

Caso práctico

Ada le ha preguntado a Juan sobre el estado actual del proyecto y él le comenta que está empezando el desarrollo de la aplicación y va a empezar a desarrollar una serie de procesos en los que se deberá almacenar la información que debe manejar la aplicación, así como modificarla o eliminar los datos que así lo requieran.

Estas acciones de tratamiento de la información deberán asegurar que no se obtengan resultados incorrectos, por errores en la ejecución de la aplicación o por las acciones de los usuarios, y además debe asegurar que los datos puedan ser accesibles por varios usuarios simultáneamente.



[Ministerio de Educación](#) (Uso educativo no)

La aplicación requiere que se puedan dar de alta nuevos usuarios en la base de datos, así como juegos y partidas. Además se podrá modificar en un determinado momento la información personal de los usuarios, de los juegos, o añadir nuevos usuarios a las partidas. También asegurará la posibilidad de suprimir cualquiera de esos datos.

Se debe asegurar que, por ejemplo, una partida no haga referencia a usuario que han sido eliminado, o a juegos que no existen. Un usuario podrá ver reducido su crédito en un determinado momento, y la nueva información de su crédito sólo deberá ser accesible cuando haya finalizado el proceso de reducción del crédito, y no mientras se realiza esa actualización, ya que el crédito disponible no estará actualizado.

Por supuesto, al ser una aplicación online, distintos usuarios podrán realizar operaciones simultáneamente, como crear partidas al mismo tiempo.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Introducción.

Caso práctico

Juan le pregunta a Ana, la alumna que se encuentra en prácticas, qué mecanismos conoce para poder manipular los datos que deben encontrarse en una base de datos, de manera que se puedan añadir nuevos datos, modificarlos o eliminarlos. Ella recuerda que estudió una serie de sentencias o comandos del lenguaje SQL que permiten realizar todas esas operaciones, y que además, desde el entorno visual de la base de datos Oracle también se pueden realizar todas esas acciones de manera más cómoda para el usuario, pero menos flexible.



[Ministerio de Educación](#) (Uso educativo nc)

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.



[Victor C.](#) (GNU/GPL)

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.

Caso práctico

Ana ha recibido el encargo de que introduzca una serie de datos en las tablas de la base de datos para poder realizar varias pruebas de su funcionamiento. No son muchos registros los que tiene que introducir, así que va a utilizar una herramienta gráfica que le ofrece el sistema gestor de base de datos que van a utilizar. Ella podría hacerlo escribiendo una serie de instrucciones que ha aprendido durante sus estudios, pero como no son muchos los registros que debe introducir, ha optado por utilizar la herramienta gráfica, ya que facilita el tratamiento de los datos para casos sencillos.



[Ministerio de Educación](#) (Uso educativo no)

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).



[Everaldo Coelho](#) (GNU/GPL)

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.

La base de datos Oracle ofrece en sus distribuciones Oracle Database Express la herramienta Application Express que se descarga al instalar Oracle DB XE, excepto en la versión 18c que hay que instalarla a parte. Si tienes instalada una versión anterior a las 18c XE podrás acceder en Windows desde Inicio > Todos los programas > Base de Datos Oracle Express Edition > Ir a Página Inicial de Base de Datos.

Nosotros seguiremos trabajando con **SQLDeveloper**, la herramienta de Oracle más extendida, cuyo objetivo es proporcionar una interfaz más amigable para la consulta y programación de la base de datos Oracle. La funcionalidad disponible en SQLDeveloper es sólo parte de la disponible a través de comandos en SQL*Plus, pero se corresponde con las tareas más habituales de interacción, programación y depuración de código sobre la base de datos. Descárgate el manual que encontrarás en el apartado *Debes conocer* para aprender cómo realizar las operaciones más básicas.

En el SGBD Mysql podemos encontrar una herramienta similar, aunque menos potente: phpmyadmin.

Debes conocer

En el siguiente enlace puedes acceder a un PDF de un manual breve y sencillo del uso básico de SQLDeveloper en español. Te será útil para el seguimiento de la unidad.

[Manual básico SQLDeveloper](#) (pdf - 1024,83 KB)

Para obtener más información sobre SQLDeveloper puedes acceder a la web oficial. Agrégalo a marcadores para tenerlo a mano.

[Acceso a manual en la web oficial de SQLDeveloper](#)

Autoevaluación

La única manera de realizar el tratamiento de datos en una base de datos es a través de una herramienta gráfica. ¿Verdadero o Falso?

- ☐ Falso.
- ☐ Verdadero.

Efectivamente, se pueden encontrar otras herramientas en modo texto en las que la manipulación de los datos se hace a través de una serie de comandos.

Incorrecto, vuelve a intentarlo, porque la respuesta que has dado no es correcta.

Solución

1. Opción correcta
2. Incorrecto

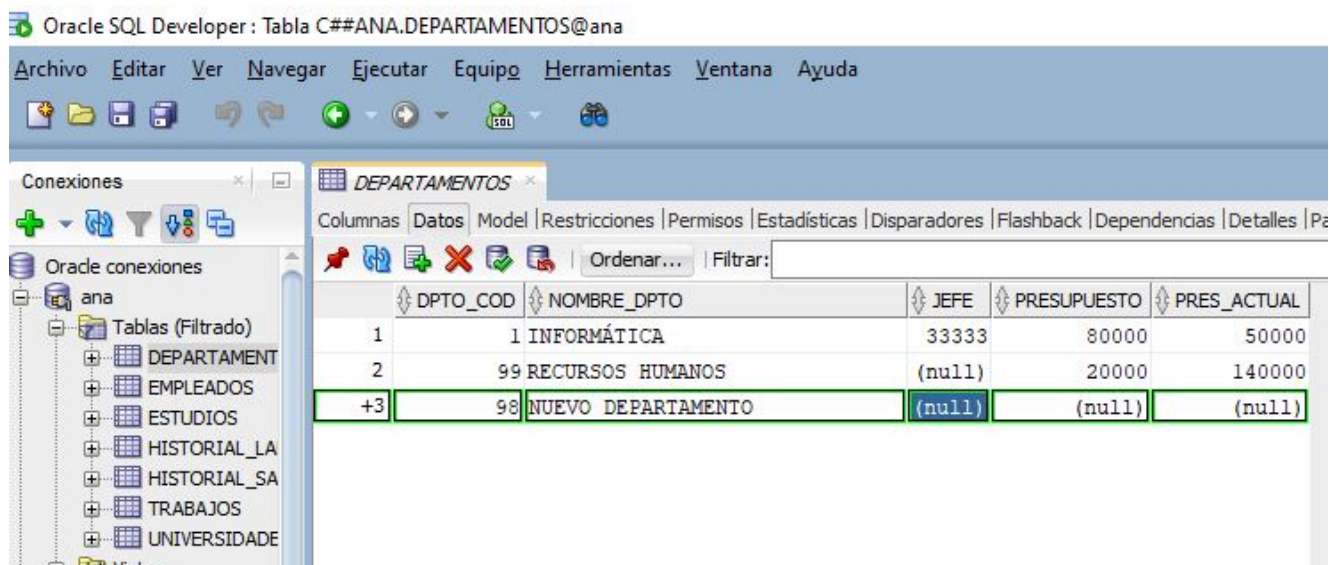
2.1.- Inserción de registros.

La inserción de registros o filas permite introducir nuevos datos en las tablas que componen la base de datos. En el [Manual básico SQLDeveloper](#), de la unidad anterior, tienes explicados los pasos para insertar filas utilizando el interface gráfico. Tan sencillo como seleccionar la tabla,

pulsar la pestaña Datos y el botón de Insertar



Ya puedes introducir datos.



[Oracle Corporation](#) (Todos los derechos reservados)

Para finalizar, cuando hayas insertado todas las filas, confirma la operación con el botón correspondiente

Confirmación Correcta si las filas se han insertado de forma correcta.

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. Por ejemplo, si se intenta introducir un texto en un campo de tipo numérico se obtendrá un error como el siguiente: "error ORA-00984: columna no permitida aqui", y no se habrá realizado ninguna operación de la inserción del nuevo registro.

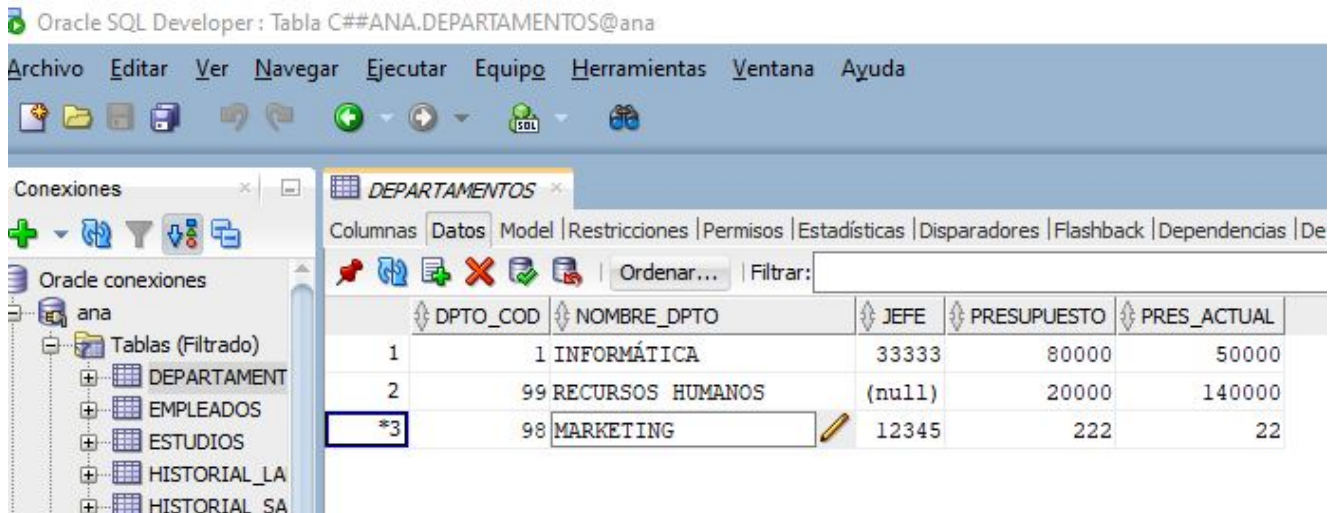
Debes conocer

Para deshacer una operación de inserción, modificación o borrado de filas que aún no ha sido confirmada utiliza el botón de Rollback


podrás deshacerlos. Tampoco podrás deshacerlos si has realizado una operación DDL, ya que éstas confirman de forma automática.

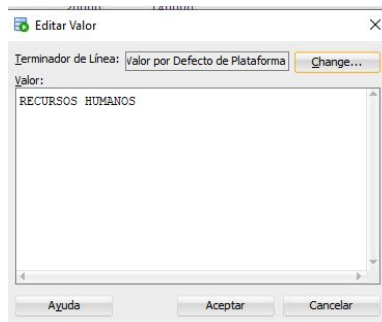
2.2.- Modificación de registros.

Para modificar los datos de alguna fila, selecciona la tabla, pulsa la pestaña Datos y sitúate directamente en la columna de la fila que quieras modificar, haz clic con el ratón y escribe el nuevo contenido.




[Oracle Corporation](#) (Todos los derechos reservados)

Si cambias a otra fila el contenido modificado se mantiene en pantalla. Otra forma de hacerlo es pulsar el botón  que aparece a continuación de la columna: se abrirá una nueva ventana para que introduzcas el nuevo valor.



[Oracle Corporation](#) (Todos los derechos reservados)


Para finalizar, cuando hayas realizado todas las modificaciones, confirma la operación con el botón correspondiente

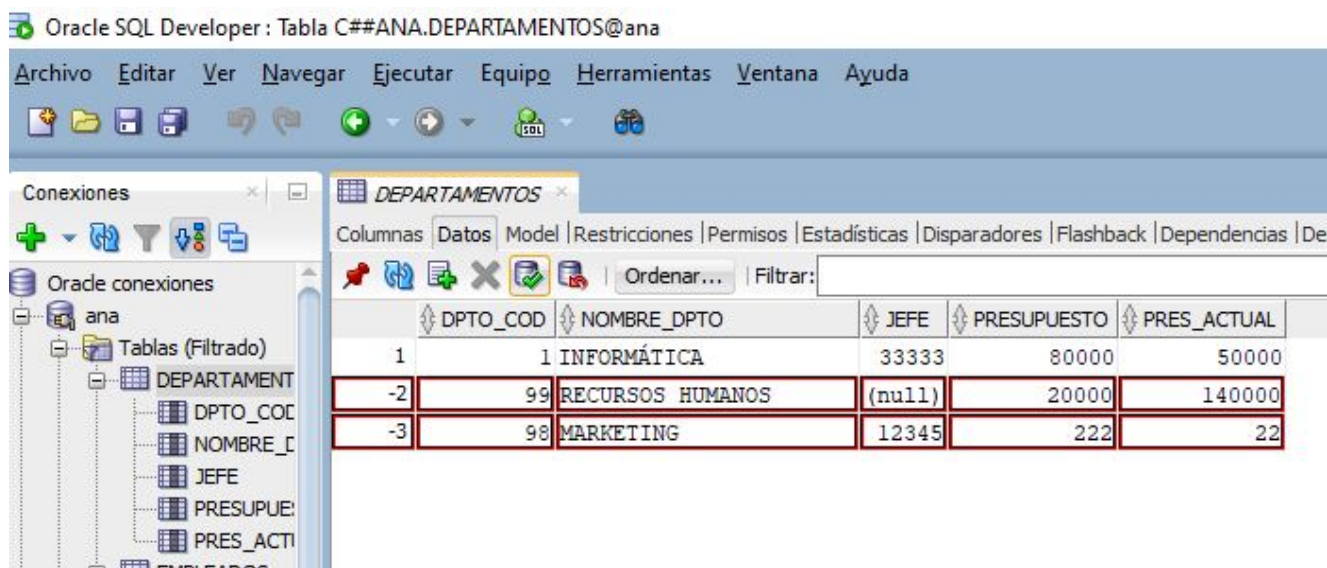
 Botón de confirmación de transacción en SQLDeveloper. Se muestra el símbolo de verificación verde sobre la representación del símbolo de la BD

. Se visualizará un mensaje en la parte inferior de la pantalla indicando *Confirmación Correcta* si las filas se han modificado de forma correcta.



Al igual que se comentó 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 confirmando la operación.

2.3.- Borrado de registros.

En el caso de que quieras eliminar un registro o fila de una determinada tabla elige la pestaña Datos y sitúate en la fila que quieras eliminar (da igual que te sitúes en el número de la fila o en cualquier columna, el resultado será el mismo). Pulsa el botón  para cada una de las filas que quieras eliminar. Al hacerlo marcará el borde la fila en rojo y colocará el signo - delante del número de cada fila. En la imagen inferior se han borrado las filas 2 y 3, aunque aún necesitaremos confirmar la operación para que se apliquen los cambios.



[Oracle Corporation](#) (Todos los derechos reservados)

Para que los cambios se apliquen y se realice el borrado efectivo de las filas, al igual que en las operaciones de inserción y modificación que debes confirmar con el botón . Recuerda que puedes deshacer los borrados, antes de confirmar, con el botón .

En cualquiera de los dos casos se visualizará en la parte inferior de la pantalla y el proceso se ha realizado de forma correcta.

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 correspondiente al intentar eliminarlo (similar a: "error ORA-02292: restricción de integridad violada - registro secundario encontrado"). 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.

Caso práctico


La aplicación web de juegos online que está desarrollando Juan, debe acceder a la base de datos desde su código fuente para recoger datos almacenados en ella. La herramienta gráfica que ha estado utilizando Ana para introducir datos no puede usarse para que la aplicación que está desarrollando realice esa misma operación. Tendrá que utilizar, desde el código fuente de la aplicación, sentencias del lenguaje SQL que permiten la manipulación de los datos. Por ello, va a practicar con Ana el uso de esas sentencias SQL desde la aplicación SQLDeveloper, para más adelante implementarlas en el código fuente de la aplicación.




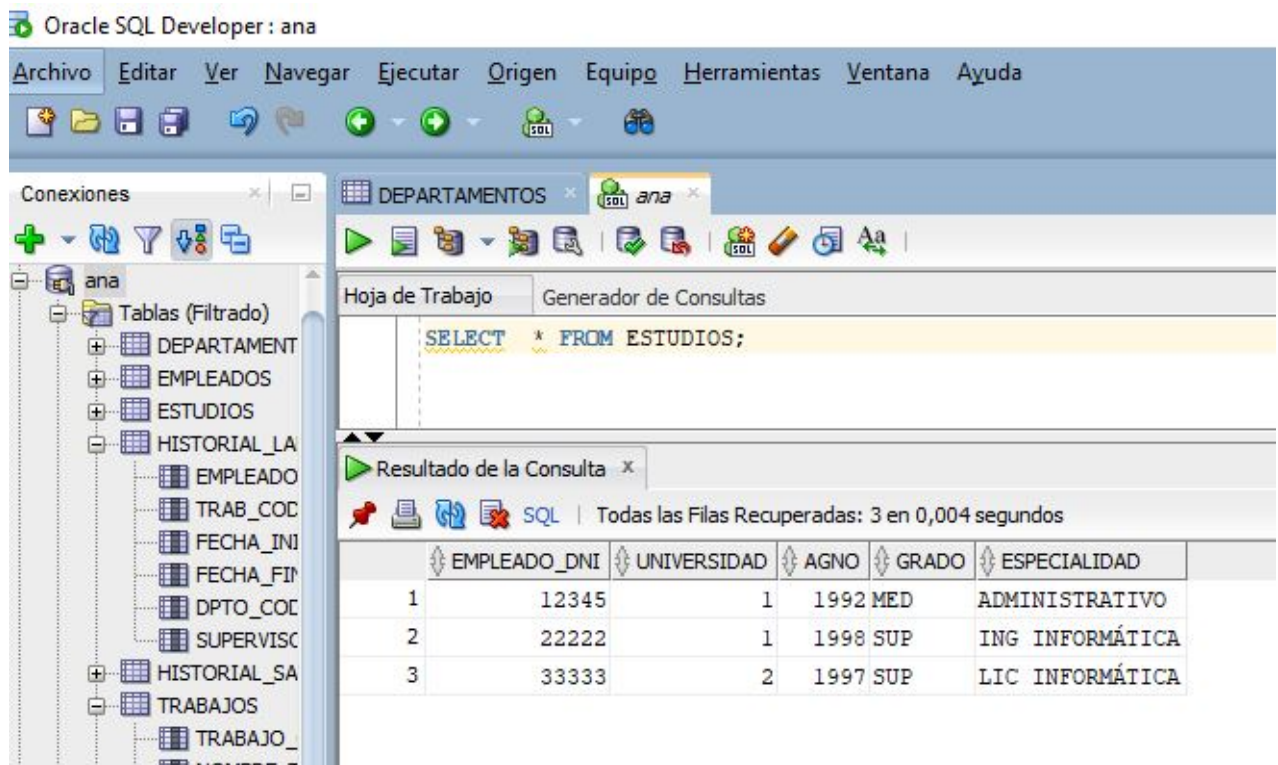
[Ministerio de Educación](#) (Uso educativo no)

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).

Como ya sabemos las sentencias SQL pueden ser ejecutadas desde la línea de comandos de SQLPlus o escribiéndolas en la Hoja de trabajo de SQLDeveloper.


Para trabajar con SQLDeveloper utiliza el botón  . Solicitará la conexión con la que queremos trabajar pudiendo seleccionarla desde la lista desplegable asociada. Una vez elegida se abrirá en la pantalla la Hoja de trabajo desde donde podremos introducir las sentencias SQL.

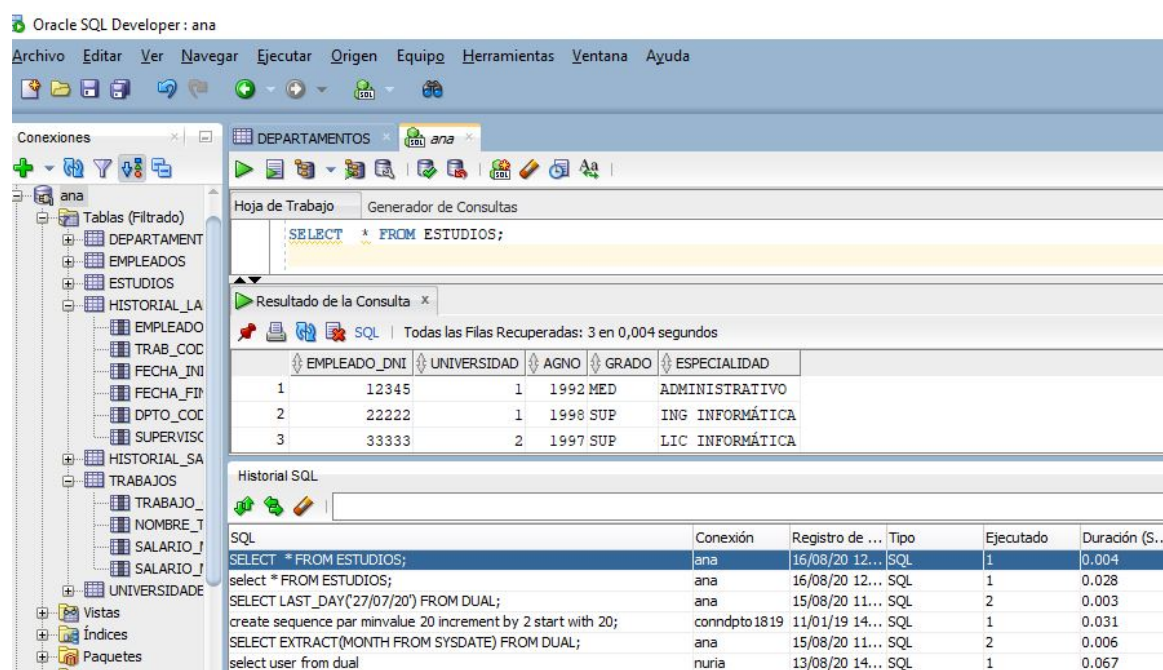
Para ejecutar la sentencias SQL activa utiliza el botón  situado en la parte superior de la pantalla. Si quieres ejecutar otras sentencias SQL, que se encuentren en la Hoja de trabajo, selecciónalas antes pulsar el botón de ejecución.



[Oracle Corporation](#) (Todos los derechos reservados)

El resultado de la operación, consulta en caso de SELECT, o estado, en caso de otras operaciones, se mostrará en la parte inferior de la pantalla.

Entre las muchas opciones interesantes de SQLDeveloper, tienes la posibilidad de acceder a todas las sentencias SQL que has escrito, aunque no las hayas guardado, con el botón  situado en la parte superior.



[Oracle Corporation](#) (Todos los derechos reservados)

3.1.- Inserción de registros.

La sentencia **INSERT** permite la inserción de nuevas filas o registros en un 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).

Para poder probar los ejemplos debes tener creadas y cargadas las tablas de **JuegosOnline** en el usuario `c##juegos` o similar. Si no lo has hecho en la unidad anterior, descárgate el script de este [enlace](#), conecta con `sys as sysdba` y a continuación ejecútalo. Recuerda que si lo haces desde `sqlplus` solo tienes que escribir la ruta y el nombre del script precedido del símbolo `@` o bien de la palabra `start`.

Antes de ejecutar el siguiente ejemplo que inserta un nuevo registro en la tabla **USUARIOS** en el que se tienen todos los datos disponibles debes ejecutar la sentencia

```
ALTER SESSION SET NLS_DATE_FORMAT='DD/MM/YYYY';
```

para que tome la fecha en ese formato en el que le estamos dando el dato fecha.

```
INSERT INTO USUARIOS (Login, Password, Nombre, Apellidos, Direccion, CP, Localidad, Provincia, Pa:
F_Ing, Correo, Credito, Sexo) VALUES ('migrod86', '6PX5=V', 'MIGUEL ANGEL', 'RODRIGUEZ RODRIGUEZ',
'47001', 'VALLADOLID', 'VALLADOLID', 'ESPAÑA', '27/04/1977', '10/01/2008', 'migrod86@gmail.com', ;
```



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, direccion,cp,localidad,provincia,pais,C
'VBROMI', 'NATALIA', 'SANCHEZ GARCIA','C/Blanca','28003','Madrid','Madrid','Spain', 'natsan63@hotr
```



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, o si no se proporcionan valores para campos que no admiten el valor **NULL**, 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',
```

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);

Incorrecto, el nombre del producto es un texto y debe ir entre comillas simples.

No es correcto, aunque se puede omitir la lista de campos, hay que indicar VALUES.

Correcto, esta sentencia se puede ejecutar correctamente.

No es cierto, el orden de los valores no coincide con el de los nombre de los campos.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

4. Incorrecto

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.

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 poniendo a Nulos la información del campo *f_nac* de todos los usuarios:

```
UPDATE USUARIOS SET Credito = 0, f_nac = 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:

```
9 fila(s) actualizada(s).
```

También podemos escribir la sentencia en la hoja de trabajo de SQLDeveloper. Por ejemplo incrementar en 10 el credito de las mujeres



Conexiones



Oracle conexiones

ana
juegos
 Tablas (Filtrado)
 JUEGOS
 PARTIDAS
 UNEN
 USUARIOS
 LOGIN
 PASSWORD
 NOMBRE
 APELLIDOS

Página de bienvenida juegos juegos~1



Hoja de Trabajo

Generador de Consultas

```
update usuarios set credito=credito+10 where sexo='M';
```

Salida de Script x

Tarea terminada en 36,991 segundos

9 filas actualizadas.

3.3.- Borrado de registros.

La sentencia **DELETE** es la que permite eliminar o borrar registros de una 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;
```

Es tan importante escribir la cláusula **WHERE** en la sentencia, si no quieres borrar la tabla entera, que incluso hay una canción que lo recuerda.. Puedes verla en este [enlace](#).

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.

Autoevaluación

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

- ☐ Verdadero.
- ☐ Falso.

Correcto, debes tener cuidado al usar la sentencia **DELETE**, porque si no se especifica qué datos se desea eliminar de la tabla, se eliminará todo su contenido.

Vuelve a intentarlo, porque la respuesta que has dado no es correcta.

Solución

1. Opción correcta
2. Incorrecto

4.- Integridad referencial.

Caso práctico

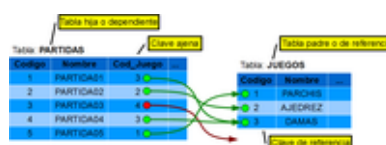
Ana tiene una duda en el planteamiento de la base de datos del proyecto de la plataforma de juegos online y se la plantea a Juan: ¿Qué ocurre si el registro correspondiente a los datos de un determinado juego es eliminado y existen partidas creadas de dicho juego? Juan le responde que para eso existe la integridad referencial, que además asegurará otras cosas como por ejemplo que no puedan existir partidas que reflejen que su creador es un usuario que no existe en la base de datos.



[Ministerio de Educación](#) (Uso educativo nc)

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**.



[Ministerio de Educación y FP](#) (Uso educativo nc)

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** de la tabla **PARTIDAS** 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:** También llamada clave foránea, 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**.

Para saber más

Descripción del concepto de integridad referencial con ejemplo.

[Integridad Referencial.](#)

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, y 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:



[Ministerio de Educación y FP](#) (Uso educativo nc)

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 o modificación 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 ni modificación:** Es la opción por defecto. En caso de que se intente borrar o modificar 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 o modificación en Cascada (ON DELETE CASCADE):** Al suprimir o modificar registros de la tabla de referencia, los registros de la tabla hija que hacían referencia a dichos registros, también son borrados o modificados.
- ✓ **Asignación de Nulo (ON DELETE SET NULL):** Los valores de la clave ajena que hacían referencia a los registros que hayan sido borrados o modificados de la tabla de referencia, son cambiados al valor **NULL**.
- ✓ **Valor por defecto (ON DELETE DEFAULT):** Los valores de la clave ajena que hacían referencia a los registros que hayan sido borrados o modificados de la tabla de referencia, son cambiados al valor especificado por defecto.

En el caso de las modificaciones se cambiaría **ON DELETE** por **ON UPDATE**

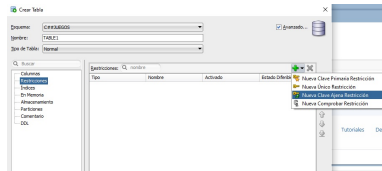
Para saber más

Apuntes sobre integridad referencial en Oracle.

[Las Restricciones de Integridad en ORACLE.](#)

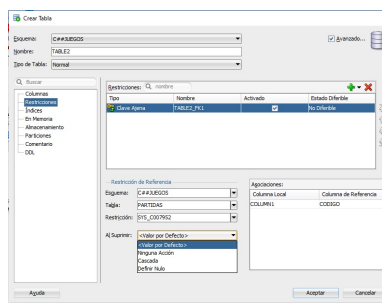
4.2.- Supresión en cascada.

Las opciones de *Supresión en Cascada* o *Definir Nulo en Suprimir* pueden establecerse desde el momento de creación de las tablas. Desde SQLDeveloper marcamos la casilla Avanzado, nos situamos en la columna que es clave ajena y seleccionamos en el árbol Restricciones. En la lista desplegable del botón Añadir seleccionamos *Nueva Clave Ajena Restricción* como se muestra en la imagen.



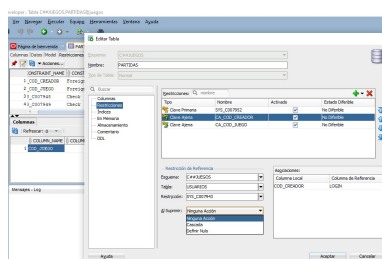
[Oracle Corporation](#) (Todos los derechos reservados)

Y podremos elegir la opción en la lista desplegable del cuadro "Al suprimir" como se muestra en la siguiente imagen.



[Oracle Corporation](#) (Todos los derechos reservados)

Si la tabla ya estaba creada, y posteriormente se desea establecer una restricción de clave ajena con una opción concreta se puede establecer desde la opción Editar tabla siguiendo los pasos similares a la definición en la creación de la nueva tabla, como se muestra en la imagen.



[Oracle Corporation](#) (Todos los derechos reservados)

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. Por ejemplo:

```
CONSTRAINT JUEGOS_CON FOREIGN KEY (Cod_Juego) REFERENCES JUEGO (Codigo) ON DELETE CASCADE;
```

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 órdenes de edición.

Caso práctico

Juan necesita un mecanismo que permita actualizaciones en la base de datos de forma masiva con una serie de condiciones que afectan no sólo a la tabla sobre la que debe hacer los cambios en los datos. Por ejemplo, va a implementar, en la aplicación que está desarrollando, una opción para que los usuarios que han creado partidas obtengan algunos beneficios en los créditos que disponen.



[Ministerio de Educación](#) (Uso Educativo nc)

Para conseguir esto no le sirven las sentencias SQL simples que has podido ver en apartados anteriores. Deberá utilizarlas en unión con consultas que determinen los registros que han de ser modificados.

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 con una sola 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 una tabla se puede ver afectada por los resultados de las operaciones en otras tablas, es decir, que con una misma instrucción se puede añadir más de un registro a una tabla, o bien actualizar o eliminar varios registros basados en otras consultas

Los valores que se añadan o se modifiquen podrán ser obtenidos 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

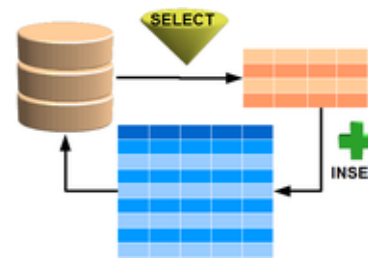
En este manual pueden encontrar una sección sobre las funciones agregadas y subconsultas (módulo 3). También puedes ver ejemplos en la parte final.

[Resumen de SQL con ejemplos, incluyendo material sobre subconsultas.](#)

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, direccion, cp, localidad, provincia, pais, Correo) VALUES ('VBROMI', 'NATALIA', 'SANCHEZ GARCIA', 'C/Blanca', '28003', 'Madrid', 'Madrid', 'Spain', 'natsan63@hotmail.com');
```



[Ministerio de Educación](#). (Uso Educativo no)

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 USUARIOS (SELECT Login, Password, Nombre, Apellidos, direccion, cp, localidad, provincia, pais, Correo FROM USUARIOS SIN CREDITO WHERE Credito = 0);
```

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

También 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** ya creada. 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.

Se puede **crear una tabla e insertar datos a partir de una consulta**. Podemos crear la tabla **USUARIOS_CON_CREDITO** partiendo de la tabla usuario, si queremos crearla exactamente con el mismo número de campos y con su contenido pondremos:

```
CREATE TABLE USUARIOS_CON_CREDITO AS SELECT * FROM USUARIOS WHERE CREDITO > 0 ;
```

Si queremos crear una tabla **USUARIAS** con la misma estructura, pero sin contenido especificaremos una condición que no se cumpla nunca y así ningún registro se copiará:

```
CREATE TABLE USUARIAS AS SELECT * FROM USUARIOS WHERE 1 < 0;
```

A partir de aquí podremos insertar en esta nueva tabla:

```
INSERT INTO USUARIAS  
SELECT * FROM USUARIOS WHERE UPPER(SEXO)='M';
```

Observa que en ese caso no es necesario incorporar en la instrucción la palabra **VALUES** ya que para cada fila insertada incorporamos los valores correspondientes a todas las columnas de la tabla de destino.

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) VALUES (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);`

Incorrecto. Falta la sentencia **SELECT**.

¡Muy bien!

No es correcto. El campo **Nombre_CLI** pertenece a la tabla **CLIENTES**.

No es cierto. El orden de las tablas es el contrario.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

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.



[Ministerio de Educación](#) (Uso educativo nc)

```
UPDATE USUARIOS SET Credito = (SELECT MAX(Credito) FROM
USUARIOS) WHERE Login IN (SELECT Cod_Creador 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 = 3) WHERE Codigo = 5;`

Incorrecto. El formato no es correcto.

No es cierto. La condición WHERE no está bien formada.

No es correcto. No se debe asignar el valor 5 al código.

¡Muy bien!

Solución

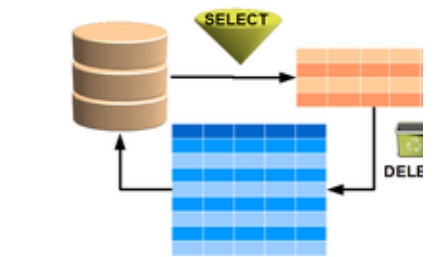
1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

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 LOGIN FROM USUARIOS, UNEN WHERE CODIGO_USUA
```



[Ministerio de Educación.](#) (Uso Educativo no)

El resultado es que se eliminan determinados registros de las tablas **USUARIOS** y **UNEN**, en concreto, aquellos registros de la tabla **UNEN** asociados a algún usuario de **PALENCIA**.

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 LOGIN, PROVINCIA FROM USUARIOS, UNEN WHERE CODIGO_USUARIO=LOGIN) WHERE PROVINCIA
```

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);`

¡Muy bien!

Incorrecto. Falta la subconsulta.

No es correcto. Las consultas deben ser las contrarias.

No es cierto. Detrás de **FROM** se debe indicar el nombre de la tabla.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

6.- Transacciones.

Caso práctico

Ana ha estado haciendo algunas pruebas del funcionamiento de la aplicación y ha observado un error: Con los créditos que dispone un determinado usuario ha empezado la creación de una nueva partida, pero antes de finalizar el proceso de creación de la partida ha utilizado un botón "Cancelar" para simular que el usuario ha optado por dar marcha atrás en la creación de la partida. En ese caso, el crédito del usuario debería permanecer inalterado, ya que no ha finalizado el proceso de creación de la partida, pero ha observado los datos que hay en la base de datos y se encuentra con que el crédito del usuario se ha decrementado.



[Ministerio de Educación](#). (Uso Educativo no)

Al comentarle el problema a Juan, éste le comenta que debe gestionar ese proceso utilizando 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 confirmadas, 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.

Un ejemplo de transacción sería el proceso de transferencia entre cuentas bancarias. Supongamos que Juan hace una transferencia de 100 euros de su cuenta a la de María. Básicamente, las operaciones a realizar en las tablas serías:

1. Actualizar la cuenta de Juan restándole al saldo 100 euros
2. Registrar el movimiento de decremento de 100 euros en los movimientos de la cuenta de Juan
3. Actualizar la cuenta de María incrementándola con 100 euros
4. Registrar el movimiento de incremento de 100 euros en los movimientos de la cuenta de María.

¿ Qué pasaría si hubiese un corte o caída en el sistema después de efectuarse el punto 2?. ¿ Dónde estarían esos 100 euros? Juan ya no los tiene, pero María tampoco.

Para evitar situaciones de estas se utilizan las transacciones, de forma que las 4 operaciones sean consideradas como una única operación y, o se realizan las 4, o no se realiza ninguna, es decir, hasta que no estén las 4 realizadas, no se confirma (**COMMIT**) la operación. Si hay problemas antes de que se realicen las 4, se deshacen (**ROLLBACK**) las operaciones hechas que han dejado a medias el proceso.

Como ves, esta característica da robustez y seguridad a un SGBD asegurando la consistencia de los datos. El SGBD Oracle es el más potente a nivel transaccional.

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 o hasta que se ejecuta una operación DDL.

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:

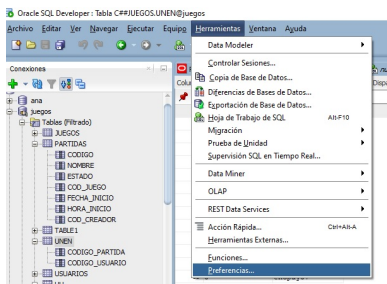
- ✓ Hacer permanentes los cambios producidos por una transacción (**COMMIT**).
- ✓ Deshacer los cambios de una transacción (**ROLLBACK**) desde que fue iniciada o desde un punto de restauración (**ROLLBACK TO SAVEPOINT**). Un punto de restauración es un marcador que puedes establecer dentro del contexto 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**). Por ejemplo podemos elegir **READ ONLY** y no se permitirán modificaciones.
- ✓ 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**) podemos elegir **IMMEDIATE** o **DEFERRED**, es decir, se comprueba la transacción inmediatamente después de cada orden DML o cuando haya sido confirmada

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:

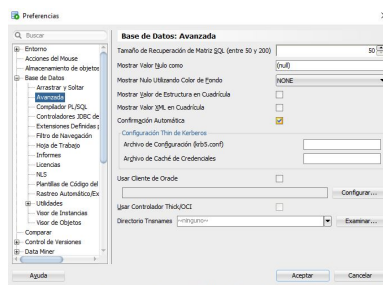
- ✓ Utilizar la sentencia **COMMIT**, la cual ordena a la base de datos que haga permanentes las acciones incluidas en la transacción.
- ✓ Ejecutar una sentencia **DDL** (como **CREATE**, **DROP**, **RENAME**, o **ALTER**). La base de datos ejecuta implícitamente una orden **COMMIT** antes y después 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.

Para que los cambios se puedan deshacer con **ROLLBACK** es necesario que la variable de confirmación (**AUTOCOMMIT**) esté a **OFF**. Si está con el valor **ON** cada sentencia se confirmará tras su ejecución.



[Oracle Corporation](#) (Todos los derechos reservados)

Podemos modificar y comprobar su valor, bien desde **SQLDeveloper** eligiendo la opción **Herramientas -> Preferencias**, abriendo la opción **Avanzada de Base de Datos** y marcando o desmarcando la casilla de **Confirmación automática**, como se muestra en la imagen.



[Oracle Corporation](#) (Todos los derechos reservados)

O bien desde **SQLPlus**. El comando **show autocommit** muestra el valor de la variable para la sesión actual. Podemos cambiar el valor de la variable con el comando **SET**

```
C:\app\admin\product\18.0.0\dbhomeXE\bin\sqlplus.exe

SQL> show autocommit
autocommit OFF
SQL> set autocommit on
SQL> show autocommit
autocommit IMMEDIATE
SQL> set autocommit off
SQL>
```

[Oracle Corporation](#) (Todos los derechos reservados)

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.

Correcto, al cerrar el entorno gráfico se vuelcan los cambios de los datos que pudieran quedar pendientes de la última transacción.

Vuelve a intentarlo, porque la respuesta que has dado no es correcta.

Solución

1. Opción correcta
2. Incorrecto

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.



[Sasa Stefanovic](#) (Dominio público)

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.

Vuelve a intentarlo, porque la respuesta que has dado no es correcta.

Correcto. Si se termina la transacción ya no se pueden deshacer los cambios.

Solución

1. Incorrecto

2. Opción correcta

Para saber más

Ejemplo sobre el uso de `Rollback`.

[Ejemplo de `Rollback`.](#)

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.



[Sasa Stefanovic](#) (Dominio público)

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;
```

Para saber más

Artículo sobre los puntos de restauración.

[Savepoint.](#)

Ejemplo sobre el uso de los puntos de restauración.

[Ejemplo de punto de restauración.](#)

Artículo sobre Control de transacciones con ejemplos

[Control transacciones](#)

7.- Problemas asociados al acceso simultáneo a los datos.

Caso práctico

Ana tiene una duda que le quiere preguntar a Juan, ya que se ha estado planteando qué ocurre en el supuesto caso de que dos operaciones simultáneas modifiquen un mismo registro. Por ejemplo, si se ofrece la posibilidad de que se puedan transferir créditos de un usuario a otro, qué ocurriría si justo en un mismo momento dos usuarios le regalan crédito a un tercero. ¿Podría ocurrir que sólo llegara a realizarse una de las dos operaciones?



[Ministerio de Educación](#) (Uso educativo nc)

Para explicarle su idea le plantea el siguiente supuesto: El usuario A no disponía de crédito antes de realizar esas operaciones. El usuario B le va a dar 100 y el C dará 50. Cuando se inicia la operación de B, observa que el saldo de A en ese momento es 0. Cuando todavía no ha terminado la operación de B, se inicia simultáneamente la de C, que consulta el saldo de A que sigue siendo 0 todavía. Cuando B termina de transferir el crédito a A, el saldo se pone a 100 puesto que tenía 0 y le suma sus 100. Pero C estaba haciendo lo mismo, y al saldo 0 que tenía cuando hizo la consulta, le suma 50, por lo que al final sólo le quedará a A como saldo 50 en vez de 150.

Juan le responde que ese tipo de problemas de acceso simultáneo a los datos están controlados en las bases de datos con lo que se denomina bloqueos.

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.



[Sasa Stefanovic](#) (Uso educativo nc)

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.

Para saber más

Interesante enlace sobre control de concurrencia.

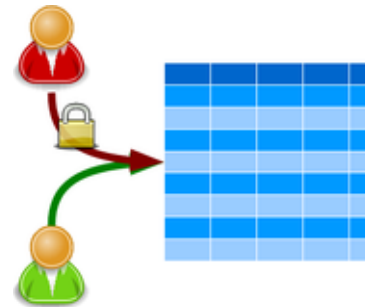
[Integridad. Control de concurrencia.](#) (0.04 MB)

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.



[Sasa Stefanovic](#) (Uso educativo no)

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.
- ☐ Bloqueo optimista.

Correcto. Este tipo de bloqueo impide que un usuario lea los datos del registro hasta que se finalice la modificación que ha empezado otro.

Incorrecto. Este bloqueo permite que se lean los datos con la versión que existía antes de que otro usuario comenzara a cambiarlos.

Solución

1. Opción correcta
2. Incorrecto

Para saber más

Documento, en inglés, sobre los bloqueos optimistas y pesimistas en Oracle con algunos ejemplos.

[Optimistic Locking with Concurrency in Oracle.](#) (0.03 MB)

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. Es el más estricto, cualquier transacción que intenta actualizar una fila en la tabla, debe esperar en cola. La instrucción que define este bloqueo será:



[Sasa Stefanovic](#) (Uso educativo nc)

```
LOCK TABLE nombreTabla IN EXCLUSIVE MODE
```

- ✓ **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.

Para saber más

Definición y ejemplos del bloqueo exclusivo y compartido

[Integridad y control de concurrencia.](#) (pdf - 307,08 KB) (0.30 MB)

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 tipos 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:

Codigo	Nombre	Cod_Juego
1	PARTIDA01	3
2	PARTIDA02	2
3	PARTIDA03	4
4	PARTIDA04	3
5	PARTIDA05	1

[Sasa Stefanovic](#) (Uso educativo no)

- ✓ **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. Las sentencias DML: **INSERT**, **UPDATE** o **DELETE**, realizan un bloqueo exclusivo por la fila afectada por su cláusula **WHERE**. Ocurre lo mismo con la sentencia **SELECT FOR UPDATE**
- ✓ **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.

Para saber más

Definición de bloqueo automático, exclusivo y compartido.

[Bloqueo automático.](#)

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:



[Sasa Stefanovic](#) (Uso educativo no)

- ✓ 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.

Para saber más

Definición y ejemplos sobre transacciones y control de concurrencia con bloqueos manuales.

[Transacciones y control de concurrencia.](#) (0.08 MB)

7.5.- Interbloqueos

Ejemplo de interbloqueo entre dos tablas con cancelación de bloqueo automático a nivel de transacción.

Intentamos crear dos tablas con referencias cruzadas, es decir, la primera hace referencia a la segunda que aún no está creada y viceversa. Vamos a solucionar el problema del interbloqueo considerando la transacción como **DEFERRED**, es decir la comprobación de las restricciones se hará en diferido.

Si creo la primera tabla y hago referencia a la segunda, se producirá un error, aún no existe la segunda tabla:

```
CREATE TABLE Gallina(  
  
idGallina number(3) PRIMARY KEY,  
  
idHuevo number(3) REFERENCES Huevo(idHuevo));  
  
CREATE TABLE Huevo(  
  
idHuevo number(3) PRIMARY KEY,  
  
idGallina number(3) REFERENCES Gallina(idGallina));
```

Informe de error -,

Error SQL: ORA-00942: la tabla o vista no existe

00942. 00000 - "table or view does not exist"

*Cause:

Podemos crear ambas tablas sólo especificando cual es la PK en cada una y a continuación **ALTER TABLE** para añadirle la referencia a la otra tabla.

```
CREATE TABLE Gallina(  
  
idGallina number(3) PRIMARY KEY,  
  
idHuevo number(3) );  
  
CREATE TABLE Huevo(  
  
idHuevo number(3) PRIMARY KEY,  
  
idGallina number(3) );  
  
ALTER TABLE Gallina ADD CONSTRAINT fkRefHuevo  
  
FOREIGN KEY(idHuevo) REFERENCES Huevo(idHuevo);
```

```
ALTER TABLE Huevo ADD CONSTRAINT fkRefGallina  
  
FOREIGN KEY(idGallina) REFERENCES Gallina(idGallina);
```

Intentamos insertar un registro en cada una:

```
INSERT INTO Gallina VALUES(4, 5);  
  
INSERT INTO Huevo VALUES(5, 4);  
  
COMMIT;
```

Da error en las órdenes de inserción, ya que al insertar en la primera tabla, aún no tenemos dato en la segunda y exactamente igual al insertar en la segunda.

Si añadimos **INITIALLY DEFERRED** después de **REFERENCES** en la orden **ALTER TABLE** la comprobación se hará después de **commit** y aunque aún no existan datos en la segunda tabla nos permitirá insertar el registro en la primera.

```
ALTER TABLE Gallina ADD CONSTRAINT fkRefHuevo  
  
FOREIGN KEY(idHuevo) REFERENCES Huevo(idHuevo) INITIALLY DEFERRED;  
  
ALTER TABLE Huevo ADD CONSTRAINT fkRefGallina  
  
FOREIGN KEY(idGallina) REFERENCES Gallina(idGallina) INITIALLY DEFERRED;  
  
INSERT INTO Gallina VALUES(4, 5);  
  
INSERT INTO Huevo VALUES(5, 4);  
  
COMMIT;
```

Las sentencias para borrar las constraints y las tablas tras el ejemplo, son:

```
ALTER TABLE Huevo DROP CONSTRAINT fkRefGallina;  
  
ALTER TABLE Gallina DROP CONSTRAINT fkRefHuevo;  
  
DROP TABLE Huevo;  
  
DROP TABLE Gallina;
```