Changing data

UNIT 8.

Aim

- Writing queries in order to mantain data stored in a table:
  - Insert new rows in a table.
  - Modify data stored in a table.
  - Delete rows from a table.

INSERT INTO - one row

INSERT [INTO] <target> [**(***column_list***)**]
VALUES **(**{DEFAULT|NULL|*expression*}[**,**...n]**)** [;]

Add one row to an existing table.

- INTO is optional – no additional functionality.
- < target > the name of the table or view in which the row is inserted.
- VALUES introduces the list of values to be inserted.
- Each value can be:
  - Usually a constant or expression,
  - DEFAULT, if the column has no DEFAULT value → NULL.
  - NULL, make sure the column allows null values..
- Assignment is by position.
- The data values supplied must match the column list.

INSERT INTO - one row

- No values for IDENTITY or computed columns.
- Before inserting, integrity rules will be checked.
- We can add a list of column names (column_List) to specify the columns for which data is supplied.
- In the column_list, columns can be in any order and some columns can be omitted.
- A default value (if defined for the column), or NULL is inserted into any column that is not specified in the list.
- If *column_list* is not specified, all the columns in the table receive data.
- The column_List is an asset:
  - It makes it easier to read.
  - The query works even if the column order in the table changes.

INSERT INTO - one row

- Example:

```
INSERT INTO oficinas (oficina, ciudad) VALUES (26, 'Elx');
```

No named columns are filled with default or null value.

If there is an office with primary key 26, an error occurs.

If a value is assigned to a foreign key, it must exist in the referenced table.

```
INSERT INTO oficinas
      VALUES (27,'Móstoles','Centro', null, default , default)
```

INSERT INTO – several rows

INSERT [INTO]<table>[**(***column_list***)**] *select_query* [;]

Using a SELECT query allows more than one row to be inserted at the same time.

- *select_query* is a SELECT without (). It can be any SELECT.
- Each row returned by the query is one list of values to be inserted in the table.
- The select list of the query must match the column list of the INSERT statement.

INSERT INTO – several rows

Example:

```
INSERT INTO trabajo SELECT ciudad, oficina, ventas
                           FROM oficinas
                           WHERE region = 'Centro';
```

SELECT... INTO

SELECT …
INTO *nb_NewTable*
FROM …

To create a new table from values in another table.

- *nb_NewTable* is the name of the table to be created.
- If there is a table with this name, an error occurs.
- Columns in the new table are inherited from the Select result :
  - The datatype of the columns.
  - The name of headers (column name or alias column).
  - But keys and indexes are not inherited.

SELECT... INTO

```
SELECT oficina AS col1, ciudad AS col2, ventas AS col3
INTO trabajo
FROM oficinas
WHERE region = 'Centro';
```

Tip:

Start with the complete SELECT that supplies the values, and then, add the INTO clause.

UPDATE

UPDATE [TOP **(** *expression* **)** [PERCENT]] <Target>
SET {*columna* = {*expression* | DEFAULT | NULL }} [**,**...*n*]
[FROM{ <source> }] [WHERE <condition>] [;]

< Target **> ::=**{ [*BDna***.**[Schna*.*]| Schna**.**]*TableViewna*}

Target is the table to be updated.

Modify the values stored in one or more columns in *Target*.

With FROM < source > we can supply a source based on several tables and have data from another table available for the assignment. If there is no FROM clause, the source is the *Table*.

With TOP, or WHERE we choose the rows to be updated.

UPDATE

- SET *columna* = *value* :
  - Columna is the name of the column to be updated.
  - Value is the value to be inserted in the column:
    - An expression,
    - The keyword DEFAULT,
    - The keyword NULL.
- IDENTITY columns cannot be updated.
- *Expression* in each assigment:
  - Must generate a value with suitable datatype.
  - Can use columns, these must be source columns.
  - If a *table* column is used the value will be the previous value (before updating).
  - The same for the WHERE clause.
  - Can be a scalar subquery.

UPDATE

```
UPDATE oficinas SET ventas = 0;
UPDATE oficinas SET ventas = DEFAULT;
UPDATE oficinas SET ventas = NULL;
UPDATE oficinas SET ventas = 0, objetivo = 0;
UPDATE TOP (10) PERCENT oficinas SET ventas = 0;
UPDATE oficinas SET ventas = 0 WHERE region = 'Este';
UPDATE empleados SET ventas = 0
        WHERE oficina IN (SELECT oficina FROM oficinas
                          WHERE region = 'Este');
UPDATE pedidos SET importe = cant * precio
FROM pedidos INNER JOIN productos
    ON fab = idfab AND producto = idproducto;
```

DELETE

DELETE
[TOP **(** *expression* **)** [PERCENT]]
[FROM]<***Table***>
[FROM <source>]
[WHERE < condition>] [;]

<*Table*> ::={ [*BDna*.[Schna.]| Schna.]*TableViewna*}

Removes **rows** from a table or view.

DELETE

- The keyword FROM before <*Table*> is optional.
- <*Table*> is the target table, the table from which the rows are to be removed.
- TOP y WHERE specifies the conditions used to limit the number of rows that are deleted. If a WHERE clause is not supplied, DELETE removes all the rows from the table.
- When using an external column in the WHERE clause, you can use the FROM <source> clause, but I recommend that you use a subquery, it's more standard.
- Before executing, integrity rules and grants are checked.

DELETE

- DELETE oficinas;
- DELETE FROM oficinas;
- DELETE oficinas WHERE region = 'Este';
- DELETE TOP (10) PERCENT FROM oficinas;
- DELETE FROM empleados
  FROM empleados INNER JOIN oficinas ON
  empleados.oficina = oficinas.oficina
  WHERE region = 'Este';
- DELETE FROM empleados
  WHERE oficina IN (SELECT oficina FROM oficinas
  WHERE region = 'Este');

TRUNCATE

Removes all rows from a table without logging the individual row deletions.

TRUNCATE TABLE [*BDna*.[Schna.]| Schna.]*Tablena*} [;]

It is functionally the same as the DELETE statement with no WHERE clause.

However, TRUNCATE TABLE:

- is faster
- uses fewer system and transaction log resources.

But it has some restrictions, you cannot use it on tables:

- that are referenced by a FOREIGN KEY constraint.
- that participate in an indexed view.