PHP-TUT-23 Bases de datos SQL (II)

En el artículo anterior hemos aprendido a crear una BBDD y las tablas que la forman. Ahora llega el momento de introducir datos.En este artículo vamos a conocer las consultas SQL más empleadas habitualmente para la inserción de datos, la modificación y eliminación de los mismos y, por supuesto, la consulta de estos, según los criterios que podamos necesitar en cada caso.

CONSULTAS DE DATOS

Vamos a trabajar con la tabla de empleados que definimos anteriormente. Repito aquí la creación de la misma, para que tengas los datos a mano. Copio la última definición de la estructura de la tabla para que podamos verla con todos los datos, con un ligero cambio para mejorar las capacidades didácticas del ejemplo:

```
CREATE TABLE IF NOT EXISTS empleados (
 numero INT NOT NULL,
 nombre VARCHAR (50),
 salario FLOAT (6,2) ZEROFILL,
 categoria CHAR (30) DEFAULT NULL,
 sexo ENUM (?M','F'),
 departamento CHAR (2),
PRIMARY KEY (numero),
FOREIGN KEY (departamento) REFERENCES departamentos
ON UPDATE CASCADE
ON DELETE SET NULL
) TYPE=InnoDB;
```

INSERCIONES

Para añadir registros a una tabla usamos la consulta INSERT INTO, que inserta un registro en la tabla que se especifique, colocando los valores que se indican en los campos correspondientes. La sintaxis genérica de esta consulta es la que se muestra a continuación:

```
INSERT INTO tabla (campo_1, campo_2, campo_3,?, campo_n) VALUES (valor_1, valor_2, valor_3, ?, valor_n);
```

Por ejemplo, para añadir el registro de un empleado a la correspondiente tabla usaríamos la siguiente consulta:

```
INSERT INTO empleados (numero, nombre, salario, categoria, sexo, departamento) VALUES (1, "Pedro García Fernández", 700, "Limpiador", "M", "AL");
```

En ocasiones se complementan, como en el ejemplo que acabamos de ver, todos los campos del registro. En ese caso SQL permite omitir los nombres de los campos. La consulta anterior podría, por lo tanto, haber quedado así:

```
INSERT INTO empleados VALUES (1, "Pedro García Fernández", 700, "Limpiador", "M", "AL");
```

Es imprescindible, en este caso, que los valores que aparecen estén en el mismo orden que los campos cuando se creó la tabla, ya que SQL respeta este orden y asigna los datos uno a uno.

También puede ser que no vayamos a grabar todos los campos del nuevo registro. Por ejemplo, en la estructura existen campos que admiten el valor NULL, es decir, que se permite crear el registro dejando ese campo sin rellenar si es necesario. Por ejemplo, suponte que aún no sabemos la categoría profesional con la que vamos a contratar a nuestro nuevo empleado. En un caso así, sí es obligatorio especificar la lista de campos que van a cumplimentarse. La consulta quedaría así:

```
INSERT INTO empleados (numero, nombre, salario, sexo, departamento) VALUES (1, "Pedro García Fernández", 700, "M", "AL");
```

Como ves, esta vez no se asigna ningún valor al campo categoria. Esto no da ningún problema. Si miras la definición de la estructura de la tabla, verás que el campo admite el valor NULL y lo toma por defecto. Pero sí es imprescindible que aparezca la relación de campos que vamos a cumplimentar para que SQL asigne cada valor al campo correcto.

Además, es necesario especificar la lista de nombres de campos en esta consulta si alguno de los campos numéricos se creó con el atributo AUTO_INCREMENT, ya que a un campo así no le asignaremos valor alguno al crear el registro. Es el propio motor de la BBDD el que se encarga de colocar el valor correspondiente en dicho campo. No olvides que estos campos se llenan

automáticamente con valores numéricos secuenciales. Como norma, para evitar problemas, especifica siempre la lista de campos en estas consultas.

RECUPERACIÓN DE DATOS

Una vez que se hayan grabado registros en una tabla es muy posible que quieras localizar los datos de un registro determinado. Para ello, empleamos la consulta SELECT, que es muy versátil, lo que resulta interesante a la hora de localizar registros en grandes tablas, pero requiere una sintaxis muy amplia. En su forma más simple es como sigue:

```
SELECT campo_1, campo_2, ..., campo_n FROM tabla WHERE condición;
```

Esto recupera los campos especificados de la tabla indicada para aquellos registros que cumplan la condición. Por ejemplo, mira la consulta siguiente:

```
SELECT nombre, salario FROM empleados WHERE departamento = "AL";
```

Con esta consulta conseguiremos los nombres y salarios de todos los empleados que trabajen en el departamento especificado, por lo que del resto de los trabajadores no se obtendrán los datos solicitados.

Ahora piensa que lo que pretendemos es recuperar todos los datos de los registros que cumplan la condición especificada. No es, en este caso, necesario teclear la lista de campos. Basta con usar, en su lugar, un asterisco. La consulta quedará así:

```
SELECT * FROM empleados WHERE departamento = "AL";
```

Y, si lo que deseamos es recuperar todos los datos, o algunos de ellos, de todos los registros de la tabla, las consultas adecuadas serían las siguientes:

```
SELECT * FROM empleados;
```

o bien:

```
SELECT nombre, salario FROM empleados;
```

Al suprimir la condición que se establece en la cláusula WHERE, la consulta se ejecuta sobre la totalidad de filas.

La cláusula WHERE, que permite establecer condiciones de selección de registros, es especialmente potente y flexible. Hablaremos más adelante de ella en profundidad. De momento, para los ejemplos que aparecen aquí haremos un uso simple de ella, tal como hemos visto hasta ahora. Más adelante, veremos cómo se pueden combinar condiciones mediante operadores y otras técnicas para lograr consultas tan precisas como sean necesarias.

A menudo, sobre todo si se trata de obtener informes a partir de los registros de una tabla, es conveniente poder agrupar éstos mediante ciertos criterios. Para ello, empleamos la cláusula GROUP BY. Suponga, por ejemplo, que desea obtener un listado de los nombres de los empleados, y sus salarios, agrupados por el departamento al que pertenecen. La consulta será la siguiente:

```
SELECT nombre, salario FROM empleados GROUP BY departamento;
```

Con esto obtendremos el nombre y el salario de todos los trabajadores (no hay cláusula WHERE, aunque podría haberla si la necesitáramos), agrupados por departamento. Esto facilita mucho determinadas tareas. Por ejemplo, imagina que deseas saber el coste mensual en salarios de cada departamento. SQL te proporciona la cláusula SUM? AS, que permite obtener sumas de valores numéricos. Mira la siguiente consulta:

```
SELECT nombre SUM (salario) AS totalSueldos FROM empleados GROUP BY departamento;
```

Con esto lograremos que los registros de los empleados aparezcan agrupados por departamentos, bajo el titular totalSueldos, con un subtotal de los salarios para cada uno de los grupos. De este modo podríamos determinar el coste de la masa salarial de la empresa, por departamentos.

Otra operación a la que tenemos acceso es la posmediación aritmética de valores numéricos. Imagina que, a efectos estadísticos, deseamos calcular cuál es el sueldo medio de los empleados, es decir, sumando el total y dividiendo entre el número de empleados. Para ello, usaremos la cláusula AVG?AS (procedente de la palabra inglesa AVERAGE, que significa promedio). Mira la siguiente consulta:

```
SELECT nombre AVG (salario) AS media FROM empleados GROUP BY departamento;
```

En ocasiones necesitamos obtener los datos de una tabla ordenados según un criterio, pero sin que se produzca una agrupación por bloques. Para ello, contamos con la cláusula ORDER BY. A su vez, la ordenación puede ser ascendente (de menor a mayor), si no le añadimos ningún modificador o (dependiendo del motor de BBDD empleado) si añadimos el modificador ASC. Para lograr una ordenación descendente (de mayor a menor) recurriremos al modificador **DESC**. Observa la siguiente consulta:

```
SELECT nombre, salario FROM empleados ORDER BY salario DESC;
```

Con esta consulta obtendremos una relación de los nombres y salarios de los empleados, ordenados según el salario, desde el que cobra más al que menos. Como puedes comprobar, el hecho de que un campo forme parte de la selección no es óbice para que, además, forme parte de los criterios empleados. Es un rasgo que confunde a muchos principiantes al empezar a diseñar consultas.

SQL es muy flexible y, en poco tiempo, manejarás estos detalles con soltura.
Lógicamente, tanto la cláusula GROUP BY, como ORDER BY pueden admitir más de un criterio. Mira la siguiente consulta:

```
SELECT * FROM empleados ORDER BY departamento ASC, salario DESC;
```

Con esto obtendremos todos los datos de todos los registros, ordenados ascendentemente por el departamento donde trabaja cada empleado. A su vez, dentro de cada departamento, estarán ordenados, descendentemente, por el salario.
A menudo es interesante saber cuántos registros tiene un grupo, o cuántos cumplen una condición. Para ello, SQL nos proporciona la cláusula COUNT?AS, que permite el conteo de registros. Lo ves en el siguiente ejemplo:

```
SELECT COUNT (*) AS total FROM empleados, ORDER BY departamento;
```

Podemos establecer una condición basada en similitudes no exactas. Esto quiere decir que podemos crear un criterio sobre patrones. Veamos qué es esto. SQL nos proporciona el operador LIKE. Esta palabra, en inglés, es un comparativo que se refiere a la similitud entre dos cosas. Suponte que, en tu tabla de empleados, tienes personal en el departamento de contabilidad, cuyo código es CO y en control de calidad, cuyo código es CC. Además, como es lógico, tendrás gente en otros departamentos, pero quieres recuperar la lista de los empleados de estos dos que he mencionado. Mira la siguiente consulta:

```
SELECT nombre, salario FROM empleados WHERE departamento LIKE "C%";
```

El símbolo % que aparece en la cadena que hay a continuación de LIKE significa "cualquier carácter o conjunto de caracteres". Es decir, se recupera el nombre y el salario de la tabla de empleados para todos aquéllos cuyo departamento empiece por la letra C y, a continuación, tenga cualquier cosa. Para ver otro ejemplo, piensa que deseas ver la lista, con todos los datos, de los empleados cuyo nombre contiene José. La consulta será la siguiente:

```
SELECT * FROM empleados WHERE nombre LIKE "%José%";
```

Otra cláusula muy interesante es LIMIT, que permite acotar la cantidad de registros obtenidos en una consulta. Por ejemplo, suponte que hacemos una consulta de selección de los empleados que trabajan en el departamento de producción, y resulta que son, digamos, 300.
Manejar tantos datos simultáneamente es, cuanto menos, engorroso e incómodo, así que trabajaremos, de momento, con los diez primeros empleados. Luego, con los diez siguientes y, así, sucesivamente. De este modo el trabajo será más llevadero. Mira la siguiente consulta:

```
SELECT * FROM empleados WHERE departamento = "PR" LIMIT 10;
```

Esto recuperará los diez primeros registros. Para obtener los diez siguientes recurriremos a la consulta que aparece a continuación:

```
SELECT * FROM empleados WHERE departamento = "PR" LIMIT 11, 20;
```

Como ve, la cláusula LIMIT debe ir seguida por el primer registro que deseamos recuperar, una coma y el último que queremos. Entonces la consulta obtiene los que están comprendidos entre ambos límites.
Antes hemos mencionado que la cláusula WHERE es muy flexible. Operadores como LIKE le dan más potencia de la que pueda parecer en principio. Pero aún hay más.
Podemos usar operadores para encadenar varias condiciones bajo una sola cláusula WHERE. En SQL consideramos dos tipos de operadores: los **aritméticos** y los **lógicos**. La lista de operadores que podemos usar es la que aparece en la tabla siguiente:

OPERADORES DE LA CLÁUSULA WHERE (aritméticos)

OPERADOR
DESCRIPCIÓN

<

Mayor que.

>

Menor que.

=
Igual a.

<=
Menor o igual que.

>=
Mayor o igual que.

!=
Distinto de.

LIKE
Parecido a.

BETWEEN
Comprendido entre dos valores.

IN
Se usa para hacer una consulta en una tabla que está en otra BBDD distinta de la que estamos usando, o para buscar el valor en una lista de posibles valores.

OPERADORES DE LA CLÁUSULA WHERE (lógicos)

OPERADOR
DESCRIPCIÓN

AND
Une dos condiciones. Deben cumplirse ambas.

OR
Une dos condiciones de modo que seleccionará los registros que cumplan, al menos, una de ellas.

NOT
Selecciona los registros que no cumplan la condición.

Como ves son pocos pero, convenientemente combinados, tienen mucha potencia para el procesamiento de datos. Por ejemplo, supón que quieres obtener una lista de los empleados cuyo sueldo es mayor que 700 euros, pero menor que 900 y que, además, trabajan en el almacén, cuyo código de departamento es AL. La consulta será la siguiente:

```
SELECT * FROM empleados WHERE salario > 700 AND salario < 900 AND departamento = "AL";
```

Ésta sería una forma de hacerlo. Y, en realidad, funciona. Pero también podemos hacerlo de otro modo, mediante el uso del operador BETWEEN. La consulta quedaría así:

```
SELECT * FROM empleados WHERE (salario BETWEEN 700 AND 900) AND departamento = "AL";
```

El uso de paréntesis, en este caso, está destinado, simplemente, a facilitar la legibilidad de la consulta por nuestra parte. En otros casos pueden usarse paréntesis para agrupar condiciones.

Por ejemplo, suponte que queremos obtener la lista de los empleados que ganan 800 euros o más y que trabajan en el almacén, o bien que trabajan en contabilidad (cuyo código de departamento puede ser, por ejemplo, CO), ganen lo que ganen. Mira la siguiente consulta:

```
SELECT * FROM empleados WHERE (salario >= 800 AND departamento = "AL") OR departamento = "CO";
```

Ahora observe la siguiente consulta. Es muy parecida, pero no igual:

```
SELECT * FROM empleados WHERE salario >= 800 AND (departamento = "AL" OR departamento = "CO");
```

Esta última nos proporciona los datos de todos los empleados cuyo salario sea de 800 euros o más y cuyo departamento sea el almacén o contabilidad, pero no muestra aquellos de contabilidad que ganen menos de 800.

El operador IN permite efectuar una consulta sobre otra base de datos diferente de la que estamos usando en este momento, así:

```
SELECT * FROM tabla WHERE campo="valor" IN "BBDD alternativa";
```

Personalmente no soy partidario de esto, dado que, en la práctica, presenta problemas. Trabaja con las tablas de la BBDD que estás usando. Si tienes que cambiar de BBDD, hazlo.

Existe otro uso muy interesante del operador IN. Imagina que quieres una lista de empleados que estén en contabilidad (CO), almacén (AL) o ventas (VE). Podrías hacerlo así:

```
SELECT * FROM empleados WHERE departamento = "AL" OR departemento = "CO" OR departamento = "VE";
```

Esto te funciona y te da la lista que quieres, pero ahora observa esta alternativa.

```
SELECT * FROM empleados WHERE departamento IN ("AL", "CO", "VE");
```

Funciona exactamente igual y da el mismo resultado, pero con un código más compacto, elegante y legible. Eso sin mencionar el echo de la mayor facilidad si la consulta la tienes que construir dinámicamente en PHP, por ejemplo.

Como ves, la cláusula WHERE puede llegar a ser muy potente, permitiendo que se efectúen consultas de selección tan filtradas como necesitemos. Pero, además, esta cláusula está también disponible, con toda su potencia y flexibilidad para las consultas de actualización y eliminación de registros.

ACTUALIZACIÓN

A menudo es necesario cambiar el valor de un campo en uno o más registros. Por ejemplo, suponte que a un empleado le cambias de departamento, o le subes el sueldo. Para todos los cambios que necesitemos realizar en los datos de nuestra BBDD contamos con UPDATE. Observa la siguiente consulta genérica:

```
UPDATE tabla SET campo_1=nuevoValor_1, campo_2=nuevoValor2, ..., campo_n=nuevoValor_n WHERE criterio;
```

Por ejemplo, si deseas cambiar a un empleado, cuyo nombre conoces, del departamento de contabilidad al de facturación, cuyo valor es, digamos FA, la consulta será la siguiente:

```
UPDATE empleados SET departamento="FA" WHERE nombre="Ángel Rodríguez Macías";
```

Como ves, no hay ninguna dificultad en mantener las tablas actualizadas, según sea necesario.

Ahora supón que decides subir el sueldo un 5% a todos los empleados que, actualmente, cobren menos de 2000 euros. Mira la siguiente consulta:

```
UPDATE empleados SET salario = salario * 1.05 WHERE salario < 2000;
```

Por cierto, si eres empresario, ésta es una buena idea. Y si eres trabajador por cuenta ajena, te parecerá aún mejor.

ELIMINACIÓN DE REGISTROS

Bromas aparte, llega el momento de que aprendamos a eliminar registros de una tabla. Un empleado que se jubila, o se marcha de la empresa, por ejemplo, crearía esta necesidad. La consulta DELETE permite hacer, precisamente, esto. Su sintaxis genérica es la siguiente:

```
DELETE * FROM tabla WHERE criterio;
```

Por ejemplo, supón que se marcha un empleado, del que sabemos el nombre. La consulta adecuada sería la siguiente:

```
DELETE * FROM empleados WHERE nombre="Pedro Martín Puig-de la Torre";
```

SUBCONSULTAS

Por lo tanto, las consultas que podemos efectuar sobre los registros de una tabla son, a modo de resumen, las cuatro que aparecen a continuación:

INSERT INTO. Para añadir registros.
SELECT. Para localizar registros.
UPDATE. Para actualizar registros.
DELETE. Para eliminar registros.

Y ahora vamos a ampliar un poco nuestro espectro de trabajo. Vamos a volver al caso del empleado que se jubila o se marcha. No debería bastarnos con eliminar el correspondiente registro de la tabla. Deberíamos tener otra tabla, que podría llamarse, por ejemplo, ex_empleados. En ella almacenaremos los datos de los empleados que ya no formen parte de la plantilla. Sin embargo, con lo que tenemos hasta ahora, la tarea es, conceptualmente, bastante engorrosa:

En primer lugar, tenemos que buscar el registro del empleado que va a marcharse, en la tabla empleados, y anotar sus datos.
A continuación, lo añadimos a la tabla ex_empleados.
Por último, borramos el registro en empleados.

Afortunadamente, SQL permite manejar, en una misma consulta, campos de diferentes tablas de la misma BBDD. Para ello, basta con que precedamos el nombre de cada campo del de la tabla con la que queremos trabajar, separando ambos por un punto.

```
INSERT INTO ex_empleados (ex_empleados.numero, ex_empleados.nombre, ex_empleados.salario, ex_empleados.categoria, ex_empleados.sexo, ex_empleados.departamento) VALUES (SELECT * FROM empleados WHERE empleados.nombre= "Ana Márquez Contreras");
```

Con esto hemos copiado el registro de una persona, localizada por su nombre, de una tabla a la otra. Ahora ya sólo hay que eliminar el registro en la tabla original, así:

```
DELETE * FROM empleados WHERE nombre= "Ana Márquez Contreras";
```

Fíjate en que, dentro de la consulta de inserción, como valores, aparecen los resultados de una consulta de selección. Esto es lo que se conoce como **subconsultas**.

Recuerde que este artículo te ha mostrado las principales características de SQL. No trates de aprenderlas de memoria. Úsalo a modo de consulta cuando tengas dudas. Para saber cómo usar estas consultas en bases de datos reales en PHP, consulta este artículo.

NOTA. En estos post no hemos detallado todas las posibilidades de SQL (para eso haría falta un libro entero), sino sólo las más relevantes y habituales que cubrirán el 95 % de tus necesdidades cómo desarrollador.