ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Sentencias SQL
y
Transacciones

Aprenderemos las **sentencias principales** de **definición** de datos, de **manipulación** de datos.

Veremos ejemplos de cómo **obtener información** de nuestra fuente de datos, así como realizar consultas.

Recordaremos qué son las transacciones SQL.

Sentencias de definición de datos

Creación de base de datos

Estas sentencias son usadas en **base de datos MySQL**.

• **ALTER DATABASE**: Cambiar **características generales** de una **base de datos**. Estas características son almacenadas en un **archivo** que suele ser "**db.opt**". Para usar este comando se necesita el **permiso ALTER**.

• **ALTER TABLE**: Modificar la **estructura** de una **tabla** que previamente hemos creado. Se pueden realizar numerosas acciones como **agregar columnas**, **borrarlas**, crear **índices**, borrarlos, cambiar la **tipología** de ciertas **columnas**, **renombrar** las columnas, también modificar la **descripción** de la **tabla** y la **tipología** de la misma.

• **CREATE DATABASE**: Creación de una **nueva base de datos**, para ello es necesario disponer del **permiso CREATE**.

"**create_specification**": se **establecerán** las distintas **características** de la base de datos. También están almacenadas en el fichero con **extensión .opt** en la raíz de la base de datos. "**character_set**": se especificará el **set de caracteres por defecto** de dicha base de datos que se está creando.

• **CREATE INDEX**: En muchas de las bases de datos, este comando **se traduce** a un comando **ALTER TABLE** para la creación de índices. Normalmente, con la creación de la tabla, se agregan todos los **índices**, pero con este comando podemos **agregarlos manualmente después de haber creado una tabla**. Para el tipo de columnas numeradas (columna1, columna2, etc.), se crea un índice de columnas múltiples. Los **índices** se forman **al unir los valores de las columnas**.

Definición de datos

*Tal y como sabemos, el comando ALTER TABLE nos modifica una tabla de base de datos. Un dato a tener en cuenta es que las **columnas están numeradas** desde el número 1 en adelante.*

*Podemos utilizar el **parámetro POSICIÓN** si estamos añadiendo una **nueva columna** en la tabla, para indicarle en qué **posición** de la misma **queremos colocarla**.*

Creación y eliminación de tabla

• **CREATE TABLE**: **Creación** de una tabla con el nombre definido. Es evidente que también necesitaremos el **permiso CREATE** para la tabla. Existen algunas **restricciones** a la hora de establecer un nombre a la tabla definida. Ocurrirá un **error** si la tabla que estamos creando **existe previamente** en la base de datos que estamos trabajando. Se puede usar "**TEMPORARY**" y será solo visible **mientras tengamos activa la conexión** con la que estamos trabajando, después, dicha tabla no existirá.

También podremos usar las **claves** "**IF NOT EXISTS**" para asegurarnos de que cierta tabla no existe.

• **DROP DATABASE**: Realizar un **borrado permanente** de todas las **tablas** de nuestra base de datos y borrar, así, dicha **base de datos**. Es un comando muy peligroso; por ello tendremos que tener habilitado el **permiso de DROP**. Si la base de datos que estamos borrando está **enlazada simbólicamente**, se **borrarán ambos objetos**.

• **DROP INDEX**: Añadiendo el **nombre del índice** y la **tabla** especificada, se ejecutará un **ALTER TABLE** justo para **borrar el índice** que estamos indicando.

• **DROP TABLE**: Borrar **una o más tablas** en nuestra base de datos. El **permiso DROP** debe estar habilitado para nuestro usuario de la base de datos.

Es otro comando que deberá de ser ejecutado con mucha precaución, ya que toda la información de la tabla que estamos indicando será eliminada. También podremos usar la clave "**IF EXISTS**" para evitar el error de cuando la tabla no exista. Este comando DROP TABLE realizará **commit automáticamente** al ser ejecutado.

• **RENAME TABLE**: Renombrar **una o más tablas**. Una vez se está ejecutando el renombrado, la tabla quedará temporalmente **bloqueada hasta finalizar la transacción**.

Sentencias de manipulación de datos

Inserción

• **DELETE**:

Eliminamos las filas de “tbl_name” que validan la condición expresada en “where_definition”.

Este comando nos **devolverá el número de registros** eliminados. **Si no** establecemos clausula “**where**”, se **eliminarán todas** las filas de la tabla.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

• **DO**:

Ejecutaremos expresiones sin obtener ningún **resultado**. Realmente, es una forma de realizar **SELECT** (expresión), pero con la ventaja de que **es más rápido** cuando no es de interés el resultado.

```
DO expr [, expr] ...
```

• **HANDLER**: Accederemos directamente a las **distintas interfaces** del **motor de la tabla**, tendremos **comandos complementarios** como OPEN, READ o CLOSE para **leer ciertos datos** de dicha tabla.

• **INSERT**:

Agregar **nuevos registros** en una tabla previamente definida. En relación a la forma **INSERT-SET** o **INSERT-VALUES**, se insertarán **valores basados en las columnas** de la tabla. También podremos encontrar **INSERT-SELECT** cogiendo registros de **otra/s tabla/s**.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

• **LOAD DATA INFILE**: L**eer los registros**.

Edición

• **REPLACE**:

Misma función que INSERT con una excepción: se **sobrescribirán** los registros para los índices de tipo PRIMARY KEY o UNIQUE teniendo en cuenta que el **registro anterior se borra** antes de agregar el nuevo registro. Solo tendrá sentido, evidentemente, si la tabla contiene este tipo de índices.

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

Ejemplo:

```
CREATE TABLE test (
        id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        data VARCHAR(64) DEFAULT NULL,
        ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
        CURRENT_TIMESTAMP,
        PRIMARY KEY (id)
);

mysql> REPLACE INTO test VALUES (1, 'Old', '2014-08-20 18:47:00');

mysql> REPLACE INTO test VALUES (1, 'New', '2014-08-20 18:47:42');

mysql> SELECT * FROM test;
+----+------+---------------------+
| id | data | ts |
+----+------+---------------------+
| 1 | New | 2014-08-20 18:47:42 |
+----+------+---------------------+
```

• **SELECT**: Realizar **consultas** de registros de una o más tablas. Podremos realizar consultas **simples, complejas o subconsultas**. En el siguiente punto, dedicaremos todo un punto a estudiar las consultas y subconsultas, veremos su sintaxis y más detalles.

• **TRUNCATE**:

Sintaxis de TRUNCATE:

```
TRUNCATE TABLE tbl_name;
```

Eliminar completamente una tabla. Es equivalente a un **DELETE**, pero con ligeras diferencias. Dependiendo del motor de base de datos, en algunos se realiza un DELETE tal como lo hemos estudiado, se borran todos los registros de esa tabla; y para otros motores de base de datos, se elimina el objeto completo de la tabla y se vuelve a crear. Hay que tener en cuenta que este tipo de operaciones **no son transaccionales** (que estudiaremos más adelante), lo que significa que nos dará un **error si dicha tabla** está **ocupada** en ese momento.

• **UPDATE**:

Sintaxis de UPDATE:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name = expr1 [, col_name = expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

Actualizará la información de las columnas que le indiquemos en la sección **SET** con **información nueva**. Con la cláusula **WHERE** indicaremos qué registros deben actualizarse, y **si no** existe esta cláusula, se **modificarán todos**.

Ejemplo UPDATE:

```
UPDATE empleados
SET sueldo_bruto = '50000', prima_objetivos = '3000'
WHERE sueldo_bruto < 45000 AND sueldo_bruto > 40000
ORDER BY antiguedad DESC
LIMIT 50
```

- ➔ **Establecer** el sueldo bruto anual a 50.000 dólares y la prima de objetivos a 3.000.
- ➔ Esto será **efectivo a** los empleados que cobren entre 40.000 y 45.000 dólares.
- ➔ Se actualizarán los **primeros 50 empleados** (**LIMIT 50**) ordenados de **mayor a menor** antigüedad en la empresa.

Consultas

SELECT

Una consulta es realizar una **pregunta** a base de datos con una serie de criterios, y esta ejecutará una respuesta a dicha pregunta. Podremos consultar **una tabla o más de una**.

Existen distintas **cláusulas vinculadas a SELECT**: podemos encontrar cláusulas de tipo **HAVING**, también podemos encontrar **ORDER BY**, **UNION**... la sentencia SELECT se puede combinar de diversas formas.

Cuando realizamos una consulta con SELECT, lo que **obtenemos** es una **tabla ficticia**, una serie de **resultados que se relacionan** acorde a nuestros requisitos en la consulta. Esta tabla de la que hablamos, evidentemente, no persiste en disco ni nada por el estilo, se mantiene **en memoria mientras** la estamos **usando**.

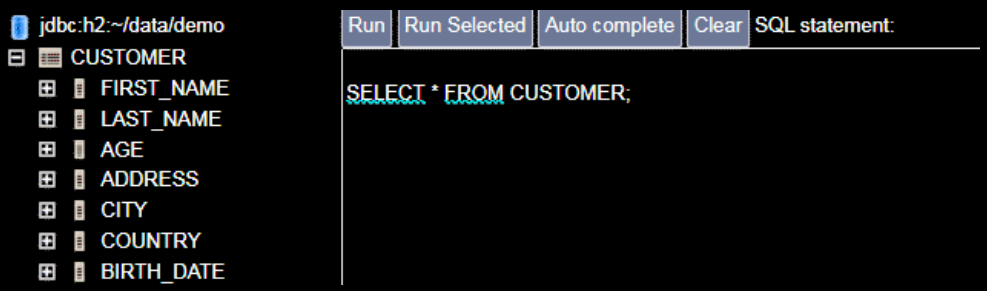
A continuación, veremos algunos de los **puntos principales** de la sintaxis de la sentencia **SELECT**:

• **SELECT**: Junto a la palabra clave "SELECT", podremos añadir [ALL/DISTINC], también asterisco [*] seguido de un listado de columnas [columnas], incluso podremos realizar un **alias** con la palabra clave **AS [nombre del alias]**. Con esta sintaxis podremos **seleccionar ciertas columnas** que van a ser mostradas junto con el orden.

Con **ALL** mostrará filas **duplicadas**.

Con **DISTINCT** incluirá en el resultado solo filas **únicas**.

• **FROM**: Usaremos la palabra clave "FROM" y a continuación el **nombre de** la **tabla/s** o **vista/s**. Hasta aquí, con los conceptos básicos que tenemos, podríamos ejecutar perfectamente una consulta básica pidiendo **todas las filas con todos sus campos** de una supuesta **tabla "CUSTOMER"**:



• **WHERE**: Especificaremos algunas **condiciones de filtro**. Usaremos dicha cláusula cuando **no queramos obtener el contenido total** de la tabla o tablas que estemos consultando. A continuación, veremos algunas de las condiciones que podremos agregar a nuestras consultas.

WHERE

Vamos a aprender a introducir algunas **condiciones** y **expresiones lógicas** en ella, de tal forma, que harán de **filtro** a la hora de preguntar a nuestra base de información y obtención de resultados.

Algunas **condiciones**:

• **Operadores**: podemos encontrar mayor que ("**>**"), mayor o igual que ("**>=**"), menor que ("**<**"), igual ("**=**"), distinto que podemos expresarlo: ("**<>**") o también: ("**!=**").

• **Nulos**: podremos añadir **IS NULL** o **IS NOT NULL** si lo que queremos es comprobar que el valor de cierta columna sea nulo o no. Un valor es nulo cuando **no existe valor**, un registro **en blanco no sería nulo**.

• **LIKE**: realiza una **comparación con los registros**. Se usarán caracteres especiales. Estos caracteres son "_" y también "%".

"**%**" indicamos que puede ir **cualquier cadena de caracteres** en esa posición.

"_" sería el mismo concepto, pero, en este caso, **solo con un carácter**.

Ejemplos:

• **WHERE** APELLIDO **LIKE** 'C%': mostraremos resultados que coincidan donde el apellido empiece por "C" seguido que cualquier cantidad de caracteres.

• **WHERE** APELLIDO **LIKE** 'A_': en este caso, el apellido empezaría por la letra "A" y luego le seguiría 1 solo carácter.

• **BETWEEN**: **Rango de valores.**

Ejemplo:

• **WHERE** EDAD **BETWEEN** 3 AND 7: mostraremos aquellas edades que estén entre 3 y 7, incluyendo dichos valores.

• **IN ()**: Mostrar una serie de resultados cuyos **valores coincidan** con los especificados en la **clave**. Ejemplo:

• **WHERE** EDAD **IN** (3, 4, 7, 8): en este caso mostraremos, por ejemplo, los usuarios cuya **edad coincida con las especificadas** en la clave IN.

Ejemplos con WHERE

Podremos **combinarlos con operadores lógicos OR** (o), **AND** (y) y **NOT** (negativa), y ayudarnos de los **paréntesis** para establecer **prioridades** entre ellos.

• **ORDER BY**: **Después del WHERE.** Como su nombre indica, la usaremos para establecer un orden a la hora de mostrar resultados según el campo o campos por los que queramos ordenar.

Podremos ayudarnos de **ASC** o **DESC** si lo que deseamos es mostrar los resultados en orden **ascendente** en el caso ASC (de menor a mayor) o mostrarlos en orden **descendente** en el caso de DESC (de mayor a menor).

Ejemplos de consultas.

En esta consulta, estamos mostrando **todas las columnas de la tabla "CUSTOMER"**:

```
SELECT * FROM CUSTOMER;
```

FIRST_NAME	LAST_NAME	AGE	ADDRESS	CITY	COUNTRY	BIRTH_DATE
juan	Lopez	10	avd jaen	Badajoz	Spain	1980-06-04 00:00:00
pepe	Tejada	20	avd 1	Sevilla	Spain	1980-06-04 00:00:00
victor	Fernandez	22	avd 2	Jaen	Spain	1960-06-04 00:00:00
jose	Assensio	40	avd 3	Huesca	Spain	1930-06-04 00:00:00
Leo	Perez	30	avd 4	Orense	Spain	2000-06-04 00:00:00
Patricia	Aranda	11	avd hoho	Barceloona	Spain	1988-06-04 00:00:00
David	Lopez	105	avd peugeot	Madrid	Spain	1966-06-04 00:00:00
Enrique	Ferreras	70	avd jujer	Malaga	Spain	1980-06-04 00:00:00
Seluis	Guzman	45	avd Madrid	Valencia	Spain	1963-06-04 00:00:00

(9 rows, 8 ms)

Consulta seleccionando todos los campos de la tabla CUSTOMER, **filtrando** por aquellos cuyo campo "LAST_NAME" **empiece por** "L" **o por** "F" y, además, **ordenados de mayor a menor** por el campo "AGE":

```
SELECT *
FROM CUSTOMER
WHERE (LAST_NAME LIKE 'L%') OR (LAST_NAME LIKE 'F%')
ORDER BY AGE DESC;
```

FIRST_NAME	LAST_NAME	AGE	ADDRESS	CITY	COUNTRY	BIRTH_DATE
David	Lopez	105	avd peugeot	Madrid	Spain	1966-06-04 00:00:00
Enrique	Ferreras	70	avd jujer	Malaga	Spain	1980-06-04 00:00:00
victor	Fernandez	22	avd 2	Jaen	Spain	1960-06-04 00:00:00
juan	Lopez	10	avd jaen	Badajoz	Spain	1980-06-04 00:00:00

(4 rows, 7 ms)

Transacciones

Son unidades o **conjuntos de acciones** que se realizan en **serie y de forma ordenada** en el sistema gestor de base de datos.

Los **objetivos**:

- Proporcionar **consistencia** en la base de datos realizando **secuencias de alta fiabilidad**, de tal forma que se pueda **volver a estados anteriores** fácilmente.
- Ofrecer **aislamiento** cuando más de un aplicativo está **accediendo** a los datos **simultáneamente**.

Comandos de control que se realizan para la ejecución de transacciones en SQL:

- **Commit**: con este comando se **persistirán** los cambios en base de datos.
- **Rollback**: **desharemos** los cambios que se hubieran ejecutado hasta el momento y se **abandonará la transacción**.
- **Savepoint**: puntos donde se podrá almacenar y, en caso de **rollback**, se podrá **volver a dicho punto** de control.

La interfaz Statement

Esta interfaz es usada **cuando se realiza una conexión con un driver** de una base de datos.

Es la **encargada de ejecutar sentencias** en nuestra aplicación y **recoger los resultados** para manipularlos más tarde.

Una vez se crea el **objeto Statement**, disponemos de un lugar adecuado para **realizar consultas SQL**. Podremos usar diferentes **métodos** para ello:

- **executeQuery (String)**: **Realiza sentencias SELECT**, siendo consultas que como resultado, este método nos **devolverá un objeto ResulSet** con toda la información resultante.
- **executeUpdate (String)**: Realiza **sentencias de manipulación** de datos, ya sean **INSERT**, **DELETE**, **UPDATE**, etc. Una vez ejecutada la sentencia que le indiquemos, como String, nos **devolverá un entero** que contiene la **cantidad de filas** que han sido **afectadas** en la operación.
- **execute (String)**: **Ejecuta cualquier acción query o update**.

  Devolverá **true** si **devuelve un Resulset**, y para acceder a él, tendremos que ejecutar el método **getResulSet ()**.

  Devolverá **false**, si lo que estamos ejecutando, por ejemplo, es un **UPDATE.** En ese caso, si queremos **saber las filas afectadas** consultaríamos el método **getUpdateCount ()**.