5. Tratamiento de datos

Class	Bases de datos
Column	Xerach Casanova
Lass Edited time	@Feb 4, 2021 10:46 PM

1. Introducción

Las operaciones de tratamiento de datos son las acciones que permiten añadir, modificar o suprimir información de la base de datos.

Este tipo de operaciones debe seguir una serie de requisitos en las relaciones existentes entre las tablas que la componen. Todas las operaciones que se realicen deben asegurar que se cumplen las relaciones existentes entre ellas en todo momento.

Además, si una aplicación falla en un momento dada no debe impedir que la información almacenada sea incorrecta, así como la posibilidad de cancelar una determinada operación no debe suponer un problema para la fiabilidad de los datos almacenados.

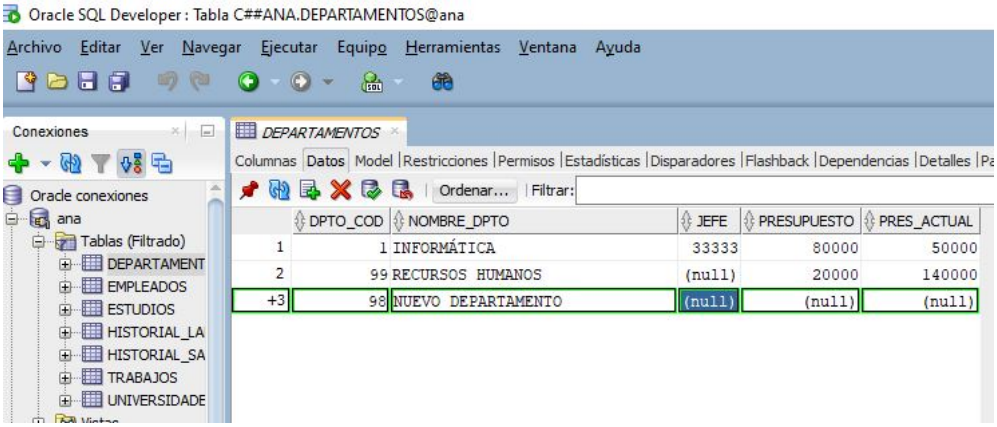
2. Edición de la información mediante herramientas gráficas

Para realizar el tratamiento de datos por línea de comandos se requiere la utilización de lenguaje SQL, sin embargo, con las herramientas gráficas para manipulación de datos, se permite la introducción, edición y borrado de datos desde un entorno gráfico.

En Oracle podemos trabajar con SQLDeveloper y en otros, como en SGDB Mysql se encuentran herramientas gráficas similares, como phpmyadmin.

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 SQLDeveloper se realizan haciendo click en cualquier tabla, pulsando en la pestaña datos y haciendo click en el icono de añadir.

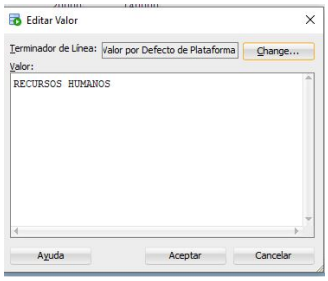


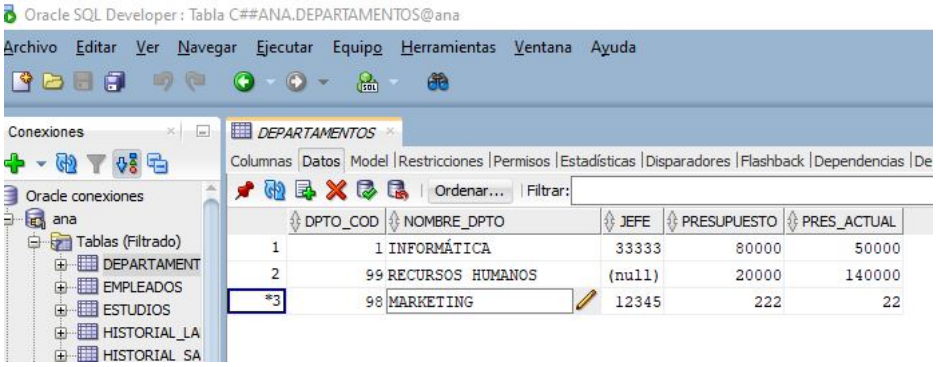
Cuando se finaliza la inserción de todas las filas se confirma la operación con el botón correspondiente y si todo va bien, se visualizará un mensaje indicando que la inserción se ha realizado de forma correcta.

Si se produce un error, se debe comprobar que mensaje muestra para solucionarlo, por ejemplo por intentar insertar datos alfanuméricos en una columna numérica.

2.2. Modificación de registros

Permite la modificación de datos ya existentes en las tablas. Para ello nos situamos directamente en la columna de la fila que se quiere modificar, hacemos click con el ratón y escribimos el nuevo contenido. También se puede realizar pulsando el icono del lápiz que aparece al lado de la columna e introduciendo el valor en la ventana emergente.



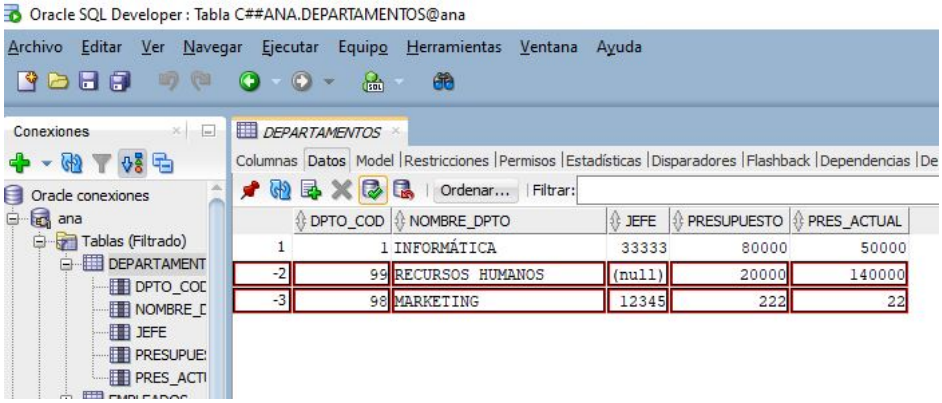


Para realizar las modificaciones se confirma la operación en el botón correspondiente y también debemos analizar los errores en caso de que se hayan producido.

2.3. Borrado de registros

Permite la eliminación de datos existentes en las tablas. Para eliminar un registro o fila de una tabla, en la misma pestaña de datos, nos situamos en la fila que se quiera eliminar, en cualquier columna y pulsamos el icono en forma de

aspas. Al hacerlo se colocará el signo - delante del número de cada fila.



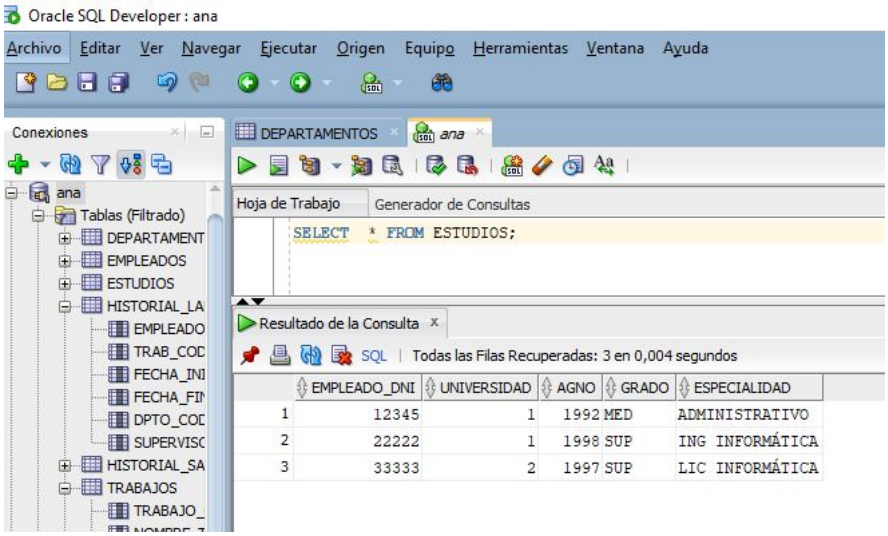
Se debe confirmar el borrado con el botón correspondiente, pero también podemos deshacer los borrados antes de confirmar en el icono que se muestra.

La eliminación de un registro no se puede realizar si un registro de otra tabla hace referencia a él y se mostrará un mensaje de error en pantalla. Si esto ocurre, para eliminar el registro se debe eliminar el registro que hace referencia a él o modificarlo para que haga referencia a otro registro.

3. Edición de la información mediante sentencias SQL

El lenguaje SQL dispone de una serie de sentencias para la edición de datos. Este conjunto de sentencias recibe el nombre de DML (Data Manipulation Language).

Estas se pueden ejecutar en SQLPlus o escribéndolas en cualquier hoja de trabajo de SQLDeveloper.



Para trabajar con SQLDeveloper pulsamos en el botón SQL, del menú superior. Se solicitará la conexión con la que vamos a trabajar y seguidamente se abrirá la hoja de trabajo. para ejecutar las sentencias se utiliza el botón play.

También se puede acceder a todas las sentencias SQL que se han escrito aunque no se hayan guardado con el botón que se indica.

3.1. Inserción de registros

La sentencia INSERT permite insertar nuevas filas o registros. El formato más sencillo es:

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

nombre_tabla es el nombre de la tabla donde se añaden los nuevos registros, lista_campos son los campos de dicha tabla en los que se van a insertar los valores de lista_valores. Es posible omitir la lista de campos si se van a insertar todos los valores de cada campo y en el orden en el que se encuentran en la tabla.

La lista de campos debe tener un valor válido en la posición correspondiente de lista_valores. Si no se recuerda bien se puede utilizar la sentencia DESCRIBE seguido del nombre de la tabla a consultar.

Con el siguiente script podemos hacer pruebas en las tablas de JuegosOnline, ejecutando el script en sqlplus.

```
-- Desde CONN SYS AS SYSDBA/ORACLE;
create user c##JUEGOS identified by juegos default tablespace users;
grant connect, resource,DBA to c##JUEGOS;
conn c##JUEGOS/juegos
CREATE TABLE USUARIOS (
         login         VARCHAR2(15) PRIMARY KEY NOT NULL,
         password      VARCHAR2(9) NOT NULL,
         nombre        VARCHAR2(25) NOT NULL,
         apellidos     VARCHAR2(30) NOT NULL,
         direccion     VARCHAR2(30) NOT NULL,
         cp            VARCHAR2(5) NOT NULL,
         localidad     VARCHAR2(25) NOT NULL,
         provincia     VARCHAR2(25) NOT NULL,
         pais          VARCHAR2(15) NOT NULL,
         f_nac      DATE,
         f_ing      DATE DEFAULT (sysdate),
         correo     VARCHAR2(25) NOT NULL,
         credito    NUMBER,
         sexo       VARCHAR2(1));

--TABLA JUEGOS
CREATE TABLE JUEGOS(
         codigo        VARCHAR2(15) PRIMARY KEY NOT NULL,
         nombre        VARCHAR2(15) NOT NULL,
         descripcion   VARCHAR2(200) NOT NULL);

--TABLA PARTIDAS

CREATE TABLE PARTIDAS(
         codigo        VARCHAR2(15) PRIMARY KEY NOT NULL,
         nombre        VARCHAR2(25) NOT NULL,
         estado        VARCHAR2(1) NOT NULL,
         cod_juego     VARCHAR2(15) NOT NULL
                       CONSTRAINT CA_cod_juego REFERENCES JUEGOS(codigo),
         fecha_inicio  DATE,
         hora_inicio   TIMESTAMP,
         cod_creador   VARCHAR2(15)
                       CONSTRAINT CA_cod_creador REFERENCES USUARIOS(login));
--TABLA UNEN
CREATE TABLE UNEN(
        codigo_partida VARCHAR2(15) NOT NULL
                       CONSTRAINT CA_codigo_partida REFERENCES PARTIDAS(codigo),
        codigo_usuario VARCHAR2(15) NOT NULL
                       CONSTRAINT CA_codigo_usuario REFERENCES USUARIOS(login),
        CONSTRAINT PK_UNEN primary key (codigo_partida, codigo_usuario));
    ALTER SESSION SET NLS_DATE_FORMAT='MM/DD/YYYY';
INSERT INTO USUARIOS VALUES('anamat56','JD9U6?','ANA M.','MATA VARGAS','GARCILASO DE LA VEGA','8924','SANTA COLOMA DE GRAMANET','BARCE
INSERT INTO USUARIOS VALUES('alecam89','5;5@PK','ALEJANDRO EMILIO','CAMINO LAZARO','PEDRO AGUADO BLEYE','34004','PALENCIA','PALENCIA',
INSERT INTO USUARIOS VALUES('verbad64','MP49HF','VERONICA','BADIOLA PICAZO','BARRANCO GUINIGUADA','35015','PALMAS GRAN CANARIA,LAS','P
INSERT INTO USUARIOS VALUES('conmar76','O1<N9U','CONSUELO','MARTINEZ RODRIGUEZ','ROSA','4002','ALMERÍA','ALMERÍA','ESPAÑA','08/09/1978
INSERT INTO USUARIOS VALUES('encpay57','FYC3L5','ENCARNACIÓN','PAYO MORALES','MULLER,AVINGUDA','43007','TARRAGONA','TARRAGONA','ESPAÑA
INSERT INTO USUARIOS VALUES('mandia79','00JRIH','MANUELA','DIAZ COLAS','214 (GENOVA)','7015','PALMA DE MALLORCA','BALEARES','ESPAÑA','
INSERT INTO USUARIOS VALUES('alibar52','IER8S','ALICIA MARIA','BARRANCO CALLIZO','HECTOR VILLALOBOS','29014','MÁLAGA','MÁLAGA','ESPAÑA
INSERT INTO USUARIOS VALUES('adofid63',';82=MH','ADOLFO','FIDALGO DIEZ','FORCALL','12006','CASTELLÓN DE LA PLANA','CASTELLÓN','ESPAÑA'
INSERT INTO USUARIOS VALUES('jesdie98','X565ZS','JESUS','DIEZ GIL','TABAIBAL','35213','TELDE','PALMAS (LAS)','ESPAÑA','10/23/1981','09
INSERT INTO USUARIOS VALUES('pedsan70','T?5=J@','PEDRO','SANCHEZ GUIL','PINTOR ZULOAGA','3013','ALACANT/ALICANTE','ALICANTE','ESPAÑA',
INSERT INTO USUARIOS VALUES('diahue96','LSQZMC','DIANA','HUERTA VALIOS','JOAQUIN SALAS','39011','SANTANDER','CANTABRIA','ESPAÑA','04/2
INSERT INTO USUARIOS VALUES('robrod74','<LQMLP','ROBERTO','RODRIGUEZ PARMO','CASTILLO HIDALGO','51002','CEUTA','CEUTA','ESPAÑA','06/28
INSERT INTO USUARIOS VALUES('milgar78','SF=UZ8','MILAGROSA','GARCIA ELVIRA','PEDRALBA','28037','MADRID','MADRID','ESPAÑA','04/12/1983'
INSERT INTO USUARIOS VALUES('frabar93','19JZ7@','FRANCISCA','BARRANCO RODRIGUEZ','BALSAS, LAS','26006','LOGROÑO','RIOJA (LA)','ESPAÑA'
INSERT INTO USUARIOS VALUES('migarc93','AAFLTW','MIGUEL ANGEL','ARCOS ALONSO','ISAAC ALBENIZ','4008','ALMERÍA','ALMERÍA','ESPAÑA','03/

--TABLA JUEGOS

INSERT INTO JUEGOS VALUES('1','Parchís','El parchís es un juego de mesa derivado del pachisi y similar al ludo y al parcheesi');
INSERT INTO JUEGOS VALUES('2','Oca','El juego de la oca es un juego de mesa para dos o más jugadores');
INSERT INTO JUEGOS VALUES('3','Ajedrez','El ajedrez es un juego entre dos personas, cada una de las cuales dispone de 16 piezas móvile
INSERT INTO JUEGOS VALUES('4','Damas','Las damas es un juego de mesa para dos contrincantes');
INSERT INTO JUEGOS VALUES('5','Poker','El póquer es un juego de cartas de los llamados de "apuestas"');
INSERT INTO JUEGOS VALUES('6','Chinchón','El chinchón es un juego de naipes de 2 a 8 jugadores');
```

```
INSERT INTO JUEGOS VALUES('7','Mus','El mus es un juego de naipes, originario de Navarra, que en la actualidad se encuentra muy extend
INSERT INTO JUEGOS VALUES('8','Canasta','La canasta o rummy-canasta es un juego de naipes, variante del rummy');
INSERT INTO JUEGOS VALUES('9','Dominó','El dominó es un juego de mesa en el que se emplean unas fichas rectangulares');
INSERT INTO JUEGOS VALUES('10','Pocha','La pocha es un juego de cartas que se juega con la baraja española');
INSERT INTO JUEGOS VALUES('11','Backgammon','Cada jugador tiene quince fichas que va moviendo entre veinticuatro triángulos (puntos) s
INSERT INTO JUEGOS VALUES('12','Billar','El billar es un deporte de precisión que se practica impulsando con un taco un número variabl

-- PARTIDAS

INSERT INTO PARTIDAS VALUES('1','Billar_migarc93_18/7','1','12','07/18/2011',TO_TIMESTAMP ('00:47:40','HH24:MI:SS'),'migarc93');
INSERT INTO PARTIDAS VALUES('2','Chinchón_mandia79_2/10','1','6','10/02/2011',TO_TIMESTAMP ('01:47:40','HH24:MI:SS'),'mandia79');
INSERT INTO PARTIDAS VALUES('3','Canasta_alibar52_26/2','0','8','02/26/2011',TO_TIMESTAMP ('08:57:33','HH24:MI:SS'),'alibar52');
INSERT INTO PARTIDAS VALUES('4','Damas_verbad64_16/3','1','4','03/16/2011',TO_TIMESTAMP ('00:53:00','HH24:MI:SS'),'verbad64');
INSERT INTO PARTIDAS VALUES('5','Chinchón_alibar52_9/9','1','6','09/09/2011',TO_TIMESTAMP ('09:10:22','HH24:MI:SS'),'alibar52');
INSERT INTO PARTIDAS VALUES('6','Oca_pedsan70_21/12','0','2','12/21/2011',TO_TIMESTAMP ('18:53:17','HH24:MI:SS'),'pedsan70');
INSERT INTO PARTIDAS VALUES('7','Canasta_encpay57_18/2','0','8','02/18/2011',TO_TIMESTAMP ('09:41:02','HH24:MI:SS'),'encpay57');
INSERT INTO PARTIDAS VALUES('8','Pocha_adofid63_26/10','1','10','10/26/2011',TO_TIMESTAMP ('02:23:43','HH24:MI:SS'),'adofid63');
INSERT INTO PARTIDAS VALUES('9','Damas_diahue96_25/6','1','4','06/25/2011',TO_TIMESTAMP ('18:11:14','HH24:MI:SS'),'diahue96');
INSERT INTO PARTIDAS VALUES('10','Parchís_encpay57_31/7','1','1','07/31/2011',TO_TIMESTAMP ('21:21:36','HH24:MI:SS'),'encpay57');

--TABLA UNEN

INSERT INTO UNEN VALUES('4','anamat56');
INSERT INTO UNEN VALUES('3','alecam89');
INSERT INTO UNEN VALUES('6','alecam89');
INSERT INTO UNEN VALUES('2','conmar76');
INSERT INTO UNEN VALUES('2','encpay57');
INSERT INTO UNEN VALUES('2','mandia79');
INSERT INTO UNEN VALUES('4','alibar52');
INSERT INTO UNEN VALUES('3','adofid63');
INSERT INTO UNEN VALUES('5','jesdie98');
INSERT INTO UNEN VALUES('8','pedsan70');
INSERT INTO UNEN VALUES('6','diahue96');
INSERT INTO UNEN VALUES('4','robrod74');
INSERT INTO UNEN VALUES('5','milgar78');
INSERT INTO UNEN VALUES('4','frabar93');
INSERT INTO UNEN VALUES('5','encpay57');
```

Antes de ejecutar las siguiente sentencias debemos ejecutar la siguiente, para que se tome la fecha en el formato que vamos a insertar registros.

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

En este ejemplo hacemos una inserción de registros indicando toda la lista de campos:

```
INSERT INTO USUARIOS (Login, Password, Nombre, Apellidos, Direccion, CP, Localidad, Provincia, Pais, F_Nac,
F_Ing, 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');
```

Y en este otro, se inserta un registro pero sin disponer de todos los datos.

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

En aquellos campos que no se especifiquen valores, se obtiene NULL. Si la lista de campos indicados no se corresponde con la lista de valores, o no se proporcionan valores para campos que no admiten valor NULL se obtiene un error en ejecución

```
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

3.2. Modificación de registros

La sentencia UPDATE permite modificar una serie de valores de determinados registros de las tablas:

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

nombre_tabla es el nombre de la tabla en la que modificaremos datos. Se indican en cada campo el nuevo valor utilizando el signo =, cada emparejamiento campo=valor, debe ir separado por comas.

La cláusula Where seguida de la condición es opcional, si se indica, la modificación afecta solo a los registros que la cumplen y si no, afectará a todos los registros.

Añadimos 200 al crédito de todos los usuarios:

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

Ponemos a cero el crédito y ponemos en nulo el valor del campo f_nac de todos los usuarios:

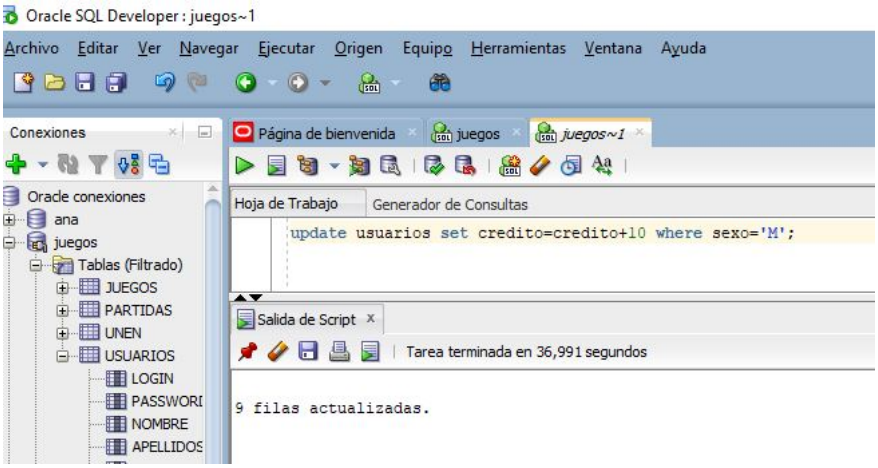
```
UPDATE USUARIOS SET Credito = 0, f_nac = NULL;
```

Damos 300 de crédito a los registros que correspondan a mujeres:

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

Cuando termina la ejecución de una sentencia UPDATE se muestra la cantidad de registros actualizados o se obtiene un error si algo ha ido mal.

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



3.2. Borrado de registros

La sentencia DELETE permite borrar registros de una tabla:

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

nombre_tabla hace referencia a la tabla donde se va a realizar la operación, la cláusula where es opcional y si no se usa se borrará el contenido de toda la tabla aunque la estructura seguirá intacta.

Elimina todos los usuarios:

```
DELETE FROM USUARIOS;
```

Borramos los usuarios que tengan crédito cero:

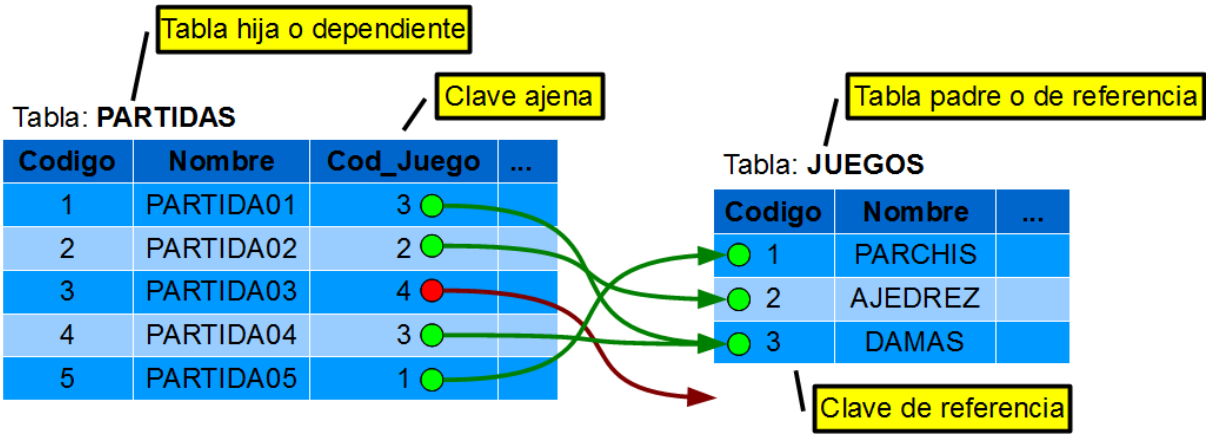
```
DELETE FROM USUARIOS WHERE Credito = 0;
```

Se obtendrá un mensaje de error si se produce algún problema o el número de filas eliminadas.

4. Integridad referencial

Dos tablas pueden estar relacionadas entre ellas por uno o más campos que tengan en común. Cada valor del campo que forma parte de la integridad referencial definida, debe corresponderse en otra tabla con otro registro que contenga el mismo valor en el campo referenciado.

En el ejemplo de juegos online, podemos tener una tabla PARTIDAS en la que existe una referencia al tipo de juego que le corresponde, mediante su código. No puede existir ninguna partida que no tenga un código que corresponda con alguno de la tabla juegos. Por tanto, en la siguiente imagen, el código de juego 4 no cumple la integridad referencial, pues no existe un juego con código 4.



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

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

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 valores.

Si modificamos 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. Del mismo modo, no podemos modificar el valor de la clave principal en un registro de la tabla padre, si una clave ajena hace referencia a ese registro.

En los borrados de registros no se pueden suprimir registros referenciados con una clave ajena desde otra tabla.

En el registro de la partida con nombre PARTIDA01, no podemos modificar el valor de Cod_Juego a 4, debido a que no existe en la tabla JUEGOS un registro con esa clave primaria.

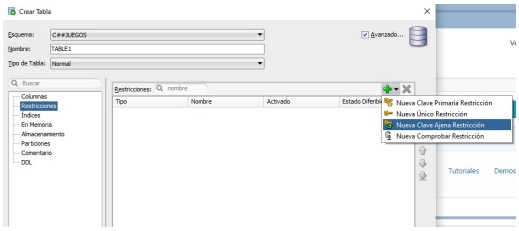
El código de juego DAMAS no se puede cambiar, ya que hay registros en la tabla PARTIDAS que hacen referencia a dicho juego a través del campo Cod_Juego.

Para borrar o modificar registros en una tabla de referencia, se puede configurar la clave ajena de distintas maneras para conservar la integridad referencial:

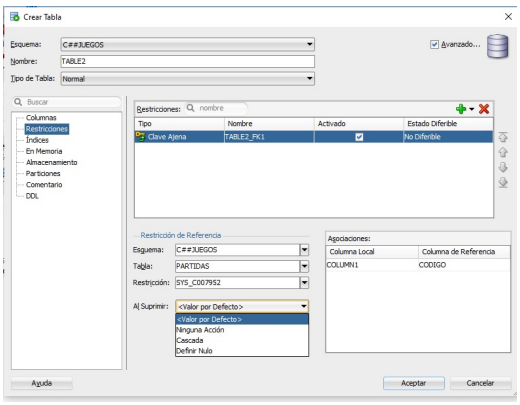
- **No permitir supresión ni modificación.** Es la opción por defecto, en caso de borrar o modificar en la tabla de referencia un registros que está siendo referenciado desde otra tabla, se produce error.
- **Supresión a modificación en Cascada (ON DELETE/UPDATE CASCADE).** Al suprimir o modificar registros en la tabla de referencia, los registros de la tabla hija también son borrados o modificados.
- **Asignación de Nulo (ON DELETE/UPDATE SET NULL).** Los valores de la clave ajena que hacían referencia a los registros borrados o modificados se cambian a NULL.
- **Valor por defecto (ON DELETE/UPDATE DEFAULT).** Los valores de la clave ajena que hacían referencia a los registros borrados o modificados se cambian a su valor por defecto.

4.2. Supresión en cascada

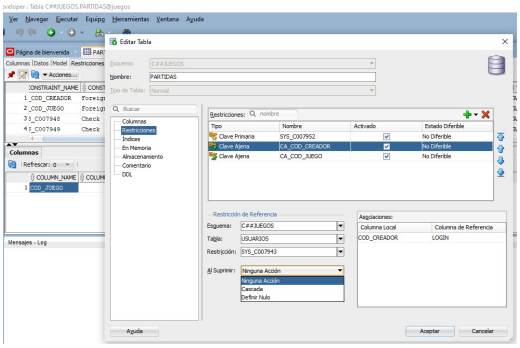
Las opciones de supresión en Cascada o Definir Nulo en suprimir se pueden establecer desde que creamos las tablas, desde SQLDeveloper marcando en la casilla avanzado, nos situamos en la columna que es clave ajena y seleccionamos en el árbol de restricciones, en la lista desplegable, del botón Añadir, seleccionamos nueva clave ajena Restricción como se muestra en imagen.



Seguidamente se selecciona "en cascada" en el desplegable "al suprimir"



Si la tabla ya estaba creada y se desea establecer una restricción de clave ajena con una opción concreta, se puede realizar desde la opción "editar tabla" siguiendo los mismos pasos.



Para realizar este tipo de operaciones, se dispone de las siguientes opciones durante la declaración de la clave ajena en la tabla (también podemos usar ON DELETE SET NULL):

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

Si queremos añadirlo después de haber sido creado, podemos usar la estructura ALTER TABLE.

5. Subconsultas y composiciones en órdenes de edición

Podemos utilizar sentencias de una forma más avanzada insertando consultas dentro de esas mismas operaciones de tratamiento de datos.

Una tabla se puede ver afectada por los resultados de las operaciones en otras tablas. Con una misma instrucción, podemos añadir más de un registro a una tabla, o actualizar o eliminar varios registros basados en otras consultas.

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

También podemos usar como condiciones en las sentencias, otras consultas.

5.1. Inserción de registros a partir de una consulta.

El siguiente INSERT

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

también se puede realizar así:

```
INSERT INTO (SELECT Login, Password, Nombre, Apellidos, direccion,cp,localidad,provincia,pais,Correo FROM USUARIOS) VALUES ('natsan63'
'VBROMI', 'NATALIA', 'SANCHEZ GARCIA','C/Blanca','28003','Madrid','Madrid','Spain', 'natsan63@hotmail.com');
```

Sustituyendo el nombre de la tabla, junto con sus campos, por una consulta equivalente.

También es posible insertar una tabla de valores que se obtienen directamente del resultado de una consulta.

Supongamos que tenemos 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;
```

En este caso, no se utiliza la palabra values, ya que no se está especificando una lista de valores.

También podemos crear una tabla e insertar datos a partir de una consulta:

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

Si queremos crear una tabla con la misma estructura pero sin contenido, podemos crear una condición que no se cumpla nunca para que no se copie ningún registro.

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

Después podremos insertar en esta nueva tabla:

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

5.2. Modificación de registros a partir de una consulta

También podemos utilizar consultas para realizar modificaciones complejas.

Las consultas pueden formar parte de cualquiera de los elementos de la sentencia UPDATE.

En el siguiente ejemplo, la sentencia modifica el crédito de los usuarios que tienen una partida creada y cuyo estado es 1 (activada). El valor que se le 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_Creador FROM PARTIDAS WHERE Estado=1);
```

5.3. Supresión de registros a partir de una consulta

Se puede realizar borrado de registros utilizando consultas como parte de las tablas donde se hará la eliminación, o como parte de la condición que delimita la operación.

```
DELETE FROM (SELECT LOGIN FROM USUARIOS, UNEN WHERE CODIGO_USUARIO=LOGIN AND PROVINCIA='PALENCIA');
```

Se eliminarán determinados registros de la tabla USUARIOS y UNEN, en concreto aquellos que en la tabla UNEN registros asociados a algún usuario de Palencia.

En este caso no ha hecho falta utilizar ninguna condición WHERE en la sentencia. Otra manera, pero utilizando la cláusula WHERE:

```
DELETE FROM (SELECT LOGIN, PROVINCIA FROM USUARIOS, UNEN WHERE CODIGO_USUARIO=LOGIN) WHERE PROVINCIA='PALENCIA';
```

6. Transacciones

Una transacción es una unidad atómica de trabajo que puede contener una o más sentencias SQL. Las transacciones agrupan sentencias SQL de tal manera que a todas ellas se le aplica una operación COMMIT (confirmadas, aplicadas o guardadas en la BBDD),o bien a todas ellas se les aplica la acción ROLLBACK (deshacer operaciones que deberían hacer sobre la BBDD).

Un ejemplo sería el proceso de transferencia entre cuentas bancarias, donde Juan hace 100 euros de transferencia a María:

- Actualizar la cuenta de Juan restando 100.
- Registrar el movimiento de decremento de 100 en los movimientos de cuenta de Juan.
- Actualizar la cuenta de María incrementando 100.
- Registrar el movimiento de incremento en los movimientos de María.

Si hubiese un corte o caída en el sistema en el punto 2, los 100 euros no los tendría ni Juan, ni María. Para evitar estas situaciones se utilizan las transacciones, consideradas como una única operación. Hasta que las 4 operaciones no estén realizadas no se confirma (COMMIT) la operación y, si hay problemas en el proceso, se deshacen (ROLLBACK) las operaciones que dejaron a medias el proceso.

Mientras no se haga un COMMIT sobre una transacción, los resultados se pueden deshacer. Así, una sentencia del lenguaje de manipulación de datos (DML) no es permanente hasta que se realiza un COMMIT sobre la transacción en la que esté incluida o hasta que se ejecuta una operación DDL.

Las transacciones de oracle cumplen con las siguientes propiedades:

- **Atomicidad.** O se realizan todas las tareas o no se realiza ninguna. No hay una transacción parcial.
- **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 operaciones hasta que finaliza.
- **Durabilidad:** los cambios efectuados por las transacciones que han volcado sus modificaciones son permanentes.

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

- **COMMIT** (hacer permanentes cambios producidos por una transacción).
- **ROLLBACK** (deshacer cambios de una transacción desde que fue iniciada o desde un punto de restauración **(ROLLBACK TO SAVEPOINT)**. ROLLBACK termina la transacción, pero ROLLBACK TO SAVEPOINT no.

- Establecer un punto intermedio **(SAVEPOINT)** a partir del cual se puede deshacer la transacción.
- Indicar propiedades de una transacción **(SET TRANSACTION)**. Por ejemplo, elegir **READ ONLY** no permite 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)**. Se puede elegir **IMMEDIATE** o **DEFERRED.**

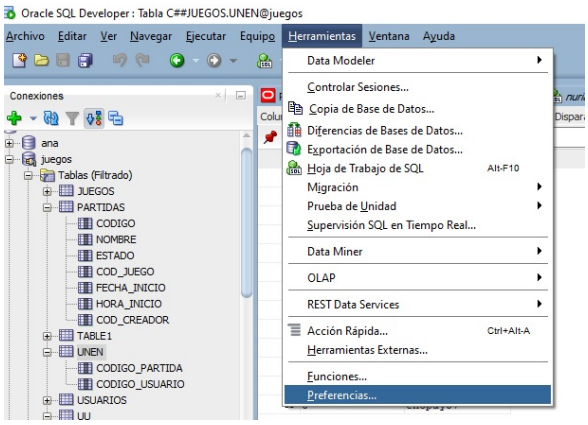
6.1. Hacer cambios permanentes

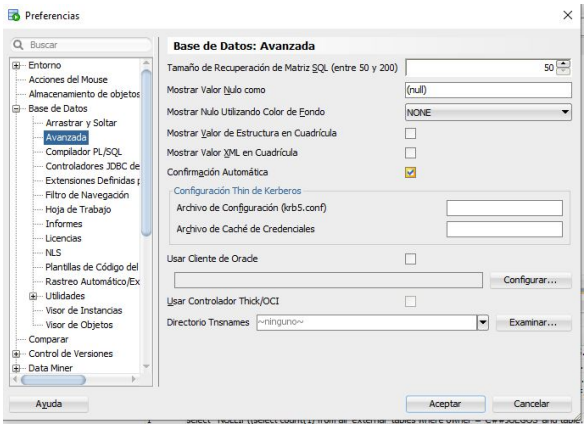
Una transacción comienza con la primera sentencia SQL ejecutable, para que los cambios producidos se hagan permanente se dispone de:

- Utilizar la sentencia COMMIT
- Ejecutar una sentencia DDL (como CREATE, DROP, RENAME O ALTER)
- Si el usuario cierra adecuadamente las aplicaciones de gestión de bases de datos de Oracle.

Para permitir ROLLBACK es necesario desactivar la variable de confirmación AUTOCOMMIT.

Para modificar el valor de la variable se puede hacer desde Herramientas/Preferencias, abriendo la opción Avanzada de Base de Datos y marcando o desmarcando la casilla de confirmación automática.





También se puede hacer desde SQLPlus con el comando show autocommit para mostrar el valor de la sesión actual y cambiar el valor con el comando SET.

6.2. Deshacer cambios

La sentencia ROLLBACK permite deshacer cambios efectuados y la da por finalizada. Se recomienda explícitamente finalizar las transacciones en las aplicaciones usando las sentencias COMMIT o ROLLBACK.

Si la transacción termina de forma anormal, los cambios que hasta el momento se hubiesen realizado serán deshechos automáticamente.

6.3. Deshacer cambios parcialmente

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

Si se usan puntos de restauración en una transacción larga, se tiene la opción de deshacer los cambios efectuados por la transacción a justo después del punto de restauración establecido. Si se produce un error no es necesario rehacer todas las sentencias, sino aquellas posteriores al punto de restauración.

```
SAVEPOINT nombre_punto_restauración;
```

```
ROLLBACK TO SAVEPOINT nombre_punto_restauración;
```

7. Políticas de boqueo

Un dato no puede ser modificado sin tener en cuenta que otros usuarios están modificando el dato al mismo tiempo. Las transacciones ejecutadas simultáneamente deben generar resultados consistentes. 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 visión consistente de los datos, incluyendo cambios visibles por transacciones del mismo usuario y las finalizadas de otros usuarios.

En una base de datos monousuario no es necesario bloquear, ya que solo modifica datos un usuario, pero en bases de datos multiusuario se debe proveer de un mecanismo para prevenir la modificación concurrente del mismo dato.

Los bloqueos permiten:

- **Consistencia.** Los datos consultados o modificados no pueden ser cambiados por otro hasta que el usuario haya finalizado la operación completa.
- **Integridad:** los datos y su estructura deben reflejar todos los cambios efectuados sobre ellos en el orden correcto.

7.1. Políticas de bloqueo

Los bloqueos afectan a la interacción de lectores y escritores. Un lector es quien consulta sobre un recurso y un escritor es quien realiza una modificación. Oracle resume las reglas del comportamiento sobre lectores y escritores de la siguiente manera.

- Un registro se bloquea solo cuando es modificado por un escritor, cuando una sentencia actualiza un registro, la transacción obtiene un bloqueo solo para ese registro.
- Un escritor de un registro bloquea a otro escritor concurrente del mismo registro.
- Un lector no bloquea a un escritor. La única excepción es SELECT... FOR UPDATE (una sentencia especial que bloquea el registro que está siendo consultado.
- Un escritor no bloquea a un lector. Cuando un registro está siendo modificado, la base de datos proporciona al lector una vista del registro si los cambios que se están realizando.

Hay dos mecanismos para el bloqueo en una base de datos:

- **Bloqueo pesimista.** Se bloquea un registro o una tabla inmediatamente, en cuanto se solicita el bloqueo. Con el bloqueo pesimista se garantiza que el registro será siempre actualizado.
- **Bloqueo optimista.** El acceso al registro o tabla solo se cierra en el momento en que los cambios realizados a ese registro se realizan en el disco. Esta situación solo es apropiada cuando hay menos posibilidad de que alguien necesite acceder al registro mientras está bloqueado.

7.2. Bloqueos compartidos y exclusivos

La base de datos utiliza dos tipos de bloqueos.

- **Bloqueo exclusivo.** 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 que puede modificar el recurso hasta que es liberado. Cualquier otra transacción debe esperar en cola.

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

- **Bloqueo compartido.** Permite que el recurso asociado sea compartido, dependiendo de la operación en la que se encuentra involucrado. Varios usuarios que leen datos pueden compartir datos realizando bloqueos compartidos.

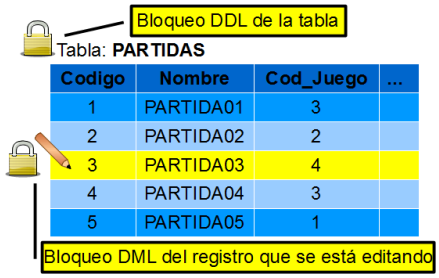
Por ejemplo, usando la sentencia SELECT... FOR UODATE, se obtiene un bloqueo exclusivo del registro y un bloqueo compartido de la tabla. El bloqueo del registro permite a otras sesiones modificar cualquier otro registro distinto al bloqueado. El bloqueo de la tabla previene que otras sesiones modifiquen la estructura de la tabla.

7.3. Bloqueos automáticos

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.

Los bloqueos que realiza Oracle se dividen en:



- **Bloqueos DML:** Protegen los datos, garantizando la integridad de los datos accedidos de forma concurrente por varios usuarios. Las sentencias DML: INSERT, UPDATE o DELETE realizan bloqueo exclusivo por las filas afectadas en su cláusula WHERE.
- **Bloqueos DDL**: Protegen la definición del esquema de un objeto mientras una operación DDL actúa sobre él. Se realizan de manera automática en cualquier transacción DDL. Los usuarios no pueden solicitar bloqueo explícito DDL.
- **Bloqueos del sistema:** Oracle usa varios tipos de bloqueo del sistema para proteger la bases de datos interna y estructuras de memoria.

7.4. Bloqueos manuales

Se pueden omitir los mecanismo de bloqueo por defecto de Oracle, usando bloqueos manuales:

- En aplicaciones que requieren consistencia en la consulta de datos a nivel transacciones o en lecturas repetitivas.
- En aplicaciones que requieren que una transacción tenga acceso exclusivo a un recurso con el fin de que no tenga que esperar a que otras terminen.

La cancelación de bloqueos se realiza a nivel de sesión o de transacción. A nivel sesión se realiza con ALTER SESSION, a nivel transacción las transacciones que incluyan las siguientes sentencias omiten el bloqueo por defecto:

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

Estos bloqueos terminan cuando termina la transacción.

7.5. Interbloqueos

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

```
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));
```

Si intentamos crear dos tablas con referencias cruzadas, dará error, porque cualquiera de las dos tablas necesita que la otra esté creada para poder crearse. Para solucionar este problema consideramos la transacción como DEFERRED, la comprobación de restricciones se hará en diferido.

Podemos crear ambas tablas sin especificar la FK y a continuación ALTER table para añadir la referencia a 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);
```

Si intentamos insertar un registro en cada una de las tablas dará error, ya que al insertar en la primera tabla aún no tenemos el dato de la segunda y viceversa.

Para solucionarlo podemos añadir 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 permite insertar 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;
```

Por último, sentencias para borrar las constraints y las tablas tras el ejemplo:

```
ALTER TABLE Huevo DROP CONSTRAINT fkRefGallina;

ALTER TABLE Gallina DROP CONSTRAINT fkRefHuevo;

DROP TABLE Huevo;
```

Mapa conceptual