

## Tutorial 6 - Building the Order Report Procedure

The Order Report Procedure takes one input parameter from the user: the Order number. This is similar to Tutorial 4, where the report took the customer ID at the run-time, and the technique of setting the Parameters component is the same.

To display all required data about the given order, the Report Procedure must search a number of tables, extracting info on the product name, quantity and price, shipping company and date etc. We could use the same approach as in the Customer report, which retrieved data first from the Customers table, then - in a separate trip to the database server - from the Orders table. In the case of Order report, the procedure would retrieve the following data:

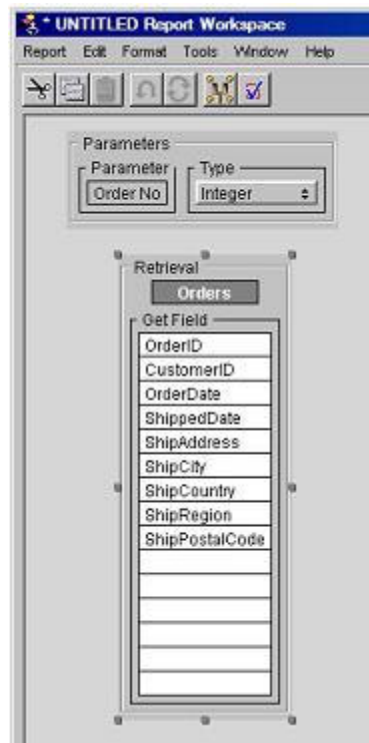
- |   |   |   |   |
|---|---|---|---|
| 1. From Orders table:   | 2. From ShippersTable:  | 3. From Order_Details table:  | 4. From Products table:                                       |
| <ul style="list-style-type: none"><li>• OrderID</li><li>• OrderDate</li><li>• CustomerID</li><li>• ShipAddress</li><li>• ShipCity</li><li>• ShipCountry</li><li>• ShipPostalCode</li><li>• ShipRegion</li><li>• ShipVia</li><li>• ShippedDate</li></ul> | <ul style="list-style-type: none"><li>• CompanyName</li></ul> | <ul style="list-style-type: none"><li>• Discount</li><li>• ProductID</li><li>• Quantity</li><li>• UnitPrice</li></ul> | <ul style="list-style-type: none"><li>• ProductName</li></ul> |

This would require four SELECT statements to be executed by the procedure:

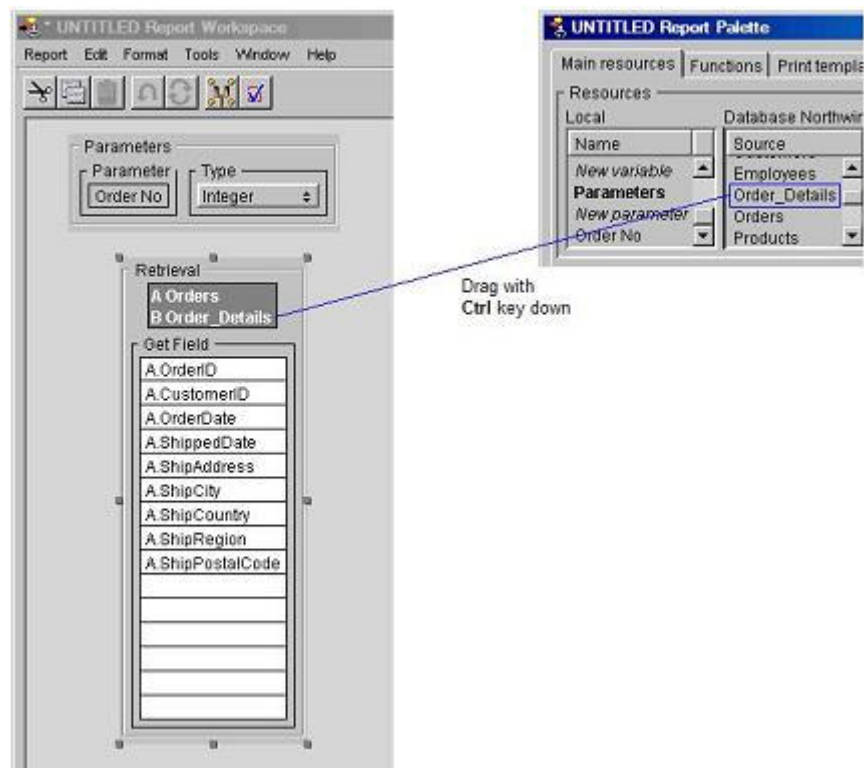
1. select  
OrderID,OrderDate,CustomerID,ShipAddress,ShipCity,ShipCountry,ShipPostalCode,ShipRegion,ShipVia,S  
from Orders where OrderID = [Order No]
2. select CompanyName from Shippers where ShipperID = [Orders.ShipVia]
3. select Discount,ProductID,Quantity,UnitPrice from Order\_Details where OrderID = [Order No]
4. select ProductName from Products where ProductID = [Order\_Details.ProductID]

That's a lot of SELECTs for a small and simple report like Order. However, Scribe offers a more efficient way of getting data from multiple related tables. Instead of bringing in four Retrieval components and doing the data retrievals in a piecemeal fashion (inefficient and needlessly taxing the database server), we will use the [Join](#) feature of the Retrieval component, described in Appendix B.

We start by setting the Parameters component to "Order No" parameter of Integer type (to set the data type, double-click on the component, then click on the data type pop-up list). Drag in the Retrieval component, expand it to 15 field slots, set the table name to Orders and load the fields as shown in the figure below:



Now, **hold down Ctrl key** and drag Order\_Details table name into the table name field:

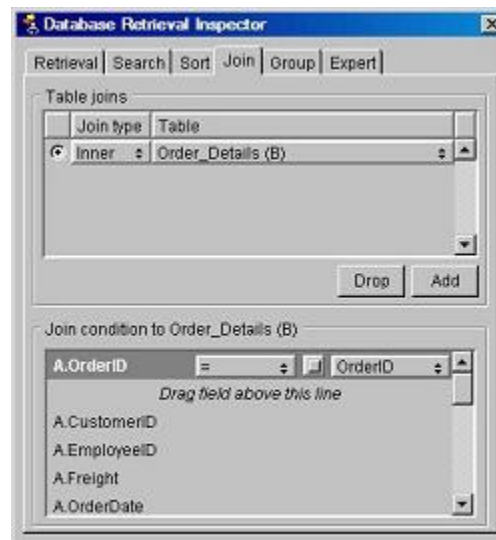


Observe what happened to the Retrieval component:

- The names of tables in the table field have been assigned an alias (letters A and B)
- All fields already in the component have been qualified with the alias (A)

The alias helps distinguish between fields that belong to different tables, ensuring the uniqueness of the alias-qualified field names.

Let us have a look at the Inspector window (for the illustration purposes, the window was made wider to show the entire contents of its views):



First of all, a new tab was added to the window: Join. Its "Table joins" box shows one join of INNER type; the joined table is Order\_Details (the primary table that Order\_Details is joined to - Order - is implied). The bottom box ("Join condition...") shows the join condition that is used to establish the join between two tables. In our example the join condition was set automatically, using Scribe's ability to identify the best candidate fields for joining:

- OrderID is a primary key field in Order (A) table
- OrderID in Order\_Details (B) table is a foreign key that references OrderID field in Order table

From these field attributes, Scribe concluded that the two fields should be used for joining the tables. If there were no related fields in Order and Order\_Details, Scribe would leave the task of identifying the join fields to the user who would manually drag the desired A-table field up and select its B-table counterpart from the pop-up list.

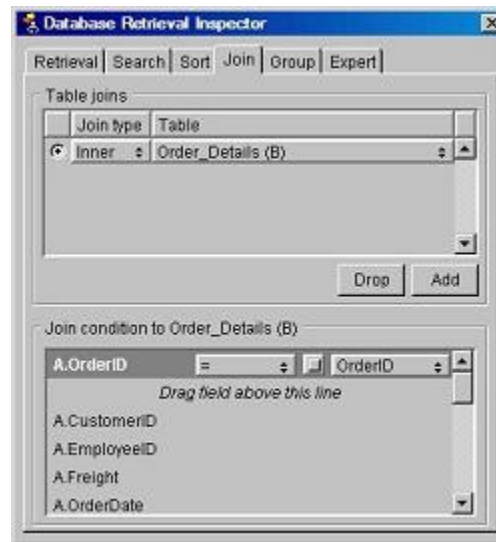
Back to the Retrieval component: load the following fields from Order\_Details table into it by clicking on (or dragging in) the following fields:

- Discount
- ProductID
- Quantity
- UnitPrice

We have two more tables to join: Shippers and Products. We add them the way we added Order\_Details table: by Ctrl-dragging the table names into the table field area of the Retrieval component. The join between Order and Shippers table should be done on the basis of ShipVia field, which stores the shipper's ID, and is thus related to the ShipperID field in Shippers table. However, ShipVia field is not a primary key field in Order table, and the spelling of its name differs from its counterpart in Shippers table, so Scribe does not try to create the join condition automatically, and expects you - the user - to set it.

Drag the A.ShipVia field up in the "Join condition..." box; the field joined to it on Shippers table side is ShipperID (the pop-up list is set to it as this is the only field that could possibly be used for joining, based on the field data type).

Load the remaining two slots in the Retrieval component with fields Shippers.CompanyName and Products.ProductName (they will show as C.CompanyName and D.ProductName) and set the Search criteria to retrieve data for the given order number:

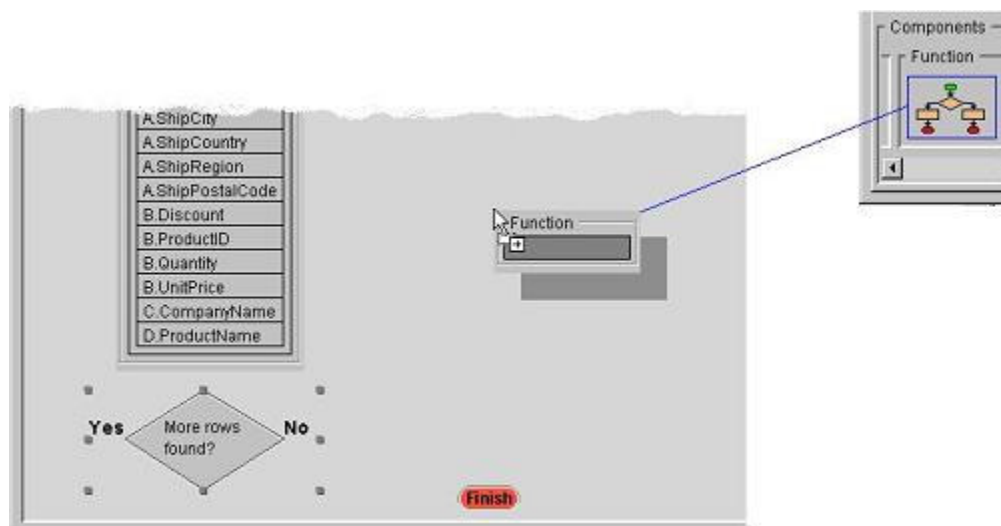


It is worth taking a look at the Expert view of the Retrieval Inspector. The SQL SELECT statement formatted by the component looks like this:

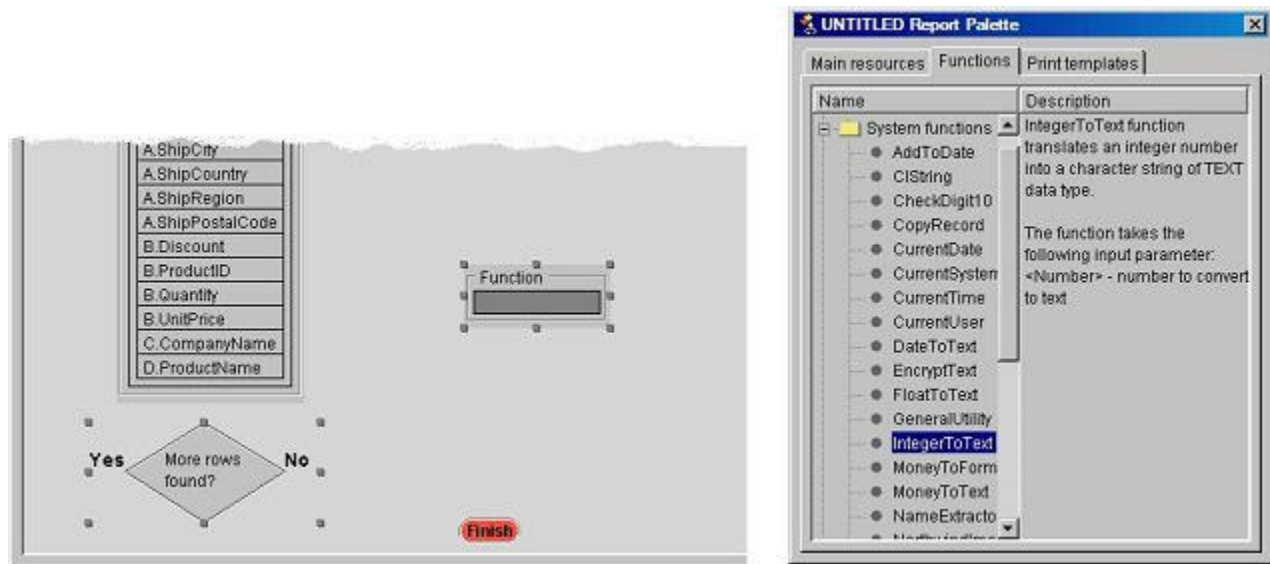
```
select A.OrderID, A.CustomerID, A.OrderDate, A.ShippedDate, A.ShipAddress,
A.ShipCity, A.ShipCountry, A.ShipRegion, A.ShipPostalCode, B.Discount, B.ProductID,
B.Quantity, B.UnitPrice, C.CompanyName, D.ProductName from Orders A JOIN
Order_Details B ON A.OrderID = B.OrderID JOIN Shippers C ON A.ShipVia = C.ShipperID
JOIN Products D ON B.ProductID = D.ProductID where A.OrderID = [Order No]
```

Here one can clearly see the formula of joining four tables together that was until now wrapped in a graphical interface of the Retrieval component.

We are now ready to format the report heading. The heading should include the order number and the customer ID that made the purchase. All this info is returned to use from the Retrieval component, but the order number is of INTEGER type, and we want to make it part of the TEXT string. To avoid being caught by a syntax check, we have to convert the INTEGER value of the order number to its textual equivalent of TEXT type. This is done by a built-in system function named IntegerToText.



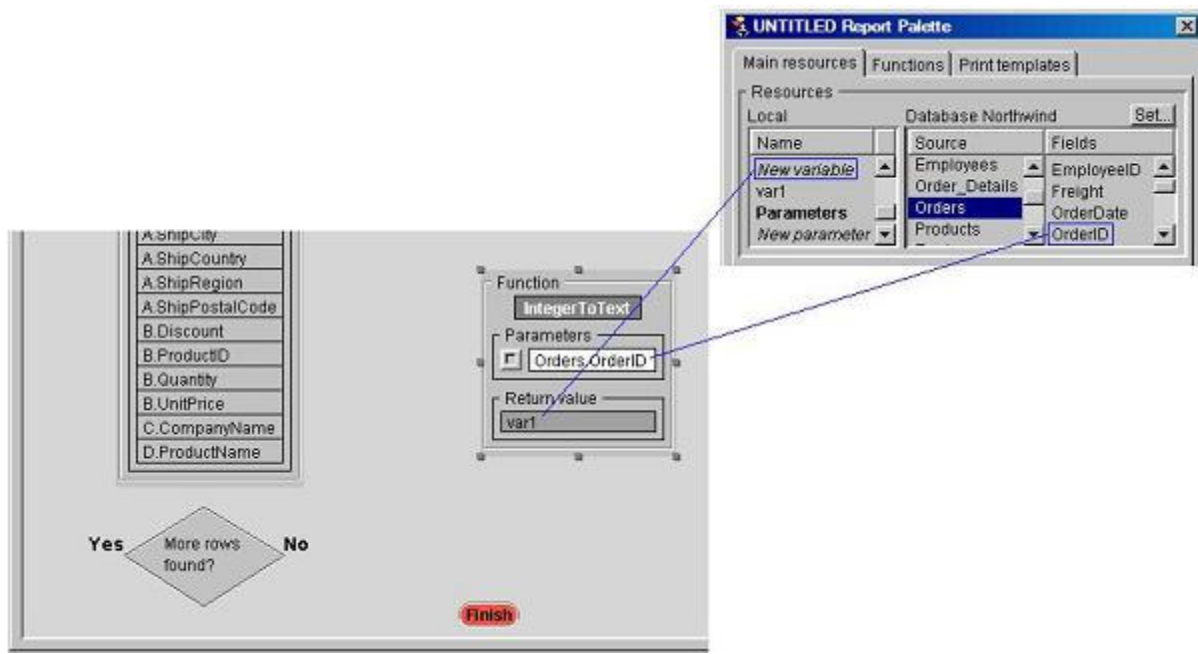
Once the Function component lands in the Workspace, the Palette's Functions tab is selected, and the Palette shows a list of resources that can be used as a built-in function in the report procedure. Initially, the list is shown as an expandable tree; click on the "System functions" node and, once it expands, click on the IntegerToText entry. The first click causes the Palette to display the function description in the right-hand side of the window; as shown in the following figure:



The second mouse click on the function name loads it into the Function component, and it is resized to show the new content, specific to the selected function:







This function takes one parameter - a variable (or a table field) of the INTEGER type, and returns its character representation - a variable of TEXT type, suitable for use in formatting text strings. Drag OrderID field from Orders table into the parameter slot (make sure the Function component is selected first), and drag the *New variable* entry from the Palette's Name column into the "Return value" field of the Function component:



Note the following:

- The OrderID field is shown in the Function component as Orders.OrderID. Outside the Retrieval component, table fields name are always fully qualified, and the qualifier is the actual table name, and not the alias (A, B etc.) assigned by the Retrieval component.
- *New variable*, upon landing in the Function, is translated into the next available generic variable name "varN". It is added to the Name column of the Palette, and can be renamed to something more meaningful by editing it in the Palette (see Tutorial 4 for example). We rename "var1" to "order #".

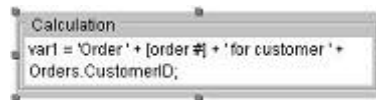
Now we have everything needed to format the report heading. We do this in the Calculation component. Drag it into the Workspace, make sure it is selected, and click on *New variable* entry in the Palette. If you renamed "var1" as suggested in the previous paragraph, the "var1" will be used again, otherwise Scribe will insert "var2" into the Calculation component. From this point on, follow these steps:

- Click on  icon in the Palette's "Operators and dividers" box.
- Type in 'Order ' in the Calculation component, after the equal sign. Do not miss a blank space before the closing quote.
- Click on  icon in the Palette, or just type in the plus.
- Click on "order #" variable in the Palette. It will be added to the Calculation component, enclosed in square brackets: [order #].
- Add another .
- Type in ' for customer ', with blank spaces separating quotes from the words.
- Add another .
- Click on CustomerID field in Orders table.
- Type (or drag in from the Palette) a semicolon (;).

Blank spaces separating operators (in this example, the "equal" sign and the "plus" signs) are optional, and serve to improve the readability of the expression, and are ignored by Scribe at the run-time. Blank spaces inside the literal constants (enclosed in single quotes) are part of the literal, and are obeyed as set in the text string.

The "plus" sign is an arithmetic operator, and means addition. In the context of numeric operands it works as usual, by adding the arithmetic values together. However, when the operands are of TEXT type, the "plus" becomes a concatenation operator, stringing the operands together into one text string. You should now have a

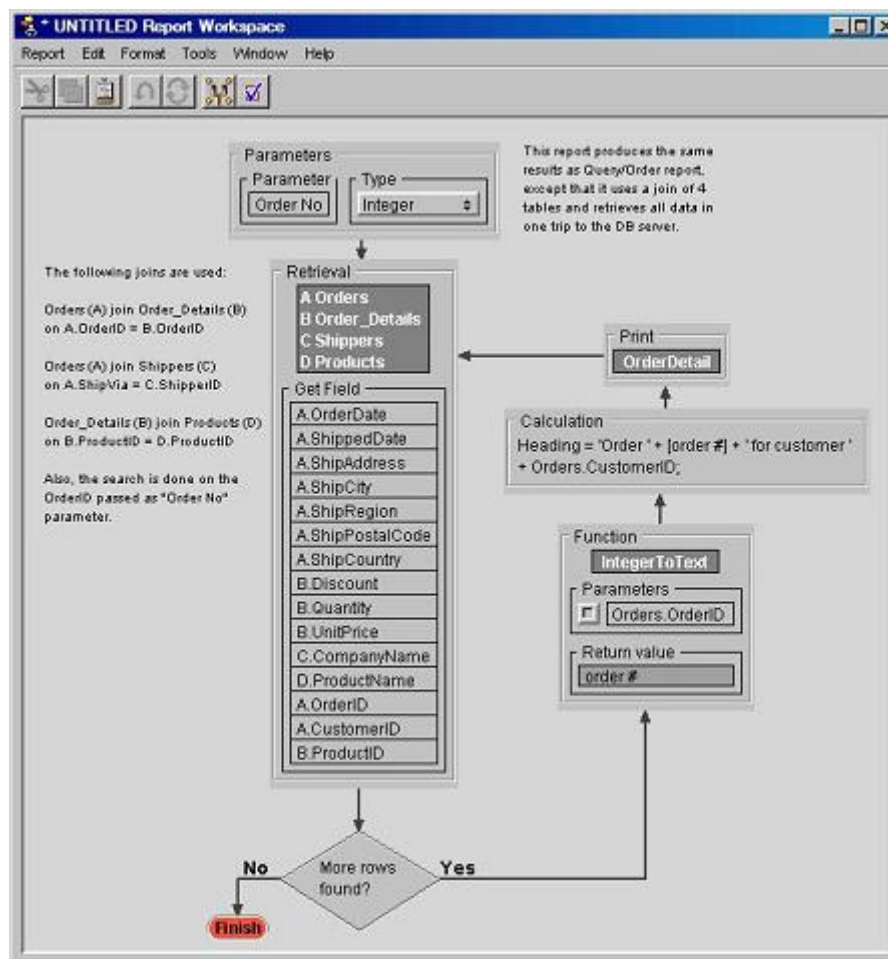
Calculation component with the following content:



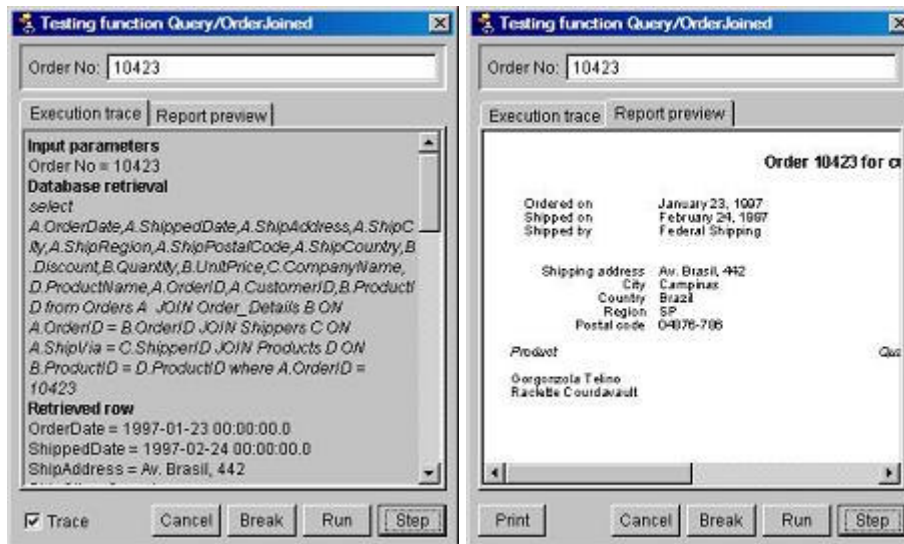
(note the closing semicolon in the Calculation expression. Every expression must end with the semicolon. Semicolons allow multiple calculation expressions to be stored in one Calculation component, saving valuable real estate in the Workspace, and keeping logically related calculations together).

Finally, let us rename the variable "var1" to "Heading". This will help us resolve the link to the OrderDetail print template in the next step. Before renaming the variable, which requires double-clicking on its name in the Palette, deselect the Calculation component, or else every click on the variable name will add it to the expression, at the location of the I-beam.

The last component to be placed into the procedure is Print. We load it into the Workspace and set to OrderDetails print template. The resulting Workspace should now look like in the following figure:



Before saving the report, we can test it:



Save the report as Query/Order.