



6.1 Introducción

Hasta ahora nos hemos dedicado a consultar datos de la base de datos mediante la sentencia SELECT. Hemos trabajado y seleccionado datos de tablas. Ha llegado el momento de cambiar los datos de las tablas de la base de datos. En esta unidad aprenderemos a insertar nuevas filas en una tabla, a actualizar los valores de las columnas en las filas y a borrar filas enteras.

6.2 Inserción de datos. Orden INSERT

Empezamos la manipulación de datos de una tabla con la orden **INSERT**. Con ella se añaden filas de datos en una tabla. El formato de esta orden es el siguiente:

```
INSERT INTO NombreTabla [(columna [, columna] ...)]
VALUES (valor [, valor] ...);
```

NombreTabla es la tabla en la que se van a insertar las filas.

[(columna [, columna] ...)] representa la columna o columnas donde se van a introducir valores. Si las columnas no se especifican en la cláusula INSERT, se consideran, por defecto, todas las columnas de la tabla.

(valor [, valor] ...) representa los valores que se van a dar a las columnas. Éstos se deben corresponder con cada una de las columnas que aparecen; además, deben coincidir con el tipo de dato definido para cada columna. Cualquier columna que no se encuentre en la lista de columnas recibirá el valor NULL, siempre y cuando no esté definida como **NOT NULL**, en cuyo caso INSERT fallará. Si no se da la lista de columnas, se han de introducir valores en todas las columnas.

Es posible introducir los valores directamente en la sentencia u obtenerlos a partir de la información existente en la base de datos mediante la inclusión de una consulta haciendo uso de la sentencia SELECT.

Caso práctico

1 Consideremos la tabla PROFESORES, cuya descripción y contenido son:

```
SQL> DESC PROFESORES
```

Nombre	¿Nulo?	Tipo
COD_CENTRO		NUMBER (4)
DNI	NOT NULL	NUMBER (10)
APELLIDOS		VARCHAR2 (30)
ESPECIALIDAD		VARCHAR2 (16)

(Continúa)



6. Manipulación de datos. INSERT, UPDATE y DELETE

6.2 Inserción de datos. Orden INSERT

(Continuación)

```
SQL> SELECT * FROM PROFESORES;
```

COD_CENTRO	DNI	APELLIDOS	ESPECIALIDAD
10	1112345	Martínez Salas, Fernando	INFORMÁTICA
10	4123005	Bueno Zarco, Elisa	MATEMÁTICAS
10	4122025	Montes García, M.Pilar	MATEMÁTICAS
15	9800990	Ramos Ruiz, Luis	LENGUA
15	1112345	Rivera Silvestre, Ana	DIBUJO
15	8660990	De Lucas Fdez, M.Angel	LENGUA
22	7650000	Ruiz Lafuente, Manuel	MATEMÁTICAS
45	43526789	Serrano Laguía, María	INFORMÁTICA

8 filas seleccionadas.

Damos de alta a una profesora con estos apellidos y nombre: 'Quiroga Martín, A. Isabel', de la especialidad 'INFORMÁTICA' y con el código de centro 45. Las columnas a las que damos valores son: APELLIDOS, ESPECIALIDAD y COD_CENTRO:

```
SQL> INSERT INTO PROFESORES (APELLIDOS, ESPECIALIDAD, COD_CENTRO)
```

```
2 VALUES ('Quiroga Martín, A.Isabel', 'INFORMÁTICA', 45);
```

```
1 fila creada.
```

Al ejecutar la sentencia, Oracle emite un mensaje (1 fila creada.) con el que indica que la fila se ha insertado correctamente. Observamos en esta sentencia que:

- Las columnas a las que damos valores se identifican por su nombre.
- La asociación columna-valor es posicional.
- Los valores que se dan a las columnas deben coincidir con el tipo de dato definido en la columna.
- Los valores constantes de tipo carácter han de ir encerrados entre comillas simples (' ') (los de tipo fecha, también).

Ahora, el contenido de la tabla PROFESORES tendrá una fila más:

```
SQL> SELECT * FROM PROFESORES;
```

COD_CENTRO	DNI	APELLIDOS	ESPECIALIDAD
10	1112345	Martínez Salas, Fernando	INFORMÁTICA
10	4123005	Bueno Zarco, Elisa	MATEMÁTICAS
10	4122025	Montes García, M.Pilar	MATEMÁTICAS
15	9800990	Ramos Ruiz, Luis	LENGUA
15	1112345	Rivera Silvestre, Ana	DIBUJO
15	8660990	De Lucas Fdez, M.Angel	LENGUA
22	7650000	Ruiz Lafuente, Manuel	MATEMÁTICAS
45	43526789	Serrano Laguía, María	INFORMÁTICA
45		Quiroga Martín, A.Isabel	INFORMÁTICA

9 filas seleccionadas.

6. Manipulación de datos. INSERT, UPDATE y DELETE

6.2 Inserción de datos. Orden INSERT



(Continuación)

Las columnas para las que no dimos valores aparecen como nulos; en este caso, la columna DNI.

Insertamos a un profesor que no tiene código de centro asignado, de apellidos y nombre 'Seco Jiménez, Ernesto' y de la especialidad 'LENGUA'. Las columnas a las que damos valores son APELLIDOS y ESPECIALIDAD:

```
SQL> INSERT INTO PROFESORES (APELLIDOS, ESPECIALIDAD)
```

```
2 VALUES ('Seco Jiménez, Ernesto', 'LENGUA');
```

```
INSERT INTO PROFESORES (APELLIDOS, ESPECIALIDAD)
```

```
*
```

```
ERROR en línea 1:
```

```
ORA-01400: no se puede realizar una inserción NULL en
```

```
("SCOTT"."PROFESORES"."COD_CENTRO")
```

Observamos que aparece un mensaje de error que indica que no se puede insertar una columna con valor NULO en la tabla si en su definición se ha especificado NOT NULL (COD_CENTRO está definido como NOT NULL).

Insertamos a un profesor de apellidos y nombre 'Gonzalez Sevilla, Miguel A.' en el código de centro 22, con DNI 23444800 y de la especialidad de 'HISTORIA':

```
SQL> INSERT INTO PROFESORES
```

```
2 VALUES (22, 23444800, 'González Sevilla, Miguel A.', 'HISTORIA');
```

```
1 fila creada.
```

No es necesario especificar el nombre de las columnas ya que las hemos dado un valor en la fila que insertamos. Este valor ha de ir en el mismo orden en que las columnas estén definidas en la tabla.

Actividades propuestas



Escribe la sentencia INSERT anterior de otra manera.

Inserta un profesor cuya especialidad supere los 16 caracteres de longitud. Comenta el resultado.

A. Inserción con SELECT

Hasta el momento sólo hemos insertado una fila, pero si añadimos a INSERT una consulta, es decir, una sentencia SELECT, se añaden tantas filas como devuelva la consulta. El formato de INSERT con SELECT es el siguiente:

```
INSERT INTO NombreTabla1 [(columna [, columna] ...)]  
SELECT {columna [, columna] ... | *}  
FROM NombreTabla2 [CLÁUSULAS DE SELECT];
```

Si las columnas no se especifican en la cláusula INSERT, por defecto, se consideran todas las columnas de la tabla.



6. Manipulación de datos. INSERT, UPDATE y DELETE

6.2 Inserción de datos. Orden INSERT



Caso práctico

- 2 Disponemos de la tabla EMPLE30, cuya descripción es la misma que la de la tabla EMPLE. Insertamos los datos de los empleados del departamento 30:

```
SQL> INSERT INTO EMPLE30
2 (EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION, DEPT_NO)
3 SELECT
4 EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION, DEPT_NO
5 FROM EMPLE
6 WHERE DEPT_NO=30;
```

6 filas creadas.

Como las tablas EMPLE y EMPLE30 tienen la misma descripción, no es preciso especificar las columnas, siempre y cuando queramos dar valores a todas las columnas. Esta sentencia daría el mismo resultado:

```
SQL> INSERT INTO EMPLE30 SELECT * FROM EMPLE WHERE DEPT_NO=30;
6 filas creadas.
```

Disponemos de la tabla NOMBRES, que tiene la siguiente descripción:

```
SQL> DESC NOMBRES
```

Nombre	¿Nulo?	Tipo
NOMBRE		VARCHAR2 (15)
EDAD		NUMBER (2)

Insertamos en la tabla NOMBRES, en la columna NOMBRE, el APELLIDO de los empleados de la tabla EMPLE que sean del departamento 20: **INSERT INTO NOMBRES (NOMBRE) SELECT APELLIDO FROM EMPLE WHERE DEPT_NO=20;**

Si, al insertar los apellidos, alguno supera la longitud para la columna NOMBRE de la tabla NOMBRES, no se insertará y aparecerá un error.

Insertar un empleado de apellido 'GARCÍA', con número de empleado 1111, en la tabla EMPLE, en el departamento con mayor número de empleados. La fecha de alta será la actual; inventamos el resto de los valores. En primer lugar, vamos a averiguar qué sentencia SELECT calcula el departamento con más empleados: **SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING COUNT(*) = (SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY DEPT_NO);**

```
DEPT_NO
-----
30
```

Ahora hacemos la inserción en la tabla EMPLE, teniendo en cuenta la SELECT anterior:

```
INSERT INTO EMPLE SELECT DISTINCT 1111, 'GARCIA', 'ANALISTA', 7566,
SYSDATE, 2000, 120, DEPT_NO
FROM EMPLE WHERE DEPT_NO= (SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO
HAVING COUNT(*)= (SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY DEPT_NO));
```

(Continúa)

6. Manipulación de datos. INSERT, UPDATE y DELETE

6.3 Modificación. Orden UPDATE



(Continuación)

Al hacer la inserción sólo desconocemos el valor de la columna DEPT_NO, que es el que devuelve la SELECT; el resto de valores, como APELLIDO, OFICIO y EMP_NO, los conocemos y, por tanto, los ponemos directamente en la sentencia SELECT. La cláusula DISTINCT es necesaria, ya que sin ella se insertarían tantas filas como empleados haya en el departamento con mayor número de empleados. Si ejecutamos la sentencia sin DISTINCT insertará más de una fila.

Insertar un empleado de apellido 'QUIROGA', con número de empleado 1112, en la tabla EMPLE. Los restantes datos del nuevo empleado serán los mismos que los de 'GIL' y la fecha de alta será la fecha actual:

```
SQL> INSERT INTO EMPLE
2  SELECT 1112, 'QUIROGA', OFICIO, DIR, SYSDATE, SALARIO,
3  COMISION, DEPT_NO
4  FROM EMPLE WHERE APELLIDO='GIL';
```

1 fila creada.

Las columnas cuyos valores desconocemos (OFICIO, DIR, SALARIO, COMISION, DEPT_NO) son las que devolverá la sentencia SELECT; en el resto de las columnas ponemos directamente sus valores.

Actividades propuestas



- 2 Dadas las tablas ALUM y NUEVOS, inserta en la tabla ALUM los nuevos alumnos.

Inserta un empleado de apellido 'SAAVEDRA' con número 2000. La fecha de alta será la actual, el SALARIO será el mismo salario de 'SALA' más el 20 por 100 y el resto de datos serán los mismos que los datos de 'SALA'.

6.3 Modificación. Orden UPDATE

Para actualizar los valores de las columnas de una o varias filas de una tabla utilizamos la orden **UPDATE**, cuyo formato es el siguiente:

```
UPDATE NombreTabla
SET columnal=valor1, ..., columnan=valorn
WHERE condición;
```

- *NombreTabla* es la tabla cuyas columnas se van a actualizar.
- *SET* indica las columnas que se van a actualizar y sus valores.
- *WHERE* selecciona las filas que se van a actualizar. Si se omite, la actualización afectará a todas las filas de la tabla.



6. Manipulación de datos. INSERT, UPDATE y DELETE

6.3 Modificación. Orden UPDATE



Caso práctico

3 Sea la tabla CENTROS, cambiamos la dirección del COD_CENTRO 22 a 'C/Pilón 13' y el número de plazas a 295:

```
UPDATE CENTROS SET DIRECCION = 'C/Pilón 13', NUM_PLAZAS = 295 WHERE COD_CENTRO = 22;
```

¿Qué hubiese ocurrido si no hubiésemos puesto la cláusula WHERE?

```
SQL> UPDATE CENTROS SET DIRECCION = 'C/Pilón 13', NUM_PLAZAS = 295;  
5 filas actualizadas.
```

```
SQL> SELECT * FROM CENTROS;
```

COD_CENTRO	T	NOMBRE	DIRECCION	TELEFONO	NUM_PLAZAS
10	S	IES El Quijote	C/Pilón 13	965-887654	295
15	P	CP Los Danzantes	C/Pilón 13	985-112322	295
22	S	IES Planeta Tierra	C/Pilón 13	925-443400	295
45	P	CP Manuel Hidalgo	C/Pilón 13	926-202310	295
50	S	IES Antoñete	C/Pilón 13	989-406090	295

Como se aprecia, se hubieran modificado todas las filas de la tabla CENTROS con la dirección 'C/Pilón 13' y 295 en número de plazas.



Actividades propuestas



Aumenta en 100 euros el salario y en 10 euros la comisión a todos los empleados del departamento 10, de la tabla EMPLE.

x A. UPDATE con SELECT

Podemos incluir una subconsulta en una sentencia UPDATE que puede estar contenida en la cláusula WHERE o puede formar parte de SET. Cuando la subconsulta (orden SELECT) forma parte de SET, debe seleccionar una única fila y el mismo número de columnas (con tipos de datos adecuados) que las que hay entre paréntesis al lado de SET. Los formatos son:

```
UPDATE <NombreTabla>
```

```
SET columna1 = valor1, columna2 = valor2, ...
```

```
WHERE columna3 = (SELECT ...);
```


6. Manipulación de datos. INSERT, UPDATE y DELETE

6.3 Modificación. Orden UPDATE



```
UPDATE <NombreTabla>
SET (columna1, columna2, ...) = (SELECT col1, col2, ...)
WHERE condición;

UPDATE <NombreTabla>
SET columna1 = (SELECT col1 ...), columna2 = (SELECT
col2 ...)
WHERE condición;
```

Caso práctico



- 4 En la tabla CENTROS la siguiente orden UPDATE igualará la dirección y el número de plazas del código de centro 10 a los valores de las columnas correspondientes que están almacenadas para el código de centro 50. Los valores actuales de estos centros son:

```
SQL> UPDATE CENTROS SET (DIRECCION, NUM_PLAZAS) = (SELECT DIRECCION,
NUM_PLAZAS FROM CENTROS WHERE COD_CENTRO = 50) WHERE COD_CENTRO= 10;
```

A partir de la tabla EMPLA, cambia el salario a la mitad y la comisión a 0, a aquellos empleados que pertenezcan al departamento con mayor número de empleados.

```
SQL> UPDATE EMPLA SET SALARIO = SALARIO/2, COMISION = 0 WHERE DEPT_NO =
(SELECT DEPT_NO FROM EMPLA GROUP BY DEPT_NO HAVING COUNT(*) =
(SELECT MAX(COUNT(*)) FROM EMPLA GROUP BY DEPT_NO));
```

Para todos los empleados de la tabla EMPLA y del departamento de 'CONTABILIDAD', cambiamos su salario al doble del salario de 'SÁNCHEZ' y su apellido, a minúscula.

```
SQL> UPDATE EMPLA SET APELLIDO = LOWER(APELLIDO),
SALARIO = (SELECT SALARIO*2 FROM EMPLA WHERE APELLIDO = 'SANCHEZ')
WHERE DEPT_NO = (SELECT DEPT_NO FROM DEPART
WHERE DNOMBRE = 'CONTABILIDAD');
```

Actividades propuestas



- 4 Modifica el número de departamento de 'SAAVEDRA'. El nuevo departamento será el departamento donde hay más empleados cuyo oficio sea 'EMPLEADO'.



6. Manipulación de datos. INSERT, UPDATE y DELETE

6.4 Borrado de filas. Orden DELETE

x 6.4 Borrado de filas. Orden DELETE

Para eliminar una fila o varias filas de una tabla se usa la **orden DELETE**. La cláusula **WHERE** es esencial para eliminar sólo aquellas filas deseadas. Sin la cláusula **WHERE** **DELETE** borrará todas las filas de la tabla. El espacio usado por las filas que han sido borradas no se reutiliza, a menos que se realice un **EXPORT** o un **IMPORT**. La condición puede incluir una subconsulta. Éste es su formato:

DELETE [FROM] NombreTabla WHERE Condición;



Caso práctico



Borramos el **COD_CENTRO 50** de la tabla **CENTROS**:

```
SQL> DELETE FROM CENTROS WHERE COD_CENTRO=50;
```

O bien:

```
SQL> DELETE CENTROS WHERE COD_CENTRO = 50;
```

Borramos todas las filas de la tabla **CENTROS**:

```
SQL> DELETE FROM CENTROS;
```

Igualmente, podríamos haber puesto: **DELETE CENTROS;**

Borramos todas las filas de la tabla **LIBRERIA** cuyos **EJEMPLARES** no superen la media de ejemplares en su **ESTANTE**:

```
SQL> DELETE FROM LIBRERIA L WHERE EJEMPLARES <
      (SELECT AVG(EJEMPLARES) FROM LIBRERIA WHERE ESTANTE = L.ESTANTE
      GROUP BY ESTANTE);
```

Borramos los departamentos de la tabla **DEPART** con menos de cuatro empleados.

```
SQL> DELETE FROM DEPART WHERE DEPT_NO IN
      (SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING COUNT(*) < 4);
```



Actividades propuestas

5 Borra de la tabla **ALUM** los **ANTIGUOS** alumnos.

Borra todos los departamentos de la tabla **DEPART** para los cuales no existan empleados en **EMPLE**.

6. Manipulación de datos. INSERT, UPDATE y DELETE

6.5 ROLLBACK, COMMIT y AUTOCOMMIT



6.5 ROLLBACK, COMMIT y AUTOCOMMIT

Supongamos que queremos borrar una fila de una tabla pero, al teclear la orden SQL, se nos olvida la cláusula WHERE y... ¡horror!, ¡borramos todas las filas de la tabla! Esto no es problema, pues Oracle permite dar marcha atrás a un trabajo realizado mediante la orden **ROLLBACK**, siempre y cuando no hayamos validado los cambios en la base de datos mediante la orden **COMMIT**.

Cuando hacemos transacciones sobre la base de datos, es decir, cuando insertamos, actualizamos y eliminamos datos en las tablas, los cambios no se aplicarán a la base de datos hasta que no hagamos un **COMMIT**. Esto significa que, si durante el tiempo que hemos estado realizando transacciones, no hemos hecho ningún **COMMIT** y de pronto se va la luz, todo el trabajo se habrá perdido, y nuestras tablas estarán en la situación de partida.

Para validar los cambios que se hagan en la base de datos tenemos que ejecutar la orden **COMMIT**:

```
SQL> COMMIT;  
Validación terminada.
```

SQL*Plus e iSQL*Plus permiten validar automáticamente las transacciones sin tener que indicarlo de forma explícita. Para eso sirve el **parámetro AUTOCOMMIT**. El valor de este parámetro se puede mostrar con la orden **SHOW**, de la siguiente manera:

```
SQL> SHOW AUTOCOMMIT;  
autocommit OFF
```

OFF es el valor por omisión, de manera que las transacciones (INSERT, UPDATE y DELETE) no son definitivas hasta que no hagamos **COMMIT**. Si queremos que INSERT, UPDATE Y DELETE tengan un carácter definitivo sin necesidad de realizar la validación **COMMIT**, hemos de activar el parámetro **AUTOCOMMIT** con la orden **SET**:

```
SQL> SET AUTOCOMMIT ON;
```

```
SQL> SHOW AUTOCOMMIT;  
autocommit IMMEDIATE
```

Ahora, cualquier INSERT, UPDATE y DELETE se validará automáticamente.

La orden **ROLLBACK** aborta la transacción volviendo a la situación de las tablas de la base de datos desde el último **COMMIT**:

```
SQL> ROLLBACK;  
Rollback terminado.
```

Una **transacción** es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo.



6. Manipulación de datos. INSERT, UPDATE y DELETE

6.5 ROLLBACK, COMMIT y AUTOCOMMIT

La Figura 6.1 muestra transacciones típicas que ilustran las condiciones de COMMIT y ROLLBACK.

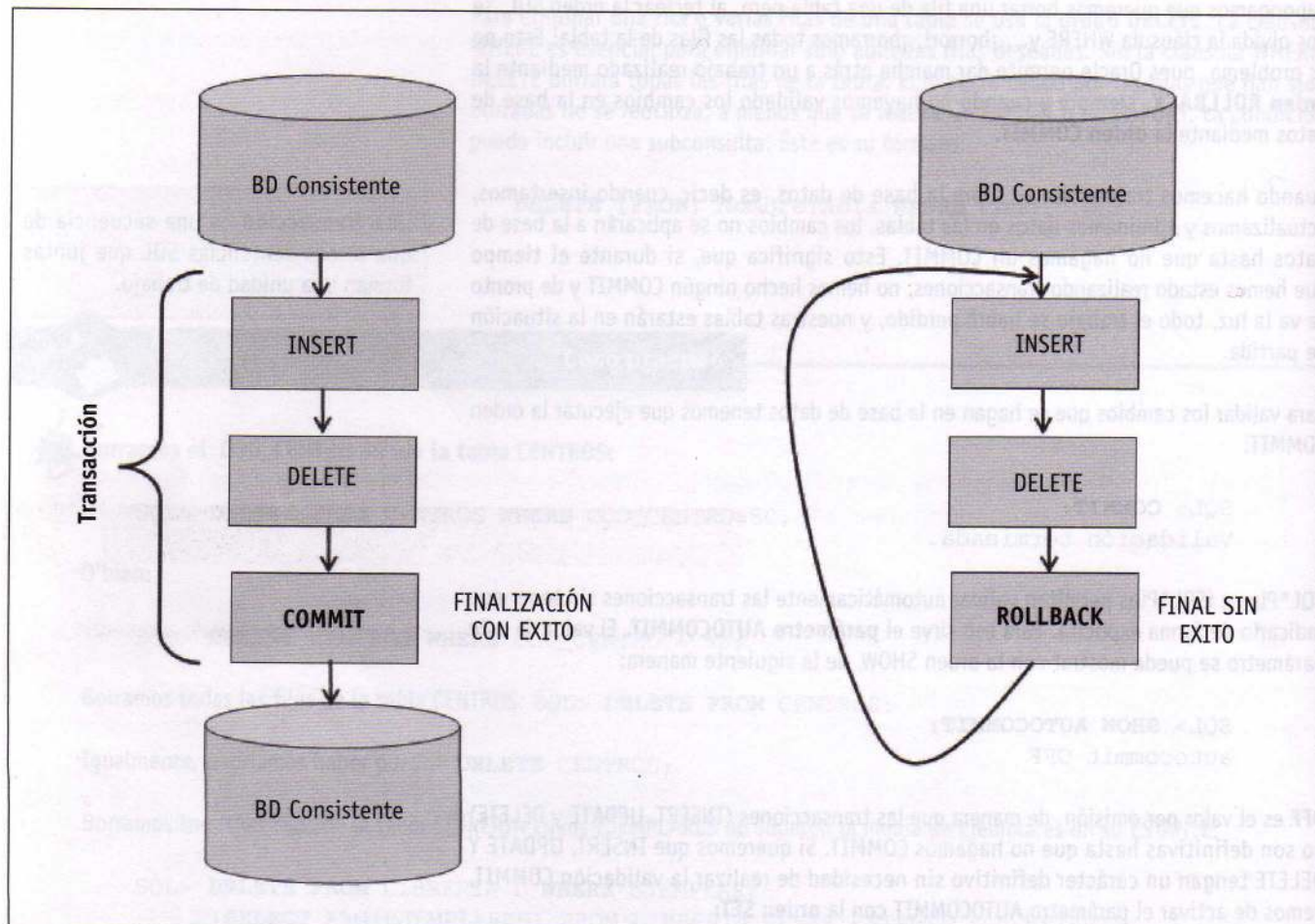


Figura 6.1. Transacciones.



Caso práctico

6 Partimos de la tabla DEPART con todas sus filas. El usuario SCOTT abre una sesión desde SQL*Plus y ejecuta la orden: `SELECT * FROM DEPART;` para consultar las filas de la tabla DEPART.

- SCOTT abre otra sesión ahora desde iSQL*Plus, ejecuta la misma orden para ver el contenido de la tabla. Ambos ven las mismas filas.
- Desde SQL*Plus borra una fila de la tabla DEPART: `DELETE DEPART WHERE DEPT_NO=20;` y consulta de nuevo la tabla. Observa que hay una fila menos.

6. Manipulación de datos. INSERT, UPDATE y DELETE

6.5 ROLLBACK, COMMIT y AUTOCOMMIT



(Continuación)

- Desde iSQL*Plus, consulta el contenido de la tabla DEPART. Observa que se muestran todas las filas.
- Desde SQL*Plus, ejecuta la orden: COMMIT; Todos los cambios realizados se validan en la base de datos.
- Desde iSQL*Plus, consulta otra vez el contenido de la tabla DEPART. Observa que ahora no se muestra la fila borrada desde SQL*Plus.
- Desde SQL*Plus, ejecuta la orden: DELETE DEPART; después consulta el contenido de la tabla, y observa que no se muestra ninguna fila.
- Se ha confundido al ejecutar la orden porque falta la cláusula WHERE en la sentencia DELETE, entonces ejecuta: ROLLBACK; De nuevo, consulta el contenido de la tabla: se muestran los datos desde el último COMMIT.

Actividades propuestas



- 6** Práctica las órdenes ROLLBACK y COMMIT abriendo dos sesiones SQL con el mismo usuario y realiza transacciones sobre tus tablas.

A. COMMIT implícito

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

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

Usar cualquiera de estas órdenes es como usar COMMIT.

B. ROLLBACK automático

Si, después de haber realizado cambios en nuestras tablas, se produce un fallo del sistema (por ejemplo, se va la luz) y no hemos validado el trabajo, Oracle hace un ROLLBACK automático sobre cualquier trabajo no validado. Esto significa que tendremos que repetir el trabajo cuando pongamos en marcha la base de datos.



6. Manipulación de datos. INSERT, UPDATE y DELETE

Conceptos básicos

Conceptos básicos



A continuación se muestra un resumen de las órdenes vistas en el tema:

INSERT

Inserción de una fila:

```
INSERT INTO NombreTabla [(columna [, columna] ...)]  
VALUES (valor [, valor] ...);
```

Inserción multifila:

```
INSERT INTO NombreTabla1 [(columna [, columna] ...)]  
SELECT {columna [, columna] ... | *}  
FROM NombreTabla2 [CLÁUSULAS DE SELECT];
```

UPDATE

Modificación de filas:

```
UPDATE <NombreTabla>  
SET columna1 = valor1, ..., columnan = valorn  
WHERE condición;
```

Modificación de filas con SELECT:

```
UPDATE <NombreTabla>  
SET columna1 = valor1, columna2 = valor2, ...  
WHERE columna3=(SELECT ....);  
UPDATE <NombreTabla>  
SET (columna1, columna2, ...)=(SELECT col1, col2, ...)  
WHERE condición;  
UPDATE <NombreTabla>  
SET columna1 = (SELECT col1 ... ), columna2 = (SELECT col2 ... )  
WHERE condición;
```

DELETE

Borrado de filas:

```
DELETE [FROM] NombreTabla WHERE condición;
```

TRANSACCIONES

Validar los cambios:

```
COMMIT;
```

Abortar transacciones:

```
ROLLBACK;
```


6. Manipulación de datos. INSERT, UPDATE y DELETE






Actividades complementarias



Actividades complementarias



Tablas PERSONAL, PROFESORES Y CENTROS

-  Modifica el número de plazas con un valor igual a la mitad en aquellos centros con menos de dos profesores.
-  Elimina los centros que no tengan personal. ✓
-  Añade un nuevo profesor en el centro o en los centros cuyo número de administrativos sea 1 en la especialidad de 'IDIOMA', con DNI 8790055 y de nombre 'Clara Salas'.
-  Borra al personal que esté en centros de menos de 300 plazas y con menos de dos profesores.
-  Borra a los profesores que estén en la tabla PROFESORES y que no estén en la tabla PERSONAL.

Tablas ARTICULOS, FABRICANTES, TIENDAS, PEDIDOS Y VENTAS (hacer DESC de las tablas)

Un almacén de distribución de artículos desea mantener información sobre las ventas hechas por las tiendas que compran al almacén. Dispone de las siguientes tablas para mantener esta información:















ARTICULOS: almacena cada uno de los artículos que el almacén puede abastecer a las tiendas. Cada artículo viene determinado por las columnas: ARTICULO, COD_FABRICANTE, PESO y CATEGORIA. La categoría puede ser 'Primera', 'Segunda' o 'Tercera'.

FABRICANTES: contiene los países de origen de los fabricantes de artículos. Cada COD_FABRICANTE tiene su país.

TIENDAS: almacena los datos de las tiendas que venden artículos. Cada tienda se identifica por su NIF.

PEDIDOS: son los pedidos que realizan las tiendas al almacén. Cada pedido se identifica por: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_PEDIDO. Cada fila de la tabla representa un pedido.

VENTAS: almacena las ventas de artículos que hace cada una de las tiendas. Cada venta se identifica por: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_VENTA. Cada fila de la tabla representa una venta.

-  Da de alta un nuevo artículo de 'Primera' categoría para los fabricantes de 'FRANCIA' y abastece con 5 unidades de ese artículo a todas las tiendas y en la fecha de hoy.
-  Inserta un pedido de 20 unidades en la tienda '1111-A' con el artículo que mayor número de ventas haya realizado.
-  Da de alta una tienda en la provincia de 'MADRID' y abastécela con 20 unidades de cada uno de los artículos existentes.
-  Da de alta dos tiendas en la provincia de 'SEVILLA' y abastécelas con 30 unidades de artículos de la marca de fabricante 'GALLO'.
-  Realiza una venta para todas las tiendas de 'TOLEDO' de 10 unidades en los artículos de 'Primera' categoría.
-  Para aquellos artículos de los que se hayan vendido más de 30 unidades, realiza un pedido de 10 unidades para la tienda con NIF '5555-B' con la fecha actual.
-  Cambia los datos de la tienda con NIF '1111-A' igualándolos a los de la tienda con NIF '2222-A'.
-  Cambia todos los artículos de 'Primera' categoría a 'Segunda' categoría del país 'ITALIA'.
-  Modifica aquellos pedidos en los que la cantidad pedida sea superior a las existencias del artículo, asignando el 20 por 100 de las existencias a la cantidad que se ha pedido.
-  Elimina aquellas tiendas que no han realizado ventas.
-  Elimina los artículos que no hayan tenido ni compras ni ventas.
-  Borra los pedidos de 'Primera' categoría cuyo país de procedencia sea 'BÉLGICA'.
-  Borra los pedidos que no tengan tienda.
-  Resta uno a las unidades de los últimos pedidos de la tienda con NIF '5555-B'.