Funcionamiento de *HeidiSQL*

Eduardo Espuch

Universitat d'Alacant
Universidad de Alicante

Resumen

Explicacion del funcionamiento de cada comando visto justo con ejemplos complejos.

Índice

1. Conceptos teóricos

Este apartado reunirá los conceptos teóricos que se puedan necesitar para realizar algunos ejercicios:

1.1. Términos de importancia

Los **valores nulos** (VNN) representan un atributo que puede ser desconocido, pero no por ello se indica con un 0, se indica con el valor NULL (vació).

Se llama **clave candidata** atributos que diferencias unas tuplas de otras, no admite nulos ni duplicados, es decir, si tenemos una clave candidata debe de existir y ser única, forzando a la tabla a comportarse como una relación. Este atributo o conjunto de atributos únicos para una entidad es decisión del administrador, no tiene porque coincidir con la realidad. Podemos distinguir dos tipos de claves candidatas:

PK **Clave primaria**: Clave primaria: se de

ne siempre, actúa como identificador principal, es el atributo que diferencia cada entidad. Siempre existe.

UK+VNN **Clave alternativa**: se dan aveces, cuando se dan varias CC y la alternativa no coincide con la primaria. Se puede utilizar para mayor restricción en la correspondencia entre clases (mas atributos identificadores con los que comparar) o por su mera existencia.

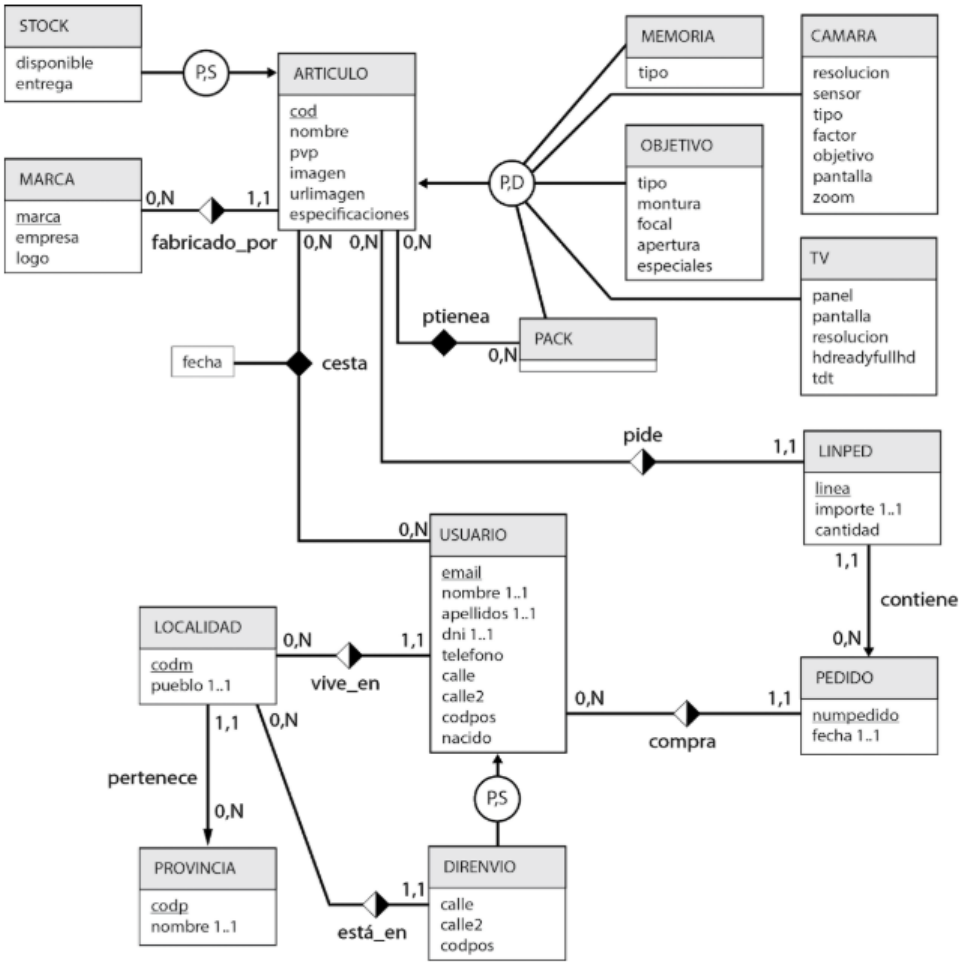
Una **clave foránea** o **clave ajena** (o Foreign Key FK) es el conjunto de atributos que hace referencia a la clave primaria de otra relacion por la integridad referencial (o a ella misma), esta si admite nulos y duplicados. Es decir, o hace referencia a un valor nulo o hace referencia a un valor almacenado en alguna clave primaria y lo puede hacer varias veces. Es posible forzar a que no admita nulos y/o duplicados, siendo eleccion del administrador implementar esta condicion.

1.2. Tipos de relaciones por cardinalidad

1. 1:1
2. 1:1+RE
3. 1:N
4. 1:N+RE
5. N:N
6. N:N+RE

1.3. Tablas de relacion

Tienda Online



```
provincia (
codp varchar(2),
nombre varchar(25) )
CP (codp)

localidad (
codm varchar(4),
pueblo varchar(50),
provincia varchar(2))
CP (codm,provincia)
CAj (provincia) -> provincia
VNN (pueblo)

usuario (
email varchar(50),
nombre varchar(35),
apellidos varchar(55),
dni varchar(12),
telefono varchar(15),
calle varchar(45),
calle2 varchar(45),
codpos varchar(5),
pueblo varchar(4),
provincia varchar(2),
nacido date)
CP (email)
CAlt (dni)
CAj (pueblo, provincia) -> localidad VNN
VNN (apellidos)
VNN (nombre)

direnvio (
email varchar(50),
calle varchar(45),
calle2 varchar(45),
codpos varchar(5),
pueblo varchar(4),
provincia varchar(2))
CP (email)
CAj (pueblo, provincia) -> localidad VNN
CAj (email) -> usuario
```

```
marca (
marca varchar(15),
empresa varchar(60),
logo blob )
CP (marca)

articulo (
cod varchar(7),
nombre varchar(45),
pvp decimal(7,2),
marca varchar(15),
imagen blob,
urlimagen varchar(100),
especificaciones text)
CP (cod)
CAj (marca) -> marca

camara (
cod varchar(7),
resolucion varchar(15),
sensor varchar(45),
tipo varchar(45),
factor varchar(10),
objetivo varchar(15),
pantalla varchar(20),
zoom varchar(40))
CP (cod)
CAj (cod) -> articulo

tv (
cod varchar(7),
panel varchar(45),
pantalla smallint(6),
resolucion varchar(15),
hdreadyfullhd varchar(6),
tdt tinyint(1) )
CP (cod)
CAj (cod) -> articulo

memoria (
cod varchar(7),
tipo varchar(30) )
CP (cod)
CAj (cod) -> articulo

objetivo (
cod varchar(7),
tipo varchar(15),
montura varchar(15),
focal varchar(10),
apertura varchar(10),
especiales varchar(35) )
CP (cod)
CAj (cod) -> articulo
```

```
pack (
cod varchar(7) )
CP (cod)
CAj (cod) -> articulo

ptienea (
pack varchar(7),
articulo varchar(7))
CP (pack,articulo)
CAj (articulo) -> articulo
CAj (pack) -> pack

stock (
articulo varchar(7),
disponible int(11),
entrega set( 'Descatalogado',
'Próximamente', '24 horas', '3/4 días',
'1/2 semanas' ) )
CP (articulo)
CAj (articulo) -> articulo

cesta (
articulo varchar(7),
usuario varchar(50),
fecha datetime )
CP (articulo,usuario)
CAj (articulo) -> articulo
CAj (usuario) -> usuario

pedido (
numPedido int(11),
usuario varchar(50),
fecha datetime )
CP (numPedido)
CAj (usuario) -> usuario VNN
VNN (fecha)

linped (
numPedido int(11),
linea int(11),
articulo varchar(7),
importe decimal(9,2),
cantidad int(11) )
CP (linea,numPedido)
CAj (articulo) -> articulo VNN
CAj (numPedido) -> pedido
VNN (importe)
```

ejemplo

añadir

2. Consulta de tablas

En este apartado consideramos sobre tablas ya creadas y aprenderemos a como realizar consultas:

2.1. Tipos de datos

Debemos conocer los tipos de datos con los que trabajaremos y asi poder hacer una correcta referencia a estos a la hora de hacer consultas. Los mas usados son:

1. VARCHAR(X), cadena de caracteres de longitud variable con un máximo de X ($1 \leq X \leq 4000$).
2. CHAR(X), cadena de caracteres de longitud fija con un máximo de X ($1 \leq X \leq 2000$).
3. INT, INTEGER, numeros enteros

4. DECIMAL(p,s), numeros con precision p y escala s $(1 \leq p \leq 38)(-84 \leq s \leq 127)$.
5. DATE, datos tipo fecha, dados de la siguiente forma: 'yyyy-mm-dd'.
6. DATETIME, datos tipo fecha y hora, dados de la siguiente forma: 'yyyy-mm-dd hh:mm:ssss'.
7. NULL, denotado por NULL, a la hora de trabajar con el hay que saber que no se pueden usar terminos de equivalencia usuales('=','¡',...), se debe de usar el comando '[not] is' para comprobar si un termino [no] es vacio.

Al usar constantes recordad que los tipo texto y los tipo fecha vienen dados entre comillas simples y el decimal en números se indica con un '.' para separar la parte entera de la fraccionaria.

Hay casos en los que la fecha no es necesario que venga dada por comillas simples pero esas ocasiones no se veran ahora, escribid siempre en el formato dado.

2.2. Consultas basicas

Partiendo de lo basico, trabajaremos con la siguiente estructura:

select-from	⇔	consulta general
where	⇔	condicionales basicos
order by	⇔	consulta general

Debe de seguir este orden y separaremos consultas con ';'.

2.2.1. Consulta general

Con los comandos select-from y order by podemos hacer una seleccion de algunos o todos los campos de una tabla y organizarlos, pero se mostarian todos los elementos existentes en los campos seleccionados. Estos funcionan de la siguiente manera:

select [opc_sel] [campo],[campo], ...
from [tabla] [ind_tabla]
order by [campo] [opc_ord], [campo] [opc_ord],...

-[opc_sel] : comando que modifica la selección, [DISTINCT] es el que usaremos
-[campo]: selecciona el campo o campos. '*' indicara todos los campos
-[tabla]: tabla con la que trabajar (mas adelante veremos como trabajar con mas)
-[ind_tabla]: renombramos temporalmente a la tabla
-[opc_ord]: ASC o DESC, modifica el como se ordena el campo en concreto.

Por ejemplo, para hacer explicaciones definimos la tabla PRUEBA={campo1,campo2,campo3} habiendo 5 elemntos en cada campo, con campo1 siendo clave primaria y el campo2 siendo tres elementos NULL y 2 siendo 'hola', campo3 seran una sucesion numerica ascendente.

Si usamos [opc_sel]=[DISTINCT], nos mostrara todas las filas que sean distintas pero tened en cuenta que si indicamos que se vea el campo1, se mostraran todos los elementos ya que no hay dos elementos iguales en el campo1.

Si usamos [opc_ord]=[ASC], mostrara en order ascendente en el campo seleccionado (el mayor sera el ultimo elemento en mostrarse en la tabla) y si es [opc_ord]=[DESC], se hara de orden descendente (el valor mas pequeño estara en la base de la tabla).

En el select podemos introducir constante, numericas, de tipo texto, de tipo fecha e incluso de tipo NULL. Habra que llamarlas de forma correcta y lo que haran sera rellenar la columna que se le reserva con la constante introducida.

Veamos un ejemplo de dos tablas, usando y sin usar campo1 en el select:

```
select DISTINCT p.campo1,'y' p.campo2
from prueba p
order by campo1 ASC;
```

campo1	y	campo2
1	y	NULL
2	y	hola
3	y	hola
4	y	NULL
5	y	NULL

```
select DISTINCT 123, p.campo2
from prueba p
order by campo2 DESC;
```

123	campo2
123	hola
123	NULL

2.3. Condicionales

Los condicionales vienen dados por lo general en el where y son el metodo mas sencillo para relacionar dos tablas por sus claves primarias, aunque tambien podemos delimitar que informacion queremos que se muestre usando otro tipo de comparadores, un resumen seria:

Si hiciesemos un select-from con dos tablas sin relacionarlas de algun modo, nos daria un producto vectorial de estas relacionando cada elemento sin importar si estos tienen relacion. Para ello hay que incluir un comando con el que relacionamos estos. Hay muchas formas que ahora indicaremos, veremos las mas sencillas primero y mas adelante ampliaremos las mas complejas:

1. tipo reunion: formado por los comando tipo join. Algunos de ellos dependen del where y otros no, ademas de que tablas se especifiquen en el from, se tratara de explicar lo mas claro y conciso posible. Haremos un breve resumen de estos e indicaremos con un tick cuales se usan en el where, el resto se veran mas adelante:

   x Equijoin: usado en where consta de igualdades usuales para concatenar filas de varias tablas

   x Self join: usado en where, usa igualdades (o desigualdades '<>','<','>','!=',...) para relacionar elementos de una misma tabla. P.e. que campo tiene igual valor que otro campo.

   - Inner join o simple join: similar al equijoin pero este no usa en entorno where. Es una forma de no olvidarse de relacionar las tablas con lo cual el where se puede omitir o reservar para otro tipo de condiciones. Usa el comando join, veremos mas adelante como se trabaja.

   x Antijoin: dadas en el entorno where, vienen denotadas por el comanto NOT IN (SUBCONSULTA) y mostrara los campos del select que no estan relacionados con la SUBCONSULTA.

   x Semijoin: es un equijoin pero se consultan unicamente las filas de una de las tablas, el resto actuan unicamente para delimitar la informacion que se muestra.

   - Outer join: Los outer join tienen tres variedades: left, right y full (no soportado en mySQL). Aunque de funcionameinto similar al inner join, segun cual escojamos, dara preferencia a mostar las filas de una tabla o de otra.

     P.e. el left join, dara preferencia a mostar todas las filas de la tabla a la izquierda (en el select) antes que a la derecha (la del join) y mostrara todas las filas de la tabla del select aunque no haya ninunga con la que se relacione en la del join (dando NULL)

2. Rangos: en el entorno where, nos permite comprobar un numero que se encuentra (o no) en un intervalo cerrado.

3. Subcadenas de caracter:en el entorno where, comprueba cadenas de caracter que contienen (o no) una subcadena de caracteres.

4. Listas o subconsultas: se ha comentado por encima al explicar el antijoin, usado en el entorno where permite comprobar que valores son iguales a una lista dada. Si hacemos una subconsulta en la que solo se comprueba un campo, obtendremos una lista de valores en relacion a ese campo.

Sabiendo que podemos usar AND o OR para encadenar condiciones y NOT para negarlas, veamos los condicionales de la siguiente manera:

2.3.1. Tipo reunion en el where

Mejor verlo con un ejemplo, trabajaremos con el siguiente ejemplo:

Usuarios

DNI (CP)	Nombre	Apellido	Tlfn (VNN)	email
1234	Juan	Martinez	NULL	jm@gmail.com
1235	Martin	Marcos	9876	mm@gmail.com
1236	Eva	Cifuentes	9877	ec@gmail.com
1237	Juan	Sirvent	NULL	js@gmail.com
1238	Jack	Daniels	9875	jd@gmail.com

Estado

DNI (CP)	estado
1234	no apto
1235	apto
1236	apto
1237	no apto
1238	apto

x Equijoin: igualdades usuales para concatenar filas de varias tablas.

```
select u.Nombre,'es',e.estado
from Usuarios u, Estado e
where u.DNI=e.DNI;
```

Nombre	es	estado
Juan	es	no apto
Martin	es	apto
Eva	es	apto
Juan	es	no apto
Jack	es	apto

```
select u.Apellido, e.estado
from Usuarios u, Estado e
where u.Tlfn is NULL AND u.nombre='Juan';
```

Apellido	estado
Martinez	no apto
Sirvent	no apto

x Self join: relaciona elementos de una misma tabla.

```
select u1.Apellido,'se llama como',u2.Apellido
from Usuarios u1, Usuarios u2
where u1.DNI>u2.DNI AND u1.Nombre=u2.Nombre;
```

Apellido	se llama como	Apellido
Sirvent	se llama como	Martinez

```
select u1.Apellido,'se llama como',u2.Apellido
from Usuarios u1, Usuarios u2
where u1.DNI<>u2.DNI AND
u1.Nombre=u2.Nombre;
```

Apellido	se llama como	Apellido
Sirvent	se llama como	Martinez
Martinez	se llama como	Sirvent

Notad como no es lo mismo > y <>, con lo primero sera si a>b y con el segundo sera si a>b o a<b, fijandonos en dos tablas con los mismos valores y clave primaria con el dni.

x Antijoin: muestra las filas de una tabla que no cumple la condicion dada por otra tabla, suele hacerse con consultas:

```
select * from Usuarios
where DNI not in (select DNI from Estado
where estado='apto');
```

DNI (CP)	Nombre	Apellido	Tlfn (VNN)	email
1234	Juan	Martinez	NULL	jm@gmail.com
1237	Juan	Sirvent	NULL	js@gmail.com

mostrara los elementos que en la tabla estado no estén definidos como 'apto'.

x Semijoin: es un equijoin pero se consultan unicamente las filas de una de las tablas, el resto actuan unicamente para delimitar la informacion que se muestra.

```
select u.* from Usuarios u, Estado e
where u.DNI=e.DNI AND e.estado='no apto';
```

DNI (CP)	Nombre	Apellido	Tlfn (VNN)	email
1234	Juan	Martinez	NULL	jm@gmail.com
1237	Juan	Sirvent	NULL	js@gmail.com

mostrara los elementos que en la tabla estado estén definidos como 'apto'.

2.3.2. Rango

Dado $x \in$[campo] y el intervalo cerrado [a,b], el comando Rango te devolvera los datos que cumplan $a \leq x \leq b$, dicho de otra manera $x \in [a, b]$. Para ellos se define en el where tal que:

campo [NOT] BETWEEN a AND b

2.3.3. Subcadenas de caracteres

Teniendo campo compuesto por cadenas de caracteres, esta funcion nos permite comprobar su una subcadena dada de diversos formatos esta dentro de alguna de las cadenas del campo. Veamos como se define en el where y los formatos disponibles y su utilidad:

campo [NOT] LIKE 'subcadena' ⇔ Busca exactamente la 'subcadena'
campo [NOT] LIKE 'subcadena%' ⇔ 'subcadena(*subcadena opcional*)'
campo [NOT] LIKE 'subcadena_' ⇔ 'subcadena(*un unico caracter opcional*)'

la subcadena y el caracter opcional se buscara en el sentido que se ponga, es decir X_ buscara a la derecha y _X buscara a la izquierda.

2.3.4. Listas

Comprueba que un campo tenga equivalente en una lista dada. Si hacemos una subconsulta en la que la select dependa de un unico campo, obtenemos tambien una lista (un vector fila o columna, no una matriz/tabla). Se define en el where con:

campo [NOT] IN (LISTA/SUBCONSULTA)

Las listas viene dadas: (dato1,dato2,dato3).

Es una forma mas sencilla y compacta de hacer condicionales tipo reunion encadenados.

2.3.5. Tipo reunion fuera del where

Los inner join se una forma de escribir los equijoin fuera del where, puede ser lioso ya que en el select pones campo que no tienen que estar estrictamente en la tabla definida en el from pero introduces la tabla mas adelante al usar el comando join. Los outer join por otro lado, se escriben como los join pero con una codigo mas (LEFT, RIGHT o FULL) delante del join y esto hara referencia a que tabla tendra preferencia de ser mostrada completamente, LEFT sera la tabla a la izquierda del join, RIGHT sera la de la derecha y FULL mostrara las dos tablas, si hay elementos no relacionados rellenara los campos necesarios con NULL. El FULL no esta definido para mySQL asique no lo veremos pero se sabe lo que hace.

Veamos un ejemplo recordando la tabla que usabamos antes pero ampliandola con un usuario mas que aun no eta dado de alta:

Usuarios

DNI (CP)	Nombre	Apellido	Tlfn (VNN)	email
1234	Juan	Martinez	NULL	jm@gmail.com
1235	Martin	Marcos	9876	mm@gmail.com
1236	Eva	Cifuentes	9877	ec@gmail.com
1237	Juan	Sirvent	NULL	js@gmail.com
1238	Jack	Daniels	9875	jd@gmail.com
1239	Franchesca	Espla	9874	fe@gmail.com

Estado

DNI (CP)	estado
1234	no apto
1235	apto
1236	apto
1237	no apto
1238	apto

inner join

```
select nombre, estado
from Usuarios u
JOIN estado e ON
e.dni=u.dni
```

nombre	estado
Juan	no apto
Martin	apto
Eva	apto
Juan	no apto
Jack	apto

left outer join

```
select nombre, estado
from Usuarios u
LEFT JOIN estado e ON
e.dni=u.dni
```

nombre	estado
Juan	no apto
Martin	apto
Eva	apto
Juan	no apto
Jack	apto
Franchesca	NULL

right outer join

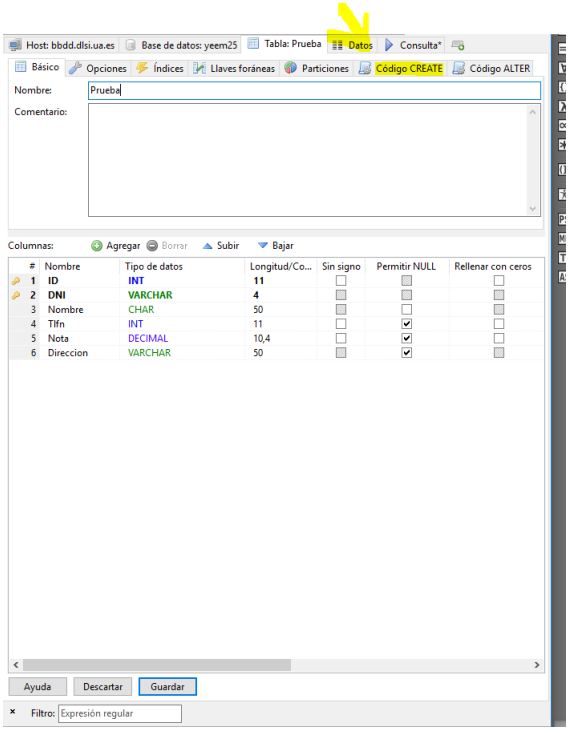
```
select nombre, estado
from Usuarios u
RIGHT JOIN estado e ON
e.dni=u.dni
```

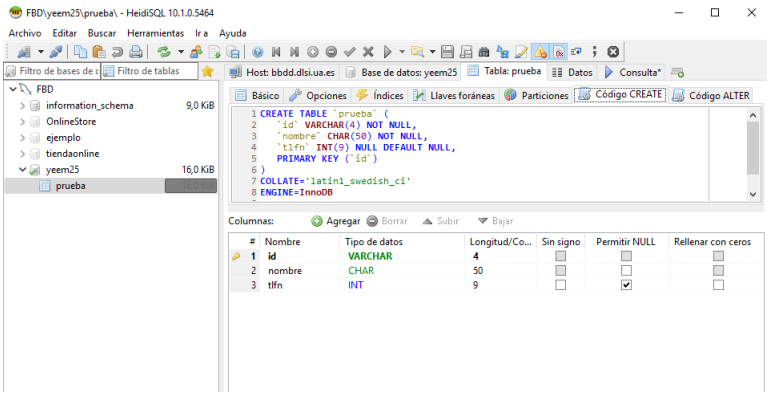
nombre	estado
Juan	no apto
Martin	apto
Eva	apto
Juan	no apto
Jack	apto

3. Creacion de tablas

Hay que tener claro las dieferencias entre las claves, aunque basta con saber la clave primaria.

Se pueden crear tablas facilmente dandole a una base de datos, click derecho, new y table, le das el nombre y añades columnas, es muy intuitivo. Podras dar a una columna formato de clave al darle click derecho a la columna en cuestion. Esta foto demuestra el entorno que apariencia debe de tener (en la pestaña codigo CREATE esta el codigo usado para generar la tabla).





Para borrar una tabla usamos DROP TABLE nombre_tabla.

Para modificar una tabla podemos usar el insert, el delete y el update (esto trabajara con las filas, podemos hacer una consulta para que esta introduzca que valores queremos introducir viniendo de otra tabla de esa misma base de datos).

De esto sinceramente no espero nada muy complicado, gg.

ejemplo de ejecucion de esto en consulta

INSERT INTO prueba VALUES ('234l','pepe',987654321);

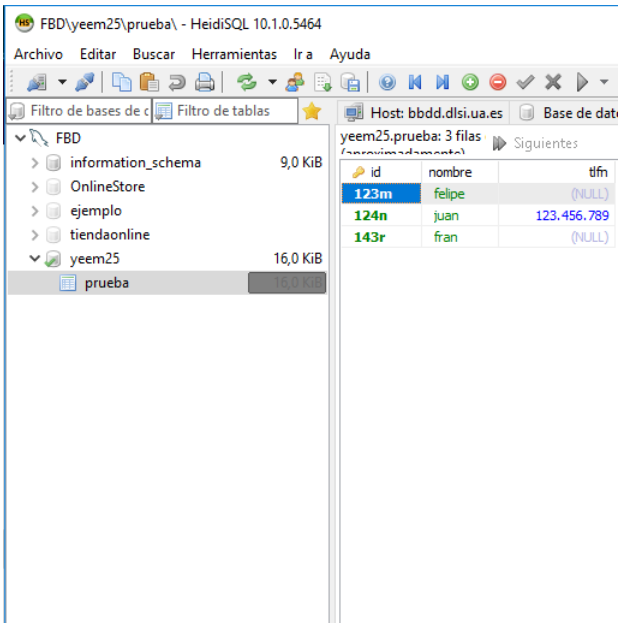
DELETE FROM prueba WHERE tlfn IS NOT NULL;

INSERT INTO prueba (id,nombre) VALUES ('235s','pepa');

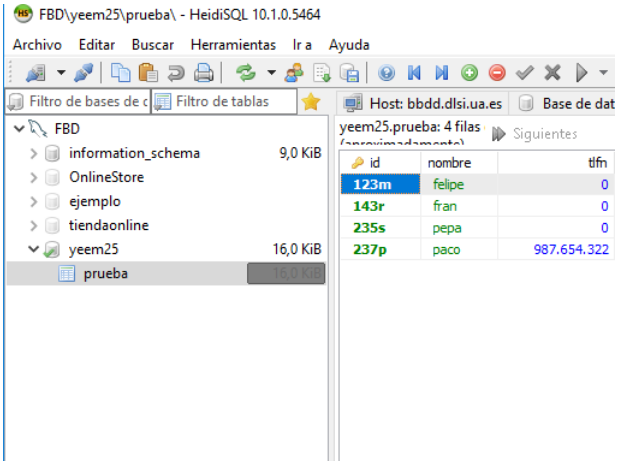
INSERT INTO prueba VALUES ('237p','paco',987654322);

UPDATE prueba SET tlfn = 0 WHERE tlfn IS NULL;

antes



despues



En laa pestaña de Llaves foraneas puedes crear las claves ajenas para una tabla haciendo referencia a otra tabla, recordad que las claves primarias son PK, las claves candidatas son UK+VNN y que por ahora las restricciones vistas en el **update** y en el **delete** son el **set NULL**, es decir, si se elimina algun valor que depende de otro pondra a NULL ese valor, y el **cascade** que eliminara o creara datos asociados a la consulta realizada (ya sea update o delete).

4. Mas comandos, calculos y agrupacion

Las funciones de calculo pueden ser usadas tanto en el select como en el where (es mas, si se ponen en el select se pueden etiquetar al ponerle justo a continuacion la etiqueta deseada, sin venir necesariamente dada como un tipo char. Sabemos las operaciones basicas (comparacion, adicion, multiplicacion, division) pero veamos algunas mas avanzadas:

Funciones de agregados:

count(*)	↔	numero de filas totales
count([DISNTINCT] expr)	↔	numero de valores distintos en expr
sum([DISTINCT] expr)	↔	suma de todos los valores en expr
avg([DISTINCT] expr)	↔	promedio de todos los valores en expr
min/max(expr)	↔	el minimo/maximo de todos los valores en expr

Otras funciones vienen dadas por tablas como 'dual'

Podemos encontrar mas funciones aqui (WIP, añadir paquete hyperlinks)

Estas funciones podemos usarlas tambien en subconsultas que estan anidadas, por lo tanto nos facilita el obtener ciertos datos de interes. Ademas, la opcion DISTINCT opcional indicada anteriormente implica que los NULL se omiten, es decir, aunque una fila admita NULL como un valor, si cuentas las filas respecto a un campo, y para una fila dada es NULL, esta se omite. Alomejor para el comando count no tiene importancia pero para el avg si, no es lo mismo la media de 4 elementos siendo uno NULL que la de 3.

Pero, ¿y si queremos separar o, mejor dicho, agrupar las funciones respecto a algun campo en especial? Para eso utilizamos el comando Group by (detras del where), el cual forzara a diferenciar las distintas filas del campo que hemos decidido que se haga una agrupacion. Es importante saber que, por lo general, el campo esencial por el que debemos de agrupar debe de ser el de algun tipo de clave candidata. (Es de gran uso para diferenciar valor aunque no se quiera indicar el campo que hace que sean distintos, es decir, la clave candidata esta en el group by pero no en el select).

Es mas, podemos filtrar estas agrupaciones usando el comando **having** a continuacion, el cual permite mostrar las filas que cumplan las condiciones indicadas. Parecera que es como un where, y no vais mal encaminados, pero where no trabaja bien para filtar las funciones agrupadas, mientras que el having si. Cabe decir que ni el where ni el having pueden trabajar con etiquetas de funciones agrupadas, si se usa debe de ser la propia funcion escrita de nuevo, u otra funcion ajena para filtar.

Podemos etiquetar subconsultas y operaciones usando FUNCION as ETIQUETA, podremos usar esta etiqueta en otras consultas o funciones como un parametro mas.

5. Tablas temporales y vistas

La caracteristica principal que comparten ambos conceptos es que ambos representan estructuras como las tablas.

Las tablas temporales podemos entenderlas como subconsultas (consultas realizadas dentro de otras consultas) usadas como columnas en el select o como tabla en el from o las temporary table en MySQL todas estas se eliminaras al finalizar y contiene los datos que le hemos asignado.

Las vistas, aunque actuan de igual forma, incluye la posibilidad de añadir o eliminar filas si la base de datos sobre la que se trabaja lo permite, la vista hara referencia a una tabla en particular aunque con mas elaboracion se pueden conectar varias tablas.

Veremos como codificarlas y entender su funcionamiento basico.

5.1. Tablas temporales

Las subconsultas como columnas calculas consisten el introducir como una columna mas del select en la consulta principal un select que consiste en una unica columna. Se debe de tener en cuenta el relacionar las subconsultas con la consulta principal, es decir, no generar que para un elemento existen una variedad de respuestas ya que esto no se puede mostrar y no seria el resultado deseado, si lo fuese significaria que es clave alternativa y puede permitirse el repetir lo que podria ser la clave primaria.

Las subconsultas en el form si seran necesaria etiquetarlas, y seria recomendable etiquetar tambien las subconsultas incluidas en esta, ya que lo que estamos haciendo es improvisar una tabla en la que el nombre sera la etiqueta que le pongamos y tendra tantas columnas como hayamos definido en ella (de ahi que sea recomendable etiquetar las subconsultas incluidas, ya que habra que hacer referencia de alguna manera a estas columnas de informacion).

Las tablas las podemos invertir si descomponemos la clave primaria de una tabla y la convertimos en las distintas columnas, es decir, si una consulta nos pide contar el dia que has asistido a clase por mes devolvera una columna mes y otro de total dias, siendo cada fila los distintos meses y los respectivos dias asistidos pero, si quisieramos la tabla invertida mostraria cada mes en distintas columnas y la

cantidad de dias. Para ello, debemos hacer tantas subconsultas como columnas queramos con una condicion que denotara a la clave primaria en la consulta, en este caso, el mes, seria importante el dejar indicada cada subconsulta con una etiqueta para entender sobre que se hace una busqueda.

Las temporary table actuan exactamente como una tabla, solo que desaparecen al finalizar la sesion, puedes definir que columnas son cada tipo de clave, si tiene foraneas conectarlas y rellenarla con una consulta, podras machacarla tanto como desees y olvidarte de usar el drop table, no hay mas, ez. Segun la configuracion de la base de datos y de si la tabla se ha definido dependiendo de otra tabla, se pueden insertar columnas en esta (modificando o sin modificar la tabla dependiente).

5.2. Vistas

Las vistas son objetos que se comprotan como una tabla y se definen a partir de una consulta, segun la definicion de la tabla sobre la que se basa puede hacer mas o menos cosas, añadir, eliminar o modificar filas serian funciones posibles de realizar sobre ellas.

Usando CREATE VIEW nombre_vista AS consulta generamos la vista y, con DROP VIEW nombre_vista la eliminamos del catalogo. Si queremos conocer la definicion de una vista usamos SHOW CREATE VIEW nombre_vista aunque no nos sera necesario.

Vistas que se basan en una consulta de una unica tabla son sencillas de tratar, incluso de modificar si la tabla lo permite, recordad que existe un cierto nivel de dependencia entre vistas y las tablas en las que toman los datos las consultas. El modificar una vista funciona igual que una tabla, usando insert, delete y update.

Las vistas cuyas consultas relacionan varias tablas son mas complejas. Podriamos crear una vista que combina dos tablas sin problema, practicamente seria una consulta que guardamos, pero si decidiesemos modificarla añadiendo valores es probable que genere errores si no consideremos que la interseccion puede afectar a mas de una tabla. Si de la vista escogemos unicamente valores que hacen referencia a una de las tablas estos valores se podrian añadir, pero alomejor no veriamos su adicion a la vista, si decidiesemos rellenar el resto de valores con nulos, daria error. Tu simplemente ten en cuenta que con mas de dos tablas hay que trabajar de otra manera.

La otra manera con la que trabajar es añadiendo la condiciton de 'WITH CHECK OPTION' la cual obliga a que se cumplan las conidiciones de la vista en el caso de querer modificarse valores de ella. Si no se usara esta condicion, los valores se añadiran a la tabla pero, al no cumplir las condiciones de la vista no se veria en ella.

Si usamos CREATE OR REPLACE omitimos el paso de de hacer DROP objeto IF EXISTS, es una instruccion que no se ve pero aparece.

6. Ejemplos complejos

Pongamos ejemplos de varias consultas y la salida esperada:

1. Lista todos los email, nombre y apellidos de usuario de Alicante, y si tienen pedidos también el total pagado por el usuario, usando dos vistas. La primera reune a los usuarios de alicante (VusuAli) y la segunda nos informa del total del importe a pagar por cada pedido en TIENDAONLINE y su fecha (Tpedidos).

   VusuAli: guarda toda la info de la tabla usuario en la que la provincia sea igual a 03

   Tpedidos: guarda el numero de pedido y la fecha de la tabla pedido y, comprueba en linped mediante el numpedido el importe y cantidad para cada pedido, multiplicando el importe por cantidad por cada linea y sumando el total de cada linea para obtener el total.

```sql
create or replace view VusuAli as select * from tiendaonline.usuario where provincia='03';

create or replace view Tpedidos as
select p.numpedido,sum(importe*cantidad) total,date(fecha)
from tiendaonline.pedido p, tiendaonline.linped l where p.numpedido=l.numpedido
group by p.numpedido,fecha;

select u.email,u.apellidos,u.nombre,pp.pagado
FROM VusuAli u
left join
(
    select usuario, sum(total) pagado
    from tiendaonline.pedido p -- necesitamos esta tabla porque la vista Tpedidos no incluye la
    join Tpedidos t on p.numpedido=t.numpedido
    group by usuario
) pp
on pp.usuario=u.email
```

VusuAli (4×89)

email	apellidos	nombre	pagado
acdlv@bitoben.mus.es	CASTAÑO DE LA VIEJA	ANTONIO	(NULL)
acg@hotmail.com	CRESPO GARCIA	ALFONSO	(NULL)
acm@colegas.com	CASARES MONTOYA	ALBERTO	9.557,90
adf@lolipop.com	DÜNHÖLTER FRIEDRICHS	ANNA	(NULL)
adlmm@ua.es	DE LA MORENA MARQUEZ	ANTONIA	2.576,00
agg@gmail.com	GIGANTE GARCIA	ASUNCION	(NULL)
agl@dlsi.ua.es	GURI LOPEZ	ALBERTO	(NULL)
alm@lolipop.com	LOPEZ MENDEZ	ALVARO	(NULL)
alr@agwab.com	LOPEZ RIPA	ALVARO	1.079,00
amg@lamail.ar	MARTINEZ GOMEZ	ANTONIO	(NULL)
amv@lamail.ar	MARTINEZ VASSALLO	ANGEL	(NULL)
apg2@colegas.com	PINTO GOMES-CASSERES	ADRIANA	(NULL)
asc@colegas.com	SOBREVIELA CORPAS	AINHOA	(NULL)
bmm@agwab.com	MARTIN MALMCRONA	BIRGIT	(NULL)
cfu@lamail.ar	FERNANDEZ UGALDE	CARLOS	(NULL)
clqs@colegas.com	QUINTERO SANZ	CESAR LUIS	(NULL)
cpa@colegas.com	PEREZ ALONSO	CAROLINA	(NULL)
csh@colegas.com	SANCHO HUESO	CAROLINA	(NULL)
csr@colegas.com	SANCHEZ REDONDO	CARLOS	(NULL)
czf@colegas.com	ZAMORA FRANCISCO	CARMEN	(NULL)
das@colegas.com	ARNANZ SANCHEZ	DANIEL	(NULL)
dgl@lamail.ar	GARCIA LAINEZ	DIANA	(NULL)
dmm@colegas.com	MALPARTIDA MORENO	DANIEL	(NULL)
dvl@colegas.com	VAZQUEZ LLORENS	DANIEL	(NULL)
emc@agwab.com	MARTIN CASTILLO	ELENA	(NULL)
emc@colegas.com	MOCHON CASTAÑOS	EMILIO	(NULL)

Se pide mostrar todos los usuarios de alicante, y si tienen pedidos mostrar el total pagado por el usuario. Para estos usamos un left join, que mostrar si o si todo lo que se haya definido a la izquierda del join (mostar los usuarios) y lo que se defina a su derecha rellenara con NULL en el caso de no existir, recordad que la estructura de los reunion fuera del where son JOIN (subconsulta)[etiqueta] ON relacion consulta-subconsulta.

En la subconsulta a la derecha del left join, a la cual se la etiqueta con pp, se trata de obtener la suma del total de cada pedido que le corresponde a cada usuario.

Si usasemos JOIN mostaria los usuarios de alicante QUE SI tienen pedidos.

2. pa que mas, si con lo que hay ahora ya esta... quedate con el que puedes mostrar dos select seguidos con las mismas columnas si usas UNION entre cada consulta