Contenido

1. INTRODUCCIÓN 1

2. LENGUAJE DE MANIPULACIÓN DE DATOS: DML 3

2.1. Inserción de datos 3

2.2. Modificación de datos 5

2.3. Eliminación de datos 6

2.4. Notas a tener en cuenta 7

2.4.1 Sobre las Vistas 7

1. INTRODUCCIÓN

El DML (*Lenguaje de Modificación de Datos*) es una de las partes fundamentales del lenguaje SQL. Lo forman las instrucciones capaces de modificar (añadir, cambiar o eliminar) los datos de las tablas.

Al conjunto de instrucciones DML que se ejecutan consecutivamente, se le llama **transacción**. Lo interesante de las transacciones es que podemos anularlas, ya que forman una unidad lógica de trabajo que hasta que no se acepten, sus resultados no serán definitivos. Las transacciones las veremos más adelante.

En todas las instrucciones DML, el único dato devuelto por el sistema es el número de filas que se han modificado al ejecutar la instrucción.

Antes de introducirnos en el estudio de las instrucciones INSERT, UPDATE y DELETE, hay que conocer como el SGBD gestiona las instrucciones de inserción, eliminación y modificación que podamos ejecutar, ya que hay dos posibilidades de funcionamiento:

- Que queden automáticamente validadas y no haya posibilidad de tirar atrás. En este caso, los efectos de toda instrucción de actualización de datos que tenga éxito son automáticamente accesibles desde el resto de conexiones de la base de datos.

- Que queden en una cola de instrucciones, que permite echar atrás. En este caso, se dice que las instrucciones de la cola están pendientes de validación, y el usuario debe ejecutar, cuando lo cree conveniente, una instrucción para validarlas (llamada COMMIT) o una instrucción para echar atrás (llamada ROLLBACK).

Este funcionamiento implica que los efectos de las instrucciones pendientes de validación no se ven por el resto de conexiones de la base de datos, pero sí son accesibles desde la conexión donde se han efectuado. Al ejecutar la COMMIT, todas las conexiones acceden a los efectos de las instrucciones validadas. En caso de ejecutar ROLLBACK, las instrucciones desaparecen de la cola y ninguna conexión (ni la propia ni el resto) no accede a los efectos correspondientes, es decir, es como si nunca hubieran existido.

Estos posibles funcionamientos forman parte de la gestión de transacciones que proporciona el SGBD y que hay que estudiar con más detenimiento. A la hora, pero, de

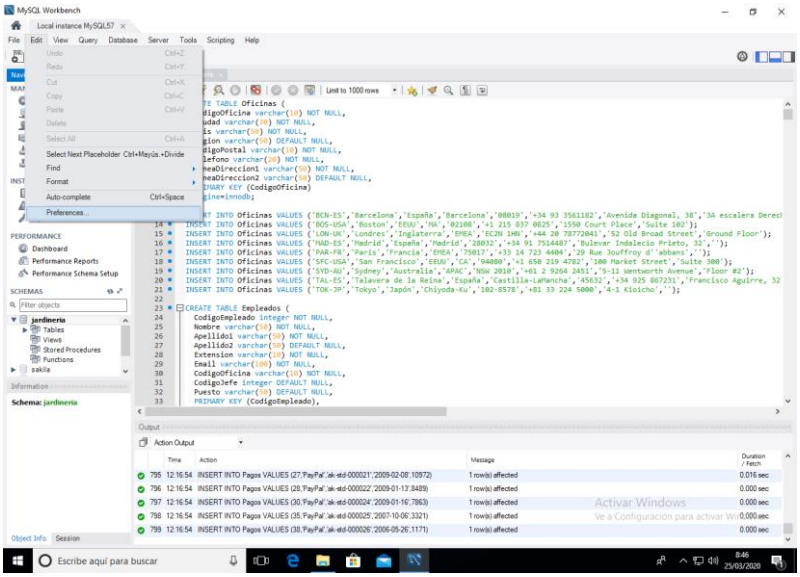
ejecutar instrucciones INSERT, UPDATE y DELETE debemos conocer el funcionamiento del SGBD para poder actuar en consecuencia.

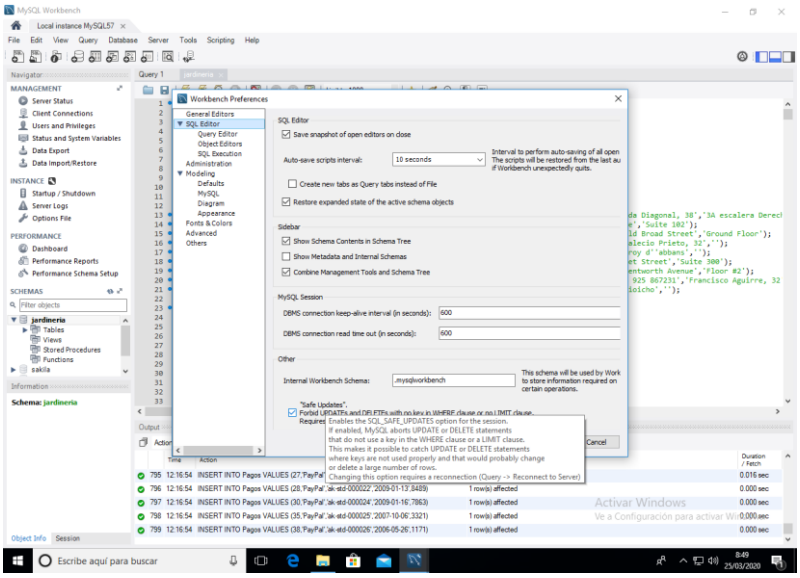
Así, por ejemplo, un SGBD MySQL funciona con validación automática después de cada instrucción de actualización de datos no se indica lo contrario y, en cambio, un SGBD Oracle funciona con la cola de instrucciones pendientes de confirmación.

En cambio, en MySQL, si se quiere desactivar la opción de autocommit que hay por defecto, habrá que ejecutar la siguiente instrucción:

```
SET autocommit = 0;
```

En nuestro caso, mantendremos la opción por defecto de MySQL y **desactivaremos la opción de "Safe Updates".** Ello permitirá que nuestras sentencias UPDATE o DELETE se ejecuten sin *seguridad* (aunque sea eliminar todos los registros de una tabla) y podamos hacer las prácticas de forma fluida.





2. LENGUAJE DE MANIPULACIÓN DE DATOS: DML

Una vez que se ha creado de forma conveniente las tablas, el siguiente paso consiste en insertar datos en ellas, es decir, añadir tuplas. Durante la vida de la base de datos será necesario, además, borrar determinadas tuplas o modificar los valores que contienen. Los comandos de SQL que se van a estudiar en este apartado son **INSERT, UPDATE y DELETE**. Estos comandos pertenecen al **DML**.

2.1. Inserción de datos

El comando INSERT de SQL permite introducir datos en una tabla o en una vista de la base de datos (una vista es una consulta que se "almacena" y se puede ejecutar cuando se desee). La sintaxis del comando es la siguiente:

```
INSERT INTO {nombre_tabla | nombre_vista } [(columna1 [,
columna2]...)]
VALUES (valor1 [, valor2] ... );
```

Indicando la tabla se añaden los datos que se especifiquen tras el apartado VALUES en un nuevo registro. Los valores deben corresponderse con el orden de las columnas. Si no es así se puede indicar tras el nombre de la tabla y entre paréntesis.

Ejemplos:

Supongamos que tenemos el siguiente diseño físico de una tabla:

```
CREATE TABLE EMPLEADOS (
  COD       NUMBER(2)     PRIMARY KEY,
  NOMBRE    VARCHAR2(50) NOT NULL,
  LOCALIDAD VARCHAR2(50) DEFAULT 'Écija',
  FECHANAC  DATE
);
```

La forma más habitual de introducir datos es la siguiente:

```
INSERT INTO EMPLEADOS VALUES (1, 'Pepe', 'Osuna', '01/01/1970');
INSERT INTO EMPLEADOS VALUES (2, 'Juan', DEFAULT, NULL);
INSERT INTO EMPLEADOS VALUES (3, 'Sara', NULL, NULL);
```

También, sería equivalente, podríamos hacerlo con un solo INSERT:

```
INSERT INTO EMPLEADOS VALUES (1, 'Pepe', 'Osuna', '01/01/1970'), (2,
'Juan', DEFAULT, NULL, (3, 'Sara', NULL, NULL);
```

Es obligatorio introducir valores para los campos COD y NOMBRE. Dichos campos no pueden tener valor NULL. Podemos insertar sólo el valor de ciertos campos. En este caso hay que indicar los campos a insertar y el orden en el que los introducimos:

```
INSERT INTO EMPLEADOS(NOMBRE, COD) VALUES ('Ana', 5);
```

Inserción de datos obtenidos de una consulta

También es posible insertar datos en una tabla que hayan sido obtenidos de una consulta realizada a otra tabla/vista u otras tablas/vistas. Su forma es:

```
INSERT INTO tabla

SELECT ...
```

Debe respetarse lo dicho anteriormente respecto a los campos. La consulta SELECT debe devolver la misma cantidad y tipo de campos que los definidos en la tabla.

Por ejemplo, suponiendo que disponemos de una tabla SOLICITANTES con el siguiente diseño:

```
CREATE TABLE SOLICITANTES (
  NUM         NUMBER(2) PRIMARY KEY,
  NOMBRE      VARCHAR2(50),
  CIUDAD      VARCHAR2(50),
  NACIMIENTO DATE,
  ESTUDIOS    VARCHAR2(50)
);


INSERT INTO EMPLEADOS
SELECT NUM, NOMBRE, CIUDAD, NACIMIENTO
FROM SOLICITANTES
WHERE ESTUDIOS='CFGS ASIR';
```

También podemos indicar los campos a insertar, teniendo en cuenta que, en este caso los campos COD y NOMBRE de la tabla EMPLEADO no aceptan valores NULL, por tanto es obligatorio introducir valores para ellos:

```
INSERT INTO EMPLEADOS(FECHANAC, NOMBRE, COD)
SELECT NACIMIENTO, NOMBRE, NUM
FROM SOLICITANTES
WHERE ESTUDIOS='CFGS ASIR';
```

2.2. Modificación de datos

Para la modificación de registros dentro de una tabla o vista se utiliza el comando UPDATE. La sintaxis del comando es la siguiente:

```
UPDATE {nombre_tabla | nombre_vista}
SET columna1=valor1 [, columna2=valor2] ...
[WHERE condición];
```

Se modifican las columnas indicadas en el apartado SET con los valores indicados. La cláusula WHERE permite especificar qué registros serán modificados.

Ejemplos:

```
-- Ponemos todos los nombres a mayúsculas
-- y todas las localidades a Estepa
UPDATE EMPLEADOS
SET NOMBRE=UPPER(NOMBRE), LOCALIDAD='Estepa';




-- Para los empleados que nacieron a partir de 1970
-- ponemos nombres con inicial mayúscula y localidades Marchena
UPDATE EMPLEADOS
SET NOMBRE=INITCAP(NOMBRE), LOCALIDAD='Marchena'
WHERE FECHANAC >= '01/01/1970';
```

Actualización de datos usando una subconsulta

También se admiten subconsultas. Por ejemplo:

```
UPDATE empleados
SET sueldo=sueldo*1.10
WHERE id_seccion = (SELECT id_seccion FROM secciones
                    WHERE nom_seccion='Producción');
```

Esta instrucción aumenta un 10% el sueldo de los empleados que están dados de alta en la sección llamada Producción.

2.3. Eliminación de datos

Es más sencilla que el resto, elimina los registros de la tabla que cumplan la condición indicada. Se realiza mediante la instrucción DELETE:

```
DELETE [ FROM ] {nombre_tabla|nombre_vista}
[WHERE condición] ;
```

Ejemplos:

```
-- Borramos empleados de Estepa
DELETE EMPLEADOS
WHERE LOCALIDAD='Estepa';


-- Borramos empleados cuya fecha de nacimiento sea anterior a 1970
-- y localidad sea Osuna
DELETE EMPLEADOS
WHERE FECHANAC < '01/01/1970' AND LOCALIDAD = 'Osuna';


-- Borramos TODOS los empleados;
DELETE EMPLEADOS;
```

Hay que tener en cuenta que el borrado de un registro no puede provocar fallos de integridad y que la opción de integridad ON DELETE CASCADE (clave secundaria o foránea) hace que no sólo se borren los registros indicados sino todos los relacionados. En la práctica esto significa que no se pueden borrar registros cuya clave primaria sea referenciada por alguna clave foránea en otra tabla, a no ser que dicha tabla secundaria tenga activada la clausula ON DELETE CASCADE en su clave foránea, en cuyo caso se borraría el/los registro/s de la tabla principal y los registros de tabla secundaria cuya clave foránea coincide con la clave primaria eliminada en la tabla primera.

Eliminación de datos usando una subconsulta

Al igual que en el caso de las instrucciones INSERT o SELECT, DELETE dispone de cláusula WHERE y en dicha cláusula podemos utilizar subconsultas. Por ejemplo:

```
DELETE empleados
WHERE id_empleado IN (SELECT id_empleado FROM operarios);
```

En este caso se trata de una subconsulta creada con el operador IN, se eliminarán los empleados cuyo identificador esté dentro de la tabla operarios.

2.4. Notas a tener en cuenta

Cuando en una sentencia DML usamos una subconsulta o consulta esta nunca se puede realizar sobre la tabla (o posibles tablas en el caso de vistas) en la que se está actuando.

Aclaraciones sobre los símbolos de la sintaxis:

```
[…] Opcional. La parte entre corchetes no es obligatoria para
construir la sentencia.
{…} Obligatoria. Señala una serie de opciones entre las que hay que
elegir una.
| Separa distintas opciones entre las que hay que elegir una
```

2.4.1 Sobre las Vistas

Aunque no vamos a trabajar con vistas, me ha parecido oportuno adjuntar lo siguiente para los que tengáis curiosidad:

Las instrucciones DML ejecutadas sobre las vistas permiten añadir o modificar los datos de las tablas relacionados con las filas de la vista. Ahora bien, no es posible ejecutar instrucciones DML sobre vistas que:

- Utilicen funciones de grupo (SUM, AVG,…)
- Usen GROUP BY o DISTINCT
- Posean columnas con cálculos (P. ej: PRECIO * 1.16)

Además no se pueden añadir datos a una vista si en las tablas referencias en la consulta SELECT hay campos NOT NULL que no aparecen en la consulta (es lógico ya que al añadir el dato se tendría que añadir el registro colocando el valor NULL en el campo).

Si tenemos la siguiente vista:

```
CREATE VIEW resumen (id_localidad, localidad, poblacion,
                     n_provincia, provincia, superficie,
                     id_comunidad, comunidad)
```

```
AS SELECT L.IdLocalidad, L.Nombre, L.Poblacion,
            P.IdProvincia, P.Nombre, P.Superficie,
            C.IdComunidad, C.Nombre
   FROM LOCALIDADES L JOIN PROVINCIAS P ON L.IdProvincia=P.IdProvincia
                      JOIN COMUNIDADES C ON
P.IdComunidad=C.IdComunidad;
```

Si realizamos la siguiente inserción

```
INSERT INTO resumen (id_localidad, localidad, poblacion)
VALUES (10000, 'Sevilla', 750000);
```

Se producirá un error, puesto que estamos insertando un registro dentro de la vista donde muchos de sus campos no tienen especificado un valor y por tanto serán insertados a NULL. El problema es que no puede insertarse un NULL en n_provincia ni id_comunidad puesto que son claves primarias de las tablas subyacentes PROVINCIAS y COMUNIDADES. La solución al problema anterior se soluciona creando un disparador (trigger) de sustitución, que veremos en el apartado de triggers.