

TEMA 5

INDICE

1.- Introducción.....	3
2.- Edición de la información mediante herramientas gráficas.....	4
2.1.- Inserción de registros.....	4
2.2.- Modificación de registros.....	5
2.3.- Borrado de registros.....	5
3.- Edición de la información mediante sentencias SQL.....	7
3.1.- Inserción de registros.....	7
3.2.- Modificación de registros.....	8
3.3.- Borrado de registros.....	9
4.- Integridad referencial.....	10
4.1.- Integridad en actualización y supresión de registros.....	10
4.2.- Supresión en cascada.....	11
5.- Subconsultas y composiciones en órdenes de edición.....	13
5.1.- Inserción de registros a partir de una consulta.....	13
5.2.- Modificación de registros a partir de una consulta.....	14
5.3.- Supresión de registros a partir de una consulta.....	14
6.- Transacciones.....	16
6.1.- Hacer cambios permanentes.....	16
6.2.- Deshacer cambios.....	17
6.3.- Deshacer cambios parcialmente.....	18
7.- Problemas asociados al acceso simultáneo a los datos.....	19
Integridad: Control de concurrencia.....	19
Introducción.....	19
Problemas clásicos de concurrencia:.....	20
Técnicas de Bloqueo.....	20
Bloqueo. Variable cerrojo.....	20
Tipos:.....	21
Asegura la seriabilidad.....	21
Prevención el interbloqueo:.....	21
Marcas de Tiempo.....	22
Marcas de tiempo multiversión.....	22
Modelo Multiversión de ORACLE:.....	23
7.1.- Políticas de bloqueo.....	23
7.2.- Bloqueos compartidos y exclusivos.....	24
7.3.- Bloqueos automáticos.....	24
7.4.- Bloqueos manuales.....	25

Tratamiento de datos.

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 obtengas 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.

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.

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.

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

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

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

Por otro lado, la ejecución de las aplicaciones puede fallar en un momento dado y eso no debe impedir que la información almacenada sea 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.

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.

La base de datos Oracle ofrece en su distribución Oracle Database Express la herramienta Application Express a la que puedes acceder en Windows desde Inicio > Todos los programas > Base de Datos Oracle Express Edition > Ir a Página Inicial de Base de Datos.

La única manera de realizar el tratamiento de datos en una base de datos es a través de una herramienta gráfica.



Falso



Verdadero

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.

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 Database Express, se deben seguir los siguientes **pasos**:

1. Ir a la **página inicial** de bases de datos de Oracle Database Express, si no te encuentras en ella.
2. Hacer clic en el botón **Explorador de objetos**. 
3. **Seleccionar una tabla** en la lista izquierda.
4. Seleccionar la pestaña **Datos** y hacer clic en el botón **Insertar Fila**. 
5. **Escribir los datos** correspondientes para cada campo del nuevo registro.
6. 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. 
7. Si la fila se ha **añadido correctamente** se mostrará el mensaje correspondiente. 

Permaneciendo en la vista de la pestaña **Datos** se podrá ver, en la parte central, la **lista de los datos** contenidos en los registros que se han ido insertando.

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: column no permitida aquí*", y no se habrá realizado ninguna operación de la inserción del nuevo registro.

En el enlace *Show Me and Try it* puedes abrir una animación en la que se demuestra el proceso de inserción de registros en una base de datos de Oracle Express utilizando su herramienta gráfica.

http://st-curriculum.oracle.com/tutorial/DBXETutorial/html/module5/les05_ins_tab10_show_me.htm

2.2.- Modificación de registros.

Para insertar registros en tablas, utilizando las herramientas gráficas que ofrece Oracle *Database Express*, se deben seguir los siguientes **pasos**:

1. Ir a la **página inicial** de bases de datos de Oracle Database Express, si no te encuentras en ella.
2. Hacer clic en el botón **Explorador de objetos**. 
3. **Seleccionar una tabla** en la lista izquierda.
4. Seleccionar la pestaña **Datos**. 
5. Debe aparecer, bajo los botones anteriores, una **lista** con los registros que previamente se hayan insertado en la tabla. 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. 
6. 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.
7. 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*.
8. 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 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.

En el enlace *Show Me and Try it* puedes abrir una animación en la que se demuestra el proceso de modificación de registros en una base de datos de Oracle Express utilizando su herramienta gráfica.

http://st-curriculum.oracle.com/tutorial/DBXETutorial/html/module5/les05_upd_rows10_show_me.htm

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 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.

Para eliminar un registro se debe utilizar:

- Botón Suprimir que se encuentra en la lista de datos junto a cada registro.
- Botón Editar junto al registro, y luego el botón Suprimir.**
- Botón derecho del ratón y seleccionar la opción Suprimir del menú desplegado.
- Seleccionar el registro y pulsar la tecla Suprimir.

En el enlace *Show Me and Try it* puedes abrir una animación en la que se demuestra el proceso de eliminación de registros en una base de datos de Oracle Express utilizando su herramienta gráfica.

http://st-curriculum.oracle.com/tutorial/DBXETutorial/html/module5/les05_del_rows10_show_me.htm

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 las aplicaciones de Oracle Express, para más adelante implementarlas en el código fuente de la aplicación.

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).

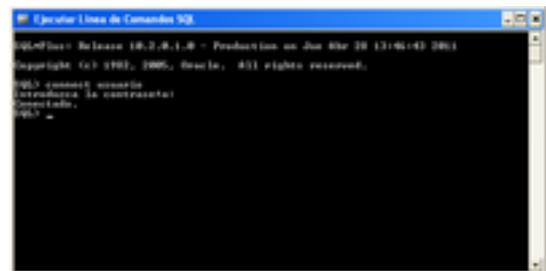
Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web *Application Express* de Oracle utilizando el botón SQL en la página de inicio, y desplegando su lista desplegable elegir **Comandos SQL** > **Introducir Comando**.



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

```
INSERT INTO nombre_tabla (lista_campos) VALUES (lista_valores);
```

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

Tanto la lista de campos *lista_campos* como la de valores *lista_valores*, tendrán separados por comas cada uno de sus elementos. Hay que tener en cuenta también que cada campo de *lista_campos* debe tener un valor válido en la posición correspondiente de la *lista_valores* (Si no recuerdas los valores

válidos para cada campo puedes utilizar la sentencia `DESCRIBE` seguida del nombre de la tabla que deseas consultar).

En el siguiente ejemplo se inserta un nuevo registro en la tabla `USUARIOS` en el que se tienen todos los datos disponibles:

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

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

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

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

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

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

Se obtiene el siguiente error:

```
ORA-00913: demasiados valores
```

¿Cuál de las siguientes sentencias INSERT es correcta?

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

3.2.- 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 borrando la información del campo *Pais* de todos los usuarios:

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

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

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

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

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

En el enlace [Show Me and Try it](http://st-curriculum.oracle.com/tutorial/DBXETutorial/html/module5/les05_upd_rows20_show_me.htm) puedes abrir una animación en la que se demuestra el proceso de modificación de registros en una base de datos de Oracle Express utilizando el lenguaje SQL.

http://st-curriculum.oracle.com/tutorial/DBXETutorial/html/module5/les05_upd_rows20_show_me.htm

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;
```

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

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

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

```
45 fila(s) suprimida(s).
```

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



Verdadero



Falso

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

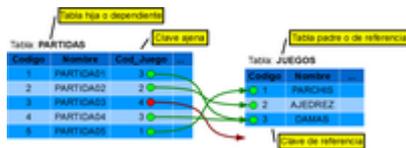
4.- Integridad referencial.

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.

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

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



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

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

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

- ✓ **Clave ajena:** Es el campo o conjunto de campos incluidos en la definición de la restricción que deben hacer referencia a una clave de referencia. En el ejemplo anterior, la clave ajena sería el campo *Cod_Juego* de la tabla *PARTIDAS*.
- ✓ **Clave de referencia:** Clave única o primaria de la tabla a la que se hace referencia desde una clave ajena. En el ejemplo, la clave de referencia es el campo *Codigo* de la tabla *JUEGOS*.
- ✓ **Tabla hija o dependiente:** Tabla que incluye la clave ajena, y que, por tanto, depende de los valores existentes en la clave de referencia. Correspondería a la tabla *PARTIDAS* del ejemplo, que sería la tabla hija de la tabla *JUEGOS*.
- ✓ **Tabla padre o de referencia:** Corresponde a la tabla que es referenciada por la clave ajena en la tabla hija. Esta tabla determina las inserciones o actualizaciones que son permitidas en la tabla hija, en función de dicha clave. En el ejemplo, la tabla *JUEGOS* es padre de la tabla *PARTIDAS*.

Descripción del concepto de integridad referencial con ejemplo.

http://es.wikipedia.org/wiki/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 pueden suponer un problema, ya que no pueden suprimirse registros que son referenciados con una clave ajena desde otra tabla.

Suponiendo el siguiente ejemplo:



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

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

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

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

- ✓ **No Permitir Supresión:** Es la opción por defecto. En caso de que se intente borrar en la tabla de referencia un registro que está siendo referenciado desde otra tabla, se produce un error en la operación de borrado impidiendo dicha acción.
- ✓ **Supresión en Cascada:** Al suprimir registros de la tabla de referencia, los registros de la tabla hija que hacían referencia a dichos registros, también son borrados.
- ✓ **Definir Nulo en Suprimir:** Los valores de la clave ajena que hacían referencia a los registros que hayan sido borrados de la tabla de referencia, son cambiados al valor `NULL`.

Apuntes sobre integridad referencial en Oracle.

<http://www.dsic.upv.es/asignaturas/facultad/lsi/trabajos/172000.doc>

4.2.- Supresión en cascada.

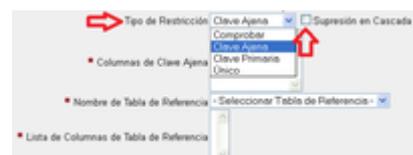
Las opciones de **Supresión en Cascada** o **Definir Nulo en Suprimir** pueden establecerse desde el momento de creación de las tablas, en el tercer paso del asistente que ofrece **Application Express de Oracle**, al establecer las claves ajenas.



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 Application Express**, 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 se está creando una restricción de clave ajena con supresión en cascada, tras usar el botón **Crear** anterior, hay que seleccionar la opción **clave ajena** en la lista desplegable, y marcar la opción **Supresión en Cascada**.



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.

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 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.

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

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

5.1.- Inserción de registros a partir de una consulta.

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

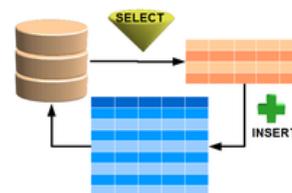
```
INSERT INTO USUARIOS (LOGIN, PASSWORD, NOMBRE, APELLIDOS, CORREO) VALUES
('natsan63',
, 'VBROMI', 'NATALIA', 'SANCHEZ GARCIA', 'natsan63@hotmail.com');
```

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

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

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

Y no sólo eso, sino que es posible insertar en una tabla valores que se obtienen directamente del resultado de una consulta. Supongamos por ejemplo, que disponemos de una tabla



USUARIOS_SIN_CREDITO con la misma estructura que la tabla **USUARIOS**. Si queremos insertar en esa tabla todos los usuarios que tienen el crédito a cero:

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

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

¿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);

5.2.- Modificación de registros a partir de una consulta.

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

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

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

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

5.3.- Supresión de registros a partir de una consulta.

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

Por ejemplo, si se ejecuta la siguiente sentencia:

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

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

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

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

¿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 WHERECodigo IN (SELECTCodigo FROM ANTIGUOS);
```



```
DELETE FROM USUARIOS WHERECodigo IN ANTIGUOS;
```



```
DELETE FROM (SELECTCodigo FROM ANTIGUOS) WHERECodigo IN (SELECTCodigo FROM USUARIOS);
```



```
DELETE FROMCodigo WHERE USUARIOS IN (SELECTCodigo FROM ANTIGUOS);
```

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.

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 aplicadas o guardadas en la base de datos, o bien a todas ellas se les aplica la acción `ROLLBACK`, que interpretamos como deshacer las operaciones que deberían hacer sobre la base de datos.

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

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

- ✓ **Atomicidad:** Todas las tareas de una transacción son realizadas correctamente, o si no, no se realiza ninguna de ellas. No hay transacciones parciales. Por ejemplo, si una transacción actualiza 100 registros, pero el sistema falla tras realizar 20, entonces la base de datos deshace los cambios realizados a esos 20 registros.
- ✓ **Consistencia:** La transacción se inicia partiendo de un estado consistente de los datos y finaliza dejándola también con los datos consistentes.
- ✓ **Aislamiento:** El efecto de una transacción no es visible por otras transacciones hasta que finaliza.
- ✓ **Durabilidad:** Los cambios efectuados por las transacciones que han volcado sus modificaciones, se hacen permanentes.

Las sentencias de control de transacciones gestionan los cambios que realizan las sentencias DML y las agrupa en transacciones. Estas sentencias te permiten realizar las siguientes acciones:

- ✓ Hacer permanentes los cambios producidos por una transacción (`COMMIT`).
- ✓ Deshacer los cambios de una transacción (`ROLLBACK`) desde que fue iniciada o desde un punto de restauración (`ROLLBACK TO SAVEPOINT`). Un punto de restauración es un marcador que puedes establecer dentro del contexto de la transacción. Debes tener en cuenta que la sentencia `ROLLBACK` finaliza la transacción, pero `ROLLBACK TO SAVEPOINT` no la finaliza.
- ✓ Establecer un punto intermedio (`SAVEPOINT`) a partir del cual se podrá deshacer la transacción.
- ✓ Indicar propiedades para una transacción (`SET TRANSACTION`).
- ✓ Especificar si una restricción de integridad aplazable se comprueba después de cada sentencia DML o cuando se ha realizado el `COMMIT` de la transacción (`SET CONSTRAINT`).

6.1.- Hacer cambios permanentes.

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

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

- ✓ Ejecutar una sentencia DDL (como `CREATE`, `DROP`, `RENAME`, o `ALTER`). La base de datos ejecuta implícitamente una orden `COMMIT` antes y después de cada sentencia DDL.
- ✓ Si el usuario cierra adecuadamente las aplicaciones de gestión de las bases de datos Oracle, se produce un volcado permanente de los cambios efectuados por la transacción.

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



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

Verdadero Falso

6.2.- Deshacer cambios.

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

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

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

Para deshacer los cambios de la transacción simplemente debes indicar:

```
ROLLBACK;
```

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

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

Verdadero Falso

Ejemplo sobre el uso de Rollback.

```
CREATE TABLE emp_name AS SELECT employee_id, last_name FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);
CREATE TABLE emp_sal AS SELECT employee_id, salary FROM employees;
CREATE UNIQUE INDEX empsal_ix ON emp_sal (employee_id);
CREATE TABLE emp_job AS SELECT employee_id, job_id FROM employees;
CREATE UNIQUE INDEX empjobid_ix ON emp_job (employee_id);

DECLARE
    emp_id          NUMBER(6);
    emp_lastname   VARCHAR2(25);
    emp_salary     NUMBER(8,2);
    emp_jobid      VARCHAR2(10);
BEGIN
    SELECT employee_id, last_name, salary, job_id INTO emp_id, emp_lastname,
           emp_salary, emp_jobid FROM employees WHERE employee_id = 120;
    INSERT INTO emp_name VALUES (emp_id, emp_lastname);
```

```

INSERT INTO emp_sal VALUES (emp_id, emp_salary);
INSERT INTO emp_job VALUES (emp_id, emp_jobid);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Inserts have been rolled back');
END;
/

```

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;
```

Artículo sobre los puntos de restauración.

<http://es.wikipedia.org/wiki/Savepoint>

Ejemplo sobre el uso de los puntos de restauración.

http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14261/sqloperations.htm#BABGAAIG

```

CREATE TABLE emp_name AS SELECT employee id, last name, salary FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);

DECLARE
  emp_id          employees.employee_id%TYPE;
  emp_lastname   employees.last_name%TYPE;
  emp_salary     employees.salary%TYPE;
BEGIN
  SELECT employee_id, last_name, salary INTO emp_id, emp_lastname,
    emp_salary FROM employees WHERE employee_id = 120;
  UPDATE emp_name SET salary = salary * 1.1 WHERE employee_id = emp_id;
  DELETE FROM emp_name WHERE employee_id = 130;
  SAVEPOINT do_insert;
  INSERT INTO emp_name VALUES (emp_id, emp_lastname, emp_salary);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK TO do_insert;
    DBMS_OUTPUT.PUT_LINE('Insert has been rolled back');
END;
/

```

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 regalaban crédito a un tercero. ¿Podría ocurrir que sólo llegara a realizarse una de las dos operaciones?

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.

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.

[Interesante enlace sobre control de concurrencia.](#)

Integridad: Control de concurrencia.

Introducción

En sistema multiusuario es imprescindible, un mecanismo de control de concurrencia para conservar la integridad de la BD.

- ✓ Todos los datos deben ser iguales para todos los usuarios.

Cuando se ejecutan varias transacciones simultáneamente pueden producirse estados inconsistentes en la BD:

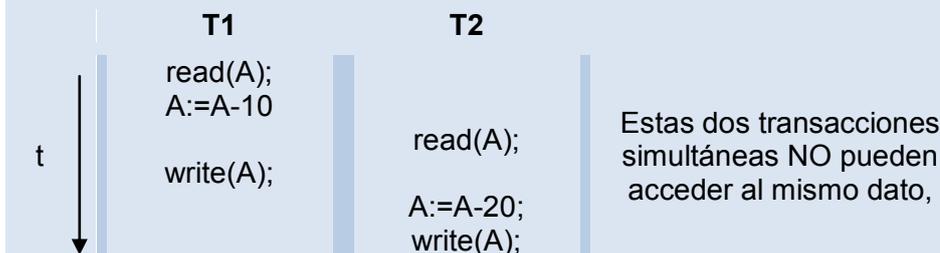
- ✓ Una transacción bancaria lee un importe y le resta 100 euros y antes de actualizar la BBDD otra transacción lee ese dato.

ORACLE es una BD multiusuario. Se necesita

- ✓ Concurrencia
- ✓ Consistencia

Maximización de concurrencia → *maximiza productividad y desarrollo*

Ejemplo de “problema” de concurrencia:



OBJETIVO para controlar la concurrencia:

Garantizar que las transacciones sean seriales así, se garantizará la consistencia de las transacciones.

Problemas clásicos de concurrencia:

- ✓ Modificación perdida
- ✓ Modificación temporal
- ✓ Totalización incorrecta
- ✓ Lectura no repetible

En **ORACLE** los fenómenos prevenibles son:

- ✓ Lectura sucia
- ✓ Lectura no repetible (borrosa)
- ✓ Lectura fantasma

Para prevenir problemas



Técnicas de Control de Concurrencia

- ✓ **Técnicas Pesimistas (Prevención)**
 - ➔ Bloqueos, Marcas de Tiempo, ...
- ✓ **Técnicas Optimistas (Corrección)**
 - ➔ Técnicas de Validación

ORACLE tiene como soluciones:

- ✓ Varios tipos de **bloqueo**
- ✓ Un **modelo** de consistencia **multiversión**.
- ✓ Niveles de **AISLAMIENTO**:
 - ➔ Aceptación de lectura (read committed)
 - ➔ Serializable
 - ➔ Modo de solo lectura (read-only mode)

ORACLE para prevenir interacción destructiva de datos entre usuarios:

- ✓ Consistencia: Asegura que los datos que estamos viendo no cambian (por otros usuarios) hasta que acabemos la transacción
- ✓ Integridad: Asegura que los datos y estructuras reflejan los cambios en una secuencia correcta.

Técnicas de Bloqueo.

Bloqueo. Variable cerrojo

Un bloqueo, asocia una variable (cerrojo) a cada elemento de datos que describe el estado de dicho elemento, respecto a las posibles operaciones que sobre él se puede realizar.

- ✓ Identificador del Elemento bloqueado
- ✓ Identificador de la Transacción que lo bloquea

Una transacción obtiene un bloqueo solicitándolo al Gestor de Bloqueos.

Un bloqueo es una garantía de ciertos derechos de exclusividad para la Transacción.

Tipos:

- ✓ Bloqueos exclusivos: Un único bloqueo por recurso
- ✓ Bloqueos Compartidos: Muchos bloqueos por recurso

ORACLE tiene los dos tipos de bloqueo así como control de consistencia multiversión para asegurar acceso concurrente a los datos.

Un **Protocolo de Bloqueo** indica cuando una transacción puede bloquear y desbloquear elementos. Si los bloqueos se realizaran de manera arbitraria se podrían producir inconsistencias.

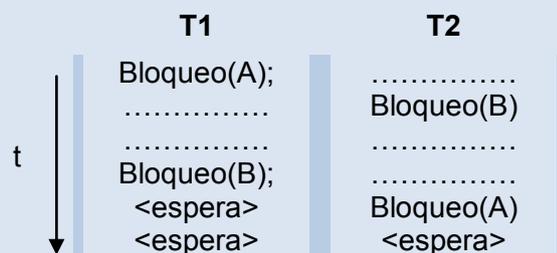
Protocolo de Bloqueo en dos Fases (Two-Phase-Locking)

- ✓ Fase de Crecimiento: Bloquea pero no Libera
- ✓ Fase de Encogimiento: Libera pero no Bloquea

Asegura la seriabilidad

Problema de Interbloqueo. DEADLOCK

Dos o más transacciones se quedan a la espera de que se liberen elementos que tiene bloqueados otra.



Puede producirse incluso con protocolos de bloqueo que garantizan la seriabilidad

Soluciones:

- ✓ **Prevención del interbloqueo**, obligando a que las transacciones bloqueen todos los elementos que necesitan por adelantado
- ✓ **Detectar el interbloqueo**, controlando de forma periódica si se ha producido, para lo que suele construir un grafo de espera:
 - ➔ Cada nodo representa una transacción
 - ➔ Existe un arco entre T_i y T_j si T_i está esperando un recurso que tiene bloqueado T_j
 - ➔ Existe interbloqueo si el Grafo tiene un ciclo.

Cada SGBD tiene su propia política para escoger las víctimas, aunque suelen ser las transacciones más recientes.

Prevención el interbloqueo:

- ✓ Bloquear todo lo necesario por adelantado
- ✓ Comprobar por adelantado la posibilidad de Interbloqueo, la transacción esperará en el caso en que de atenderse su petición se produciría Interbloqueo.
- ✓ Nuevos protocolos de bloqueo que además de garantizar la seriabilidad, eviten el interbloqueo. (ej. Protocolo del árbol).
- ✓ Utilización de Marcas de Tiempo para cada transacción $MT(T_i)$:
 - ➔ Usar expropiación y retroceso de Transacciones
 - ➔ esperar-morir (Wait – Die)

→ herir-esperar (Wound – Wait)

Detección del Interbloqueo y Recuperación

Periódicamente se comprueba la existencia de Interbloqueo:

- ✓ Si se detecta Interbloqueo, se elige una transacción como víctima y se la mata.
- ✓ Se puede producir inanición si se elige siempre a la misma víctima.

¿Cuándo usar prevención y cuando detección?

GRANULARIDAD del Bloqueo: para mejorar rendimiento

Son posibles diversos niveles de bloqueo

- ✓ un campo de un registro
- ✓ un registro
- ✓ un fichero
- ✓ la base de datos

Granularidad gruesa → menor gestión de bloqueos y menor nivel de concurrencia.

Granularidad fina → mayor gestión de bloqueos y mayor nivel de concurrencia.

ORACLE bloquea automáticamente:

- ✓ **Objetos de usuario** (tablas, filas, etc. (estructuras y datos))
- ✓ **Objetos del sistema** à invisibles a los usuarios (estructuras de datos compartidas en memoria, filas de diccionario de datos, etc.)

Marcas de Tiempo

En lugar de determinar el orden entre las transacciones en conflicto en función del momento del acceso a los elementos, determinan por adelantado una ordenación de las transacciones.

El interbloqueo es imposible.

Una marca de tiempo es un identificador único asociado a cada transacción

Las actualizaciones físicas se retrasan hasta la confirmación de las transacciones. No se puede actualizar ningún elemento actualizado por otra transacción más reciente.

Protocolos:

- ✓ **Wait-die** que fuerza a una transacción a esperar en caso de que entre en conflicto con otra transacción cuya marca de tiempo sea más reciente, o a morir (abortar y reiniciar) si la transacción que se está ejecutando es más antigua.
- ✓ **Wound-wait**, que permite a una transacción matar a otra que posea una marca de tiempo más reciente, o que fuerza a la transacción peticionaria a esperar.

Marcas de tiempo multiversión

Varias transacciones leen y escriben diferentes versiones del mismo dato siempre que cada transacción sea un conjunto consistente de versiones de todos los datos a los que accede.

ORACLE utiliza un sistema multiversión

Protocolo:

- ✓ Está basado en marcas de tiempo
- ✓ El control de concurrencia es de varias versiones a la vez de un item de datos.

- ✓ Cuando una transacción requiere acceder a un ítem, la marca de tiempo de la transacción es comparada con las marcas de tiempo de las diferentes versiones del ítem.
- ✓ Se elige una versión de los ítems para mantener la serializabilidad del plan de ítems que se este ejecutando.

Modelo Multiversión de ORACLE:

Consistencia en lectura:

- ✓ Imaginar cada usuario operando con una copia privada de la BD (modelo consistente multiversión)
- ✓ Características
 - garantiza que los datos no cambian durante la ejecución
 - Los lectores no esperan a los que escriben o a otros lectores
 - Los que escriben no esperan a los lectores, pero sí a otros escritores de los mismos datos
- ✓ Niveles de concurrencia
 - Sentencia
 - Transacción

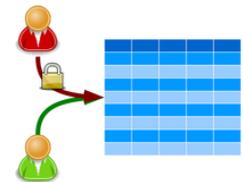
Cuando se lee y modifica al mismo tiempo, Oracle crea:

- ✓ Conjunto de datos (vistas) consistente en lectura
- ✓ Cuando se modifica (antes del COMMIT)
 - se almacenan los valores antiguos de los datos en *Segmentos de Rollback*
 - Crea la Vista Consistente a partir de:
 - § Información actual en el *Área Global del Sistema*
 - § Información antigua en los *Segmentos de Rollback*
- ✓ Al hacer el COMMIT:
 - Se hacen los cambios permanentes para todas las vistas posteriores

[http://ocw.uc3m.es/ingenieria-informatica/disenio-y-administracion-de-bases-de-datos/teoria/Tema4 6%28Administracion Concurrencia%29.pdf](http://ocw.uc3m.es/ingenieria-informatica/disenio-y-administracion-de-bases-de-datos/teoria/Tema4%20Administracion%20Concurrencia%29.pdf)

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, 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.

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

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

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

<http://www.oraFAQ.com/papers/locking.pdf>

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.

Definición y ejemplos del bloqueo exclusivo y compartido

<http://dircompucv.ciens.ucv.ve/generador/sites/administracion-de-bd/archivos/Integridad.pdf>

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.

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

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.

Definición de bloqueo automático, exclusivo y compartido.

<http://books.google.com/books?id=zYBWm5-X5usC&lpg=PA166&ots=ha7MVfOo9n&pg=PA166#v=onepage&q&f=true>

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 puede 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.

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

<http://www.fdi.ucm.es/profesor/hector/DB-II/transacciones.pdf>