The SQL Language Part VII - Data updating and erasing

1. Introduction

Up to now we have considered, only, selection queries i.e., queries that collect data from the table of a relational database. Selection queries are, by far, the most used ones, yet other kinds of queries do exist.

Actually, it is possible to use SQL to create a whole data base starting from scratch. However, since the basic objects of a Data Base are created in a visual way, in this chapter we will focus exclusively on queries that can be used to: *(i)* create a table, *(ii)* append a record to a table, *(iii)* modify one or more field of one or more records and *(iv)* delete some records.

2. Creating Tables

In order to create a new table the following syntax must be used:

<*command create table*> : : =

CREATE TABLE <*Table Name*>

(<*First Field*> <*Data Type*> (<*Dimension*>) [<*Kind of constraint*>] [{,<*Second Field* > < *Data Type* > (<*Dimension* >)[< *Kind of constraint* >] …}]**)**

Where:

<*Table Name*> is the name given to the table;

<*i-th field*> is the name given to the *i*-th field;

<*Data Type*> is the type of the data that will be stored in the field;

(<*dimension*>) is the max length of the data contained in the field;

<*Kind of constraint*> is the constraint (such as unique) of the data of contained in the field.

The main kinds of constraints that can be declared are shown in Table 7.1

Tab. 7.1. *Kinds Of Constraints*

CONSTRAINT	MEANING
NOT NULL	*The field cannot be null*
UNIQUE	*Each value must be unique*
PRIMARY KEY	*PK → It automatically implies a UNIQUE and NOT NULL constraint*
FOREIGN KEY	*FK → It automatically implies integrity constraints*
CHECK	*Values must fulfill some condition (for instance greater or less than)*
DEFAULT	*A default value is defined and used to fill the field if the user leaves the field blank*

Allowed Data Types are listed in Table 7.2.

Tab. 7.2. *Permitted Data Types*

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

For instance, using the following SQL code, an empty table as the one shown in Table 7.3 is obtained:

```
CREATE TABLE PEOPLE
    (Person_ID Integer, Primary Key, Last_Name, Text (50), First_Name, Text (50), Address, Text (250), _
      City, Text (50))
```

Tab.7.3. *The generated PEOPLE Table*

Person_ID *	Last_Name	First_Name	Address	City

Sometimes it may be useful to create a table T_2 that is a copy (or a partial copy) of an existing one T_1 and that contains only a subset of the records of T_1. To this aim it is sufficient to write a standard selection query adding in the **SELECT** statement the **INTO** keywords, followed by the name given to the new table. The selection query defines the records that have to be inserted in the new table.

For instance the following SQL query creates a new Table called DISC_PRODUCTS that contains all the products, registered in the main PRODUCTS table, that are not sold anymore (i.e., the discontinued ones).

```
SELECT PRODUCTS.ProductID, Products.ProductName, Products.UnitPrice INTO DISC_PRODUCTS
FROM PRODUCTS
WHERE PRODUCTS.Discontinued = True
```

Please note that, in this case, DISPROD is a partial copy of the original table, as it does not contain all the original fields (for instance the Discontinued field is not present in DISC_PRODUCTS)

3. Add Records to an existing table

In order to add one or more records to an existing table it is sufficient to use the **INSERT INTO** operator. Its syntax is the following one.

```
<Insert Command> : : = INSERT INTO <Table Name>
[(Fields List)] <Origin>
```

Where:

<*Table Name*> is the name of the destination table;

[(*Fields List*)] is a list that identifies all or some fields where data must be added;

<*Origin*> defines the values to be inserted.

Concerning the origin, data can be directly included in the statement or they can be taken from another table or from another query. Let us see a couple of example to clarify this concept.

In the first case the insertion is obtained using an explicit list and in this case we have that:

```
<Origin> : : VALUES <(Values List)>
```

For example if we write:

```
INSER INTO CATEGORIES (ID, Name, Description)
VALUES (24, 'New_Category', Null)
```

A new record will be inserted in the CATEGORIES table. Specifically the following value will be used to fill its fields: ID = 24, Name = New_Category and Description = Null.

Also note that, as shown in the syntax the fields list is optional. If the list is omitted, the values will be inserted, sequentially, from the first to the last field. So, the query above could also have been written as:

```
INSER INTO CATEGORIES
VALUES (24, 'New_Category', Null)
```

Clearly, the fields list becomes mandatory if:

- The value list does not includes values for all the fields of the table (it is obvious that if some fields are not included they must be able to accept null values);
- Specifies values in an order that differs from the one of the fields of the table.

As an alternative it is possible to add records starting from records of a table or of a query. In this case <Origin> will be substituted by the name of a Table (i.e., all records will be copied) or by a selection query. In either case the destination table must have, exactly, the same structure of the table (or query) from which data are collected. For instance, the following query add to the NO_DESCR table all the records of the CATEGORIES table having a null description:

```
INSER INTO NODESCR
SELECT *
FROM CATEGORIES
WHERE IsNull(Description)
```

4. Updating one or more fields

The **UPDATE** keywords can be used to modify records that are already stored in a table of a relational data base. Its syntax is as follows:

```
<Update Command> : : = UPDATE <Table Name>
SET <Field Name> = <Expression>
[{,Field Name> = <Expression>} …]
[WHERE <Condition>]
```

Where:

< *Table Name* > is the name of the Table to be updated;

SET <*Field Name*> = <*Expression*> identifies the fields and the values to be updated;

[WHERE <*Condition*>] identifies the records to be updated; if omitted all records will be modified.

Also note that expression can be a single value, an algebraic expression or the result of a scalar subquery.

An example of a simple expression is shown below; the query modify the name of the 23th category and delete its description:

```
UPDATE CATEGORIES
SET Name = 'Seaweed', Description = NULL
WHERE Category_ID = 23
```

Conversely, the following query duplicates the price of the products that are sold at 10€ or less and it also homogenizes (to the minimum) their reorder level; the last point is made using a sub-query.

```
UPDATE PRODUCTS
SET Unit_Price = 2*Unit_Price, Reorder_Level = (SELECT MIN(Reorder_Level) FROM PRODUCTS)
WHERE Unit_Price <= 10
```

We conclude this section with a last example. Here a subquery is used to define the products whose price must be updated (to 10€):

```
UPDATE PRODUCTS
SET Unit_Price = 10
WHERE Unit_Price = (SELECT MIN(Unit_Price) FROM PRODUCTS)
```

5. Deleting one or more records

The **DELETE** keyword makes it possible to cancel one or more records of a table. Its syntax is the following one:

```
<Delete Command> : : = DELETE FROM <Table Name>
[WHERE <Condition>]
```

The elements of the syntax have the same meaning as before; for instance, the following query deletes the record of the product with the minimum price (or the records in case of products with the same minimum price):

```
DELETE FROM PRODUCTS
WHERE Unit_Price = (SELECT MIN(UnitPrice) FROM PRODUCTS)
```

6. A VBA EXAMPLE

Suppose you have a PRODUCTS table containing the information of all the products that are marketed, at present, by a company. This table also have a field called “Discontinued” containing a Boolean field: if the value is true the corresponding product is discontinued and so it is no longer available for sale.

Once in a while it may be wise to clean the PRODUCTS table, getting rid of all discontinued products; yet in a Data Base data should never be deleted (data always have some hidden value), so rather than cancelling out the records of the discontinued products (of the PRODUCTS table) we will append them in a second table called DISCONTINUED PRODUCTS.

To this aim we will make use of the following queries:

```
' Query Name = Create Table and Append
SELECT Products.ProductID, Products.ProductName, Products.UnitPrice INTO [Discontinued Products]
FROM Products
WHERE Products.Discontinued =True
```

```
' Query Name = Append
INSERT INTO [Discontinued Products] ( ProductID, ProductName, UnitPrice )
SELECT Products.ProductID, Products.ProductName, Products.UnitPrice
FROM Products
WHERE Products.Discontinued =True
```

```
' Query Name = Delete
DELETE FROM Products
WHERE Products.Discontinued =True
```

Specifically:

- The first query will be used only once when the PRODUCTS table is “cleaned” for the very first time. This query, indeed, create the DISCONTINUED PRODUCTS table and fills it with all the records of all PRODUCTS table that are labelled as “discontinued”;
- The second query will be used starting from the second update, to append to the DISCONTINUED PRODUCTS table the records, of the PRODUCTS table, labelled as “discontinued”;
- The third query will be launched after all the discontinued products have been appended in the DISCONTINUED PRODUCTS table, to delete the corresponding records from the main PRODUCTS table.

These queries will be launched using the following Form (Figure 7.1)

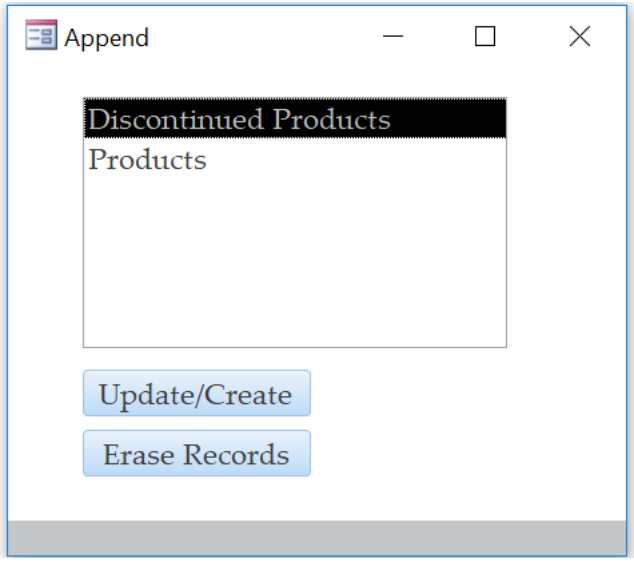

Fig. 7.1. *The Append form*

Briefly, its functioning is the following one:

- The user can select between Discontinued Products and Products in the list Box;
- The list box is filled at start up with the name of some specific Tables among the ones available in the DB. Specifically, these are Products and Discontinued Products.
- If Discontinued Products is selected, both command buttons are available:
    - If the user clicks on Update/Create, all the discontinued products are moved from Products to Discontinued Products;
    - If the user clicks on Erase Records the Discontinued Table is cancelled (this should never been made; this option has been included just for didactical reasons);
    - If Products is selected, only the Update/Create button is activated; as before, if the Discontinued Products table already exist, clicking the button will move all the discontinued products from Products to Discontinued Products. Otherwise, before doing that, also the Discontinued Table will be created.

The full VBA code is shown below:

```
Private Sub CmdAppend_Click()
   DoCmd.SetWarnings False ' No warnings when we execute the query
   If Find_Disc Then ' If the Discontinued Table has been already created, Find_Disc is a function defined next
      DoCmd.OpenQuery "Append"
   Else
      DoCmd.OpenQuery "Create Table and Append" ' The table has to be made
      CmdDelete.Enabled = True
   End If
   DoCmd.OpenQuery "Delete" 'Discontinued products are deleted form the PRODUCTS Table
   DoCmd.SetWarnings True
   Call Find_Tables ' A procedure that searches all the tables of the DB and use them to fill the List Box
End Sub

Private Sub CmdDelete_Click()
   DoCmd.DeleteObject acTable, "Discontinued Products" ' This delete the Discontinued Product table
   Call Find_Tables
   CmdDelete.Enabled = False
End Sub

Private Sub ComboTables_AfterUpdate()
 If Me.ComboTables = "Discontinued Products" Then
   CmdDelete.Enabled = True
 Else
   CmdDelete.Enabled = False
 End If
End Sub

Private Sub Form_Load()
Dim i As Integer
   Call Find_Tables ' To populate the list box
   If Me.ComboTables = "Discontinued Products" Then
      CmdDelete.Enabled = True
   Else
      CmdDelete.Enabled = False
   End If
End Sub
```

Two functions are used to populate the List Box and to check if the Discontinued Products Table has already been created (note that the result of this check will determine the type of update query to be invoked).

The first function explores TableDef (the collection of all the Tables of the Data Base) to see which tables are available; the second one searches through the List-Values of the List Box to see if the “Discontinued Products” Table has already been added or not.

The full code is shown below:

```
Public Sub Find_Tables()
Dim TblNames As String
Dim F As Form
Set F = Forms("Append") ' The form we are operating on
' To keep lines below short, we'll store the quotation mark as a variable named QM Dim QM As String
TblNames = ""
QM = Chr(34) ' this is "
Dim db As DAO.Database
Dim tdf As DAO.TableDef
Set db = CurrentDb
   For Each tdf In db.TableDefs ' ignore system and temporary tables
      If Not (tdf.Name Like "MSys*" Or tdf.Name Like "~*" Or tdf.Name = "USysApplicationLog") Then
         TblNames = TblNames & QM & tdf.Name & QM & ";" 'Creates a list of Tables
      End If
   Next tdf
   With F!ComboTables
      .RowSourceType = "Value List" 'Elements in the List Box are passed as a List
      .RowSource = TblNames ' The previously created list
      .Value = .ItemData(0) ' Select the first value of the list
   End With
Set tdf = Nothing
Set db = Nothing
Set F = Nothing
End Sub

Public Function Find_Disc() As Boolean
Dim F As Form
Set F = Forms("Append")
Find_Disc = False
If F.ComboTables.ListCount > 0 Then
     For i = 0 To F.ComboTables.ListCount - 1 ' Check the names in the List Box to see if Discontinued has already added
       If F.ComboTables.ItemData(i) = "Discontinued Products" Then
          Find_Disc = True
       End If
     Next i
   End If
Set F = Nothing
End Function
```

Also note that the a particular string is created with the following instruction (included in the For Each loop):

```
TblNames = TblNames & QM & tdf.Name & QM & ";" 'Creates a list of Tables
```

Since QM contains a quotation marks (i.e., the symbol") a string like the following one will be generated:

```
“[Name of Table 1]”; “[Name of Table 2]”; … ; “[Name of Table n]”;
```

In this peculiar form, this string can be used (as is) to populate the values of the List Box. This is made with the following code:

```
  With F!ComboTables
     .RowSourceType = "Value List" 'Elements in the List_Box are passed as a List
     .RowSource = TblNames ' The previously created list
     .Value = .ItemData(0) ' Select the first value of the list
  End With
```