JOINS EN MYSQL BIEN EXPLICADO. TODO LO QUE NECESITAS SABER

Publicado el 11 de marzo de 2017 por VicHaunter

El mundo de las bases de datos a veces se puede volver un poco complicado, sobretodo con el tema de las tablas relacionales. Precisamente por eso voy a enseñarte **cómo funcionan los joins y cómo usarlos correctamente**, a ver si nos hacemos masters.

Si has terminado aquí es porque igual que yo has dado mil vueltas por internet tratando de entender como funcionan. El problema es que la mayoría de las fuentes tienen información casi especializada.

Al final, tras conseguir aprender cómo hacer joins en mysql, me gustaría compartirlo de forma más amena. Además, si te queda cualquier duda no te lo pienses y déjame un comentario, te echaré un cable encantado.

Qué es un JOIN en MySQL

Los JOIN son usados en una sentencia SQL para recuperar datos de varias tablas al mismo tiempo. Estas tablas tienen que estar relacionadas de alguna forma, por ejemplo, una tabla usuarios, y otra tabla juegos que contiene también la id del usuario al que pertenece el juego:

Tabla Usuarios:

ID	username

Tabla Juegos:

ID	juegoname	ID_usuario

Como puedes observar, podríamos asociar cada juego a un usuario mediante su ID. De esta forma con un JOIN uniríamos las dos tablas y extraeríamos en una sola consulta por ejemplo:

username
juegoname

¿Qué son las tablas relacionales?

Bueno, este punto lo meto por que el ejemplo de arriba es lo básico, pero en temas de rendimiento o escalabilidad es bastante limitado.

Las tablas relacionales, son tablas que se utilizan como intermediarios de otras dos tablas. Normalmente contienen solamente la id de cada uno de los elementos de otras tablas a asociar, y será con lo que trabajaremos para entender los JOINS.

No porque sea mi capricho, es que casi siempre te lo encontrarás de esta manera cuando trabajes con bases de datos MySQL.

Un ejemplo sacado del de arriba sería tener 3 tablas, usuarios, juegos y juegousuario (para seguir un poquito las convenciones).

Tabla usuario:

ID	username

Tabla juegos:

ID	juegoname

Tabla juegousuario:

ID_username	ID_juegoname

Ahora piénsalo detenidamente. Si tienes una tabla con todos los juegos que existen, y otra tabla con 200 usuarios, bastará con que cada usuario elija los juegos que tiene. Entonces tú guardas el id de ese usuario y el id del juego que tiene, y un mismo usuario puede tener varios juegos, o varios usuarios tener un juego en concreto.

Lo solucionaríamos con **un doble JOIN** que devolverá los usuarios y juegos por los parámetros que le hayamos pedido.

Datos de ejemplo

Por si quieres trastear, voy a dejarte aquí una consulta completa con estructura de tabla y datos para que la importes en una base de datos y puedas jugar con ella.

```
SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for juegos
-- ----------------------------
DROP TABLE IF EXISTS `juegos`;
CREATE TABLE `juegos` (
`ID` int(11) NOT NULL AUTO_INCREMENT,
`juegoname` varchar(255) DEFAULT NULL,
PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of juegos
-- ----------------------------

INSERT INTO `juegos` VALUES ('1', 'Final Fantasy VII');
INSERT INTO `juegos` VALUES ('2', 'Zelda: A link to the past');
INSERT INTO `juegos` VALUES ('3', 'Crazy Taxy');
INSERT INTO `juegos` VALUES ('4', 'Donkey Kong Country');
INSERT INTO `juegos` VALUES ('5', 'Fallout 4');
INSERT INTO `juegos` VALUES ('6', 'Saints Row III');
INSERT INTO `juegos` VALUES ('7', 'La taba');

-- ----------------------------
-- Table structure for juegousuario
-- ----------------------------
DROP TABLE IF EXISTS `juegousuario`;
CREATE TABLE `juegousuario` (
`ID_usuario` int(11) NOT NULL,
`ID_juego` int(11) NOT NULL,
UNIQUE KEY `id_user_juego` (`ID_usuario`,`ID_juego`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

```
-- ----------------------------
-- Records of juegousuario
-- ----------------------------
INSERT INTO `juegousuario` VALUES ('1', '1');
INSERT INTO `juegousuario` VALUES ('1', '2');
INSERT INTO `juegousuario` VALUES ('1', '3');
INSERT INTO `juegousuario` VALUES ('1', '4');
#INSERT INTO `juegousuario` VALUES ('1', '5');
INSERT INTO `juegousuario` VALUES ('1', '6');
INSERT INTO `juegousuario` VALUES ('1', '7');
INSERT INTO `juegousuario` VALUES ('2', '3');
INSERT INTO `juegousuario` VALUES ('2', '7');
INSERT INTO `juegousuario` VALUES ('4', '1');
INSERT INTO `juegousuario` VALUES ('4', '2');
INSERT INTO `juegousuario` VALUES ('4', '4');
INSERT INTO `juegousuario` VALUES ('4', '7');

-- ----------------------------
-- Table structure for usuarios
-- ----------------------------
DROP TABLE IF EXISTS `usuarios`;
CREATE TABLE `usuarios` (
`ID` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(255) DEFAULT NULL,
PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of usuarios
-- ----------------------------
INSERT INTO `usuarios` VALUES ('1', 'vichaunter');
INSERT INTO `usuarios` VALUES ('2', 'pepito');
INSERT INTO `usuarios` VALUES ('3', 'jaimito');
INSERT INTO `usuarios` VALUES ('4', 'ataulfo');
SET FOREIGN_KEY_CHECKS=1;
```

Recordarte que estoy ignorando todos los parámetros de rendimiento y demás porque no es de lo que se trata el tema. Solo un unique index para que no haya propietarios repetidos en la tabla relacional. Si luego te interesa podemos revisar tu estructura.

> **Léete también** **Páginas AMP, qué ventajas y desventajas de configurarlas tienes**

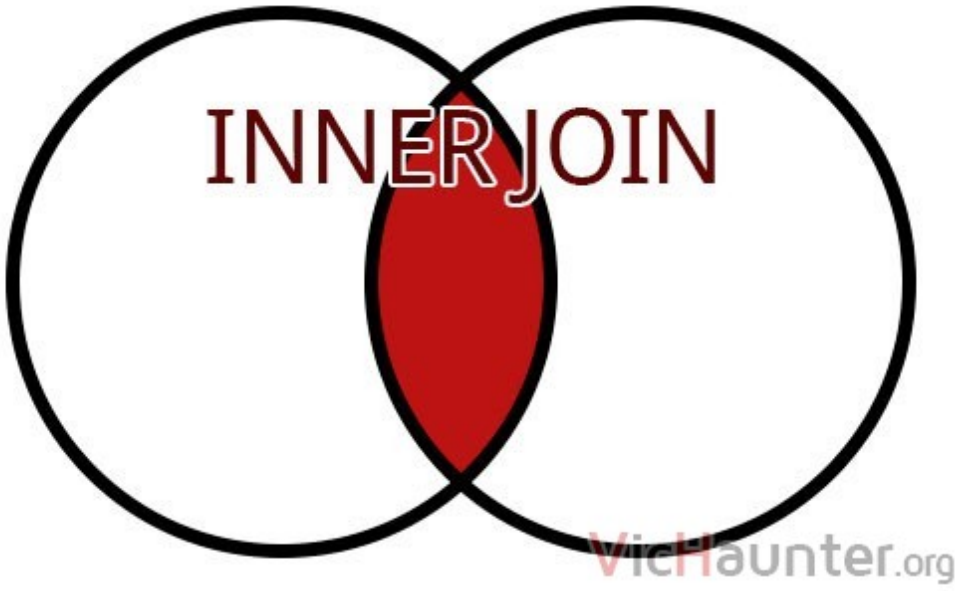
¿Qué tipos de join existen?

Ahora es cuando empieza la parte divertida, la teoría o el tocho, pero voy a ver si lo puedo hacer de forma amena.

Tenemos **varios tipos de JOINS**, los vamos a ver uno por uno y cada uno de ellos te servirá para extraer los datos de una forma específica. Dependiendo de lo que necesites tendrás que echar mano de uno u otro, o incluso combinarlos entre ellos, que es lo más complejo.

Cómo usar INNER JOINS y para qué sirven

Vale, empezaremos por los más estándar. Los puedes encontrar en el código como INNER JOIN o simplemente JOIN. Este tipo de unión te ayuda a combinar varias tablas, y te devuelve únicamente los datos que estén disponibles en todas las tablas a la vez.



Esto significa, que si por ejemplo haces un INNER JOIN para ver los juegos que tiene cada usuario, solo devolverá datos siempre que un juego pertenezca a un usuario. Si un juego no tiene ningún propietario, pero existe en la tabla, no aparecerá, y si un usuario no tiene ningún juego asociado tampoco verás a ese usuario.

La consulta sería algo así:

```
1 SELECT
2 usuarios.username,
3 juegos.juegoname
4 FROM
5 usuarios
6 INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
7 INNER JOIN juegos ON juegousuario.ID_juego = juegos.ID
```

Como puedes ver, al utilizar una tabla relacional **tienes que hacer dos joins**. El primer JOIN nos une la tabla usuarios, **con la tabla juegousuario**. Ahora que ya la tenemos unida, podemos utilizar los datos de la tabla juegousuario, y lanzar la **consulta con la tabla juegos en otro JOIN**.

Como este join devuelve solo los datos que hay en las dos tablas, esperamos un resultado como este:

vichaunter	Final Fantasy VII
vichaunter	Zelda: A link to the past
vichaunter	Crazy Taxy

vichaunter	Donkey Kong Country
vichaunter	Saints Row III
vichaunter	La taba
pepito	Crazy Taxy
pepito	La taba
ataulfo	Final Fantasy VII
ataulfo	Zelda: A link to the past
ataulfo	Donkey Kong Country
ataulfo	La taba

Como puedes ver, no hay ni rastro del usuario jaimito, ni del juego Fallout 4, pues ni el uno tiene juegos, ni el juego es de nadie.

Por supuesto esta consulta la podemos hacer simplemente con dos tablas, evitando una relacional. En una tienda de alquiler, en la que solo hay un juego de cada, bastaría con poner un campo id_usuario a cada juego como en la primera tabla, y hacer solo un join.

Cómo usar un LEFT JOIN

En este caso, el left join devuelve todos los resultados que coincidan en la primera tabla, con los datos que tenga de la segunda. En el caso de que falte algún dato, devolverá un valor null en lugar del dato, pero seguiremos teniendo el valor de la primera tabla.

Por ejemplo, si quieres saber todos los juegos de los usuarios, con un left join tendremos una lista completa de todos los usuarios, incluso si no tienen ningún juego.

```
SELECT
usuarios.username,
juegos.juegoname
FROM
usuarios
LEFT JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
LEFT JOIN juegos ON juegousuario.ID_juego = juegos.ID
```

El resultado sería el siguiente:

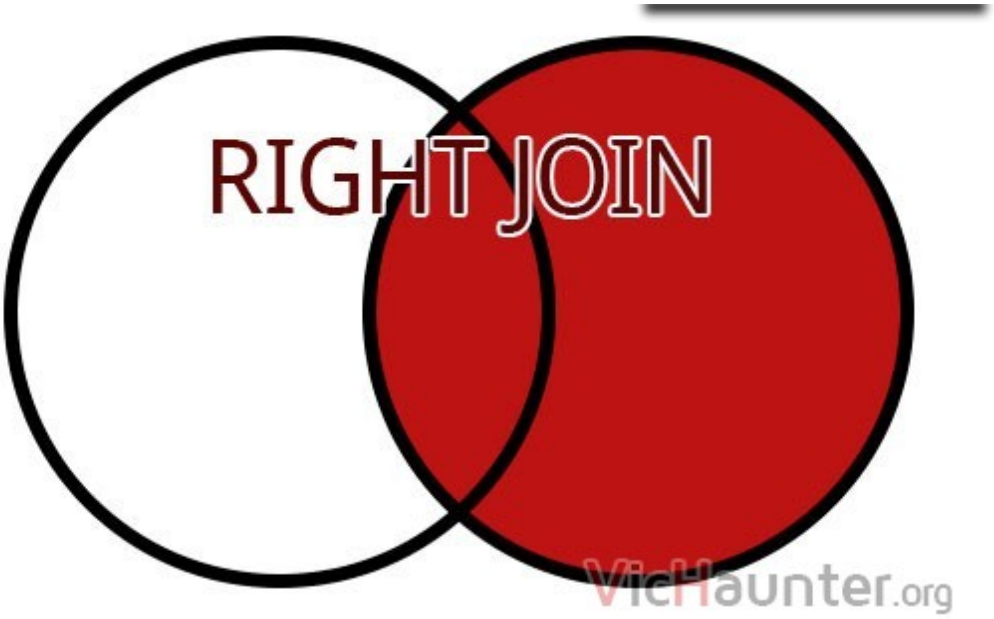
vichaunter	Final Fantasy VII
vichaunter	Zelda: A link to the past
vichaunter	Crazy Taxy
vichaunter	Donkey Kong Country
vichaunter	Saints Row III
vichaunter	La taba
pepito	Crazy Taxy
pepito	La taba
jaimito	
ataulfo	Final Fantasy VII
ataulfo	Zelda: A link to the past
ataulfo	Donkey Kong Country
ataulfo	La taba

Si te fijas, ahora sí que tenemos a jaimito, pero sin embargo nos muestra que no hay juego. Esto puede ser muy útil, por ejemplo para localizar todos los usuarios que no tienen juegos (con un WHERE juegoname IS NULL, por ejemplo).

Cómo usar un RIGHT JOIN

En el caso del RIGHT JOIN, pasa exactamente lo mismo que con el anterior, pero con la diferencia de que devuelve todos los datos de la tabla con la que se relaciona la anterior. Si estamos ejecutando un SELECT en la tabla usuarios, las demás serán tablas con las que se relaciona.

> **Léete también** **¿Te ha aparecido SlurpConfirm404 en los logs? Averigua por qué**



Por ejemplo, supon que quieres saber todos los juegos que tienes, y a qué usuarios pertenecen (o simplemente si le pertenece a algún usuario). Lo harías de esta forma:

```
SELECT
usuarios.username,
juegos.juegoname
FROM
usuarios
RIGHT JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
RIGHT JOIN juegos ON juegousuario.ID_juego = juegos.ID
```

Con ~~in~~esperados resultados:

vichaunter	Final Fantasy VII
ataulfo	Final Fantasy VII
vichaunter	Zelda: A link to the past
ataulfo	Zelda: A link to the past
vichaunter	Crazy Taxy
pepito	Crazy Taxy
vichaunter	Donkey Kong Country
ataulfo	Donkey Kong Country
	Fallout 4
vichaunter	Saints Row III
vichaunter	La taba
pepito	La taba
ataulfo	La taba

¡Ahora han cambiado las tornas! Como puedes observar, ya no hay ni rastro de jaimito, pero sin embargo aparece Fallout 4 sin estar asociado a ningún usuario. Esto realmente tiene muchas aplicaciones interesantes, solo [Privacy & Cookies Policy] inación.

Cómo usar OUTER JOIN o FULL OUTER JOIN

El **OUTER JOIN consiste en recuperar TODOS los datos** que haya en ambas tablas, tanto los que tienen contenido en ambos extremos, como los que no. Es la oveja negra en MySQL, ya que no es directamente compatible, pero sí se puede conseguir un efecto similar.

Para poder hacer este tipo de consultas tendrás que echar mano de UNION, que sirve precisamente para combinar los resultados de varios SELECT en una sola consulta (¿nunca lo has usado? es como hacer trampa).

```
SELECT
usuarios.username,
juegos.juegoname
FROM
usuarios
LEFT JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
LEFT JOIN juegos ON juegousuario.ID_juego = juegos.ID

UNION

SELECT
usuarios.username,
juegos.juegoname
FROM
usuarios
RIGHT JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
RIGHT JOIN juegos ON juegousuario.ID_juego = juegos.ID
```

Como seguro que ya te habrás fijado, la norma **para poder utilizar UNION**, es que en **todos los SELECT debe haber los mismos campos seleccionados**. En este caso username y juegoname, ya que si no no funcionará.

Esta consulta devolvería:

vichaunter	Final Fantasy VII	

vichaunter	Zelda: A link to the past
vichaunter	Crazy Taxy
vichaunter	Donkey Kong Country
vichaunter	Saints Row III
vichaunter	La taba
pepito	Crazy Taxy
pepito	La taba
jaimito	
ataulfo	Final Fantasy VII
ataulfo	Zelda: A link to the past
ataulfo	Donkey Kong Country
ataulfo	La taba
	Fallout 4

Como ves salen todos los datos, incluso los que no tienen nada en uno de los lados. De esta forma puedes crear varios filtros distintos después de extraerlos por ejemplo. Así con una única consulta a la base de datos tendrías todo lo que necesitas.

Entendiendo los JOINS

Vale, guay, muy bonito todo lleno de circulitos, consultas y tal, pero aún puedes tener alguna duda. ¿Por qué está haciendo las consultas con dos joins? ¿se pueden hacer con uno solo? ¿y si voy a juntar 4 tablas cómo lo hago?

Tranquilo, como comentaba en el título trato de dar una **explicación de joins que se entiende**, pero antes necesitábamos la base. Hasta ahora al menos ya sabes los tipos de joins que hay, y los datos que devuelve cada uno.

Si no es así, pégale un repaso a la sección anterior, y si aun así te quedan dudas puedes dejar un comentario.

Explicación del código de un join

Empezaré por lo más importante, la respuesta a la pregunta de cómo leches salen los datos de las tablas para usarse. Bien, cogeré una consulta de las primeras, a lo facilito:

```
1 SELECT
2 usuarios.username,
```

```
3 juegos.juegoname
4 FROM
5 usuarios
6 INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
7 INNER JOIN juegos ON juegousuario.ID_juego = juegos.ID
```

Veamos, diseccionaré toda la consulta SQL por si las dudas. Cada parte sirve para lo siguiente:

SELECT
usuarios.username,
juegos.juegoname

Aquí estás especificando que lo que quieres hacer es seleccionar datos (o un SELECT), en el que indicas los campos que quieres que devuelva, en el formato nombretabla.nombrecampo. De esta forma el select sabe de qué tabla coger el dato, ya que por ejemplo la columna ID se llama igual en las dos.

FROM
usuarios

El from indica la tabla principal a partir de la cual vamos a empezar a estraer datos, y de la que cogeremos un campo para relacionarla con otra tabla.

INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario

Aquí empieza la fiesta. INNER JOIN es el tipo de join que quieres hacer, ya has visto que puede ser INNER, LEFT o RIGHT.

Lo siguiente será indicarle sobre qué tabla queremos que coja los datos, en este caso la tabla juegousuario (que es nuestra relacional).

Bien, a partir de ahí **el ON, sería algo así como un where**, donde indicamos, el campo de nuestra tabla inicial (la del from) con el campo que relaciona, y la tabla sobre la que vamos a combinar datos, con su campo relacionado.

En este caso la ID en la tabla usuarios coincidirá con ID_usuario, en la tabla juegousuario. En ese momento **los campos de esa tabla que tengan coincidencias pasarán a poderse utilizar dentro de la consulta**, por ejemplo en el SELECT para mostrarlos, o como hemos hecho aquí en el siguiente join para unirla con una tercera tabla.

INNER JOIN juegos ON juegousuario.ID_juego = juegos.ID

Empezamos como antes, pero ahora sobre la tabla juegos. Lo siguiente que quieres hacer es de la tabla juegousuario (que tiene la id del usuario y la del juego), nos busque los juegos con esa ID. Como esos datos de juegousuario han quedado disponislbes en el primer join para toda la consulta, podemos usarlos aquí o en cláusulas where, etc.

> **Léete también** **Añadir extensiones a Dreamweaver CS4 (.phtml)**

Por tanto, pasa lo siguiente:

Seleccionamos unos datos, de la tabla usuarios. Unimos la tabla usuarios a la tabla juegousuario para saber las ids de los juegos que tiene cada usuario. Luego simplemente unimos esa tabla juegousuario a la tabla juegos mediante esas ids que están ligadas al usuario.

El resultado es una lista del tipo que hayamos elegido, con relaciones entre usuarios y juegos.

¿Puedo meter más de 3 tablas?

¡Por supuesto! Puedes hacer joins con las tablas que quieras, siempre que tu servidor aguante la carga. Piensa que cuantas más tablas metas en el join, y más datos tengan, más lenta será la consulta, o incluso puede saturar el servidor.

Un ejemplo de consulta con más joins, sería si por ejemplo queremos añadir categorías a los juegos y mostrarlos en la lista de usuario-juego-categoría. Voy a suponer que para esto tenemos una columna ID_categoría en la tabla juegos.

```
1 SELECT
2 usuarios.username,
3 juegos.juegoname,
4 categoria.catname
5 FROM
6 usuarios
7 INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
8 INNER JOIN juegos ON juegousuario.ID_juego = juegos.ID
9 INNER JOIN categorias ON juegos.ID_categoria = categorias.ID
```

Como ves, con solo añadir una línea, y el campo en el select hemos hecho que nos devuelva también el nombre de categoría para esa consulta.

¿Y un join con solo dos tablas?

También puedes hacerlo, de hecho te pongo este ejemplo basado en el esquema que serían las tablas de la sección **qué es un join en MySQL** (la primera del todo).

En ese caso la consulta se uniría directamente con la ID del usuario y el campo ID_usuario de la tabla juegos.

```
1 SELECT
2 usuarios.username,
3 juegos.juegoname
4 FROM
5 usuarios
6 INNER JOIN juegos ON usuarios.ID = juegos.ID_usuario
```

Tendrías los datos de los usuarios, y de los juegos que tiene cada uno. El problema de este tipo de tablas, es que suele estar limitado a un registro por cada relación, es decir, que no podrías asignar varios juegos a un solo usuario a no ser que tuvieras más campos en tabla, o lo procesaras luego en php por ejemplo.

Entonces, ¿Cómo uso los joins?

Si te estás haciendo esta pregunta no he sido todo lo claro que pretendía. Te rogaría me dejaras un comentario con tus dudas, te las explico y mejoraré el contenido del artículo en consecuencia.

La idea es que dependiendo de lo que necesites extraer de la base de datos usarás unos u otros.

Si vas a hacer unas **tablas con relaciones simples** 1 a 1 (1:1), bastaría con que uses el modelo del principio con **un solo join**.

En caso de que quieras utilizar otras relaciones tendrás que echar mano de una tabla relacional y hacer consultas con varios joins. En relaciones del tipo:

1 a muchos (1:n)

muchos a 1 (n:1)

muchos a muchos (n:n)

Estos tipos de relaciones, que son los que he usado en los demás ejemplos, te dan mucha más versatilidad. Te permiten que la relacional sirva para cualquiera de los 3 casos, ya que las limitaciones las pondrás en el código, y una asociación nueva será tan simple como añadir un campo con las ids.

Vale, hasta aquí si lo has ido pillando todo vamos bien, pero queda una duda que no se suele explicar en ningún sitio.

¿Se pueden utilizar varios JOIN diferentes en una misma consulta?

Sí, por supuesto. Puedes utilizar los joins que mejor te convenga para conseguir los resultados que te hacen falta. Dependiendo del que utilices se aplicarán las reglas que hemos visto arriba en cada JOIN.

Por ejemplo, supongamos que tenemos que unir varias tablas, y que terminamos con una consulta similar a esta:

```
1 SELECT *
2 FROM
3 contrataciones.empleado e
4 INNER JOIN persona.contacto c ON c.ID_contacto = e.ID_contacto
5 LEFT JOIN contrataciones.candidato dc ON dc.ID_empleado = e.ID_empleado
6 INNER JOIN ventas.vendedor vv ON vv.ID_vendedor = e.ID_empleado
7 LEFT JOIN ventas.orden vo ON vo.ID_personaventas = vv.ID_vendedor
8 LEFT JOIN ventas.zona vz ON vz.ID_zona = vv.ID_zona
```

A simple vista puede parecer un lío, pero si la explicamos detenidamente verás toda la lógica. Como puedes observar se hacen uso de INNER y LEFT para extraer los datos, dependiendo de los que hace falta en cada caso.

JOIN 1: El primer join une a los empleados y contactos. Esta nos devolverá solo a los empleados que tengan contactos (o datos en ambas tablas).

JOIN 2: Al hacer un LEFT en este caso se mantendrán todos los resultados del primer join, y se añadirán los datos sobre contrataciones para cada uno de los empleados que ya teníamos.

JOIN 3: En este se unen los vendedores al resultado del join anterior (recuerda que va en cascada), y al ser un INNER se eliminarán los resultados de antes que no tuvieran valores en los dos campos usados en la consulta.

JOIN 4: Otro OUTER, en este caso extrayendo todas las filas del resultado anterior y las ventas, pero haya coincidencias o no en la tabla de ventas.

JOIN 5: Igual que la consulta de antes, está pensada para añadir las zonas a los datos que ya tenemos, pero sin omitir ninguno de los vendedores, tengan esos territorios o no.

Así puede que parezca un poco confuso, pero si juegas un poco con ellos en seguida verás qué datos filtra cuando cambias un left por right por ejemplo (que haría que eliminara los que no tengan dato a la derecha, pero sí puedan tener vacío a la izquierda.

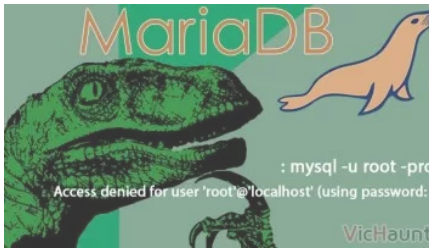
Reuerda que también puedes hacer joins con una sola tabla, por ejemplo si tienes un valor parent por ejemplo. Puedes ir escalando un nivel con cada join left y así conseguir devolver todos los niveles de categorías. Pruébalo y me cuentas.

¿Te ha quedado claro como funcionan los joins y como usarlos? Comenta y comparte

Relacionado

Resetear contraseña de root en MySQL de MAC
22 de noviembre de 2013
En «Ayuda»

MySQL index unique error aumenta el valor del auto increment [Solucionado]
21 de octubre de 2013
En «Ayuda»



Cómo solucionar que root no puede acceder a MariaDB tras instalarlo
7 de mayo de 2017
En «Ayuda»

Etiquetado BASES DE DATOS MYSQL PHP SINTAXIS

Publicado en DESARROLLO WEB por **VicHaunter** el 11 de marzo de 2017

ANTERIOR SIGUIENTE

datos
INSERT INTO `juegousuario` VALUES ('1', '5');
Este registro no debería existir para que funcionen correctamente todos los ejemplos.

Muy buen tutorial

^ | v • Responder • Compartir ›

VicHaunter Moderador ➚ Leijaca • hace un año

Tienes razón, gracias por la observación, ya lo he corregido.

^ | v • Responder • Compartir ›

Hausser Hssr • hace 2 años

tienes un tuturial igual pero en vez de hacer consultas seria agregar datos y actualizarlos con tablas referenciadas hasta 3 niveles....

^ | v • Responder • Compartir ›

disson cetre • hace 2 años • edited

Hola tengo una duda al respecto,

Investigué que es ideal usar Join en vez de Select, así que me tomé la tarea de modificar una consulta.
En mi caso tengo cuatro tablas:
T1_Pregunta
T2_Opciones
T3_Registro
T4_Usuario

En T4_Usuario tengo los nombres de los usuarios que contestarán a las preguntas (T1_Pregunta), T2_Opciones, está ligada a una o varias preguntas (T1_Pregunta) y en T3_Registro guardo el T4_Usuario.ID - T1_Pregunta.ID - T2_Opciones.ID.
Finalmente me gustaría saber si utilizando joins, puedo cargar sólo las T2_Opciones que aún no existen en T3_Registro utilizando el ID por supuesto.

Espero haber sido claro, gracias!

^ | v • Responder • Compartir ›

VicHaunter Moderador ➚ disson cetre • hace 2 años

Realmente depende mucho de cada caso. Si bien un join es casi siempre la mejor opción para juntar datos de varias tablas y devolverlos en una misma consulta, para hacer filtros depende de la ocasión. Piensa que un join es un select que utiliza los campos anteriores para filtrar los resultados y añadirlos, por lo que select sigue

siendo más rápido siempre que no se hagan en subconsultas para múltiples resultados.

En tu caso es mucho mejor hacer algo como esto:
SELECT
*
FROM
Opciones o
WHERE o.id NOT IN (SELECT r.opcion_id
FROM Registro r GROUP BY r.opcion_id)

Lo que haces aquí es seleccionar todos los que no tengan una id determinada, y esta lista viene de una subconsulta con todas las ids de opciones de la tabla registro. De esta forma estás ejecutando solo dos consultas ya que recogiendo esta lista (siempre que la tabla no sea exageradamente grande), es más rápido.

Responder • Compartir ›

Kethox • hace 2 años

Porfin una web que lo explica bien, muchas gracias por el aporte !!

1 Responder • Compartir ›

jared ➚ Kethox • hace 4 días

todavia no se muere

Responder • Compartir ›

VicHaunter Moderador ➚ jared
• hace 2 días

Espero que de momento no xD

Responder • Compartir ›

German • hace 2 años

Buenas tardes, duda:

Tomando en cuenta el ejemplo primero, cómo sería el query que me traiga los usuarios que no tengan el juego ´X´ ?

Responder • Compartir ›

Storm Sith 13 • hace 2 años

como se pondria una condicion

Responder • Compartir ›

CJ • hace 2 años

En forma teórica, siempre me ha costado entender un join, pero con esta explicación, me ha quedado claro que es un JOIN

que es un JOIN.
Solamente me queda si el Outer apply es un tipo de join?

Responder • Compartir ›

VicHaunter Moderador ➚ CJ • hace 2 años

No exactamente. Los operadores APPLY sirven para unir expresiones. Es decir, sería parecido a un join pero filtrando con la segunda expresión cada fila de la primera.

Me lo apunto para un artículo y me alegro de

BUSCAR

Buscar …

¿ME INVITAS A UNA CERVEZA?

Si te ha sido de tanta ayuda que me lo quieres agradecer de alguna forma y no sabes como, esta puede ser una opción..

Donar

COMENTARIOS RECIENTES

Ana Torres en Tarda dos meses un paquete desde china!! Que lo traen, ¿en bicicleta?

Jose Juarez en ¿Cuanto puede tardar mi paquete en pasar aduanas?

giampiero 2018 en Tarda dos meses un paquete desde china!! Que lo traen, ¿en bicicleta?

Hsm Leiva Jara en Como solucionar el error MSCOMCTL.OCX y mscomct2.ocx no registrado en Windows 10, 8 y 7

michelle cabrera en Por qué hace toques fantasma mi smartphone y como solucionarlo

CATEGORIAS

- Alternativas
- Ayuda
- Cómo funciona
- Comprar en China
- Desarrollo Web
- Electrónica
- General
- Informática
- Marketing Online
- Otros
- Programación
- Reviews
- Tecnología
- Videojuegos

ARCHIVOS

- Año 2015
- Año 2014
- Año 2013
- Año 2012
- Año 2011
- Año 2010
- Año 2009
- Año 2008

ENTRADAS RECIENTES

Cómo automatizar backups en PROXMOX y copiarlos desde tu raspberry al NAS

by VicHaunter on 15 de octubre de 2019

El UMIDIGI F2 lo parte con descuentazos en su promoción inicial del 8 aniversario [patrocinado]

by VicHaunter on 14 de octubre de 2019

UMIDIGI F2 precio desvelado por solo 179.99$! Podrá con el Xiaomi Redmi Note 8 Pro? [patrocinado]

by VicHaunter on 10 de octubre de 2019

Review a fondo del difusor de aromaterapia Xiaomi HL Happy Life

by VicHaunter on 5 de octubre de 2019

Review a fondo de la lámpara BlitzWolf® BW-LT19

by VicHaunter on 30 de septiembre de 2019