BLOQUE 4 – TRATAMIENTO DE DATOS

BASES DE DATOS, 1° DAM

DR. JORGE JUAN MUNOZ MORERA

INSERT INTO

La sentencia INSERT INTO se utiliza para añadir nuevos registros en una tabla. Existen dos formas de utilizar esta sentencia.

En la primera forma se especifican los nombres de columnas y los valores a insertar. Esta forma se utiliza cuando hay columnas para las que no queremos insertar valores. El número de valores debe coincidir con el de columnas, y sus tipos también.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

En la segunda forma no se especifican los nombres de las columnas. En este caso, el número de valores debe coincidir con el número de columnas que tenga la tabla, y se asignarán los valores por orden.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

INSERT INTO - EJEMPLOS

```
INSERT INTO customers(CustomerID, CompanyName, City, Country) VALUES ('Jorge','Vedruna','Seville','Spain');
```

```
INSERT INTO customers(CustomerID, CompanyName, Address, City, PostalCode, Country)
VALUES ('Pedro','Vedruna','Seville','Triana','41927', 'Spain');
```

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
INSERT [INTO] { nombre_tabla | nombre_vista } [(lista columnas)]
{ VALUES (expresion [, expresion ]...) | sentencia_SELECT }
```

- *LOW PRIORITY*: hace que la inserción se retrase mientras haya clientes leyendo de la tabla afectada. Sirve solo para tablas que admiten bloqueos, como son las *MyISAM*, *MEMORY* y *MERGE*.
- *DELAYED*: permite continuar con otras operaciones mientras la inserción se retrasa hasta que no haya clientes accediendo a la tabla, es decir, es como la anterior pero permite continuar trabajando.
- *HIGH PRIORITY*: deshabilita el efecto de la variable de sistema *–low-priority-updates* si está activa (=1). Esta variable hace que las operaciones de modificación tengan menor prioridad que las de consulta.
- *IGNORE*: obvia los errores en la inserción. Por ejemplo, no podremos insertar registros con claves repetidas pero tampoco nos informará.
- *INTO*: es una cláusula opcional para indicar el nombre de la tabla o vista.
- *PARTITION*: especifica las particiones donde se pretenden insertar los datos.
- *DEFAULT*: sirve para usar el valor del campo por defecto creado cuando se definió la tabla.

- *Lista columnas*: podemos incluir entre paréntesis grupos de valores para insertar más de una fila en la misma sentencia.
- *Values*: es el conjunto de valores expresados mediante expresiones (normalmente valores literales) o procedentes de una consulta.
- *ON DUPLICATE KEY UPDATE*: cuando el valor de una clave (primaria o secundaria) se repite, ésta cláusula permite actualizar uno o varios campos del registro correspondiente.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
INSERT [INTO] { nombre_tabla | nombre_vista } [(lista columnas)]
{ VALUES (expresion [, expresion ]...) | sentencia_SELECT }
```

Ejemplo:

```
INSERT INTO Northwind.Employees (EmployeeID, LastName, FirstName) VALUES(0,'Munoz','Jorge Juan');
```

¿Qué es lo que hace el 0?

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

En este caso se permite especificar el nombre y valor de las columnas explícitamente con *SET*.

Ejemplo:

```
INSERT INTO Northwind.Employees SET EmployeeID=0,LastName='Munoz',FirstName='Jorge Juan';
```

INSERT INTO SELECT

La sentencia INSERT INTO SELECT se utiliza para copiar datos de una tabla e insertarlos en otra. Es necesario que los tipos de datos de una y otra tabla coincidan. Los registros que ya existan en la tabla destino no se ven alterados.

Si queremos copiar todas las columnas de una tabla a otra:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Si queremos copiar solo algunas de las columnas de una tabla en otra:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Ahora podemos insertar los valores procedentes de otra tabla o vista especificada mediante un *SELECT*.

Ejemplo:

```sql
INSERT INTO employees(LastName, FirstName)
SELECT LastName, FirstName FROM employees
WHERE EmployeeID>=5
AND EmployeeID<=9;
```

INSERT INTO SELECT - EJEMPLOS

```
INSERT INTO customers(CustomerID, CompanyName, City, Country)
SELECT SupplierID, CompanyName, City, Country FROM Suppliers;
```

```
INSERT INTO customers(CustomerID, CompanyName, Address, City, PostalCode, Country)
SELECT SupplierID, CompanyName, Address, City, PostalCode, Country FROM Suppliers;
```

```
INSERT INTO customers(CustomerID, CompanyName, City, Country)
SELECT SupplierID, CompanyName, City, Country FROM Suppliers
WHERE Country='Germany';
```

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.

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Ejemplo:

```
REPLACE INTO Northwind.Employees (EmployeeID, LastName, FirstName)
VALUES(30,'MunozMORERA','Jorge Juan');
```

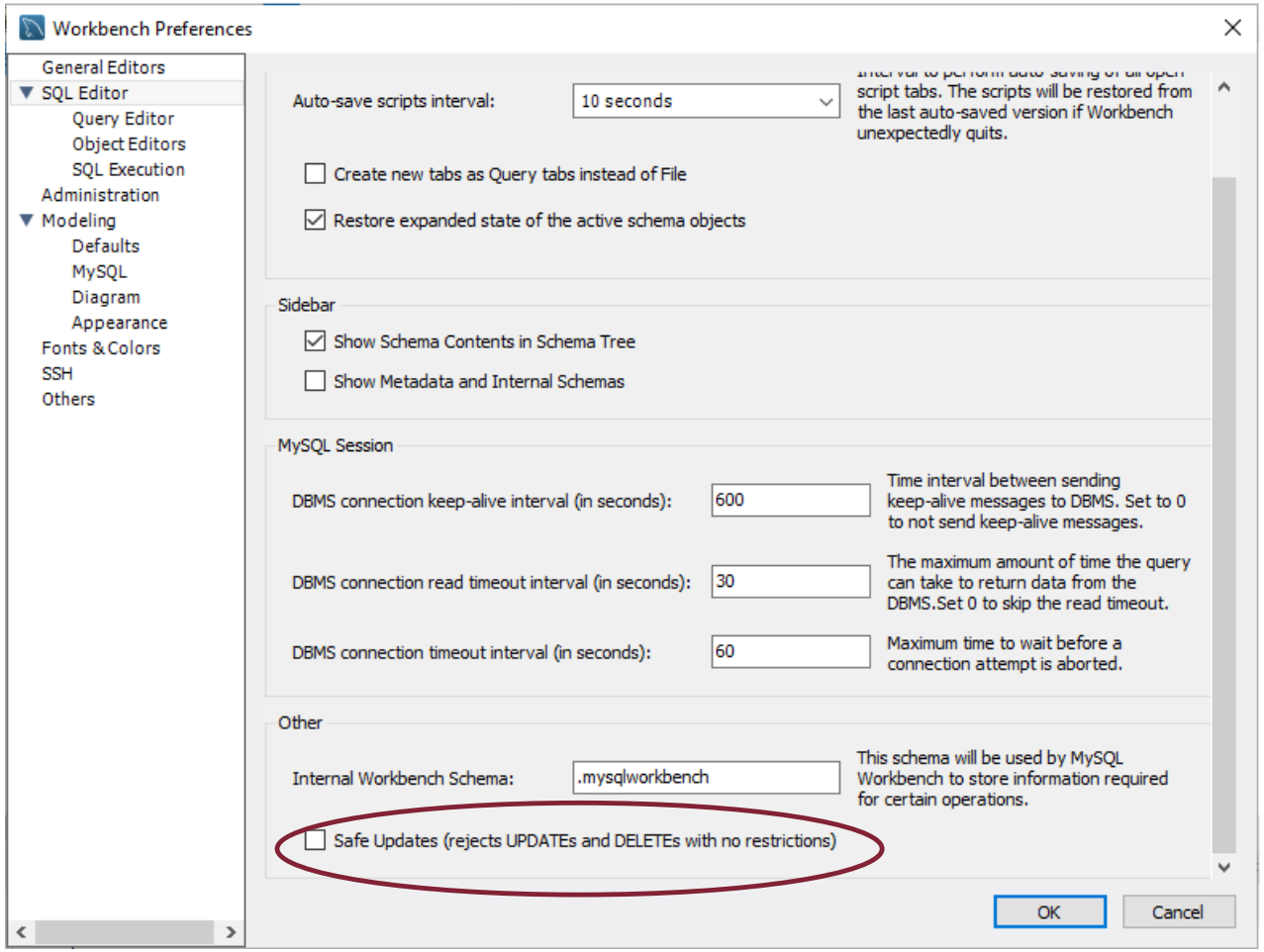
Formato 2

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
```

Formato 3

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
```

DELETE


Workbench Preferences
General Editors
SQL Editor
Query Editor
Object Editors
SQL Execution
Administration
Modeling
Defaults
MySQL
Diagram
Appearance
Fonts & Colors
SSH
Others
Auto-save scripts interval:
10 seconds
script tabs. The scripts will be restored from the last auto-saved version if Workbench unexpectedly quits.
Create new tabs as Query tabs instead of File
Restore expanded state of the active schema objects
Sidebar
Show Schema Contents in Schema Tree
Show Metadata and Internal Schemas
MySQL Session
DBMS connection keep-alive interval (in seconds):
600
Time interval between sending keep-alive messages to DBMS. Set to 0 to not send keep-alive messages.
DBMS connection read timeout interval (in seconds):
30
The maximum amount of time the query can take to return data from the DBMS.Set 0 to skip the read timeout.
DBMS connection timeout interval (in seconds):
60
Maximum time to wait before a connection attempt is aborted.
Other
Internal Workbench Schema:
.mysqlworkbench
This schema will be used by MySQL Workbench to store information required for certain operations.
Safe Updates (rejects UPDATEs and DELETEs with no restrictions)
OK
Cancel

DELETE

La sentencia DELETE se utiliza para eliminar registros de una tabla. La forma general de esta sentencia es:

```
DELETE FROM table_name WHERE condition;
```

Hay que tener cuidado con esta sentencia, puesto que si omitimos el WHERE, borraremos todos los registros de la tabla, pero la tabla, su estructura y sus atributos se mantendrán:

```
DELETE FROM table_name;
```

DELETE - EJEMPLOS

```sql
DELETE FROM Customers WHERE CustomerID='101';
```

```sql
DELETE FROM Customers;
```

¡NO EJECUTAR ESTE!

UPDATE

La sentencia UPDATE se utiliza para modificar registros existentes en la base de datos.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Hay que tener cuidado con esta sentencia, puesto que si omitimos el WHERE, se actualizarán todos los registros de la tabla.

UPDATE - EJEMPLOS

```sql
UPDATE Customers
SET ContactName = 'Juan', City='Frankfurt'
WHERE CustomerID='ALFKI';
```

```sql
UPDATE Customers
SET ContactName = 'Juan'
WHERE Country='Mexico';
```

```sql
UPDATE Customers
SET ContactName = 'Juan';
```

CREATE DATABASE/TABLE, ALTER TABLE

CREATE DATABASE:
https://www.mysqltutorial.org/mysql-create-database/

CREATE TABLE:
https://www.mysqltutorial.org/mysql-create-table/

AUTO_INCREMENT:
https://www.w3schools.com/sql/sql_autoincrement.asp

FOREIGN KEY:
https://www.mysqltutorial.org/mysql-foreign-key/

ALTER TABLE:
https://www.mysqltutorial.org/mysql-alter-table.aspx