TEMA 9: PUESTA AL DÍA DE LOS DATOS.

INSERT, UPDATE, DELETE (Lenguaje DML)

OBJETIVOS:

- Utilizar asistentes y herramientas gráficas
- Identificar las herramientas y sentencias para modificar el contenido de la base de datos.
- Insertar, borrar y actualizar datos en las tablas.
- Incluir en una tabla la información resultante de la ejecución de una consulta.
- Adoptar medidas para mantener la integridad y consistencia de la información.
- Diseñar guiones de sentencias para llevar a cabo tareas complejas.
- Reconocer el funcionamiento de las transacciones.
- Anular parcial o totalmente los cambios producidos por una transacción.
  Identificar los efectos de las distintas políticas de bloqueo de registros.
- Utilizar el lenguaje de definición de datos.

HERRAMIENTAS GRÁFICAS PARA LA EDICIÓN DE DATOS

Existen multitud de herramientas gráficas para la edición de datos, algunas, incorporadas como parte del software del gestor de base de datos, por ejemplo, el entorno gráfico de Access; otras herramientas se distribuyen como paquetes a añadir al SGBD, como phpMyAdmin de MySQL; y el software de terceros, programas por los que hay que pagar una licencia aparte como *TOAD, Navicat* o *Aqua Data Studio.* Otros gestores como Oracle, no incorporan expresamente una herramienta de edición de datos como tal (se pueden consultar, pero no se puede editar datos desde Enterprise Manager), pero se aprovechan de herramientas de terceros para esta labor.

Edición con phpMyAdmin

Una de las muchas utilidades de phpMyAdmin es la inserción de datos a través de formularios web, donde, de forma muy sencilla, se escriben los valores para cada campo de la tabla deseada. Tan sólo hay que seleccionar la pestaña *Insertar* y seleccionar la tabla donde se va a insertar los registros. Se rellenan los valores y se pulsa continuar

¿Sabías que ... ? En MySQL existe la sintaxis **extended insert,** que permite insertar varios registros con **un** sólo INSERT. (se verá más adelante)

Access como entorno gráfico para otros gestores

Si se conoce el entorno Access es muy sencillo conectado con otros gestores, y de esta manera, aprovecharse del conocimiento de la interfaz de Microsoft para administrar otros gestores de bases de datos. Esto es posible gracias a la utilización de conectores ODBC. ODBC son las siglas de *Open Database Connectivity*. Es una herramienta que, de forma estándar, permite conectar aplicaciones a cualquier gestor de bases de datos. Muchas aplicaciones ofimáticas, como Excel, Word y el propio Access permite el acceso remoto a datos de un SGBD. Para esto, tan sólo es necesario instalar el driver ODBC para ese SGBD. De esta manera, se puede crear una DS o *Data Source Name,* es decir, una referencia a cierta base de datos.

Esta referencia se crea en Windows a través del panel de Control, Herramientas Administrativas, y la opción *Orígenes de Datos ODBC.* Cada origen de datos requiere, además de un nombre para el propio origen, el nombre del usuario, la contraseña, el nombre de la base de datos y los datos de conexión al servidor (Dirección IP y puerto TCP /UDP por donde se conecta).

A través de un origen de datos ODBC, se pueden enlazar a Access las tablas de un SGDB para el cual se ha configurado la DSN. Para ello, a través de la pestaña de Access *Datos Externos,* opción *más,* se elige la opción *bases de datos de ODBC* ya continuación se siguen los pasos del asistente. Después, se elige la opción de vincular al origen de datos creando una tabla vinculada, y finalmente, se selecciona la DSN creada anteriormente desde la pestaña *Origen de datos de equipo.*

o **Actividad 1:** Descarga desde la página web de mysql www.mysql.com, apartado Downloads, connectors, el conector ODBC para MySQL. Instálalo en tu ordenador y conéctate a la base de datos jardinería de la máquina virtual de prácticas. Si lo prefieres, conéctate a algún otro servidor que tengas disponible. (Asegúrate, en ese caso, de tener desactivada la opción bind-address del servidor mysql en el fichero /etc/mysql/my.cnf para poder conectar desde fuera de la propia máquina virtual).

La sentencia INSERT

La sentencia INSERT de SQL permite insertar una fila en una tabla, es decir, añadir un registro de información a una tabla.

El formato de uso es muy sencillo:

```
INSERT [INTO] nombre_tabla [(nombre_columna, ... )]
VALUES ({expr | DEFAULT}, ... )
```

nombre.tabla es el nombre de la tabla donde se quiere insertar la fila. Después del nombre de la tabla, de forma optativa, se pueden indicar las columnas donde se va a insertar la información. Si se especifican las columnas, la lista de valores (VALUES) a insertar se asociará correlativamente con los valores a las columnas indicadas. Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla. A continuación se muestran unos cuantos ejemplos:

```
desc mascotas;
+---------+-------------+------+-----+---------+
| Field   | Type        | Null | Key | Default |
+---------+-------------+------+-----+---------+
| codigo  | int(11)     | NO   | PRI | NULL    |
| nombre  | varchar (50)| YES  |     | NULL    |
| raza    | varchar(50) | YES  |     | NULL    |
| cliente | varchar(9)  | YES  |     | NULL    |
+---------+-------------+------+-----+---------+
```

```
#INSERT especificando la lista de columnas
INSERT INTO mascotas (Codigo, Nombre, Raza)
VALUES
(1,'Pequitas','Gato Común Europeo')
```

Este tipo de INSERT, hace corresponder a la columna Codigo el valor 1, a la columna Nombre el valor 'Pequitas' y a la columna raza el valor 'Gato Común Europeo'. La Columna cliente, queda con un valor NULL, puesto que no se ha indicado un valor.

```
#INSERT sin especificar la lista de columnas.
INSERT INTO mascotas VALUES
(2, 'Calcetines', 'Gato Común Europeo', '59932387L')
```

En este caso, al no especificarse la lista de columnas, hay que indicar todos los Valores para todas las columnas en el orden en que están definidas las columnas en la tabla.

```
#INSERT con columnas con valores por defecto
INSERT INTO vehiculos VALUES ('1215 BCD','Toledo TDI', DEFAULT);
```

Aquí, se ha usado el valor DEFAULT para asignar el valor por defecto a la tercera columna de la tabla vehículos, es decir, la columna marca tiene definida la asignación por defecto del valor 'Seat'.

Si se construye una sentencia INSERT con más o menos campos en la lista de valores que el número de columnas especificadas (o número de columnas de la tabla) el SGBD informará del error.

```
   #INSERT Errónea
insert into vehículos (Matricula,Modelo,Marca)
                     VALUES ('4123 BFH','Ibíza');
   ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

La sentencia INSERT extendida

La sintaxis extendida de INSERT para gestores tipo MySQL es la siguiente:

```
INSERT [INTO] nombre_tabla [(nombre_columna, ... )]
VALUES ({expr |DEFAULT}, ... ),( ... ), ...
```

Los puntos suspensivos del final indican que se puede repetir varias veces la lista de valores. Así, MySQL admitiría algo del estilo:

```
insert into vehiculos (Matricula,Modelo,Marca)
VALUES ('4123 BFH','Ibiza','Seat'),
('1314 FHD' ,'Toledo' ,'Seat'),
('3923 GJS','León','Seat');
```

INSERT y SELECT

Una variante de la sentencia INSERT consiste en una utilizar la sentencia SELECT para obtener un conjunto de datos y, posteriormente, insertarlos en la tabla.

```
INSERT
[INTO] nombre_tabla [(nombre_columna, ... )]
SELECT ... FROM ...
```

Se puede ejecutar la siguiente consulta:

```
#Inserta en una tabla Backup todos los vehículos
INSERT INTO BackupVehiculos
SELECT * FROM vehiculos;
```

La sentencia SELECT debe devolver tantas columnas como columnas tenga la tabla donde se introduce la información. En el ejemplo anterior, la tabla Backup-Vehiculos tiene una estructura idéntica a la tabla vehículos.

Se puede ver, además, que la sentencia SELECT puede ser tan compleja como se desee, usando filtros, agrupaciones, ordenaciones, etc.

La sentencia UPDATE

La sentencia UPDATE de SQL permite modificar el contenido de cualquier columna y de cualquier fila de una tabla. Su sintaxis es la siguiente:

```
UPDATE nombre_tabla
SET nombre_coll=exprl [, nombre_col2=expr2]  ...
[WHERE filtro]
```

La actualización se realiza dando a las columnas nombre.col1 , nombre.col2 ... los valores exprl, expr2, ... Se actualizan todas las filas seleccionadas por el filtro indicado mediante la cláusula WHERE. Esta cláusula WHERE, es idéntica a la que se ha utilizado para el filtrado de registros en la sentencia SELECT.

Por ejemplo, si se desea actualizar el equipo de 'Pau Gasol' porque ha fichado por otro equipo, por ejemplo, los 'Knicks', habría que ejecutar la siguiente sentencia:

```
UPDATE jugadores SET Nombre_equipo='Knicks'
WHERE Nombre='Pau Gasol';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

El gestor informa de que el filtro seleccionó una fila, y que, por tanto, cambió 1 fila, en este caso, la columna Nombre.equipo de esa fila seleccionada.

Es posible cambiar más de una columna a la vez:

```
UPDATE jugadores SET Nombre_equipo='Knicks', Peso=210
WHERE Nombre='Pau Gasol';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Si se omite el filtro, el resultado es la modificación de todos los registros de la tabla, por ejemplo, para cambiar el peso de los jugadores de la NBA de libras a kilos:

```
UPDATE jugadores SET Peso=Peso*0.4535;
```

La sentencia DELETE

En SQL se utiliza la sentencia DELETE para eliminar filas de una tabla. Su sintaxis es:

```
DELETE FROM nombre_tabla
     [WHERE filtro]
```

El comando DELETE borra los registros seleccionados por el filtro WHERE, que es idéntico al de la sentencia SELECT.

Si se desea borrar al jugador' Jorge Garbajosa' de la base de datos, habría que escribir la siguiente sentencia:

```
DELETE FROM jugadores WHERE Nombre='Jorge Garbajosa';
Query OK, 1 row affected (0.01 se e)
```

Si se omite el filtro, el resultado es el borrado de todos los registros de la tabla:

```
DELETE FROM jugadores;
Query OK, 432 rows affected (2.42 sec)
```

La sentencias UPDATE y DELETE con subconsultas

Es posible actualizar o borrar registros de una tabla filtrando a través de una subconsulta. La única limitación es que hay gestores que no permiten realizar cambios en la tabla que se está leyendo a través de la sub consulta.

Por ejemplo, si se desea eliminar los empleados 'Representante Ventas' que no tengan clientes se podría codificar:

```
DELETE FROM Empleados
WHERE CodigoEmpleado Not in
  (SELECT CodigoEmpleadoRepVentas
       FROM Clientes)
              AND Puesto='Representante Ventas';
```

No sería posible, sin embargo, escribir una sentencia de este tipo para borrar los clientes con LimiteCredito=0, puesto que se leen datos de la misma tabla que se borran:

```
DELETE FROM Clientes
WHERE CodigoCliente in
(SELECT CodigoCliente
   FROM Clientes WHERE LimiteCredito=0);
ERROR 1093 (HYOOO): You can't specify target table 'Clientes' for update
in FROM clause
```

Borrado y modificación de registros con relaciones

Hay que tener en cuenta que no siempre se pueden borrar o modificar datos: Considérese por ejemplo, que un cliente llama a una empresa pidiendo darse de baja como cliente, pero el cliente tiene algunos pagos pendientes. Si el operador de la BBDD intenta eliminar el registro (DELETE), el SGBD debería informar de que no es posible eliminar ese registro puesto que hay registros relacionados.

O por ejemplo, se desea cambiar (UPDATE) el nombre de un equipo de la NBA (que es su clave primaria), ¿qué sucede con los jugadores? También habrá que cambiar el nombre del equipo de los jugadores, puesto que el campo Nombre_Equipo es una clave foránea.

En este punto, hay que recordar las cláusulas REFERENCES de la sentencia CREATE TABLE para crear las relaciones de clave foránea-clave primaria de alguna columna de una tabla:

```
definición_referencia:
  REFERENCES nombre_tabla [(nombre_columna, ... )]
[ON DELETE opción_referencia]
[ON UPDATE opción_referencia]

opción_referencia:
CASCADE | SET NULL | NO ACTION
```

Las cláusulas ON DELETE y ON UPDATE personalizan el comportamiento de estos dos casos. Si por ejemplo, se intenta eliminar un registro con otros registros relacionados, y se ha seleccionado la opción ON DELETE NO ACTION y ON UPDATE NO ACTION el comportamiento sería el siguiente:

```
#dos tablas relacionadas en mysql
#han de ser innodb para soportar FOREING KEYS

CREATE TABLE clientes (
dni varchar(15) PRIMARY KEY,
nombre varchar(50),
direccion varchar(50)
)engine=innodb;

CREATE TABLE pagos_pendientes(
dni varchar (15) ,
importe double,
FOREIGN KEY(dni) REFERENCES clientes(dni)
on delete NO ACTION
on update NO ACTION
)engine=innodb;



#un cliente y dos pagos pendientes
```

```
INSERT INTO clientes
    VALUES ('5555672L','Pepe Cifuentes','C/Los almendros,23');
INSERT INTO pagos_pendientes VALUES ('5555672L',500);
INSERT INTO pagos_pendientes VALUES ('5555672L',234.5);

#Se intenta borrar el cliente y no es posible
DELETE FROM clientes WHERE dni='5555672L';


ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails ('gestion/pagos_pendientes',
CONSTRAINT 'pagos_pendientes_ibfk_1'
FOREIGN KEY ('dni') REFERENCES 'clientes' ('dni'))

#Se intenta modificar el dni del cliente y no lo permite
UPDATE clientes set dni='55555555L' WHERE dni='5555672L';
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails ('gestion/pagos_pendientes',
CONSTRAINT 'pagos_pendientes_ibfk_1'
FOREIGN KEY ('dni') REFERENCES 'clientes' ('dni'))



#de igual modo si se intenta borrar la tabla clientes,
#tampoco podemos

DROP TABLE clientes;
ERROR 1217 (23000): Cannot delete or update
a parent row: a foreign key constraint fails
```

OTRO SUPUESTO DONDE SE ACTUALIZA Y SE BORRA EN CASCADA.

```
#dos tablas relacionadas en mysql

create table clientes (
dni varchar(15) primary key,
nombre varchar(50),
direccion varchar(50)
) engine=innodb;


create table pagos_pendientes(
dni varchar (15) ,
importe double,
foreign key (dni) references clientes(dni)
on delete CASCADE on update CASCADE
) engine=innodb;

#un cliente y dos pagos pendientes
INSERT INTO clientes values ('5555672L','Pepe Cifuentes','C/Losalmendros,23');
INSERT INTO pagos_pendientes VALUES ('5555672L',500);
INSERT INTO pagos_pendientes VALUES ('5555672L',234.5);

#se borra el cliente ...
DELETE FROM clientes WHERE dni='5555672L';
Query OK, 1 row affected (0.00 sec)

#además, se verifica que ha borrado en cascada sus pagos pendientes.
SELECT * FROM pagos_pendientes;
Empty set (0.00 sec)

#si en lugar de borrar el cliente, se hubiera cambiado el dni:
UPDATE clientes set dni='55555555L' WHERE dni='5555672L';
Query OK, 1 row affected (0.02 sec)
```

```
#ha cambiado el dni de los pagos en cascada.
SELECT * FROM pagos_pendientes;

+-----------+---------+
|   dni     | importe |
+-----------+---------+

| 55555555L | 500   |
| 55555555L | 234.5 |

+--------+----
```

Recuerda. En Oracle, sólo existe la cláusula ON DELETE. Para implementar ON UPDATE, se debe generar un *TRIGGER (Se verán en el próximo curso, en Administración de BD*)

o **Actividad 2:** Crea las tablas de mascotas y clientes con la opción ON UPDATE y ON DELETE a SET NULL y comprueba el resultado de ejecutar las inserciones, actualizaciones y borrados de los ejemplos anteriores.