Tema 10. Manipulación de datos

Contenido

1 Inserción de datos

La adición de datos a una tabla se realiza mediante la instrucción INSERT. Su sintaxis fundamental es:

```
INSERT INTO tabla [(listaDeCampos)]
VALUES (valor1 [, valor2 ...])
```

La tabla representa la tabla a la que queremos añadir el registro y los valores que siguen a VALUES son los valores que damos a los distintos campos del registro. Si no se especifica la lista de campos, la lista de valores debe seguir el orden de las columnas según fueron creados (es el orden de columnas según las devuelve el comando DESCRIBE).

La lista de campos a rellenar se indica si no queremos rellenar todos los campos. Los campos no rellenados explícitamente con la orden INSERT, se rellenan con su valor por defecto (DEFAULT) o bien con NULL si no se indicó valor alguno. Si algún campo tiene restricción de obligatoriedad (NOT NULL), ocurrirá un error si no rellenamos el campo con algún valor.

Por ejemplo, supongamos que tenemos una tabla de clientes cuyos campos son: dni, nombre, apellido1, apellido2, localidad y dirección; supongamos que ese es el orden de creación de los campos de esa tabla y que la localidad tiene como valor por defecto Palencia y la dirección no tiene valor por defecto. En ese caso estas dos instrucciones son equivalentes:

```
INSERT INTO clientes VALUES( '11111111','Pedro','Gutiérrez', 'Crespo',DEFAULT,NULL);

INSERT INTO clientes(dni,nombre,apellido1,apellido2)
VALUES('11111111','Pedro','Gutiérrez', 'Crespo');
```

Son equivalentes puesto que en la segunda instrucción los campos no indicados se rellenan con su valor por defecto y la dirección no tiene valor por defecto. La palabra DEFAULT fuerza a utilizar ese valor por defecto.

Notas

- Las columnas a las que damos valores se identifican por su nombre.
- La asociación columna-valor es por posición: dni -> '11111111', apellido1 -> 'Gutierrez'.
- Los valores que se dan a las columnas deben coincidir con el tipo de dato de la columna.
- Los valores constantes de tipo carácter o fecha deben ir entre comillas simples.
- Las columnas para las que no damos valores se rellenan con su valor por defecto o si no lo tiene con NULL.

Ejemplo 1

Nombre de Columna	Tipo de Dato	Nulo	V
JOB_ID	VARCHAR2(10)	No	-
JOB_TITLE	VARCHAR2(35)	No	-
MIN_SALARY	NUMBER(6,0)	Yes	-
MAX_SALARY	NUMBER(6,0)	Yes	-

```
INSERT INTO JOBS (JOB_ID, JOB_TITLE, MIN_SALARY)
VALUES ('REPRE', 'Representante', 1800);
1 fila(s) insertada(s).
```

AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
PU_MAN	Purchasing Manager	8000	15000
PU_CLERK	Purchasing Clerk	2500	5500
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
SH_CLERK	Shipping Clerk	2500	5500
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000
HR_REP	Human Resources Representative	4000	9000
PR_REP	Public Relations Representative	4500	10500
REPRE	Representante	1800	-

Ejemplo 2

Nombre de Columna	Tipo de Dato	Nulo	V
JOB_ID	VARCHAR2(10)	No	-
JOB_TITLE	VARCHAR2(35)	No	-
MIN_SALARY	NUMBER(6,0)	Yes	-
MAX_SALARY	NUMBER(6,0)	Yes	-

```
INSERT INTO JOBS (JOB_ID, MIN_SALARY)
VALUES ('CLA_VE', 2000);
```

Esta consulta nos daría error si en la definición de la columna especificamos JOB_TITLE y/o MAX_SALARY como NOT NULL. En ese caso no podemos insertar una fila con valor nulo en esas columnas.

Ejemplo 3

```
INSERT INTO JOBS
VALUES ('COM', 'Comercial', 1500, 2000);
```

Si damos un valor para cada columna, no es necesario indicar el nombre de las columnas.

1.1 INSERT con SELECT

Hasta ahora solo hemos insertado una fila, pero si a INSERT añadimos una consulta, se añaden tantas filas como devuelva la consulta. Si no se especifican los nombres de columna en INSERT, por defecto se consideran todas las columnas de la tabla.

```
INSERT INTO Nombretabla1 [(columna [, columna]…)]
SELECT {columna [, columna] … | *}
FROM Nombretabla2 [cláusulas de select];
```

Ejemplo 4

- Primero creamos la tabla test.

```
create table test(id number(6) primary key,
name varchar2(20), salary number(8,2));
```

- Insertamos en esta tabla el identificador de empleado, el apellido y el salario de los empleados del departamento 30.

```
insert into test
    select employee_id, last_name, salary
    from employees
    where department_id=30;
```

Ejemplo 5

```
insert into test (id, name)
    select employee_id, first_name
    from employees
    where job_id='SA_MAN';
```

En este caso es necesario escribir los nombres de columna id y name

Ejemplo 6

Insertar en la tabla JOBS el identificador REP, el nombre de trabajo 'Repartidor' y el salario máximo y mínimo, el mismo que el del puesto de trabajo Sales Manager.

```
insert into jobs
    select 'REP', 'Repartidor', min_salary, max_salary
    from jobs
    where job_title='Sales Manager';
```

Ejemplo 7

Podemos utilizar funciones a la hora de insertar valores.

JOB_HISTORY

Nombre de Columna	Tipo de Dato	Nulo
EMPLOYEE_ID	NUMBER(6,0)	No
START_DATE	DATE	No
END_DATE	DATE	No
JOB_ID	VARCHAR2(10)	No
DEPARTMENT_ID	NUMBER(4,0)	Yes

```
insert into job_history
values (110,'1/11/2001', sysdate, 'SA_MAN', 90);
```

2 Modificación de datos

La modificación de los datos de los registros la implementa la instrucción UPDATE.

Sintaxis:

```
UPDATE tabla
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:

```
UPDATE clientes SET provincia='Ourense'
WHERE provincia='Orense';
```

```
UPDATE productos SET precio=precio*1.16;
```

El primer dato actualiza la provincia de los clientes de Orense para que aparezca como Ourense.

El segundo UPDATE incrementa los precios en un 16%. La expresión para el valor puede ser todo lo compleja que se desee (en el ejemplo se utilizan funciones de fecha para conseguir que los partidos que se jugaban hoy, pasen a jugarse el martes):

```
UPDATE partidos SET fecha= NEXT_DAY(SYSDATE,'Martes')
WHERE fecha=SYSDATE;
```

Si se omite WHERE, la actualización afectará a todas las filas de la tabla.

En la condición se pueden utilizar cualquiera de los siguientes operadores de comparación:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

También se puede utilizar:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

2.1 UPDATE con SELECT

Podemos incluir una subconsulta en una sentencia UPDATE. La SELECT puede estar contenida en la cláusula WHERE o puede formar parte de SET.

Cuando la subconsulta (SELECT) forma parte de **SET** debe seleccionar una **única fila** y el **mismo número de columnas**, que las que hay entre paréntesis al lado de SET.

Formato 1:

```
UPDATE nombretabla
SET (col1, col2,…)=(SELECT col1,col2,…)
WHERE condición
```

Formato 2:

```
UPDATE nombretabla
SET columna1=(SELECT col1…),
 columna2 =(SELECT col2…), …
WHERE condición
```

Formato 3:

```
UPDATE nombretabla
SET columna1=valor1,
 columna2=valor2 …
WHERE condición=(select…)
```

Ejemplo 1:

```
UPDATE TEST
SET (name, salary)=(select first_name, salary
 from employees
 where employee_id=148)
WHERE id=114.
```

En este caso modifica el nombre y salario del id 114 con el nombre y salario del empleado 148.

Ejemplo 2:

```
UPDATE TEST
SET (name, salary)=(select first_name, salary
 from employees
 where employee_id=148)
```

En este caso modifica el nombre y salario todos los empleados con el nombre y salario del empleado 148

3 Eliminación de datos

Se realiza mediante la instrucción DELETE:

```
DELETE [FROM] tabla
[WHERE condición]
```

Es más sencilla que las anteriores, elimina los registros de la tabla que cumplan la condición indicada.

Si no se especifica la cláusula WHERE, borramos TODAS las filas de la tabla.

Ejemplo 1:

```
DELETE FROM empleados
WHERE seccion=23;
```

Borra todas las filas de la tabla empleados de la sección 23.

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 hace que no sólo se borren los registros indicados en el SELECT, sino todos los relacionados.

Ejemplo 2:

```
DELETE FROM test
WHERE salary>(select avg(salary)
 from employees
 where department_id=110)
```

Borra todas las filas de la tabla test en las que el salario sea mayor que la media del salario de los empleados del departamento 110.

Ejemplo 3:

```
DELETE FROM departments
WHERE department_id=50
```

Esto generaría un error.

No permite borrar ya que hay una clave ajena en la tabla employees que hace referencia al departamento 50 (por defecto, si no se especifica nada, la opción de borrado es restrict).

3.1 Truncar una tabla

Truncar una tabla es quitar todas las filas de una tabla sin registrar las eliminaciones individuales de filas. Desde un punto de vista funcional, TRUNCATE TABLE es equivalente a la instrucción DELETE sin una cláusula WHERE; no obstante, TRUNCATE TABLE es más rápida y utiliza menos recursos de registros de transacciones y de sistema.

Ejemplo:

```
TRUNCATE TABLE DEPART;
```

Borra las filas de la tabla DEPART, no borra la tabla.

4 Transacciones

Una transacción es un grupo de acciones que hacen transformaciones (inserción, modificación o eliminación (inserción, modificación o eliminación de datos) en las tablas preservando la consistencia de los datos.

Es decir, una transacción es una o varias sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose en grupo.

No todas las operaciones SQL son transaccionales. Sólo son transaccionales las operaciones correspondientes al DML, es decir, sentencias SELECT, INSERT, UPDATE y DELETE.

4.1 COMMIT y ROLLBACK

Para confirmar una transacción se utiliza la sentencia COMMIT. Cuando realizamos COMMIT los cambios se escriben en la BD.

Para deshacer una transacción se utiliza la sentencia ROLLBACK. Cuando realizamos ROLLBACK se deshacen todas las modificaciones realizadas por la transacción en la BD, quedando ésta en el mismo estado que antes de iniciarse la transacción.

Ejemplo:

Un ejemplo son las transferencias bancarias. Para realizar una transferencia de dinero entre dos cuentas debemos descontar el dinero de una cuenta, realizar el ingreso en la otra cuenta y grabar las operaciones y movimientos necesarios, actualizar los saldos ... .

Si en alguno de estos puntos se produce un fallo en el sistema podríamos haber descontado el dinero de una de las cuentas y no haberlo ingresado en la otra. Por lo tanto, todas estas operaciones deben ser correctas o fallar todas.

En estos casos, al confirmar la transacción (COMMIT) o al deshacerla (ROLLBACK) garantizamos que todos los datos quedan en un estado consistente.

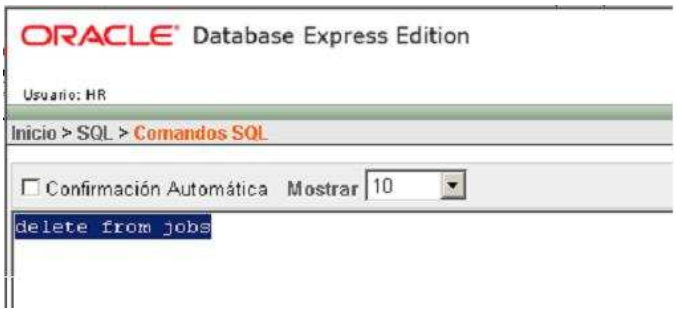
En una transacción los datos modificados no son visibles por el resto de usuarios hasta que se confirme la transacción.

Mientras se procesa una transacción, ningún otro usuario puede realizar cambios sobre los datos afectados por ella.

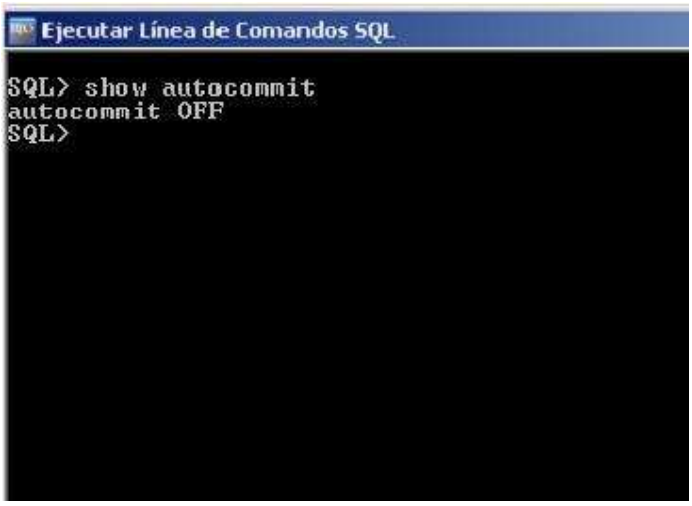
4.2 Validación automática

Podemos validar automáticamente las transacciones sin tener que indicarlo de forma explícita, es decir, sin tener que hacer un COMMIT.

Desde el entorno gráfico de OracleXE, podemos marcar la casilla Confirmación automática.

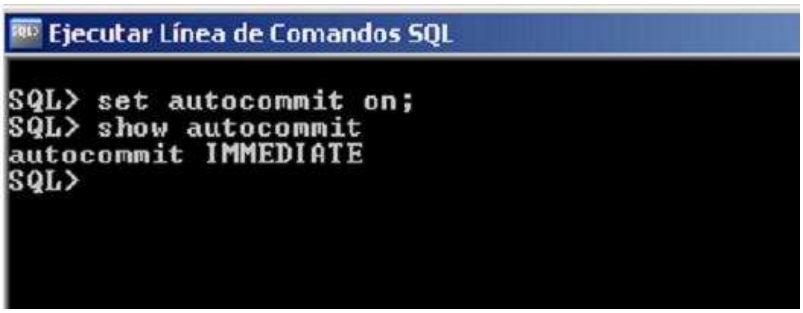


Desde SQL*Plus, utilizaremos el parámetro AUTOCOMMIT. Para ver su valor:



OFF es el valor por omisión, de modo que las transacciones INSERT, UPDATE, DELETE, no son definitivas hasta que no hagamos COMMIT.

Para activar el AUTOCOMMIT:



Hay varias órdenes SQL que fuerzan a que se ejecute un COMMIT sin necesidad de indicarlo

QUIT	DROP VIEW	DISCONNECT
CREATE TABLE	CONNECT	REVOQUE
DROP TABLE	GRANT	AUDIT
EXIT	ALTER	NOAUDIT
CREATE VIEW		

4.3 ROLLBACK

La orden ROLLBACK, aborta la transacción o cambio, volviendo a la situación de las tablas de la BD desde el último COMMIT.

Notar que si AUTOCOMMIT o la confirmación automática está activada, de nada servirá hacer un ROLLBACK. Es decir para que podamos aplicar ROLLBACK, deberemos tener la confirmación automática desactivada. Si después de haber realizado cambios en nuestras tablas, se produce un fallo del sistema (se va la luz) y no hemos validado el trabajo, Oracle hace un ROLLBACK automático sobre cualquier trabajo no validado. Tendremos que repetir el trabajo desde el último COMMIT.