

Tema 5 TRATAMIENTO DE DATOS

1. Introducción

Las bases de datos no tienen razón de ser sin la posibilidad de hacer operaciones para el tratamiento de la información almacenada en ellas. Por operaciones de tratamiento de datos se deben entender las acciones que permiten añadir información en ellas, modificarla o bien suprimirla.

En esta unidad podrás conocer que existen distintos medios para realizar el tratamiento de los datos. Desde la utilización de herramientas gráficas hasta el uso de instrucciones o sentencias del lenguaje SQL que permiten realizar ese tipo de operaciones de una forma menos visual pero con más detalle, flexibilidad y rapidez. El uso de unos mecanismos u otros dependerá de los medios disponibles y de nuestras necesidades como usuarios de la base de datos.

Pero la información no se puede almacenar en la base de datos sin tener en cuenta que debe seguir una serie de requisitos en las relaciones existentes entre las tablas que la componen. Todas las operaciones que se realicen respecto al tratamiento de los datos deben asegurar que las relaciones existentes entre ellos se cumplan correctamente en todo momento.

Por otro lado, la ejecución de las aplicaciones puede fallar en un momento dado y eso no debe impedir que la información almacenada sea correcta. O incluso el mismo usuario de las aplicaciones debe tener la posibilidad de cancelar una determinada operación y dicha cancelación no debe suponer un problema para que los datos almacenados se encuentren en un estado fiable.

Todo esto requiere disponer de una serie de herramientas que aseguren esa fiabilidad de la información, y que además puede ser consultada y manipulada en sistemas multiusuario sin que las acciones realizadas por un determinado usuario afecte negativamente a las operaciones de los demás usuarios.

El lenguaje SQL dispone de una serie de sentencias para la edición (inserción, actualización y borrado) de los datos almacenados en una base de datos. Ese conjunto de sentencias recibe el nombre de **Data Manipulation Language (DML)**.

2. Inserción de registros

La sentencia **INSERT** permite la inserción de nuevas filas o registros en una tabla existente.

El formato más sencillo de utilización de la sentencia **INSERT** tiene la siguiente sintaxis:

INSERT INTO nombre_tabla (lista_campos) VALUES (lista_valores);

Donde *nombre_tabla* será el nombre de la tabla en la que quieras añadir nuevos registros. En *lista_campos* se indicarán los campos de dicha tabla en los que se desea escribir los nuevos valores indicados en *lista_valores*. Es posible omitir la lista de campos (*lista_campos*), si se indican todos los valores de cada campo y en el orden en el que se encuentran en la tabla.

Tanto la lista de campos *lista_campos* como la de valores *lista_valores*, tendrán separados por comas cada uno de sus elementos. Hay que tener en cuenta también que cada campo de *lista_campos* debe tener un valor válido en la posición correspondiente de la *lista_valores* (Si no recuerdas los valores válidos para cada campo puedes utilizar la sentencia **DESCRIBE** seguida del nombre de la tabla que deseas consultar).

En el siguiente ejemplo se inserta un nuevo registro en la tabla **USUARIOS** en el que se tienen todos

los datos disponibles:

```
INSERT INTO USUARIOS (Login, Password, Nombre, Apellidos, Direccion, CP, Localidad,
Provincia, Pais, F_Nacimiento, F_Ingreso, Correo, Credito, Sexo) VALUES ('migrod86', '6PX5=V',
'MIGUEL ANGEL', 'RODRIGUEZ RODRIGUEZ', 'ARCO DEL LADRILLO,PASEO', '47001',
'VALLADOLID', 'VALLADOLID', 'ESPAÑA', '27/04/1977', '10/01/2008', 'migrod86@gmail.com',
200, 'H');
```

En este otro ejemplo, se inserta un registro de igual manera, pero sin disponer de todos los datos:

```
INSERT INTO USUARIOS (Login, Password, Nombre, Apellidos, Correo) VALUES ('natsan63',
'VBROMI', 'NATALIA', 'SANCHEZ GARCIA', 'natsan63@hotmail.com');
```

Al hacer un *INSERT* en el que no se especifiquen los valores de todos los campos, se obtendrá el valor *NULL* en aquellos campos que no se han indicado.

Si la lista de campos indicados no se corresponde con la lista de valores, se obtendrá un error en la ejecución. Por ejemplo, si no se indica el campo Apellidos pero sí se especifica un valor para dicho campo:

```
INSERT INTO USUARIOS (Login, Password, Nombre, Correo) VALUES ('caysan56', 'W4IN5U',
'CAYETANO', 'SANCHEZ CIRIZA', 'caysan56@gmail.com');
```

Se obtiene el siguiente error:

ORA-00913: demasiados valores

AUTOEVALUACIÓN:

¿Cuál de las siguientes sentencias INSERT es correcta?:

- INSERT INTO PRODUCTOS (Codigo, Nombre, Existencias) VALUES (3, Leche, 100);
- INSERT INTO PRODUCTOS (3, 'Leche', 100);
 - INSERT INTO PRODUCTOS (Codigo, Nombre, Existencias) VALUES (3, 'Leche', 100);
 - INSERT INTO PRODUCTOS (Codigo, Nombre, Existencias) VALUES ('Leche', 3, 100);

3. Modificación de registros

La sentencia *UPDATE* permite modificar una serie de valores de determinados registros de las tablas de la base de datos.

La manera más sencilla de utilizar la sentencia *UPDATE* tiene la siguiente sintaxis:

```
UPDATE nombre_tabla SET nombre_campo = valor [, nombre_campo = valor]...
[ WHERE condición ];
```

Donde *nombre_tabla* será el nombre de la tabla en la que quieras modificar datos. Se pueden especificar los nombres de campos que se deseen de la tabla indicada. A cada campo especificado se le debe asociar el nuevo valor utilizando el signo =. Cada emparejamiento *campo=valor* debe

separarse del siguiente utilizando comas (,).

La cláusula *WHERE* seguida de la condición es opcional (como pretenden indicar los corchetes). Si se indica, la actualización de los datos sólo afectará a los registros que cumplen la condición. Por tanto, ten en cuenta que si no indicas la cláusula *WHERE*, los cambios afectarán a todos los registros.

Por ejemplo, si se desea poner a 200 el crédito de todos los usuarios:

```
UPDATE USUARIOS SET Credito = 200;
```

En este otro ejemplo puedes ver la actualización de dos campos, poniendo a 0 el crédito y borrando la información del campo *Pais* de todos los usuarios:

```
UPDATE USUARIOS SET Credito = 0, Pais = NULL;
```

Para que los cambios afecten a determinados registros hay que especificar una condición. Por ejemplo, si se quiere cambiar el crédito de todas la mujeres, estableciendo el valor 300:

```
UPDATE USUARIOS SET Credito = 300 WHERE Sexo = 'M';
```

Cuando termina la ejecución de una sentencia *UPDATE*, se muestra la cantidad de registros (filas) que han sido actualizadas, o el error correspondiente si se ha producido algún problema. Por ejemplo podríamos encontrarnos con un mensaje similar al siguiente:

```
45 fila(s) actualizada(s).
```

4. Borrado de registros

La sentencia *DELETE* es la que permite eliminar o borrar registros de un tabla.

Esta es la sintaxis que debes tener en cuenta para utilizarla:

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

Al igual que hemos visto en las sentencias anteriores, *nombre_tabla* hace referencia a la tabla sobre la que se hará la operación, en este caso de borrado. Se puede observar que la cláusula *WHERE* es opcional. Si no se indica, debes tener muy claro que se borrará todo el contenido de la tabla, aunque la tabla seguirá existiendo con la estructura que tenía hasta el momento. Por ejemplo, si usas la siguiente sentencia, borrarás todos los registros de la tabla *USUARIOS*:

```
DELETE FROM USUARIOS;
```

Para ver un ejemplo de uso de la sentencia *DELETE* en la que se indique una condición, supongamos que queremos eliminar todos los usuarios cuyo crédito es cero:

```
DELETE FROM USUARIOS WHERE Credito = 0;
```

Como resultado de la ejecución de este tipo de sentencia, se obtendrá un mensaje de error si se ha producido algún problema, o bien, el número de filas que se han eliminado.

45 fila(s) suprimida(s).

AUTOEVALUACIÓN:

¿Si no se especifica una condición en la sentencia DELETE se borra todo el contenido de la tabla especificada?:

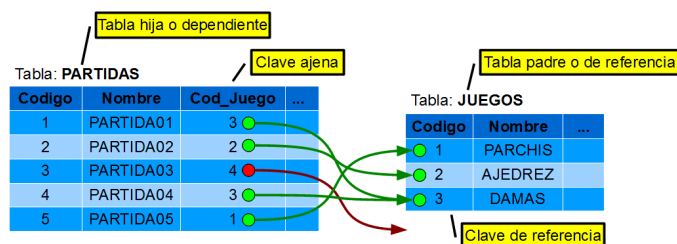
Verdadero.

Falso.

Integridad referencial

Dos tablas pueden ser relacionadas entre ellas si tienen en común uno o más campos, que reciben el nombre de **clave ajena**. La restricción de integridad referencial requiere que haya coincidencia en todos los valores que deben tener en común ambas tablas. Cada valor del campo que forma parte de la integridad referencial definida, debe corresponderse, en la otra tabla, con otro registro que contenga el mismo valor en el campo referenciado.

Siguiendo con el ejemplo de juegos **online**, supongamos que en una determinada partida de un juego, se han unido una serie de usuarios. En la tabla de *PARTIDAS* existe un campo de referencia al tipo de juego al que corresponde, mediante su código de juego. Por tanto, no puede existir ninguna partida cuyo código de juego no se corresponda con ninguno de los juegos de la tabla *JUEGOS*.



En este ejemplo, no se cumple la integridad referencial, porque la partida “*PARTIDA03*” corresponde al juego cuyo código es 4, y en la tabla *JUEGOS* no existe ningún registro con ese código.

Para que se cumpla la integridad referencial, todos los valores del campo *Cod_Juego* deben corresponderse con valores existentes en el campo *Codigo* de la tabla *JUEGOS*.

Cuando se habla de integridad referencial se utilizan los siguientes términos:

clave ajena: Es el campo o conjunto de campos incluidos en la definición de la restricción que deben hacer referencia a una clave de referencia. En el ejemplo anterior, la clave ajena sería el campo *Cod_Juego* de la tabla *PARTIDAS*.

Clave de referencia: Clave única o primaria de la tabla a la que se hace referencia desde una clave ajena. En el ejemplo, la clave de referencia es el campo *Codigo* de la tabla *JUEGOS*.

Tabla hija o dependiente: Tabla que incluye la clave ajena, y que, por tanto, depende de los valores existentes en la clave de referencia. Correspondería a la tabla *PARTIDAS* del ejemplo, que sería la tabla hija de la tabla *JUEGOS*.

Tabla padre o de referencia: Corresponde a la tabla que es referenciada por la clave ajena en la tabla hija. Esta tabla determina las inserciones o actualizaciones que son permitidas en la tabla hija, en función de dicha clave. En el ejemplo, la tabla *JUEGOS* es padre de la tabla *PARTIDAS*.

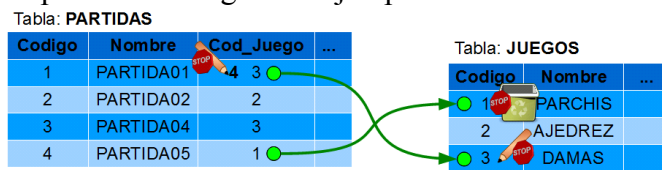
Integridad en actualización y supresión de registros

La relación existente entre la clave ajena y la clave padre tiene implicaciones en el borrado y modificación de sus valores.

Si se modifica el valor de la clave ajena en la tabla hija, debe establecerse un nuevo valor que haga referencia a la clave principal de uno de los registros de la tabla padre. De la misma manera, no se puede modificar el valor de la clave principal en un registro de la tabla padre, si una clave ajena hace referencia a dicho registro.

Los borrados de registros en la tabla de referencia también puede suponer un problema, ya que no pueden suprimirse registros que son referenciados con una clave ajena desde otra tabla.

Suponiendo el siguiente ejemplo:



En el registro de la partida con nombre “PARTIDA01” no puede ser modificado el campo *Cod_Juego* al valor 4, porque no es una clave ajena válida, puesto que no existe un registro en la tabla *JUEGOS* con esa clave primaria.

El código del juego “DAMAS” no puede ser cambiado, ya que hay registros en la tabla *PARTIDAS* que hacen referencia a dicho juego a través del campo *Cod_Juego*.

Si se eliminara en la tabla *JUEGOS* el registro que contiene el juego “PARCHIS”, la partida “PARTIDA05” quedaría con un valor inválido en el campo *Cod_Juego*.

Cuando se hace el borrado de registros en una tabla de referencia, se puede configurar la clave ajena de diversas maneras para que se conserve la integridad referencial:

No Permitir Supresión: Es la opción por defecto. En caso de que se intente borrar en la tabla de referencia un registro que está siendo referenciado desde otra tabla, se produce un error en la operación de borrado impidiendo dicha acción.

Supresión en Cascada: Al suprimir registros de la tabla de referencia, los registros de la tabla hija que hacían referencia a dichos registros, también son borrados.

Definir Nulo en Suprimir: Los valores de la clave ajena que hacían referencia a los registros que hayan sido borrados de la tabla de referencia, son cambiados al valor *NULL*.

5. Subconsultas y composiciones en órdenes de edición

Anteriormente has podido conocer una serie de instrucciones del lenguaje SQL que han servido para realizar operaciones de inserción, modificación y eliminación de registros. Tal como las hemos analizado, esas operaciones se realizan sobre registros de una misma tabla, pero vamos a ver que esas mismas sentencias pueden utilizarse de una forma más avanzada insertando consultas dentro de esas mismas operaciones de tratamiento de datos.

Por tanto, veremos que los resultados de las operaciones pueden afectar a más de una tabla, es decir, que con una misma instrucción se pueden añadir registros a más de una tabla, o bien actualizar o eliminar registros de varias tablas simultáneamente.

Los valores que se añadan o se modifiquen podrán ser obtenidos también como resultado de una consulta.

Además, las condiciones que hemos podido añadir hasta ahora a las sentencias, pueden ser también consultas, por lo que pueden establecerse condiciones bastante más complejas.

PARA SABER MÁS.

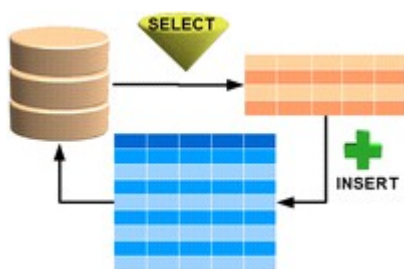
En este manual pueden encontrar una sección sobre las funciones agregadas y subconsultas (módulo 3). También puedes ver ejemplos en la parte final.

Texto enlace: Resumen de SQL con ejemplos, incluyendo material sobre subconsultas

URL: <http://es.scribd.com/doc/56646934/SQL-BASICO-Material-de-Apoyo>

Título: Enlace a material de apoyo de SQL básico

5.1. Inserción de registros a partir de una consulta



Anteriormente hemos visto la posibilidad de insertar registros en una tabla a través de la sentencia *INSERT*, por ejemplo:

```
INSERT INTO USUARIOS (LOGIN, PASSWORD, NOMBRE, APELLIDOS, CORREO)
VALUES ('natsan63', 'VBROMI', 'NATALIA', 'SANCHEZ GARCIA', 'natsan63@hotmail.com');
```

Esta misma acción se puede realizar usando una consulta *SELECT* dentro de la sentencia *INSERT*, así por ejemplo, la equivalente a la anterior sería:

```
INSERT INTO (SELECT LOGIN, PASSWORD, NOMBRE, APELLIDOS, CORREO FROM
USUARIOS) VALUES ('natsan63', 'VBROMI', 'NATALIA', 'SANCHEZ GARCIA',
'natsan63@hotmail.com');
```

Puedes observar que simplemente se ha sustituido el nombre de la tabla, junto con sus campos, por una consulta equivalente.

Y no sólo eso, sino que es posible insertar en una tabla valores que se obtienen directamente del resultado de una consulta. Supongamos por ejemplo, que disponemos de una tabla *USUARIOS_SIN_CREDITO* con la misma estructura que la tabla *USUARIOS*. Si queremos insertar en esa tabla todos los usuarios que tienen el crédito a cero:

```
INSERT INTO USUARIOS_SIN_CREDITO SELECT * FROM USUARIOS WHERE Credito =
0;
```

Observa que en ese caso no se debe especificar la palabra *VALUES*, ya que no se está especificando una lista de valores.

AUTOEVALUACIÓN

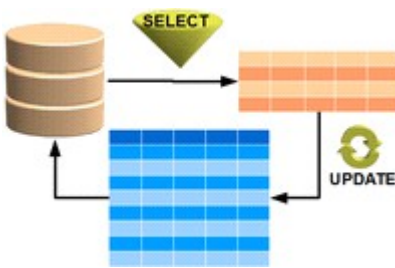
¿Cuál de las siguientes sentencias *INSERT* es la correcta para insertar en la tabla *CLIENTES* los nombres de los registros de la tabla *NUEVOS_CLIENTES*, suponiendo que los campos que contienen los nombres se llaman *Nombre_CLI* en la tabla *CLIENTES* y *Nombre_NCLI* en la tabla *NUEVOS_CLIENTES*?:

```

INSERT INTO CLIENTES (Nombre_CLI) VALUES Nombre_NCLI FROM
NUEVOS_CLIENTES;
INSERT INTO CLIENTES (Nombre_CLI) (SELECT Nombre_NCLI FROM
NUEVOS_CLIENTES);
INSERT INTO CLIENTES VALUES (SELECT Nombre_CLI, Nombre_NCLI FROM
NUEVOS_CLIENTES);
INSERT INTO NUEVOS_CLIENTES (Nombre_CLI) VALUES (SELECT
Nombre_NCLI FROM CLIENTES);

```

5.2. Modificación de registros a partir de una consulta



La acción de actualizar registros mediante la sentencia **UPDATE** también puede ser utilizada con consultas para realizar modificaciones más complejas de los datos. Las consultas pueden formar parte de cualquiera de los elementos de la sentencia **UPDATE**.

Por ejemplo, la siguiente sentencia modifica el crédito de aquellos usuarios que tienen una partida creada y cuyo estado es 1 (activada). El valor del crédito que se les asigna es el valor más alto de los créditos de todos los usuarios.

```

UPDATE USUARIOS SET Credito = (SELECT MAX(Credito) FROM USUARIOS) WHERE
Login IN (SELECT Cod_Crea FROM PARTIDAS WHERE Estado=1);

```

AUTOEVALUACIÓN

¿Cuál de las siguientes sentencias **UPDATE** es la correcta para actualizar en la tabla **USUARIOS** el crédito del usuario con código 3 para asignarle el mismo crédito que el del usuario con código 5?:

```

UPDATE USUARIOS SET Credito = Credito WHERE Codigo = 3 AND WHERE
Codigo = 5;

```

```

UPDATE USUARIOS SET Credito = (SELECT Credito FROM USUARIOS WHERE
Codigo = 3 AND WHERE Codigo = 5);

```

```

UPDATE USUARIOS SET Codigo = 5 WHERE (SELECT Credito FROM
USUARIOS WHERE Codigo = 3);

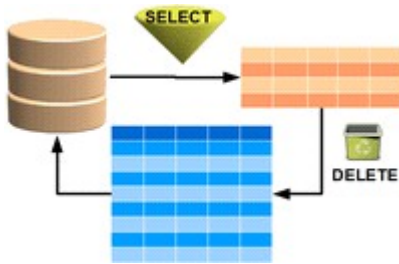
```

```

UPDATE USUARIOS SET Credito = (SELECT Credito FROM USUARIOS WHERE
Codigo = 5) WHERE Codigo = 3;

```


5.3. Supresión de registros a partir de una consulta



Al igual que las sentencias *INSERT* y *UPDATE* vistas anteriormente, también se pueden hacer borrados de registros utilizando consultas como parte de las tablas donde se hará la eliminación o como parte de la condición que delimita la operación.

Por ejemplo, si se ejecuta la siguiente sentencia:

```
DELETE FROM (SELECT * FROM USUARIOS, PARTIDAS WHERE Login=Cod_Crea AND Estado=0);
```

El resultado es que se eliminan determinados registros de las tablas *USUARIOS* y *PARTIDAS*, en concreto, aquellos usuarios que han creado alguna partida cuyo estado es 0 (desactivada).

Puedes observar que no se ha establecido ninguna condición *WHERE* en la sentencia, ya que se ha incluido dentro de la consulta. Otra manera de realizar la misma acción, pero utilizando la cláusula *WHERE* es la siguiente:

```
DELETE FROM (SELECT * FROM USUARIOS, PARTIDAS WHERE Login=Cod_Crea) WHERE Estado=0;
```

AUTOEVALUACIÓN

¿Cuál de las siguientes sentencias **DELETE** es la correcta para eliminar de la tabla **USUARIOS** todos aquellos cuyo código se encuentra en una tabla llamada **ANTIGUOS**?:

DELETE FROM USUARIOS WHERE Codigo IN (SELECT Codigo FROM ANTIGUOS);

DELETE FROM USUARIOS WHERE Codigo IN ANTIGUOS;

DELETE FROM (SELECT Codigo FROM ANTIGUOS) WHERE Codigo IN (SELECT Codigo FROM USUARIOS);

DELETE FROM Codigo WHERE USUARIOS IN (SELECT Codigo FROM ANTIGUOS);

6. Transacciones

Una transacción es una unidad atómica (no se puede dividir) de trabajo que contiene una o más sentencias SQL. Las transacciones agrupan sentencias SQL de tal manera que a todas ellas se le aplica una operación *COMMIT*, que podríamos traducir como aplicadas o guardadas en la base de datos, o bien a todas ellas se les aplica la acción *ROLLBACK*, que interpretamos como deshacer las operaciones que deberían hacer sobre la base de datos.

Mientras que sobre una transacción no se haga *COMMIT*, los resultados de ésta pueden deshacerse. El efecto de una sentencia del lenguaje de manipulación de datos (**DML**) no es permanente hasta que se hace la operación *COMMIT* sobre la transacción en la que esté incluida.

Las transacciones de Oracle cumplen con las propiedades básicas de las transacciones en base de datos:

Atomicidad: Todas las tareas de una transacción son realizadas correctamente, o si no, no se realiza ninguna de ellas. No hay transacciones parciales. Por ejemplo, si una transacción actualiza 100 registros, pero el sistema falla tras realizar 20, entonces la base de datos deshace los cambios realizados a esos 20 registros.

Consistencia: La transacción se inicia partiendo de un estado consistente de los datos y finaliza dejándola también con los datos consistentes.

Aislamiento: El efecto de una transacción no es visible por otras transacciones hasta que finaliza.

Durabilidad: Los cambios efectuados por las transacciones que han **volcado** sus modificaciones, se hacen permanentes. Las sentencias de control de transacciones gestionan los cambios que realizan las sentencias DML y las agrupa en transacciones. Estas sentencias te permiten realizar las siguientes acciones:

Hacer permanentes los cambios producidos por una transacción (**COMMIT**).

Deshacer los cambios de una transacción (**ROLLBACK**) desde que fue iniciada o desde un **punto de restauración** (**ROLLBACK TO SAVEPOINT**). Un punto de restauración es un marcador que puedes establecer dentro del contexto de la transacción. Debes tener en cuenta que la sentencia **ROLLBACK** finaliza la transacción, pero **ROLLBACK TO SAVEPOINT** no la finaliza.

Establecer un punto intermedio (**SAVEPOINT**) a partir del cual se podrá deshacer la transacción.

Indicar propiedades para una transacción (**SET TRANSACTION**).

Especificar si una restricción de integridad aplazable se comprueba después de cada sentencia DML o cuando se ha realizado el **COMMIT** de la transacción (**SET CONSTRAINT**).

6.1. Hacer cambios permanentes

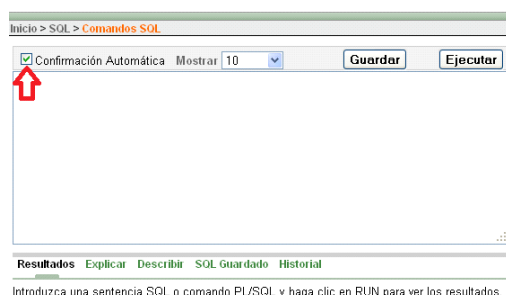
Una transacción comienza cuando se encuentra la primera sentencia SQL ejecutable. Para que los cambios producidos durante la transacción se hagan permanentes (no puedan deshacerse), se dispone de las siguientes opciones:

Utilizar la sentencia **COMMIT**, la cual ordena a la base de datos que haga permanentes las acciones incluidas en la transacción.

Ejecutar una sentencia **DDL** (como **CREATE**, **DROP**, **RENAME**, o **ALTER**). La base de datos ejecuta implícitamente una orden **COMMIT** antes y después de cada sentencia DDL.

Si el usuario cierra adecuadamente las aplicaciones de gestión de las bases de datos Oracle, se produce un volcado permanente de los cambios efectuados por la transacción.

Desde la aplicación gráfica **Application Express**, la ejecución de sentencias SQL desde *Comandos SQL* permite que se hagan los cambios permanentes tras su ejecución, marcando la opción *Confirmación Automática*.



AUTOEVALUACIÓN:

¿Si se cierra correctamente la aplicación gráfica después de haber realizado una operación de modificación de datos, y no se ha indicado la opción de Confirmación automática, ni se ha ejecutado la sentencia **COMMIT**, se quedan guardados los cambios efectuados por la transacción?:

Verdadero.

Falso.

6.2. Deshacer cambios



La sentencia **ROLLBACK** permite deshacer los cambios efectuados por la transacción actual, dándola además por finalizada.

Siempre se recomienda que explícitamente finalices las transacciones en las aplicaciones usando las sentencias **COMMIT** o **ROLLBACK**.

Recuerda que si no se han hecho permanentes los cambios de una transacción, y la aplicación termina incorrectamente, la base de datos de Oracle retorna al estado de la última transacción volcada, deshaciendo los cambios de forma implícita.

Para deshacer los cambios de la transacción simplemente debes indicar:

ROLLBACK;

Hay que tener en cuenta que si una transacción termina de forma anormal, por ejemplo, por un fallo de ejecución, los cambios que hasta el momento hubiera realizado la transacción son deshechos de forma automática.

AUTOEVALUACIÓN:

¿Se pueden deshacer los cambios con la sentencia **ROLLBACK** después de que se haya ejecutado **COMMIT**?:

Verdadero.

Falso.

6.3. Deshacer cambios parcialmente



Un punto de restauración (**SAVEPOINT**) es un marcador intermedio declarado por el usuario en el contexto de una transacción. Los puntos de restauración dividen una transacción grande en pequeñas partes.

Si usas puntos de restauración en una transacción larga, tendrás la opción de deshacer los cambios efectuados por la transacción antes de la sentencia actual en la que se encuentre, pero después del punto de restauración establecido. Así, si se produce un error, no es necesario rehacer todas las sentencias de la transacción completa, sino sólo aquellos posteriores al punto de restauración.

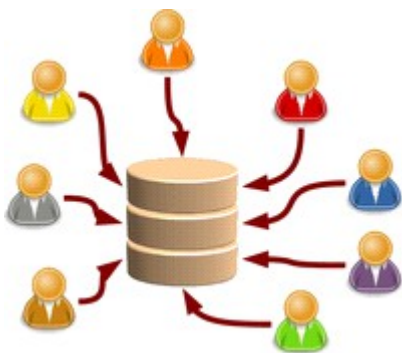
Para establecer un punto de restauración se utiliza la sentencia *SAVEPOINT* con la sintaxis:

SAVEPOINT nombre_punto_restauración;

La restauración de los cambios hasta ese punto se hará con un comando con el siguiente formato:

ROLLBACK TO SAVEPOINT nombre_punto_restauración;

6.4. Problemas asociados al acceso simultáneo a los datos



En una base de datos a la que accede un solo usuario, un dato puede ser modificado sin tener en cuenta que otros usuarios puedan modificar el mismo dato al mismo tiempo. Sin embargo, en una base de datos multiusuario, las sentencias contenidas en varias transacciones simultáneas pueden actualizar los datos simultáneamente. Las transacciones ejecutadas simultáneamente, deben generar resultados consistentes. Por tanto, una base de datos multiusuario debe asegurar:

concurrency de datos: asegura que los usuarios pueden acceder a los datos al mismo tiempo.

consistencia de datos: asegura que cada usuario tiene una vista consistente de los datos, incluyendo los cambios visibles realizados por las transacciones del mismo usuario y las transacciones finalizadas de otros usuarios.

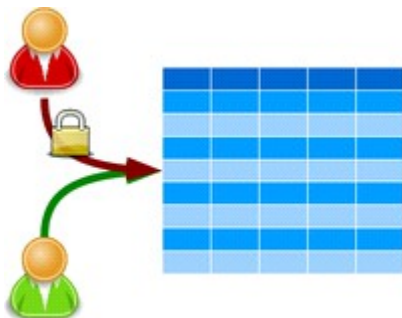
En una base de datos monousuario, no son necesarios los bloqueos ya que sólo modifica la información un solo usuario. Sin embargo, cuando varios usuarios acceden y modifican datos, la base de datos debe proveer un mecanismo para prevenir la modificación concurrente del mismo dato. Los bloqueos permiten obtener los siguientes requerimientos fundamentales en la base de datos:

consistencia: Los datos que están siendo consultados o modificados por un usuario no pueden ser cambiados por otros hasta que el usuario haya finalizado la operación completa.

integridad: Los datos y sus estructuras deben reflejar todos los cambios efectuados sobre ellos en el orden correcto.

La base de datos Oracle proporciona concurrency de datos, consistencia e integridad en las transacciones mediante sus mecanismos de bloqueo. Los bloqueos se realizan de forma automática y no requiere la actuación del usuario.

6.4.1. Políticas de bloqueo



La base de datos permite el uso de diferentes tipos de bloqueos, dependiendo de la operación que realiza el bloqueo.

Los bloqueos afectan a la interacción de lectores y escritores. Un lector es una consulta sobre un recurso, mientras que un escritor es una sentencia que realiza una modificación sobre un recurso. Las siguientes reglas resumen el comportamiento de la base de datos Oracle sobre lectores y escritores:

Un registro es bloqueado sólo cuando es modificado por un escritor: Cuando una sentencia actualiza un registro, la transacción obtiene un bloqueo sólo para ese registro.

Un escritor de un registro bloquea a otro escritor concurrente del mismo registro: Si una transacción está modificando una fila, un bloqueo del registro impide que otra transacción modifique el mismo registro simultáneamente.

Un lector nunca bloquea a un escritor: Puesto que un lector de un registro no lo bloquea, un escritor puede modificar dicho registro. La única excepción es la sentencia *SELECT ... FOR UPDATE*, que es un tipo especial de sentencia *SELECT* que bloquea el registro que está siendo consultado.

Un escritor nunca bloquea a un lector: Cuando un registro está siendo modificado, la base de datos proporciona al lector una vista del registro sin los cambios que se están realizando.

Hay dos mecanismos para el bloqueo de los datos en una base de datos: el bloqueo **pesimista** y bloqueo **optimista**. En el bloqueo pesimista de un registro o una tabla se realiza el bloqueo inmediatamente, en cuanto el bloqueo se solicita, mientras que en un bloqueo optimista el acceso al registro o la tabla sólo está cerrado en el momento en que los cambios realizados a ese registro se actualizan en el disco. Esta última situación sólo es apropiada cuando hay menos posibilidad de que alguien necesite acceder al registro mientras está bloqueado, de lo contrario no podemos estar seguros de que la actualización tenga éxito, porque el intento de actualizar el registro producirá un error si otro usuario actualiza antes el registro. Con el bloqueo pesimista se garantiza que el registro será actualizado.

AUTOEVALUACIÓN:

Supongamos que un usuario está en proceso de modificación de un registro, y otro en ese mismo momento quiere leer ese mismo registro. ¿Qué tipo de bloqueo debes establecer para que el segundo usuario obtenga los datos con los cambios que está efectuando el primero?

Bloqueo pesimista. que se finalice la modificación que ha empezado otro.

Bloqueo optimista.