C.F.G.S. DESARROLLO DE APLICACIONES WEB

DISTANCIA

MÓDULO:

BASES DE DATOS

Unidad 5:

Tratamiento de Datos.

ÍNDICE DE CONTENIDOS

OBJETIVOS

En esta unidad se pretende alcanzar los siguientes objetivos:

- Manejar con fluidez las órdenes para insertar, modificar y eliminar filas de una tabla
- Entender el concepto de integridad de datos
- Entender los conceptos de COMMIT y ROLLBACK
- Utilizar órdenes para crear y suprimir vistas y sinónimos
- Descubrir las ventajas de recurrir a los sinónimos
- Utilizar sinónimos y vistas.

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.

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 incorrecta. 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.

2.- Edición de la información mediante herramientas gráficas.

Los sistemas gestores de bases de datos como el de Oracle, pueden ofrecer mecanismos para la manipulación de la información contenida en las bases de datos. Principalmente se dividen en herramientas gráficas y herramientas en modo texto (también reciben el nombre de terminal, consola o línea de comandos).

Para realizar el tratamiento de los datos por línea de comandos se requiere la utilización de un lenguaje de base de datos como SQL, lo cual implica el conocimiento de dicho lenguaje.

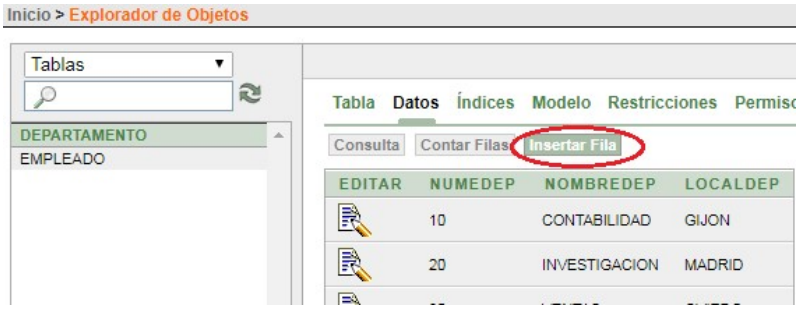
En cambio, si se dispone de herramientas gráficas para la manipulación de los datos, no es imprescindible conocer las sentencias de un lenguaje de ese tipo, y permite la introducción, edición y borrado de datos desde un entorno gráfico con la posibilidad de uso de ratón y una ventana que facilita esas operaciones con un uso similar a las aplicaciones informáticas a las que estamos acostumbrados como usuarios.

2.1.- Inserción de registros.

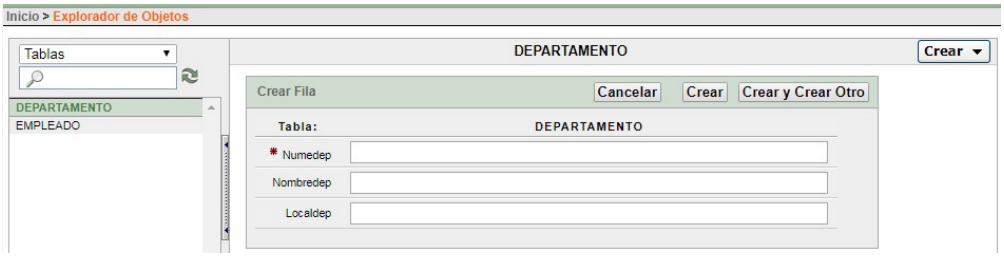
La inserción de registros permite introducir nuevos datos en las tablas que componen la base de datos. Para insertar registros en tablas, utilizando las herramientas gráficas que ofrece Oracle 10g Express Edition, se deben seguir los siguientes pasos:

Una vez conectados a Oracle con el usuario correspondiente:

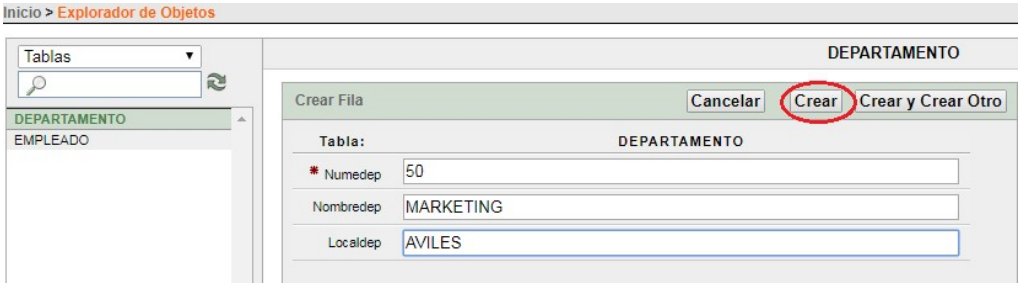
- En la **página inicial**, hacer clic en el botón ***Explorador de objetos***.
- **Seleccionamos una tabla** en la lista izquierda.
- Seleccionamos la pestaña **Datos** y hacer clic en el botón **Insertar Fila.**



Nos aparecerá la pantalla siguiente:

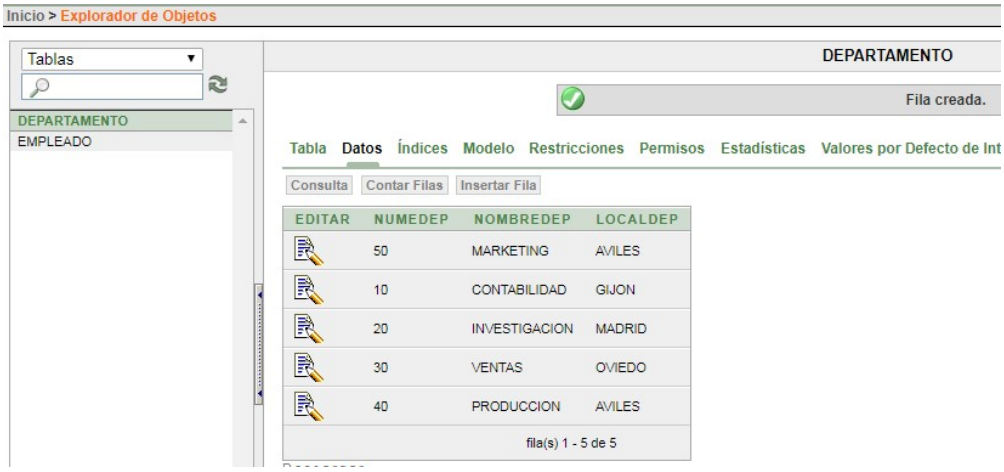


- **Escribimos los datos** correspondientes para cada campo del nuevo registro.



- Hacer clic en el botón **Crear** para guardar los datos introducidos, o **Crear** *y* **Crear Otro** si se desea seguir añadiendo otro registro nuevo. Se utilizará el botón *Cancelar* si no se desea guardar los datos.

- Si la fila se ha **añadido correctamente** se mostrará el mensaje correspondiente y podremos ver los datos insertados en las tablas:



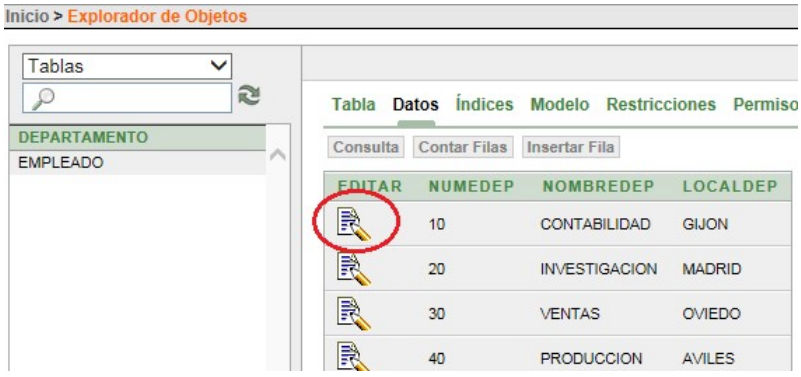
En caso de que se haya producido un **error** al intentar insertar los datos, habrá que comprobar el mensaje que se muestra, e intentar solucionar el problema.

2.2.- Modificación de registros.

Para modificar registros utilizando las herramientas gráficas que ofrece Oracle 10g Express Edition, se deben seguir los siguientes pasos:

Una vez conectados a Oracle con el usuario correspondiente:

- En la **página inicial**, hacer clic en el botón ***Explorador de objetos***.
- **Seleccionamos una tabla** en la lista izquierda.
- Seleccionamos la pestaña **Datos**, nos saldrán los datos que tenemos insertados en la tabla, y vemos que en el lado izquierdo de cada registro aparece el **icono** que permite la modificación de los datos del registro que se encuentra en la misma fila.



- Tras hacer clic en el **icono**, se muestran los campos que forman el registro con los datos que contiene actualmente. Para **modificar cualquier dato** simplemente debemos escribirlo en el campo correspondiente. Así habrá que modificar todos los datos necesarios.

- Para aceptar los cambios realizados, se debe hacer clic en el botón **Aplicar Cambios**, pero si se desea volver al estado anterior, simplemente hay que utilizar el botón **Cancelar.**

- Si todo ha ido bien aparecerá un mensaje informando que los cambios se han aplicado.

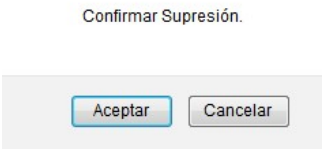
- Los cambios efectuados se pueden **comprobar** en la lista de registros mostrada en la parte inferior de la ventana.
- Al igual que en la inserción de registros, al aceptar los cambios realizados en los datos, éstos se comprobarán automáticamente para ver si cumplen con los requisitos establecidos en la tabla. En caso de que no se cumplan, aparecerá un mensaje informando del error que se ha producido. Una vez solucionado el problema se podrá volver a intentar aplicar los cambios efectuados.

2.3.- Borrado de registros.

En el caso de que quieras eliminar un registro de una determinada tabla hay que seguir, en principio, los **mismos pasos** que se han comentado anteriormente para **editar un registro**. Es decir, una vez seleccionada la tabla, elegir la pestaña Datos y hacer clic en el botón **Editar** junto al registro que quieres suprimir.

Cuando se muestra la información del contenido del registro, puedes observar en la parte superior derecha que dispones de un botón **Suprimir,** junto al que has podido utilizar en el apartado anterior para Aplicar Cambios.

Al hacer clic en ese botón verás una ventana de diálogo donde solicita que confirmes si deseas borrar el registro.



Si todo ha ido bien se mostrará un mensaje informando de ello:

La eliminación de un registro no podrá realizarse si un registro de otra tabla hace referencia a él. En ese caso, se mostrará el mensaje de error correspondiente al intentar eliminarlo. Si ocurriera esto, para eliminar el registro se debe eliminar el registro que hace referencia a él, o bien modificarlo para que haga referencia a otro registro.

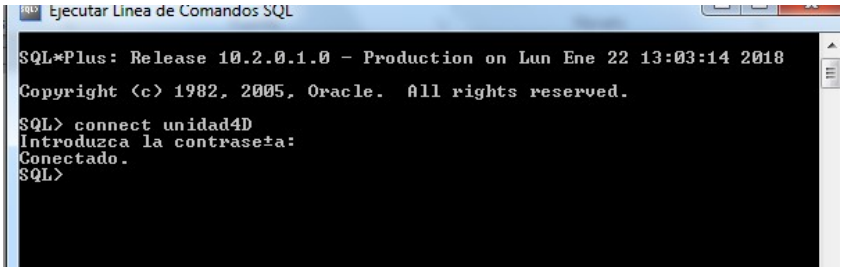
3.- Edición de la información mediante sentencias SQL.

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).**

Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web **Oracle 10g Express Edition**, en la página de **Inicio**, y desplegando su lista desplegable elegir **Comandos SQL → Introducir Comando.**

También se pueden indicar las sentencias SQL desde el entorno de *SQL*Plus* que ofrece Oracle y que puedes encontrar en **Inicio → Todos los programas → Base de Datos Oracle 10g Express Edition →Ejecutar Línea de Comandos SQL.**

Si optas por abrir esa aplicación (*Ejecutar Línea de Comandos SQL*), el primer paso que debe realizarse para manipular los datos de una determinada tabla, es conectarse utilizando un nombre de usuario con los permisos necesarios para hacer ese tipo de operaciones a la tabla deseada. Utiliza para ello la orden **CONNECT** seguida del nombre de usuario. Seguidamente, solicitará la contraseña correspondiente a dicho usuario.



Para ejecutar cualquiera de las sentencias SQL que aprenderás en los siguientes puntos, simplemente debes escribirla completa y pulsar **Intro** para que se inicie su ejecución.

3.1.- Inserción de registros.

La sentencia **INSERT** permite la inserción de nuevas filas o registros en una tabla existente.
El formato de la sentencia **INSERT** tiene la siguiente sintaxis:

```
INSERT INTO nombre_Tabla (expre_colum1, expre_colum2,….) VALUES (valor1, valor2, ……);
```

Dónde:

- **nombre_Tabla:** Será el nombre de la tabla en la que quieras añadir nuevos registros.
- **expre_colum1, expre_colum2,…:** Es la lista de columnas o campos de dicha tabla en los que se desea escribir los nuevos valores.
- **valor1, valor2, ..:** lista de valores a cargar en los campos indicados.

Si no se especifican las columnas, se considera, por defecto, todas las columnas. Hay que introducir los tatos en el orden en el que se encuentran en la tabla.
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.
Los valores deben de coincidir con el tipo de dato definido para cada columna.

Ejemplo:
Vamos a insertar un nuevo registro en la tabla DEPARTAMENTO en el que se tienen todos los datos disponibles:

```
INSERT INTO DEPARTAMENTO (numedep, nombredep, localdep) VALUES (50, 'PUBLICIDAD', 'AVILES');
```

- **Consideraciones a tener en cuenta.**

  Cuando insertamos hay que asegurarse que los datos que se insertan son, al menos, los obligatorios de un registro de esa tabla, es decir:

  - La clave primaria y que su valor no exista ya.
  - Campos definidos en la tabla como no nulos
  - Campos que son clave ajena de otra tabla. En este caso podría no ser obligatorio en función de cómo está definida la restricción de clave ajena: **ON DELETE SET NULL/ON DELETE CASACADE/RESTRICT.**

    En el caso de que esté definida la restricción **ON DELETE CASCADE / RESTRICT** será obligatorio poner un valor que además exista como clave primaria en la tabla a la que hace referencia. → **CONSISTENCIA**

3.2.- Modificación de registros.

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

El formato de la sentencia **UPDATE** tiene la siguiente sintaxis:

```
UPDATE nombre_tabla SET expre_colum1= valor1 [, expre_colum2= valor2].[ WHERE condición ];
```

Dónde:

- **nombre_tabla** será el nombre de la tabla en la que quieras modificar datos.
- Se pueden especificar los nombres de campos o columnas que se deseen de la tabla indicada.
- A cada campo especificado se le debe asociar el nuevo valor utilizando el signo **=**.
- Cada emparejamiento **expre_colum1 = 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.

Ejemplos:

Si en la tabla DEPARTAMENTO queremos modificar el campo LOCALDEP y poner Aviles, pero solo en los registros donde LOCALDEP sea OVIEDO:

```
UPDATE DEPARTAMENTO SET localdep = 'AVILES' WHERE  localdep= 'OVIEDO';
```

Si no ponemos condición:

```
UPDATE DEPARTAMENTO SET localdep = 'AVILES'
```

Cambiaríamos todos los registros de la tabla.

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.

3.3.- Borrado de registros.

La sentencia **DELETE** es la que permite eliminar o borrar registros de una tabla.
El formato de la sentencia **DELETE** tiene la siguiente sintaxis:

```
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.

Ejemplos:

Si quiero borrar los departamentos que están en AVILES, pondría:

```
DELETE FROM DEPATAMENTO WHERE localdep = 'AVILES';
```

Sin embargo si ponemos:

```
DELETE FROM DEPATAMENTO;
```

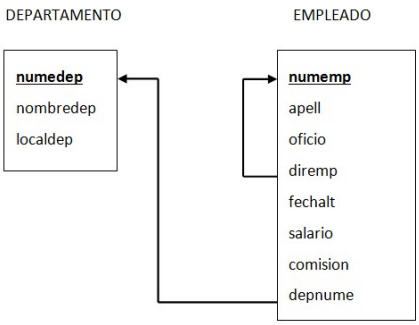
Se borraría el contenido completo de la tabla DEPARTAMENTO.

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.

4.- 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.

En nuestra base de datos de la práctica del tema anterior teníamos la tabla **DEPARTAMENTO** y la tabla **EMPLEADO**.



En la tabla **EMPLEADO** no puede haber empleados que trabajen en un departamento que no esté en la tabla **DEPARTAMENTO**.
Es decir que para que se cumpla la integridad referencial, todos los valores del campo **depnume** deben corresponderse con valores existentes en el campo **numedep**

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 **depnume** de la tabla **EMPLEADO.**
- **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 **numedep** de la tabla **DEPARTAMENTO**
- **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 **EMPLEADO** del ejemplo, que sería la tabla hija de la tabla **DEPARTAMENTO**
- **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 **DEPARTAMENTO** es padre de la tabla **EMPLEADO**.

4.1.- Integridad en actualización y supresión de registros.

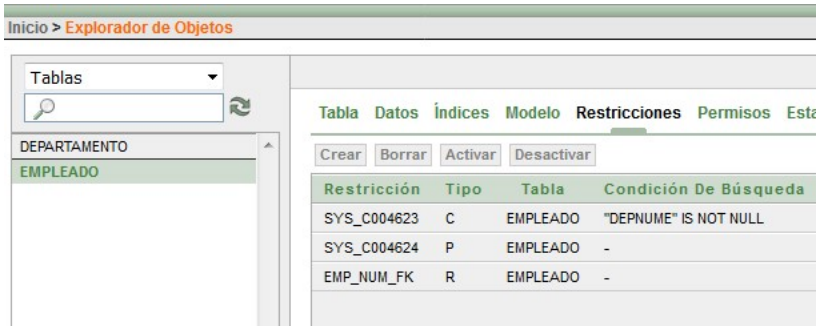
La relación existente entre la clave ajena y la clave padre tiene implicaciones en el borrado y modificación de sus valores.
Si se modifica el valor de la clave ajena en la tabla hija, debe establecerse un nuevo valor que haga referencia a la clave principal de uno de los registros de la tabla padre. 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 pueden suponer un problema, ya que no pueden suprimirse registros que son referenciados con una clave ajena desde otra tabla.
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.**

4.2.- Supresión en cascada.

Las opciones de **Supresión en Cascada o Definir Nulo en Suprimir** pueden establecerse desde el momento de creación de las tablas, al establecer las claves ajenas.

Si la tabla ya estaba creada, y posteriormente se desea establecer una restricción de clave ajena con opción de **Supresión en Cascada,** se puede establecer desde el **Explorador de objetos de Oracle**, seleccionando la tabla que contiene el campo con la clave ajena. La pestaña **Restricciones** ofrece la posibilidad de crear, borrar, activar y desactivar restricciones de este tipo.



Si estas operaciones se quieren realizar con **código SQL**, se dispone de las siguientes opciones durante la declaración de la clave ajena de la tabla: utilizar la opción **ON DELETE CASCADE** para hacer la supresión en cascada, o bien **ON DELETE SET NULL** si se prefiere definir nulo en suprimir.

Hay que recordar que una declaración de este tipo debe hacerse en el momento de crear la tabla (**CREATE TABLE***)* o modificar su estructura (**ALTER TABLE**).

5.- Subconsultas y composiciones en ordenes de edición.

Hemos visto 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.

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**.

Esta misma acción se puede realizar usando una consulta **SELECT** dentro de la sentencia **INSERT**.

El formato tiene la siguiente sintaxis:

```
INSERT INTO TABLA1 [expre_colum1 [,expre_colum2]..)]
SELECT { expre_colum1[,expre_colum2]..| *} FROM TABLA2 [clausulas de la select];
```

Ejemplo:

```
INSERT INTO  EMPLE30
SELECT NUMEMP, apell, oficio, diremp, fechalt, salario, comision, depnume
FROM EMPLEADO WHERE depnume= 30;
```

Previamente hemos tenido que crear la tabla EMPLE30 con la misma estructura que EMPLEADO.

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.

Resumen de la instrucción INSERT

- Si insertamos valores en todos los campos de la tabla (respetar el orden de columnas que tiene la tabla).

```
INSERT INTO   TABLA  VALUES (valor1, valor2,……, valor N)
```

- Si insertamos solo algunos campos de la tabla

```
INSERT INTO   TABLA (col1, col2, ….,col N) VALUES (valor1, valor2,……, valor N)
```

- Si lo hacemos a través de una consulta

```
INSERT INTO   TABLA  (consulta_select ….  que devuelva tantas columnas como tiene la tabla)
```

- Si lo hacemos a través de una consulta con solo algunas columnas

```
INSERT INTO   TABLA  (col1, col2,…)  (consulta_select …. que devuelva tantas columnas como se especifiquen)
```

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**.

El formato tiene la siguiente sintaxis:

```
UPDATE TABLA SET COL1=VALOR1, COL2=VALOR2,...WHERE COL3 = (SELECT ....);

UPDATE TABLA SET (COL1, COL2, COL3,...) = (SELECT COL1, COL2, COL3,...) WHERE CONDICIÓN;
```

5.3.- Borrado 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.

```
DELETE FROM (SELECT .... FROM TABLA WHERE condición);

También

DELETE FROM (SELECT .... FROM TABLA WHEREcondicion1) WHERE condicion2;
```

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 **Oracle 10g Express Edition**, la ejecución de sentencias SQL desde **Inicio → Todos los programas → Base de Datos Oracle 10g Express Edition →Ejecutar Línea de Comandos SQL.** permite que se hagan los cambios permanentes tras su ejecución, marcando la opción **Confirmación Automática**. (**AUTOCOMMIT**).

Podemos saber en qué situación está el parámetro **AUTOCOMMIT** con la instrucción

```
SHOW AUTOCOMMIT;
```

Y si queremos modificarlo con la instrucción

```
SET AUTOCOMMIT ON/OFF; (Valor por omisión OFF)
```

Existen varias órdenes en SQL que fuerzan a que se ejecute un **COMMIT** sin necesidad de indicarlo

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

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.

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:

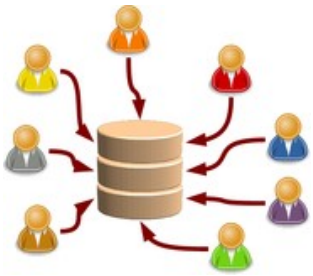
```
SAVEPOINT nombre_punto_restauración;
```

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

```
ROLLBACK TO SAVEPOINT nombre_punto_restauración;
```

7.- Problemas asociados al acceso simultaneo de 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:

- **Concurrencia 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.

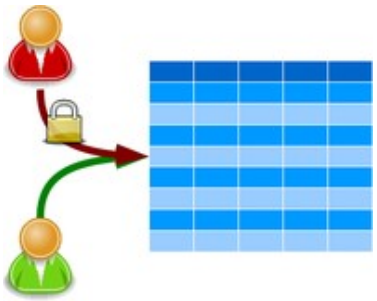
Los bloqueos permiten obtener los siguientes requerimientos fundamentales en la base de datos:

- **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 concurrencia 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.

7.1.- 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.

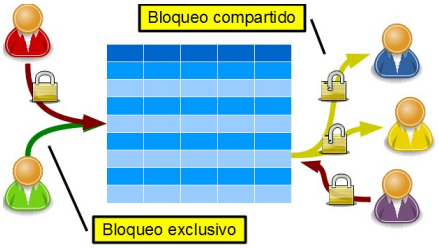
Las siguientes reglas resumen el comportamiento de la base de datos Oracle sobre lectores y escritores:

- 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. Con el bloqueo pesimista se garantiza que el registro será actualizado.
- 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.

7.2.- Bloqueos compartidos y exclusivos.



En general, la base de datos usa dos tipos de bloqueos: bloqueos exclusivos y bloqueos compartidos. Un recurso, por ejemplo un registro de una tabla, sólo puede obtener un bloqueo exclusivo, pero puede conseguir varios bloqueos compartidos:

- **Bloqueo exclusivo**: Este modo previene que sea compartido el recurso asociado. Una transacción obtiene un bloqueo exclusivo cuando modifica los datos. La primera transacción que bloquea un recurso exclusivamente, es la única transacción que puede modificar el recurso hasta que el bloqueo exclusivo es liberado.
- **Bloqueo compartido**: Este modo permite que sea compartido el recurso asociado, dependiendo de la operación en la que se encuentra involucrado. Varios usuarios que estén leyendo datos pueden compartir los datos, realizando bloqueos compartidos para prevenir el acceso concurrente de un escritor que necesita un bloqueo exclusivo. Varias transacciones pueden obtener bloqueos compartidos del mismo recurso.

Por ejemplo, supongamos que transacción usa la sentencia **SELECT … FOR UPDATE** para consultar un registro de una tabla. La transacción obtiene un bloqueo exclusivo del registro y un bloqueo compartido de la tabla. El bloqueo del registro permite a otras sesiones que modifiquen cualquier otro registro que no sea el registro bloqueado, mientras que el bloqueo de la tabla previene que otras sesiones modifiquen la estructura de la tabla. De esta manera, la base de datos permite la ejecución de todas las sentencias que sean posibles.

7.3.- Bloqueos automáticos.

La base de datos Oracle bloquea automáticamente un recurso usado por una transacción para prevenir que otras transacciones realicen alguna acción que requiera acceso exclusivo sobre el mismo recurso. La base de datos adquiere automáticamente diferentes tipo de bloqueos con diferentes niveles de restricción dependiendo del recurso y de la operación que se realice.
Los bloqueos que realiza la base de datos Oracle están divididos en las siguientes categorías:

- **Bloqueos DML**: Protegen los datos, garantizando la integridad de los datos accedidos de forma concurrente por varios usuarios. Por ejemplo, evitan que dos clientes compren el último artículo disponible en una tienda online. Estos bloqueos pueden ser sobre un sólo registro o sobre la tabla completa.
- **Bloqueos DDL**: Protegen la definición del esquema de un objeto mientras una operación DDL actúa sobre él. Los bloqueos se realizan de manera automática por cualquier transacción DDL que lo requiera. Los usuarios no pueden solicitar explícitamente un bloqueo DDL.
- **Bloqueos del sistema**: La base de datos Oracle usa varios tipos de bloqueos del sistema para proteger la base de datos interna y las estructuras de memoria.

7.4.- Bloqueos manuales.

Hemos visto que la base de datos Oracle realiza bloqueos de forma automática para asegurar la concurrencia de datos, su integridad y consistencia en la consulta de datos. Sin embargo, también puedes omitir los mecanismos de bloqueo que realiza por defecto la base de datos Oracle. Esto puede ser útil en situaciones como las siguientes:

- En aplicaciones que requieren consistencia en la consulta de datos a nivel de transacciones o en lecturas repetitivas: En este caso, las consultas obtienen datos consistentes en la duración de la transacción, sin reflejar los cambios realizados por otras transacciones.
- En aplicaciones que requieren que una transacción tenga acceso exclusivo a un recurso con el fin de que no tenga que esperar que otras transacciones finalicen.

La cancelación de los bloqueos automáticos se pueden realizar a nivel de sesión o de transacción. En el nivel de sesión, una sesión puede establecer el nivel requerido de aislamiento de la transacción con la sentencia **ALTER SESSION**.
En el nivel de transacción, las transacciones que incluyan las siguientes sentencias SQL omiten el bloqueo por defecto:

- **SET TRANSACTION ISOLATION LEVEL**
- **LOCK TABLE**
- **SELECT...FOR UPDATE**

Los bloqueos que establecen las sentencias anteriores terminan una vez que la transacción ha finalizado.