UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL INTERMEDIO

www.sceu.frba.utn.edu.ar/e-learning

Módulo 2

Mas coceptos sobre Mysql

Claves foráneas y algunas consultas mas complejas.

Presentación de la Unidad:

En esta unidad veremos algunos conceptos mas en lo que hace a las tablas y las consultas de una base de datos.

Vamos a ver que son las claves foráneas y como armar algunas consultas mas complejas tomando datos de mas de una tabla a la vez o utilizando subconsultas.

Objetivos:

- Aprender a definir claves foráneas en las tablas de la base de datos.
- Realizar consultas mas complejas combinando datos de mas de una tabla o utilizando subconsultas.

Temario:

Claves foráneas

Joins

Subconsultas

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

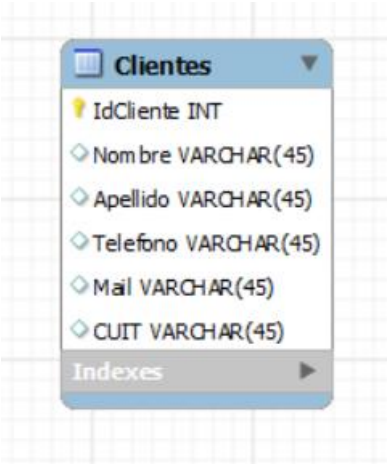
Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

CLAVES FORÁNEAS Y OTRAS CONSULTAS.

Claves foráneas

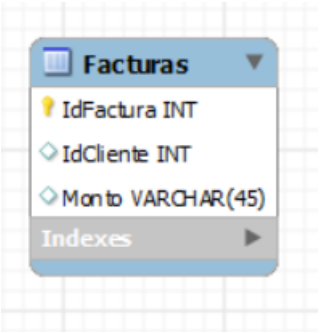
La clave primaria de una tabla SQL es la que me sirve para identificar de manera unívoca un registro. Existen otro tipo de claves, denominadas claves foráneas, en inglés, foreign keys, que hacen referencia a una clave de otra tabla.

Pongamos un ejemplo supongamos que tenemos una tabla de Clientes con los siguientes datos:



El campo IdCliente es la PRIMARY KEY de la tabla Clientes.

Y por otro lado tenemos una tabla de Facturas donde tenemos almacenadas todas las facturas de las compras realizadas por los clientes con los siguientes datos:



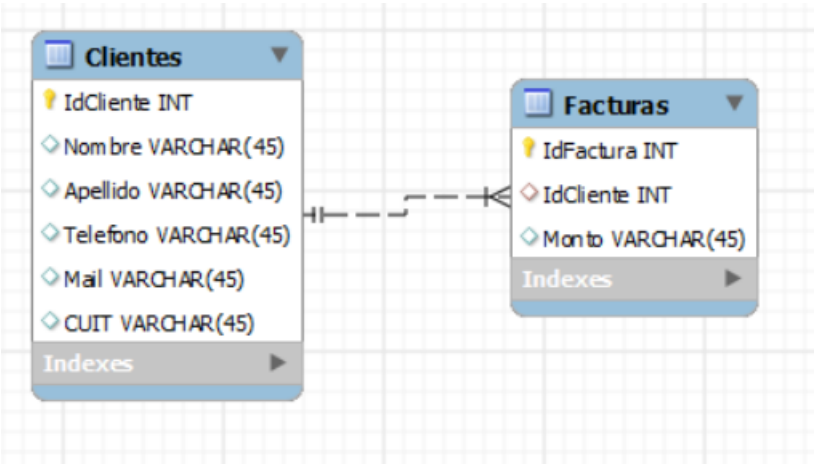
El campo IdFactura es la PRIMARY KEY de la tabla Facturas.

Ahora bien, el dato IdCliente está haciendo referencia al campo IdCliente de la tabla de Clientes.

Y por otro lado no sería correcto que una factura en la tabla de Facturas tenga un IdCliente asociado que no exista en la tabla de Clientes.

Nosotros tenemos la posibilidad, en este caso, de indicarle a Mysql estas características y de indicarle como debe comportarse en cada situación.

Las dos tablas las podemos graficar indicando una relación de 1 a muchos entre la tabla de Clientes y la tabla de Facturas.



Hagamos un repaso de los tipos de realciones que podemos establecer entre tablas de nuestra base de datos para entendernos mejor.

Entendiendo las relaciones en mysql

Qué mejor que ver todos los casos posibles de relaciones que se nos pueden dar para terminar de entenderlos de forma sencilla.

Relaciones de uno a uno

Estas relaciones existen por ejemplo en el caso de una persona y su dni, una persona sólo puede tener un dni, y un dni sólo puede pertenecer a una persona.

Para llevar a cabo esta relación en nuestra base de datos simplemente debemos crear nuestra tabla usuarios y nuestra tabla dnis, para hacer referencia al dni de cada usuario nos basta con crear un campo en la tabla dnis el cuál actuará como clave foránea haciendo referencia al usuario a través de su id.

Relaciones de uno a muchos

El ejemplo perfecto para estas relaciones es entre usuarios y posts de un blog, un usuario puede tener muchos posts, pero un post sólo puede pertenecer a un usuario, sirve lo mismo que en la relación de uno a uno. La única diferencia entre estas dos relaciones en este aspecto, es que la **clave foránea** entre usuarios y dnis puede estar tanto en la tabla usuarios con un campo id_dni como en la tabla dnis con un campo id_usuario.

En cambio, en una relación de uno a muchos la clave foránea siempre debe estar en la tabla que hace la relación de muchos, en este caso sería la tabla posts.

Relaciones de muchos a muchos

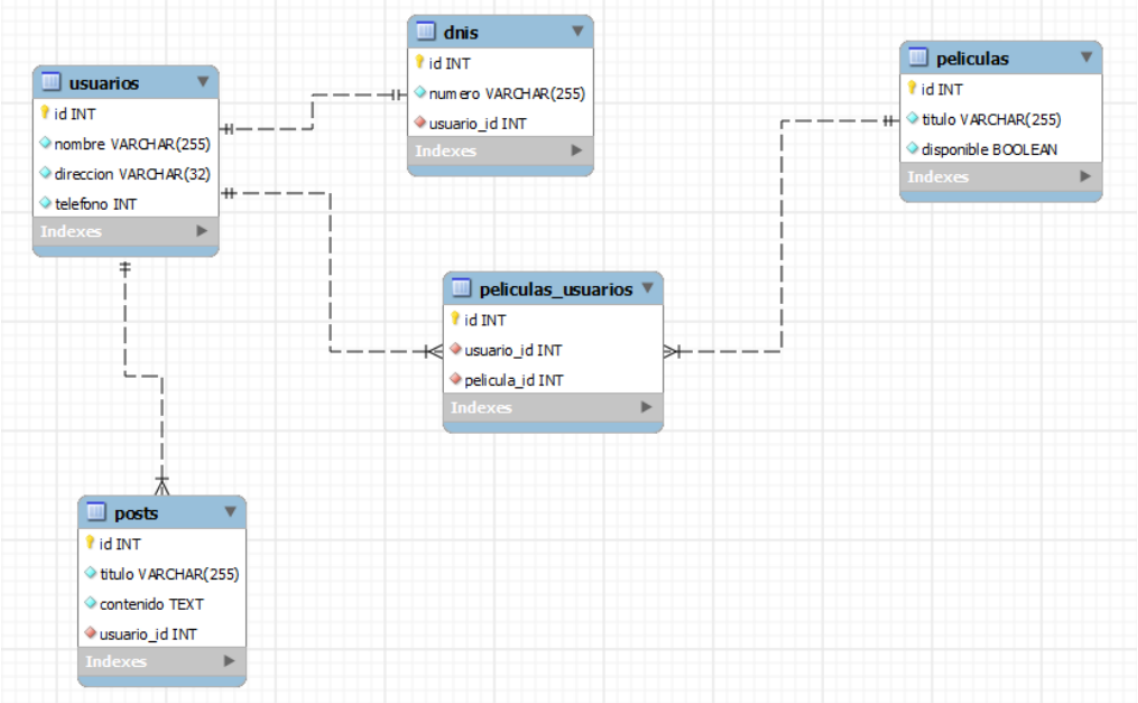
Este tipo de **relaciones** vienen a ser las más complicadas, aunque realmente no lo son, para el ejemplo podemos decir que la relación entre usuarios y peliculas(alquileres de un videoclub), un usuario puede alquilar muchas películas, y una película puede ser alquilada por muchos usuarios.

Estas relaciones no pueden ser llevadas a cabo con simples claves foráneas ya que necesitaríamos una por cada registro, cosa completamente inviable. **Para este tipo de relaciones debemos crear una tercera tabla**, conocida como **tabla pívote**, que por convención su nombre suele ser usuarios_peliculas para nuestro caso, es decir, los nombres de las dos tablas separados por guiones.

Estas tablas deben contener como mínimo dos campos, usuario_id y pelicula_id que harán referencia a las claves primarias de sus respectivas tablas.

La función de esta tabla es la de poder enlazar a los usuarios y las películas a través de sus claves primarias, es decir, si tenemos un usuario con id 1 y una película con id 120 en sus respectivas tablas, para poder unirlos, deberíamos crear un nuevo registro en la tabla usuarios_peliculas con esos datos.

Para tenerlo más claro, veamos un sencillo diagrama sobre las relaciones de las que hemos hablado y como debemos interpretarlas en nuestra base de datos.



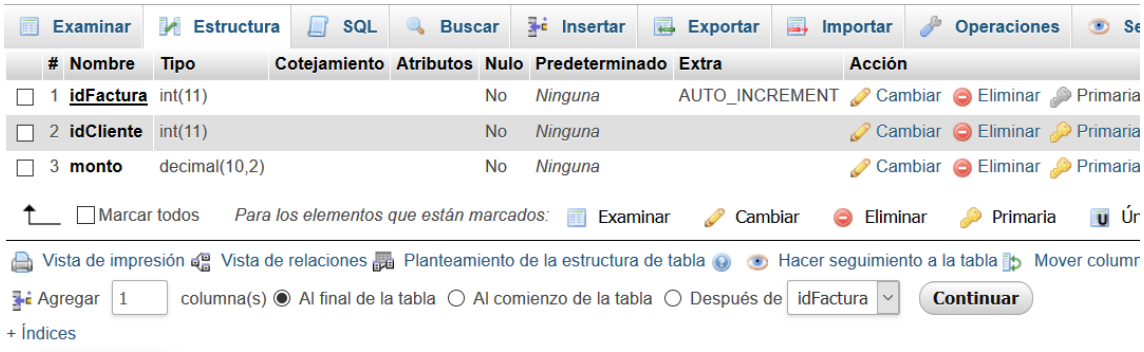
Como definir las claves foráneas

Continuemos con el ejemplo que habíamos planteado al principio, nuestras tablas de Clientes y de Facturas. Vamos a ver como hacemos para representar esto en MySQL, en la práctica, cuando definimos las tablas.

En primer lugar definimos las tablas como estamos acostumbrados, cada una con su clave primaria.

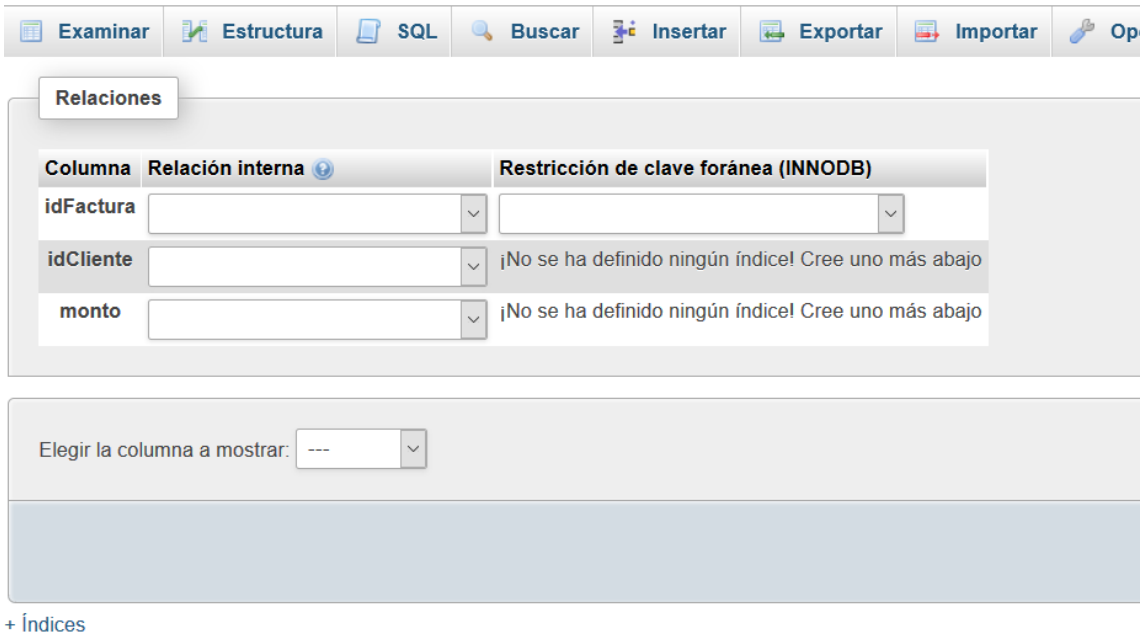
Una vez que tenemos la tabla de clientes y de facturas definidas vamos a seleccionar la tabla de facturas y vamos a ir a la solapa estructura.

Allí vamos a ver la definición de la tabla y debajo hay un vínculo que se llama Vista de relaciones.

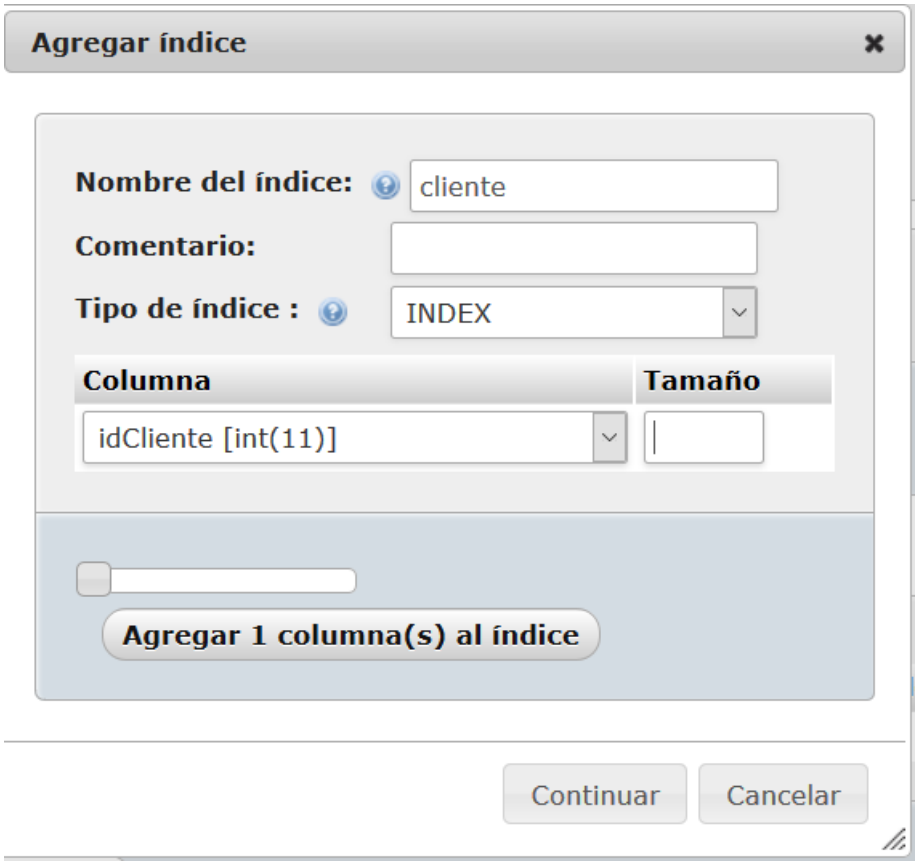


Si presionamos en ese vínculo vamos a ir a una pantalla en la cual podemos definir las restricciones de clave foránea. Lo que queremos hacer en este caso es decirle a Mysql que el campo idCliente es clave foránea de la tabla de clientes haciendo referencia al campo idCliente en dicha tabla.

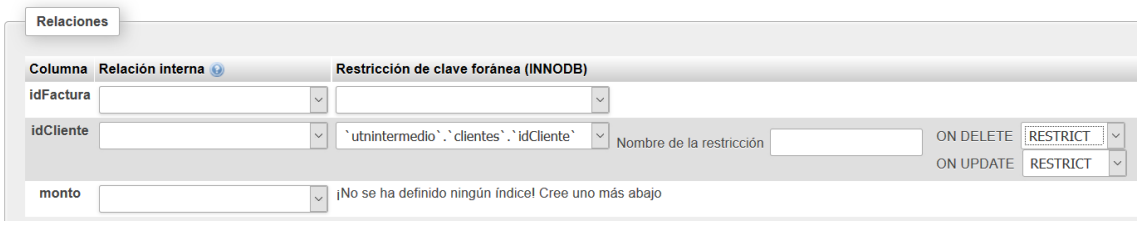
Para poder definir una clave foránea, ésta debe ser clave en la tabla en la que se encuentra, es decir que el dato idCliente en la tabla de Facturas debería ser clave para poder definirlo como clave foránea de otra tabla. Y en este caso no lo es. Es por eso que no me aparece habilitada la posibilidad de definirla como tal y me aparece la leyenda ¡No se ha definido ningún índice! Cree uno más abajo.



Para crear un índice presionamos en el vínculo que se ve debajo y lo creamos. En este caso vamos a definir un índice con el nombre cliente y del tipo INDEX en la colomna Facturas.idCliente



Ahora si nos va a aparecer habilitada la opción para seleccionar a que columna hace referencia la columna idCliente de la tabla de facturas. Nosotros le vamos a decir que hace referencia a la columna idCliente de la tabla de Clientes y automáticamente se habilitan dos opciones para indicarle como se tienen que comportar las tablas ante un DELETE o un UPDATE.



Las restricciones foreign key tienen cláusulas ON DELETE y ON UPDATE que le indican a Mysql como se debe comportar frente a las eliminaciones y modificaciones de las tablas referenciadas en la restricción.

Las opciones posibles son:

RESTRICT
SET NULL
NO ACTION
CASCADE

Veamos que significa cada una de ellas:

CASCADE: Borra o actualiza el registro en la tabla padre y automáticamente borra o actualiza los registros coincidentes en la tabla hija. Tanto ON DELETE CASCADE como ON UPDATE CASCADE están disponibles en MySQL 5.0. Entre dos tablas, no se deberían definir varias cláusulas ON UPDATE CASCADE que actúen en la misma columna en la tabla padre o hija.

SET NULL: Borra o actualiza el registro en la tabla padre y establece en NULL la o las columnas de clave foránea en la tabla hija. Esto solamente es válido si las columnas de clave foránea no han sido definidas como NOT NULL. MySQL 5.0 soporta tanto ON DELETE SET NULL como ON UPDATE SET NULL.

NO ACTION: significa *ninguna acción* en el sentido de que un intento de borrar o actualizar un valor de clave primaria no sera permitido si en la tabla referenciada hay una valor de clave foránea relacionado. En MySQL 5.0, InnoDB rechaza la operación de eliminación o actualización en la tabla padre.

RESTRICT: Rechaza la operación de eliminación o actualización en la tabla padre.

NO ACTION y RESTRICT son similares en tanto omiten la cláusula ON DELETE u ON UPDATE. (Algunos sistemas de bases de datos tienen verificaciones diferidas o retrasadas, una de las cuales es NO ACTION. En MySQL, las restricciones de claves foráneas se verifican inmediatamente, por eso, NO ACTION y RESTRICT son equivalentes.)

Una cosa a tener en cuenta es que se puede definir una restricción en para el delete y otra para el update, es decir puedo definir una restricción de clave foránea con la clausula ON DELETE NO ACTION y ON UPDATE CASCADE. Si bien no es lo mas habitual está permitido.

Pueden hacer la prueba de ir definiendo las restricciones con distintos valores e ir corroborando el resultado esperado.

Por ejemplo pueden definir el cliente 5 con 3 facturas asociadas y teniendo la clausula ON DELETE CASCADE, al intentar borrar el cliente 5 de la tabla de clientes debería borrarle las tres filas de la tabla de facturas asociadas a ese cliente.

En cambio, la misma operación, ante una clausula ON DELETE NO ACTION no tendría que permitirles el delete sobre la tabla de clientes porque ese cliente tiene facturas asociadas.

Join

El Join es utilizado para listar datos que pertenecen a dos tablas diferentes. En nuestro ejemplo, supongamos que yo quisiera listar todas las facturas con los datos de los clientes. Si tomo los datos de la tabla de facturas, allí tengo el dato del idCliente, pero en mi listado quiero mostrar el Nombre y Apellido del cliente con el idFactura y el monto. En ese caso tendría que tomar los datos de ambas tablas. Y hay un dato que es el que actúa como nexo entre ambas, el idCliente.

Los 3 tipos de Join mas comunes son:

- **INNER JOIN**:

  Devuelve todas las filas que con una coincidencia en ambas tablas. Si hay un dato que está en una de las tablas pero no en la otra esa fila no es listada.

  En nuestro ejemplo, tengo los clientes con idCliente es 1, 2, 3, 4 y 5 en la tabla de Clientes y tengo facturas en la tabla de Facturas asociadas a los clientes 1, 2 y 3, para los clientes 4 y 5 no se listará ninguna fila.

- **LEFT JOIN**:

  El **LEFT JOIN** se diferencia del INNER JOIN en que si los datos de la tabla de la izquierda no se encuentran presentes en la tabla de la derecha se generará una fila con los valores correspondientes a la tabla de la derecha en NULL y los datos correspondientes a la tabla de la izquierda.

- **RIGHT JOIN**:

  El **RIGHT JOIN** es igual al LEFT JOIN pero en el sentido inverso. Si los datos de la tabla de la derecha no se encuentran presentes en la tabla de la izquierda se generará una fila con los valores correspondientes a la tabla de la izquierda en NULL y los datos correspondientes a la tabla de la derecha.

Vamos a cargar algunos datos en las tablas para poder ejecutar las consultas y comprobar los resultados devueltos.

La tabla de Clientes tendrá los siguientes datos:

idCliente	Nombre	Apellido	Mail	Telefono	CUIT
1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236
3	Esteban	Rodriguez	estebanrod@yahoo.com.ar	49785532	20123569987
4	Estela	Mendoza	estela.mendoza@gmail.com	43655522	27201554887

Y la tabla de Facturas:

idFactura	idCliente	monto
1	1	1550.50
2	1	1800.00
3	1	2500.00
4	2	1890.99
5	2	1750.00
6	3	1954.70

Es decir que el cliente con id 1 tendrá las facturas 1, 2 y 3 asociadas; el cliente 2 las facturas 4 y 5 y el cliente 3 la factura 6. El cliente 4 no tiene facturas asociadas.

Veamos entonces que sucede con las consultas de acuerdo a lo explicado anteriormente.

Inner Join

Si ejecutamos la consulta:

```
SELECT *
FROM facturas
INNER JOIN clientes ON clientes.idCliente = facturas.idCliente
```

Los datos que recuperamos son los siguientes:

idFactura	idCliente	monto	idCliente	Nombre	Apellido	Mail	Telefono	CUIT
1	1	1550.50	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
2	1	1800.00	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
3	1	2500.00	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
4	2	1890.99	2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236
5	2	1750.00	2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236
6	3	1954.70	3	Esteban	Rodriguez	estebanrod@yahoo.com.ar	49785532	20123569987

Como pueden observar para el cliente 4 que no tiene facturas asociadas no se genera ninguna fila, puesto que para el id 4 tenemos una fila en la tabla de clientes pero no en la tabla de facturas.

Left Join

Si ejecutamos la consulta:

```
SELECT *
FROM clientes
LEFT JOIN facturas ON clientes.idCliente = facturas.idCliente
```

Los datos que recuperamos son los siguientes:

idCliente	Nombre	Apellido	Mail	Telefono	CUIT	idFactura	idCliente	monto
1	Juan	Perez	juanperez@gmail.com	45226565	20194748364	1	1	1550.50
1	Juan	Perez	juanperez@gmail.com	45226565	20194748364	2	1	1800.00
1	Juan	Perez	juanperez@gmail.com	45226565	20194748364	3	1	2500.00
2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236	4	2	1890.99
2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236	5	2	1750.00
3	Esteban	Rodriguez	estebanrod@yahoo.com.ar	49785532	20123569987	6	3	1954.70
4	Estela	Mendoza	estela.mendoza@gmail.com	43655522	27201554887	*NULL*	*NULL*	*NULL*

Como ven se genera una fila para el cliente cuyo id es 4 que posee una fila en la tabla de Clientes pero tiene ninguna fila en la tabla de facturas.

Right Join

Si ejecutamos la consulta:

```
SELECT *
FROM facturas
RIGHT JOIN clientes ON clientes.idCliente = facturas.idCliente
```

Los datos que recuperamos son los siguientes:

idFactura	idCliente	monto	idCliente	Nombre	Apellido	Mail	Telefono	CUIT
1	1	1550.50	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
2	1	1800.00	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
3	1	2500.00	1	Juan	Perez	juanperez@gmail.com	45226565	20194748364
4	2	1890.99	2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236
5	2	1750.00	2	Susana	Garcia	susanagarcia@hotmail.com	45551899	27227981236
6	3	1954.70	3	Esteban	Rodriguez	estebanrod@yahoo.com.ar	49785532	20123569987
NULL	*NULL*	*NULL*	4	Estela	Mendoza	estela.mendoza@gmail.com	43655522	27201554887

Como ven se genera una fila para el cliente cuyo id es 4 que posee una fila en la tabla de Clientes pero tiene ninguna fila en la tabla de facturas.

Como habrán notado para lograr el mismo resultado invertimos el orden de las tablas.

Subconsultas

Una subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT que llamaremos consulta principal.

Se puede encontrar en la lista de selección, en la cláusula WHERE o en la cláusula HAVING de la consulta principal.

Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis, no puede contener la cláusula ORDER BY, ni puede ser la UNION de varias sentencias SELECT.

Se suelen utilizar, entre otras cosas, junto a las funciones agregadas que vimos anteriormente MIN, MAX, SUM, AVG, etc.

He aquí un ejemplo de una comparación común de subconsultas que no puede hacerse mediante un join. Encuentra todos los valores en la tabla t1 que son iguales a un valor máximo en la tabla t2:

```
SELECT column1 FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Traslademos esto al ejemplo con el que veníamos trabajando. Para eso vamos a modificar el monto de la factura con id 5 que corresponde al cliente 2 con el valor 2500. Entonces los datos de nuestra tabla serán estos:

idFactura	idCliente	monto
1	1	1550.50
2	1	1800.00
3	1	2500.00
4	2	1890.99
5	2	2500.00
6	3	1954.70

Si ejecutamos la siguiente sentencia:

```
SELECT *
FROM facturas
WHERE monto = ( SELECT MAX( monto ) FROM facturas )
```

Y el resultado obtenido es el siguiente:

idFactura	idCliente	monto
3	1	2500.00
5	2	2500.00

Las subconsultas también pueden utilizarse con las palabras ANY, IN y ALL.

Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

Pensando en nuestro ejemplo podriamos listar todos los clientes que tengan facturas con la siguiente sentencia:

```
SELECT *
FROM clientes
WHERE clientes.idCliente
IN (SELECT idCliente FROM facturas)
```

Eso nos dará como resultado los clientes 1, 2 y 3.

En cambio si ejecutamos la siguiente sentencia:

```
SELECT *
FROM clientes
```

```
WHERE clientes.idCliente
NOT IN (SELECT idCliente FROM facturas)
```

Obtendremos como resultado el cliente 4.

Para profundizar mas en el tema de las subconsultas les sugiero que recurran al manual de MYsql: http://dev.mysql.com/doc/refman/5.7/en/subqueries.html