LENGUAJE DE MANIPULACIÓN DE DATOS: DML. Modificación de una base de datos.

5.1. INTRODUCCIÓN

En esta unidad veremos 3 sentencias SQL DML para la modificación de datos. Además aprenderemos el funcionamiento de las transacciones y realizaremos una introducción al lenguaje PL/SQL.

5.2. 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.

5.2.1. Inserción de datos

El comando INSERT de SQL permite introducir datos

lods

en una tabla o en una vista de la base de datos. 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
La forma más habitual de introducir datos es la siguiente:

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

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';
```

5.2.2. 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 localidad 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.

5.2.3. 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 cláusula 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.

5.2.4. Ejecución de comandos DML sobre vistas.

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 referenciadas 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.**

5.3. GESTIÓN DE TRANSACCIONES

En términos teóricos, una transacción es un conjunto de tareas relacionadas que se realizan de forma satisfactoria o incorrecta como una unidad. En términos de procesamiento, las transacciones se confirman o se anulan. Para que una transacción se confirme se debe garantizar la permanencia de los cambios efectuados en los datos. Los cambios deben conservarse aunque el sistema se bloquee o tengan lugar otros eventos imprevistos.

Existen 4 propiedades necesarias, que son conocidas como propiedades **ACID**:

- atomicidad (Atomicity)
- coherencia (Consistency)
- aislamiento (Isolation)
- permanencia (Durability).

Estas propiedades garantizan un comportamiento predecible, reforzando la función de las transacciones como proposiciones de todo o nada.

> **Atomicidad:** Una transacción es una unidad de trabajo el cual se realiza en su totalidad o no se realiza en ningún caso. Las operaciones asociadas a una transacción comparten normalmente un objetivo común y son interdependientes. Si el sistema ejecutará únicamente una parte de las operaciones, podría poner en peligro el objetivo final de la transacción.
>
> **Coherencia:** Una transacción es una unidad de integridad porque mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado de datos igualmente coherente.
>
> **Aislamiento:** Una transacción es una unidad de aislamiento, permitiendo que transacciones concurrentes se comporten como si cada una fuera la única transacción que se ejecuta en el sistema. El aislamiento requiere que parezca que

cada transacción sea la única que manipula el almacén de datos, aunque se puedan estar ejecutando otras transacciones al mismo tiempo. Una transacción nunca debe ver las fases intermedias de otra transacción.

Permanencia: Una transacción también es una unidad de recuperación. Si una transacción se realiza satisfactoriamente, el sistema garantiza que sus actualizaciones se mantienen aunque el equipo falle inmediatamente después de la confirmación. El registro especializado permite que el procedimiento de reinicio del sistema complete las operaciones no finalizadas, garantizando la permanencia de la transacción.

En términos más prácticos, una transacción está formada por una serie de instrucciones DML. Una transacción comienza con la primera instrucción DML que se ejecute y finaliza con una operación COMMIT (si la transacción se confirma) o una operación ROLLBACK (si la operación se cancela). Hay que tener en cuenta que cualquier instrucción DDL o DCL da lugar a un COMMIT implícito, es decir todas las instrucciones DML ejecutadas hasta ese instante pasan a ser definitivas.

Existen tres comandos básicos de control en las transacciones SQL:

- **COMMIT.** Para guardar los cambios.
- **ROLLBACK.** Para abandonar la transacción y deshacer los cambios que se hubieran hecho en la transacción.
- **SAVEPOINT.** Crea checkpoints, puntos concretos en la transacción donde poder deshacer la transacción hasta esos puntos.

Los comandos de control de transacciones **se usan sólo con INSERT, DELETE y UPDATE.** No pueden utilizarse creando tablas o vaciándolas porque las operaciones se guardan automáticamente en la base de datos.

Para poder hacer uso de transacciones en SQL*Plus debemos tener desactivado el modo AUTOCOMMIT. Podemos ver su estado con la orden:

```
SHOW AUTOCOMMIT
```

Para desactivar dicho modo, usamos la orden:

```
SET AUTOCOMMIT OFF
```

5.3.1. COMMIT

La instrucción **COMMIT** hace que los cambios realizados por la transacción sean definitivos, irrevocables. Se dice que tenemos una transacción confirmada. Sólo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el COMMIT ya que las instrucciones ejecutadas pueden afectar a miles de registros. Además

el cierre correcto de la sesión da lugar a un COMMIT, aunque siempre conviene ejecutar explícitamente esta instrucción a fin de asegurarnos de lo que hacemos.

5.3.2. ROLLBACK

Esta instrucción regresa a la instrucción anterior al inicio de la transacción, normalmente el último COMMIT, la última instrucción DDL o DCL o al inicio de sesión. Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación. Un abandono de sesión incorrecto o un problema de comunicación o de caída del sistema dan lugar a un ROLLBACK implícito.

5.3.3. SAVEPOINT

Esta instrucción permite establecer un punto de ruptura. El problema de la combinación ROLLBACK/COMMIT es que un COMMIT acepta todo y un ROLLBACK anula todo. SAVEPOINT permite señalar un punto intermedio entre el inicio de la transacción y la situación actual. Su sintaxis es:

```
SAVEPOINT nombre;
```

Para regresar a un punto de ruptura concreto se utiliza ROLLBACK TO SAVEPOINT seguido del nombre dado al punto de ruptura. También es posible hacer ROLLBACK TO nombre de punto de ruptura. Cuando se vuelve a un punto marcado, las instrucciones que siguieron a esa marca se anulan definitivamente.

Ejemplo de uso:

```
SET AUTOCOMMIT OFF;

CREATE TABLE T (FECHA DATE);

INSERT INTO T VALUES ('01/01/2017');
INSERT INTO T VALUES ('01/02/2017');

SAVEPOINT febrero;

INSERT INTO T VALUES ('01/03/2017');
INSERT INTO T VALUES ('01/04/2017');

SAVEPOINT abril;

INSERT INTO T VALUES ('01/05/2017');

ROLLBACK TO febrero;
-- También puede escribirse ROLLBACK TO SAVEPOINT febrero;
-- En este ejemplo sólo se guardan en la tabla
-- los 2 primeros registros o filas.
```

5.3.4. Estado de los datos durante la transacción

Si se inicia una transacción usando comandos DML hay que tener en cuenta que:

- Se puede volver a la instrucción anterior a la transacción cuando se desee.
- Las instrucciones de consulta SELECT realizadas por el usuario que inició la transacción muestran los datos ya modificados por las instrucciones DML.
- El resto de usuarios ven los datos tal cual estaban antes de la transacción, de hecho los registros afectados por la transacción aparecen bloqueados hasta que la transacción finalice. Esos usuarios no podrán modificar los valores de dichos registros.
- Tras la transacción todos los usuarios ven los datos tal cual quedan tras el fin de la transacción. Los bloqueos son liberados y los puntos de ruptura borrados.

5.3.5. Concurrencia de varias transacciones (Bloqueos)

Cuando se realizan varias transacciones de forma simultánea, pueden darse diversas situaciones en el acceso concurrente a los datos, es decir, cuando se accede a un mismo dato en dos transacciones distintas. Estas situaciones son:

- ☐ **Lectura sucia (Dirty Read)**. Una transacción lee datos que han sido escritos por otra transacción que aún no se ha confirmado.
- ☐ **Lectura no repetible (Non-repeateable Read)**. Una transacción vuelve a leer los datos que ha leído anteriormente y descubre que otra transacción confirmada ha modificado o eliminado los datos.
- ☐ **Lectura fantasma (Phantom Read).** Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisface una condición de búsqueda y descubre que otra transacción confirmada ha insertado filas adicionales que satisfacen la condición.

Para una mejor gestión de estas situaciones debemos indicar el nivel de aislamiento que deseamos. De las cuatro propiedades de ACID de un SGBD, la propiedad de aislamiento es la más laxa. Un nivel de aislamiento bajo aumenta la capacidad de muchos usuarios para acceder a los mismos datos al mismo tiempo, pero también aumenta el número de efectos de concurrencia (como lecturas sucias). Un mayor nivel de aislamiento puede dar como resultado una pérdida de concurrencia y el aumento de las posibilidades de que una transacción bloquee a otra.

Podemos solicitar al SGBD cuatro niveles de aislamiento. De menor a mayor nivel de aislamiento, tenemos:

- **READ UNCOMMITTED** (Lectura no confirmada). Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones. Permite que sucedan las 3 situaciones indicadas previamente: lecturas fantasma, no repetibles y sucias.
- **READ COMMITTED** (Lectura confirmada). Los datos leídos por una transacción pueden ser modificados por otras transacciones. Se pueden dar lecturas fantasma y lecturas no repetibles.
- **REPEATEABLE READ** (Lectura repetible). Consiste en que ningún registro leído con un SELECT se puede cambiar en otra transacción. Solo pueden darse lecturas fantasma.

- **SERIALIZABLE**. Las transacciones ocurren de forma totalmente aislada a otras transacciones. Se bloquean las transacciones de tal manera que ocurren unas detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente si puede asegurar que no hay conflicto con el acceso a los datos.

Nivel de aislamiento y Lecturas

Nivel de aislamiento	Lecturas sucias	Lecturas no repetibles	Lecturas fantasma
READ UNCOMMITTED	SÍ	SÍ	SÍ
READ COMMITTED	NO	SÍ	SÍ
REPEATEABLE READ	NO	NO	SÍ
SERIALIZABLE	NO	NO	NO

Internamente el SGBD proporciona dicho nivel de aislamiento mediante bloqueos en los datos.
En Oracle, el nivel por defecto es READ COMMITED. Además de éste, solo permite SERIALIZABLE. Se puede cambiar ejecutando el comando:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```