Tratamiento de Datos y Consultas (UDs 5 y 4)

Adaptado para MySQL Server

Contenido

Sentencias DML

En el lenguaje SQL existen 4 sentencias que forman el DML (Data Manipulation Language, Lenguaje de Manipulación de Datos). Son aquellas sentencias que nos permiten manipular la información que almacenamos en las Bases de Datos.

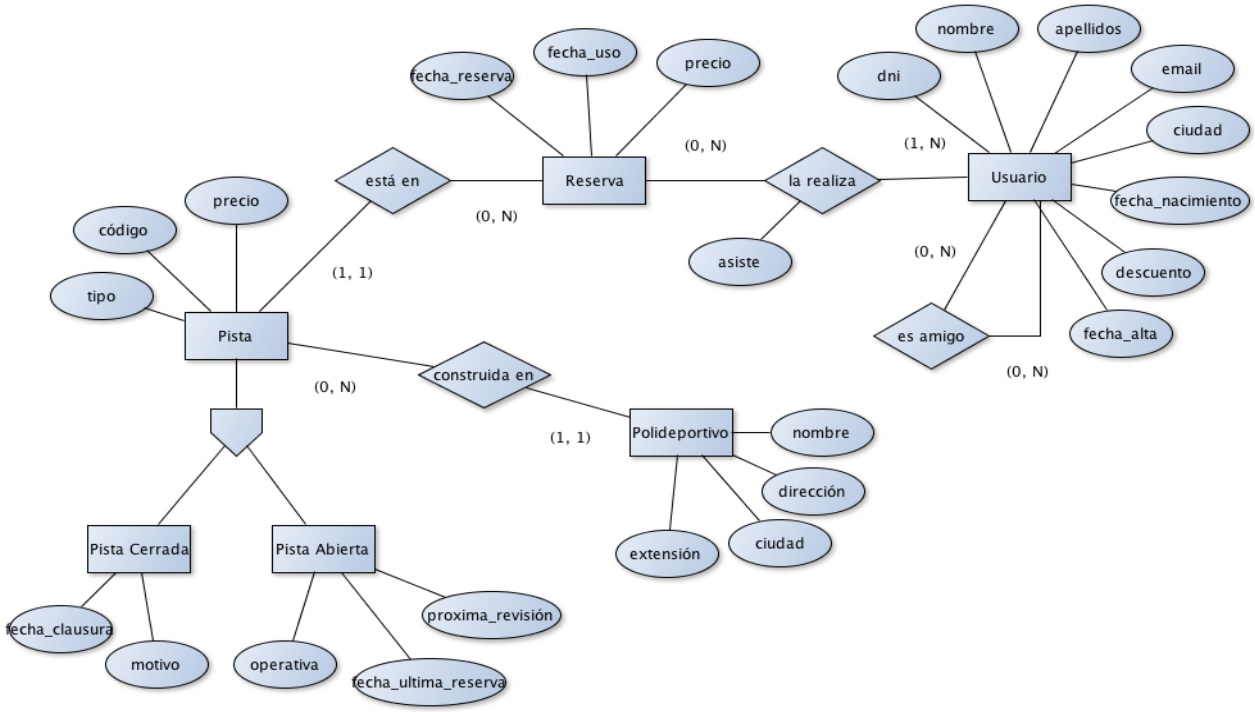
Existen 4 instrucciones que nos permitirán insertar datos en una tabla (INSERT), modificar esos datos (UPDATE), eliminarlos (DELETE) y consultarlos (SELECT).

Base de Datos de prueba

Todos los ejemplos de este bloque se han escrito tomando como referencia la Base de Datos **reservas** que se proporciona junto con este documento. También pueden aparecer algunos

Desarrollo de Aplicaciones Multiplataforma Fernando Valdeón
Bases de datos
ejemplos tomando como referencia las Bases de Datos de ejemplos oficiales de MySQL

A continuación se muestran el modelo entidad-relación y relacional de esta Base de Datos de prueba:



pistas (#id, codigo, tipo, precio, -id_polideportivo)
pistas_abiertas(#-id_pista, operativa, precio, fecha_ultima_reserva, proxima_revision)
pistas_cerradas (#-id_pista, fecha_clausura, motivo)
polideportivos (#id, nombre, direccion, ciudad, extension)
reservas (#id, fecha_reserva, fecha_uso, precio, -id_pista)
usuarios (#id, dni, nombre, apellidos, email, ciudad, fecha_nacimiento, descuento, fecha_alta)
usuario_usuario (#(-id_usuario, -id_amigo))
usuario_reserva (#(-id_usuario, -id_reserva), asiste)

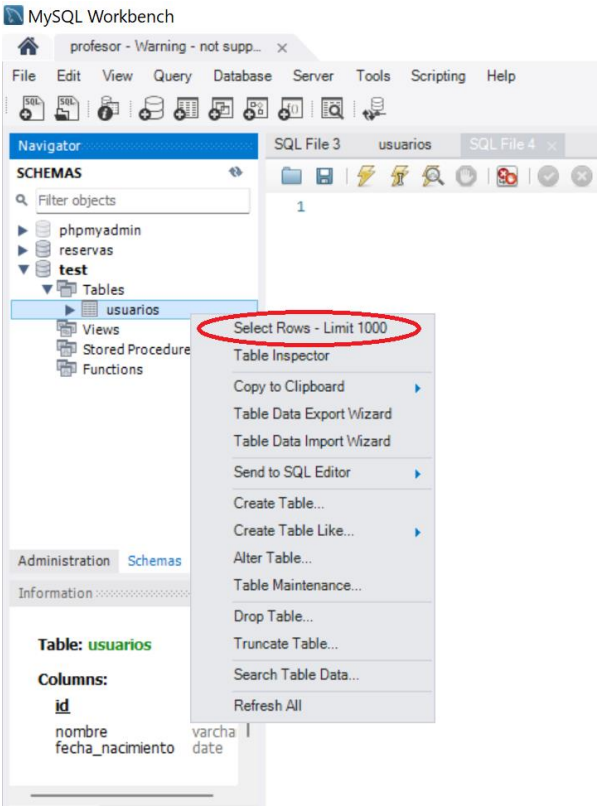
La base de datos puede descargarse desde este enlace

Modificar Información

Mediante MySQL WorkBench

Para editar los datos que contiene una tabla mediante el cliente Workbench puedo usar su interfaz gráfica.

Primero mostraré los datos de la tabla, puede que haya datos o esté vacía. Localizo la tabla en el panel de navegación y pulso botón derecho sobre ella y selecciono **Select Rows (imagen derecha)**.

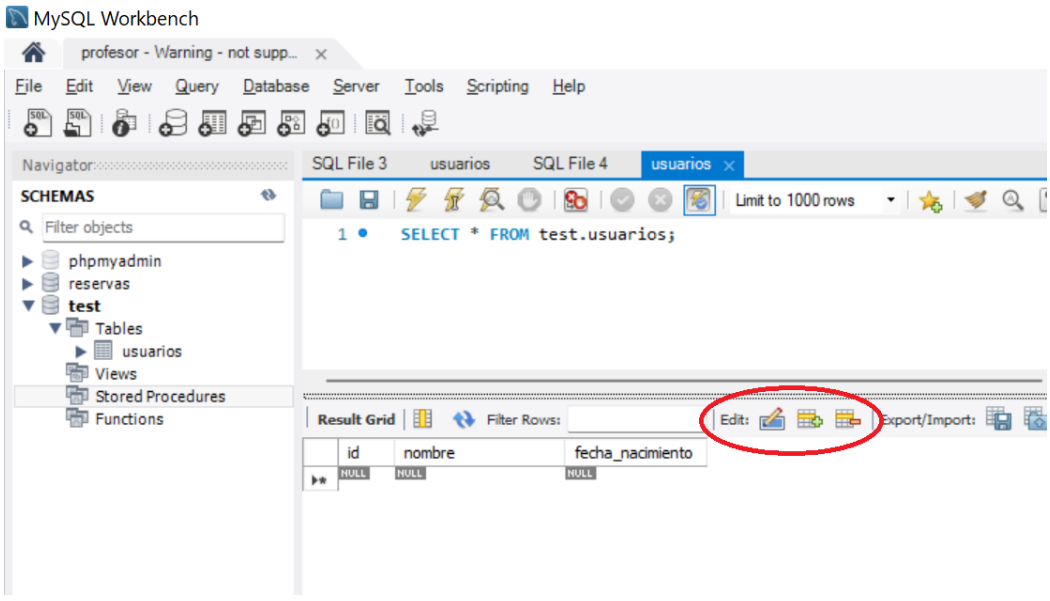


Cuando muestro el contenido de la tabla, puede que haya filas de datos o esté vacía. Para insertar, modificar o borrar filas puedo pulsar sobre los botones que veo en la imagen inferior:

El primero me permite modificar los valores de la fila seleccionada.

El Segundo, me permite añadir una nueva fila a la tabla y dar valores a las columnas.

El tercero me permite eliminar las filas seleccionadas.



Después de realizar los cambios (inserciones, modificaciones o borrados) debemos pulsar sobre el botón **Apply Changes** abajo a la derecha.

Inserción de registros

La inserción de nuevos registros a una tabla se efectúa con la sentencia INSERT, que tiene el siguiente formato:

```
INSERT INTO nombre_tabla [ '('columnas')' ]
```

```
{ VALUES '(' { valores } ')',} | consulta
```

Indicar las columnas en las que insertas datos es opcional. Si no se indican se debe dar valor a todas las columnas en el orden en que están en la tabla. Si se indican las columnas, se da solamente valor a esos campos.

Veamos algunos ejemplos:

- Para insertar una fila con todos los campos de la tabla

```
-- La tabla pistas tiene los campos: (id, codigo, tipo, precio, id_polideportivo)
INSERT INTO pistas
VALUES (1 , 'A34565', 'tenis', '7.34', '12');
```

- Para insertar una fila en una tabla dando valor a ciertas columnas

```
INSERT INTO usuarios (dni, nombre, apellidos, email, fecha_nacimiento)
VALUES ('123456789A', 'Antonio', 'García', 'agarcia@gmail.com', '1990-12-12');

INSERT INTO usuarios
VALUES (200, '123456789A', 'Antonio', 'García', 'agarcia@gmail.com', 'Zaragoza',
'1990-12-12', 0.3, '2003-02-01');

INSERT INTO usuarios (id, dni, nombre, apellidos, email, fecha_nacimiento)
VALUES (45, '123456789A', 'Antonio', 'García', 'agarcia@gmail.com', '1990-12-12');
```

- Para insertar varias filas en una tabla

```
INSERT INTO usuarios (dni, nombre, apellidos, email, fecha_nacimiento)
VALUES ('123456789A', 'Pepe', 'Sanz', 'psanz@gmail.com', '1990-12-12'),
   ('987654321Z', 'Luis', 'Peréz', 'lperez@gmail.com', '1988-01-03');
```

- Para insertar el resultado de una consulta en una tabla (suponiendo que la tabla otros_usuarios existe y tiene al menos los campos indicados del mismo tipo que en la tabla usuarios)

```
INSERT INTO usuario (dni, nombre, apellidos, email, fecha_nacimiento)
SELECT dni, nombre, apellidos, email, fecha_nacimiento
FROM otros_usuarios;
```

- También puedo crear una tabla e insertar datos basándome en la ejecución de una consulta:

```
CREATE TABLE tabla1 AS SELECT dni, nombre, apellidos, email, fecha_nacimiento
FROM otros_usuarios;
```

Modificación de registros

La modificación de registros ya insertados en la tabla se realiza con la sentencia UPDATE, que tiene el siguiente formato:

```
UPDATE nombre_tabla
SET columna = valor [ {, columna = valor} ]
```

```
[ WHERE condiciones ]
```

Veamos algunos ejemplos:

- Actualizar una columna de una fila

```
UPDATE usuarios
SET nombre = 'Felipe'
WHERE id = 12;
```

- Actualizar varias columnas de una fila

```
UPDATE usuarios
SET nombre = 'Felipe', dni = '123654789H'
WHERE id = 15;
```

- Actualizar una columna de varias filas

```
UPDATE pistas
SET precio = precio + precio * 0.10
WHERE precio < 20 AND tipo = 'tenis';
```

- Actualizar una columna utilizando una subconsulta:

```
-- Reduce el precio de las pistas que no se han reservado todavía
UPDATE pistas
SET precio = precio - precio * 0.1
WHERE id NOT IN (SELECT id_pista FROM reservas);
```

Cabe destacar que la ausencia de condiciones (WHERE) en una sentencia de actualización, ejecutaría dicha sentencia sobre todas las filas de la tabla.

Eliminación de registros

El borrado de filas de una tabla se efectúa con la sentencia DELETE, que tiene el siguiente formato:

```
DELETE FROM nombre_tabla
[ WHERE condiciones ]
```

Veamos algunos ejemplos:

- Elimina todos los usuarios

```
DELETE FROM usuarios;
```

- Borrar una fila estableciendo una condición

```
DELETE FROM pistas
WHERE id = 10;
```

- Borrar varias filas estableciendo varias condiciones

```
DELETE FROM pistas
WHERE tipo = 'baloncesto' OR codigo = 'BAL001';
```

- Borrar filas relacionando varias tablas

```
-- Elimina los usuarios que se dieron de alta antes de 2014
-- y aún no han reservado ninguna pista
DELETE FROM usuarios
WHERE id NOT IN (SELECT id_usuario FROM usuario_reserva)
AND fecha_alta < '2014-01-01';
```

Cabe destacar que la ausencia de condiciones en una sentencia de borrado, eliminaría todas las filas de la tabla.

Consulta de registros

La sentencia SELECT

La consulta de registros es la operación más compleja, y también la más ejecutada, de una Base de Datos. Se lleva a cabo con la sentencia SELECT, que tiene el siguiente formato:

```
SELECT columnas
FROM tablas
[ WHERE condiciones ]
[ GROUP BY columnas ]
[ HAVING condiciones_de_grupo ]
[ ORDER BY columnas_a_ordenar [ASC|DESC] ]
```

Veamos para que sirve cada una de las cláusulas de esta sentencia:

SELECT . . . FROM . . .

La cláusula SELECT se utiliza para selecciona las columnas que se quieren visualizar como resultado de la consulta. Se puede seleccionar cualquier columna de las tablas afectadas por la consulta (cláusula FROM), valores constantes establecidos a la hora de ejecutar la consulta, o bien el comodín '*' para indicar que se quieren visualizar todas las columnas afectadas.

La cláusula FROM permite indicar con qué tablas se trabajará en la consulta. No siempre serán tablas de las que se visualicen columnas, puesto que muchas veces sólo se utilizarán para relacionar unas tablas con otras. En cualquier caso, se usen para que se visualicen sus campos o bien para relacionar otras tablas (que no están directamente relacionadas), se deben indicar en esta cláusula.

Veamos algunos ejemplos:

- Consulta de una columna de una tabla

```
--  Nombre de todos los usuarios (incluye repeticiones)
SELECT nombre
FROM usuarios;
```

- Consulta de dos columnas de una tabla

```
-- Nombre y apellidos de todos los usuarios
SELECT nombre, apellidos
FROM usuarios;
```

- Consulta de todas las columnas de una tabla

```
-- Toda la información de todos los usuarios
SELECT *
FROM usuarios;
```

WHERE

La cláusula WHERE permite establecer condiciones sobre que filas se mostrarán en una sentencia de consulta. En ausencia de esta cláusula se muestran todos los registros de la tabla (aunque sólo las columnas establecidas en la cláusula SELECT). Si se indican condiciones mediante la cláusula WHERE sólo se mostrarán aquellas filas que las cumplan.

Veamos algunos ejemplos:

```
-- Nombre y dirección de los polideportivos de Zaragoza
SELECT nombre, direccion
FROM polideportivos
WHERE ciudad = 'Zaragoza';
```

Además, nos permitirá establecer condiciones para establecer lo que se conoce como un INNER JOIN (implícito) entre dos o más tablas:

```
-- Código y tipo de las pistas de tenis que están operativas
SELECT pistas.codigo, pistas.tipo
FROM pistas, pistas_abiertas
WHERE pistas.id = pistas_abiertas.id_pista AND
    pistas_abiertas.operativa = 1 AND pistas.tipo = 'tenis';
```

```
-- Código y tipo de las pistas de los polideportivos
-- de Zaragoza
SELECT pistas.codigo, pistas.tipo
FROM pistas, polideportivos
WHERE pistas.id_polideportivo = polideportivos.id AND polideportivos.ciudad = 'Zaragoza'
```

De manera que si utilizamos *alias* para los nombres de las tablas, podemos escribir la misma consulta algo más rápido:

```
-- Código y tipo de las pistas de los polideportivos
-- de Zaragoza
SELECT P.codigo, P.tipo
FROM pistas P, polideportivos PP
WHERE P.id_polideportivo = PP.id AND PP.ciudad = 'Zaragoza'
```

GROUP BY / HAVING

Las cláusulas GROUP BY y HAVING permiten crear agrupaciones de datos y establecer condiciones sobre dichas agrupaciones, respectivamente. Aspectos importantes:

- Solo agrupo cuando quiero obtener datos a partir de una función de agregación.
- En la cláusula SELECT solo puedo indicar funciones de agregación y el campo por el que

agrupo.

```sql
-- Número de polideportivos hay en cada ciudad
SELECT ciudad, COUNT(*) AS cantidad
FROM polideportivos
GROUP BY ciudad;
```

```sql
-- Número de polideportivos hay en cada ciudad, solamente de aquellas
-- ciudades donde hay más de 10.000
SELECT ciudad, COUNT(*) AS cantidad
FROM polideportivos
GROUP BY ciudad
HAVING COUNT(*) > 10000;
```

```sql
-- Número de usuarios en cada ciudad
SELECT ciudad, COUNT(*) AS cantidad
FROM usuarios
GROUP BY ciudad;
```

También es posible añadir cláusulas WHERE para filtrar registros y agrupar el resultado final

```sql
-- Precio medio, por tipo de pista, de las pistas que no están operativas
SELECT P.tipo, AVG(P.precio) AS precio_medio
FROM pistas P, pistas_abiertas PA
WHERE P.id = PA.id_pista AND PA.operativa = FALSE
GROUP BY P.tipo;
```

En los casos en los que usemos las cláusulas WHERE y HAVING junto con GROUP BY, debemos tener en cuenta que:

- WHERE: filtra los registros antes de agrupar
- HAVING: filtra los resultados después de agrupar

Además, hay que tener en cuenta que en ocasiones nos puede interesar agrupar por algún campo que no va a ser mostrado como resultado de la consulta en la cláusula SELECT. En este caso, agruparíamos por id pero mostraríamos el nombre del polideportivo, ya que puede ser un dato más interesante para el usuario. Además, en casos en que el campo que queramos mostrar pueda repetirse, no conviene agrupar por dicho campo puesto que agruparía registros que en principio no tienen nada que ver. Además, en los dos siguientes ejemplos, unimos dos tablas para realizar la consulta y trabajar con los datos relacionados entre ambas (utilizando la cláusula WHERE para realizar la unión o *JOIN* que más adelante se explicará con más detalle):

```sql
-- Cantidad de pistas que hay en cada polideportivo
SELECT PP.nombre, COUNT(*) AS numero_pistas
FROM polideportivos PP, pistas P
WHERE PP.id = P.id_polideportivo
GROUP BY PP.id;
```

```sql
-- Número de reservas que ha hecho cada usuario
SELECT U.apellidos, U.nombre, U.ciudad, COUNT(*) AS numero_reservas
FROM usuarios U, usuario_reserva UR
WHERE U.id = UR.id_usuario
GROUP BY U.id
```

```
ORDER BY U.apellidos;
```

```
-- Número de pistas que hay de cada tipo en el polideportivo 'ACTUR 1'
SELECT P.tipo, COUNT(*) AS numero_pistas
FROM pistas P, polideportivos PP
WHERE P.id_polideportivo = PP.id AND PP.nombre = 'ACTUR 1'
GROUP BY P.tipo;
```

ORDER BY

La cláusula ORDER BY permite ordenar el resultado de cualquier consulta atendiendo al campo o campos especificados en esta cláusula, ya sea en orden ascendente (ASC) o descendente (DESC)

```
-- Nombre y apellidos de los usuarios, ordenados por nombre
SELECT nombre, apellidos
FROM usuarios
ORDER BY nombre ASC;
```

Operadores

A la hora de establecer condiciones en una sentencia de consulta, podremos utilizar los siguientes operadores:

- =: Igual
- <: Menor
- >: Mayor
- < =: Menor o igual
- >=: Mayor o igual
- <>: Distinto
- NOT: Operador lógico para la negación de condiciones
- AND: Operador lógico para la conjunción de condiciones
- OR: Operador lógico para la disyunción de condiciones
- DISTINCT: Se utiliza para indicar a la cláusula SELECT que no se muestren valor de columnas repetidos
- LIKE: Permite comprobar si una cadena de caracteres cumple algún patrón determinado:

Permite la expresión de patrones a través de dos caracteres comodín:

- El carácter '_' para expresar un único carácter
- El carácter '%' para expresar cualquier secuencia de caracteres o incluso la secuencia vacía

Veamos un par de ejemplos:

```
-- Nombre de los polideportivos que están en una ciudad
-- cuyo nombre empieza por Z y tiene 8 caracteres
SELECT nombre
FROM polideportivos
WHERE ciudad LIKE 'Z_______';
```

```
-- Nombre de los polideportivos que están en una ciudad
-- cuyo nombre empieza por Z
SELECT nombre
FROM polideportivos
WHERE ciudad LIKE 'Z%';
```

- IN | NOT IN: Permite comprobar si un valor coincide (o no) con algún valor especificado como un conjunto.

```
-- Nombre y extensión de los polideportivos de
-- Zaragoza, Huesca y Teruel
SELECT nombre, extension
FROM polideportivos
WHERE ciudad IN ('Zaragoza', 'Huesca', 'Teruel');
```

- IS NULL | IS NOT NULL: Se utiliza para comprobar si un valor es igual (o no) a NULL. Dicha comprobación no debe realizar con ningún otro operador.

```
-- Nombre y apellidos de los usuarios que no indicaron su fecha de nacimiento
SELECT nombre, apellidos
FROM usuarios
WHERE fecha_nacimiento IS NULL;
```

- BETWEEN: Permite comprobar si el valor de una columna está comprendido entre dos valores determinados:

```
-- Nombre y apellidos de los usuarios que tienen un descuento
-- entre 10 y 20 %
SELECT nombre, apellidos
FROM usuarios
WHERE descuento BETWEEN 1 AND 2;
```

Operadores Aritméticos

Por otra parte, también tenemos los operadores aritméticos habituales de los lenguajes de programación:

Operador	Operación
+	Suma
-	Resta
*	Producto
/	División decimal
div	División entera
mod ó %	Módulo: resto entero de división.

```
-- Calcula el 10% de las ganancias por reservas del año pasado
SELECT (SUM(precio) * 0.1) AS “impuestos”
FROM reservas
```

```
WHERE YEAR(fecha_reserva) = YEAR(CURDATE()) - 1;
```

Funciones agregadas

Son funciones que proporciona el lenguaje SQL, que permiten realizar operaciones sobre los datos de una base de datos:

- COUNT: Devuelve el número de filas seleccionadas. Entre los paréntesis podemos indicar * o un campo de la tabla.
  - (*): cuenta la cantidad de registros
  - (columna): cuenta la cantidad de valores de esa columna (no cuenta valores null). También puedo usarlo para contar valores distintos.

```
-- Número de pistas
SELECT COUNT(*)
FROM pistas;

-- Número de polideportivos en Zaragoza
SELECT COUNT(*)
FROM polideportivos
WHERE ciudad = 'Zaragoza';

-- Número de ciudades distintas
SELECT COUNT(DISTINCT ciudad)
FROM usuarios;
```

- SUM: Devuelve la suma de todos los valores de una columna

```
-- Cuánto dinero costaría alquilar todas las pistas del
-- polideportivo cuyo id es 23
SELECT SUM(precio)
FROM pistas
WHERE id_polideportivo = 23;
```

- MIN: Devuelve el valor mínimo de una columna

```
-- Cuánto vale la pista más barata
SELECT MIN(precio)
FROM pistas;
```

- MAX: Devuelve el valor máximo de una columna

```
-- Cuánto vale la pista más cara
SELECT MAX(precio)
FROM pistas;
```

- AVG: Devuelve el valor medio de los valores de una columna

```
-- Valor medio de las pistas
SELECT AVG(precio)
FROM pistas;
```

Hay que tener en cuenta que, excepto la función COUNT, todas las demás devolverán el valor NULL si no hay columnas sobre las que puedan operar. La función COUNT, sin embargo, devolverá el valor 0 en ese caso.

Subconsultas

La creación de subconsultas permite utilizar el resultado de una consulta como valor de entrada para la condición de otra consulta principal.

Veamos unos ejemplos

```
-- Código y tipo de la pista más barata
SELECT codigo, tipo
FROM pistas
WHERE precio = (SELECT MIN(precio)
        FROM pistas);

-- Codigo y tipo de las pistas cuyo precio está por encima de la media
SELECT codigo, tipo
FROM pistas
WHERE precio > (SELECT AVG(precio)
        FROM pistas)

-- Nombre y apellidos de los usuarios que aún no han realizado
-- ninguna reserva
SELECT nombre, apellidos
FROM usuarios
WHERE id NOT IN (SELECT id_usuario
         FROM usuario_reserva)

-- Mostrar codigo y tipo de las pistas que se han reservado este año
SELECT codigo, tipo
FROM pistas
WHERE id IN (SELECT id_pista
       FROM reservas
       WHERE YEAR(fecha_reserva) = YEAR(CURDATE()))
```

Consultas con varias tablas

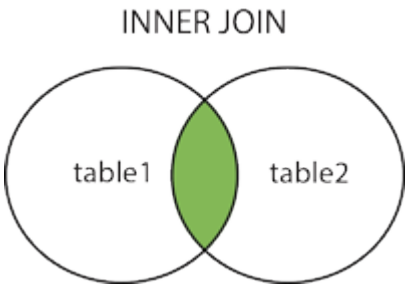
Como se ha visto anteriormente, aplicando la cláusula WHERE, introducíamos una posibilidad más a la hora de realizar consultas sobre los datos de nuestra base de datos, lo que se conoce como una consulta de varias tablas o combinación de tablas (en inglés JOIN).

Veamos varios ejemplos:

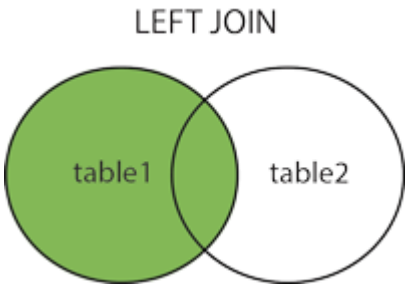
```
-- Mostrar, para cada polideportivo, el código y tipo de las pistas
-- que tiene
SELECT PP.id, PP.nombre, P.codigo, P.tipo
FROM polideportivos PP, pistas P
WHERE PP.id = P.id_polideportivo
```

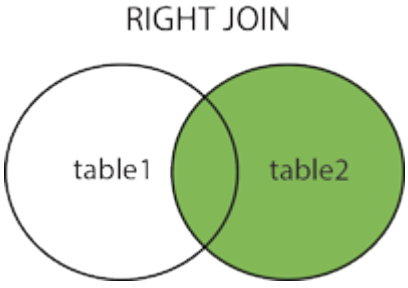
```
-- Consulta equivalente
SELECT PP.id, PP.nombre, P.codigo, P.tipo
FROM polideportivos PP INNER JOIN pistas P ON PP.id = P.id_polideportivo
```

De esta forma, al incluir a más de una tabla en la cláusula FROM estamos realizando lo que se conoce como una combinación interna (INNER JOIN), de forma que cabe la posibilidad de que sólo se muestren algunos datos de alguna de las tablas, puesto que la combinación interna sólo se queda con aquellos registros que están relacionadas con algún registro de la otra tabla.



Si ahora tenemos en cuenta que algún cliente puede no haber realizado pedido alguno, veremos como no aparecen en el resultado de la consulta anterior. En algunos casos eso será lo que queramos, pero quizás en otros casos nos interesa que su nombre aparezca, aunque no esté vinculado con ninguno de los pedidos. En este caso nos interesa lo que se conoce como OUTER JOIN. En resumen, si alguna fila de cualquier tabla de la consulta puede no estar relacionado con alguna de las otras, puede ser interesante utilizar un OUTER JOIN. Decidir si utilizar un LEFT OUTER JOIN o bien un RIGHT OUTER JOIN depende de si el dato que puede no tener relación con la otra tabla está a la izquierda o la derecha, respectivamente, según el sentido en que se escribe el código *SQL*.






```
-- Mostrar, para cada pista, el codigo de reserva que ha tenido
-- Si nunca se ha reservado, se mostrarán sólo sus datos
```

```sql
-- (En este caso puede pasar que una pista no esté relacionada con
-- ninguna reserva, como se puede ver en el modelo E-R)
SELECT P.id, P.codigo, P.tipo, R.id AS codigo
FROM pistas P LEFT OUTER JOIN reservas R ON P.id = R.id_pista;
ORDER BY P.codigo
```

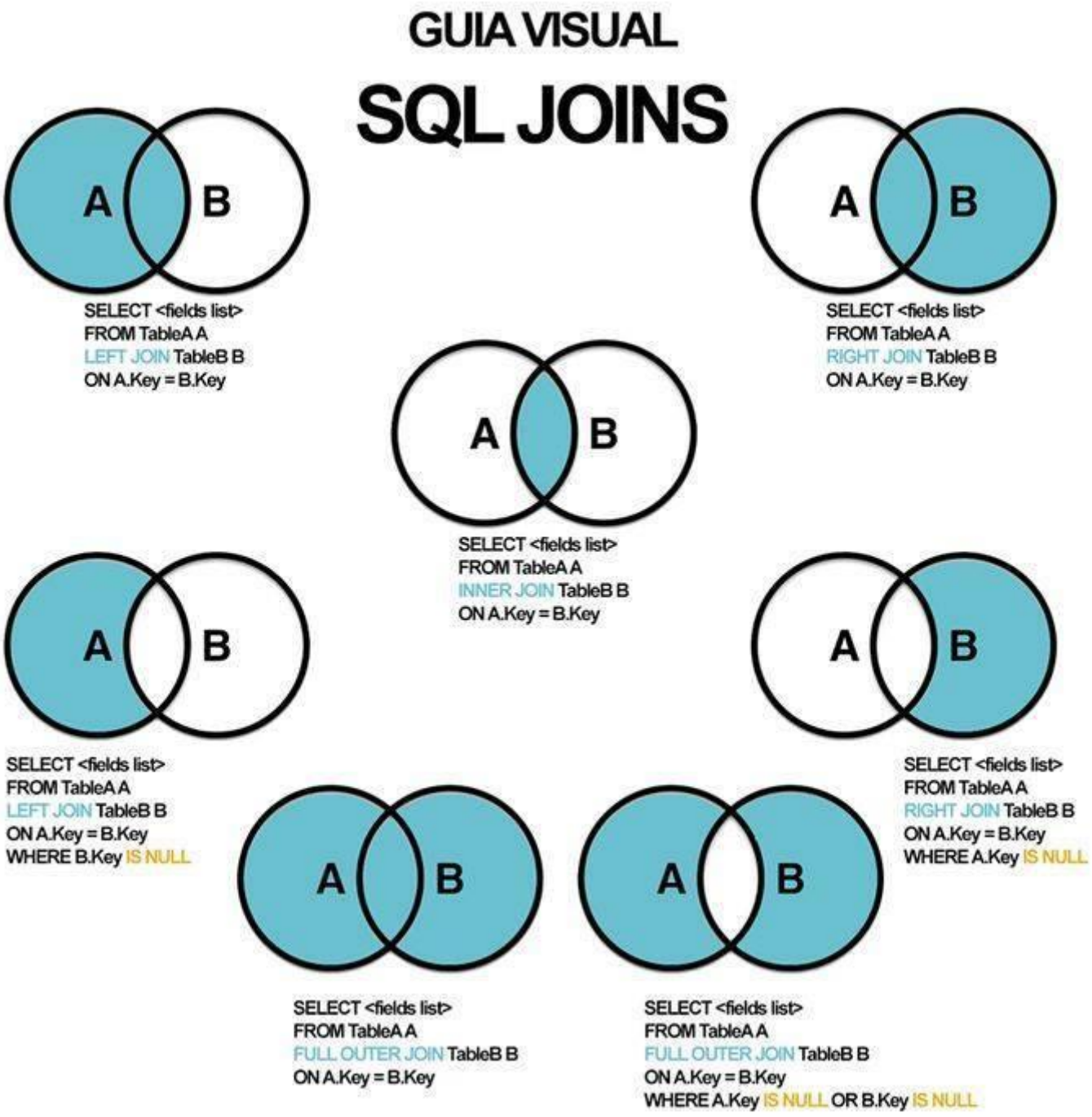
De esta forma mostraremos también los datos de los pistas que no estén relacionados con ninguna reserva. Hay que tener en cuenta que, sólo en este caso, es relevante el orden en el que se especifican las tablas a la hora de definir el JOIN puesto que se incluirán *aquellas filas de la tabla del lado izquierdo que no tengan relación con las de la tabla del lado derecho*. Es por ello que en los INNER JOIN no se tiene que indicar el sentido de la unión.

```sql
-- Mostrar cuántas veces se ha reservado cada pista
SELECT P.id, P.codigo, P.tipo, COUNT(R.id) AS reservas
FROM pistas P LEFT OUTER JOIN reservas R ON P.id = R.id_pista
GROUP BY P.id
ORDER BY P.codigo;
```

```sql
-- Mostrar cuántas reservas ha hecho cada usuario
-- (Es posible que algún usuario no haya hecho reservas. Ver E-R)
SELECT U.dni, U.nombre, U.apellidos, COUNT(R.id) AS numero_reservas
FROM usuarios U LEFT OUTER JOIN usuario_reserva UR ON U.id = UR.id_usuario
        LEFT OUTER JOIN reservas R ON UR.id_reservas = R.id
        LEFT OUTER JOIN pistas P ON R.id_pista = P.id;
```

En definitiva, a la hora de construir una consulta *SQL* hay que añadir en la cláusula FROM todas aquellas tablas que estén involucradas en la consulta, bien porque se muestre alguna de sus columnas en la cláusula SELECT, porque se establezca alguna condición con WHERE, se agrupe por alguno de sus campos o incluso simplemente dicha tabla haga de *puente* entre dos tablas que deban estar involucradas en dicha consulta.

Con respecto al número de tablas que pueden ser incluidas en un *JOIN*, hay que tener en cuenta que el límite en *MySQL* es de 61.



Unión e intersección de consultas

La unión de consultas permite unir los resultados de dos consultas totalmente diferentes como si fuera el de una sola. Se realiza mediante la instrucción UNION y muestra los resultados sin repeticiones.

```
-- Código y tipo de las pistas abiertas y cerradas, indicando
-- el estado actual
```

```
SELECT 'abierta', codigo, tipo
FROM pistas
WHERE id IN (SELECT id_pista FROM pistas_abiertas)
UNION
SELECT 'cerrada', codigo, tipo
FROM pistas
WHERE id IN (SELECT id_pista FROM pistas_cerradas);
```

La intersección de consultas muestra sólo los valores que aparecen en las dos consultas que se intersectan. Se realiza mediante la instrucción INTERSECT:

```
-- Ciudades con polideportivos que cuentan con usuarios registrados
SELECT ciudad
FROM usuarios
INTERSECT
SELECT ciudad
FROM polideportivos;
```

Consultas usando Vistas

La creación de vistas permite almacenar consultas como si se trataran de nuevas tablas con la finalidad de utilizar el resultado de las mismas en otras consultas más complejas. Cuando se crea una vista se genera lo que se conoce como una 'tabla lógica' que permite asignar un nombre al resultado de una consulta y utilizar ésta más adelante y siempre actualizada.

Hay que tener en cuenta que realmente la consulta que se ha creado como vista no se encuentra almacenada, sino que tiene que ser generada cada vez que se deba utilizar.

Veamos un ejemplo:

```
-- Vista que almacena la consulta que mostraría el número de
-- pistas que hay en cada polideportivo
CREATE VIEW pistas_por_polideportivo
AS
SELECT PP.id, PP.nombre, COUNT(*) AS cantidad
FROM polideportivos PP, pistas P
WHERE PP.id = P.id_polideportivo
GROUP BY PP.id
```

Si ahora suponemos que nos pidieran conocer el polideportivo o polideportivos que más pistas tiene, sólo tendríamos que realizar una consulta utilizando la vista creada anteriormente.

```
-- Nombre del polideportivo que más pistas tiene
SELECT nombre
FROM pistas_por_polideportivo
WHERE cantidad = (SELECT (MAX(cantidad)
          FROM pistas_por_polideportivo)
```

Funciones de MySQL

Funciones para cadenas de caracteres

- CHAR_LENGTH(str): Devuelve la longitud, en caracteres, de una cadena de texto

```
> SELECT CHAR_LENGTH('Esto es una cadena');
-> 18
```

- CONCAT(str1, str2, . . .): Concatena las cadenas de texto que se pasan como parámetros

```
> SELECT CONCAT('Esto', ' forma ', 'una cadena');
-> 'Esto forma una cadena'
```

- LOWER(str): Devuelve la cadena convertida a minúsculas

```
> SELECT LOWER('Bases de Datos Relacionales');
-> 'bases de datos relacionales'
```

- REPLACE(str, from_str, to_str): Reemplaza todas las ocurrencias de 'from_str' por 'to_str' que aparezcan en la cadena 'str'

```
> SELECT REPLACE('Access es un buen SGBD', 'Access', 'MySQL');
-> 'MySQL es un buen SGBD'
```

- SUBSTRING(str, pos), SUBSTRING(str, pos, len): Devuelve una subcadena de 'str' comenzando en la posición 'pos'. En el segundo caso tomará caracteres hasta completar una subcadena de tamaño 'len'

```
> SELECT SUBSTRING('MySQL es un buen SGBD', 7);
-> 'es un buen SGBD'

 > SELECT SUBSTRING('MySQL es un buen SGBD', 7, 2);
-> 'es'
```

- UPPER(str): Devuelve la cadena convertida a mayúsculas

```
> SELECT UPPER('Esto es una cadena');
-> 'ESTO ES UNA CADENA'
```

Funciones de fecha

- CURDATE(): Devuelve la fecha actual

```
> SELECT CURDATE();
-> '2012-01-09'
```

- CURTIME(): Devuelve la hora actual

```
> SELECT CURTIME();
-> '19:23:02'
```

- CURRENT_TIMESTAMP(): Devuelve la fecha y hora actual

```
> SELECT CURRENT_TIMESTAMP();
-> '2012-01-09 19:23:02'
```

- MONTH(fecha): Devuelve el número de mes para la fecha especificada (1-12)

```
> SELECT MONTH('1998-01-01 23:59:59');
-> 1
```

- MONTHNAME(fecha): Devuelve el nombre del mes para la fecha especificada

```
> SELECT MONTHNAME('1998-01-01 23:59:59');
-> January
```

- WEEKDAY(fecha): Devuelve el día de la semana para la fecha especificada (0-6)

```
> SELECT WEEKDAY('1998-01-01 23:59:59');
-> 3
```

- YEAR(fecha): Devuelve el número de año para la fecha especificada

```
> SELECT YEAR('1998-01-01 23:59:59');
-> 1998
```

- DAY(fecha): Devuelve el día del mes de la fecha especificada (1-31)

```
> SELECT DAY('1998-01-01 23:59:59');
-> 1
```

- DAYNAME(fecha): Devuelve el nombre del día de la semana de la fecha especificada

```
> SELECT DAYNAME('1998-01-01 23:59:59');
-> Thursday
```

- DATEDIFF(expr, expr2): Devuelve el número de días entre la fecha inicial 'expr' y la fecha final 'expr2'

```
> SELECT DATEDIFF(CURDATE(),'2021-01-09');
-> 365
```

- DATE_ADD(fecha, INTERVAL expr1 type), DATE_SUB(fecha, INTERVAL expr type): Realizan operaciones aritméticas con fechas

Como valor de 'type' se pueden especificar los partes de fecha con los que se quiera realizar el cálculo. Estos son algunos:

- SECOND: Segundos
- MINUTE: Minutos
- HOUR: Horas
- DAY: Días
- WEEK: Semanas

- MONTH: Meses
- YEAR: Años
- DAY_HOUR: Días Horas
- YEAR_MONTH: Años-Meses

```
> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY):
-> '1998-01-01 23:59:59'

 > SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'

 > SELECT DATE_ADD('1998-01-02', INTERVAL 1-1 YEAR_MONTH);
-> '1999-02-02'
```

Ejercicios

1. Con las siguientes tablas, realizar las consultas que se enumeran a continuación

Empleados (#id, nombre, apellidos, oficio, fecha_alta, salario, comision, -id_departamento)
Departamentos (#id, nombre, ubicacion)

- Nombre y apellidos de los empleados cuyo nombre empieza por 'A', ordenado por apellido
- Nombre, apellidos y departamento de los empleados que trabajan como 'Analista'
- Mostrar el salario medio de los empleados que trabajan como 'Programador'
- Nombre y apellidos de los empleados que no han conseguido ninguna comisión, ordenador por apellido
- Nombre, apellidos y oficio de los empleados que ganan menos de 1000 euros
- Oficio y fecha de alta de los empleados que trabajan en el departamento de 'Ventas', ordenado por fecha de alta
- Nombre, apellidos y antigüedad de todos los empleados
- ¿Cuántos empleados se han contratado este año?

2. Dadas estas tablas, realiza las siguientes consultas
   Empleados (#id, dni, nombre, salario)
   Vendedores (#id, nro_vendedor, zona, -id_empleado)
   Polizas (#id, nro_poliza, importe, beneficiario, -id_vendedor, fecha, fecha_vencimiento)
   Empleado_Jefe (#(-id_empleado, -id_jefe))

   - Nombre de los vendedores de la zona Norte
   - Nombre del jefe del vendedor con número de vendedor 123456
   - ¿Cuántos vendedores hay en la zona norte?
   - Número de vendedores cuyo nombre empieza por 'A'
   - ¿Cuántos vendedores tienen jefe?
   - ¿Cuántos vendedores no tienen jefe?
   - Nombre de los empleados cuyo jefe es José Pérez
   - Importe total de las pólizas vendidas por vendedores de la zona Norte
   - Nº de póliza e importe de las pólizas vendidas por los vendedores cuyo jefe es Pablo Collado

- ¿Cuántas pólizas se vendieron cada mes del año pasado?
- Número e importe de las pólizas que vencen mañana

3. Dadas estas tablas, realiza las siguientes consultas

   Comunidades (#id, nombre, nro_habitantes)
   Ciudades (#id, nombre, nro_habitantes, -id_comunidad)
   Rios (#id, nombre, caudal, longitud)
   Comunidad_Rio (#(-id_comunidad, -id_rio), nro_kilometros)
   Rio_Ciudad (#(-id_rio, -id_ciudad))

   - Nombre y número de habitantes de las ciudad de La Rioja
   - Nombre y longitud de los ríos que pasan por la Comunidad de Aragón
   - Nombre de los ríos que pasan por Castilla la Mancha y Andalucía (por ambas)
   - Caudal total de los ríos que pasan por Navarra
   - ¿Cuál es el río más largo de España?
   - ¿Por cuántas ciudades pasa el río Ebro?
   - ¿Cuál es el río que más ciudades atraviesa?
   - Nombre de los ríos que sólo cruzan una ciudad
   - Número de ciudades que atraviesa cada río

4. Realiza las siguientes consultas sobre estas tablas

   Autores (#id, nombre, fecha_nacimiento, fecha_fallecimiento, nacionalidad)
   Obras (#id, titulo, fecha, -id_museo)
   Museos (#id, nombre, direccion, ciudad, pais)
   Obra_Autor (#(-id_obra, -id_autor))

   - ¿Cuántas obras hay en el museo del Prado?
   - ¿Cuántas obras ha creado Pablo Picasso?
   - Título de las obras que hay en el museo Reina Sofía, ordenado por título
   - Nombre y dirección de los museos de España, ordenado por nombre
   - Título de la obra y autor de aquellas obras de autores extranjeros que se encuentran en España, ordenado por autor
   - ¿Cuántas obras las han realizado autores españoles?
   - ¿Cuántos museos hay?
   - ¿Cuántas obras hay en cada museo?
   - Nombre y fecha de creación de las obras creadas antes del año 2000, ordenado por fecha
   - Nombre y edad de los autores que siguen vivos en la actualidad
   - Nombre y edad a la que murieron los autores ya fallecidos
   - ¿En qué mes nació Picasso?
   - Para cada obra de arte del museo del Prado, mostrar el título y el mes y año de creación