

Reviewing How a Data-Field Computation works in Orixa

Orixa uses Functions within the definitions of a data-table to enable automation of computation of the values of certain data-fields. This allows much of the data-entry process to be stream-lined, making it quicker and easier for users.

This process is just a normal part of SQL Syntax and is common across very many types of database. The SQL Key Words "COMPUTED" and "GENERATED" are used as part of a column definition. For these columns the values they contain will then be set by the statement which follows "COMPUTED AS" or "GENERATED AS" in the data-tables definition.

Often this value is a SUM or AGGREGATE of values in a child data-table, or a look-up of a value (such as a Price) in a Linked data-table. To access these values, the database calls a FUNCTION, as is explained below.

A worked example of Data-field Computation

Example: An Orixa System includes a "Deliveries" data-table, where Products are delivered and the company pays an DeliveryCost, based on the Kilo weight of the delivered items.

The Data-definition of the "Deliveries" data-table (simplified)

```
CREATE TABLE "Deliveries"
( "ID" INTEGER DEFAULT UID() NOT NULL,
  "WayBillNum" VARCHAR(20) COLLATE "ANSI",
  "FarmersID" INTEGER,
  "ProductsID" INTEGER DESCRIPTION 'Custom, Products, DeliveryProductsList',
  "Units" FLOAT DEFAULT 0 NOT NULL,
  "Kg" FLOAT DEFAULT 0 NOT NULL,
  "UnitPrice" DECIMAL(19,4) GENERATED ALWAYS AS IF(Complete = true THEN UnitPrice ELSE
  ProductPurchasePrice(ProductsID)),
  "ProductCost" DECIMAL(19,4) COMPUTED ALWAYS AS Kg * UnitPrice,
  "SupervisorID" INTEGER DESCRIPTION 'ReuseLast',
  "DeliveryCost" DECIMAL(19,4) GENERATED ALWAYS AS IF(Complete = true THEN DeliveryCost ELSE
  ProductTransportCost(ProductsID) * Kg),
  "DateCreated" TIMESTAMP DEFAULT Current_Timestamp NOT NULL,
  "Complete" BOOLEAN DEFAULT false NOT NULL
)
```

Notice in this data-table

1. ProductsID includes a definition for a custom look-up, so that specific products are available in the pick-list to choose for delivery, but the table actually stores the ID (a whole number).
2. The **UnitPrice** is generated automatically from the price set for the Product, using the "ProductPurchasePrice" function.
3. **ProductCost** is UnitPrice * Kg.
4. **DeliveryCost** is generated using the ProductTransportCost function.

Very importantly, notice that in the above example all GENERATED columns do NOT run if the record is marked "Complete", this is important, because Products.Price and Products.TransportCost may vary over time. If a Deliveries record is edited after the price has changed the GENERATED columns would re-compute. Therefore it is important that once the data entry is complete the record is marked "Complete" this will "lock" its values, so that recomputation cannot occur.

Details of the "ProductTransportCost" Function

```
FUNCTION "ProductTransportCost" (IN "aID" INTEGER)
  RETURNS FLOAT
BEGIN
  DECLARE Crsr CURSOR FOR Stmt;
```

```

DECLARE Result Float;
PREPARE Stmt FROM
' SELECT
    TransportCost
FROM Products
WHERE ID = ? ';
OPEN Crsr USING aID;
FETCH FIRST FROM Crsr('TransportCost') INTO Result;
RETURN Result;
END

```

The function uses a SQL CURSOR and STATEMENT This is made available in the function through the line:

```
DECLARE Crsr CURSOR FOR Stmt;
```

The above line tells the Function that the following code will contain SQL Statements (held in the variable Stmt).

Calling "OPEN Crsr" will cause the Statement to be run, and will make the data contained available through code such as
 FETCH First FROM Crsr('FieldName') INTO VariableName.

Note how a FUNCTION has a Result (which itself should be declared at the start of the Function), and should end with the line:

```
RETURN Result;
```

To pass the Result back to the receiving process.

If you read the above ProductTransportCost Function you can see that it declares a short SQL Statement to return the current value of the "TransportCost" field in the Products data-table. The function passes in the ProductsID, and this is passed in as a parameter at the point where the "?" occurs in the statement.

```
OPEN Crsr USING aID;
```

The "aID" variable appears in the first line of the Function as an "IN" Variable. This means it is passed by the calling code.

Once the Crsr is OPEN, and we have called FETCH FIRST, we have found the Transport cost and put its value into the Result.

This is returned to the Deliveries data-table, where it is multiplied by the value of the "Kg" field to generate the DeliveryCost.

Using Orixas Tools to do your own review of field-level computations in your App

Find the definition of the data-field you are checking

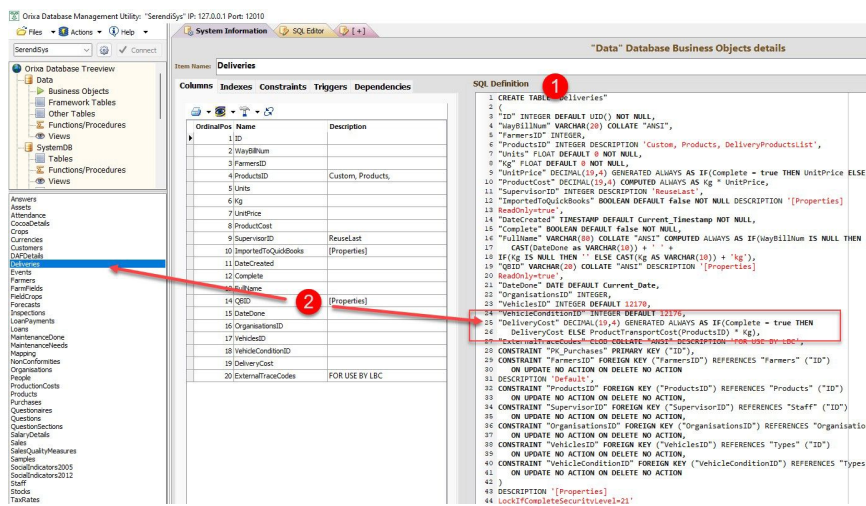
The screenshot shows the Orixas Database Management Utility interface. On the left, the 'Business Objects' tree is visible, with 'StockPurchaseItems' selected. The main window displays the 'Columns' tab for this object. A table lists the columns with their names, descriptions, data types, lengths, and generation rules. The 'TransportCost' column is highlighted in red, and its generation rule is shown as 'COMPUTED: IF(Complete = true then COALESCE(TransportCost, 0) ELSE PurchasePrice * QtyOrdered)'. The 'GenCompDefault' column shows the computed value for this field. On the right, a 'Data for: 353364 (BDMte: 2023-06-30/0000/C3 ORIXAS)' window shows the data for the 'StockPurchaseItems' table, with the 'TransportCost' field highlighted in red.

DB Utility Viewing Field-Generation Definition

1. Open the Orixas Database Management Utility, and select the "Business Objects" heading of the Database Treeview.
2. Click on the name of the Business Object. In the example above the "StockPurchaseItems" Business Object has been selected.
3. Find the field that you want to analyse. The GenCompDefault column of the "Columns" grid will show details of the COMPUTATION or GENERATION for the particular field. An Edit Form for the "StockPurchaseItems" record is open showing the COMPUTED value. Note that the field is shown in a dark-brown colour, indicating that it is COMPUTED, and therefore cannot be edited by staff.

When Orixas systems become more complex, COMPUTATION and GENERATION processes can become hard to untangle. For example the COMPUTATION may include references to other fields in the data-table.

Reviewing the definition of a data-field using the SQL Definition



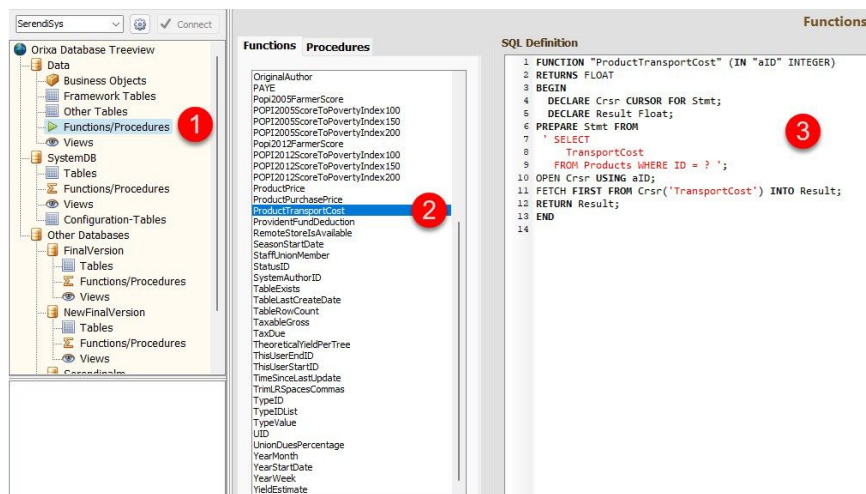
SQL Definition in DB Utility

The same information can also be viewed in the "SQL Definition" of the table.

1. SQL Definition is always displayed on the right hand side of the DBUtility when a data-table is being viewed.
2. Click on the desired data-table and then find the definition of the particular data-field. Details of the function that it references can be seen.

Finding the Function that has been referenced in the data-field definition

Once you have discovered the particular function which defines the generation of a data-field, you can then check its SQL definition to understand how it works.



Finding a specific function definition in the DB Utility

1. In the Database Management Utility click on the "Functions/Procedures" heading for the main "Data" database.
2. Find the function's name, and click on it.
3. The SQL Definition will display on the right hand side of the screen.

Once you have completed this process you can see that the DeliveryCost in the Deliveries data-table is generated using a "TransportCost" value in the Products data-table. If the TransportCost is zero, then the DeliveryCost will not be generated correctly. Also, if the data-record is marked "complete" the value will not recompute. This is important as it means that once a record is "Complete" computation of the TransportCost will not be triggered again which guarantees that the value will be permanently fixed regardless of any updates to the database.