JOINS en MySQL

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 de usuarios, y otra tabla juegos que contiene también el id del usuario al que pertenece el juego:

Tabla Usuarios:

ID	username
1	Juan

Tabla Juegos:

ID	juegoname	ID_user

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

El problema es que el modelo anterior no está optimizado.

¿Qué son las tablas relacionales o de relación?

El ejemplo de arriba es básico, y además no está optimizado, ya que en temas de rendimiento o escalabilidad es bastante limitado. Si hay más de un usuario que tiene el mismo juego, tendremos que repetir ID y juegoname por cada usuario.
Las tablas relacionales, son tablas que se utilizan como intermediarios de otras dos. Normalmente contienen solamente el id de cada uno de los elementos de otras tablas a asociar, y será con lo que trabajaremos para entender los JOINS.
Este modelo, casi siempre te lo encontrarás de esta manera cuando trabajes con bases de datos MySQL.
Usando el modelo anterior necesitamos tener 3 tablas, usuarios, juegos y juegousuario (para que el modelo esté Normalizado).

Tabla usuarios:

ID	username

1	Juan

Tabla juegos:

ID	juegoname

Tabla juegousuario:

ID_JUEGO	ID_USUARIO

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 de acuerdo a los parámetros que le hayamos pedido.

Datos de ejemplo

A fin de que puedas probar, dejo 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;
----------------------------------------
--- Estructura de la tabla 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;
-- ----------------------------
-- Registros para 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');


-- ----------------------------
-- Estructura de la tabla 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;
-- ----------------------------
-- Registros para 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');
-- ----------------------------
-- Estructura de la tabla 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', 'rodrigo');
INSERT INTO `usuarios` VALUES ('2', 'pepito');
INSERT INTO `usuarios` VALUES ('3', 'jaimito');
INSERT INTO `usuarios` VALUES ('4', 'ataulfo');
```

SET FOREIGN_KEY_CHECKS=1;

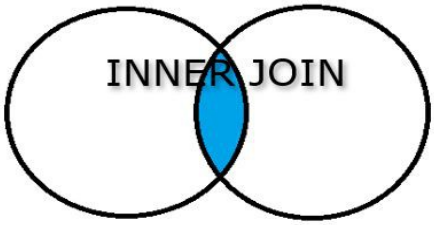
Estamos ignorando en este modelo de ejemplo, todos los parámetros de rendimiento y demás porque no es de lo que se trata el tema. Usamos un unique index con ambas claves incluidas, para que no se repitan registros en la tabla relacional.

¿Qué tipos de join existen?

Tenemos varios tipos de JOINS, los vamos a ver uno por uno y cada uno de ellos 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 variastablas, y te devuelve únicamente los datos que estén disponibles en todas las tablas a la vez.



Esto significa, que si 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:

```
SELECT usuarios.username,
       juegos.juegoname
  FROM usuarios INNER JOIN juegousuario ON usuarios.ID  = juegousuario.ID_usuario
                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 une la tabla usuarios, con la tabla juegousuario.
Una vez unida, usamos los datos de la tabla juegousuario, y lanzamos la consulta con la tabla juegos con otro JOIN.

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

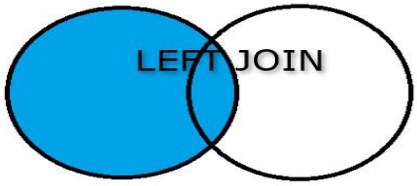
USERNAME	JUEGONAME
rodrigo	Final Fantasy VII
rodrigo	Zelda: A link to the past
rodrigo	Crazy Taxy

rodrigo	Donkey Kong Country
rodrigo	Saints Row III
rodrigo	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 jaimito no tiene juegos, y el juego no lo tiene 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 uno, 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:

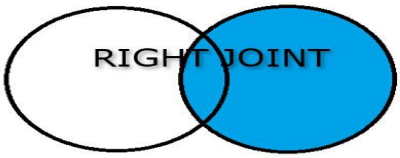
USERNAME	JUEGONAME
rodrigo	Final Fantasy VII
rodrigo	Zelda: A link to the past
rodrigo	Crazy Taxy
rodrigo	Donkey Kong Country
rodrigo	Saints Row III
rodrigo	La taba
pepito	Crazy Taxy

pepito	La taba
jaimito	
ataulfo	Final Fantasy VII
ataulfo	Zelda: A link to the past
ataulfo	Donkey Kong Country
ataulfo	La taba

Ahora sí 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.



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;
```

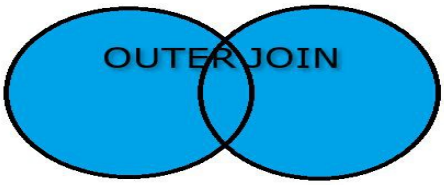
Con inesperados resultados:

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

Como puede observar, ya no hay ni rastro de jaimito, pero sin embargo aparece Fallout 4, pero no está asociado a ningún usuario.
Esto realmente tiene muchas aplicaciones interesantes, es cuestión de tener que resolver problemas reales.

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;
```

Para poder usar UNION, todos los SELECT deben tener los mismos campos seleccionados. En este caso username y juegoname, ya que si no, no funcionará.

Esta consulta devolvería:

USERNAME	JUEGONAME
rodrigo	Final Fantasy VII
rodrigo	Zelda: A link to the past
rodrigo	Crazy Taxy
rodrigo	Donkey Kong Country
rodrigo	Saints Row III
rodrigo	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 en forma independiente, por ejemplo. Así con una única consulta a la base de datos tendrías todo lo que necesitas.

Entendiendo los JOINS

Quizás puedas 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?
Hemos visto hasta ahora los tipos de joins que hay, y los datos que devuelve cada uno. Veamos ahora cómo funcionan.

Explicación del código de un join

Comencemos por responder a la pregunta de ¿cómo salen los datos de las tablas para su uso?
Partamos de una de las primeras consultas:

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

Veamos, qué hace cada parte de la consulta:

SELECT usuarios.username,
juegos.juegoname

Especificamos aquí que quieres seleccionar datos (un SELECT), en el que indicas los campos que quieres que devuelva, en el formato nombretabla.nombrecampo.
Le indicamos de qué tabla tomar el dato, y cómo se llama el campo, ya que por ejemplo la columna ID se llama igual en las dos tablas. (si tuviéramos que usarlo)

FROM usuarios

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

INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario

El INNER JOIN es el tipo de join hacemos, puede ser INNER, LEFT o RIGHT.

Lo siguiente será indicarle sobre qué tabla queremos que tome 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 el 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 disponibles en el primer join para toda la consulta, podemos usarlos aquí o en cláusulas where, etc.

Esto es lo que sucede:

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

El resultado es una lista de los campos que hayamos elegido, relacionando usuarios y juegos.

¿Puedo hacer joins con más de tres tablas?

¡Por supuesto! Puedes hacer joins con las tablas que quieras, siempre que el servidor aguante la carga.
A mayor cantidad de tablas en el join, y más datos tengan éstas, más lenta será la consulta, tanto, que puede llegar a saturar al servidor.

Un ejemplo de consulta con más joins, sería si por ejemplo le añadimos categorías a los juegos y mostrar un la lista de usuario – juego - categoría.
Supongamos que contamos con una columna ID_categoría en la tabla juegos, y la tabla categoria.

```
SELECT  usuarios.username,
        juegos.juegoname,
        categoria.catname
 FROM  usuarios   INNER JOIN juegousuario ON usuarios.ID = juegousuario.ID_usuario
                  INNER JOIN juegos ON juegousuario.ID_juego = juegos.ID
                  INNER JOIN categorias ON juegos.ID_categoria = categorias.ID;
```

Con solo añadir una línea, y el campo en la lista de select hemos hecho que nos devuelva también el nombre de categoría para esta consulta.

¿Y un join con solo dos tablas?

También se puede hacer, dejo este ejemplo. En ese caso la consulta se uniría directamente con la ID del usuario y el campo ID_usuario de la tabla juegos.

```
SELECT usuarios.username,
       juegos.juegoname
 FROM  usuarios  INNER JOIN juegos ON usuarios.ID = juegos.ID_usuario;
```

En este caso obtenemos los nombres de los usuarios, y nombre 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 se podría asignar varios juegos a un solo usuario a no ser que tuvieras más campos en la tabla, o lo procesaras luego en php por ejemplo.

Entonces, ¿Cómo uso los joins?

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 las usadas en los ejemplos, dan mucha versatilidad. Permiten que la tabla relacional sirva para cualquiera de los 3 casos, ya que las limitaciones se ponen en el código, y una asociación nueva será tan simple como añadir un campo con las ids.

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

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:

```
SELECT *
 FROM  contrataciones.empleado e INNER JOIN persona.contacto c ON c.ID_contacto = e.ID_contacto
                                 LEFT JOIN contrataciones.candidato dc ON dc.ID_empleado = e.ID_empleado
                                 INNER JOIN ventas.vendedor vv ON vv.ID_vendedor = e.ID_empleado
                                 LEFT JOIN ventas.orden vo ON vo.ID_personaventas = vv.ID_vendedor
                                 LEFT JOIN ventas.zona vz ON vz.ID_zona = vv.ID_zona;
```

A simple vista parece complejo, pero si analizamos detenidamente podrá verse su lógica. Como se observa 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.

Puede que parezca un poco confuso, pero si juegas un poco con ellos, 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.

Recuerda 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.