

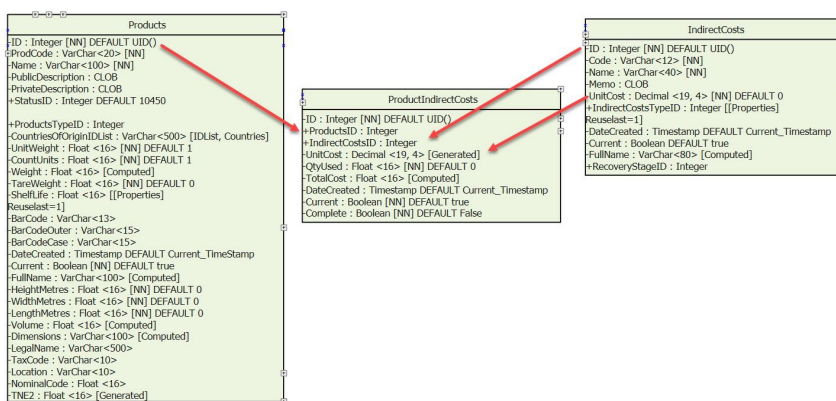
Batch insertion of multiple rows of data in a child table

There are many cases where it can be useful to add one or more rows to a child data-table for every row of a large parent table. For example adding a particular "Indirect Cost" (such as a fixed cost for warehousing, or labour handling) to the Costing of a Product.

In these cases manual data entry is a waste of valuable staff time. SQL statements can be written to automate this process. If the process needs to occur multiple times, these SQL statements can be converted into Stored Procedures, which can be run whenever needed.

The following article shows the steps in this process, including steps to check that appropriate records will be updated. With automated SQL it is very easy for a statement to have unintended consequences, so testing is very important.

Example case study



Products, IndirectCosts, ProductIndirectCosts

A company has a **Products** and an **IndirectCosts** Business Object. Products contains details of Products the Company sells. IndirectCosts includes rows such as "Warehouse handling Fee" and "Inbound Delivery Cost", each IndirectCost has a price, allowing the company to compute profitability for Products. A child data-table **ProductIndirectCosts** contains one or more rows of data showing all IndirectCosts linked to each Product record.

The ProductIndirectCosts data-table includes "ProductsID" and "IndirectCostsID" columns which link directly to the parent data-tables. The "UnitCost" column in the ProductIndirectCosts data-table is **generated**, meaning it is pulled through directly from the UnitCost of the IndirectCosts data-table.

The company sells large numbers of products, and wants to add IndirectCosts to **all** products according to certain categories. For example "export products" might need to be linked to an IndirectCost for export costs.

What is needed is a SQL statement that INSERTs multiple rows in the ProductIndirectCosts table.

Example SQL

Simplest case

```
INSERT INTO "ProductIndirectCosts"
( "ProductsID",
  "IndirectCostsID",
  "QtyUsed"
)
SELECT
  "ID",
  18081,
  1
FROM Products P
WHERE P.StatusID = 10451
```

1. Create INSERT section of script, with name of the child data-table, and list of columns for insertion. Remember you do not need to add columns which are created by the database, or generated by

- computations, so there is no need to add fields such as "ID", "DateCreated" or "QtyUsed".
Note that the list of columns in the INSERT section is surrounded by brackets. Double-quotes around column-names are optional, but are good practice and stop errors occurring where column-names are the same as SQL key-words.
- Create a SELECT section of script. This should ask the database to return a set of records to insert.
In this case just an "ID" from the Products data-table is returned, with a simple number 18081, for the IndirectCostsID. The number of columns in the SELECT statement must exactly match the number in the INSERT section, and the data-types of the data must also match.
- Create a WHERE section of script, this limits the rows returned in the set of records that will be used for insertion. Without a WHERE section, the database will insert a row for **every** row in the Products data-table.

What are the numbers 18081 and 10451?

18081 is an ID in the "IndirectCosts" and 10451 is an ID in the "Status" data-table. What these numbers correlate to is explained below:

The screenshot shows the 'Products data' form for 'ODRC/C8. ORGANIC DR CONGO R&G COFFEE (8x227g)'. The 'StatusID' field is set to '03. Active Sales Item'. A red arrow labeled '1' points to this field. Another red arrow labeled '2' points to the 'Status data ID: 10,451' in the linked record's detail form.

Finding the "ID" for a linked record

- The user has found one sales product, with StatusID "Active Sales Item"
- Clicking on "View" beside "StatusID" opens the Status Edit Form. Here the ID of this record is visible.
This shows that the 10451 = "Active Sales Item" products

The screenshot shows the 'IndirectCosts data' form. The 'IndirectCosts' field is highlighted with a red box and number '1'. Below it, a table lists various indirect costs. The row with ID 18081, Code 'ftf01', and Memo 'FT Licence Fee' is highlighted with a red box and number '2'.

ID	Code	IndirectCost	Memo
18063	rhd01	RH&D Consolidator	16/4/1
18075	store01	Storage Consolidator	16/4/1
18078	org01	Organic Licence Fee	
18081	ftf01	FT Licence Fee	
18090	GoodsIn01	Inbound Delivery	13/6/1
18403	disp02	Multiple Dispatch Consolidator	16/4/1
18418	pick01	Order Picking Consolidator	16/4/1
20757	disp03	General Dispatch	
25089	Zaytoun01	Zaytoun Commission	
310538	oil interest	Canaan Interest	Canaan
344491	OH/N10C	Case rebarcoding Boughey	
352161	HealthCert	Health Certificate Honey	Gepa c
353259	Pallet 500g	Pallet charge 500g	Gepa €
353261	Pallet 2.5kg	Pallet charge 2.5kg	Gepa €

Finding ID for IndirectCosts

- The user has opened the "IndirectCosts" View-Grid.
- They have found the row with the ID 18081. This links to "FT License Fee"
This shows that 18081 = "FT License Fee"

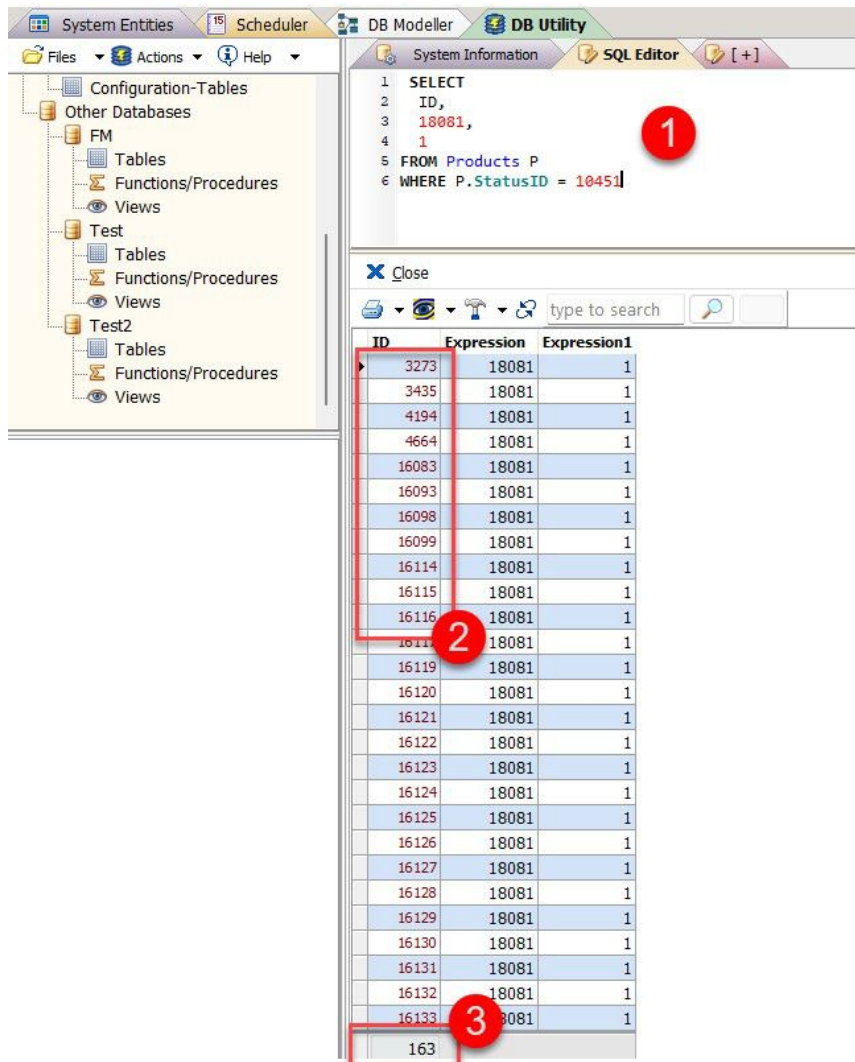
Therefore, running the script will add a Cost for "FT License Fee" to ALL Active Sales Products, in this example this adds 163 records, as shown in the following section.

Important tests and checks

Once you have written the script **always** test the SELECT section prior to running the script. In the above case, the section:

```
SELECT
    ID,
    18081,
    1
FROM Products P
WHERE P.StatusID = 10451
```

Should be run in the Database Management Utility. In the example shown, this results in the following screen:



Testing a SELECT statement for an INSERT Process

1. Test SQL added, by right-clicking and selecting "Show raw data in grid" from the bottom of the menu.
2. The ID's of the Products that will be added are shown. Take some time to check that this list contains records you want. You may have to open Edit Forms for different Products to test this.
3. The total number of rows that will be inserted. This is a useful number to cross check.

The Developer can then review and check that the SELECT statement works, then if it is correct, they can **add** the INSERT section at the top, and run it.

More Complex Case: Multiple IndirectCosts Rows at the same time

```
INSERT INTO "ProductIndirectCosts"
( "ProductsID",
  "IndirectCostsID",
  "QtyUsed"
)
SELECT
    P.ID,
```

```

I.ID,
1
FROM Products P, IndirectCosts I
WHERE P.StatusID = 10451
AND I.ID IN (18081, 18075)

```

The above SQL will add **two** rows in the ProductIndirectCosts table for each Product ID. Note the comma after "Products P". This makes a **cartesian join** to the IndirectCosts table.

As with the earlier case, the SELECT statement can be tested in the Database Management Utility.

Reuse of SQL once written

Just change the "magic numbers"

In SQL of this kind, 18081 and 10451 are sometimes called "magic numbers", as they don't immediately show the details of the records they are linked to.

If the user saves the above script, they can then edit the magic numbers replacing them with other **valid** numbers and re-run the script. If a magic number is used which does not correlate with an ID in the table the SQL will fail with an error.

Change the WHERE statement

Any valid WHERE Statement can be used in these situations. For example the WHERE statement can be extended with references to other columns in the Products data-table, for example the "ProductsTypeID" column, or could include references to another data-table such as SalesInvoiceItems.

The SalesInvoiceItems data-table contains a list of **all products that have ever been sold**. This makes it useful when updating and inserting records related to products, if you want to find Products meeting that criteria.

Example SQL

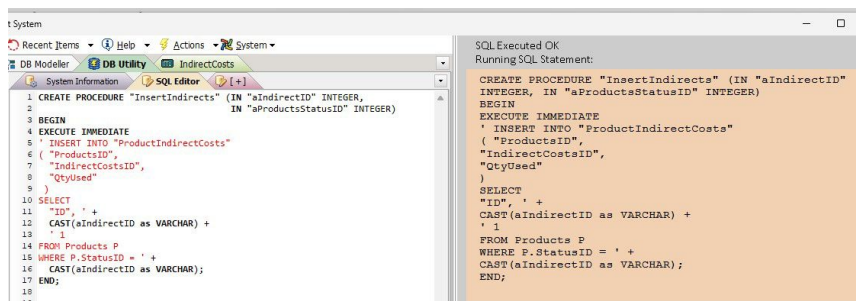
```

WHERE P.ID IN ( SELECT ProductsID FROM SalesInvoiceItems )

```

The AND statements can be extended just as the WHERE statement is extended. In the "multiple rows" example, the section (18081, 18075) could be changed to include any valid SQL, or any other statements relating to either data-table could be added.

Converting the SQL into a Stored Procedure



Convert statement to procedure

SQL Statement

```

CREATE PROCEDURE "InsertIndirects" (IN "aIndirectID" INTEGER,
                                     IN "aProductsStatusID" INTEGER)

BEGIN
EXECUTE IMMEDIATE
' INSERT INTO "ProductIndirectCosts"
  ( "ProductsID",
    "IndirectCostsID",
    "QtyUsed"
  )
SELECT
  "ID", ' +
  CAST(aIndirectID as VARCHAR) +
  ' 1
FROM Products P
WHERE P.StatusID = ' +
  CAST(aIndirectID as VARCHAR);

```

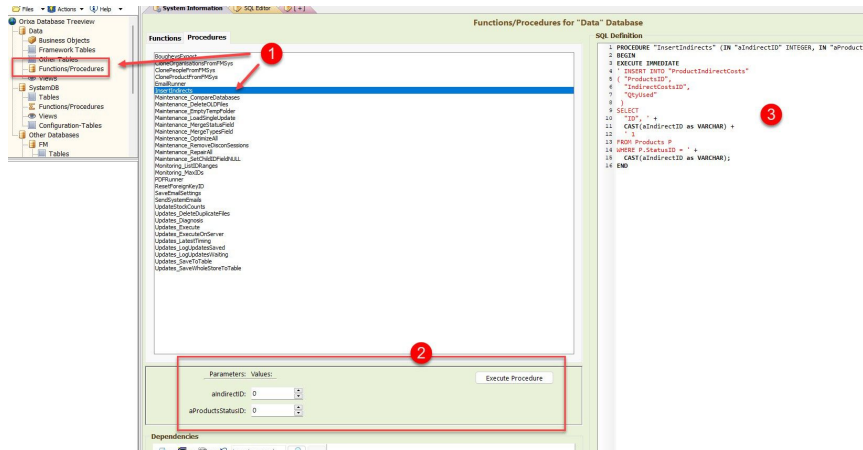
END ;

The above SQL takes the statement we have been using earlier, and converts it into SQL Script which can be run as a procedure.

Note that the EXECUTE IMMEDIATE statement must be used prior to calling SQL within a procedure, and the SQL statement must be enclosed in single-quotes.

The SQL is changed by adding two **parameters** to the procedure. "aIndirectID" and "aProductsStatusID", if you look at the SQL in detail you can see that these parameters are passed into the SQL by adding single quotes, and remembering to CAST the variables to the correct data-type so they can be read as part of the EXECUTE IMMEDIATE statement.

Running the stored procedure you have created



Newly created stored procedure

1. As soon as the SQL is run to create the procedure, it will appear in the "Procedures" tab of the Database Management Utility.
2. To Execute the procedure, type correct values into the fields shown for each parameter, and click "Execute Procedure."
3. The detailed SQL of the procedure is shown on the right, which you can read to understand the procedures functionality.

Because the SQL has been tested, once a procedure is created it is not usually necessary to check the outcome prior to running it.

Undoing / Deleting wrongly inserted data

It is quite common to run an INSERT statement and then find that multiple rows of data have been added wrongly.

In this case the user will want to remove these incorrectly added rows. To do this:

1. Open the ViewGrid for the BusinessObject containing the badly inserted data.
2. Find the range of IDs (minimum and maximum) for these records. As the statement occurred as a single SQL operation these IDs should be continuous, ie 101, 102, 103, 104 ...
3. Run a DELETE statement using this set of IDs

Example Delete statement

```
DELETE FROM ProductIndirectCosts  
WHERE ID BETWEEN 101 AND 104
```

In the above, replace "101" and "104" with your own minimum and maximum values.