

Realización de consultas.

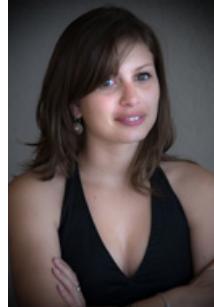
Caso práctico

En la unidad anterior **Noiba y Vindio han dejado lista la base de datos que le encargaron en Talleres Faber, sobre un taller mecánico, para empezar a introducir la información en las tablas**, de forma que se pueda acceder a los datos como si de una sola tabla se tratara. En este momento ha acabado la primera parte de su proyecto y quieren mostrar el resultado.

La empresa ha crecido mucho desde que empezó su actividad como un pequeño taller mecánico que reparaba los vehículos de los vecinos del barrio.

Actualmente tiene varios empleados y el número de clientes es cada vez mayor.

Los socios de la empresa **están deseando ver cómo el trabajo de Noiba y Vindio simplifican las tareas de gestión y organización**, y quieren comprobar que ofrece todas las prestaciones adicionales que habían prometido.



[Alain Bachellier \(CC BY-NC-SA\)](#)

Una de las cosas más importantes que ofrece una base de datos es la opción de poder consultar los datos que guarda, por eso **Noiba y Vindio** van a intentar sacar el máximo partido a las tablas que han guardado y sobre ellas van a obtener toda aquella información que su cliente les ha solicitado. Sabemos que dependiendo de quién consulte la base de datos, se debe ofrecer un tipo de información u otra. Es por esto que deben crear distintas consultas y vistas.

En esta unidad **aprenderemos a extraer la información de una base de datos** estableciendo los requisitos y criterios para seleccionar los datos.

Por lo que has estudiado en unidades anteriores, **conoces las ventajas que ofrece un SGBD**, pero también sabes que si no conocemos la forma de relacionarnos con la base de datos, la información puede estar “disponible” pero no podremos acceder a ella.

El SGBD va a ser capaz, a partir de las instrucciones, de hacer **consultas enlazando las tablas mediante las relaciones definidas en el diseño y en la creación**. Por eso es tan importante que todo se haya hecho respetando los **criterios establecidos por el modelo**, tanto en la fase de diseño lógico como físico. De no hacerlo así nos podríamos encontrar con que, una vez realizado todo el trabajo y después de registrar los datos en las tablas, no tenemos acceso a lo que buscamos o que la información devuelta por las consultas resulta incoherente.

El siguiente paso sería introducir los datos en las tablas, pero para facilitar la comprensión del proceso vamos a partir de una base de datos ya creada y que contenga información. Explicaremos a continuación como hacer consultas en ella, tanto desde herramientas gráficas como utilizando el lenguaje **SQL**.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

1.- Herramientas para la realización de consultas.

Caso práctico

Noiba y Vindio saben que para conseguir el acceso a los datos es muy importante conocer bien:

- ✓ El lenguaje que nos permita construir las consultas.
- ✓ Las herramientas gráficas de que dispone el SGBD para ayudarnos.

Pues bien, vamos a revisar con ella **las herramientas gráficas que se utilizan con algunos SGBD** muy conocidos como: asistentes para realizar consultas, entornos para introducir consultas en modo diseño, editores para crear nuestras sentencias en SQL etc.



Alain Bachellier (CC BY-NC-SA)

Como sabemos **un SGBD integra**, además de la base de datos, **un conjunto de herramientas que nos facilitan el diseño, la gestión y la administración de la base de datos**.

En unidades anteriores hemos visto cómo algunas de esas herramientas nos pueden ayudar a establecer tanto el diseño lógico como el diseño físico de la base de datos. En el caso de las consultas también contamos cada vez con un mayor número de utilidades.

Podemos hacer una clasificación de estas herramientas en dos tipos:

- ✓ **Herramientas proporcionadas por el gestor:** algunos gestores proporcionan un entorno integrado donde se incluyen asistentes, editores gráficos, etc. En esta categoría nos encontramos con Microsoft Access, OpenOffice Base, Workbench o phpMyAdmin. Las dos últimas se utilizan, exclusivamente, para interactuar con servidores MySQL.
- ✓ **Herramientas externas al gestor:** en este caso hablamos de herramientas que añaden funcionalidades y son compatibles con distintas bases de datos. Navicat y HeidiSQL pertenecen a este tipo de herramientas.

A continuación veremos algunos ejemplos de los distintos tipos de herramientas que hemos citado, teniendo en cuenta que se trata de un pequeño ejemplo de las numerosas herramientas presentes en el mercado, tanto de software libre como propietario.

Autoevaluación

De las siguientes aplicaciones ¿alguna es una herramienta gráfica para realizar consultas en MySQL? Selecciona la opción correcta.

- MySQL Workbench.
- PhpMyAdmin.
- Todas ellas.
- Ninguna.

Incorrecto, revisa el resto de opciones.

Incorrecto, no es la respuesta acertada.

Correcto, todas estas herramientas tienen un interfaz gráfico para consultas con MySQL.

No es la respuesta acertada.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

1.1.- Asistentes y herramientas gráficas de diseño.

Tanto incluidos en el propio SGBD, como cuando se trata de aplicaciones externas al propio gestor, existen herramientas gráficas que nos ayudan a realizar consultas a una o más tablas de una base de datos de una forma fácil y sencilla, incluso sin necesidad de conocer el lenguaje SQL. Podemos clasificar esas herramientas en:

- ✓ Asistentes para consultas.
- ✓ Herramientas para el diseño de consultas.
- ✓ Herramientas de edición de SQL.



Everaldo Coelho and YellowIcon (GNU/GPL)

Creación de consultas desde un asistente para consultas.

Estos asistentes son aplicaciones que nos permiten seleccionar las tablas, los campos, el orden de las filas, establecer criterios de selección, grupos y cálculos, etc., siguiendo unos pasos muy sencillos.

Ejemplos de estos asistentes los podemos ver **OpenOffice Base** y en **Microsoft Access**, aunque también incorporan este tipo de asistentes bases de datos como Oracle.

Creación de consultas desde la ventana de diseño de consultas.

Tanto en los propios SGBD como en aplicaciones que se pueden instalar para aumentar las prestaciones y que son externas al SGBD, podemos encontrar herramientas gráficas para diseñar consultas mediante opciones como:

- ✓ Representaciones gráficas de tablas y campos.
- ✓ Opciones con listas desplegables para elegir los tipos de datos, los tipos de índices, etc.
- ✓ Posibilidad de establecer los criterios de selección, de ordenación, etc., seleccionando opciones o eligiendo operadores, etc.

Se trata de herramientas que permiten al usuario elaborar consultas sencillas de forma intuitiva, con el simple manejo de recursos gráficos presentados en una ventana donde se encuentran disponibles todos los elementos que representan las distintas cláusulas que una consulta puede recoger. Aplicaciones como **Workbench**, **phpMyAdmin** o **Navicat** incorporan este tipo de herramientas.

Recomendación

Para ver un ejemplo de cómo crear consultas mediante un asistente te recomendamos que visualices el siguiente vídeo dónde se explica cómo usar el asistente para crear consultas en Access. Pulsa sobre el siguiente enlace para visualizarlo.

[Creación de consultas mediante el asistente de Access](#)

Para saber más

Algunas herramientas que permiten trabajar con diferentes SGBD, entre ellos MySQL, son **Navicat** y **SQLMaestro**.

Es interesante que accedas a su páginas oficiales y que navegues por ellas para comprobar qué aplicaciones ofrecen para la gestión de servidores de bases de datos. Los enlaces son los siguientes:

[Página oficial de Navicat](#)

[Página oficial de SQLMaestro](#)

En ambos casos puedes descargar la aplicación para MySQL (son versiones de prueba de un determinado número de días). Las puedes instalar en tu ordenador, ejecutarlas, y recorrer las principales opciones. Más adelante las usaremos en algunos ejercicios.

También es interesante que te descargues el manual oficial de esas aplicaciones, en formato [pdf](#) (está en inglés).

1.2.- Editores de SQL.

Aunque hemos citado una serie de herramientas que nos pueden facilitar el diseño de consultas, **la herramienta más potente** a la hora de establecer criterios para seleccionar la información almacenada en una base de datos es el conocimiento del lenguaje SQL. Todos los SGBD tienen un editor que, con más o menos elementos de ayuda, nos permite introducir nuestras instrucciones y nos muestra dónde se producen los posibles errores.



[Everaldo Coelho and YellowIcon \(GNU/GPL\)](#)

Veamos un **ejemplo con Workbench**.

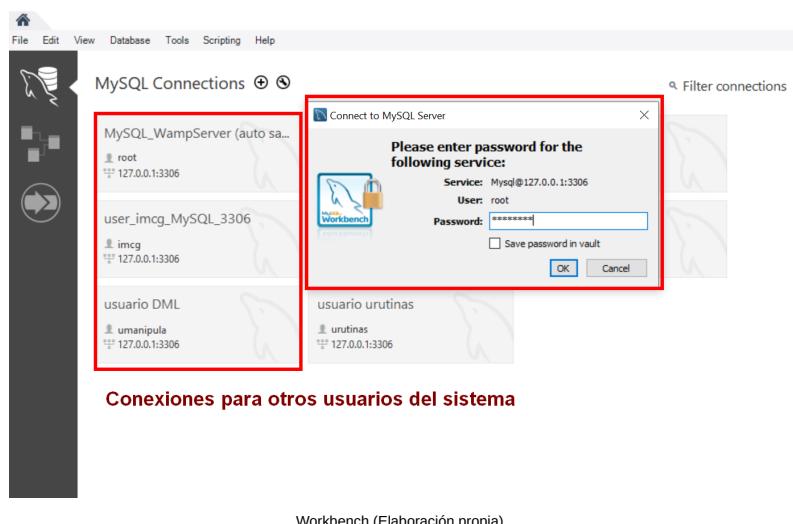
◀ 1 2 3 4 ▶

Conexión con Workbench al servidor MySQL

Nos conectamos al servidor MySQL mediante la conexión de Workbench que se ha creado para el usuario root, poniendo la contraseña que dimos en la instalación de MySQL.

Lo normal es que existan diferentes usuarios en el SGBD, cada uno de ellos con ciertos permisos o privilegios sobre las diferentes bases de datos instaladas en el Servidor. Para esos otros usuarios aparecen conexiones en la pantalla inicial de Workbench.

De momento, en vuestro caso, tendréis sólo una conexión para el usuario root.



Workbench (Elaboración propia)

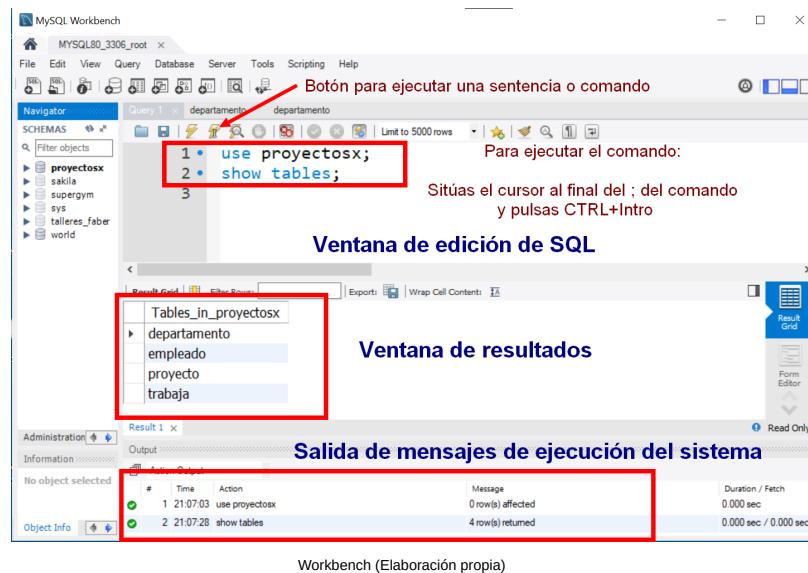
En el Editor de SQL activamos base de datos y mostramos sus tablas

Observa que en la ventana de edición de SQL se escriben las sentencias.

Para ejecutar una sentencia, hacemos los siguiente:

- ✓ Situar el cursor sobre esa sentencia o al final del ; (punto y coma) y pulsar las teclas Ctrl+ Intro, o bien pulsar sobre el botón superior del rayo indicado en la imagen.

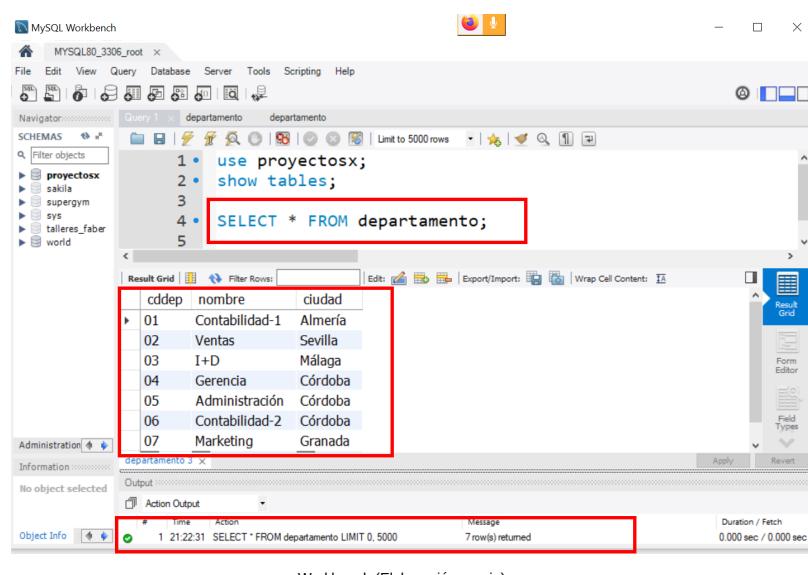
Otras secciones son la **Ventana con los resultados** de ejecutar las entencias y la **ventana en la que el sistema informa** del éxito de ejecución o los errores encontrados. En este ejemplo, no hay errores en la ejecución.



En el editor de SQL escribimos una consulta, sin errores

La consulta `SELECT * FROM departamento;` muestra todos los datos existentes en la tabla `departamento`.

Observa que se ejecuta sin errores, y se muestran todos los datos de los departamentos.



En el Editor SQL escribimos una consulta, con errores

La consulta SELECT * FROM departamentos;

Observa que se ejecuta con errores y el sistema avisa de ello. El error es, que el nombre de la tabla no es correcto, es departamento sin s al final.

The screenshot shows the MySQL Workbench interface. In the 'Query' tab, there is a list of SQL statements:

- 1 • use proyectosx;
- 2 • show tables;
- 3
- 4 • **SELECT * FROM departamentos;**
- 5

The fourth statement, which contains the error, is highlighted with a red box. Below the query window, the 'Output' pane displays an error message:

El sistema nos informa de un error

#	Time	Action	Message	Duration / Fetch
1	21:37:31	SELECT * FROM departamentos LIMIT 0, 5000	Error Code: 1146. Table 'proyectosx.departamentos' doesn't exist	0.000 sec

A red box also surrounds the entire 'Output' pane. At the bottom of the window, it says 'Workbench (Elaboración propia)'

2.- El lenguaje de manipulación de datos: LMD.

Caso práctico

En este punto **Noiba** se dispone a trabajar con el lenguaje de manipulación de datos (**LMD**), y dentro de él comenzará por las instrucciones que le van a permitir hacer consultas a los datos contenidos en Talleres Faber.

Conociendo esas instrucciones, con consultas fáciles de escribir, podrá saber cosas tan importantes como:

- ✓ El importe facturado este mes.
- ✓ Las reparaciones en las que interviene cada mecánico.
- ✓ Qué tipo de avería es la más común en el taller.
- ✓ Cuanto tiempo, por término medio, lleva una reparación, etc.



Alain Bachellier (CC BY-NC-SA)

En la unidad anterior clasificamos las sentencias del lenguaje SQL en 4 tipos ¿recuerdas?

Tipos de sentencias SQL

Lenguaje SQL	Sentencias
<u>LDD</u> (Lenguaje de Definición de Datos)	Incluye sentencias para gestionar las. estructuras
<u>LMD</u> (Lenguaje de Manipulación de Datos)	Incluye sentencias para gestionar los datos .
<u>LCD</u> (Lenguaje de Control de Datos)	Incluye sentencias para gestionar la seguridad y los permisos .
<u>LTC</u> (Lenguaje de Control de Transacciones)	Incluye sentencias para gestionar transacciones

Pues bien, ahora vamos a tratar con el LMD o DML que es una parte del lenguaje SQL que se encarga de la gestión de los datos almacenados. El LMD tiene instrucciones para ayudarnos a:

- ✓ Insertar filas (INSERT).
- ✓ Consultar los datos almacenados (SELECT).
- ✓ Modificar esos datos (UPDATE).
- ✓ Borrar filas (DELETE).

Sin ninguna duda, la operación que más veces se realiza sobre una base de datos es la consulta.

Reflexiona

Tanto en el LDD, como en el LMD existen sentencias que nos permiten *Modificar* o *Borrar*. La diferencia está en que cuando tratamos con el *Lenguaje de definición de datos*, las instrucciones de modificación y/o borrado afectan a la estructura de la base de datos: tablas, columnas, índices, etc.; es decir, a los objetos que la componen.

Cuando utilizamos instrucciones correspondientes al *Lenguaje de manipulación de datos*, los cambios que hagamos afectarán a los datos contenidos en la base de datos: las filas de las tablas.

De hecho, los nombres de las sentencias que se usan para crear, modificar y borrar elementos en ambos sublenguajes son diferentes.

Autoevaluación

Siguiendo la reflexión del párrafo anterior, de la siguiente lista, ¿Cuál crees que sería la diferencia entre la instrucción **DROP** que ya conoces y que pertenece al LLD y la instrucción **DELETE** que pertenece al LMD?

- Con ambas instrucciones se pueden borrar tablas, según los criterios que se especifiquen.
- La instrucción **DELETE** sirve para borrar filas, **DROP** borra tablas completas.
- DROP** sirve para borrar columnas y **DELETE** para borrar filas.

No. Con **DELETE** no podemos borrar objetos, solo datos.

Correcto. Con **DROP** borramos tablas, índices, etc., pero no los datos contenidos en las filas.

No. Con **DROP** no podemos borrar columnas, recuerda que la sentencia era **ALTER TABLE... DROP**.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto

3.- Sentencias para la consulta: SELECT.

Caso práctico

Como hemos visto, las herramientas gráficas nos ofrecen una gran ayuda a la hora de enviar consultas a nuestra base de datos. Pero sin duda para tener mayor control sobre cualquier SGBD, y en concreto sobre las consultas a los datos que almacena, **Juan** recomienda a **Noiba y Vindio que lo más seguro es conocer las instrucciones de SQL que integran el LMD o DML**.

Para realizar consultas **se utiliza la sentencia SELECT**. Siguiendo el criterio de **Juan**, aprenderemos a utilizar las opciones más importantes de SELECT, desde consultas sencillas a una sola tabla hasta consultas complejas combinando datos de varias tablas.



[Jonny Goldstein \(CC BY\)](#)

¿Cómo podemos seleccionar los datos que nos interesen dentro de una base de datos? Para recuperar o seleccionar los datos, de una o varias tablas puedes valerte del lenguaje SQL, para ello utilizarás la sentencia SELECT, que consta, como veremos de muchas opciones para poder construir consultas todo lo complejas que puedas imaginar. Veremos en esta unidad las opciones más importantes de la sentencia SELECT.

SELECT es la sentencia que tiene mayores variaciones y opciones sintácticas en todos los SGBD del mercado y también la más compleja y potente de todas las instrucciones SQL.

Utilizaremos esta sentencia para consultar los datos de una base de datos. En la sentencia SELECT el usuario **especifica lo que quiere obtener, no cómo, ni dónde**.

Como hemos dicho presenta muchas opciones. Las más comunes son:

```
SELECT  
[ALL | DISTINCT | DISTINCTROW ]  
{Expresión | Columna1, ... | * }  
FROM TablasReferenciadas  
[WHERE Condiciones]  
[GROUP BY {NombreColumna | Expresión | Posición} [ASC | DESC], ...]  
[HAVING Condiciones]  
[ORDER BY { NombreColumna | Expresión | Posición } [ASC | DESC] , ...]  
[LIMIT { [desplazamiento,] contador | contador OFFSET desplazamiento } ]
```

Antes de empezar a analizar la función de cada una de las cláusulas u opciones de esta sentencia vemos algunas **consideraciones generales**:

- ✓ Para realizar una consulta a una o varias tablas con esta sentencia es imprescindible utilizar al menos las palabras:
 - ◆ SELECT seguido del nombre de la columna o columnas que se quieran mostrar.
 - ◆ FROM seguido del nombre de la tabla o tablas de las que se obtienen los datos.
- ✓ El resto de las cláusulas son opcionales, pero en el caso de que se incluyan debe respetarse el orden descrito anteriormente.
- ✓ La sentencia SELECT no se limita a nombres de columnas de tablas o a un *, pueden ser otras expresiones, funciones, etc.; incluso aunque no correspondan a ninguna tabla.

Ejemplo:

```
SELECT 7*4, SIN(3.1414/2), CURRENT_DATE();
```

- ✓ También es válido, y además necesario cuando se seleccionan varias columnas con el mismo nombre en diferentes tablas, especificar el nombre de la columna separado con un punto del nombre de la tabla, de la siguiente forma:

NombreTabla.NombreColumna

Ejemplo:

```
SELECT CLIENTES.Nombre, CLIENTES.Apellidos FROM CLIENTES;
```

- ✓ Se pueden consultar datos de tablas de una base de datos que no está actualmente en uso estableciendo el nombre de la base de datos a la que pertenece separado con un punto del nombre de la tabla:

NombreBaseDeDatos.NombreTabla

Ejemplo:

```
SELECT * FROM TalleresFaber.CLIENTES;
```

Recomendación

Las cláusulas ALL y DISTINCT son opcionales.

- ✓ Si incluyes la cláusula ALL después de SELECT, indicarás que quieras seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.
- ✓ Si incluyes la cláusula DISTINCT después de SELECT, se suprimirán aquellas filas del resultado que tengan igual valor que otras.

Autoevaluación

Contesta si es verdadera o falsa la siguiente afirmación:

El orden de las cláusulas en la sentencia SELECT puede modificarse.

Verdadero Falso

Falso

Debe respetarse el orden de las cláusulas definido en la sintaxis.

Para saber más

Como hemos dicho, la sentencia SELECT admite muchas cláusulas y además éstas presentan diferencias en función del SGBD elegido. A lo largo de esta unidad iremos viendo las más utilizadas y que, en muchos casos, son comunes a un gran número de gestores. Para conocer todas las opciones que admite la sentencia SELECT en MySQL pincha en el siguiente enlace:

[Sintaxis de la sentencia SELECT](#)



3.1.- Cláusulas de SELECT I.

Como hemos visto, el **formato básico** de esta sentencia incluye como mínimo la palabra SELECT y la cláusula FROM. Empezaremos por describir su sintaxis y seguiremos con el resto de las cláusulas que más se utilizan.



Everaldo Coelho and
YellowIcon (GNU/GPL)

Formato básico de una consulta.

SELECT

A continuación de SELECT se especifican las columnas que se desea mostrar. Podemos referenciarlas:

- ✓ **Con un ***: Indica que se consultan todas las columnas de la tabla.
- ✓ **Con una lista de nombres de columnas** separadas por comas.

FROM

Indica la tabla o tablas que se van a consultar, escribiendo sus nombres separados por comas si se trata de más de una tabla.

Ejemplo: Mostrar el código, nombre y apellidos de todos los clientes

```
SELECT CodCliente, Apellidos, Nombre FROM CLIENTES;
```

Alias de columnas.

El encabezado de las columnas que aparece en las consultas, es el nombre de la columna o campo correspondiente. Si éste resulta demasiado largo o es el resultado de un cálculo, se puede cambiar usando un alias de columna. Es posible asignar un alias a cualquiera de las expresiones SELECT.

Para ello, se escribe tras el nombre de columna la cláusula AS seguida del nuevo nombre que va a aparecer en el encabezado entre comillas, aunque esta palabra es opcional.

El alias puede usarse dentro de toda la consulta en cualquier referencia a la columna correspondiente, por ejemplo, dentro de las cláusulas **WHERE**, **HAVING** o **GROUP BY** que veremos a continuación.

Ejemplo:

```
SELECT CodCliente AS "Código de cliente", Apellidos, Nombre FROM CLIENTES;
```

Alias de tablas.

Un alias se puede utilizar también para **asociar un nuevo nombre a una tabla**. Se trata de un nombre, generalmente más corto, que se utiliza sobre todo cuando consultamos varias tablas y hay nombres de columnas que coinciden.

Ejemplo:

```
SELECT E.Apellidos, E.Nombre, C.Apellidos, C.Nombre FROM EMPLEADOS E, CLIENTES C WHERE..... ;
```

(Hemos asignado el alias E a la tabla EMPLEADOS y el alias C a la tabla CLIENTES por eso los nombres de las columnas van precedidos del alias E ó c seguido de un punto).

Columnas calculadas.

Una consulta en SQL puede incluir columnas cuyos valores se obtienen haciendo cálculos a partir de los datos almacenados en las columnas de las tablas. Cuando queremos mostrar el resultado de la columna calculada dentro de SELECT es conveniente utilizar un **alias de columna** para evitar que se incluya como cabecera la propia función u operación realizada.

Ejemplo:

```
SELECT IdReparación, (FechaSalida – FechaEntrada) AS "Días empleados" FROM REPARACIONES;
```

Poniendo un filtro a la consulta.

WHERE

Permite obtener los datos de las filas que cumplen con la condición expresada. Tras WHERE se escribe una expresión de tipo relacional o de comparación que usará alguno de los operadores relativos y lógicos que veremos después.

El formato de la condición es: expresión operador expresión

Pueden construirse **condiciones múltiples** con los operadores lógicos AND, OR ó NOT.

En una cláusula WHERE se puede usar cualquier función disponible en MySQL, excluyendo sólo las de resumen o reunión. Estas funciones están diseñadas específicamente para usarse en cláusulas GROUP BY y se verán en otro apartado.

Ejemplo: Mostrar la matrícula de los vehículos cuyo color sea azul y su marca sea 'SEAT';

```
SELECT Matricula FROM VEHICULOS WHERE color="azul" OR marca="SEAT";
```

A lo largo de esta unidad realizaremos numerosos ejercicios para aprender a manejar el lenguaje SQL. Como base de la mayoría de los ejercicios usaremos la **base de datos** denominada **CAMPEONATO**, que tendrás que desplegar en tu servidor MySQL. Te indicamos a continuación el modelo y script SQL de la base de datos y la forma de instalarla en tu servidor MySQL.

Como cliente de MySQL, puedes usar la línea de comandos o cualquier cliente gráfico, como por ejemplo Workbench.

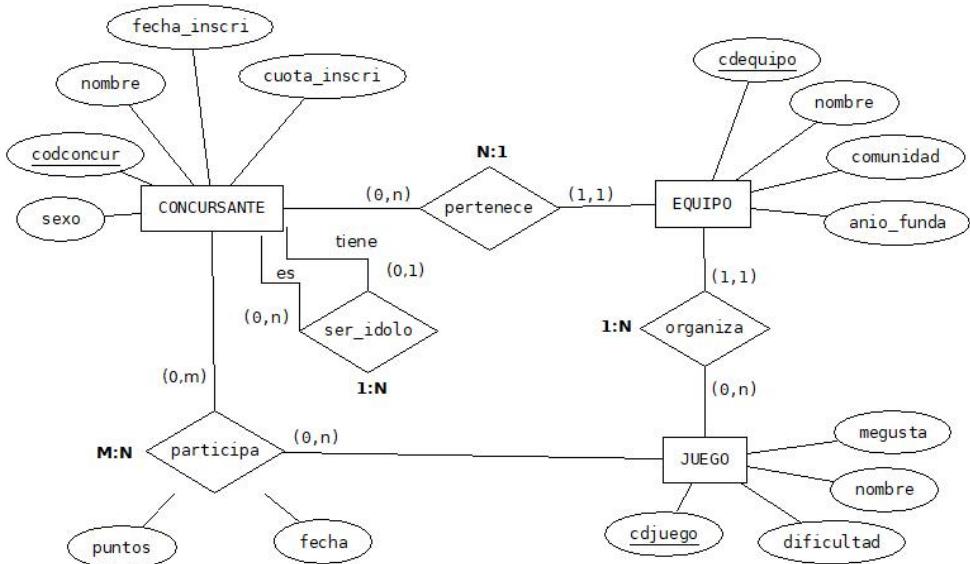
A continuación vemos:

- ✓ El modelo la base de datos de ejemplos, denominada campeonato.
- ✓ Cómo desplegarla en nuestro servidor MySQL y lanzar consultas desde el cliente Workbench.

◀ 1 2 3 4 5 6 ▶

Modelo ER de la base de datos CAMPEONATO

El modelo ER de la base de datos campeonato es el siguiente:



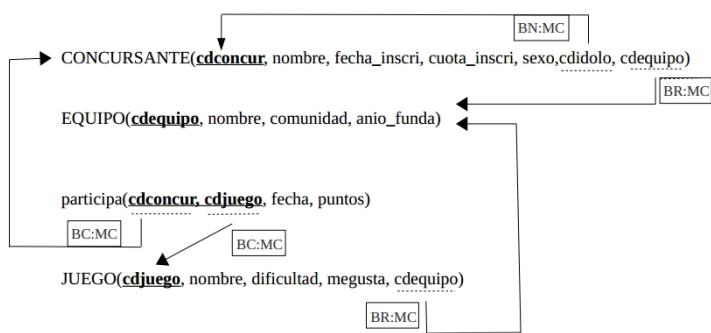
Libre Office Writer (Elaboración propia)

Modelo Relacional de la base de datos CAMPEONATO

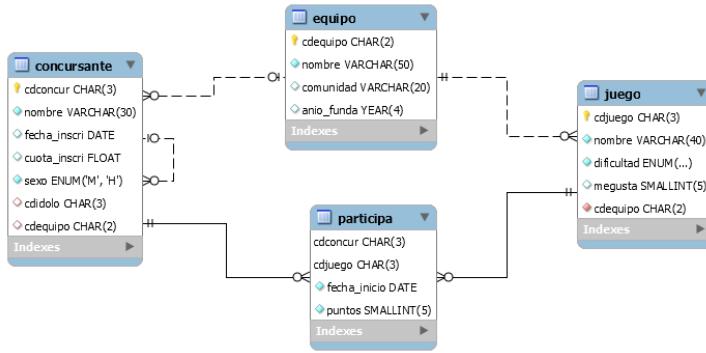
El modelo relacional correspondiente a la base de datos campeonato es el siguiente: te mostramos el **modelo textual** a la izquierda y de **forma gráfica** a la derecha (elaborado con la herramienta de modelado de Workbench).

En el modelo textual se ha indicado la restricción de cada clave ajena o foránea en cuanto a los borrados y las modificaciones:

- ✓ BR: borrado restrictivo, BN:borrado con puesta a nulos, BC: borrado en cascada.
- ✓ MC: modificación en cascada.



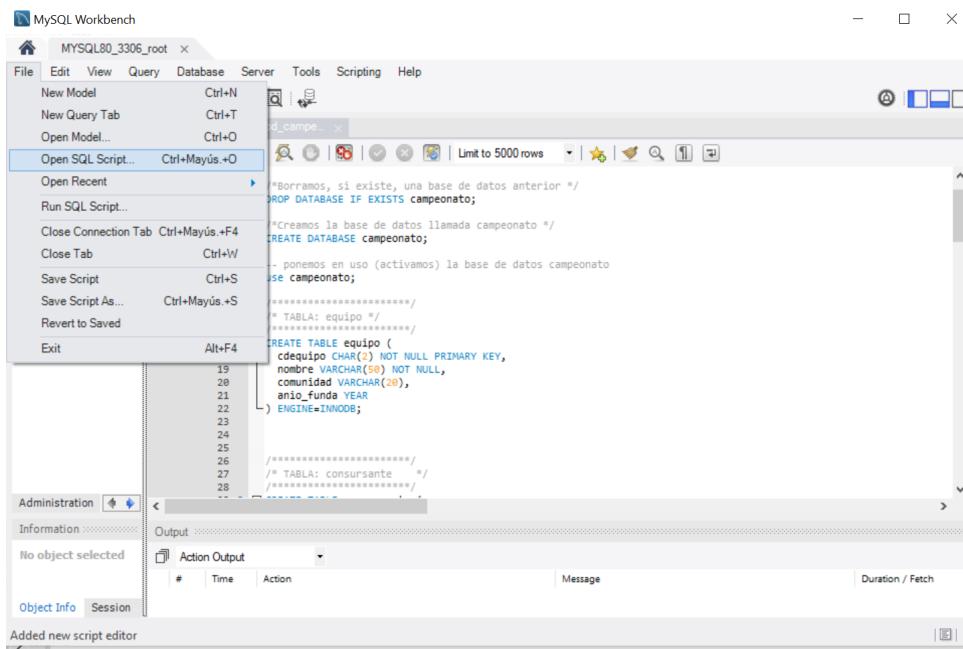
Libre Office Writer (Elaboración propia)



Workbench (Elaboración propia)

Carga o despliega la base de datos en MySQL

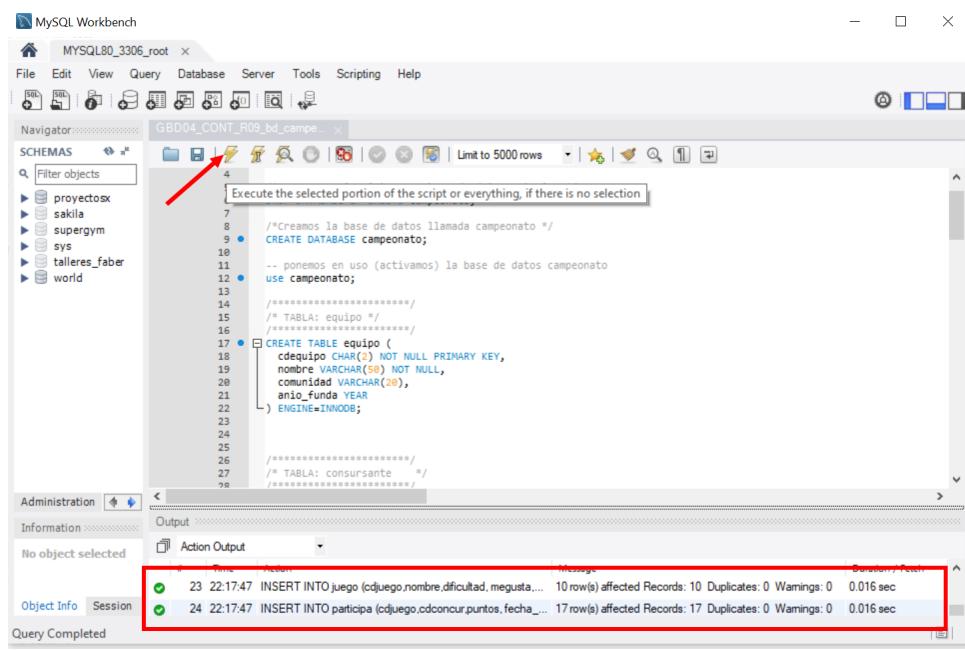
- 1.- **Descarga** desde el siguiente enlace el **script de creación de la base de datos campeonato**: [script de creación base de datos campeonato](#) (zip - 1,80 KB)
- 2.- Una vez descargado, lo debes **descomprimir** para acceder a la **fichero** de extensión .sql
- 3.- **Inicia sesión o conéctate al servidor MySQL** con el cliente Workbench. Para ello, asegúrate de tener iniciado o puesto en marcha el servidor de bases de datos MySQL e inicia sesión en como el usuario root, indicando su contraseña.
- 4.- **Carga en Workbench** el **script** que descargaste en el apartado anterior, para ello ve al menú **File/Open SQL Script...** y busca en tus carpetas locales el archivo que descargaste. Una vez localizado, lo seleccionas y pulsas en Abrir. Aparecerá el script en pantalla.



Workbench (Elaboración propia)

Ejecuta el script para implantar la base de datos CAMPEONATO

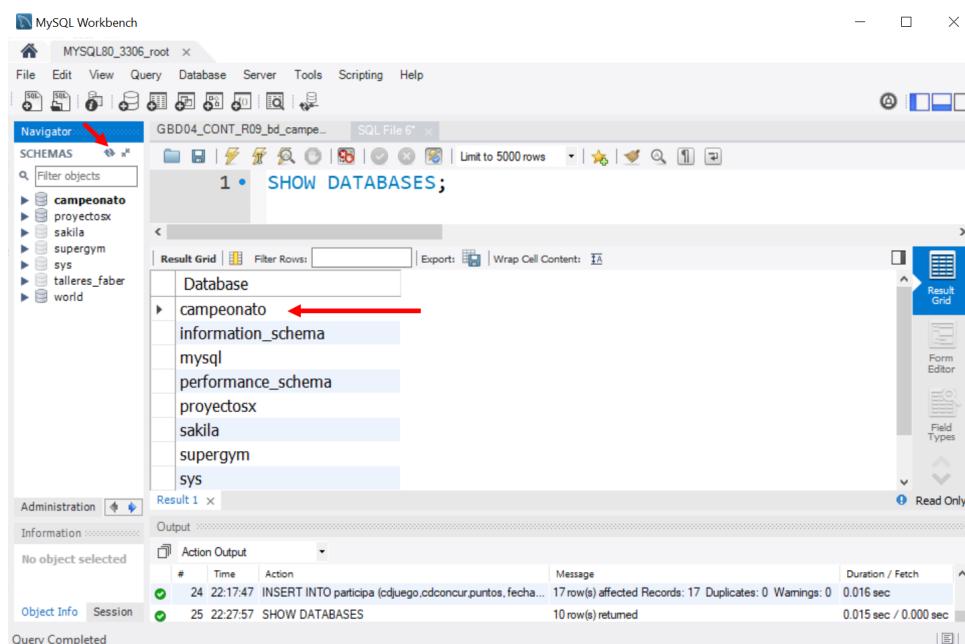
- 1.- Con en script SQL en pantalla, pulsa sobre **File/Run SQL Script** o bien sobre el botón de atajo "el rayo amarillo". Se ejecutará el script y dejará implantada la base de datos en el servidor.
- 2.- Puedes revisar las sentencias SQL de creación de la base de datos y así repasar su estructura.



Workbench (Elaboración propia)

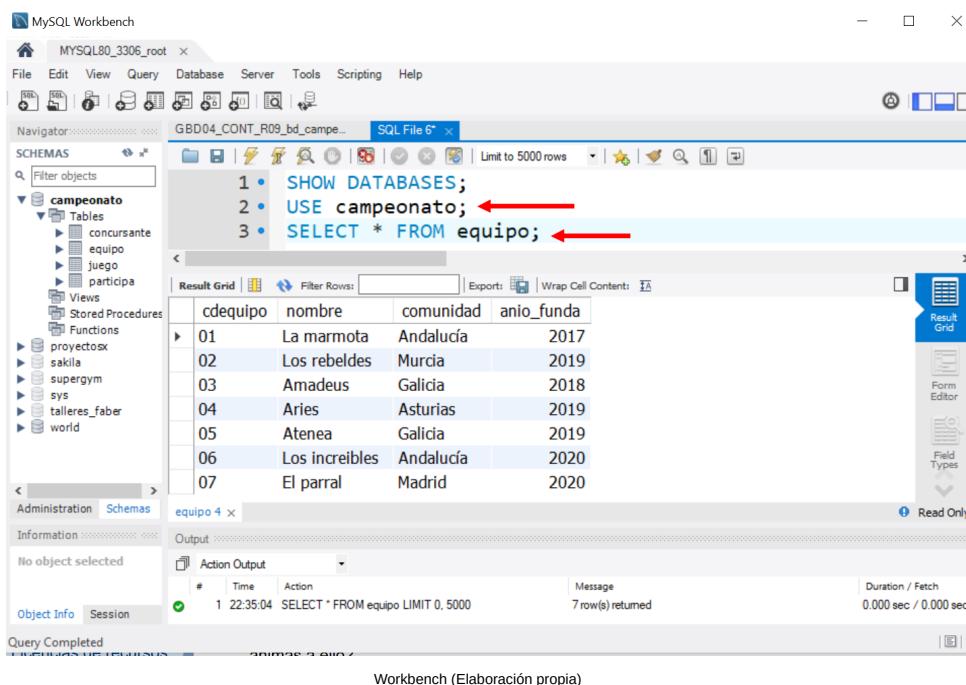
Muestra la base de datos en tu Servidor

- 1.- Actualiza el cliente gráfico pulsando sobre la doble flecha, esquina superior izquierda y verás la nueva base de datos, o bien, escribe en una nueva ventana de SQL (**File/New Query Tab**) la sentencia: SHOW DATABASES; y comprobarás que ya está la base de datos en tu servidor.



Pon en uso la base de datos y ejecuta algunas consultas

- 1.- Pon en uso la base de datos, si no lo está, ejecutando USE campeonato; (aparecerá en negrita en la sección de la izquierda).
- 2.- Puedes ver los datos de cada tabla ejecutando la sentencia SELECT que corresponda. Por ejemplo, para ver los datos de la tabla equipo, escribe en el ventana de SQL la sentencia: SELECT * FROM equipo; y ejecútala pulsando sobre le "rayo amarillo con cursor" o bien pulsando las teclas CTRL+INTRO.
- 3.- Puedes seguir viendo el contenido del resto de tablas, ejecutando la correspondiente sentencia SELECT ¿te animas a ello?



Workbench (Elaboración propia)

Recomendación

Todas las consultas de ejemplo sobre la **base de datos CAMPEONATO** y las que se piden en los ejercicios deben probarse sobre el servidor MySQL. Puedes escribir las consultas en la línea de comandos o en cualquier cliente gráfico de MySQL, como Workbench.

Es muy importante que se almacenen todas las sentencias que se realicen en los distintos ejercicios. Para ello puedes utilizar cualquier editor de texto plano.

3.2.- Cláusulas de SELECT II.

Ya has visto cómo hacer consultas sencillas sobre una base de datos. A continuación, a ver otras cláusulas que permitan ordenar los resultados y filtrar las filas que interesa seleccionar, según ciertos criterios.



Ordenando el resultado.

ORDER BY Permite obtener los datos **ordenados ascendente o descendente** con relación a una columna o a varias anidadas. Se puede elegir el orden ascendente o descendente añadiendo a continuación ASC o DESC. Por defecto se usa el orden ASC.

Ejemplo:

```
SELECT Marca, Modelo FROM VEHICULOS ORDER BY Marca ASC, Modelo ASC;
```

(Obtendremos un listado con la marca y el modelo de los vehículos que se reparan en nuestro taller, ordenado alfabéticamente por marca y, en caso de coincidencia, ordenado ascendente por modelo).

Mostrar todas las filas o eliminar repeticiones.

ALL

Permite que se puedan presentar todas las filas afectadas por el resto de condiciones de la consulta aunque algunas estén repetidas. Es la opción por defecto.

Ejemplo:

```
SELECT ALL Matricula FROM REPARACIONES;
```

(Muestra un listado de las matrículas de los vehículos que han sufrido reparaciones. Si el vehículo se ha reparado más de una vez, su matrícula aparecerá repetida).

DISTINCT y DISTINCTROW

Cuando seleccionamos algunas columnas de una o varias tablas pueden mostrarse valores de filas repetidos. Esta cláusula permite que se puedan presentar todas las filas afectadas por el resto de condiciones de la consulta pero las repetidas aparecen sólo una vez.

Ejemplo:

```
SELECT DISTINCT Matricula FROM REPARACIONES;
```

(Muestra un listado de las matrículas de los vehículos que han sufrido reparaciones. Si el vehículo se ha reparado más de una vez, así evitaremos que la matrícula aparezca repetida).

Agrupando filas.

GROUP BY

Es posible agrupar las filas que devuelve una consulta según los valores de una columna, usando la cláusula GROUP BY. Aunque en algunos casos podríamos hacer esto mismo usando la opción DISTINCT, esta cláusula es más potente y presenta las siguientes diferencias:

- ✓ La salida se ordena según la columna indicada.
- ✓ Se eliminan los valores duplicados aunque la consulta no devuelva filas duplicadas.
- ✓ Permite usar funciones de resumen o reunión. Por ejemplo: COUNT(), SUM(), MAX(), MIN(), AVG(), STD(), VARIANCE().

Estas funciones también se pueden usar sin la cláusula GROUP BY siempre que en la consulta no se seleccionen otras columnas. Estas consultas se denominan **consultas de resumen** y las veremos en esta unidad.

Esta cláusula, combinada con las funciones anteriores, permite hacer cálculos sobre una serie de filas agrupadas. A estas consultas las denominaremos **consultas agrupadas** y se verán en un apartado posterior.

Filtrando grupos.

HAVING

Permite hacer selecciones en situaciones en las que no es posible usar WHERE. La cláusula WHERE no se puede aplicar a columnas calculadas mediante las funciones de reunión que hemos mencionado en el apartado anterior. Tanto la cláusula GROUP BY como HAVING se tratarán más adelante en las **consultas agrupadas**.

Limitando las filas devueltas.

LIMIT

Permite limitar el número de filas devueltas. Se suele utilizar para no sobrecargar demasiado al servidor o a la aplicación que va a recibir los resultados de la consulta.

LIMIT puede admitir uno o dos parámetros:

- ✓ Cuando se usa un solo dígito indica el número de filas devueltas.
- ✓ Cuando se usan dos dígitos el primero indica el número de la primera fila y el segundo el número de filas a recuperar.

Ejemplos:

```
SELECT * FROM clientes LIMIT 5; -- Devuelve las 5 primeras filas.
```

```
SELECT * FROM clientes LIMIT 0,5; -- Devuelve las 5 primeras filas. (La primera fila es la 0)
```

```
SELECT * FROM clientes LIMIT 5,10; -- Devuelve a partir de fila 6, las 10 siguientes
```

Ejercicio resuelto

Con base de datos TalleresFaber

Queremos consultar la referencia, descripción e importe de las 10 actuaciones más baratas que se pueden llevar a cabo en nuestro taller.

[Mostrar retroalimentación](#)

```
SELECT Referencia, Descripcion, Importe  
FROM ACTUACIONES  
ORDER BY Importe  
LIMIT 10;
```

Con la base de datos campeonato

- 1.- Obtener el código, nombre y cuota de inscripción de cinco de los concursantes con mayor cuota.
- 2.- Lista el código, nombre y código de equipo de los concursantes del equipo '03' o '04'. Ordena el listado de forma ascendente, primero por equipo y luego por nombre.

[Mostrar retroalimentación](#)

1.-

```
SELECT cdconcur, nombre, cuota_inscri
FROM concursante
ORDER BY cuota_inscri DESC
LIMIT 5;
```

2.-

```
SELECT cdconcur, nombre, cdequipo
FROM concursante
WHERE cdequipo = '03' OR cdequipo='04'
ORDER BY cdequipo, nombre;
```

Autoevaluación

Contesta si esta afirmación es verdadera o falsa.

En la siguiente consulta:

```
SELECT Matricula, Avería
FROM REPARACIONES
WHERE FechaEntrada>"2011-01-01";
```

Es necesario añadir la cláusula DISTINCT para evitar filas duplicadas.

- Verdadero Falso

Verdadero

Cierto. Un mismo vehículo puede haber sufrido varias reparaciones por la misma avería (Ejemplo pinchazo de una rueda).

Contesta si esta afirmación es verdadera o falsa.

En la siguiente consulta:

```
SELECT cdconcur
FROM participa
WHERE fecha_inicio > '2019-03-01';
```

Es necesario añadir la cláusula DISTINCT para evitar filas duplicadas.

- Verdadero Falso

Verdadero

Cierto. Un mismo concursante puede haber iniciado su participación en varios juegos después de esa fecha.

4.- Operadores.

Caso práctico



Nate Steiner (CC0)

María, que junto con Félix son los propietarios de la empresa **BK Sistemas Informáticos**, se ha interesado por la marcha del proyecto que desarrollan **Noiba** y **Vindio**. Éstos le comentan que están preparando una batería de consultas, según requerimientos del cliente del taller mecánico. Ahora están revisando **los operadores que se pueden usar dentro de las expresiones en el SGBD** que está utilizando, en este caso MySQL, y que le ayudarán, entre otras cosas, a establecer condiciones de búsqueda básicas.

Aunque ya hemos visto en la unidad anterior algunos operadores que pueden intervenir en una sentencia **SQL**, dado que en la mayor parte de los casos se incluyen en sentencias de tipo **SELECT**, vamos a ampliar en este apartado los tipos de operadores.

Con el lenguaje **SQL** podemos utilizar **muchos operadores diferentes para cada tipo de columna**. Veremos los operadores de que dispone MySQL, pero la mayoría de ellos son compatibles con otros SGBD. Pero ¿qué es un operador?

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas o hacer comparaciones, entre otras funciones

Los operadores se utilizan para construir expresiones que se usan en cláusulas **WHERE**, **ORDER BY** y **HAVING** y además se pueden emplear directamente en las sentencias.

Debes conocer

Con frecuencia utilizaremos la sentencia **SELECT** acompañada de expresiones muy extensas y resultará difícil saber qué parte de dicha expresión se evaluará primero, por ello es conveniente conocer el orden de precedencia en MySQL.

Para consultar el orden de prioridad de los operadores disponibles en MySQL pincha en el siguiente enlace:

[Orden de prioridad de los operadores.](#)

Reflexiona

En caso de que tengas que combinar varios operadores en una misma expresión es aconsejable que utilices tantos paréntesis como consideres necesario para que éstos se evalúen en el orden que te interese. De este modo evitarás sorpresas desagradables.

Para saber más

Si quieres conocer todos los operadores que se pueden usar en MySQL consulta el siguiente enlace:

[Operadores en MySQL](#)

4.1.- Operador de asignación y Operadores aritméticos.

Sabemos que los operadores son símbolos que permiten realizar distintos tipos de operaciones. Vamos a verlos.



Everaldo Coelho and
YellowIcon (GNU/GPL)

Operador de asignación.

En MySQL podemos crear variables de sesión y usarlas posteriormente en expresiones. Podemos crear variables de dos formas:

- ✓ Con SET y un símbolo =

Ejemplo:

```
SET @hoy= CURRENT_DATE();
SELECT @hoy;
```

(Veremos la fecha actual)

- ✓ Con SET y utilizando el operador de asignación :=

Ejemplo:

```
SET @X:=10;
SELECT @X;
```

(Veremos el valor almacenado 10)

Una variable sin asignar será de tipo cadena y su valor será NULL.(ausencia de valor)

Operadores aritméticos.

Los operadores aritméticos se aplican a valores numéricos, ya sean enteros o en coma flotante. El resultado siempre es un valor numérico, entero o en coma flotante.

MySQL dispone de los operadores aritméticos habituales: **suma, resta, multiplicación, división, potencia, etc.**

Operadores aritméticos y significado

Operador	Significado
+	Se utiliza para sumar dos números y, como operador unario, para simbolizar signo positivo de un número.
-	Se utiliza para hallar la diferencia entre dos números y, como operador unario, para simbolizar signo negativo de un número.
*	Se utiliza para multiplicar dos números.
/	Se utiliza para dividir dos números.
^	Se utiliza para elevar un número a la potencia del exponente (número ^ exponente).
\	Se utiliza para dividir dos números y el resultado cociente en forma de entero (división entera) entero.
% o mod	Dividen dos números y devuelven el resto entero de la división.
-	Se obtiene un valor de signo contrario.

El orden de precedencia, entre los operadores aritméticos, es el siguiente:

- 1.- Exponenciación.
- 2.- Operadores de signo.
- 3.- Multiplicación y división.
- 4.- Suma y resta.

Ejemplo:

- ✓ El resultado de la siguiente operación: $10 + 6 / 2$ es 13 (primero se divide $6/2$ y a ese resultado se suma 10).
- ✓ El resultado de la siguiente operación: $(10+6) / 2$ es 8 (primero se resuelve el paréntesis $(10+6)$ y ese resultado se divide entre 2).

Por tanto, los paréntesis permitirán forzar el orden de evaluación de una expresión.

Autoevaluación

Calcular el resultado de la siguiente expresión: `SELECT (3*3+20)^2/5`

- 168,2
- 6,2000
- 952,2
- 3,84

Cierto has establecido correctamente el orden de las operaciones.

Falso. Revisa de nuevo la prioridad de los operadores.

Error. Revisa de nuevo la prioridad de los operadores.

Incorrecto. Revisa de nuevo la prioridad de los operadores.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

4.2.- Operadores de comparación.



Los puedes conocer con otros nombres como **relacionales**, nos permitirán comparar expresiones, que pueden ser valores concretos de campos, variables, etc.

Los operadores de comparación son símbolos que se usan como su nombre indica para comparar dos valores. Estos operadores devuelven los valores lógicos de verdadero o falso. (1 ó 0). A estas expresiones cuyo resultado de evaluación es verdadero o falso se les denomina **expresiones lógicas**

[Everaldo Coelho and YellowIcon \(GNU/GPL\)](#)

Estos operadores son habituales en cualquier lenguaje de programación, pero algunos lenguajes como SQL añade algunos muy útiles que se usan frecuentemente.

Operadores de comparación y significado

Operador	Significado
< =	Menor o igual.
<	Menor.
>	Mayor.
> =	Mayor o igual
=	Compara dos expresiones y devuelve 1, si son iguales y 0, si son diferentes.
< =>	Funciona como =, salvo que si una de las dos expresiones o las dos es nula, el resultado no es NULL. Si se comparan dos expresiones nulas, el resultado es verdadero (1).
< > ó !=	Si las expresiones comparadas son diferentes, el resultado es verdadero, y si son iguales, el resultado es falso.

Estos operadores tienen todos ellos la misma prioridad.

Comparación de cadenas de caracteres y de fechas.

Los operadores relacionales pueden usarse para comparar cadenas.

- ✓ Cuando se **comparan cadenas**, se considera menor la cadena que aparezca antes por orden alfabético no distinguiendo entre mayúsculas y minúsculas, a no ser que se realicen sobre campos definidos con el atributo BINARY.
- ✓ Las **cadenas constantes** se han de expresar encerradas entre comillas dobles o simples.
- ✓ Si son fechas, se considera menor cuánto más antigua sea.

Autoevaluación

De las siguientes consultas selecciona las que estén enunciadas correctamente:

SELECT IdReparacion, Matricula FROM REPARACIONES WHERE Km>=50000;

SELECT IdReparacion, Matricula FROM REPARACIONES WHERE FechaSalida=2011-01-04;

SELECT IdReparacion, Matricula FROM REPARACIONES WHERE Matricula= 5566 ABC;



```
□ SELECT IdReparacion, Matricula FROM REPARACIONES WHERE FechaSalida!=FechaEntrada;
```

Mostrar retroalimentación

Solución

1. Correcto
2. Incorrecto
3. Incorrecto
4. Correcto

4.3.- Operadores lógicos.



Everaldo Coelho and
YellowIcon (GNU/GPL)

Habrá ocasiones en las que tengas que evaluar más de una expresión y necesites verificar que se cumple una única condición, otras veces comprobar si se cumple una u otra o ninguna de ellas. Para poder hacer esto utilizaremos los operadores lógicos.

Los operadores lógicos se usan para crear expresiones lógicas complejas.

Solo existen dos valores posibles para los resultados: **verdadero** y **falso**, pero MySQL añade un tercer valor: **desconocido** para poder trabajar con valores NULL.

Valores lógicos

1	TRUE
0	FALSE
desconocido	NULL

Operadores lógicos y significado

Operador	Función
AND ó && 	Devuelve el valor TRUE cuando las dos condiciones son verdaderas y FALSE si alguna de ellas es falsa.
OR ó	Devuelve el valor TRUE si cualquiera de las dos condiciones es verdadera y FALSE cuando las dos condiciones son falsas
NOT ó ! 	Devuelve lo opuesto a la condición que sigue a NOT. Si el operador es NULL devuelve NULL.
XOR	Devuelve NULL si cualquiera de los operadores es NULL. Cuando uno de los operadores es verdadero, devuelve TRUE y si ambos son verdaderos o falsos devuelve FALSE.

Todos estos operadores tienen menor precedencia en las expresiones que los operadores aritméticos. Entre ellos, el orden de precedencia es el siguiente:

- 1.- Operadores de comparación.
- 2.- Operador <i>NOT.</i>
- 3.- Operador AND.
- 4.- Operador OR.

Al igual que todos los operadores binarios que veremos, estos operadores se pueden asociar, es decir, se pueden crear expresiones como

A AND B AND C.

El hecho de que se requieran dos operandos significa que las operaciones se realizan tomando los operandos dos a dos, y estas expresiones se evalúan de izquierda a derecha. Primero se evalúa A <i>AND</i> B, y el resultado, R, se usa como primer operando de la siguiente operación R <i>AND</i> C.

Ejemplo:

Si queremos obtener aquellos concursantes cuya cuota de inscripción sea menor o igual a 100€ y mayor que 20€ y del equipo '04'

```
SELECT *
FROM concursante
WHERE cuota_inscri>20 AND cuota_inscri<=100 AND cdequipo='04';
```

Ejercicio resuelto

En la base de datos TalleresFaber, consultar las matrículas de los vehículos que hayan entrado a reparar en 2011 con menos de 100000 Km y que hayan salido del taller sin reparar:

[Mostrar retroalimentación](#)

```
SELECT IdReparacion, Matricula
FROM REPARACIONES
WHERE (FechaEntrada>='2011-01-01' AND km<100000) AND Reparado=False;
```

En la base de datos campeonato obtén las siguientes consultas:

- 1.- Todos los datos de los juegos con dificultad alta o media y con 3 o más megusta.
- 2.- Nombre, comunidad y año de fundación de los equipos de Andalucía o Murcia, con año fundación posterior al 2018.

[Mostrar retroalimentación](#)

1.-

```
SELECT *
FROM juego
WHERE (dificultad='alta' OR dificultad='media') AND megusta>=3;
```

2.-

```
SELECT nombre, comunidad, anio_funda
FROM equipo
WHERE (comunidad='Andalucía' OR comunidad ='Murcia') AND anio_funda>2018;
```

4.4.- Operadores especiales.

En MySQL disponemos de varios operadores adicionales que vemos a continuación.



Everaldo Coelho (YellowIcon)
(GNU/GPL)

Comparación con patrones de cadena.

LIKE y NOT LIKE

- ✓ Se utilizan en comparación de cadenas con las construcciones WHERE y HAVING.
- ✓ Para buscar cadenas que coincidan con una cadena patrón, se usa la cláusula LIKE dentro de la cláusula WHERE y unos comodines para crear el patrón de búsqueda..
- ✓ Tanto si la cadena como el patrón son NULL, el resultado es NULL.
- ✓ LIKE no distingue entre mayúsculas y minúsculas.

La sintaxis de LIKE es:

WHERE Expresión [NOT] LIKE patrón

En SQL de MySQL tenemos dos caracteres especiales que pueden aparecer en cualquier posición del patrón:

Comodines y significado

Comodín	Significado
%	Representa a cualquier cadena de 0 o más caracteres.
_	Representa a un carácter cualquiera.

Por ejemplo:

```
SELECT Nombre, Apellidos  
FROM CLIENTES  
WHERE Apellidos LIKE "M%";
```

(Para obtener los nombres y apellidos de los clientes cuyo primer apellido comience por la letra M).

Por ejemplo:

```
SELECT Nombre, Apellidos  
FROM CLIENTES  
WHERE Apellidos NOT LIKE "%Sánchez%";
```

(Para obtener los nombres y apellidos de los clientes cuyo apellido **no sea Sánchez** en ningún caso).

Operador de rango.

BETWEEN

Sirve para comprobar si una expresión está comprendida en un determinado rango de valores. La sintaxis es:

WHERE Expresión [NOT] BETWEEN ValorInicial AND ValorFinal.

Ejemplo:

```
SELECT Referencia, Descripcion  
FROM ACTUACIONES  
WHERE TiempoEstimado BETWEEN 0.30 AND 0.60;
```

(Obtener un listado con los números de referencia y la descripción de las actuaciones cuyo tiempo estimado de realización esté entre 30 y 60 minutos).

También podríamos haber escrito:

```
SELECT Referencia, Descripcion  
FROM ACTUACIONES  
WHERE TiempoEstimado >=0.30 AND TiempoEstimado <= 0.60;
```

Reflexiona

Observa la diferencia entre las dos soluciones del ejemplo anterior:

- ✓ Con BETWEEN la expresión que se evalúa se escribe una sola vez.
- ✓ En el segundo ejemplo las expresiones a ambos lados de AND deben escribirse completas.

Operador de pertenencia a un conjunto.

IN, NOT IN

El operador IN devuelve 1 (verdadero) si el valor de la expresión es igual a alguno de los valores de la lista y falso en caso contrario. El operador NOT IN devuelve 1 (verdadero) si el valor de la expresión no está en la lista.

La sintaxis es:

```
WHERE Expresión IN [NOT IN] (Valor1, Valor2, ..... )
```

Ejemplo:

```
SELECT Matricula  
FROM VEHICULOS  
WHERE Marca IN ('Seat', 'Ford', 'Peugeot');
```

(Listado de matrículas de los vehículos que hemos registrado de las marcas: Seat, Ford o Peugeot).

También podríamos haber escrito:

```
SELECT Matricula  
FROM VEHICULOS  
WHERE Marca LIKE 'Seat' OR Marca LIKE 'Ford' OR Marca LIKE 'Peugeot';
```

Reflexiona

¿Cómo buscaríamos en una cadena el carácter % ó el _?

[Mostrar retroalimentación](#)

Para poder buscar un carácter % ó _ como carácter dentro de una cadena, éste deberá ir precedido de un carácter de escape, que es la barra invertida o contrabarra. **Ejemplo:** Queremos ver el Nombre de una serie de artículos que en la columna IVA contienen el símbolo % al final.

SELECT Nombre FROM ARTICULOS WHERE IVA LIKE '%\%';

Del carácter de escape se habló en el recurso: **Reglas para escribir sentencias SQL** de la unidad anterior.

Ejercicio Resuelto

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Todos los datos de los concursantes con fecha de inscripción entre el 1 de enero de 2019 y el 31 de diciembre de 2019, incluidos esos límites, ordenado el listado de fecha más reciente a más antigua.
- 2.- Todos los datos de los equipos cuya comunidad no es Madrid ni Murcia.
- 3.- Todos los datos de los juegos que empiezan por la letra G.

[Mostrar retroalimentación](#)

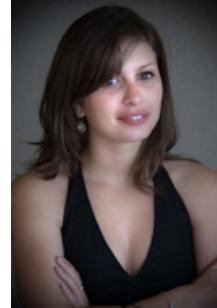
```
-- 1.-  
SELECT *  
FROM concursante  
WHERE fecha_inscri BETWEEN '2019-01-01' AND '2019-12-31';  
-- 2.-  
SELECT *  
FROM equipo  
WHERE comunidad NOT IN ('Madrid', 'Murcia');  
-- 3.-  
SELECT *  
FROM juego  
WHERE nombre LIKE 'G%';
```

5.- Tratamiento de valores nulos.

Caso práctico

En este punto **Noiba** plantea revisar el tratamiento que se hace en los **SGBD de los valores nulos**. Ella sabe que no es lo mismo que un campo contenga el valor nulo o que contenga un cero.

Conviene que nos detengamos con ella en este punto porque de no hacerlo así no podríamos diferenciar cuando el valor de un dato en una columna no ha sido introducido y por eso lo desconocemos, de cuando ese dato existe y es un cero o un espacio en blanco.



Alain Bachellier (CC BY-NC-SA)

¿Cómo indicamos que no hay valor en un determinado campo de una tabla? Lo haremos con un valor nulo.

Datos y nulos

Un dato es una unidad de información con sentido que se almacena en un campo. Dependiendo del contenido de la información almacenada, los datos pertenecen a diferentes tipos: cadenas de caracteres, fechas, o números, etc.

Un valor nulo es un valor que no es assignable a ningún tipo de datos y que significa que no se ha definido ningún valor inicial para ese dato, es decir que el valor de ese dato es **desconocido**.

Un cero o un espacio en blanco es un valor que se ha asignado a un campo, por tanto el campo tiene un contenido: el cero o el espacio en blanco.

Se dice que el contenido de una columna para un elemento es **NONE** si está completamente vacía, no se cargó en su momento ningún valor.

Este valor es compatible con todos los tipos de datos y significa ausencia de valor.

Comprobación de valor nulo.

IS NULL, IS NOT NULL

Para comprobar si en una columna hay un valor nulo se utiliza la expresión:

NombreColumna IS NULL

Ejemplo.

La siguiente consulta muestra la matrícula y la fecha de entrada de los vehículos que están actualmente en nuestro taller de reparaciones, es decir, para los que no se ha definido una fecha de salida.

```
SELECT Matrícula, FechaEntrada  
FROM REPARACIONES  
WHERE FechaSalida IS NULL;
```

Por el contrario, para comprobar si en una columna hay algún valor, se emplea la expresión:

NombreColumna IS NOT NULL

Ejemplo.

La siguiente consulta muestra la matrícula y la fecha de entrada y fecha de salida de los vehículos que ya han salido del taller, pues su fecha de salida ya tiene un valor definido.

```
SELECT Matrícula, FechaEntrada, FechaSalida  
FROM REPARACIONES  
WHERE FechaSalida IS NOT NULL;
```

Debes conocer

Los operadores aritméticos con valores nulos devuelven un valor nulo.

Autoevaluación

Marca en la siguiente lista los datos que contienen un valor null:

- El número de teléfono comodín de un cliente que no tiene teléfono.

- El número de teléfono de un cliente que no nos ha proporcionado ese dato.

- La nota de un alumno que no se ha presentado a un examen.

- La nota de un alumno que no ha solucionado correctamente ninguna pregunta.

[Mostrar retroalimentación](#)

Solución

1. Incorrecto
2. Correcto
3. Correcto
4. Incorrecto

6.- Funciones.

Caso práctico

Noiba y Vindio han reunido información sobre los distintos tipos de operadores que pueden intervenir en las consultas SQL. A menudo estos operadores intervienen en expresiones construidas también con funciones, por eso considera importante consultar la documentación relativa a las funciones que podrían ser de utilidad. Dado que MySQL dispone de **multitud de funciones** no considera necesario conocerlas todas. Siguiendo sus pasos nos conformaremos con agruparlas por tipos y ver las más comunes. Para el resto de funciones incluiremos el enlace correspondiente.



Alain Bachellier ([CC BY-NC-SA](#))

¿Has pensado en todas las operaciones que puedes realizar con los datos que guardas en una base de datos? Seguro que son muchísimas. Pues bien, en casi todos los Sistemas Gestores de Base de Datos existen funciones ya creadas que facilitan la creación de consultas más complejas. Dichas funciones varían según el SGBD, veremos aquí las que utiliza MySQL.

Las funciones son realmente operaciones que se realizan sobre los datos y que realizan un determinado cálculo. Para ello necesitan unos datos de entrada llamados parámetros o argumentos y en función de éstos, se realizará el cálculo de la función que se esté utilizando. Normalmente los parámetros se especifican entre paréntesis.

Las funciones se usan dentro de expresiones y actúan sobre los valores de las columnas de las tablas, con variables o con constantes. Se utilizan en cláusulas SELECT, WHERE y ORDER BY.

Las funciones pueden producir dos tipos de resultados:

- ✓ Modificar la información original.
- ✓ Indicar alguna información sobre el elemento al que se aplica.

Podemos clasificar las funciones en varios tipos:

Funciones aritméticas:

- ✓ Estas funciones **trabajan con datos de tipo numérico**.
- ✓ Los datos de tipo numérico incluyen los dígitos del 0 al 9, el punto decimal, y el signo menos.
- ✓ Los literales numéricos no se encierran entre comillas.

Funciones de cadena de caracteres:

- ✓ Estas funciones **trabajan con datos de tipo carácter**.
- ✓ Este tipo de datos incluyen cualquier carácter alfanumérico: letras, números y caracteres especiales.
- ✓ Se deben encerrar entre comillas simples.
- ✓ Mediante estas funciones podremos calcular el número de caracteres de una cadena, convertir una cadena a mayúsculas o a minúsculas, añadir o eliminar caracteres a la izquierda o a la derecha, etc.

Funciones de manejo de fechas y horas:

- ✓ En MySQL existen numerosas funciones para trabajar con las fechas y las horas.
- ✓ Por defecto el formato de almacenamiento de las fechas es AAAA-MM-DD y de las horas HH:MM:SS pero existen muchas funciones que nos permiten descomponer la fecha y la hora, obtener la fecha y hora actuales, etc.

Funciones de comparación:

Se trata de funciones que **comparan los valores de cada una de las columnas** en el interior de una fila para obtener: el menor o el mayor valor de ellos, así como comprobar si el valor de una columna es nulo.

Debes conocer

En el siguiente documento te presentamos un resumen de las funciones de cada tipo que se utilizan con más frecuencia en MySQL y algunos ejemplos de aplicación:

[Resumen de las funciones más frecuentes en MySQL.](#) (pdf - 262,21 KB)

Para saber más

Como hemos dicho, existen muchas funciones que pueden aplicarse a las columnas de una tabla. En algunas herramientas gráficas se encuentran accesibles todas las funciones disponibles agrupadas por categorías.

Para consultar la documentación relativa a todas las funciones ya implementadas en MySQL puedes acceder a lsiguiente enlace:

[Funciones internas para MySQL.](#)

Autoevaluación

La siguiente consulta:

SELECT Matricula FROM REPARACIONES WHERE MONTH(FechaSalida)="January";
Obtiene las matrículas de los vehículos que han salido del taller en Enero. ¿Verdadero o Falso?

- Verdadero Falso

Falso

MONTH devuelve el número del mes de la fecha, no el nombre del mes.

7.- Consultas multitablea.

Caso práctico



[Jonny Goldstein \(CC BY\)](#)

Hasta ahora nos hemos dejado guiar por **Noiba y Vindio para aprender a realizar consultas sencillas que manejaban una sola tabla**. Pero Juan, les idnica que ha llegado el momento de realizar consultas más complejas para obtener datos de varias tablas. Está claro que **en el diseño de Talleres Faber intervienen múltiples tablas y que en el trabajo diario con la base de datos estas consultas van a ser frecuentes**. Tendremos que seguir los pasos de Alejandra para aprender a combinar las tablas y hacer consultas que muestren los resultados buscados.

En las consultas realizadas hasta ahora sólo hemos utilizado una tabla que se indicaba a la derecha de la cláusula FROM, pero hay veces que una consulta necesita columnas de varias tablas.

Ejemplo 1.

Imagina que, en la base de datos campeonato, queremos un listado en el que se muestren los nombres de todos los juegos, su dificultad y el nombre del equipo organizador. Tendremos que consultar las tablas juego y equipo. ¿Y cómo las consultamos? Es importante que al consultarlas las combinemos de forma correcta, esto es, por las columnas que permiten relacionar esas tablas, que normalmente será la columna que es clave ajena en una tabla y la clave primaria de la otra tabla. En las siguientes imágenes te lo explicamos.

◀ 1 2 3 4 ▶

Tabla con los datos de todos los juegos

La columna cdequipo es clave ajena o foránea en la tabla juego, y permite relacionar esta tabla con la tabla equipo por esa columna.

The screenshot shows the MySQL Workbench interface with the following details:

- File Edit View Query Database Server Tools Scripting Help**
- Navigator** pane: SCHEMAS (campeonato selected), Tables (concursan, equipo, juego, participa), Views, Stored Proced, Functions, projectosx, sakila, supergym, sys, talleres_faber, world.
- Query Editor**: A single query window with the SQL command: `1 • SELECT * FROM campeonato.juego;`
- Result Grid**: Displays the results of the query. The columns are: cdjuego, nombre, dificultad, me gusta, and cdequipo. The last column, cdequipo, is highlighted with a red border.
- Output**: Shows the execution log: "1 14:07:30 SELECT * FROM campeonato.juego LIMIT 0, 5000" and "10 row(s) returned".

cdjuego	nombre	dificultad	me gusta	cdequipo
CON	Conflict of Nations	alta	3	03
ELV	Elvenar	baja	3	01
FOE	Forge of Empires	alta	4	06
GOE	Goodgame Empire	alta	HULL	01
GW2	Guild Wars 2.	baja	5	01
HAB	HABBO Hotel	media	4	02
LOL	League of Legends.	media	5	03
MUO	MU online	alta	HULL	04
VIK	Vikings	media	5	01
WOW	World of Warcraft	baja	3	03

Tabla con los datos de todos los equipos

La columna cdequipo es clave primaria en la tabla equipo, y permite relacionar esta tabla con la tabla juego por esa columna, ya que en la tabla juego existe una clave ajena o foránea que la referencia.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema with the 'campeonato' database selected. The 'Tables' section under 'campeonato' lists 'concursan', 'equipo', 'juego', and 'participa'. The 'equipo' table is highlighted. The main pane shows the results of the query:

```
1 • SELECT * FROM campeonato.equipo;
```

cdequipo	nombre	comunidad	anio_funda
01	La marmota	Andalucía	2017
02	Los rebeldes	Murcia	2019
03	Amadeus	Galicia	2018
04	Aries	Asturias	2019
05	Atenea	Galicia	2019
06	Los increíbles	Andalucía	2020
07	El parral	Madrid	2020

The 'Output' pane at the bottom shows the executed query and the number of rows returned:

```
1 14:04:27 SELECT * FROM campeonato.equipo LIMIT 0, 5000  
7 row(s) returned
```

Consulta multitabla

En la siguiente imagen puedes ver el resultado de combinar las tablas **equipo** y **juego** y listar para cada juego, su nombre, dificultad, código de equipo y nombre de equipo organizador.

```
SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'  
FROM juego j, equipo e  
WHERE j.cdequipo=e.cdequipo;
```

Observa que:

- ✓ En este ejemplo, las columnas que permiten combinar las dos tablas tienen el mismo nombre, pero eso no es obligatorio, pueden tener nombres diferentes.
- ✓ Se han utilizado alias de tabla para poder diferenciar las columnas que tienen el mismo nombre y pertenecen a tablas diferentes.

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```

1 • SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'
2   FROM juego j, equipo e
3 WHERE j.cdequipo=e.cdequipo;

```

The results grid displays the following data:

juego	dificultad	cdequipo	equipo
Conflict of Nations	alta	03	Amadeus
Elvenar	baja	01	La marmota
Forge of Empires	alta	06	Los increbles
Goodgame Empire	alta	01	La marmota
Guild Wars 2.	baja	01	La marmota
HABBO Hotel	media	02	Los rebeldes
League of Legends.	media	03	Amadeus
MU online	alta	04	Aries
Vikings	media	01	La marmota
World of Warcraft	baja	03	Amadeus

Below the grid, the status bar says "Result 3" and "Object Info Session".

Workbench (Elaboración propia)

Producto cartesiano

Observa que si al hacer la consulta multitabla omitimos el filtro de combinación de las tablas, (... WHERE j.cdequipo=e.cdequipo) lo que se obtiene es lo que se llama 'el producto cartesiano' de esas tablas, esto es, cada fila de una tabla combinada con todas las filas de la segunda tabla.

```

SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'
FROM juego j, equipo e;

```

The screenshot shows the MySQL Workbench interface with a query editor window. The query is identical to the one above:

```

1 • SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'
2   FROM juego j, equipo e;

```

The results grid displays a much larger number of rows, indicating a full Cartesian product between the two tables. The first few rows are identical to the previous result, but the count continues up to 18 rows.

juego	dificultad	cdequipo	equipo
Conflict of Nations	alta	01	La marmota
Conflict of Nations	alta	02	Los rebeldes
Conflict of Nations	alta	03	Amadeus
Conflict of Nations	alta	04	Aries
Conflict of Nations	alta	05	Atenea
Conflict of Nations	alta	06	Los increibles
Conflict of Nations	alta	07	El parral
Elvenar	baja	01	La marmota
Elvenar	baja	02	Los rebeldes
Elvenar	baja	03	Amadeus
Elvenar	baja	04	Aries
Elvenar	baja	05	Atenea
Elvenar	baja	06	Los increibles

Below the grid, the status bar says "Result 5" and "Object Info Session".

Workbench (Elaboración propia)

Cuando utilizamos varias tablas tenemos que tener en cuenta:

- ✓ Se pueden unir tantas tablas como deseemos.
- ✓ En la cláusula SELECT se pueden citar columnas de todas las tablas.
- ✓ Si hay columnas que tienen el mismo nombre en distintas tablas de la cláusula FROM se deben especificar poniendo el nombre de la tabla antes del nombre de la columna:

NombreTabla.NombreColumna

- ✓ Si el nombre de una columna no coincide en dos tablas no es necesaria la notación anterior, pero es conveniente.
- ✓ En la cláusula WHERE es necesario especificar el criterio que se sigue para combinar las tablas ya que si no se especifica el resultado será el producto cartesiano que combina todas las filas de una tabla con cada una de las filas de la otra.

Ejemplo 2.

En la base de datos TalleresFaber:

Obtener un listado con los nombres y apellidos de los clientes y las matrículas de los coches que nos han traído a reparar al taller.

```
SELECT Nombre, Apellidos, Matricula
FROM CLIENTES, VEHICULOS
WHERE CLIENTES.CodCliente = VEHICULOS. CodCliente;
```

Ejemplo 3.

Consultar los datos personales de los empleados cuya categoría sea “oficial”, junto con las horas trabajadas en cada intervención.

```
SELECT DNI, Nombre, Apellidos, Horas FROM EMPLEADOS, Intervienen
WHERE EMPLEADOS.CodEmpleado = Intervienen.CodEmpleado AND Categoria LIKE "Oficial%";
```

Reflexiona

Cuando realizamos **consultas a varias tablas y no especificamos las columnas por las que se relacionan**, el resultado de la consulta es el **producto cartesiano**, tal y como hemos explicado. Este resultado casi siempre será un error. En muy pocos casos nos interesa mezclar filas de una tabla con filas de otra, sin ningún criterio para establecer esa combinación.

Ejemplo: si combinamos en una consulta CLIENTES con VEHICULOS sin hacer coincidir el CodCliente en ambas tablas, obtendremos un listado que mezcla cada cliente con todos los vehículos de la tabla. Desde un punto de vista lógico esta consulta no tendría ningún sentido.

Autoevaluación

Observa la siguiente consulta y elige la opción correspondiente de la lista:

```
SELECT Matricula, Horas, DNI FROM REPARACIONES, Intervienen, EMPLEADOS
WHERE REPARACIONES.IdReparacion = Intervienen.IdReparacion
AND Intervienen.CodEmpleado = EMPLEADOS.CodEmpleado;
```

- No tiene errores.
- Las tablas no están relacionadas correctamente en la cláusula WHERE.
- No se pueden consultar esas 3 tablas por no estar relacionadas. Error en la cláusula FROM.
- Las columnas seleccionadas no corresponden a las tablas relacionadas. Error en SELECT.

Correcto. Las columnas seleccionadas muestran los vehículos y los empleados que han intervenido en su reparación, así como las horas que han empleado.

Falso. El criterio para combinar las tablas es el adecuado.

Falso. Las tablas pueden consultarse porque hay relación entre ellas.

Falso. Las columnas corresponden a las tablas que están relacionadas en la cláusula FROM.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

Ejercicio Resuelto Multitable

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Listado con el código, nombre y dificultad de cada juego y código y nombre del equipo que lo organiza.
- 2.- Como la anterior, pero solo se muestran los juegos de dificultad baja.
- 3.- Listado de los códigos y nombre de cada concursante y código y nombre de los juegos en los que participa.

[Mostrar retroalimentación](#)

```
-- 1
SELECT j.cdjuego, j.nombre 'juego', j.dificultad, e.cdequipo 'equipo', e.nombre
FROM juego j, equipo e
WHERE j.cdequipo=e.cdequipo;

-- 2
SELECT j.cdjuego, j.nombre 'juego', j.dificultad, e.cdequipo 'equipo', e.nombre
FROM juego j, equipo e
WHERE j.cdequipo=e.cdequipo AND j.dificultad='baja';

-- 3
SELECT c.cdconcur, c.nombre 'concursante', p.cdjuego, j.nombre 'juego'
FROM concursante c, participa p, juego j
WHERE c.cdconcur=p.cdconcur AND p.cdjuego=j.cdjuego;
```

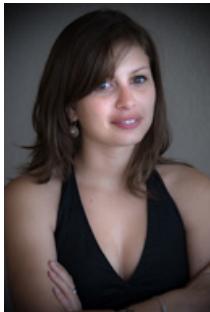
Observa que se ha puesto un alias a algunas columnas y también a las tablas.

Las columnas que aparecen en la cláusula WHERE se denominan **columnas de emparejamiento, composición o combinación** ya que son las que permiten combinar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Combinamos tablas que estén relacionadas entre sí y además, una de las columnas de combinación será clave principal en su tabla, y en otra tabla será

clave ajena. Cuando combinamos columnas debemos especificar de la siguiente forma:
NombreTabla1.Camporelacionado1 = NombreTabla2.Camporelacionado2

8.- Consultas de resumen.

Caso práctico



[Alain Bachelier \(CC BY-NC-SA\)](#)

Noiba está reuniendo información sobre los distintos tipos de consulta que se pueden realizar sobre las tablas de la base de datos Talleres Faber. En este momento veremos cómo obtener resúmenes de los datos aplicando sobre las columnas una serie de **funciones de grupos**. Estas funciones van a permitir calcular, por ejemplo, el precio medio de los recambios, el número de vehículos que visitan el taller en un mes, el tiempo máximo que dura una reparación, etc.

Está claro que se trata de **cuestiones que pueden tener mucho interés para el taller mecánico**. Veamos cómo obtener esos datos

La sentencia SELECT nos va a permitir obtener resúmenes de los datos de modo vertical. Para ello consta de una serie de cláusulas específicas (GROUP BY, HAVING) y tenemos también unas **funciones** llamadas **de agrupamiento o de agregado** que son las que nos dirán qué cálculos queremos realizar sobre los datos (sobre la columna).

Hasta ahora las consultas que hemos visto daban como resultado un subconjunto de filas de la tabla de la que extraímos la información. Sin embargo, este tipo de consultas que vamos a ver no corresponde con ningún valor de la tabla sino un total calculado sobre los datos de la tabla. Esto hará que las consultas de resumen tengan limitaciones que iremos viendo.

Las funciones que podemos utilizar se llaman de agrupamiento (de agregado). Éstas toman un grupo de datos (una columna) y producen un único dato que resume el grupo. Por ejemplo, la función SUM() acepta una columna de datos numéricos y devuelve la suma de estos.

Las funciones de grupos de valores ignoran los valores nulos a la hora de realizar los cálculos.

Estas funciones se muestran en la siguiente tabla:

Funciones de agragado y descripción

FUNCIONES DE COLUMNAS	Descripción
COUNT(Expresión o Columna)	Cuenta el número de valores no nulos de la columna que aparece como argumento. Si ponemos * cuenta las filas que tengan valor no nulo.
SUM(Expresión o Columna)	Calcula la suma de los valores numéricos indicados en el argumento.
MIN(Expresión o Columna)	Obtiene el valor mínimo del argumento indicado.
MAX(Expresión o Columna)	Obtiene el valor máximo del argumento indicado.
AVG(Expresión o Columna)	Obtiene la media aritmética del argumento indicado. No considera los valores nulos.

El argumento de estas funciones suele ser un nombre de columna.

El simple hecho de utilizar una función de agregado en una consulta la convierte en consulta de resumen.

Todas las funciones tiene una sintaxis similar a la siguiente:

FUNCION ([DISTINCT | ALL] expresión)

Debemos tener en cuenta que:

- ✓ La palabra ALL indica que se tienen que tomar todos los valores de la columna. Es el valor por defecto.
- ✓ La palabra DISTINCT indica que se considerarán todas las repeticiones del mismo valor como uno solo (considera valores distintos).
- ✓ El grupo de valores sobre el que actúa la función lo determina el resultado de la expresión que será el nombre de una columna o una expresión basada en una o varias columnas. Por tanto, en la expresión nunca puede aparecer ni una función de agregado ni una subconsulta.
- ✓ Todas las funciones se aplican a las filas del origen de datos una vez ejecutada la cláusula WHERE (si la tuviéramos).
- ✓ Todas las funciones (excepto COUNT) ignoran los valores NULL.
- ✓ Podemos encontrar una función de agrupamiento dentro de una lista de selección en cualquier sitio donde pudiera aparecer el nombre de una columna. Es por eso que puede formar parte de una expresión pero no se pueden anidar funciones de este tipo.
- ✓ No se pueden mezclar funciones de columna con nombres de columna ordinarios, aunque hay excepciones que veremos más adelante.

Reflexiona

Analiza las siguientes consultas:
SELECT COUNT(DISTINCT Marca) "MARCAS" FROM VEHICULOS;
SELECT DISTINCT COUNT(Marca) "MARCAS" FROM VEHICULOS;

¿De qué forma interviene la cláusula DISTINCT en ambas?

[Mostrar retroalimentación](#)

En la primera consulta contamos en la tabla VEHICULOS cuantas marcas distintas aparecen. Si la marca aparece repetida se cuenta una sola vez.

En la segunda consulta la cláusula DISTINCT se aplica después de contar. Eliminaría las filas duplicadas que devuelva la consulta. En este caso no tiene sentido porque la consulta devuelve un único valor, una sola fila, por tanto no es posible que haya duplicados.

Ejercicio resuelto

Realiza las siguientes consultas a las tablas de TalleresFaber:

- 1.- Precio medio de los recambios.
- 2.- Número de vehículos que nos visitaron en Enero.
- 3.- Tiempo máximo que dura una reparación.

[Mostrar retroalimentación](#)

```
-- 1  
SELECT Avg(PrecioReferencia) AS "Precio medio recambios"
```

```

FROM RECAMBIOS;

-- 2
SELECT count(DISTINCT Matricula) AS "Número de vehículos"
FROM REPARACIONES
WHERE FechaEntrada BETWEEN "2011-01-01" AND "2011-01-31";

-- 3
SELECT Max(HorasEmpleadas) AS "Horas empleadas"
FROM REPARACIONES;

```

Ejercicio resuelto consultas resumen.

Realiza las siguientes consultas en la base de datos campeonato:

- 1.- Total de concursantes y la cuota media.
- 2.- Total de concursantes que tienen equipo.
- 3.- Total de comunidades en las que hay equipos.
- 4.- Fecha más reciente con inscripción de concursantes

[Mostrar retroalimentación](#)

```

-- 1
SELECT COUNT(*) 'Total concursantes', AVG(cuota_inscri) 'cuota media'
FROM concursante;

-- 2
SELECT COUNT(*) 'Concursantes con equipo'
FROM concursante
WHERE cdequipo IS NOT NULL;

-- o bien
SELECT COUNT(cdequipo) 'Concursantes con equipo'
FROM concursante;

-- 3
SELECT COUNT(DISTINCT comunidad) 'Comunidades'
FROM equipo;

-- 4
SELECT MAX(fecha_inscri) ' Fecha últimas inscripciones'
FROM concursante;

```

- ✓ Observa que en la consulta 1.-, la cuota media la puedes redondear con 2 decimales, si usas la función ROUND().
- ✓ Observa que en la consulta 4.- puedes formatear la fecha de manera apropiada usando la función date_format()

```

-- 1
SELECT COUNT(*) 'Total concursantes', ROUND(AVG(cuota_inscri),2) 'cuota media'
FROM concursante;

-- 4
SELECT DATE_FORMAT(MAX(fecha_inscri), '%d/%m/%Y') ' Fecha últimas inscripciones'
FROM concursante;

```



9.- Consultas agrupadas.

Caso práctico



[Jonny Goldstein \(CC BY\)](#)

Cuando hemos recorrido con **Noiba** y **Vindio** las cláusulas de la sentencia SELECT, hemos visto que podíamos utilizar GROUP BY para agrupar filas y HAVING para establecer condiciones en las filas agrupadas. Combinando estas cláusulas con las funciones de grupo del apartado anterior, **Juan** les dice a **Noiba** y **Vindio** que podrán encontrar soluciones a cuestiones que hasta ahora no podían resolver y que pueden ser útiles.

Por ejemplo, saber cuántos vehículos de cada marca visitan el taller podría resultar interesante a la hora de especializar a los mecánicos en determinados motores.

En el apartado anterior hemos usado funciones de grupo para calcular totales, obteniendo un único dato resumen. Las consultas agrupadas nos permitirán realizar **cálculos de grupo** sobre filas que tienen un valor coincidente en una o varias columnas, y obtener **subtotales**.

Como **ejemplo**, observa las siguientes imágenes:

- ✓ A la izquierda se muestran todos los juegos con su código, nombre, dificultad y equipo organizador. ¿Cómo podemos obtener para cada equipo el total de juegos que organiza? Para ello hay que agrupar (uso de GROUP BY) por equipo y para cada equipo contar los juegos que tiene. (Observa que es contar las filas en las que aparece repetido el código de equipo)
- ✓ A la derecha se muestra para cada equipo el total de juegos que organiza (Consulta agrupada).

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema 'campeonato' with tables like concursante, equipo, juego, and participa. The main area shows a query editor with the following SQL code:

```
111 • SELECT cdjuego, nombre, dificultad, cdequipo
112   FROM juego;
113
```

Below the query is a 'Result Grid' showing the following data:

cdjuego	nombre	dificultad	cdequipo
CON	Conflict of Nations	alta	03
ELV	Elvenar	baja	01
FOE	Forge of Empires	alta	06
GOE	Goodgame Empire	alta	01
GW2	Guild Wars 2.	baja	01
HAB	HABBO Hotel	media	02
LOL	League of Legends.	media	03
MUO	MU online	alta	04
VIK	Vikings	media	01
WOW	World of Warcraft	baja	03

Workbench (Elaboración propia)

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema is set to 'campeonato'. The SQL editor contains the following query:

```

110 L /*
111 • SELECT cdequipo, COUNT(*) 'Total juegos'
112 FROM juego
113 GROUP BY cdequipd;

```

The Result Grid displays the following data:

cdequipo	Total juegos
01	4
02	1
03	3
04	1
06	1

Below the grid, the status bar shows 'Result 38' and 'Output'.

Imagina que en la base de datos de TalleresFaber queremos saber, por ejemplo, ¿cuántos coches de cada marca han visitado nuestro taller? Hasta ahora lo que podíamos hacer era mostrar las filas de la tabla automóviles, ordenadas por marca y contar visualmente cuántas filas había por cada marca, de la siguiente forma:

Ejemplo:

```

SELECT Marca
FROM VEHICULOS
ORDER BY Marca;

```

¿Y si la tabla tuviera miles de filas? No podríamos hacerlo así. Es el momento de recurrir a la cláusula GROUP BY.

Seguidamente se obtiene un listado con dos columnas: la marca y el número de vehículos que visitan nuestro taller. La consulta devolverá una fila por cada marca distinta, con el nombre de la marca y el número de vehículos de esa marca que nos han visitado. El listado estará ordenado por marcas.

```

SELECT Marca, count(*) AS 'Número de vehículos'
FROM VEHICULOS
GROUP BY Marca
ORDER BY Marca;

```

Tanto GROUP BY como HAVING se usan en consultas de resumen. La sintaxis de estas consultas es:

```

SELECT NombreColumna,..., Función,... FROM NombreTabla [WHERE Condiciones....]
GROUP BY NombreColumnaDeGrupo [HAVING Condiciones] [ORDER BY Criterio]

```

Ejercicio resuelto

Obtener la media de horas empleadas en cada tipo de actuación, mostrando únicamente aquéllas cuya media no sea inferior a una hora.

[Mostrar retroalimentación](#)

```

SELECT Descripcion, AVG(Horas) AS "Horas por término medio"
FROM ACTUACIONES, Realizan
WHERE ACTUACIONES.Referencia = Realizan.Referencia
GROUP BY ACTUACIONES.Referencia
HAVING AVG(Horas) >= 1;

```

Ejercicio Resuelto Consultas Agrupadas

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Total de concursantes en cada equipo. Muestra el código de equipo y total de concursantes, ordenando de más a menos concursantes.
- 2.- La cuota media de inscripción de los concursantes femeninos y masculinos.
- 3.- Total de puntos acumulados que lleva cada concursante que participa en juegos. Ordena el resultado de más a menos puntos.
- 4.- Total de puntos acumulados que lleva cada concursante que participa en juegos. Muestra solo los que llevan entre 100 y 300 puntos.
- 5.- Total de concursantes del equipo '02' inscritos por año.

[Mostrar retroalimentación](#)

```

-- 1
SELECT cdequipo, count(*) 'Total concursantes'
FROM concursante
WHERE cdequipo IS NOT NULL
GROUP BY cdequipo
ORDER BY COUNT(*) DESC;

-- 2
SELECT sexo, AVG(cuota_inscri)
FROM concursante
GROUP BY sexo;

-- 3
SELECT cdconcur, SUM(puntos) 'puntos'
FROM participa
GROUP BY cdconcur
ORDER BY SUM(puntos) DESC;

-- 4
SELECT cdconcur, SUM(puntos) 'puntos'
FROM participa
GROUP BY cdconcur
HAVING SUM(puntos) BETWEEN 100 AND 300
ORDER BY SUM(puntos) DESC;

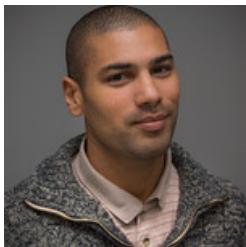
-- 5
SELECT year(fecha_inscri) 'Año', count(*) 'Total inscripciones'
FROM concursante
WHERE cdequipo = '02' AND fecha_inscri IS NOT NULL
GROUP BY year(fecha_inscri);

```



10.- Unión de consultas.

Caso práctico



En este apartado vamos a aprender a diseñar un tipo de consultas que **puede ser de gran utilidad en nuestro taller**. Alejandra se encuentra en muchos casos con tablas que, teniendo la misma estructura, contienen información que se refiere a distintos períodos de tiempo. Por ejemplo las facturas de los clientes almacenadas en distinta tabla cada año (**facturas_2010**, **facturas_2011**). Vindio y Noiba nos van a mostrar cómo podríamos unir la información que contienen dos o más tablas en una consulta, para saber, por ejemplo, la facturación acumulada desde que Talleres Faber inició su actividad.

Alain Bachellier (CC BY-NC-SA)

Seguro que cuando empiezas a trabajar con bases de datos llegará un momento en que dispongas de varias tablas con los mismos datos guardados para distintos registros y quieras unirla en una única tabla. ¿Esto se puede hacer? Es una operación muy común junto a otras. Al fin y al cabo, una consulta da como resultado un conjunto de filas y con conjuntos podemos hacer entre otras, la operación de unión de consultas

Esta operación se realiza sobre:

- ✓ **Dos tablas** con las mismas columnas.
- ✓ **Dos consultas** que contienen las mismas columnas.
- ✓ **Una tabla y una consulta** que tienen las mismas columnas.

El resultado es una tabla con las filas de la primera tabla y las filas de la segunda.

La sintaxis para realizar una UNION es:

```
SELECT.....FROM.....WHERE..... UNION [ALL | DISTINCT] SELECT..... FROM.....WHERE....
```

Si no se especifica ALL o DISTINCT por defecto se usa DISTINCT.

Las condiciones en las que se establece una consulta de unión son:

- ✓ Las consultas unidas deben tener el mismo número de columnas
- ✓ Las columnas tendrán como encabezado, el resultado de la consulta, el identificador de columna de la tabla que está a la izquierda del operador UNION.
- ✓ Se pueden establecer criterios o condiciones para la unión, no unir todas las columnas de las tablas, ordenar el resultado, etc.
- ✓ Si en las tablas unidas hay filas que contienen la misma información, sólo aparece una fila en el resultado de la consulta y no dos repetidas. (UNION DISTINCT)
- ✓ Se puede hacer que las filas repetidas aparezcan en el resultado de la consulta usando UNION ALL.
- ✓ Sobre el resultado final no se pueden aplicar otras cláusulas como GROUP BY, únicamente puede contener <i>ORDER BY</i> al final de la sentencia SELECT.

Ejercicio resuelto

Supongamos que tenemos dos tablas de FACTURAS: Una denominada **FACTURAS_2010** con las facturas emitidas ese año y otra **FACTURAS_2011** con las facturas emitidas en este año. Realizar una consulta de unión entre las facturas de ambas tablas correspondientes al mes de enero.

[Mostrar retroalimentación](#)

```
SELECT *
FROM FACTURAS_2010
WHERE FechaFactura BETWEEN '2010-01-01' AND '2010-01-31'
UNION
SELECT *
FROM FACTURAS_2010
WHERE FechaFactura BETWEEN '2011-01-01' AND '2011-01-31';
```

Autoevaluación

Contesta verdadero o falso según creas que se puede realizar o no la siguiente consulta de unión entre las tablas siguientes:

Recambios

Recambios_chapa	Recambios_mecánica
Codigo_Recambio Varchar(15)	IdRecambio Varchar(10)
Nombre Varchar(60)	Descripcion Varchar(100)
Unidades Varchar(50)	UnidadBase Varchar(50)
Cantidad int	Stock Smallint
Precio float	PrecioReferencia Decimal(6,2)

```
SELECT IdReparación, Nombre, Unidades, Cantidad
FROM Recambios_chapa
UNION
SELECT IdRecambio, Descripcion, UnidadBase, Stock
FROM Recambios_mecanica
ORDER BY Nombre LIMIT 100;
```

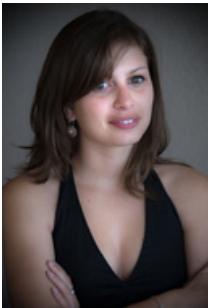
- Verdadero Falso

Verdadero

El nombre de las columnas no tiene porqué coincidir. Es necesario que los datos sean compatibles.

11.- Combinaciones internas.

Caso práctico



[Alain Bachelier \(CC BY-NC-SA\)](#)

Cuando en el apartado 7 hemos aprendido a **hacer consultas a los datos contenidos en varias tablas combinando las filas**, nos hemos dejado por el camino un tipo de consultas que pueden ser también de utilidad en un taller como talleres Faber, porque permiten mostrar, no sólo los empleados y las reparaciones en las que han intervenido, sino que se podrá conocer también los empleados que no hayan intervenido en ninguna reparación. Lo vemos con **Noiba**.

Las combinaciones de tablas que hemos visto hasta ahora en consultas multitabla son “**Composiciones o combinaciones internas**”. En estas se combina información procedente de dos o más tablas, formando filas relacionadas según la condición de búsqueda establecida con WHERE que generalmente es la **clave principal** de una tabla y la **clave ajena relacionada** en la otra tabla.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que MySQL incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de combinación de tablas respecto a las condiciones o filtros de selección de registros. Además, esta nueva sintaxis permitirá realizar dos tipos de combinación de tablas:

- ✓ La composición o combinación interna.
- ✓ La composición o combinación externa.

No centramos en este apartado en la combinación interna.

Estas composiciones se denominan internas porque en el resultado no aparece ninguna fila de una tabla que no tenga correspondencia con las filas de la tabla con la que se combina.

Las sintaxis de las combinaciones internas que vemos a continuación son equivalentes:

```
SELECT ... FROM NombreTabla1 , NombreTabla2 WHERE condición de relación..
```

```
SELECT ... FROM NombreTabla1 INNER JOIN NombreTabla2 ON condición de relación..
```

Ejemplo 1.

En la base de datos campeonato, la consulta que muestra para cada juego su nombre, dificultad, código y nombre de su equipo organizador se puede formular de dos formas:

- ✓ Sin JOIN, con WHERE.

```
SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'  
FROM juego j, equipo e  
WHERE j.cdequipo=e.cdequipo;
```

- ✓ Con JOIN interno o INNER JOIN.

```
SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'  
FROM juego j  
INNER JOIN equipo e ON j.cdequipo=e.cdequipo;
```

Una buena práctica es utilizar JOIN en vez de solo WHERE, pues así separamos las condiciones de combinación de las tablas con el filtro de selección. Observa el siguiente ejemplo.

Ejemplo 2.

En la consulta anterior, supongamos que el listado solo se desea para los juegos cuyo equipo organizador es de Andalucía o Murcia. Las consultas serían:

- ✓ Sin JOIN, con WHERE.

```
SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'  
FROM juego j, equipo e  
WHERE j.cdequipo=e.cdequipo AND e.comunidad IN ('Andalucia', 'Murica');
```

- ✓ Con JOIN interno o INNER JOIN.

```
SELECT j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'  
FROM juego j  
INNER JOIN equipo e ON j.cdequipo=e.cdequipo  
WHERE e.comunidad IN('Andalucia', 'Murica');
```

Otras sintaxis de combinación interna, igualmente equivalentes, son:

```
SELECT ... FROM NombreTabla1 JOIN NombreTabla2 ON condición de relación..  
SELECT ... FROM NombreTabla1 JOIN NombreTabla2 WHERE condición de relación..  
SELECT ....FROM NombreTabla1 INNER JOIN NombreTabla2 USING(ColumnaRelacionada)..
```

Las columnas que aparecen en la cláusula WHERE, on o USING, se denominan **columnas de emparejamiento, combinación o composición** ya que son las que permiten combinar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Combinamos tablas que estén relacionadas entre sí y además, una de las columnas de emparejamiento será clave principal en su tabla y clave ajena en la otra tabla. Si las dos columnas de combinación tienen el mismo nombre, se puede usar USING(nombre_columna) o bien si se usa WHERE u on, habrá que calificar la columna con el alias o nombre de la tabla correspondiente.

Observa las siguientes sconsultas sobre la base de datos TalleresFaber.

Ejemplo 3.

En la salida de la siguiente consulta sólo aparecen los datos de los clientes que tengan alguna factura registrada. Los clientes que no tengan facturas no aparecen. Para los clientes con varias facturas su nombre y apellidos se repite

```
SELECT Nombre, Apellidos, IdFactura  
FROM CLIENTES  
INNER JOIN FACTURAS ON CLIENTES.CodCliente = FACTURAS.CodCliente;
```

De otra forma, con USING sería:

```
SELECT Nombre, Apellidos, IdFactura  
FROM CLIENTES
```

```
INNER JOIN FACTURAS USING(CodCliente);
```

Ejemplo 4.

Esta consulta combina 3 tablas para obtener el importe (unidades*precio) de los recambios sustituidos en cada reparación.

```
SELECT REPARACIONES.IdReparacion, sum(Unidades * PrecioReferencia)
AS 'Importe recambios'
FROM (REPARACIONES INNER JOIN Incluyen ON REPARACIONES.IdReparacion=Incluyen.IdReparacion)
INNER JOIN RECAMBIOS ON Incluyen.IdRecambio=RECAMBIOS.IdRecambio
GROUP BY IdReparacion;
```

Reflexiona

Compara las siguientes alternativas para definir combinaciones internas:

- a. SELECT ... FROM NombreTabla1 INNER JOIN NombreTabla2 ON condición de relación..
- b. SELECTFROM NombreTabla1 INNER JOIN NombreTabla2 USING(ColumnaRelacionada)

¿Por qué podemos considerar que la sintaxis empleada en la opción b es más restrictiva que la de la opción a?

[Mostrar retroalimentación](#)

Porque para relacionar las tablas con USING la columna por la cual se establece la combinación se tiene que llamar igual en ambas tablas, y para establecer una relación las columnas deben tener el mismo tipo de datos y tamaño, pero pueden tener distinto nombre de columna.

Ejercicio Resuelto INNER JOIN

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Listado con el código, nombre y dificultad de cada juego y código, nombre y comunidad del equipo que lo organiza. En el listado deben aparecer sólo los juegos de dificultad alta o media que organizan equipos de Andalucía o Galicia.
- 2.- Listado de los códigos y nombre de cada concursante y código y nombre de los juegos en los que participa. Ordena el listado por nombre de concursante.
- 3.- Como la consulta anterior, pero también debe listarse el código y nombre del equipo que organiza cada juego.
- 4.- Para cada concursante listar su código y nombre y el código y nombre de su ídolo. Ordena el listado por nombre de concursante.

[Mostrar retroalimentación](#)

```
-- 1
SELECT j.cdjuego, j.nombre 'juego', j.dificultad, e.cdequipo, e.nombre 'equipo'
FROM juego j
INNER JOIN equipo e ON j.cdequipo=e.cdequipo
WHERE j.dificultad IN ('alta', 'media') AND e.comunidad IN('Andalucía', 'Galicia');
```

```
-- 2
SELECT c.cdconcur, c.nombre 'concursante', p.cdjuego, j.nombre 'juego'
FROM concursante c
INNER JOIN participa p ON c.cdconcur=p.cdconcur
INNER JOIN juego j ON p.cdjuego=j.cdjuego
ORDER BY c.nombre;

-- 3
SELECT c.cdconcur, c.nombre 'concursante', p.cdjuego, j.nombre 'juego'
FROM concursante c
INNER JOIN participa p ON c.cdconcur=p.cdconcur
INNER JOIN juego j ON p.cdjuego=j.cdjuego
INNER JOIN equipo e ON e.cdequipo=j.cdequipo
ORDER BY c.nombre;

-- 4
SELECT c.cdconcur, c.nombre 'concursante', id.cdconcur, id.nombre 'ídolo'
FROM concursante c
INNER JOIN concursante id ON c.cdidolo=id.cdconcur
ORDER BY c.nombre;
```

12.- Combinaciones externas.

Caso práctico

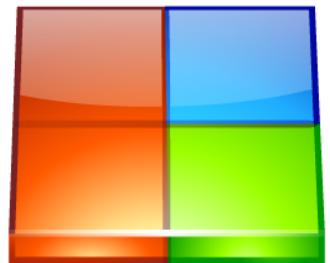


Alain Bachellier ([CC BY-NC-SA](#))

Revisando la información que es necesario obtener en el taller mecánico, **Vindio** se ha dado cuenta que hay casos en los que es necesario mostrar en los listados filas de una tabla que no están relacionadas con la tabla con la que se combina. Por ejemplo, si se desea un listado con todos los clientes y las facturas enviadas, y en el listado se desea que aparezcan también los clientes a los que aún no se le han enviado facturas. ¿Cómo obtener esos datos? **Noiba**, le recuerda que para esos casos existen las combinaciones externas.

¿Has pensado que puede que te interese seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla? Esto puede ser necesario.

Por ejemplo, en la base de datos campeonato, hay concursantes que no tienen equipo, van por libre. Si se deseara un listado con los nombres de todos los concursantes y el nombre de su equipo, con una combinación interna no lo podríamos obtener, pues sólo se combinan las filas en las que hay coincidencia en las columnas de combinación. ¿Cómo se puede obtener? En este caso hay que usar una combinación externa.



Everaldo Coelho ([YellowIcon](#)) ([GNU/GPL](#))

Una **combinación externa** es una variedad de composición de tablas que permite seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla con la que se combina. Cuando no existe correspondencia en la segunda tabla se combina con NULL.

La combinación externa puede expresarse de dos formas:

- ✓ **Combinación externa a la izquierda: LEFT JOIN.** Cada fila de la tabla colocada a la izquierda que no se pueda combinar con ninguna fila de la segunda tabla, añade una fila al resultado con los valores de las columnas de la primera tabla (izquierda) y NULL para todas las columnas de la segunda que no han encontrado resultados coincidentes.
- ✓ **Combinación externa a la derecha: RIGHT JOIN.** Cada fila de la tabla colocada a la derecha que no se pueda combinar con ninguna fila de la primera tabla, añade una fila al resultado con los valores de las columnas de la segunda tabla (derecha) y NULL para todas las columnas de la primera que no encuentren coincidencias.

La sintaxis equivalente en ambos casos para LEFT JOIN:

```
SELECT ....FROM NombreTabla1 LEFT JOIN NombreTabla2 ON condición de relación..  
SELECT ....FROM NombreTabla1 LEFT JOIN NombreTabla2 USING(ColumnaRelacionada)...  
SELECT ....FROM NombreTabla1 LEFT OUTER JOIN NombreTabla2 ON condición de relación..
```

La sintaxis equivalente en ambos casos para RIGHT JOIN:

```
SELECT ....FROM NombreTabla1 RIGHT JOIN NombreTabla2 ON condición de relación..  
SELECT ....FROM NombreTabla1 RIGHT JOIN NombreTabla2 USING(ColumnaRelacionada)...  
SELECT ....FROM NombreTabla1 RIGHT OUTER JOIN NombreTabla2 ON condición de relación..
```

Vemos algunos ejemplos con la base de datos campeonato.

Ejemplo 1.

Listado con el código y nombre de cada concursante y el código y nombre de su equipo. En el listado deben aparecer todos los concursantes, aunque no tengan equipo. Ordenar el listado por nombre de concursante. Con LEFT JOIN.

```
SELECT c.cdconcur, c.nombre 'concursante', e.cdequipo, e.nombre 'equipo'  
FROM concursante c  
LEFT JOIN equipo e ON c.cdequipo=e.cdequipo  
ORDER BY c.nombre;
```

Para los concursantes que no tengan equipo, aparecerá el valor NULL en las columnas e.cdequipo y e.cdnombre.

Ejemplo 2.

Listado con el código y nombre de cada concursante y el código y nombre de su equipo. En el listado deben aparecer todos los concursantes, aunque no tengan equipo. Ordenar el listado por nombre de concursante. Con RIGHT JOIN.

```
SELECT c.cdconcur, c.nombre 'concursante', e.cdequipo, e.nombre 'equipo'  
FROM equipo e  
RIGHT JOIN concursante c ON c.cdequipo=e.cdequipo  
ORDER BY c.nombre;
```

Para los concursantes que no tengan equipo, aparecerá el valor NULL en las columnas e.cdequipo y e.cdnombre.

Ejemplo 3.

Listado con el código y nombre de cada concursante y el código y nombre de su equipo. En el listado deben aparecer todos los concursantes, aunque no tengan equipo. Ordenar el listado por nombre de concursante. Con LEFT JOIN y usando la función IFNULL() para personalizar los valores nulos.

```
SELECT c.cdconcur, c.nombre 'concursante', IFNULL(e.cdequipo,'***'),IFNULL(e.nombre, 'SIN EQUIPO') 'equipo'  
FROM concursante c  
LEFT JOIN equipo e ON c.cdequipo=e.cdequipo  
ORDER BY c.nombre;
```

Para los concursantes que no tengan equipo, aparecerá el valor *** en la columna e.cdequipo y SIN EQUIPO en la columna e.cdnombre.

Vemos algunos ejemplos con la base de datos TalleresFaber

Ejemplo 4.

```
SELECT Nombre, Apellidos, IdFactura  
FROM CLIENTES  
LEFT JOIN FACTURAS ON CLIENTES.CodCliente = FACTURAS.CodCliente;
```

(En este caso aunque tengamos algún cliente al que no le hayamos girado aún ninguna factura se mostrará su nombre y apellidos. En el IdFactura aparecerá NULL).

Ejemplo 5.

```
SELECT Nombre, Apellidos, IdFactura  
FROM CLIENTES  
RIGHT JOIN FACTURAS ON CLIENTES.CodCliente = FACTURAS.CodCliente;
```

(En este caso el resultado mostraría el número de las facturas aunque no existiera ningún cliente relacionado. En las columnas nombre y apellidos mostraría NULL).

(Este caso no podría darse puesto que hemos exigido en la definición de la relación que no pueda haber ninguna factura que no esté asociada a ningún cliente. Integridad referencial).

Ejercicio resuelto

Con la base de datos TalleresFaber, obtener un listado de los vehículos registrados en nuestra base de datos que no hayan sufrido reparaciones.

[Mostrar retroalimentación](#)

```
SELECT VEHICULOS.Matricula, Marca, Modelo, Color, Avería
FROM VEHICULOS
LEFT JOIN REPARACIONES ON VEHICULOS.Matricula = REPARACIONES.Matricula
WHERE Avería IS NULL;
```

Ejercicio Resuelto LEFT JOIN/RIGHT JOIN

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Listado con el código y nombre de todos los concursantes y el nombre y puntos del juego en el que participan. Si un concursante no participa en juegos, en el nombre y puntos del juego deben aparecer tres asteriscos. Ordenar el listado por nombre de concursante.
- 2.- Listado con el código y nombre de todos los equipos, y el total de concursantes que tienen. Deben aparecer en el listado todos los equipos y si no tiene concursantes el equipo, aparecerá un 0 en ese total. Ordenar el listado de más a menos concursantes.

[Mostrar retroalimentación](#)

```
-- 1
SELECT c.cdconcur, c.nombre 'concursante', IFNULL(j.nombre, '***') 'juego', IFNULL(p.puntos,'***') 'puntos'
FROM concursante c
LEFT JOIN participa p ON c.cdconcur=p.cdconcur
LEFT JOIN juego j ON j.cdjuego=p.cdjuego
ORDER BY c.nombre;

-- 2
SELECT e.cdequipo, e.nombre 'equipo', count(c.cdconcur) 'concursantes'
FROM equipo e
LEFT JOIN concursante c ON e.cdequipo=c.cdequipo
GROUP BY e.cdequipo
ORDER BY count(*) DESC;
```

Para saber más

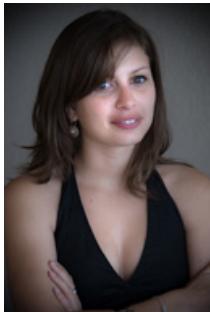
Para más información sobre las combinaciones de tablas en MySQL consulta el siguiente enlace.

[Combinación de tablas en MySQL](#)



13.- Subconsultas.

Caso práctico



[Alain Bachelier \(CC BY-NC-SA\)](#)

Siguiendo los pasos de Noiba, parece que ya estamos listos para escribir cualquier tipo de consulta en SQL que nos permita acceder a la información almacenada, pero no es así. En muchas ocasiones para realizar una consulta necesitaremos los datos devueltos por otra consulta. Investigaremos con Alejandra si este tipo de consultas, denominadas **subconsultas**, están recogidas en el SQL de nuestro SGBD. MySQL ,y cómo se utilizan.

Para entender la necesidad de realizar una subconsulta veremos un ejemplo: supongamos que queremos obtener un listado con las actuaciones que se llevan a cabo en nuestro taller, cuyo importe esté por encima del importe medio.

Con lo que hemos visto hasta ahora necesitaríamos hacer dos consultas:

- ✓ Obtener el importe medio de las actuaciones.
- ✓ Obtener un listado con la descripción de las actuaciones cuyo importe sea mayor que el importe medio.

En primer lugar haríamos la primera consulta:

```
SELECT AVG(Importe)  
FROM ACTUACIONES;
```

Supongamos que el Importe medio obtenido sea 100. Ahora haríamos la segunda consulta basada en el resultado anterior:

```
SELECT Descripcion  
FROM ACTUACIONES  
WHERE Importe >100;
```

Las subconsultas permiten escribir estas dos consultas en una, escribiendo tras la cláusula WHERE otra sentencia SELECT. Así el SELECT principal actúa sobre las filas devueltas por el SELECT anidado.

La sintaxis será:

```
SELECT NombreColumna,... FROM NombreTabla,... WHERE Expresión OperadorRelacional  
(SELECT....)
```

En nuestro ejemplo sería:

```
SELECT Descripcion  
FROM ACTUACIONES  
WHERE Importe >  
(SELECT avg(Importe) FROM ACTUACIONES);
```

La subconsulta entre paréntesis (subconsulta anidada) se ejecuta primero y el valor extraído se introduce en la consulta más externa.

Subconsultas que devuelven valores simples.

Se trata de subconsultas que devuelven una fila o un valor simple. En estas se utilizan los operadores de comparación (<, >, <>, <=, >=, =).

Si la subconsulta devuelve más de una fila da ERROR.

Vemos otro **ejemplo**:

Listado con el nombre de los concursantes del mismo equipo que Ismael Rojo.

```
SELECT cdconcur, nombre
FROM concursante
WHERE cdequipo = (SELECT cdequipo FROM concursante WHERE nombre LIKE 'Ismael Rojo');
```

Subconsultas que devuelven listas de valores.

Son subconsultas que devuelven más de una fila o más de un valor. En ese caso utilizaremos el operador IN ó NOT IN en la cláusula WHERE.

Por **ejemplo**:

Listado de los concursantes que aún no están participando en ningún juego.

```
SELECT *
FROM concursante
WHERE cdconcur NOT IN (SELECT cdconcur FROM participa);
```

El tipo de dato de la expresión situada después de WHERE debe coincidir con el tipo de dato devuelto por la subconsulta.

Dentro de una consulta se pueden usar varias subconsultas enlazadas con el operador AND. Las subconsultas se pueden usar dentro de otras sentencias como DELETE, INSERT y UPDATE.

Las subconsultas pueden anidarse de forma que una subconsulta aparezca en la cláusula WHERE de otra subconsulta. También pueden aparecer subconsultas tras las cláusulas FROM y HAVING.

Reflexiona

En la práctica una consulta consume mucho más tiempo y memoria cuando se incrementa el número de niveles de anidamiento, además resulta más difícil de leer y de mantener cuando contiene más de dos niveles de anidamiento.

Ejercicio resuelto

En la base de datos TalleresFaber obtener los nombres de los empleados que no hayan intervenido en reparaciones desde Enero.

[Mostrar retroalimentación](#)

```
SELECT Apellidos, Nombre
FROM EMPLEADOS
WHERE CodEmpleado NOT IN
(SELECT CodEmpleado FROM Intervienen, REPARACIONES
WHERE REPARACIONES.IdReparacion = Intervienen.IdReparacion AND FechaSalida>='2011-01-01');
```

Ejercicio Resuelto Subconsultas

En la base de datos campeonato, realiza las siguientes consultas:

- 1.- Código y nombre del o los concursantes con cuota mayor o igual que la media.
- 2.- Datos del o los concursantes que se inscribieron en la fecha más reciente.
- 3.- Datos de los juegos con la misma dificultad que el nombre del juego 'Elvenar'.
- 4.- Datos de los juegos cuyos equipos son de Andalucía.

[Mostrar retroalimentación](#)

```
-- 1
SELECT cdconcur, nombre
FROM concursante
WHERE cuota_inscri >=(SELECT AVG(cuota_inscri) FROM concursante);

-- 2
SELECT *
FROM concursante
WHERE fecha_inscri =(SELECT MAX(fecha_inscri) FROM concursante);

-- 3
SELECT *
FROM juego
WHERE dificultad = (SELECT dificultad FROM juego WHERE nombre LIKE 'Elvenar');

-- 4
SELECT *
FROM juego
WHERE cdequipo IN (SELECT cdequipo FROM equipo WHERE comunidad LIKE 'Andalucia');
```

Para saber más

Puedes consultar otros ejemplos de subconsultas y más posibilidades de uso en el siguiente enlace:

[Subconsultas en MySQL](#)

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 01.00.00	Fecha de actualización: 23/07/20
Versión inicial de los materiales.	

