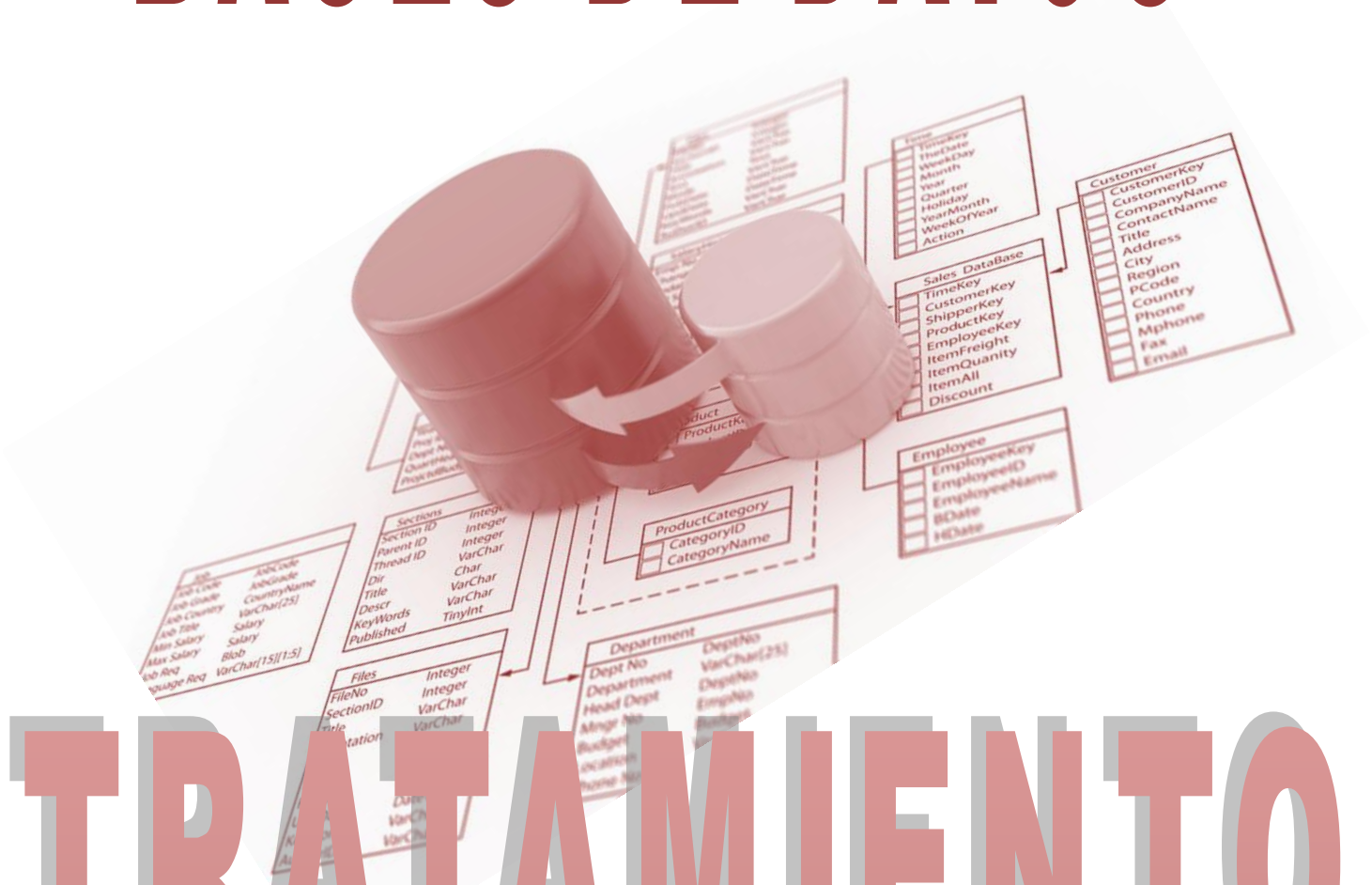
BASES DE DATOS



TRATAMIENTO DE DATOS

1 Introducción

Hasta ahora hemos estudiado el cómo recuperar datos almacenados en las tablas de nuestra base de datos. En este tema vamos a tratar el de la **actualización** de esos datos, es decir insertar nuevas filas, borrar filas o cambiar el contenido de las filas de una tabla. Estas operaciones modifican los datos almacenados en las tablas pero no su estructura, ni su definición.

Empezaremos por ver cómo insertar nuevas filas (con la sentencia INSERT INTO), veremos una variante (la sentencia SELECT... INTO), después veremos cómo borrar filas de una tabla (con la sentencia DELETE) y por último cómo modificar el contenido de las filas de una tabla (con la sentencia UPDATE). Si trabajamos en un entorno multiusuario, todas estas operaciones se podrán realizar siempre que tengamos los permisos correspondientes.

2 Inserción de datos en una tabla. INSERT

En general, un SGBD basado en SQL proporciona tres maneras de añadir nuevas filas de datos a una base de datos:

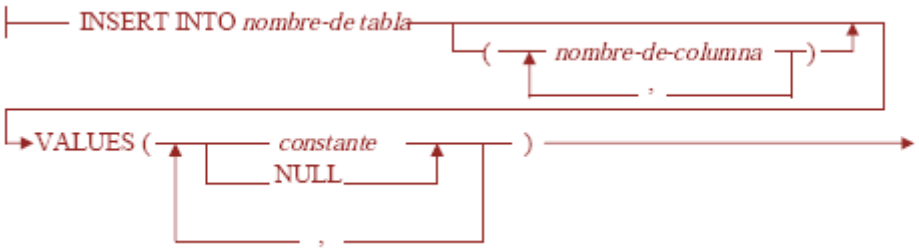
_ Una sentencia INSERT de una fila añade una única nueva fila de datos a una tabla.

_ Una sentencia INSERT multifila extrae filas de datos de otra parte de la base de datos y las añade a una tabla. Se utiliza habitualmente en procesamiento de fin de periodo cuando filas “antiguas” de una tabla se trasladan a una tabla histórica.

_ Una utilidad de carga masiva añade datos a una tabla desde un archivo externo a la base de datos. Se utiliza habitualmente para cargar inicialmente la base de datos o para incorporar datos transferidos desde otro sistema informático.

2.1 Insertar una fila

La cláusula **INTO** de la sentencia INSERT especifica la tabla que recibe la nueva fila (la tabla destino) y la cláusula **VALUE** especifica los valores de los datos que contendrá la nueva fila. La lista de columnas indica qué valor va a qué columna de la nueva fila.



La lista de valores y la lista de columnas deben contener el mismo número de elementos y el tipo de dato de cada valor debe ser compatible con el tipo de dato de la columna correspondiente, o en caso contrario se producirá un error.

Ejemplo:

Se acaba de contratar un nuevo vendedor, Henry Jacobsen, con los siguientes datos personales:

***Nombre**: Henry Jacobsen*
***Edad**: 36*
***Número de empleado**: 111*
***Título**: Director de ventas*
***Oficina**: Atlanta (número de oficina 13)*
***Fecha de contrato**: 25 de julio de 1990*
***Cuota**: Aún no asignada*
***Ventas anuales hasta la fecha**: $0.00*

La sentencia INSERT que añade al Sr. Jacobsen a la base de datos como nuevo vendedor es la siguiente:

```
INSERT INTO REPVENTAS
(NOMBRE, EDAD, NUM_EMPL, VENTAS, TITULO, CONTRATO, OFICINA_REP)
VALUES ('Henry Jacobsen', 36, 111, 0.00, 'Dir Ventas', '25-JUL-90', 13);
```

Si todo se ha desarrollado correctamente, nos mostrará el siguiente mensaje:

1 fila insertada.

Las filas de una tabla no están ordenadas, por lo que no existe la noción de insertar la fila "al comienzo" o "al final" o "entre dos filas" de la tabla. Una consulta posterior de la tabla REPVENTAS incluirá la nueva fila, pero puede aparecer en cualquier punto entre las filas del resultado de la consulta.

Ejemplo:

Supongamos que el Sr. Jacobsen recibe ahora su primer pedido, de InterCorp, un nuevo cliente que tiene asignado el número de cliente 2.126. El pedido es para 20 Widgets ACI-41004, por un precio total de $2.340, y le ha sido asignado el número de pedido 113.069.

```
INSERT INTO CLIENTES
(EMPRESA, NUM_CLIE, LIMITE_CREDITO, REP_CLIE)
VALUES ('InterCorp', 2126, 15000.00, 111);

INSERT INTO PEDIDOS
(IMPORTE, FAB, PRODUCTO, CANT, FECHA_PEDIDO, NUM_PEDIDO, CLIE, REP)
VALUES (2340.00, 'ACI', '41004', 20, GETDATE(), 113069, 2126, 111);
```

La sentencia INSERT puede ser larga si hay muchas columnas de datos, pero su formato sigue siendo muy sencillo. La segunda sentencia INSERT utiliza la constante de sistema **GETDATE**() en la cláusula VALUES, haciendo que se inserte la fecha actual como fecha de pedido.

En la práctica, los datos referentes a un nuevo cliente, un nuevo pedido o un nuevo vendedor casi siempre se añaden a una base de datos mediante una aplicación que permita realizar la entrada de datos a través de un formulario. Cuando la entrada de datos está completa en el formulario, la aplicación inserta la nueva fila de datos utilizando sentencias SQL que están "incrustadas" dentro de la propia aplicación. Sin embargo, independientemente de que se utilice SQL dentro de una aplicación o de forma interactiva, la sentencia INSERT es la misma.

Cuando SQL inserta una nueva fila de datos en una tabla, automáticamente asigna un valor **NULL** a cualquier columna cuyo nombre falte en la lista de columnas de la sentencia INSERT.

Ejemplo:

INSERT INTO REPVENTAS
(NOMBRE, EDAD, NUM_EMPL, VENTAS, TITULO, CONTRATO, OFICINA_REP)
VALUES ('Henry Jacobsen', 36, 111, 0.00, 'Dir Ventas', '25-JUL-90, 13);

Como resultado, la fila recién añadida tiene un valor NULL, en las columnas CUOTA y DIRECTOR que han sido omitidas.

Podemos hacer más explícita la asignación del valor NULL incluyendo las columnas en la lista y asignando la palabra clave NULL:

INSERT INTO REPVENTAS
(NOMBRE, EDAD, NUM_EMPL, VENTAS, TITULO, CONTRATO, OFICINA_REP, CUOTA, DIRECTOR)
VALUES ('Henry Jacobsen', 36, 111, 0.00, 'Dir Ventas', '25-JUL-90, 13, NULL, NULL);

Podemos omitir la lista de columnas de la sentencia INSERT. En este caso SQL genera automáticamente una lista formada por todas las columnas de la tabla, en el mismo orden en que aparecen en la tabla.

Ejemplo:

INSERT INTO REPVENTAS
VALUES (111, 'Henry Jacobsen', 36, 13, 'Dir Ventas', '25-JUL-90', NULL, NULL, 0.00);

2.2 Insertar varias filas

En esta forma de la sentencia INSERT, los valores de datos para las nuevas filas no son especificados explícitamente dentro del texto de la sentencia. En su lugar, la fuente de las nuevas filas es una consulta de base de datos especificada en la sentencia.

Ejemplo:

Supongamos que se desea copiar el número de pedido, la fecha y el importe de todos los pedidos remitidos con anterioridad al 1 de enero de 2014, desde la tabla PEDIDOS en otra tabla llamada ANTPEDIDOS.

INSERT INTO ANTPEDIDOS (NUM_PEDIDO, FECHA_PEDIDO, IMPORTE)
SELECT NUM_PEDIDO, FECHA_PEDIDO, IMPORTE
FROM PEDIDOS
WHERE FECHA_PEDIDO < '01-ENE-14';

Lo mismo que la sentencia INSERT de una sola fila, la sentencia identifica la tabla ANTPEDIDOS que va a recibir las nuevas filas y las columnas que reciben los datos,. El resto de la sentencia es una consulta que recupera datos de la tabla PEDIDOS.

El estándar SQL especifica varias **restricciones** lógicas sobre la consulta que aparece dentro de la sentencia INSERT:

_ La consulta no puede contener una cláusula ORDER BY.

_ El resultado de la consulta debe contener el mismo número de columnas que hay en la lista de columnas de la sentencia INSERT y los tipos de los datos deben ser compatibles columna a columna.

_ La consulta no puede ser la UNION de varias sentencias SELECT diferentes. Sólo puede especificarse una única sentencia SELECT.

_ La tabla destino de la sentencia INSERT no puede aparecer en la cláusula FROM de la consulta o de ninguna subconsulta que ésta contenga.

2.3 Carga masiva

Los datos a insertar en una base de datos son con frecuencia extraídos de otro sistema automático o recolectados de otros lugares y almacenados en un archivo secuencial. Para cargar los datos en una tabla, se podría escribir un programa con un bucle que leyera cada registro del archivo y utilizara la sentencia INSERT de una fila para añadir la fila a la tabla.

Sin embargo, el recargo de hacer que el SGBD ejecute repetidamente sentencias INSERT de una fila puede ser bastante alto. Si insertar una sola fila tarda medio segundo para una carga de sistema típica, éste es un rendimiento probablemente aceptable para un programa interactivo. Pero este rendimiento rápidamente pasa a ser inaceptable cuando se aplica a la tarea de cargar 50.000 filas de datos de una vez. En este caso, la carga de los datos requeriría más de 6 horas.

Por esta razón, todos los productos SGBD comerciales incluyen una utilidad de carga masiva que carga los datos desde un archivo a una tabla a alta velocidad.

3 Eliminación de datos de una tabla. DELETE

Una fila de datos se suprime de una base de datos cuando la entidad representada por la fila desaparece del mundo exterior. Por ejemplo:

_ Cuando un cliente cancela un pedido, la correspondiente fila de la tabla PEDIDOS debe ser suprimida.

_ Cuando un vendedor abandona la empresa, la fila correspondiente de la tabla REPVENTAS debe ser eliminada.

_ Cuando una oficina de ventas se cierra, la fila correspondiente de la tabla OFICINAS debe ser cancelada. Si los vendedores de la oficina son despedidos, sus filas también deberían ser suprimidas de la tabla REPVENTAS. Si son reasignados, sus columnas OFICINA_REP deben ser actualizadas.

3.1 Eliminar filas

La sentencia DELETE elimina filas seleccionadas de datos de una única tabla. La cláusula FROM especifica la tabla destino que contiene las filas. La cláusula WHERE especifica qué filas de la tabla van a ser suprimidas.

Ejemplo:
Eliminar a Henry Jacobsen de la tabla REPVENTAS:

DELETE FROM REPVENTAS
WHERE NOMBRE = 'Henry Jacobsen';

La cláusula WHERE de este ejemplo identifica una sola fila de la tabla REPVENTAS, que SQL elimina de dicha tabla.

Ejemplo:
Eliminar todos los pedidos de InterCorp (número de cliente 2.126):

DELETE FROM PEDIDOS
WHERE CLIE = 2126;

En este caso, la cláusula WHERE selecciona varias filas de la tabla PEDIDOS y SQL elimina las filas seleccionadas de la tabla.

Ejemplo:
Suprimir todos los pedidos remitidos antes del 15 de noviembre de 2014:

DELETE FROM PEDIDOS
WHERE FECHA_PEDIDO < '15-NOV-14';

3.2 Eliminar todas las filas

Si se omite la cláusula WHERE de una sentencia DELETE, **se suprimen todas las filas de la tabla destino.**

Ejemplo:
Suprimir todos los pedidos:

DELETE FROM PEDIDOS;

Esta sentencia DELETE produce una tabla vacía, no borra la tabla PEDIDOS de la base de datos. La definición de la tabla PEDIDOS y sus columnas siguen estando almacenadas en la base de datos y podemos insertar nuevas filas.

Para eliminar la definición de la tabla de la base de datos, debe utilizarse la sentencia **DROP TABLE**.

Por eso es importante especificar siempre una condición de búsqueda y tener cuidado de que se seleccionan realmente filas que se desean. Cuando se utiliza SQL interactivo, es buena idea utilizar primero la cláusula WHERE en una sentencia SELECT para visualizar las filas seleccionadas, asegurarse de que son las que realmente se desea suprimir y sólo entonces utilizar la cláusula WHERE en la sentencia DELETE.

3.3 Utilizar subconsultas

A veces, la selección de las filas a suprimir debe efectuarse en base a datos contenidos en otras tablas.

Ejemplo:
Suprimir los pedidos realizados por la vendedora 102 (Sue Smith):

```
DELETE FROM PEDIDOS
WHERE REP = 102;
```

Imaginemos que no sabemos cuál es el número de vendedor de Sue Smith y tampoco queremos consultarlo en la base de datos antes de hacer el borrado.

```
DELETE FROM PEDIDOS, REPVENTAS
WHERE PEDIDOS.REP = REPVENTAS.NUM_EMPL
AND REPVENTAS.NOMBRE = 'Sue Smith:'
```

ATENCIÓN: no se puede utilizar una combinación o "***join***" en una sentencia DELETE. La anterior sentencia DELETE es ilegal y nos daría el siguiente mensaje:

Error: Más de una tabla especificada en la cláusula FROM.

El modo de manejar la petición es hacer que en la condición de búsqueda haya una subconsulta. He aquí una forma válida de la sentencia DELETE que realiza la petición:

```
DELETE FROM PEDIDOS
WHERE REP = (SELECT NUM_EMPL
FROM REPVENTAS
WHERE NOMBRE = 'Sue Smith');
```

Primero, la consulta halla el número de empleado de Sue Smith (el 102) y después la cláusula WHERE selecciona los pedidos tratados por ese número de empleado, que son los que finalmente se borran.

4 Modificación de datos de una tabla. UPDATE

Los valores de los datos almacenados en una base de datos se modifican cuando se producen cambios en los elementos correspondientes del mundo exterior. Por ejemplo:

_ Cuando un cliente llama para modificar la cantidad de un pedido, la columna CANT de la fila apropiada en la tabla PEDIDOS debe ser modificada.

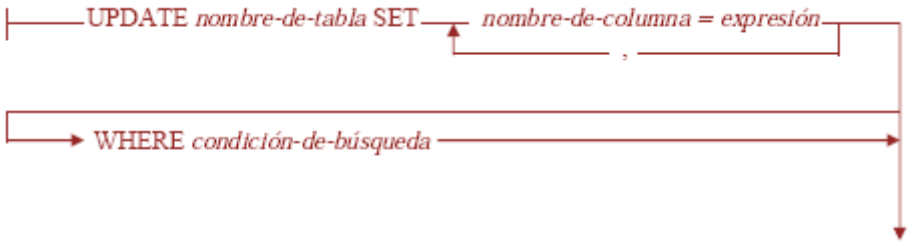
_ Cuando un director se traslada de una oficina a otra, la columna DIR en la tabla OFICINAS y la columna OFICINA_REP en la tabla REPVENTAS deben ser modificadas para que reflejen la nueva situación.

4.1 Modificar una o más filas

La sentencia **UPDATE** modifica los valores de una o más columnas en las filas seleccionadas de una única tabla. La tabla destino a actualizar se indica en la sentencia y es necesario disponer de permiso para actualizar la tabla, así como para cada una de las columnas individuales que se desean modificar.

La cláusula **WHERE** selecciona las filas de la tabla a modificar.

La cláusula **SET** especifica qué columnas se van a actualizar y calcula los nuevos valores.



Haciendo uso de la clausula WHERE solo se actualizan las filas que cumplan la condición.

Ejemplo:

Asignar a la empresa Acme Manufacturing el límite de crédito 60.000 y la vendedora Mary Jones (número de empleado 109).

UPDATE CLIENTES
SET LIMITE_CREDITO = 60000.00, REP_CLIE = 109
WHERE EMPRESA = 'Acme Mfg.';

En este ejemplo, la cláusula WHERE identifica una sola fila de la tabla CLIENTES y la cláusula SET asigna nuevos valores a dos de las columnas de esta fila.

Ejemplo:

Transferir todos los vendedores de la oficina de Chicago (número 12) a la oficina de New York (número 11) y rebajar sus cuotas un 10 por 100.

UPDATE REPVENTAS
*SET OFICINA_REP = 11, CUOTA = CUOTA * 0.9*
WHERE OFICINA_REP = 12;

En este caso, la cláusula WHERE selecciona varias filas de la tabla REPVENTAS y el valor de las columnas OFICINA_REP y CUOTA se modifica en todas ellas.

La cláusula SET en la sentencia UPDATE es una lista de asignaciones separadas por comas y en la que se debe cumplir lo siguiente:

- Cada asignación identifica una columna destino a actualizar y especifica cómo calcular el nuevo valor para la columna destino.
- Cada columna destino debería aparecer solamente una vez en la lista; no debería haber dos asignaciones para la misma columna destino.
- No pueden incluirse funciones de columna ni subconsultas en la clausula SET.
- La expresión en cada asignación puede ser cualquier expresión SQL válida que genere un valor tipo de dato apropiado para la columna destino.
- La expresión se debe poder calcular con los valores de la fila que actualmente está actualizándose en la tabla destino.

4.2 Modificar todas las filas

La cláusula WHERE en la sentencia UPDATE es opcional. Si se omite la cláusula WHERE, entonces se actualizan todas las filas de la tabla destino.

Ejemplo:

Elevar todas las cuotas un 5%:

```
UPDATE REPVENTAS
SET CUOTA = 1.05 * CUOTA;
```

4.3 UPDATE con subconsulta

Al igual que con la sentencia DELETE, las subconsultas pueden jugar un papel importante en la sentencia UPDATE ya que permiten seleccionar las filas a actualizar en base a información contenida en otras tablas.

Ejemplo:

Elevar en 5.000 el límite de crédito de cualquier cliente que haya hecho un pedido atendido por la vendedora Sue Smith:

```
UPDATE CLIENTES
SET LIMITE_CREDITO = LIMITE_CREDITO + 5000.00
WHERE REP = (SELECT NUM_EMPL
FROM REPVENTAS
WHERE NOMBRE = 'Sue Smith');
```

5 Inserciones, borrados y modificaciones e integridad referencial

En bases de datos en las que hemos implementado integridad referencial en algunas de sus tablas debemos considerar las reglas de borrado y modificaciones que impusimos en el diseño a la hora de eliminar o modificar registros.

5.1 Inserción

Es posible que cuando queramos hacer inserciones individuales o masivas de datos, la integridad referencial suponga una fuente de errores, especialmente cuando hay relaciones entre campos de dos tablas.

Este tipo de problemas se pueden evitar insertando primero los datos e impidiendo después las restricciones de integridad.

En todo caso, para inserciones masivas debemos prestar atención al orden de inserción de los datos. Para ello, introduciremos primero los datos de las tablas principales (aquellas a las que apuntan las claves ajenas) o las que no tienen claves ajenas y después los datos de las tablas secundarias o las que contienen claves ajenas que apuntan a dichas tablas principales.

5.2 Modificación

La modificación de datos no suele generar problemas ya que se suele definir la integridad referencial de tipo **CASCADE** de modo que cualquier cambio en atributos clave se propaga automáticamente a las claves ajenas con las que se relaciona.

Si será fuente de errores si la restricción es del tipo **SET NULL**.

5.3 Borrado

Cuando borramos registros debemos tener en cuenta el tipo de borrado definido. Si, como suele ocurrir, se definió el borrado en modo CASCADE, los datos de registros relacionados se borrarán automáticamente en todas las tablas, de forma que esta operación debe hacerse cuando se está muy seguro de sus implicaciones.

- Si el modo es **CASCADE** debemos eliminar en primer lugar los registros de las tablas principales. Por ejemplo, al eliminar un departamento desaparecerían también los registros de profesores que trabajen en él.
- Si el modo es **NO ACTION** o **RESTRICT** debemos eliminar en primer lugar los registros de las tablas que contienen las claves ajenas y después los de las tablas principales. En este caso tendríamos que eliminar primero los profesores de un departamento para poder eliminar el departamento.
- Si el modo es **SET NULL** deberemos borrar los datos de casa tabla en cualquier orden.

5.4 Restricciones foreign key en ORACLE

La restricción **foreign key** tiene la cláusula ***on delete***, que es opcional. Esta cláusula especifica cómo debe actuar Oracle frente a eliminaciones en las tablas referenciadas en la restricción.

Las opciones para estas cláusulas son las siguientes:

- **set null**: indica que si eliminamos un registro de la tabla referenciada (TABLA2) cuyo valor existe en la tabla principal (TABLA1), dicho registro se elimine y los valores coincidentes en la tabla principal se actualicen a "null".

- **cascade**: indica que si eliminamos un registro de la tabla referenciada en una foreign key (TABLA2), los registros coincidentes en la tabla principal (TABLA1),

también se eliminen; es decir, si eliminamos un registro al cual una clave foránea referencia, dicha eliminación se extiende a la otra tabla (integridad referencial en cascada).

- **no action**: es la predeterminada; indica que si se intenta eliminar un registro de la tabla referenciada por una foreign key, Oracle no lo permita y muestre un mensaje de error. Se establece omitiendo la cláusula on delete al establecer la restricción.

La sintaxis completa para agregar esta restricción a una tabla es la siguiente:

```
ALTER TABLE TABLA1
 ADD CONSTRAINT NOMBRERESTRICCION
 FOREIGN KEY (CAMPOCLAVEFORANEA)
 REFERENCES TABLA2(CAMPOCLAVEPRIMARIA)
 ON DELETE OPCION;
```

Ejemplo:

Definimos una restricción foreign key a la tabla LIBROS estableciendo el campo *codigoeditorial* como clave foránea que referencia al campo *codigo* de la tabla EDITORIALES. La tabla EDITORIALES tiene como clave primaria el campo *codigo*. Especificamos la acción en cascada para las eliminaciones:

```
ALTER TABLE LIBROS
 ADD CONSTRAINT FK_LIBROS_CODIGOEDITORIAL
 FOREIGN KEY (CODIGOEDITORIAL)
 REFERENCES EDITORIALES (CODIGO)
 ON DELETE CASCADE;
```

Si luego de establecer la restricción anterior, eliminamos una editorial de EDITORIALES cuyo valor de código está presente en LIBROS, se elimina dicha editorial y todos los libros de tal editorial.

Para definir una restricción foreign key sobre la tabla LIBROS estableciendo el campo *codigoeditorial* como clave foránea que referencia al campo *codigo* de la tabla EDITORIALES especificando la acción de poner a NULL escribimos:

```
ALTER TABLE LIBROS
 ADD CONSTRAINT FK_LIBROS_CODIGOEDITORIAL
 FOREIGN KEY (CODIGOEDITORIAL)
 REFERENCES EDITORIALES (CODIGO)
 ON DELETE SET NULL;
```

Si luego de establecer la restricción anterior, eliminamos una editorial de EDITORIALES cuyo valor de código está presente en LIBROS, se elimina dicha editorial y todos los valores de libros que coinciden con tal editorial se ponen a null.

En ORACLE la restricción foreign key **NO** tiene una cláusula para especificar acciones para actualizaciones. Si intentamos actualizar un registro de la tabla referenciada por una restricción foreign key cuyo valor de clave primaria existe referenciado en la tabla que tiene dicha restricción, la acción no se ejecuta y aparece un mensaje de error.

Esto sucede porque, por defecto (y como única opción), para actualizaciones existe NO ACTION.

NOTA: Una restricción **foreign key** no puede modificarse, debe eliminarse (con **alter table** y **drop constraint**) y volverse a crear.

6 Transacciones. Sentencias de procesamiento de transacciones: COMMIT, ROLLBACK

Las transacciones aportan una fiabilidad superior a las bases de datos. Si disponemos de una serie de operaciones SQL que deben ejecutarse en conjunto, con el uso de transacciones podemos tener la certeza de que nunca nos quedaremos a medio camino de su ejecución. De hecho, podríamos decir que las transacciones aportan una característica de "***deshacer***" a las aplicaciones de bases de datos. Las tablas que soportan transacciones son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor, ya que las instrucciones se ejecutan o no en su totalidad.

Los pasos para usar transacciones son:

- Iniciar una transacción con el uso de la sentencia **START TRANSACTION** o **BEGIN**.
- Actualizar, insertar o eliminar registros en la base de datos.
- Si se quieren los cambios, completar la transacción con el uso de la sentencia **COMMIT**. Únicamente cuando se procesa un COMMIT los cambios hechos por las consultas serán permanentes.
- Si sucede algún problema, podemos hacer uso de la sentencia **ROLLBACK** para cancelar los cambios que han sido realizados por las consultas ejecutadas hasta el momento.

Ejemplo1:

Insertar un cliente nuevo y el pedido que acaba de realizar

```
BEGIN
INSERT INTO CLIENTES
(NUMCLIE, EMPRESA, REP_CLIE, LIMITE_CREDITO)
VALUES
(2130, 'Trivial Pursuit', NULL, 10000);
INSERT INTO PEDIDOS
(NUMPEDIDO, FECHAPEDIDO, CLIE, REP, ID_FAB, ID_PRODUCTO, CANT, IMPORTE)
VALUES
(115000,'15/10/1998',2130, 106 ,'FEA','112',3,444);
COMMIT;
END;
```

Ejemplo 2:

Introducir tres pedidos y actualizar las existencias y el valor de las existencias cada vez que se hace un pedido.

```
DECLARE
cantidad NUMBER;
fabricante VARCHAR(3);
producto VARCHAR(5);
BEGIN
cantidad:=1;
fabricante:='REI';
producto:='A245C';
INSERT INTO PEDIDOS
(NUMPEDIDO, FECHAPEDIDO, CLIE, REP, FAB, PRODUCTO, CANT, IMPORTE)
VALUES
(115400,'17/10/1998', 2101, 101, fabricante, producto, cantidad, 79);

UPDATE PRODUCTOS
SET EXISTENCIAS=EXISTENCIAS-cantidad, VALOR=EXISTENCIAS*PRECIO
WHERE ID_FAB=fabricante AND ID_PRODUCTO=producto;

cantidad:=2;
fabricante:='FEA';
producto:='114';
INSERT INTO PEDIDOS
(NUMPEDIDO, FECHAPEDIDO, CLIE, REP, FAB, PRODUCTO, CANT, IMPORTE)
VALUES
(115500,'18/10/1998', 2101, 101, fabricante, producto, cantidad, 79);

UPDATE PRODUCTOS
SET EXISTENCIAS=EXISTENCIAS-cantidad, VALOR=EXISTENCIAS*PRECIO
WHERE ID_FAB=fabricante AND ID_PRODUCTO=producto;

COMMIT;
END;
```

Ejemplo 3:

Vamos a ver como cancelar los cambios con ROLLBACK si ocurre algún problema. Al igual que antes introduciremos dos pedidos nuevos, pero vamos a “cometer un error” intentando introducir una clave duplicada en el segundo pedido.

NOTA: vamos a utilizar una función para mostrar mensajes por consola. Si el mensaje no se muestra ejecutar:

set serveroutput on

en la hoja de trabajo de SQL y después repetir la transacción.

```
DECLARE
cantidad NUMBER;
fabricante VARCHAR(3);
producto VARCHAR(5);

BEGIN
cantidad:=3;
fabricante:='QSA';
producto:='XK47';
INSERT INTO PEDIDOS
(NUMPEDIDO, FECHAPEDIDO, CLIE, REP, FAB, PRODUCTO, CANT, IMPORTE)
VALUES
(115600,'20/08/1998', 2101, 101, fabricante, producto, cantidad, 1260);

cantidad:=2;
fabricante:='FEA';
producto:='112';
INSERT INTO PEDIDOS
(NUMPEDIDO, FECHAPEDIDO, CLIE, REP, FAB, PRODUCTO, CANT, IMPORTE)
VALUES
(115600,'18/10/1998',2101,101,fabricante,producto,cantidad,310.8);

EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error en la transacción. Se deshacen las
modificaciones');
ROLLBACK;
END;
```

Si vamos a mirar la tabla PEDIDOS, vemos que no se ha realizado ninguna modificación en ella.

Ahora corregimos el “error” y ponemos el valor 115700 al NUMPEDIDO del segundo pedido. Ejecutamos y vemos que ambos pedidos han sido incorporados a la tabla sin ningún problema.

7 Políticas de bloqueo de tablas

Bloquear una tabla o vista permite que nadie más pueda hacer uso de la misma de modo que tenemos la exclusividad de lectura y/o escritura sobre ella.

7.1 Bloqueos por defecto

- Las sentencias del DML (Lenguaje de Manipulación de Datos) pueden producir bloqueos sobre las filas de la tabla:
- Una sentencia SELECT normal no bloquea filas.
- Las sentencias INSERT, UPDATE o DELETE realiza un bloqueo ROW EXCLUSIVE de las filas afectadas por el WHERE.
- La sentencia SELECT ... FOR UPDATE NOWAIT realiza un bloqueo ROW EXCLUSIVE de las filas afectadas por el WHERE.
- Las sentencias COMMIT y ROLLBACK desbloquean las filas bloqueadas anteriormente dentro de la transacción actual.
- Aunque una fila este bloqueada (por otra transacción), siempre podemos hacer una SELECT sobre esa fila. Los valores retornados son los anteriores al bloqueo.
- Las sentencias UPDATE Y DELETE pueden provocar o sufrir esperas si hay conflictos con otra transacción.

7.2 Bloquear una tabla

LOCK TABLE

Bloquea una tabla o vista:

```
LOCK TABLE [esquema.] table [opciones] IN lockmode MODE [NOWAIT]

LOCK TABLE [esquema.] view [opciones] IN lockmode MODE [NOWAIT]
```

donde opciones: PARTITION, SUBPARTITION, @dblink

donde lockmodes: EXCLUSIVE, SHARE, ROW EXCLUSIVE, SHARE ROW EXCLUSIVE
ROW SHARE* | SHARE UPDATE*

Si no ponemos NOWAIT Oracle esperará hasta que la tabla esté disponible.

Se pueden bloquear varias tablas en un solo comando si las escribimos separadas por comas.

```
LOCK TABLE tabla1,t abla2, tabla3 IN ROW EXCLUSIVE MODE;
```

UNLOCK TABLES

Desbloquea las tablas. Sólo podemos desbloquear todas las tablas a la vez.

Índice de contenidos

Fuentes

Consultas a la base de datos Joaquín Álvarez García/Pilar García López

IES Nº 1 de Gijón

aulaclic

Bases de Datos edit. RA-MA, autor Luis Hueso Ibáñez

Y otros.