

# TEMA 5: TRATAMIENTO DE DATOS

---

1.	INTRODUCCIÓN .....	1
2.	EDICIÓN DE LA INFORMACIÓN MEDIANTE HERRAMIENTAS GRÁFICAS .....	1
2.1.	INSERCIÓN DE REGISTROS .....	3
2.2.	MODIFICACIÓN DE REGISTROS.....	5
2.3.	BORRADO DE REGISTROS .....	7
3.	EDICIÓN DE INFORMACIÓN MEDIANTE SENTENCIAS SQL.....	9
3.1.	INSERCIÓN DE REGISTROS .....	10
3.2.	MODIFICACIÓN DE REGISTROS.....	13
3.3.	BORRADO DE REGISTROS .....	15
4.	INTEGRIDAD REFERENCIAL .....	16
4.1.	INTEGRIDAD EN ACTUALIZACIÓN Y SUPRESIÓN DE REGISTROS .....	17
4.2.	SUPRESIÓN EN CASCADA.....	19
5.	SUBCONSULTAS Y COMPOSICIONES EN ÓRDENES DE EDICIÓN .....	22
5.1.	INSERCIÓN DE REGISTROS A PARTIR DE UNA CONSULTA.....	22
5.2.	MODIFICACIÓN DE REGISTROS A PARTIR DE UNA CONSULTA .....	24
5.3.	SUPRESIÓN DE REGISTROS A PARTIR DE UNA CONSULTA .....	25
6.	TRANSACCIONES .....	27
6.1.	HACER CAMBIOS PERMANENTES .....	28
6.2.	DESHACER CAMBIOS .....	30
6.3.	DESHACER CAMBIOS PARCIALMENTE .....	31
7.	PROBLEMAS ASOCIADOS AL ACCESO SIMULTÁNEO A LOS DATOS.....	32
7.1.	POLÍTICAS DE BLOQUEO .....	33
7.2.	BLOQUEOS COMPARTIDOS Y EXCLUSIVOS .....	34
7.3.	BLOQUEOS AUTOMÁTICOS .....	35
7.4.	BLOQUEOS MANUALES .....	36
8.	ENLACES DE REFUERZO Y AMPLIACIÓN.....	38

# 1. INTRODUCCIÓN

---

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

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

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

Por otro lado, la ejecución de las aplicaciones puede fallar en un momento dado y eso no debe impedir que la información almacenada sea 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 afecten negativamente a las operaciones de los demás usuarios.

# 2. EDICIÓN DE LA INFORMACIÓN MEDIANTE HERRAMIENTAS GRÁFICAS

---

Los sistemas gestores de bases de datos como el de Oracle y también MySQL, 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.

## TEMA 5: Tratamiento de datos

### Oracle

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 11g Express Edition > Get Started.

### MySQL

MySQL Community Server dispone de la herramienta o cliente gráfico Workbench para poder trabajar en modo gráfico con MySQL. Pero existen otras herramientas o clientes gráficos para trabajar con MySQL, entre ellos vamos a destacar PhpMyAdmin y Navicat para MySQL, de los cuales te indicamos más abjo enlaces para su descarga y uso.

Para trabajar con Workbench, puedes acceder en Windows desde Inicio > Todos los programas > Base de Datos MySQL > Workbench. Realizas la conexión al servidor con tu usuario y contraseña y ya puedes empezar a trabajar.

Todos los ejemplos de MySQL los puedes probar en la BD *proyectosx*. Es importante que tengas clara la estructura o esquema de la BD (las tablas que la forman y las relaciones entre ellas) para entender bien todos los ejemplos propuestos.

Como ya has comprobado al ejercitar con los ejemplos de *SELECT*, en el conjunto de datos de prueba se ha procurado, abarcar los distintos casos que pueden darse en una situación del mundo real que estamos tratando. En especial:

- Departamentos sin trabajadores.
- Empleados con datos desconocidos (no son columnas obligatorias), o que aún no se han asignado a un departamento, marcados como NULL.
- Empleados que no están en ningún proyecto, que están en uno o en varios.
- Departamentos que no gestionan proyectos, o bien gestionan un proyecto o varios proyectos.
- Ciudades con un departamento o con varios.
- Existen columnas con valores NULL, indicando valor desconocido (fecha\_ingreso) o dato no necesario (cdjefe del empleado "A11").

### Recomendación

Para trabajar con **MySQL** desde un cliente gráfico, puedes utilizar además de Workbench los clientes gráfico Navicat y PhpMyAdmin.

Te indicamos para cada uno de ellos el enlace de descarga y un enlace a un tutorial de uso.

Enlace de descarga de **Navicat para MySQL** (uso de 30 días)

[Descarga de Navicat para MySQL \(30 días\)](#)

Tutorial de uso de Navicat para MySQL: <https://www.youtube.com/watch?v=Y84g6knDPcM>

## TEMA 5: Tratamiento de datos

La forma más sencilla de dejar configurado el cliente gráfico **PhpMyAdmin** para MySQL es descargar, en el caso de Windows, un paquete como **WampServer** que ya lleva incluido PhpMyAdmin, en concreto incluye: Apache, MySQL, Php y PhpMyAdmin listos para trabajar de manera conjunta.

Enlace de descarga de **WampServer** para Windows:

[Descarga de Wampser](#)

La instalación de Wampserver es muy sencilla, basta con ejecutar el archivo descargado y seguir el tutorial de instalación y configuración. Recuerda que instala su propio servidor MySQL, por lo que si ya tienes un MySQL instalado controlar que ambos no pueden estar iniciados a la vez escuchando por el mismo puerto.

Desde el panel de control que proporciona WampServer tendremos acceso, entre otros, a PhpMyAdmin.

Tutorial: <https://www.youtube.com/watch?v=9-B9M5hQnYM>

## 2.1. INSERCIÓN DE REGISTROS

### Oracle

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**.



Explorador de Objetos

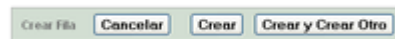
3. Seleccionar una tabla en la lista izquierda.

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.

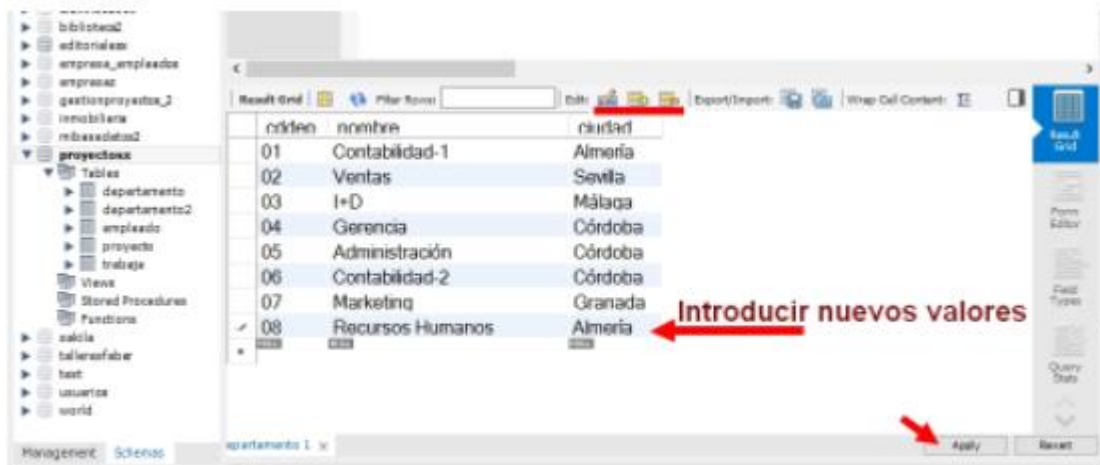
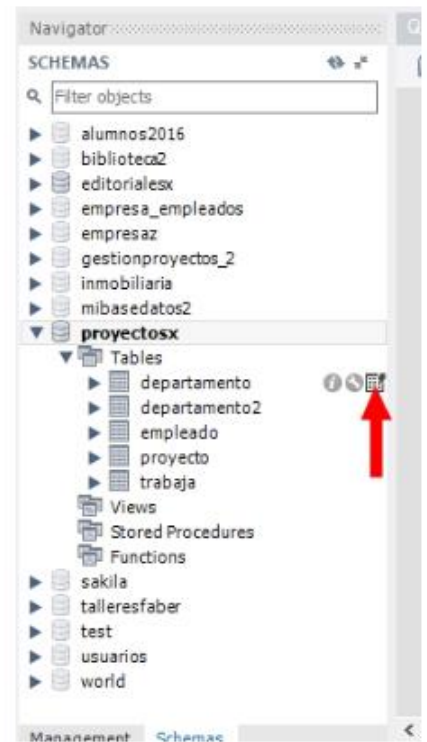
En caso de que se haya producido un **error** al intentar insertar los datos, habrá que comprobar el mensaje que se muestra, e intentar solucionar el problema. Por ejemplo, si se intenta introducir un texto en un campo de tipo numérico se obtendrá un error como el siguiente: "*error ORA-00984: columna no permitida aquí*", y no se habrá realizado ninguna operación de la inserción del nuevo registro.

## TEMA 5: Tratamiento de datos

### MySQL

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 MySQL, como es Workbench, debes seguir los siguientes **pasos**:

1. Ir a la **página inicial** de Workbench, si no te encuentras en ella y abrir conexión con el servidor.
2. **Seleccionar o poner en uso la base de datos** con la que vas a trabajar, haciendo doble clic sobre ella.
3. **Desplegar el contenedor de las tablas** de la base de datos, haciendo clic sobre el icono correspondiente.
4. **Seleccionar una tabla** en la lista izquierda.
5. En su **menú contextual** (un clic izquierdo) seleccionar la opción **Select Rows**. Aparecerán todas las filas o registros de la tabla con un encabezado de botones que nos permiten insertar, editar y eliminar filas, entre otras opciones.
6. Selecciona el botón o icono que aparece con el **símbolo +** y cuya acción es **Insertar nueva Fila (Insert new row)**
7. **Escribe los datos** correspondientes para cada campo del nuevo registro.



8. Haz clic en el botón **Apply** para guardar los datos introducidos, o si se desea seguir añadiendo otros registros, sigue introduciendo datos en la siguiente fila y al final es cuando debes pulsar el botón **Apply**. Se utilizará el botón **Cancelar (Revert)** si no se desea guardar los datos.
9. Si la fila o filas se han **añadido correctamente** se mostrará un mensaje indicando que la operación se ha realizado con éxito. En caso contrario, se mostrará un mensaje de error.

## TEMA 5: Tratamiento de datos

Observa que antes de aplicar o confirmar la operación, se muestra el SQL correspondiente a la sentencia que se va a ejecutar.

Permaneciendo en la ventana proporcionada por la opción *Select Rows* se podrá ver, 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: "*columna no permitida aquí*", y no se habrá realizado ninguna operación de la inserción del nuevo registro.

### Recomendación

En el siguiente enlace puedes abrir un videotutorial en el que se muestra cómo realizar la inserción de registros o filas en una base de datos MySQL utilizando Workbench. [Insertar filas en MySQL con Workbench.](#)

## 2.2. MODIFICACIÓN DE REGISTROS

### Oracle

Para modificar 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**.

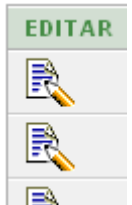


3. **Seleccionar una tabla** en la lista izquierda.

4. Seleccionar la pestaña **Datos**.

Tabla **Datos** Índices Modelo Rest

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.

### MySQL

La modificación de registros o valores de una fila permite cambiar los datos en las tablas que componen la base de datos. Para modificar registros o filas en tablas, utilizando las herramientas gráficas que ofrece MySQL, como es Workbench, se deben seguir los siguientes **pasos**:

1. Ir a la **página inicial** de Workbench, si no te encuentras en ella y abrir conexión con el servidor.
2. **Seleccionar o poner en uso la base de datos** con la que vas a trabajar, haciendo doble clic sobre ella.
3. **Desplegar el contenedor de las tablas** de la base de datos, haciendo clic sobre el icono correspondiente.
4. **Seleccionar una tabla** en la lista izquierda.
5. **En su menú contextual** (un clic izquierdo) seleccionar la opción **Select Rows**. Aparecerán todas las filas o registros de la tabla con un encabezado de botones que nos permiten insertar, editar o modificar y eliminar filas, entre otras opciones.
6. Sitúa el **cursor en la fila y columna cuyo valor vayas a modificar**.
7. Selecciona el botón o icono que aparece con un **lápiz de edición** y cuya acción es **modificar una Fila (Edit current row)**
8. **Escribe los nuevos datos** correspondientes para cada campo o columna que quieras modificar en esa fila o registro.
9. Hacer clic en el botón **Apply** para guardar los datos modificados, o si se desea seguir modificando otros registros, continúa moviéndote con el cursor a las correspondientes columnas de las filas a modificar y al final, es cuando debes pulsar el botón **Apply**. Se utilizará el botón **Cancelar** (Revert) si no se desea modificar los datos.
10. Si la fila o filas se han **modificado correctamente** se mostrará el mensaje correspondiente a que la operación se ha realizado con éxito. En caso contrario, se mostrará un mensaje de error.

Observa que antes de aplicar o confirmar la operación, se muestra el SQL correspondiente a la sentencia que se va a ejecutar.

Permaneciendo en la ventana proporcionada por la opción *Select Rows*, se podrá ver la **lista de los datos** contenidos en los registros que se han ido modificando.

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.

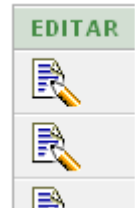
### Debes conocer

En el siguiente enlace puedes abrir un videotutorial en el que se muestra cómo realizar la modificación de filas en tablas de una base de datos MySQL, utilizando Workbench. [Modificar filas en MySQL con Workbench.](#)

## 2.3. BORRADO DE REGISTROS

### Oracle

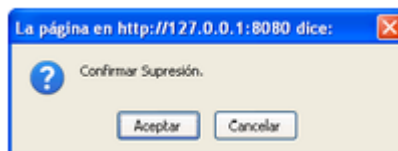
En el caso de que quieras eliminar un registro de una determinada tabla hay que seguir, en principio, los **mismos pasos** que se han comentado anteriormente para **editar un registro**. Es decir, una vez seleccionada la tabla, elegir la pestaña Datos y hacer clic en el botón *Editar* junto al registro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

**Suprimir**

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.

### MySQL

En el caso de que quieras eliminar un registro o fila de una determinada tabla, debes seguir los **mismos pasos** que se han comentado anteriormente para **editar o modificar un registro**. Es decir, una vez seleccionada la tabla, los pasos a seguir son los siguientes.

1. En su **menú contextual** (un clic izquierdo) selecciona la opción **Select Rows**. Aparecerán todas las filas o registros de la tabla con un encabezado de botones que nos permiten insertar, editar o modificar y eliminar filas, entre otras opciones.
2. Sitúa el **cursor en la fila que deseas eliminar**.
3. Selecciona el botón o icono que aparece con un **menos (-)** y cuya acción es **eliminar una Fila (Delete selected rows)**
4. Debes hacer clic en el botón **Apply** para guardar los cambios, o si se desea seguir eliminando otros registros, continúa posicionándote en las filas a eliminar y pulsando el *botón menos (-)* y al final, es



## TEMA 5: Tratamiento de datos

cuando debes pulsar el botón **Apply**. Se utilizará el botón *Cancelar (Revert)* si no se desea eliminar las filas.

- Puedes seleccionar el conjunto de filas a eliminar, manteniendo pulsada la tecla CTRL mientras haces clic en cada fila a eliminar, y posteriormente pulsar el *botón (-)* y luego *Apply*.
5. Si la fila o filas se han **eliminado correctamente** se mostrará el mensaje correspondiente a que la operación se ha realizado con éxito. En caso contrario, se mostrará un mensaje de error.

Observa que antes de aplicar o confirmar la operación, se muestra el SQL correspondiente a la sentencia que se va a ejecutar.

Permaneciendo en la ventana proporcionada por la opción *Select Rows* se podrá ver, la **lista de los datos** contenidos en los registros actuales y podrás comprobar que se han eliminado las filas indicadas.

Al igual que se comentó en la inserción y modificación de registros, al aceptar los cambios realizados en los datos, en este caso eliminación de filas, 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.

Ten en cuenta que 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: "*restricción de integridad referencial violada*"). 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.

Lo normal es, que si se viola la integridad referencial, el registro se deje sin eliminar, pues esa restricción de integridad proviene del diseño de la base de datos para que los datos almacenados sean en todo momento correctos.

### Debes conocer

En el siguiente enlace puedes abrir un videotutorial en el que se muestra cómo realizar la eliminación de filas en una tabla de una base de datos MySQL utilizando Workbench. [Eliminando filas en MySQL con Workbench.](#)

### 3. EDICIÓN DE INFORMACIÓN MEDIANTE SENTENCIAS SQL

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

Las sentencias DML en las que nos vamos a centrar para insertar, modificar y eliminar filas de una tabla son las siguientes:

- **INSERT** -- insertar o añadir filas
- **UPDATE** -- modificar o actualizar columnas de las filas
- **DELETE** -- eliminar filas

También veremos cómo utilizar estas sentencias junto con la sentencia **SELECT**, para ampliar sus posibilidades.

#### Oracle

Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web *Application Express* de Oracle utilizando el botón "SQL Workshop" en la página de inicio (Home), y *seleccionando el botón* -> "SQL commands".

#### MySQL

Las sentencias SQL que se verán a continuación **pueden ser ejecutadas desde:**

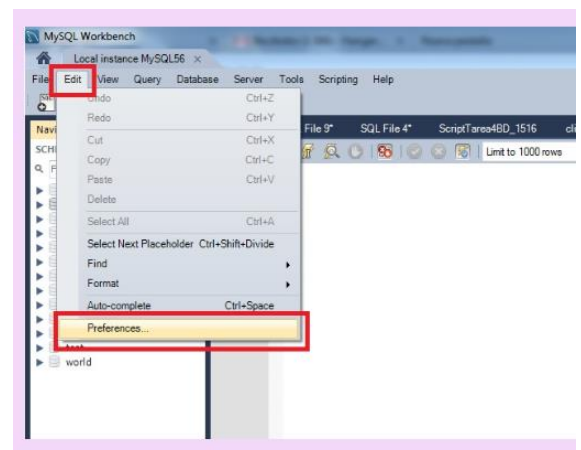
1. La ventana de SQL de Workbench, tal y como has hecho para las consultas SQL en la Unidad anterior.
2. La línea de comandos o cliente en modo texto, visto también en la unidad anterior para realizar consultas SQL.

#### Debes conocer

Por defecto, MySQL Workbench tiene activa la opción "SQL\_SAFE\_UPDATES" que al estar marcada no permite ejecutar las sentencias de actualización (UPDATE) y borrado (DELETE).

Para poder ejecutar este tipo de sentencias hay que desactivar esta opción. Para ello, una vez que establezcamos una conexión y entremos en MySQL Workbench, seleccionamos sobre el menú **EDIT** la opción **PREFERENCES**

y una vez dentro, sobre "SQL Editor" desmarcamos la opción "Safe Updates"



### 3.1. INSERCIÓN DE REGISTROS

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

Existen dos formas básicas de la sentencia **INSERT** :

- Añadir una fila indicando explícitamente los valores que deben tomar las columnas, o
- Extraer las filas de una tabla ya existente y añadirlas a otra tabla (utilizando consultas **SELECT** con **INSERT**).

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 **ASISTENTE** en el que se tienen todos los datos disponibles:

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES ('AS0015', 'Julia', 'Hernández', 'Sáez', 'M', '11/10/1992', 'BK Programación');
```

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

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, sexo, fechaNac) VALUES ('AS0016', 'María', 'Durán', 'M', '03/30/1979');
```

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 "empresa" pero sí se especifica un valor para dicho campo:

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, sexo, fechaNac) VALUES ('AS0017', 'Lucas', 'López', 'H', '12/22/1989', 'Big Soft');
```

## TEMA 5: Tratamiento de datos

Se obtiene el siguiente error:

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

Los mismos ejemplos vistos anteriormente para Oracle son válidos para MySQL.

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

### MySQL

En el siguiente ejemplo se inserta en la tabla **departamento**, el departamento con código "10", llamado "I+D" y situado en la ciudad de "Sevilla", escribiremos:

```
INSERT INTO departamento (cddep,nombre,ciudad) VALUES ('10','I+D', 'Sevilla');  
  
o bien:  
  
INSERT INTO departamento VALUES ('10','I+D', 'Sevilla');
```

Observa que en este caso podemos omitir el nombre de las columnas, porque estamos proporcionando valores para todas ellas y además en el orden en el que están definidas

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

```
INSERT INTO departamento (cddep,nombre) VALUES ('11','Ventas');
```

Al hacer un **INSERT** en el que no se especifiquen los valores de todos los campos, en la columna que se omita, siempre que no sea obligatoria, se asignará bien el valor indicado por defecto o bien **NULL**..

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 'ciudad' pero sí se especifica un valor para dicho campo:

```
INSERT INTO departamento (cddep,nombre) VALUES ('11','Ventas', 'Almeria');
```

Se obtiene un error diciendo que se están proporcionando más valores de los indicados.

¿Podemos insertar varias filas con una única sentencia **INSERT**? Si se puede hacer, y es lo que se conoce como **INSERT extendido**.

Un **INSERT extendido**, consiste en una sola sentencia **INSERT** que permite insertar varias filas a la tabla de manera más eficiente.

Por ejemplo, podemos insertar los datos de tres departamentos de la siguiente forma:

```
INSERT departamento VALUES ('20','DEPAR20','ALMERÍA'),  
('21','DEPART21','ALMERIA'), ('22','DEPART22','GRANADA');
```

## TEMA 5: Tratamiento de datos

Otra cuestión a tener en cuenta, cómo insertamos filas en las que la clave primaria es un AUTONUMÉRICO. A continuación lo vemos.

Para **insertar valores AUTONUMÉRICOS**, podemos proceder de las siguientes formas:

- **Omitimos el nombre de la columna autonumérica.**

Por ejemplo, si una tabla de nombre 'alumnos' tiene como clave primaria una columna autonumérica el 'idalumnos' y queremos insertar el siguiente alumno, podemos hacerlo de la siguiente forma:

```
INSERT INTO alumnos (nombre, apellidos, idcurso)
VALUES('Daniel', 'Ro', 2);
```

Siendo la columna 'idalumnos' un autonumérico, su valor vendrá dado por el último alumno más uno.

- **Si no omitimos esa columna.** En su lugar ponemos un 0 y el sistema calculará automáticamente el valor numérico correspondiente (damos valor a todas las columnas).

Por ejemplo, si una tabla de nombre alumnos tiene como columna autonumérica el idalumnos y queremos insertar el siguiente alumno, podemos hacerlo de las siguientes formas:

```
INSERT INTO alumnos (idalumnos, nombre, apellidos, idcurso) VALUES (0,'Manuel','Bellido', 2);
```

o bien, como estamos proporcionando todos los datos:

```
INSERT INTO alumnos VALUES (0,'David','Segura', 2);
```

### Insertar filas con REPLACE

Funciona igual que INSERT excepto por el hecho de que si el valor de la clave primaria del registro a insertar coincide con un valor existente, éste se borrará para insertar el nuevo. Por ejemplo, si ya existe el departamento de código '09' (misma clave primaria), con la siguiente sentencia, lo sustituiría por los nuevos valores indicados tras VALUES.

```
REPLACE INTO departamento VALUES ('09', 'InformáticaReplace', 'Almería');
```

## 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.

### Oracle

Por ejemplo, si se desea poner a 20 el precio de cada conferencia:

```
UPDATE CONFERENCIA SET precio = 20;
```

En este otro ejemplo puedes ver la actualización de dos campos, poniendo a 20 el precio de las conferencias y borrando la información del campo *tema* de todas las conferencias:

```
UPDATE CONFERENCIA SET precio = 20, tema = NULL;
```

Para que los cambios afecten a determinados registros hay que especificar una condición. Por ejemplo, si se quiere cambiar el precio de todas las conferencias del turno de tarde, estableciendo el valor 30:

```
UPDATE CONFERENCIA SET Credito = 30 WHERE turno = 'T';
```

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:

```
3 fila(s) actualizada(s).
```

**Los mismos ejemplos vistos anteriormente para Oracle son válidos para MySQL.**

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

## TEMA 5: Tratamiento de datos



Por ejemplo, si ejecutamos la siguiente sentencia, en la que no incluimos **WHERE**, estaríamos cambiando la ciudad de todos los departamentos:

```
UPDATE departamento  
SET ciudad='Córdoba';
```

Se podrá cambiar una o más filas con una sola sentencia **UPDATE**, esto dependerá del criterio establecido en la cláusula **WHERE**.

Por ejemplo, cambiar la ciudad donde se ubica el departamento "09", y hacer que sea "Córdoba", escribiremos:

```
UPDATE departamento  
SET ciudad='Córdoba'  
WHERE cddep='09';
```

En este caso sólo se modifica el registro de clave primaria '09'. Si no existe, no se ejecutará, avisando de que no se ha modificado ninguna fila.

Por ejemplo, si se desea incrementar en 5, las horas de trabajo dedicadas al proyecto "AEE" por todos los empleados que trabajan en él, deberíamos escribir:

```
UPDATE trabaja  
SET nhoras=nhoras+5  
WHERE cdpro='AEE';
```

En este caso se pueden modificar varias filas a la vez, todas aquellas en las que figure el proyecto 'AEE'. Si no existe, no se ejecutará nada indicando con un mensaje que se han actualizado 0 filas.

En este otro ejemplo puedes ver la actualización de dos campos del departamento '22', poniendo a 'I+D' el nombre del departamento y a 'Almeria' el nombre de su ciudad.

```
UPDATE departamento SET nombre = 'I+D', ciudad = 'Almeria'  
WHERE cddep='22';
```

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:

```
5 fila(s) actualizada(s).
```

En la cláusula **WHERE** de un **UPDATE** se puede incluir una subconsulta **SELECT**, para indicar qué registro o registros deben ser modificados, así como en la **SET** para indicar el nuevo valor.

### Para saber más

En el siguiente enlace *MySQL con Clase* puedes ver más ejemplos del uso de **UPDATE** para modificar filas o registros de una tabla. [Ejemplos de UPDATE en MSQl](#)

### 3.3. BORRADO DE REGISTROS

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

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

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

Al igual que hemos visto en las sentencias anteriores, nombre\_tabla hace referencia a la tabla sobre la que se hará la operación, en este caso de borrado. Se puede observar que la cláusula **WHERE** es opcional. Si no se indica, debes tener muy claro que se borrará todo el contenido de la tabla, aunque la tabla seguirá existiendo con la estructura que tenía hasta el momento.

#### Oracle

Por ejemplo, si usas la siguiente sentencia, borrarás todos los registros de la tabla *PONENTE*:

```
DELETE FROM PONENTE;
```

Para ver un ejemplo de uso de la sentencia **DELETE** en la que se indique una condición, supongamos que queremos eliminar todos los ponentes cuya especialidad es Programación:

```
DELETE FROM PONENTE WHERE especialidad = 'Programación';
```

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.

```
2 fila(s) suprimida(s)
```

**Los mismos ejemplos vistos anteriormente para Oracle son válidos para MySQL.**

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

#### MySQL

Por ejemplo, si usas la siguiente sentencia, borrarás todos los registros de la tabla *empleado*:

```
DELETE FROM empleado;
```

Para ver un ejemplo de uso de la sentencia **DELETE** en la que se indique una condición, supongamos por ejemplo que se desea eliminar los empleados que pertenecen al departamento "22", escribiríamos:

```
DELETE FROM empleado
WHERE cddep='22';
```

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, por ejemplo.

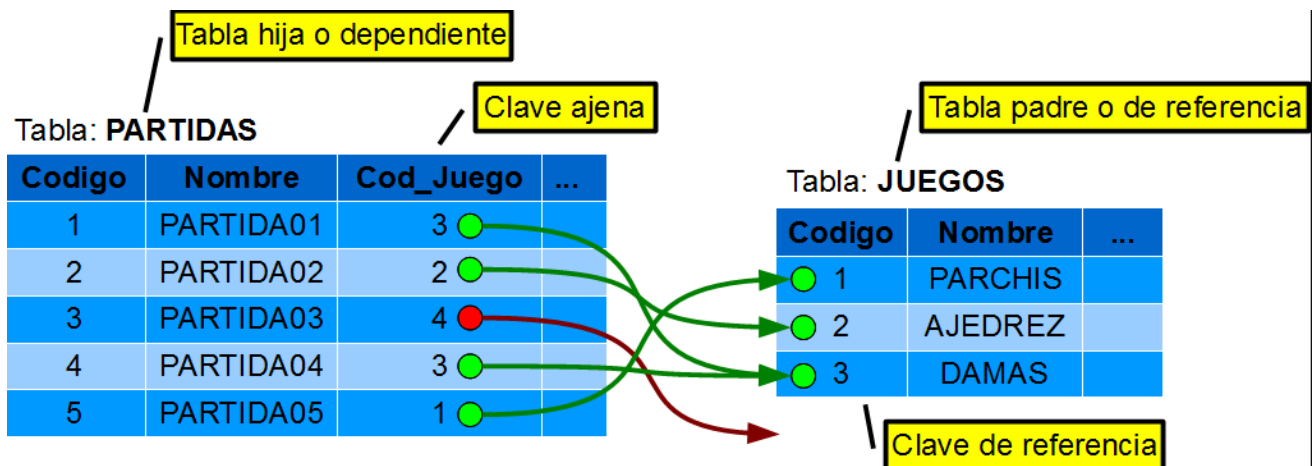
```
10 fila(s) suprimida(s).
```



## 4. INTEGRIDAD REFERENCIAL

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

Por ejemplo, supongamos que tenemos las tablas *PARTIDAS* y *JUEGOS*. 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*.

## TEMA 5: Tratamiento de datos

### Debes conocer

Recuerda que MySQL permite varios tipos o motores de almacenamiento. Los dos tipos de almacenamiento más populares son:

- **MyISAM:** No gestiona integridad referencial, ni transacciones. Los datos se almacenan en un formato independiente, lo que permite pasar tablas entre distintas plataformas. Este tipo de almacenamiento es especialmente útil cuando los datos no cambian con frecuencia y la operación más utilizada es la de lectura.
- **InnoDB:** Si gestiona integridad referencial mediante claves foráneas o ajenas, y soporta control de transacciones, y bloqueo por registro. Para lograr estos objetivos se sacrifica un poco la velocidad de la que MySQL presume. MySQL bloquea las tablas de tipo MyISAM cuando se realiza una inserción o una actualización en un registro hasta que la operación se termine. Sin embargo, si las tablas son tipo InnoDB, el bloqueo sólo se realiza en el registro que se esté modificando, lo que permite que más usuarios utilicen de forma simultánea la base de datos. Realmente se diseñó para obtener el máximo rendimiento al procesar grandes volúmenes de datos.

En el siguiente enlace encontrarás más información sobre dos motores de almacenamiento de MySQL, el motor InnoDB y MyISAM. [Motor MyISAM y motor InnoDB](#)

### Para saber más

Descripción del concepto de integridad referencial con ejemplo. [Integridad Referencial.](#)

## 4.1. INTEGRIDAD EN ACTUALIZACIÓN Y SUPRESIÓN DE REGISTROS

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

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

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

Tabla: **PARTIDAS**

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

Tabla: **JUEGOS**

Codigo	Nombre	...
1	PARCHIS	
2	AJEDREZ	
3	DAMAS	

## TEMA 5: Tratamiento de datos

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**.

### Modificaciones y restricciones de Integridad

1) Si se intenta hacer un cambio que entra en conflicto con alguna de las reglas integridad de la tabla, se producirá un error y el cambio no será llevado a efecto. Por ejemplo, si intentamos modificar el valor de la clave primaria de una fila por un valor que exista previamente (recuerda que no pueden existir dos filas con el mismo valor de clave primaria), la operación no se efectuará. Esto lo gestiona automáticamente el SGBDR, asegurando la integridad de los datos de la base de datos.

2) Si modificamos el valor de una columna que forma parte de una restricción de clave foránea o ajena (**FOREIGN KEY**) el SGBDR que soporta integridad referencial (caso de MySQL motor InnoDB) actuará según lo indicado en **ON UPDATE (RESTRICT, NO ACTION, CASCADE, SET NULL)**.

### Eliminación de filas y Reglas de Integridad

Por supuesto el borrado de filas no debe dejar la base de datos en un estado de inconsistencia. Esto quiere decir que el SGBDR que soporte Integridad referencial (caso de motor InnoDB en MySQL) comprobará las restricciones de clave ajena, **FOREIGN KEY**, definidas y actuará según lo indicado en **ON DELETE (RESTRICT, NO ACTION, CASCADE, SET NULL)**.

Veamos algún ejemplo para que practiques con tu base de datos **proyectosx** en MySQL



Por ejemplo, si intentamos eliminar un departamento que es referenciado en la tabla de empleados y en la tabla de proyectos, es decir, un departamento donde trabaja al menos un empleado o un departamento con proyecto asignado, el SGBDR abortará la operación de borrado. Sólo se podrán eliminar departamentos en los que no trabaje ningún empleado y no tenga proyecto asignado. Esto es así porque se especificó **ON DELETE RESTRICT** al definir la clave ajena *cddep* de la tabla *empleado* y en la tabla *proyecto*.

Ejemplo: Eliminar el departamento '02'

```
DELETE FROM departamento
WHERE cddep='02';
```

No eliminará al departamento '02' si existen filas en las tablas hijas (empleado o proyecto) cuyo valor de columna cddep sea "02".

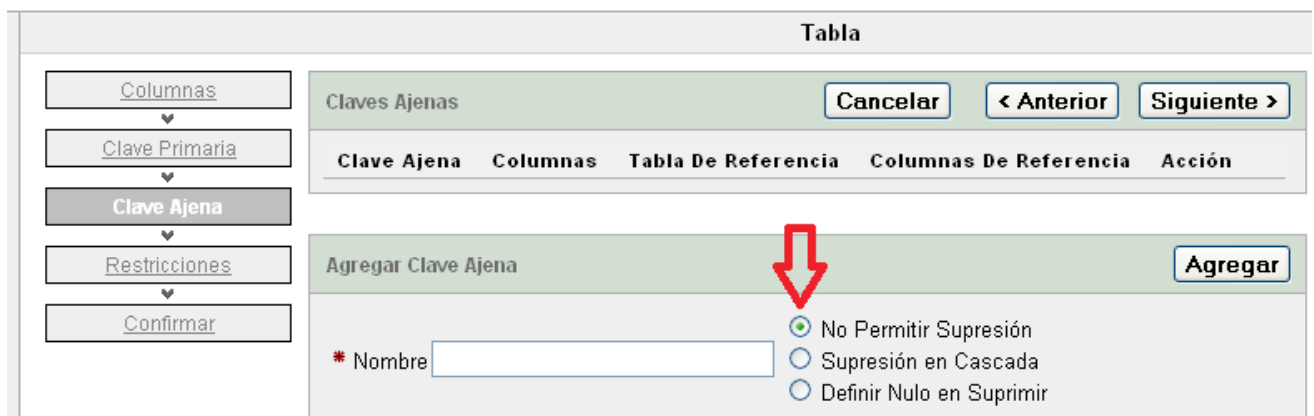
#### Para saber más

En el siguiente enlace encontrarás más información y ejemplos sobre la actualización y eliminación de registros en MySQL, cuando se utiliza el motor InnoDB soportando Integridad Referencial. [Ejemplos de Integridad Referencial en MySQL](#)

## 4.2. SUPRESIÓN EN CASCADA

### Oracle

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.



Tabla

Claves Ajenas

Cancelar < Anterior Siguiente >

Clave Ajena	Columnas	Tabla De Referencia	Columnas De Referencia	Acción
Agregar Clave Ajena				
* Nombre		<input checked="" type="radio"/> No Permitir Supresión <input type="radio"/> Supresión en Cascada <input type="radio"/> Definir Nulo en Suprimir		

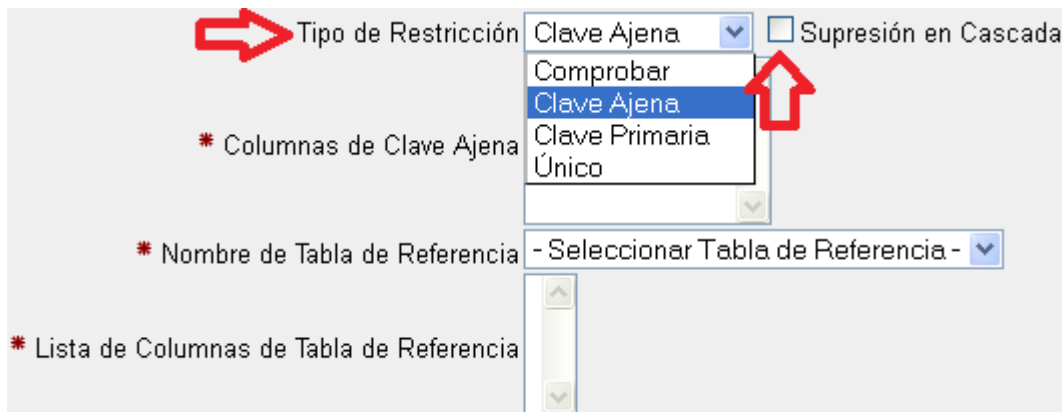
Agregar

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.

Tabla	Datos	Índices	Modelo	Restricciones	Permisos	Estadísticas	Valores por Defecto d
Crear	Borrar	Activar	Desactivar				
Restricción	Tipo	Tabla	Condición De Búsqueda	Suprimir Regla	Estado		
PARTIDAS_CON	R	PARTIDAS	-	NO ACTION	ENABLED		

## TEMA 5: Tratamiento de datos

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.



Tipo de Restricción: Clave Ajena (seleccionada), Comprobar, Clave Ajena (destacada), Clave Primaria, Único. ☒ Supresión en Cascada.

\* Columnas de Clave Ajena: [ ]

\* Nombre de Tabla de Referencia: - Seleccionar Tabla de Referencia -

\* Lista de Columnas de Tabla de Referencia: [ ]

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**).

### MySQL

Los mismos ejemplos vistos anteriormente para Oracle con SQL son válidos para MySQL.

Vamos a ver otros ejemplos en MySQL para que practiques con tu base de datos **proyectosx** y el cliente gráfico de MySQL Workbench.

Las reglas de integridad referidas al comportamiento de la base de datos en las operaciones de modificación y eliminación de registros se pueden establecer mediante:

- Sentencias SQL.
- Una herramienta gráfica como Workbench.

Con **sentencias SQL**, se pueden establecer las restricciones de integridad en el momento de crear la tabla (**CREATE TABLE**) o modificar su estructura (**ALTER TABLE**):

Recuerda que por defecto, si no se indica nada, la restricción será equivalente a haber puesto **RESTRICT** o **NO ACTION**, para eliminaciones y modificaciones restrictivas, utilizar la opción **CASCADE** para hacer la eliminación o modificación en cascada, o bien **SET NULL** si se prefiere poner a valor nulo la clave ajena en eliminaciones o modificaciones.

Por ejemplo, la siguiente restricción en la tabla empleado, indica que las modificaciones se realicen en cascada y las eliminaciones sean restrictivas:

## TEMA 5: Tratamiento de datos

```
CONSTRAINT emple_dep FOREIGN KEY (cddep) REFERENCES departamento (cddep)
ON UPDATE CASCADE
```

Por tanto, si actualizamos el valor de la clave primaria *cddep* en la tabla *departamento* el cambio se transmitirá a todos los empleados (también los proyectos) que referenciaban a ese departamento. Esto es así porque en la creación de la esas tablas de la base de datos se estableció la política de Integridad referencial (*ON UPDATE CASCADE*).

Por ejemplo, si modificas el código del departamento '02' por el valor '42' mediante la sentencia:

```
UPDATE departamento
SET cddep='42'
WHERE cddep='02';
```

comprobarás que se modifica el valor de la columna *cddep* del departamento "02" por "42", en la tabla departamento, y también en las tablas hijas proyecto y empleado. El cambio se transmite en cascada tal y como se ha indicado en la reglas FOREIGN KEY correspondientes.

Haz la comprobación y cambia de nuevo al valor '02'.

**Mediante una herramienta gráfica como Workbench**, se pueden indicar también estas restricciones de forma visual. Recuerda que para ello los pasos a seguir, una vez estamos en la ventana de creación o modificación de la tabla, son:

1. Selecciona la pestaña *Foreign Keys* de la ventana de edición de la tabla.
2. En la siguiente ventana, selecciona la clave ajena de la lista que aparece bajo el encabezado *Foreign Key Name*.
3. En esa misma ventana, a la derecha, bajo el encabezado *Foreign Key Options* puedes desplegar las listas relativas a *On Update* y *On Delete*, para elegir la opción deseada: *RESTRICT*, *CASCADE*, *SET NULL*, *NO ACTION*.
4. Pulsa el botón *Apply*, para confirmar los cambios, o si quieres cancelar la operación, pulsa el botón *Revert*.

### Recomendación

En el siguiente enlace puedes consultar un tutorial para ver cómo realizar estas restricciones de Integridad tanto con sentencias SQL como con la herramienta gráfica Workbench. [Restricciones Integridad en SQL y con Workbench](#)

## 5. SUBCONSULTAS Y COMPOSICIONES EN ÓRDENES DE EDICIÓN

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

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

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

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

### Para saber más

**Oracle:** En este manual pueden encontrar una sección sobre las funciones agregadas y subconsultas (módulo 3). También puedes ver ejemplos en la parte final. [Resumen de SQL con ejemplos, incluyendo material sobre subconsultas.](#)

**MySQL:** En el siguiente enlace dispones de un tutorial en el que puedes repasar las consultas y subconsultas en SQL, entre otras informaciones. [Resumen de SQL con ejemplos de subconsultas](#)

### 5.1. INSERCIÓN DE REGISTROS A PARTIR DE UNA CONSULTA

#### Oracle

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

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES
('AS0015','Julia', 'Hernández','Sáez','M', '11/10/1992','BK Programación');
```

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 codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa FROM ASISTENTE VALUES ('AS0015','Julia',
'Hernández','Sáez','M', '11/10/1992','BK Programación');
```

## TEMA 5: Tratamiento de datos

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 *ASISTENTES\_SUPLENTE*s con la misma estructura que la tabla *ASISTENTE*. Si queremos insertar en esa tabla todos los usuarios que tienen el campo empresa a null:

```
INSERT INTO ASISTENTES_SUPLENTE SELECT * FROM ASISTENTE WHERE empresa = null;
```

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

**Los mismos ejemplos vistos anteriormente para Oracle con SQL son válidos para MySQL.**

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

### MySQL

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

```
INSERT INTO departamento (cddep,nombre,ciudad) VALUES ('07','I+D', 'Sevilla');
```

En este caso, se indican de forma explícita los valores a insertar en la fila.

¿Y si quiero insertar filas o valores que provienen de otra tabla? ¿Cómo lo podemos hacer?

Se puede combinar un INSERT con una SELECT de la siguiente forma:

```
INSERT [INTO] <tabla> [(<columna>,...)]  
SELECT ...
```

En este caso, los valores a insertar en una tabla no se indican explícitamente, sino que se obtienen a partir de una consulta (SELECT) de las filas de otra tabla.

Por ejemplo, si quisiéramos añadir todos los empleados del departamento “03” al proyecto con código “ECS”, suponiendo que existe tal proyecto en la tabla de proyectos, la sentencia a utilizar sería:

```
INSERT INTO trabaja (cdemp,cdpro)  
SELECT cdemp,'ECS'  
FROM empleado  
WHERE cddep='03';
```



## TEMA 5: Tratamiento de datos

Puedes comprobar que se han insertado esos datos mediante la consulta:

```
SELECT * FROM trabaja  
WHERE cdpro='ECS';
```

Observa que en el INSERT no se ha indicado ningún valor para la columna *nhoras* y que por tanto se le ha asignado el valor por defecto (cero) que se indicó en la definición de esa columna mediante la cláusula DEFAULT.

Veamos otro ejemplo: si quisiéramos añadir los empleados que no tienen proyecto asignado, al proyecto con código "ECS", suponiendo que existe tal proyecto en la tabla de proyectos, la sentencia a utilizar sería:

```
INSERT INTO trabaja (cdemp,cdpro)  
SELECT cdemp,'ECS'  
FROM empleado  
WHERE cdemp NOT IN (SELECT DISTINCT cdemp  
FROM trabaja);
```

## 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**.

### Oracle

Por ejemplo, la siguiente sentencia modifica el precio de las conferencias que se celebren en salas cuya capacidad sea mayor que 200. El precio que se les asignará será el de la conferencia que tenga mayor precio.

```
UPDATE CONFERENCIA SET precio = (SELECT MAX(precio) FROM CONFERENCIA) WHERE sala IN  
(SELECT nombre FROM SALA WHERE capacidad >200);
```

### MySQL

Los mismos ejemplos vistos anteriormente para Oracle con SQL son válidos para MySQL.

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

Por ejemplo, si se quiere aumentar en 2 las horas trabajadas, pero sólo a los empleados que trabajan en proyectos que controla el departamento '04', la sentencia sería:

```
UPDATE trabaja  
SET nhoras=nhoras+2  
WHERE cdpro IN (SELECT cdpro  
FROM proyecto  
WHERE cddep='04');
```

## TEMA 5: Tratamiento de datos

Vemos otro ejemplo, si queremos asignar a los empleados sin departamento, al departamento o uno de los departamentos en cuyos proyectos se ha trabajado cero horas, la sentencia sería:

```
UPDATE empleado
SET cddep = (SELECT p.cddep FROM proyecto p
            INNER JOIN trabaja t ON t.cdpro=p.cdpro
            GROUP BY t.cdpro
            HAVING SUM(t.nhoras)=0
            ORDER BY cddep
            LIMIT 1)
WHERE cddep IS NULL;
```

**IMPORTANTE:** En caso de que para poder realizar la actualización de una tabla necesitemos consultar la misma tabla que se está actualizando, MySQL daría un error ya que la tabla que se está actualizando estaría bloqueada y no permitiría realizar consultas.

**Por ejemplo,** si quisiéramos aumentar el salario en 100 euros a aquellos empleados cuyo salario medio sea menor a la media de los salarios de los empleados, la sentencia de actualización tendría que ser:

```
UPDATE EMPLEADO SET SALARIO=SALARIO+100 WHERE SALARIO <=
(SELECT AVG(SALARIO) FROM EMPLEADO);
```

Pero esta sentencia de actualización daría error porque en la subconsulta aparece la tabla EMPLEADO que es la misma que vamos a actualizar.

Entonces, **¿no se podrían realizar este tipo de sentencias?** Sí se podría y la forma sería creando una tabla temporal de la tabla que queremos consultar. Por ejemplo, en el caso anterior se podría poner:

```
UPDATE EMPLEADO SET SALARIO=SALARIO+10 WHERE SALARIO <=
(SELECT AVG(SALARIO) FROM (SELECT * FROM EMPLEADO) AS EMP);
```

### 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.

#### Oracle

Por ejemplo, si se ejecuta la siguiente sentencia:

```
DELETE FROM (SELECT * FROM ASISTENTE WHERE Empresa='Bigsoft' AND sexo='H');
```

## TEMA 5: Tratamiento de datos

El resultado es que se eliminan determinados registros de la *tabla ASISTENTE*, en concreto, aquellos asistentes varones que pertenezcan a la empresa 'Bigsoft'.

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 ASISTENTE WHERE Empresa='Bigsoft') WHERE sexo='H';
```

Los mismos ejemplos vistos anteriormente para Oracle con SQL son válidos para MySQL.

Vamos a ver otros ejemplos en MySQL para que practiques tu base de datos **proyectosx**.

### MySQL

Por ejemplo, para eliminar los proyectos en los que no trabaje ningún empleado, redactaríamos la siguiente sentencia:

```
DELETE FROM proyecto  
WHERE cdpro NOT IN (SELECT cdpro FROM trabaja);
```

Vemos otro ejemplo, para eliminar los empleados que trabajan en proyectos que controla el departamento '02' escribiríamos:

```
DELETE FROM empleado  
WHERE cdemp IN (SELECT t.cdemp FROM trabaja t  
                INNER JOIN proyecto p ON p.cdpro=t.cdpro  
                WHERE p.cddep='02');
```

**IMPORTANTE:** Al igual que en la modificación de registros, si la sentencia de borrado incluye una subconsulta sobre la misma tabla donde se van a borrar los registros, MySQL daría un error ya que la tabla estaría bloqueada y no se permite realizar consultas.

Al igual que antes, sí se podrían realizar estas sentencias de borrado utilizando una tabla temporal.

**Por ejemplo**, si quisiéramos eliminar aquellos empleados cuyo salario sea mayor que la media de los salarios de los empleados, la sentencia de borrado tendría que ser:

```
DELETE FROM EMPLEADO WHERE SALARIO >  
(SELECT AVG(SALARIO) FROM (SELECT * FROM EMPLEADO) AS EMP1);
```

## 6. TRANSACCIONES

---

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

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

Las transacciones de Oracle y MySQL cumplen con las propiedades básicas de las transacciones en base de datos (**ACID**):

- **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.

MySQL soporta transacciones dependiendo del tipo de tabla o motor utilizado. Las tablas que permiten transacciones son de tipo *InnoDB*.

Las tablas que soportan transacciones son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor, ya que las instrucciones se ejecutan o no en su totalidad. Por otra parte, las transacciones pueden aumentar el tiempo de proceso de instrucciones.

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*.

### MySQL

Los pasos para usar transacciones en MySQL son:

1. Iniciar una transacción con **START TRANSACTION** o **BEGIN**
2. Actualizar datos (insertar, eliminar o modificar registros).
3. Confirmar la transacción con **COMMIT**. Los cambios hechos serán permanentes.
4. Deshacer la transacción con **ROLLBACK** si ha habido algún problema con la ejecución de alguna sentencia de la transacción y ésta queda a medias. (Una cancelación, una excepción, un fallo en sistema, etc.). De esta forma, no se graba ningún cambio en la BD.

En tablas *InnoDB* toda la actividad del usuario se produce dentro de una transacción.

Si está activado el modo de ejecución automático (*autocommit*) cada sentencia SQL conforma una transacción individual por sí misma.

MySQL siempre comienza una nueva conexión con la ejecución automática habilitada, es decir, con la variable **AUTOCOMMIT= ON** (o 1), por lo que cada instrucción individual SQL se trata como una transacción en sí misma, y se confirmará en cuanto que la ejecución termine.

Puedes comprobar el estado de esta variable mediante la orden:

```
mysql> show variables like '%autocommit%';
```

Se puede cambiar el valor de la variable **AUTOCOMMIT** a **OFF** (o 0), ejecutando:

```
mysql> SET autocommit = OFF;
```

## TEMA 5: Tratamiento de datos

Recuerda que todas las órdenes que puedes dar en modo comando (prompt mysql>), las puedes dar desde la ventana de SQL de Workbench.

Si el modo de ejecución automática se deshabilitó con `SET AUTOCOMMIT=0`, entonces puede considerarse que un usuario siempre tiene una transacción abierta.

Una sentencia COMMIT o ROLLBACK termina la transacción vigente y comienza con una nueva.

Ambas sentencias liberan todos los bloqueos InnoDB que se establecieron durante la transacción vigente.

Si la conexión tiene la ejecución automática habilitada, el usuario puede igualmente llevar a cabo una transacción

con varias sentencias si la comienza explícitamente con START TRANSACTION o BEGIN, y la termina con COMMIT o ROLLBACK.

A continuación tienes un ejemplo que te ayudará a ver de forma práctica el manejo de transacciones.

EJEMPLO:

1. Iniciamos transacción
2. Incrementamos en 50 las horas dedicadas por los empleados que trabajan en el proyecto 'DAG'
3. Finalizamos la transacción, confirmando con COMMIT

```
START TRANSACTION;  
UPDATE trabaja  
SET nhoras=nhoras+50  
WHERE cdpro='DAG';  
COMMIT;
```

Una vez finalizada la transacción con COMMIT, los cambios son permanentes y no se pueden deshacer.

Si en vez de COMMIT hubiésemos finalizado la transacción con ROLLBACK, no se ejecuta la sentencia UPDATE de este ejemplo,

es decir, se deshace la transacción realizada. Lo probaremos en el siguiente apartado.

Hay instrucciones SQL que, dadas sus características, **implican confirmación automática**. Éstas son:

- **Instrucciones del DDL que definen o modifican objetos de la base de datos. CREATE, ALTER, DROP, RENAME, TRUNCATE.**
- **Instrucciones que modifican tablas de la base de datos administrativa mysql. GRANT y REVOKE**
- Instrucciones de control de transacciones y bloqueo de tablas. START TRANSACTION BEGIN, LOCK y UNLOCK.
- **Instrucciones de carga de datos. LOAD DATA. Instrucciones de administración de tablas. ANALIZE, CHECK, OPTIMIZE y REPAIR.**

## 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 y MySQL 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.



Vamos a iniciar una transacción para incrementar en 100 las horas que se hayan trabajado en el proyecto 'DAG'

```
START TRANSACTION;
UPDATE trabaja
SET nhoras=nhoras+100
WHERE cdpro='DAG';
```

Consulta ahora la tabla *trabaja* y comprobarás que se han incrementado las horas correspondientes.

Vamos a deshacer los cambios, (la transacción está abierta), para ello ejecutamos **ROLLBACK;** un **ROLLBACK**.

Vuelve a consultar la tabla *trabaja*, y comprobarás que se han deshecho los cambios.

### Para saber más

Ejemplo sobre el uso de **Rollback**. en Oracle [Ejemplo de Rollback](#).

En el siguiente enlace puedes ver más ejemplos de transacciones en MySQL [Transacciones en MySQL](#)

## 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.

Otros comandos para la gestión de transacciones son los que gestionan los puntos de salvaguarda o savepoint:

- **SAVEPOINT id.** Es una sentencia que permite nombrar cierto paso en un conjunto de instrucciones de una transacción.
- **ROLLBACK TO SAVEPOINT id.** Permite deshacer el conjunto de operaciones realizadas a partir del identificador de savepoint.
- **RELEASE SAVEPOINT id.** Elimina o libera el savepoint creado.

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;
```

### Para saber más

Artículo sobre los puntos de restauración. [Savepoint.](#)

Ejemplo sobre el uso de los puntos de restauración en Oracle. [Ejemplo de punto de restauración.](#)



**Cualquier operación COMMIT o ROLLBACK sin argumentos eliminará todos los savepoints creados.**

Por ejemplo, comprueba que las siguientes sentencias solo confirman el primer INSERT.

```
START TRANSACTION;
INSERT INTO proyectosx.departamento VALUES('88','depTrans1','ciudad1');
SAVEPOINT sp1;
INSERT INTO proyectosx.departamento VALUES
('90','depTrans2','ciudad2'),('91','depTrans3','ciudad3');
ROLLBACK TO SAVEPOINT sp1;
```



## 7. PROBLEMAS ASOCIADOS AL ACCESO SIMULTÁNEO A LOS DATOS

---

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

- **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.

Las bases de datos Oracle y MySQL proporcionan concurrencia de datos, consistencia e integridad en las transacciones mediante sus mecanismos de bloqueo. Los bloqueos se realizan de forma automática y no requieren la actuación del usuario.

### Reflexiona

Por ejemplo, si se ofrece la posibilidad de que un empleado transfiera horas trabajadas a otro empleado, qué ocurriría si justo en un mismo momento dos empleados le transfieren horas a un tercero. ¿Podría ocurrir que sólo llegara a realizarse una de las dos operaciones?

Para explicar la idea de acceso simultáneo y su problemática imagina 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.

### Para saber más

Interesante enlace sobre control de concurrencia en Oracle. [Integridad. Control de concurrencia.](#)

Interesante enlace sobre transacciones y control de concurrencia en MySQL. [Transacciones y concurrencia en MySQL](#)

## 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.

Bloquear una tabla o vista permite que nadie más pueda hacer uso de la misma de modo que tenemos la exclusividad de lectura y/o escritura sobre ella.

MySQL y Oracle permiten el bloqueo de tablas por parte de los clientes con el objeto de cooperar con otras sesiones de clientes o evitar que estos puedan modificar datos que necesitamos en nuestra sesión.

Los bloqueos permiten simular transacciones así como acelerar operaciones de modificación o de inserción de datos.

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 y también MySQL (con su motor InnoDB) 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.

### Para saber más

Documento, en inglés, sobre los bloqueos optimistas y pesimistas en Oracle con algunos ejemplos. [Optimistic Locking with Concurrency in Oracle](#).

En este enlace puedes ampliar la información sobre bloqueos optimistas y pesimistas en MySQL [Bloqueos optimista y pesimista en MySQL](#)

## 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.

### Debes conocer

La política de bloqueos de MySQL sigue la siguiente secuencia:1. Ordena internamente las tablas a bloquear.2. Si una tabla debe bloquearse para lectura y escritura sitúa la solicitud de bloqueo de escritura en primer lugar.3. Se bloquea cada tabla hasta que la sesión obtiene todos sus bloqueos. De este modo se evita el conocido 'deadlock' o '**bloqueo mutuo**', fenómeno que hace que el acceso a los bloqueos se pueda prolongar indefinidamente al no poder ningún proceso obtenerlos. El uso de bloqueos está íntimamente ligado a las transacciones, especialmente para tabla de tipo transaccional InnoDB. En tablas InnoDB, los bloqueos se obtienen a nivel de fila, permitiendo que varios usuarios puedan bloquear varias filas simultáneamente. En tablas InnoDB todo es una transacción, de hecho, como ya hemos comentado, si autocommit está activado (=1 u ON), cada operación SQL es en sí misma una transacción. En este caso podemos iniciar una transacción con START TRANSACTION o BEGIN y terminarla con COMMIT para que los cambios sean permanentes o ROLLBACK para deshacer los cambios. *En el caso de autocommit inactivo se considera que siempre hay una transacción en curso, en cuyo caso las sentencias COMMIT y ROLLBACK suponen el final de dicha transacción y comienzo de la siguiente.* **Tipos de bloqueo en InnoDB.** En este tipo de tablas se diferencian los dos tipos de bloqueo indicados anteriormente:

- **Bloqueo compartido.** Permite a una transacción la lectura de filas. En este caso, varias transacciones pueden adquirir bloqueos sobre las mismas filas pero ninguna transacción puede modificar dichas filas hasta que no se liberen los bloqueos.
- **Bloqueo exclusivo.** Permite a una transacción bloquear filas para actualización o borrado. En este caso las transacciones que deseen adquirir un bloqueo exclusivo deberán esperar a que se libere el bloqueo sobre las filas afectadas.

## TEMA 5: Tratamiento de datos

También se soporta el ‘bloqueo de múltiple granularidad’, según el cual una transacción puede indicar que va a bloquear algunas filas de una tabla bien para lectura o para escritura. Es lo que se denomina una intención de bloqueo. De este modo, es más fácil para MySQL gestionar posibles conflictos evitando los temidos *deadlocks*, ya que permite a varias transacciones compartir la reserva de una tabla puesto que el bloqueo se produce fila a fila en el momento de la modificación. Así, si dos transacciones reservan la misma tabla, solo en el momento en que una de ellas esté modificando una fila, ésta quedará bloqueada.

### Para saber más

Definición y ejemplos del bloqueo exclusivo y compartido [Integridad y control de concurrencia](#)

## 7.3. BLOQUEOS AUTOMÁTICOS

La base de datos Oracle y MySQL bloquea automáticamente un recurso usado por una transacción para prevenir que otras transacciones realicen alguna acción que requiera acceso exclusivo sobre el mismo recurso. La base de datos adquiere automáticamente diferentes tipos de bloqueos con diferentes niveles de restricción dependiendo del recurso y de la operación que se realice.

Los bloqueos que realiza la base de datos Oracle y MySQL 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 y MySQL usa varios tipos de bloqueos del sistema para proteger la base de datos interna y las estructuras de memoria.

### Para saber más

En el siguiente enlace dispone de amplia información sobre los bloqueos de MySQL [Bloqueos en MySQL](#)

## 7.4. BLOQUEOS MANUALES

Hemos visto que la base de datos Oracle y MySQL 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. 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.

### Oracle

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.

### MySQL

Bloquear una tabla o una vista permite que nadie más pueda hacer uso de la misma, de manera que tenemos la exclusividad de lectura y/o escritura sobre ella.

MySQL permite el bloqueo de tablas por parte de clientes con el objeto de cooperar con otras sesiones de clientes o de evitar que estos puedan modificar datos que necesitemos en nuestra sesión.

La instrucción para bloqueo de tablas es **LOCK**. Su sintaxis es:

<b>LOCK TABLES</b>
<i>nombre_tabla</i> [[AS] <i>alias</i> ] <i>tipo_bloqueo</i>
[, <i>nombre_tabla</i> [[AS] <i>alias</i> ] <i>tipo_bloqueo</i>
Tipo_bloqueo:
<i>READ</i> [ <i>LOCAL</i> ]   [ <i>LOW_PRIORITY</i> ] <i>WRITE</i>

## TEMA 5: Tratamiento de datos

Para desbloquear tablas (solo podemos desbloquear todas las tablas a la vez), se usa la instrucción:

**UNLOCK TABLES**

### Tipos de bloqueo READ y WRITE:

- **READ:** En este caso la sesión o cliente que tiene el bloqueo puede leer pero ni él ni ningún otro cliente podrá escribir en la tabla.

Es un bloqueo que pueden adquirir varios clientes simultáneamente y permite que cualquiera pueda leer las tablas bloqueadas.

El modificador **LOCAL** permite que haya inserciones concurrentes de otros clientes mientras dura el bloqueo.

- **WRITE:** La sesión o cliente que adquiere este tipo de bloqueo puede leer y escribir en la tabla, pero ningún otro cliente podrá acceder a ella o bloquearla.

Por defecto, los bloqueos de escritura tiene mayor prioridad que los de lectura, de manera que si una sesión solicita un bloqueo de escritura éste tendrá prioridad sobre otras solicitudes de bloqueo de lectura. De este modo, hasta que la sesión que solicitó el bloqueo de escritura no libere dicho bloqueo, no se podrán adquirir nuevos bloqueos de lectura.

El modificador **LOW PRIORITY** permite que los bloqueos de lectura se adquieran antes que el de escritura. De hecho, el bloqueo de escritura solo se obtendrá cuando no queden bloqueos de lectura pendientes.

Por ejemplo, para bloquear la tabla departamento en modo de solo lectura:

```
USE proyectosx;
LOCK TABLES departamento READ;

SELECT * FROM departamento; //realiza la lectura y muestra los departamentos.

UPDATE departamento //da error, pues la tabla está bloqueada para lectura
SET ciudad="ciudad"
WHERE nombre='depar20';

UNLOCK TABLES; //desbloquea la tabla
```

### Para saber más

Definición y ejemplos sobre transacciones y control de concurrencia con bloqueos manuales. [Transacciones y control de concurrencia.](#)

Ejemplos de bloqueos manuales en MySQL [Bloqueos manuales en MySQL](#)

## 8. ENLACES DE REFUERZO Y AMPLIACIÓN

---

### MySQL

- Documentación oficial del SGBD MySQL de la sintaxis de la sentencia INSERT(En Inglés): <https://dev.mysql.com/doc/refman/8.0/en/insert.html>
- Documentación oficial del SGBD MySQL de la sintaxis de la sentencia DELETE(En Inglés): <https://dev.mysql.com/doc/refman/8.0/en/delete.html>
- Documentación oficial del SGBD MySQL de la sintaxis de la sentencia UPDATE(En Inglés): <https://dev.mysql.com/doc/refman/8.0/en/update.html>
- Documentación oficial del SGBD MySQL de la sintaxis de las sentencias COMMIT, ROLLBACK y START TRANSACTION para realizar TRANSACCIONES (En Inglés): <https://dev.mysql.com/doc/refman/8.0/en/commit.html>
- Documentación oficial del SGBD MySQL de la sintaxis de las sentencias SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT para realizar TRANSACCIONES (En Inglés): <https://dev.mysql.com/doc/refman/8.0/en/savepoint.html>

### ORACLE

- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia INSERT(En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_9014.htm](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_9014.htm)
- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia DELETE(En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_8005.htm#i2143613](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_8005.htm#i2143613)
- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia UPDATE(En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_10007.htm#i2189756](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_10007.htm#i2189756)
- Documentación oficial del SGBD ORACLE de la sintaxis de las sentencias COMMIT y SET TRANSACTION para realizar TRANSACCIONES con ejemplos (En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_10005.htm](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_10005.htm)
- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia ROLLBACK para realizar TRANSACCIONES con ejemplos (En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_9021.htm#i2104635](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_9021.htm#i2104635)
- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia SAVEPOINT para realizar TRANSACCIONES con ejemplos (En Inglés): [https://docs.oracle.com/cd/B12037\\_01/server.101/b10759/statements\\_10001.htm#BABFIJGC](https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_10001.htm#BABFIJGC)

### PostgreSQL

- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia INSERT (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-insert.html>

## TEMA 5: Tratamiento de datos

- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia DELETE (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-delete.html>
- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia UPDATE (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-update.html>
- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia COMMIT para realizar TRANSACCIONES con ejemplos (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-commit.html>
- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia ROLLBACK para realizar TRANSACCIONES con ejemplos (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-rollback.html>
- Documentación oficial del SGBD PostgreSQL de la sintaxis de la sentencia SAVEPOINT para realizar TRANSACCIONES con ejemplos (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-savepoint.html>

### Ejemplos On-Line sobre la utilización de INSERT, UPDATE y DELETE

- INSTRUCCIÓN INSERT INTO: [http://www.w3schools.com/sql/sql\\_insert.asp](http://www.w3schools.com/sql/sql_insert.asp)
- INSTRUCCIÓN INSERT INTO SELECT: [http://www.w3schools.com/sql/sql\\_insert\\_into\\_select.asp](http://www.w3schools.com/sql/sql_insert_into_select.asp)
- INSTRUCCIÓN UPDATE: [http://www.w3schools.com/sql/sql\\_update.asp](http://www.w3schools.com/sql/sql_update.asp)
- INSTRUCCIÓN DELETE: [http://www.w3schools.com/sql/sql\\_delete.asp](http://www.w3schools.com/sql/sql_delete.asp)