Módulo SQL – Base de datos

Contenidos:

- Introducción
- Diseño de bases de datos:
  - Requerimientos
  - Modelo Entidad-Relación
  - Paso a tablas
  - Creación de tablas
- Tratamiento de datos y consultas
  - Modificar Información
  - Consulta de registros
  - Consultas con varias tablas

Diseño de base de datos:

Conceptos básicos:

- **Campo o atributo:** Área de almacenamiento para almacenar datos de un tipo específico. No pueden almacenarse tipos diferentes una vez definido.
- **Registro o tupla:** Colección de datos relacionados. Dichos datos pueden ser de diferentes tipos.
- **Tabla o archivo:** Colección de registros.

Diseño de base de datos:

Conceptos básicos:

Tabla o archivo

Campo o atributo

Id	Marca	Modelo	Matricula	Color
1	Seat	Ibiza	7777 HYX	Rojo
2	Honda	Accord	2567 FGH	Verde
3	Peugeot	208	1946 LJC	Amarillo
4	Ford	Mustang	9951 KTR	Azul

Registro o tupla

Tratamiento de datos y consultas

Modificar información

Modificar información

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
```

Modificar información

Ejemplos:

```
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',
```

```
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');
```

Modificar información

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 ]
```

Modificar información

Modificación de registro

- Actualizar una columna de una fila

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

- Actualizar varias columnas de una fila

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

- Actualizar una columna de varias filas

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

Modificar información

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 ]
```

Modificar información

Eliminación de registro

- **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';
```

Modificar información

Eliminación de registros

OJO! Si no indicamos el comando where, nos borrará todos los registros de la tabla en cuestión.

https://www.youtube.com/watch?v=i_cVJglz_Cs

Tratamiento de datos y consultas

Consulta de registros

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] ]
```

Consulta de registros

SELECT . . . FROM . . .

La cláusula SELECT se utiliza para seleccionar 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.

Modificar información

SELECT

- **Campo o atributo:** Área de almacenamiento para almacenar datos de un tipo específico. No pueden almacenarse tipos diferentes una vez definido.

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

- **Tabla o archivo:** Colección de registros.

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

Consulta de registros

Funciones agregadas

- **COUNT:** Devuelve el número de filas seleccionadas

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

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

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

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

Consulta de registros

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.

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

Consulta de registros

WHERE:

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 = TRUE AND pistas.tipo = 'tenis';
```

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'
```

Consulta de registros

Subconsultas:

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

```sql
-- Código y tipo de la pista más barata
SELECT codigo, tipo
FROM tipo
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)
```

Consulta de registros

GROUP BY / HAVING:

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

```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;
```

Consulta de registros

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;
```

Consulta de registros

Funciones agregadas

- **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;
```

Consulta de registros

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

Consulta de registros

Operadores:

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

```sql
-- 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%';
```

Consulta de registros

Operadores:

- 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.

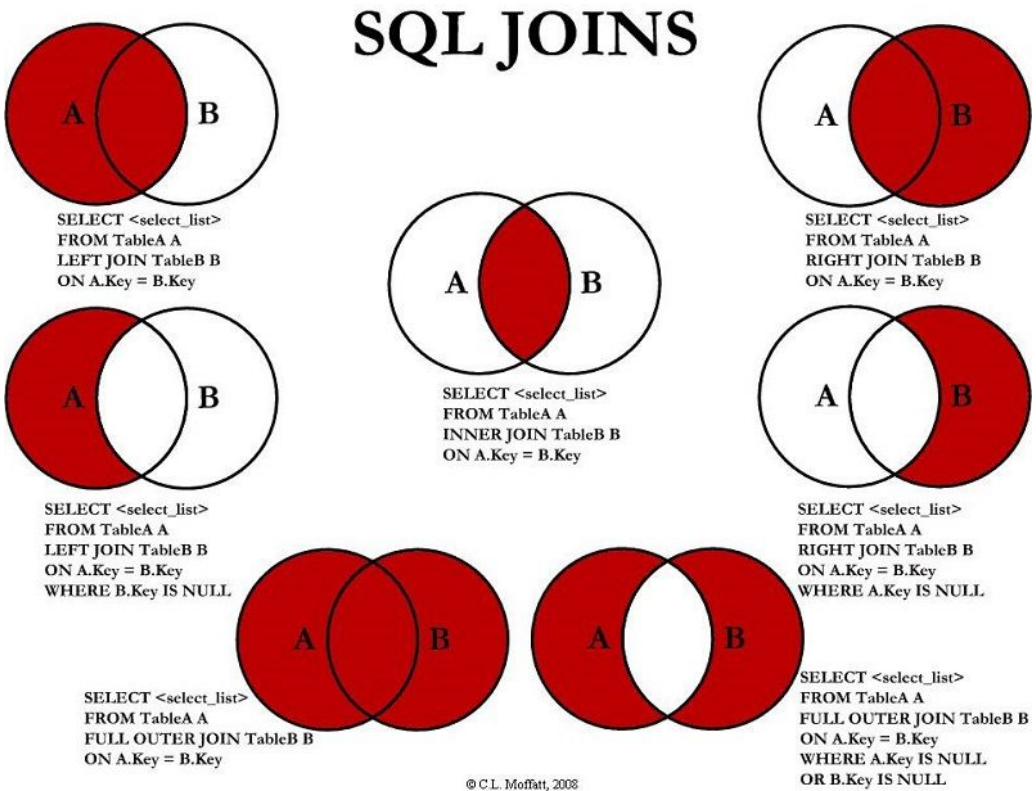
```
-- Nombre y apellidos de los usuarios que no indicaron su fecha de nacimiento
SELECT nombre, apellidos
FROM usuarios
WHERE fecha_nacimiento IS NULL;
```

- 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 0.1 AND 0.2;
```

Final Boss

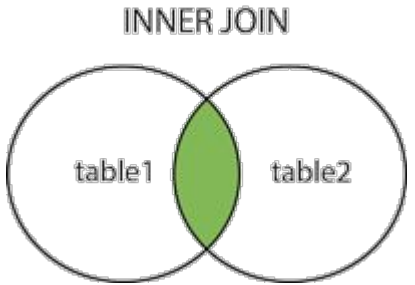
Consulta de registros: Consultas con varias tablas:



Consulta de registros: 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).

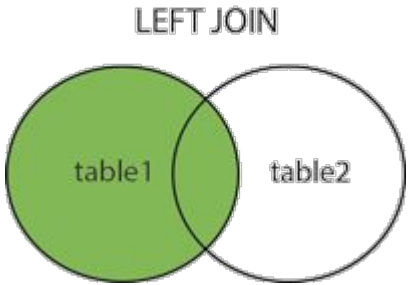
```sql
-- Mostrar, para cada polideportivo, el código y tipo de las pistas
-- que tiene
SELECT PP.id, PP.nombre, P.codigo, P.tipo
FROM polideportivos PP, pistas P
WHERE PP.id = P.id_polideportivo

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

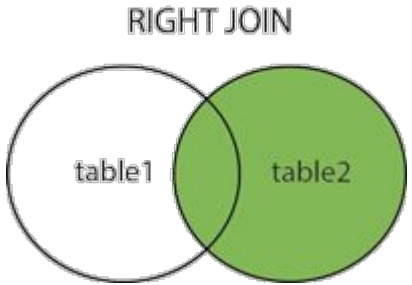


Consulta de registros: Consultas con varias tablas:

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





Consulta de registros: Consultas con varias tablas:

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

Consulta de registros: Consultas con varias tablas:

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;
```

Consulta de registros: Consultas con varias tablas:

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

Consulta de registros: Consultas con varias tablas:

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

Consulta de registros: Consultas con varias tablas:

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);
```

Consulta de registros: Consultas con varias tablas:

Unión e intersección de consultas

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;
```

Fin