lydia Roderic',445332221),
                                              ('Ebbe Therese',488982635),
                                              ('Sofie Mariona',412436773);

INSERT INTO pedidos (id_cliente,factura) VALUES (4,160),
                                                (2,48),
                                                (3,64),
                                                (4,92),
                                                (10,550);
```

```
SET FOREIGN_KEY_CHECKS = 0;
ALTER TABLE pedidos ADD FOREIGN KEY(id_cliente) REFERENCES clientes(id_cliente);
SET FOREIGN_KEY_CHECKS = 1;
```

La siguiente sentencia SQL devolverá todos los clientes con pedidos:

```
SELECT c.nombre,p.id_pedido
FROM clientes c
INNER JOIN pedidos p
ON c.id_cliente=p.id_cliente;
```

Si hay filas en clientes que no tienen coincidencias en pedidos, los clientes no se mostrarán. La sentencia anterior mostrará el siguiente resultado:

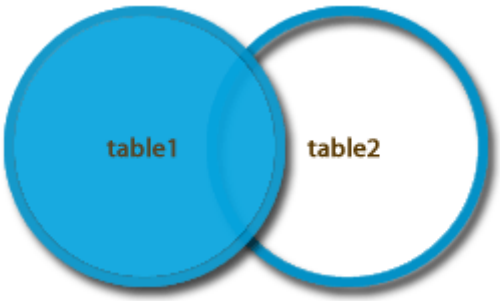
nombre	id_pedido
Ebbe Therese	235
Lydia Roderic	234
Sofie Mariona	233
Sofie Mariona	236

Sofie Mariona aparece dos veces ya que ha realizado dos pedidos. No aparece Marco Lambert, pues no ha realizado ningún pedido.

2. LEFT JOIN

LEFT JOIN mantiene todas las filas de la tabla izquierda (la tabla1). Las filas de la tabla derecha se mostrarán si hay una coincidencia con las de la izquierda. Si existen valores en la tabla izquierda pero no en la tabla derecha, ésta mostrará null.

La representación de **LEFT JOIN** en una imagen es:



Tomando de nuevo las tablas de clientes y pedidos, ahora queremos mostrar todos los clientes, y cualquier pedido que pudieran haber encargado:

```
SELECT c.id_cliente,c.nombre,p.id_pedido
FROM clientes c
LEFT JOIN pedidos p
ON c.id_cliente=p.id_cliente;
```

La sentencia anterior devolverá lo siguiente:

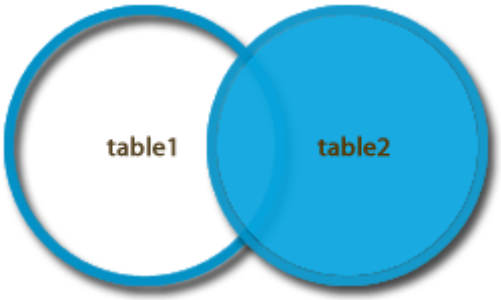
nombre	id_pedido
Ebbe Therese	235
Lydia Roderic	234
Marco Lambert	(null)
Sofie Mariona	233
Sofie Mariona	236

Ahora vemos que se muestran todas las filas de la tabla Clientes, que es la tabla de la izquierda, tantas veces como haya coincidencias con el lado derecho. Marco Lambert no ha realizado ningún pedido, por lo que se muestra null.

3. RIGHT JOIN

Es igual que LEFT JOIN pero al revés. Ahora se mantienen todas las filas de la tabla derecha (tabla2). Las filas de la tabla izquierda se mostrarán si hay una coincidencia con las de la derecha. Si existen valores en la tabla derecha pero no en la tabla izquierda, ésta se mostrará null.

La imagen que representa a RIGHT JOIN es:



De nuevo tomamos el ejemplo de Clientes y Pedidos, y vamos a hacer el mismo ejemplo anterior, pero cambiando LEFT por RIGHT:

```
SELECT p.id_pedido, c.nombre
FROM clientes c RIGHT JOIN pedidos p
```

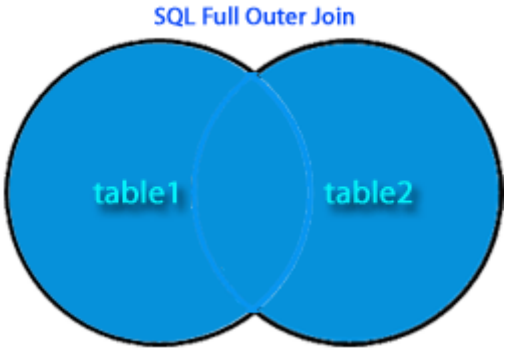
ON c.id_cliente=p.id_cliente;

Ahora van a aparecer todos los pedidos, y los nombres de los clientes que han realizado un pedido. Nótese que también se ha cambiado el orden, y se han ordenado los datos por id_pedido.

PedidoID	NombreCliente
233	Sofie Mariona
234	Lydia Roderic
235	Ebbe Therese
236	Sofie Mariona

4. OUTER JOIN

OUTER JOIN o FULL OUTER JOIN devuelve todas las filas de la tabla izquierda (tabla1) y de la tabla derecha (tabla2). Combina el resultado de los joins LEFT y RIGHT. Aparecerá null en cada una de las tablas alternativamente cuando no haya una coincidencia.
La imagen que representa el OUTER JOIN es la siguiente:



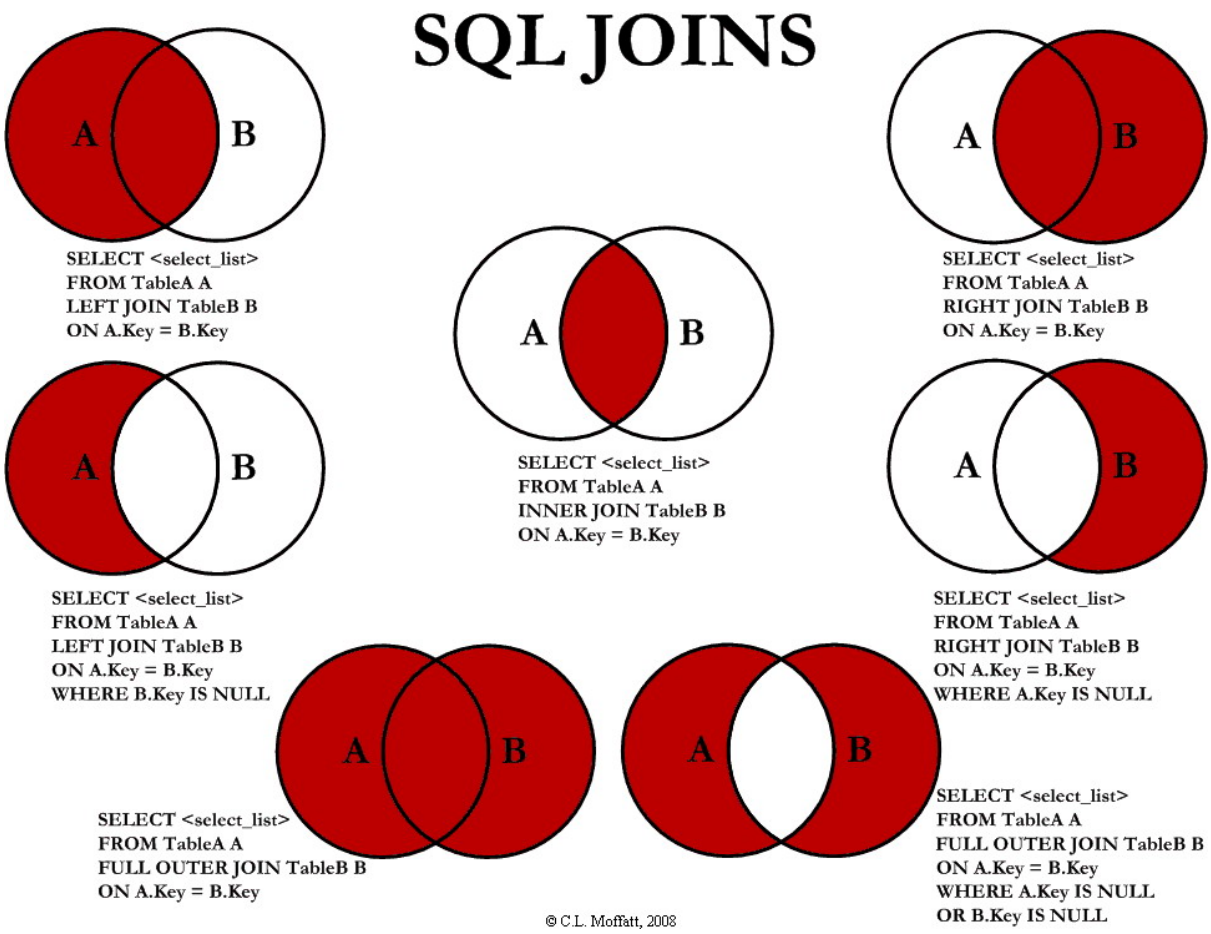
Vamos a obtener todas las filas de las tablas clientes y pedidos:
La sentencia devolverá todos los Clientes y todos los Pedidos, si un cliente no tiene pedidos mostrará null en id_pedido, y si un pedido no tuviera un cliente mostraría null en nombre (en este ejemplo no sería lógico que un Pedido no tuviera un cliente).

La sintaxis de OUTER JOIN o FULL OUTER JOIN no existen en MySQL, pero se puede conseguir el mismo resultado de diferentes formas, esta es una:
SELECT c.id_cliente,p.id_pedido
FROM clientes c
LEFT JOIN pedidos p

ON c.id_cliente=p.id_cliente

UNION

SELECT c.id_cliente,p.id_pedido
FROM clientes c
RIGHT JOIN pedidos p
ON c.id_cliente=p.id_cliente;



Alter table para agregar un campo a una tabla

Para modificar la estructura de una tabla existente, usamos "alter table".
"alter table" se usa para:
- agregar nuevos campos,
- eliminar campos existentes,
- modificar el tipo de dato de un campo,
- agregar o quitar modificadores como "null", "unsigned", "auto_increment",
- cambiar el nombre de un campo,

- agregar o eliminar la clave primaria,
- agregar y eliminar índices,
- renombrar una tabla.

"alter table" hace una copia temporal de la tabla original, realiza los cambios en la copia, luego borra la tabla original y renombra la copia.
Aprenderemos a agregar campos a una tabla.
Para ello utilizamos nuestra tabla "libros", definida con la siguiente estructura:
- código, int unsigned auto_increment, clave primaria,
- titulo, varchar(40) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned.

Necesitamos agregar el campo "cantidad", de tipo smallint unsigned not null, tipeamos:

ALTER TABLE libros ADD cantidad smallint unsigned not null;

Usamos "alter table" seguido del nombre de la tabla y "add" seguido del nombre del nuevo campo con su tipo y los modificadores.
Agreguemos otro campo a la tabla:

ALTER TABLE libros ADD edicion date;

Si intentamos agregar un campo con un nombre existente, aparece un mensaje de error indicando que el campo ya existe y la sentencia no se ejecuta.
Cuando se agrega un campo, si no especificamos, lo coloca al final, después de todos los campos existentes; podemos indicar su posición (luego de qué campo debe aparecer) con "after":

ALTER TABLE libros ADD cantidad tinyint unsigned AFTER autor;

Alter table para eliminar un campo de una tabla

"alter table" nos permite alterar la estructura de la tabla, podemos usarla para eliminar un campo.
Continuamos con nuestra tabla "libros".
Para eliminar el campo "edicion" tipeamos:

ALTER TABLE libros DROP edicion;

Entonces, para borrar un campo de una tabla usamos "alter table" junto con "drop" y el nombre del campo a eliminar.
Si intentamos borrar un campo inexistente aparece un mensaje de error y la acción no se realiza.
Podemos eliminar 2 campos en una misma sentencia:

ALTER TABLE libros DROP editorial, DROP cantidad;

Si se borra un campo de una tabla que es parte de un índice, también se borra el índice.
Si una tabla tiene sólo un campo, este no puede ser borrado.
Hay que tener cuidado al eliminar un campo, este puede ser clave primaria. Es posible eliminar un campo que es clave primaria, no aparece ningún mensaje:

ALTER TABLE libros DROP codigo;

Si eliminamos un campo clave, la clave también se elimina.

Alter table para modificar un campo de una tabla

Con "alter table" podemos modificar el tipo de algún campo incluidos sus atributos.
Continuamos con nuestra tabla "libros", definida con la siguiente estructura:
- código, int unsigned,
- titulo, varchar(30) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned,
- cantidad int unsigned.

Queremos modificar el tipo del campo "cantidad", como guardaremos valores que no superarán los 50000 usaremos smallint unsigned, tipeamos:

ALTER TABLE libros MODIFY cantidad smallint unsigned;

Usamos "alter table" seguido del nombre de la tabla y "modify" seguido del nombre del nuevo campo con su tipo y los modificadores.
Queremos modificar el tipo del campo "titulo" para poder almacenar una longitud de 40 caracteres y que no permita valores nulos, tipeamos:

ALTER TABLE libros MODIFY titulo varchar(40) not null;

Hay que tener cuidado al alterar los tipos de los campos de una tabla que ya tiene registros cargados. Si tenemos un campo de texto de longitud 50 y lo cambiamos a 30 de longitud, los registros cargados en ese campo que superen los 30 caracteres, se cortarán (en versiones nuevas de MySQL 8.x genera un error y no modifica la estructura de la tabla)
Igualmente, si un campo fue definido permitiendo valores nulos, se cargaron registros con valores nulos y luego se lo define "not null", todos los registros con valor nulo para ese campo cambiarán al valor por defecto según el tipo (cadena vacía para tipo texto y 0 para numéricos), ya que "null" se convierte en un valor inválido.
Si definimos un campo de tipo decimal(5,2) y tenemos un registro con el valor "900.00" y luego modificamos el campo a "decimal(4,2)", el valor "900.00" se convierte en un valor inválido para el tipo, entonces guarda en su lugar, el valor límite más cercano, "99.99" (en versiones nuevas de MySQL genera un error y no modifica la estructura de la tabla).

Si intentamos definir "auto_increment" un campo que no es clave primaria, aparece un mensaje de error indicando que el campo debe ser clave primaria. Por ejemplo:

ALTER TABLE libros MODIFY codigo int unsigned auto_increment;

Subconsultas

El uso de subconsultas es una técnica que permite utilizar el resultado de una tabla **SELECT** en otra consulta **SELECT**. Permite solucionar consultas complejas mediante el uso de resultados previos conseguidos a través de otra consulta.
El **SELECT** que se coloca en el interior de otro **SELECT** se conoce con el término de **SUBSELECT**. Ese **SUBSELECT** se puede colocar dentro de las cláusulas **WHERE**, **HAVING**, **FROM** o **JOIN**.

Subconsultas simples

Las subconsultas simples son aquellas que devuelven una única fila. Si además devuelven una única columna, se las llama subconsultas escalares, ya que devuelven un único valor.
La sintaxis es:

```
SELECT listaExpresiones
FROM tabla
WHERE expresión OPERADOR
                        (SELECT listaExpresiones
                        FROM tabla);
```

El operador puede ser >,<,>=,<=,!=, = o IN.
Ejemplo:

```
SELECT nombre_empleado, paga
FROM empleados
WHERE paga <
                (SELECT paga FROM empleados
                WHERE nombre_empleado='Martina');
```

Esta consulta muestra el nombre y paga de los empleados cuya paga es menor que la de la empleada Martina. Para que funcione esta consulta, la subconsulta sólo puede devolver un valor (solo puede haber una empleada que se llame Martina.

Se pueden usar subconsultas las veces que haga falta:

```
SELECT nombre_empleado, paga
FROM empleados
WHERE paga <
                (SELECT paga FROM empleados
                WHERE nombre_empleado='Martina')
AND paga >
                (SELECT paga FROM empleado
                WHERE nombre_empleado='Luis');
```

En realidad lo primero que hace la base de datos es calcular el resultado de la subconsulta:

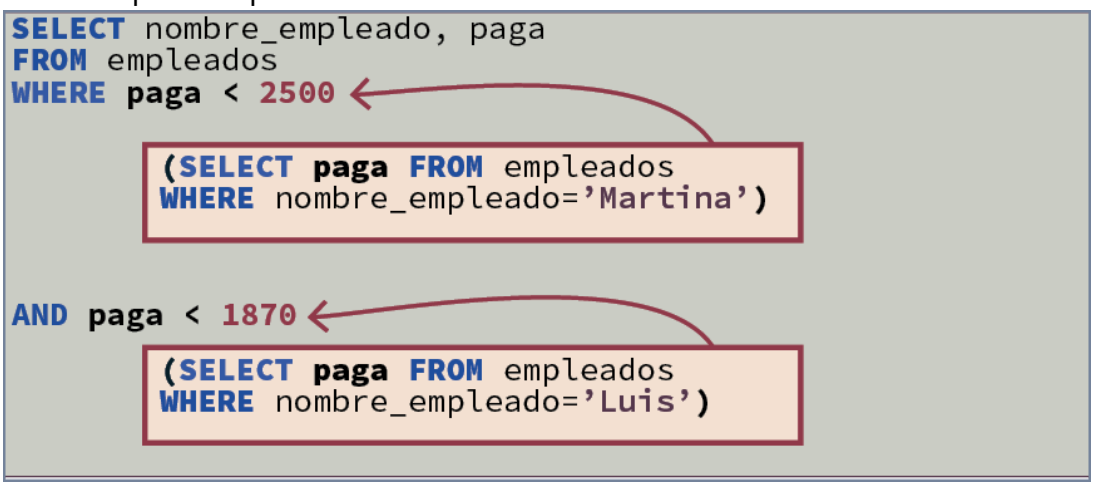


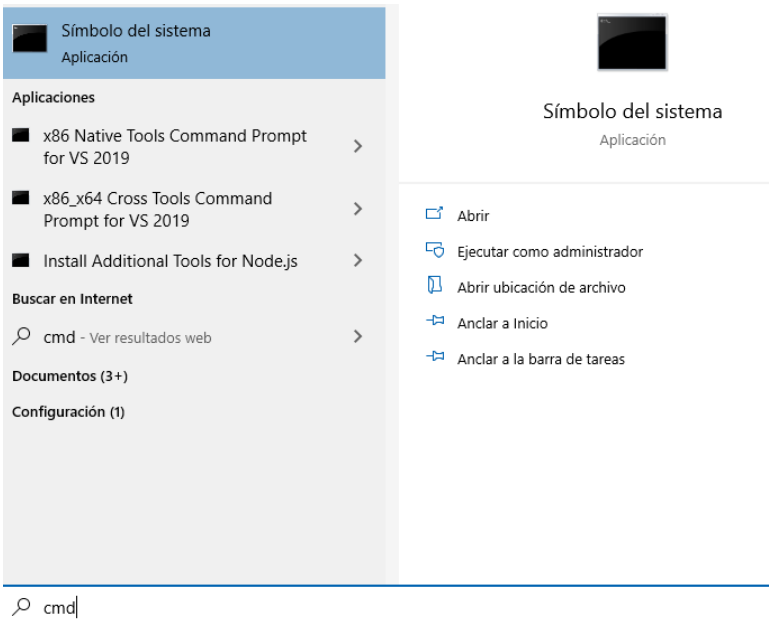
Ilustración 26. Ejemplo de funcionamiento de subconsultas

La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luís (1870 euros) y lo que gana Martina (2500) .
Las subconsultas siempre se deben encerrar entre paréntesis y se deberían (aunque no es obligatorio, sí altamente recomendable) colocar a la derecha del operador relacional.
Una subconsulta que utilice los valores >,<,>=,... tiene que devolver un único valor, de otro modo ocurre un error.

Además tienen que devolver el mismo tipo y número de datos para relacionar la subconsulta con la consulta que la utiliza (no puede ocurrir que la subconsulta tenga dos columnas y ese resultado se compara usando una sola columna en la consulta general).

Backup con mysqldump

1) Acceder a la ubicación del archivo mysqldump.exe con cmd (en modo administrador)



Si no se cambió la ruta de instalación, escribir:

cd C:\Program Files\MySQL\MySQL Server 8.0\bin

```
C:\WINDOWS\system32>cd C:\Program Files\MySQL\MySQL Server 8.0\bin
```

2) Hacer el backup

Escribir mysqldump -u *nombre de usuario* -p *nombreBD* > back.sql
Pedirá el password del usuario root.

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -u root -p ejemplo > back.sql
Enter password: ****
```

3) Restaurar base de datos

Escribir mysql -u *nombre de usuario* -p *nombreBD* < back.sql
Pedirá el password del usuario root. Debe estar creada la base de datos *nombreBD* en VS Code, que estará en primera instancia vacía hasta hacer el restore.

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p ejemplo < back.sql
Enter password: ****
```