1. MODIFICACIÓN, BORRADO E INSERCIÓN DE DATOS.

MODIFICACION

Para modificar los datos se utiliza la sentencia `UPDATE` tal y como se muestra a continuación:

```
UPDATE  nombreTabla  [, nombreTabla2 ..]
SET   Campo1=Expr1 | DEFAULT, Campo2=Expr2 ..
[WHERE <condicion>]
```

⇒ **nombreTabla**: Es el nombre de la tabla que se va a actualizar.

⇒ **SET**: Especifica la lista de nombres de columnas o variables que se van a actualizar.

⇒ **Campo1**,....: Es una columna que contiene los datos que se van a cambiar. Se debe encontrar en la tabla o en la vista especificadas en la cláusula UPDATE.

⇒ **Expr1**,..: Un valor literal, una expresión o una instrucción SELECT entre paréntesis **que devuelve un solo valor**. **DEFAULT**: Especifica que el valor predeterminado definido para la columna debe sustituir al valor existente en la columna.

⇒ La cláusula **WHERE** es opcional, y su formato es el mismo que el usado en la sentencia `SELECT`. Si no se indica, todas las filas de la tabla serán modificadas.

Ej: Al empleado cuyo número es 44444 se le cambia el departamento a 2 y el salario a 250000.

```
UPDATE EMPLEADO
      SET NUMDEPT=2, SALARIO=250000
      WHERE NSS=44444;
```

Ej: Hacer que el empleado cuyo numero es 44444 gane el salario máximo.

```
UPDATE EMPLEADO
      SET SALARIO = (SELECT MAX(SALARIO) FROM EMPLEADO)
      WHERE NSS=44444;
-- error
```

Esta instrucción dará error. La versión que empleamos de MySQL no soporta instrucciones UPDATE con cláusulas que hagan referencia a la misma tabla que se intenta actualizar.

Ej: A los empleados del departamento de investigación, aumentarles el salario un 15%

```
UPDATE EMPLEADO
SET SALARIO=SALARIO*1.15
      WHERE NUMDEPT = (SELECT NUMDEP FROM DEPARTAMENTO
         WHERE NOMBREDEP='Investigacion');
```

También podría indicarse como sigue:

```
UPDATE EMPLEADO, departamento
SET SALARIO=SALARIO*1.15
      WHERE NUMDEPT=NUMDEP
   AND NOMBREDEP='Investigacion';
```

La misma sentencia anterior podría escribirse empleando JOIN :

```
UPDATE EMPLEADO JOIN departamento on NUMDEPT=NUMDEP
SET SALARIO=SALARIO*1.15
      WHERE  NOMBREDEP='Investigacion';
```

Aunque no es muy recomendable, y otras versiones de SQL no lo soportan, podríamos actualizar más de una tabla a la vez. Así, con la siguiente instrucción, podríamos cambiar la dirección (de la tabla Empleados) de todos los empleados del departamento de Investigación, y el numero de empleados de ese mismo departamento (de la tabla Departamentos)

```
UPDATE EMPLEADO, departamento
SET dirección='XXXXXXX' ,
 numempdep=999
      WHERE NUMDEPT=NUMDEP
   AND NOMBREDEP='Investigacion';
```

Es importante considerar que la actualización de datos en una tabla puede afectar a otras tablas que estén relacionadas con ella. Por ello, es necesario mantener la integridad referencial. Si, por ejemplo, queremos modificar la clave del departamento de Investigación, que tiene valor 2, dándole valor 5, podremos o no realizar esta acción según como hayamos especificado la integridad referencial. Si la I.R. está como Restrict o No Action, no podremos ejecutar la sentencia siguiente:

```
UPDATE DEPARTAMENTO
SET NUMDEP=5
WHERE NOMBREDEP ='Investigacion';
```

Necesitaremos que la I.R. esté establecida On Cascade (y se modificarán los valores de la columna Empleado.Numdept) o Set Null (y pondrá en Null la columna Empleado.Numdept de los registros afectados, siempre que esa columna no tenga una restricción NOT NULL) para poder ejecutarla.

Debemos también tener en cuenta otras tablas a las que pudieran afectarles los cambios. Siguiendo con este ejemplo, si modificamos la clave del departamento de Investigación, y esta modificación se propaga **On Cascade** a la tabla Empleados, debería hacerlo también a todas las tablas de la BD que la tengan como clave foránea: Proyectos.Numdep y Departamento-Ubi.Numdep.

BORRADO DE DATOS

Para borrar tuplas de una tabla se utiliza la sentencia `DELETE`, cuyo formato general es:

DELETE FROM Tabla

[**WHERE** <condicion>]

Como en el caso de `UPDATE`, la cláusula `WHERE` es una condición cuyo formato es el mismo que el de una sentencia `SELECT`, y si no se especifica ninguna, se borrarán todas las tuplas de la tabla.

Por ejemplo, para borrar el departamento "Comunicación":

```
DELETE FROM DEPARTAMENTO
      WHERE NOMBREDEP='Comunicación';
```

Es muy recomendable antes de borrar realizar la misma consulta pero de selección, para visualizar previamente los registros afectados.

Ej: Borrar los empleados que participan en más de dos proyectos. **Debo especificar una única tabla de la que borrar.** Si necesito otras tablas para especificar condiciones, deben aparecer en subconsultas.

```
DELETE FROM Empleado
WHERE nss IN (select nss FROM empleadoproyecto
              GROUP BY nss
              HAVING COUNT(*) >2);
```

La misma consulta anterior, reescrita como una consulta multitabla, me daría un error sintáctico.

```
DELETE FROM Empleado e , empleadoproyecto  ep    -- ERROR
where e.nss = ep.nss
group by e.nss
having count(*) >2;
```

Es importante considerar que la eliminación de datos en una tabla puede afectar a otras tablas que estén relacionadas con ella, como veíamos en la sentencia UPDATE. Por ello, es necesario mantener la integridad referencial.

AÑADIR REGISTROS. *INSERT INTO*

Agrega uno o varios registros a una tabla.

⇒ **Inserción de un o varios registros cuando conocemos los valores a insertar:** Se puede utilizar la instrucción INSERT INTO para agregar **un o varios registros a una tabla**. En este caso, se especifica el nombre y valor de cada campo del registro. Debe especificar cada uno de los campos del registro a los que se va a asignar un valor y dicho valor para ese campo. Cuando no especifique un campo, se inserta el valor predeterminado o Null en las columnas no especificadas. Los registros se agregan al final de la tabla.

INSERT INTO Tabla [(campo1, campo2, ...)]

VALUES (valor1, valor2, ...), [(valor1, valor2, ...), (valor1, valor2, ...), …]

Si omite la lista de identificadores de campos, la cláusula VALUES debe incluir un valor para cada campo de la tabla o, de lo contrario, la instrucción INSERT fallará. Utilice una instrucción INSERT INTO adicional con una cláusula VALUES para cada registro adicional que desee crear.

⇒ **Inserción de uno o varios registros cuyos valores obtenemos de una subconsulta:** También puede utilizar la instrucción INSERT INTO para añadir un **conjunto de registros de otra tabla** utilizando la cláusula SELECT ... FROM tabla .En este caso, la cláusula SELECT especifica los campos a añadir a la tabla destino especificada.

INSERT INTO Tabla [(campo1, campo, ...)]

SELECT campo1, campo2[, ...]

FROM tablaOrigen

EJ: Por ejemplo, suponiendo que tenemos definida una tabla `COPIADEPT` con la misma estructura que DEPARTAMENTO. Insertar todos los registros de departamento en COPIADEPT

```
INSERT INTO COPIADEPT
SELECT * FROM DEPARTAMENTO    ;
```

EJ: Por ejemplo, suponiendo que tenemos definida una tabla `BUENOSDEPT` con la misma estructura que `DEPARTAMENTO`. Insertar los departamentos con más de 3 empleados:

```
INSERT INTO BUENOSDEPT
SELECT D.*
FROM DEPARTAMENTO  D, EMPLEADO E
WHERE  D.Numdep= E.Numdept
GROUP BY D.Numdep, D.Nombredep, D.Numempdep, D.NSSgerente,
 D.fechainicgerente
HAVING COUNT(*) >=3;
```

NOTA.- La claúsula GROUP podría indicarse sólo con la columna D.Numdep en mysql, pero en otras versiones sería necesario especificar todas las columnas de la tabla Departamento, puesto que son las que aparecen en la lista de campos de SELECT.

También puede crearse una tabla como resultado de una consulta.

CREATE TABLE NombreTabla **AS** consulta

Crear una tabla de nombre Dep12 con los datos de los departamentos de código 1 y 2

```
CREATE TABLE Dep12 as
SELECT * FROM departamento WHERE NUMDEP IN (1,2);
```