

Tutorial
Orders Management
Intranet



Table of contents

| | |
|--------------------------------------|----|
| Introduction | 3 |
| Plan the Orders Management Intranet | 4 |
| Build site front-end | 7 |
| Implement user authentication | 10 |
| Display available products | 13 |
| Add products to orders | 16 |
| Complete order process | 22 |
| Process User Reply for Order | 37 |
| Create page to display current order | 40 |
| Manage Orders | 44 |
| List, sort and filter orders | 45 |
| Modify order status | 53 |
| Other Resources | 56 |
| Copyright | 57 |

Introduction

Here you will create a web application which will allow the creation and management of orders. Users will have the possibility to register to your web-site, and then choose from the list of available products the ones to add to their order. In the products list, a link allows adding a product to the cart, and also specify the number of similar items to add.

Besides adding items to a cart, users have two options: to clear the order (cart) or to complete it. By clearing the order, all details regarding it are lost. It is erased from the temporary session variables, as well as from the database. By clicking on the complete link, the order status will change in the database from initiated to pending, thus becoming visible to the site administrators.

An order can have 5 different states:

1. initiated - the ordering process has started, but has yet to complete. An order will remain in this state until the user will decide to complete it.
2. after the user clicks the complete order link, there are two possibilities, each imposing a different state for the order: if there are enough products on stock to satisfy the entire order (all ordered products exist in the desired quantity), the order goes into pending. It can be viewed by site administrators, and forwarded to the Shipping department.
3. If there are not enough stocks to fulfill the order entirely, an e-mail is sent to the user, offering two options: either to cancel the order, or to wait for a new transport of the desired goods. While the user reply is awaited, the order state is Waiting for acknowledgment.
4. If the user chooses to cancel the order, it will be deleted from the database, together with all the associated products. If the user decides to wait, then the order state is changed to acknowledged, and becomes visible for site administrators.
5. The last state an order can have is processing, which means that the order has been forwarded to the Shipping department.

Regarding the stock calculations, the ordered quantity is subtracted from the available quantity only if all ordered products can be delivered. In this case, the order goes into pending, and the stock is diminished by the amount ordered.

In this tutorial you will construct both sides of this application: a front-end that allows users complete their orders, receive e-mails and take decisions, and the administrative section, allowing site administrators monitor and manage orders. User and product management is not covered in this tutorial, as it is done in a similar manner.

If you have the **MX Kollection 3** bundle installed, then you have all the needed tools. Otherwise, the following separate products should be installed on your computer in order to complete the tutorial:

- **ImpAKT** - for all of the basic database operations.
- **MX Send E-mail** - to be able to send e-mail messages from the site.
- **MX User Login** - to implement the user authentication.
- **MX Includes** - to display content from different pages in the same file.
- **MX Query Builder**
- **MX Breadcrumbs** - to create a navigation trail, on top of your pages.
- **MX Tree Menu** - to create the category listing as a tree menu.

The estimated completion time for this tutorial is about 200 minutes. It depends on your authoring knowledge with **Macromedia Dreamweaver** (MX or MX 2004) and **MX Kollection 3**.

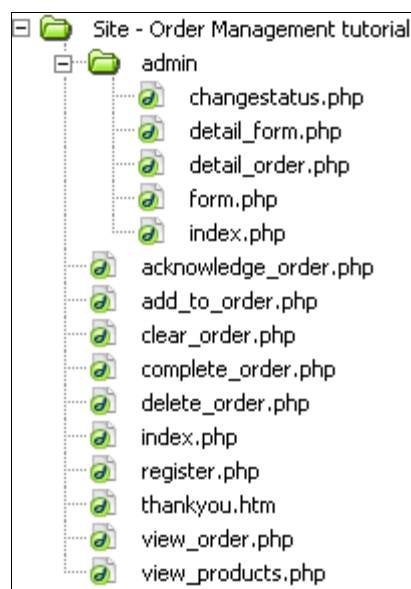
It's recommended that you follow this tutorial in the intended order to avoid potential problems.

Plan the Orders Management Intranet

This first section of this tutorial helps you create all the files and database tables needed for the application.

Before you start building this application, make sure you have a correctly configured **Dreamweaver** site, and a working database connection. For more instructions regarding these actions, consult the *Getting started* help file, which can be found in **Help -> InterAKT -> Getting Started**.

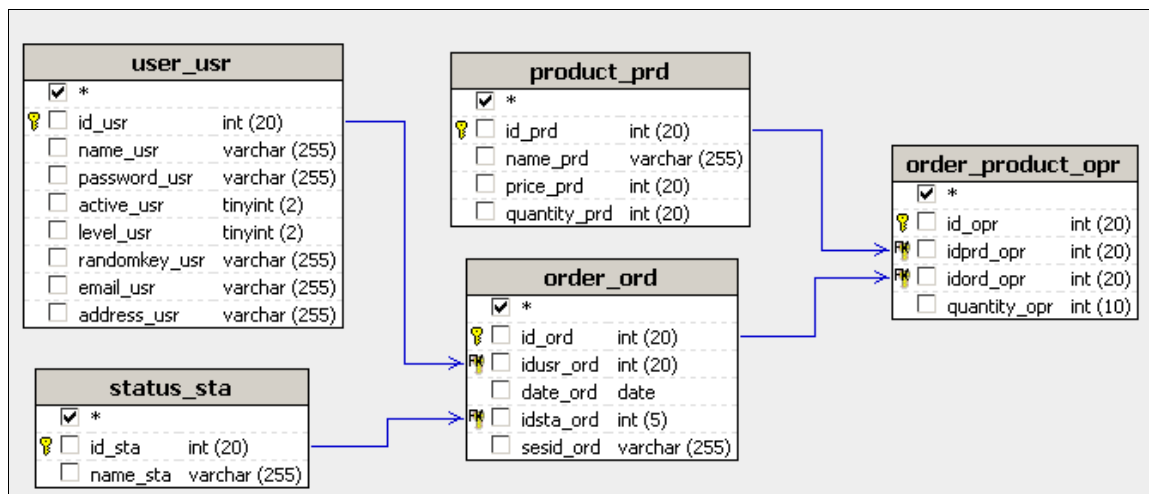
Through the tutorial, you will have to create several files in your site's root. You can create them at the very beginning, so that you will not waste time with this operation again. To create files and folders in the site's root, use the corresponding options in the **File** menu of the **Files** tab. All files created in this tutorial can also be found inside the downloaded package, and you can use them to compare your work with what has been already done. The file structure will look as in the example below, and you can create it easily by unpacking the *zip* file corresponding to your server model from `\tutorials\Orders Management\` in your site root:



The use of each file will be explained upon building its content.

Besides the files and folders seen in the image above, the final application will contain some more, generated by **Dreamweaver** and **MX Kollection**. For example, the *forgot_password* and *activate* pages will be generated automatically by the user authentication system, while the connection folder is created by **Dreamweaver**.

After having created the files for your pages, it is time to set up the database that will hold the information to display. For this tutorial you will use several tables containing various data regarding the ads and users. The database structure is as displayed in the image below:



Note: The database diagram in the image above was built with **MX Query Builder** (also referred as **QuB**) to better illustrate the database structure. You do not need to build it in order to complete this tutorial.

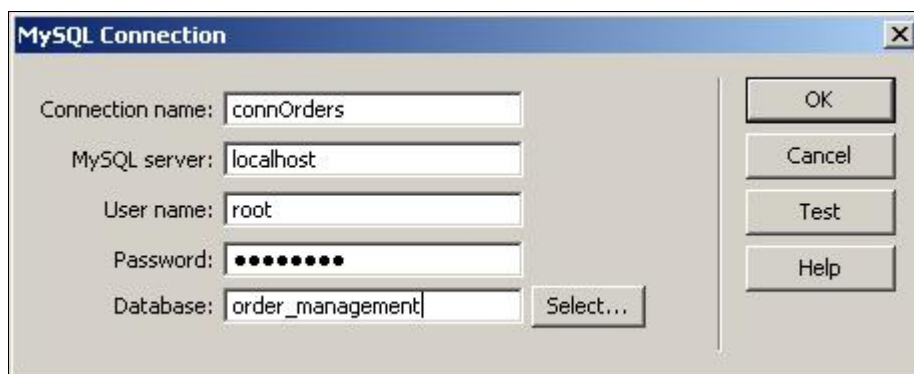
Each table and column's importance will be detailed in what follows:

- The `user_usr` table stores information regarding the user. Its fields are used for:
 - `id_usr`** - primary key, provides the unique identifier for each record
 - `name_usr`** - the name and surname for each registered user
 - `password_usr`** - the password chosen at registration, stored encrypted
 - `active_usr`** - stores 1 or 0, to determine if the account was activated by the user. It is used to prevent creating fake accounts, as it uses the e-mail address to send the activation link.
 - `level_usr`** - stores the access level for users. For this application, only 2 user levels will be defined: 0 - corresponds to regular shoppers, 1 - for site administrators
 - `randomkey_usr`** - stores a randomly generated key, used for account activation, to prevent activating other accounts.
 - `email_usr`** - stores the user e-mail address. It is also used as the username
 - `address_usr`** - stores the user's address. This will be used as shipping and billing address.
- The `product_prd` table stores information regarding the available products:
 - `id_prd`** - primary key, uniquely identifies a product
 - `name_prd`** - the product denomination
 - `price_prd`** - the product price. The amount of money the user will have to pay for one unit of the item
 - `quantity_prd`** - the number of items available on stock. This decreases when users purchase items, and increases when new stock is received.
- The `status_sta` table stores the possible states of an order:
 - `id_sta`** - the status identifier.

- **name_sta** - the state label (e.g. initiated, pending)
4. The *order_ord* table stores information that relates directly to an order:
 - **id_ord** -the order unique identifier.
 - **idusr_ord** - the identifier of the user that completed (or at least started the order)
 - **date_ord** - the date when the order was submitted
 - **idsta_ord** - stores the order state, as a foreign key to the state_sta table.
 - **sesid_ord** - stores the session id in effect when the user started the order. It is a unique value, used to link the user's session to the order, and in the acknowledgement and delete order process, to avoid security issues (instead of passing the order ID, the session ID is passed. It is much more difficult to fake).
 5. The *order_product_orp* table is a linking table for the order and product tables, replacing a many-to-many relation. It contains the following fields:
 - **id_opr** - the primary key and unique identifier
 - **idord_opr** - stores the order identifier
 - **idprd_opr** - stores the ID of the products added to the order
 - **quantity_opr** - stores the ordered quantity of a certain product.

The scripts necessary to create this table structure are provided inside the downloaded package, in the `\tutorials\Orders Management\db\` folder, as an sql or mdb file. Use the one that suits your particular database server. You can either create a new database, or use an existing one, and import the contents of the scripts to create the table structure, together with the sample data: user names, and products.

Once the database and files have been set up, you have to connect your application to the database server. to do so, open the index file of your new application, and define a new **Dreamweaver** connection. if using files from the zip package, and a connection is already available, you will have to edit the connection and provide your particular connection settings:



The screenshot shows a 'MySQL Connection' dialog box with the following fields and values:

- Connection name: connOrders
- MySQL server: localhost
- User name: root
- Password: [masked]
- Database: order_management (with a 'Select...' button next to it)

Buttons on the right: OK, Cancel, Test, Help.

Build site front-end

In this section you will build the application front-end, which will be used by visitors to create their orders from the list of available products. All pages you will create next are available directly to the regular shopper, from the main menu.

Before creating any pages, you must configure your site for the use of **MX Kollection**: set up user authentication option and e-mail server settings. To change these options, you will use the **InterAKT Control Panel**, with its appropriate sections.

To setup the site's login settings, follow the next steps:

1. Open the site's index page. Then open the **InterAKT Control Panel > Login Settings**.
2. In the user interface that opens, the **Options** tab, use the **Encrypt password** options, and use **Username, password** and **access level** to validate accounts. Also, you can set the period to keep a user logged in when the **Remember me** option is checked.

The screenshot shows a window titled "Login Settings" with a close button (X) in the top right corner. It features four tabs: "Options", "Database", "Session", and "User levels". The "Options" tab is active. Below the tabs, there is a section titled "Specify general login settings". The settings include: "Encrypt password:" with a checked checkbox; "Validate against:" with two radio buttons, the second of which ("Username, password and access level") is selected; and "Auto login validity:" with a text input field containing "30" and the word "days" to its right. At the bottom left, a yellow tooltip contains the text "Enter the login validity period." On the right side of the dialog, there are three buttons: "OK", "Cancel", and "Help".

3. Next, move on to the database tab, to configure which table columns will be used, and in which manner. Configure the dialog box similar to the image below:

The screenshot shows a 'Login Settings' dialog box with a 'User levels' tab selected. The dialog is titled 'Specify database information' and contains several fields for configuring user access levels. The fields are: Connection (connOrders), Table (user_usr), Primary key (id_usr), Username (email_usr), Password (password_usr), E-mail (email_usr), Active (active_usr), Level (level_usr), and Random key (randomkey_usr). The 'Primary key' and 'Level' fields have a 'Numeric' checkbox checked. A 'Define...' button is next to the Connection field. At the bottom, a yellow tooltip reads: '? Select the table column storing the random key.' On the right side of the dialog, there are 'OK', 'Cancel', and 'Help' buttons.

| Field | Value | Options |
|-------------|---------------|---|
| Connection | connOrders | Define... |
| Table | user_usr | |
| Primary key | id_usr | <input checked="" type="checkbox"/> Numeric |
| Username | email_usr | |
| Password | password_usr | |
| E-mail | email_usr | |
| Active | active_usr | |
| Level | level_usr | <input checked="" type="checkbox"/> Numeric |
| Random key | randomkey_usr | |

? Select the table column storing the random key.

4. In the User levels tab you can configure the access levels, as well as the redirect pages for each of them. Also, some global redirect pages can be defined here. The site's index is also the login page, as all other pages will require the user to be logged in, in order to allow any actions. Configure the dialog box as shown below:

Login Settings

Options | Database | Session | **User levels**

Define global redirect pages

Login page: Browse...

Default redirect on success: Browse...

Default redirect on fail: Browse...

Define user level information

User levels:

| Level | Redirect on succ... | Redirect on fail |
|-------|---------------------|------------------|
| 0 | view_products.php | index.php |
| 1 | admin/index.php | index.php |

Level:

Redirect on success: Browse...

Redirect on fail: Browse...

? Where to go if login fails for the current level.

OK
Cancel
Help

- Once the login settings are setup, you can close the Login settings user interface. The next step is configuring your e-mail server settings. This section is available from the **InterAKT Control Panel > E-mail settings**. If using a **PHP** server model, you can skip this step, as the settings are already entered in the `php.ini` configuration file. If using **ASP** or **ColdFusion**, you must enter the **host**, **username**, **password** and **default sender** e-mail address in the dialog box:

E-mail Settings

Define SMTP options

Mail server:

Port:

Username:

Password:

Default sender:

? Enter the e-mail address to send from.

OK
Cancel
Help

When all the initial settings are set, you can continue and build the application. First up is the site's main page, which will store a login form.

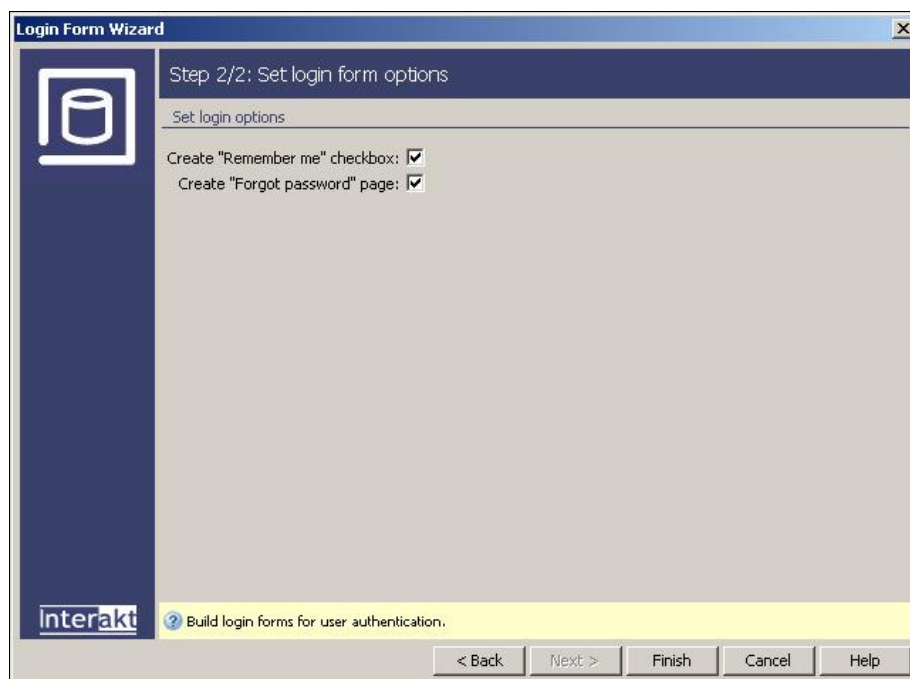
Implement user authentication

In this section of the tutorial you will create all elements relative to the site's user authentication part. This will contain:

- A login form that allows user to enter the site - this is also the main page
- Protection for all pages, based on user levels - pages in the admin folder can only be accessed by level 1, and some of the front-end pages by level 0.
- User registration, with account activation - this will allow new users create accounts with your site.

First, the site's main page, and login form. To build this page, you will use one of the wizards provided by **MX User Login** (part of **MX Kollection 3**): the **Login Form Wizard**. Follow the next steps to create the page:

1. Open the *index* page in **Dreamweaver**.
2. Apply the **Login Form Wizard** from the **MX Kollection** tab of the **Insert** bar. Configure each step as follows:
 - In the first step, there are no options to set. Simply check if the Login Settings set earlier are OK, as they will be used for the login process.
 - In the second step you have two check boxes: **Remember me** - which will generate a checkbox in the login form, allowing the auto-login feature (it is based on cookies), and the **Create Forgot Password** option. If checked, it will automatically create the entire password retrieval page. Check both options before closing the wizard.



3. After all options are set, simply click the **Finish** button to close the wizard and add the login form to the page.

Username: *

Password: *

Remember me:

[Forgot your password?](#)

The second element to create is the registration page. Building it is as easy as the login page, as you will use a wizard once more. To create the registration page, follow the next steps:

1. Open the *register* page in **Dreamweaver**.
2. Apply the **User Registration Wizard** from the **MX Kollection** tab of the **Insert bar**.
3. Configure the wizard's options in each step as follows:
 - Step one resembles the **Login Form Wizard**. It only allows you to check the validity of the configured login settings.
 - In step two you have to define the fields that will be used by the registration operation. Remove the **active**, **level** and **randomkey** columns from the list. To remove a field, select it in the grid and press the **Minus (-)** button.

User Registration Wizard

Step 2/4: Configure the form fields

Set form fields properties

Form fields:

| Column | Label | Display as | Submit as |
|--------------|-----------|----------------|-----------|
| name_usr | Name: | Text field | Text |
| email_usr | Email: | Text field | Text |
| password_usr | Password: | Password field | Text |
| address_usr | Address: | Text area | Text |

Label:

Display as:

Submit as:

Default value:

interakt

- The third step allows defining validation rules for each of the fields involved in the registration process. This step does not appear if **MX Form Validation** is not installed. If available, set the **E-mail** field to use e-mail address as validation type, then click the Next button to continue with the wizard.
- The last step has two checkboxes for configuring registration related options: whether to send a welcome e-mail message, or to use account activation. Check both options and click Finish to close the wizard.

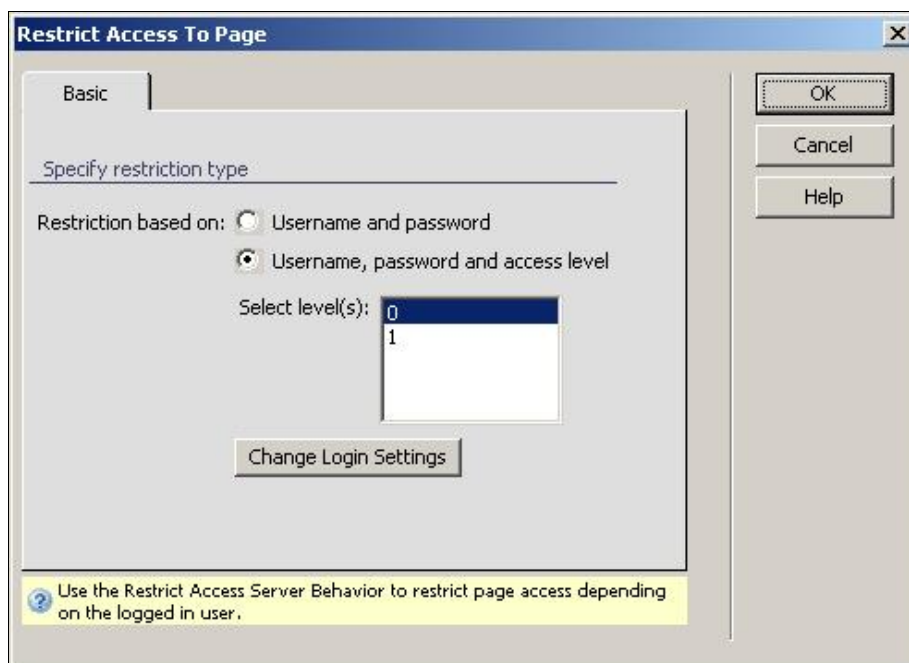
- The wizard will add both **HTML** elements, and code to the page, and will also generate a new page, named *active*. This page is used in the activation process: it contains an **Update Transaction**, that receives the randomly generated key as URL parameter, and changes the value in the *active_usr* column from 0 to 1. To reach this page, users must click the link offered in the activation e-mail message.

The last action to take regarding the user authentication portion of this tutorial, is to protect various pages from users that do not have the right to access them. Two type of restrictions will be applied in what follows:

- the simple **Restrict Access to Page** server behavior on all pages that need protecting in the front-end section
- the **Restrict Access to Files in Folder** command, for all files in the admin area.

The pages to protect in the front-end section are: *view_products*, *view_order*, *clear_order*, *complete_order* and *add_to_order*. The other pages are either public, or available only through e-mail links (you will get to these pages later on). to restrict access to these pages, follow the next steps:

- Open the page to protect in **Dreamweaver**.
- Apply the **Restrict Access to Page** server behavior from the **Server Behaviors tab > + >MX Kollection > User Login**. Configure the user interface as follows:
 - Select the *username*, *password* and *access level* as restriction elements
 - For the access level, select 0, to allow regular shoppers access these pages.
 - Close the dialog box by clicking the OK button and save the page.



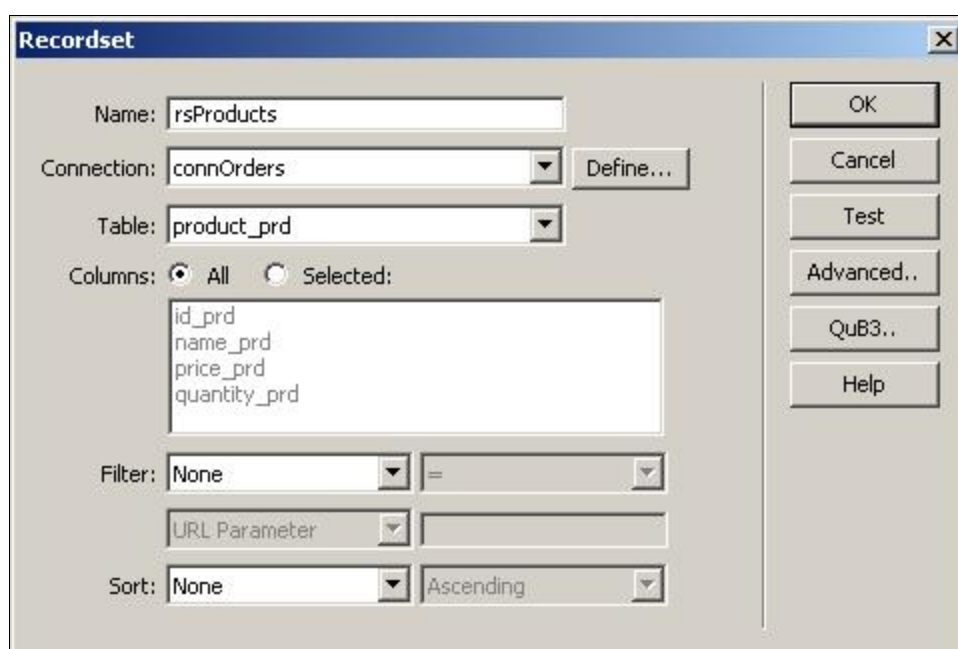
- Repeat steps 1 and 2 for each page in the list above. When all pages are protected, you can move on and continue with the development of the application.

Display available products

Once the user authentication system is in place, you can continue and construct the rest of the application. The first thing to do now is display a list of available products. And since the *view_products* page was selected as the default redirect when the login operation is successful, it will be created first.

To create the page elements, follow the next steps:

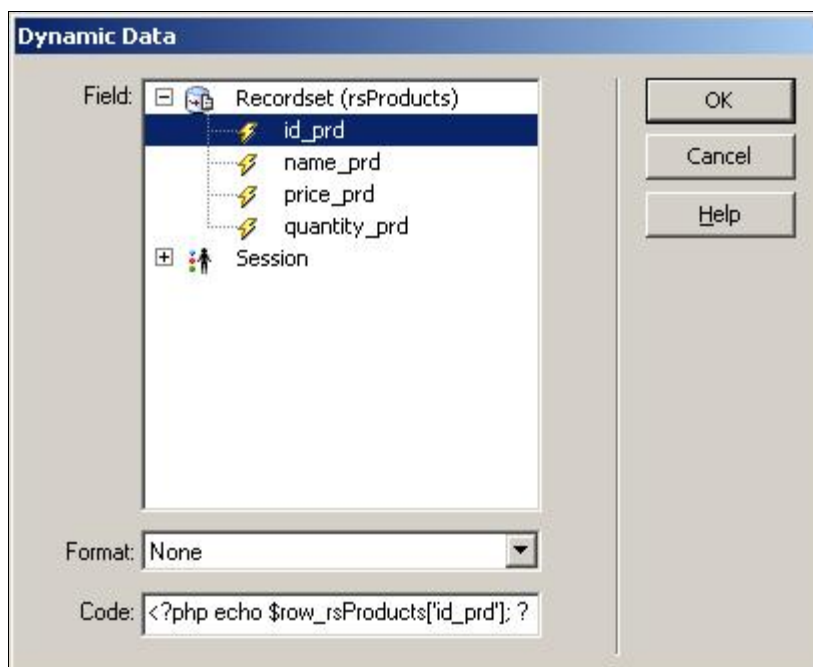
1. Open the *view_products* in **Dreamweaver**.
2. The page will display all available products in the database. To retrieve them, you must create a recordset. Click on the Plus (+) button of the **Bindings** tab, and select the **Recordset (Query)** option.
 - In the dialog box that opens, enter a name for the recordset (e.g. *rsProducts*), select the *connOrders* database connection created at the beginning of the tutorial and the *product_prd* table.



- After you've set the new recordset options, click the **OK** button to add it to the page.
3. To display the records from the newly created recordset, you will use a dynamic table. You can add a **Dynamic Table** from the **Application** tab of the **Insert** bar. Configure it to display all records, and set border, padding and spacing options according to your particular taste in design:



4. The table displays all recordset retrieved columns, including the product ID, which is of no use for the end-user. Therefore, remove this column entirely. Also, change the column headers from the table column names into something more human readable (e.g. **name_prd** into Name, **price_prd** into Price, quantity_prd into **Available**) and set the **Header** property on them.
5. What the user is interested in is the possibility of adding one of the products to his order. To do so, you must add a new link next to each product, pointing to the *add_to_order* page. And in order for the page to recognize the product to add, you must pass its ID as an URL parameter.
6. To add the new link, create a new column at the end of the table. Right click in one of the last cells, and select **Table > Insert rows or columns**. Add a new column after the current selection. In the second cell, enter the "Add to order" text. Right click on it, and select the Make link option. In the dialog box select the *add_to_order* page, and then click on the Parameters button to define the URL parameters to pass.
7. Add a new parameter named **id_prd**, and for the value, click the **Dynamic Data** icon. Select the **id_prd** field of the *rsProducts* recordset:

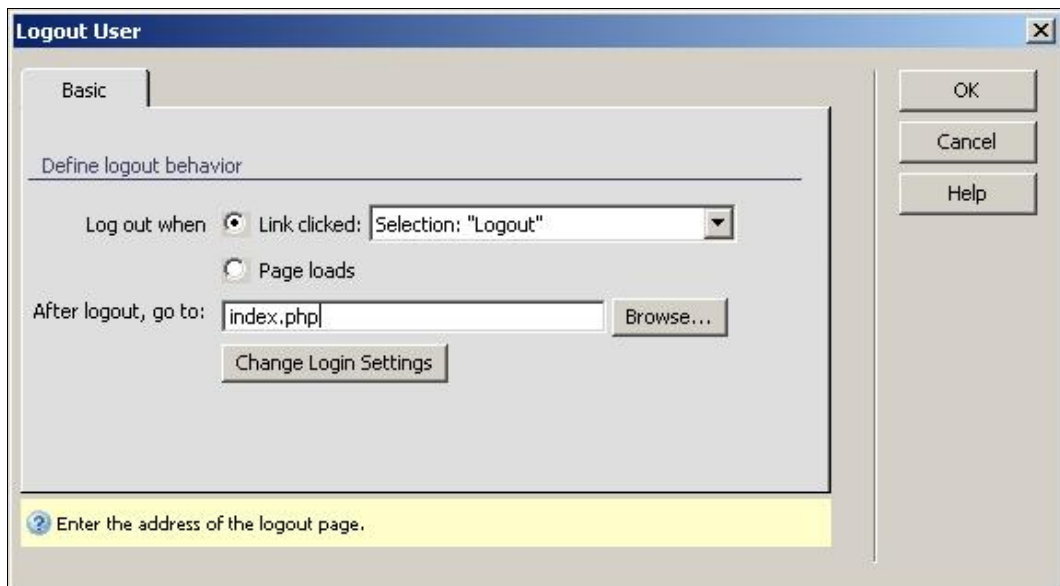


8. Confirm the changes to all interfaces, by clicking the OK buttons. Save the page and upload it to the server. To access it, you must have created your own account, or use one of the pre-made user names.

| Name | Price Available | | |
|-------------|-----------------|----|------------------------------|
| Mouse | 25 | 0 | Add to order |
| Keyboard | 30 | 13 | Add to order |
| Monitor 17" | 150 | 5 | Add to order |

Below the product list, a series of links must be available for the visitor, in order to allow completing or clearing the order. If using the files from the zip package, these links are already on page, but if you've created a blank file you must manually add them:

- View order - link to *view_order*
- Clear order - link to *clear_order*
- Complete order - link to *complete_order*
- Logout - to create this link, enter the text, select it, and apply the Logout User server behavior from the **Server Behaviors Tab > + >MX Kollection > User Login**. In the dialog box that opens, set the redirect page to the site index, and click the **OK** button to apply it.



Now that the product list page is completed, you can continue and create the logic in the *add_to_order* page, that will add a new element to the order.

Add products to orders

This section handles the page for adding new products to an existing order. Due to the structure of this application however, this page will also perform other tasks (like initiating the order). Creating a new order, means adding elements into two database tables: the order's main properties go into the **order_ord** table (the ID, the date and state, as well as the session ID), while the correspondence to the products it contains is made through the *order_product_opr* table (which stores pairs of ID's: order - product).

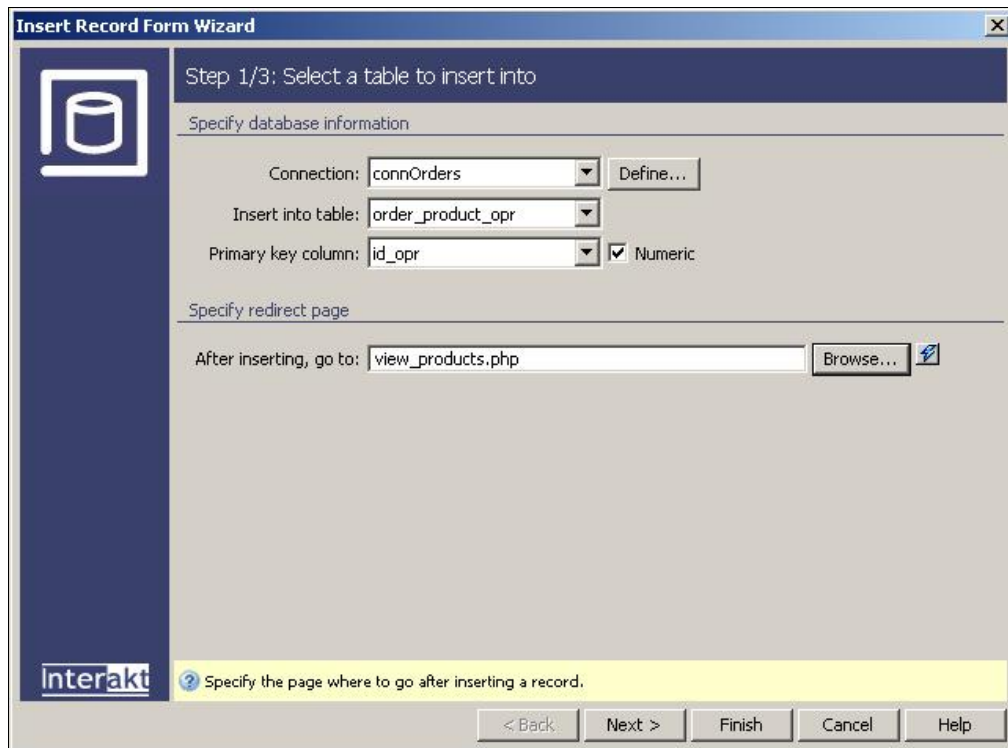
When a user enters the site, after logging in, there is no existing order into which he can add its products. Therefore one must be created, that is unique, and tied to the current browsing session. To achieve this, the session ID is used as a property for the order, and the mechanism that takes place on the add to order page is as follows:

- After a user clicks the add to order link for a product, the *add_to_order* page is loaded with the product's ID as URL parameter.
- The page code must check if an order associated to this session already exist, by verifying if an order ID is stored in the session variable **kt_order_id**. If found, it is the ID of the order that will be used throughout the insert operations into the *order_product_opr* table.
- If an existing order is not found with the ID in the **kt_order_id** session variable, the session ID is retrieved, and a table search is conducted, to see if an existing order exists for this session, but is not loaded in the *kt_order_id* variable. If one is found, it's ID is retrieved and loaded into a session variable (the one checked at step 2) and the insert transaction carries on.
- If no matching record is found in the database, it means that no order was ever associated to this user's session. Therefore, a new order is created, by inserting into the *order_ord* table a new record, with the present date, the initiated (the 0 value) state, and with the current user's ID. Then, the last inserted ID is also saved into a session variable.

All the actions described above take place before the actual insert operation, that adds a new product to a order, and they cannot be done automatically by **MX Kollection**. Some custom code is necessary to achieve this set of operations. **MX Kollection** helps however, by the use of its **Custom Trigger** server behavior, that allows entering code, and deciding when to execute it. Start by opening the *add_to_order* page in Dreamweaver.

Before you can add a custom trigger on the page, you must have a transaction (either Insert, Update, Delete or custom). Since the page's purpose is to insert products into the order, you will apply the **Insert Record Wizard**. You can access it from the **MX Kollection** tab of the Insert bar, and must configure it as follows:

1. In the wizard's first step, select the database connection created at the beginning of this tutorial. For the table to insert into, use *order_product_orp* (it is the linking table, that only stores pairs of ID's and the quantity). For the redirect page, use the product list (*view_products*):



Insert Record Form Wizard

Step 1/3: Select a table to insert into

Specify database information

Connection: connOrders Define...

Insert into table: order_product_opr

Primary key column: id_opr Numeric

Specify redirect page

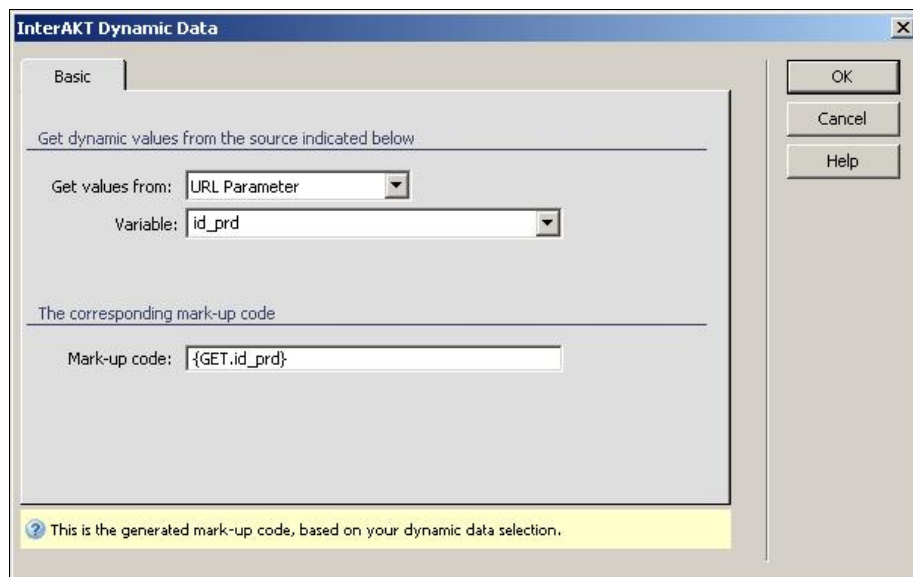
After inserting, go to: view_products.php Browse...

Interakt

Specify the page where to go after inserting a record.

< Back Next > Finish Cancel Help

2. In the second step, you must decide what fields will be used for the existing database tables. Configure them as follows:
 - The **idprd_opr** field is passed to the page as an URL parameter. It should not have a form field correspondent, as it will be invisible for the user. In the Display as drop-down menu, select Hidden field. For the default value, click the **InterAKT Dynamic Data** icon, and select URL parameter. Enter the name of the parameter - **id_prd**



InterAKT Dynamic Data

Basic

Get dynamic values from the source indicated below

Get values from: URL Parameter

Variable: id_prd

The corresponding mark-up code

Mark-up code: {GET.id_prd}

This is the generated mark-up code, based on your dynamic data selection.

OK Cancel Help

- The **idord_opr** field will take its value from a session variable (you can name the variable as you choose, as long as it is consistent throughout the tutorial - let's name it **kt_order_id** -). Set it to hidden field, and for the default value use the

InterAKT Dynamic Data again. Select the Session variable this time, and for the name enter `kt_order_id`.

- Since the quantity is a property that can be changed by the user, leave it as a text field that submits a numeric value.

Insert Record Form Wizard

Step 2/3: Configure the form fields

Set form fields properties

Form fields: + -

| Column | Label | Display as | Submit as |
|--------------|-----------|--------------|-----------|
| idprd_opr | | Hidden field | Numeric |
| idord_opr | | Hidden field | Numeric |
| quantity_opr | Quantity: | Text field | Numeric |

Label: Quantity:

Display as: Text field

Submit as: Numeric

Default value:

Interakt

The list of fields associated to the insert transaction.

< Back Next > Finish Cancel Help

3. The third step of the wizard allows you to define validation rules for each field involved in the transaction. You can safely skip this step, and close the wizard by pressing the Finish button.

At this point the page contains the form allowing the insert into the database - as HTML element, and the Insert transaction - regarding the logic. Since a transaction exist, this means you can add triggers to the page. The **Custom Trigger** is the most useful at this point. To add a custom trigger to the page, access it from **Server Behaviors > MX Kollection > Forms > Custom trigger**. In the user interface that opens, you can enter code. The code executing the operations mentioned on top of this page is provided below:

- For the **PHP_MySQL, PHP_ADODB** server model:

```
if(!isset($_SESSION['kt_order_id'])) {
    $id = session_id();
    $query = "INSERT INTO order_ord (idusr_ord,date_ord,idsta_ord,
    sesid_ord) VALUES
    ('".$_SESSION['kt_login_id']."', '".date("Y-m-d",time())."',0,
    '$id')";
    $tNG->connection->execute($query);
    $_SESSION['kt_order_id'] = $tNG->connection->Insert_ID('order_ord',
    'id_ord');
}
$tNG->setColumnValue("idord_opr", $_SESSION['kt_order_id']);
return NULL;
```

- For **ColdFusion**

The **ColdFusion** logic is a bit different than that for **PHP**. First, in the **Custom Trigger**, add this line:

```
return Request.AliasCustomTrigger(tNG);
```

Now close the **Custom Trigger** and go into the **Code View** in **Dreamweaver**. Add the

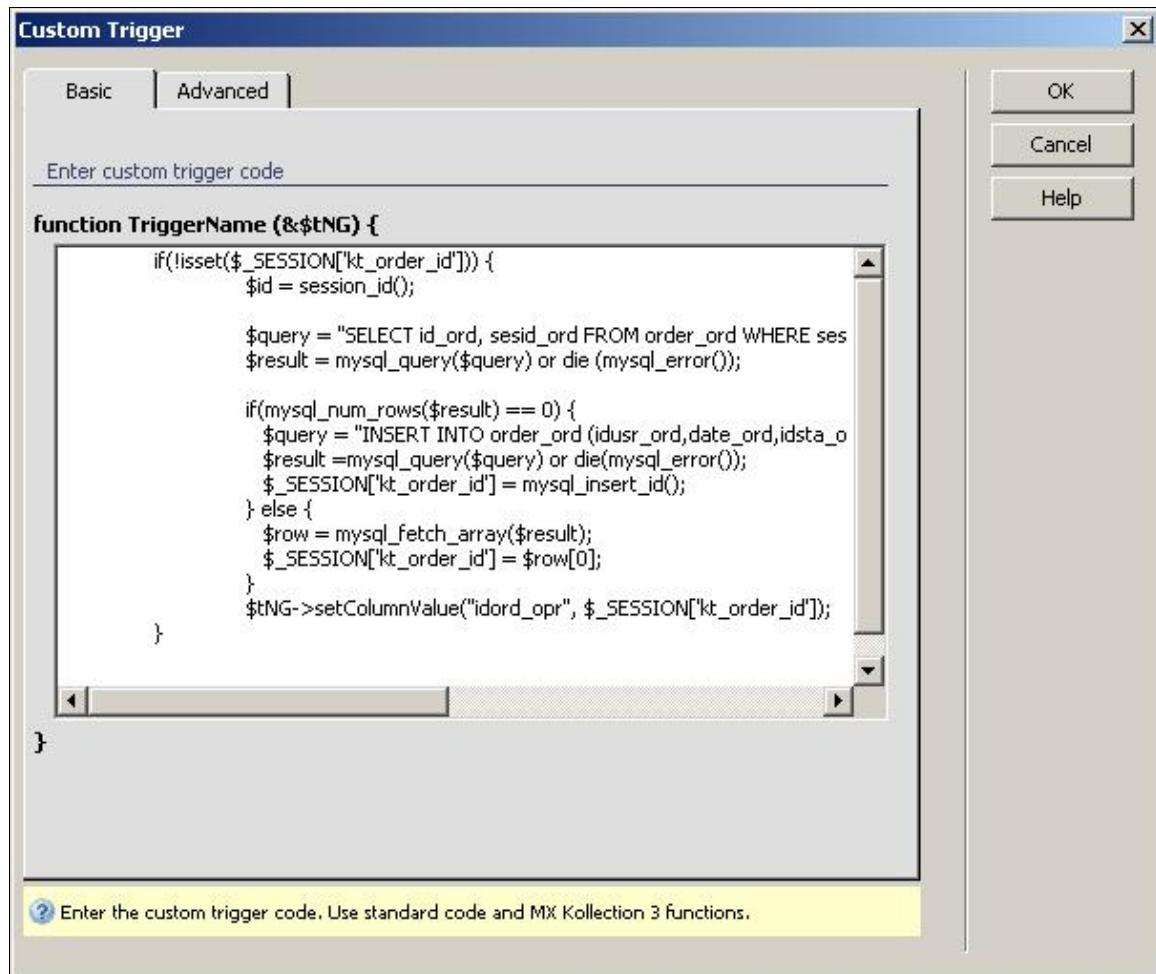
following code immediately before the Custom Trigger code begins. You will recognize the custom Trigger code because it begins with this comment: `//start Trigger_Custom trigger` . Here is the code to add:

```
<cffunction name="AliasCustomTrigger">
<cfargument name="tNG" required="true">
<cfset var query = "">
<cfset var result= "">
<cfif NOT isDefined("Session.kt_order_id")>
<cfset id = Session.CFID>

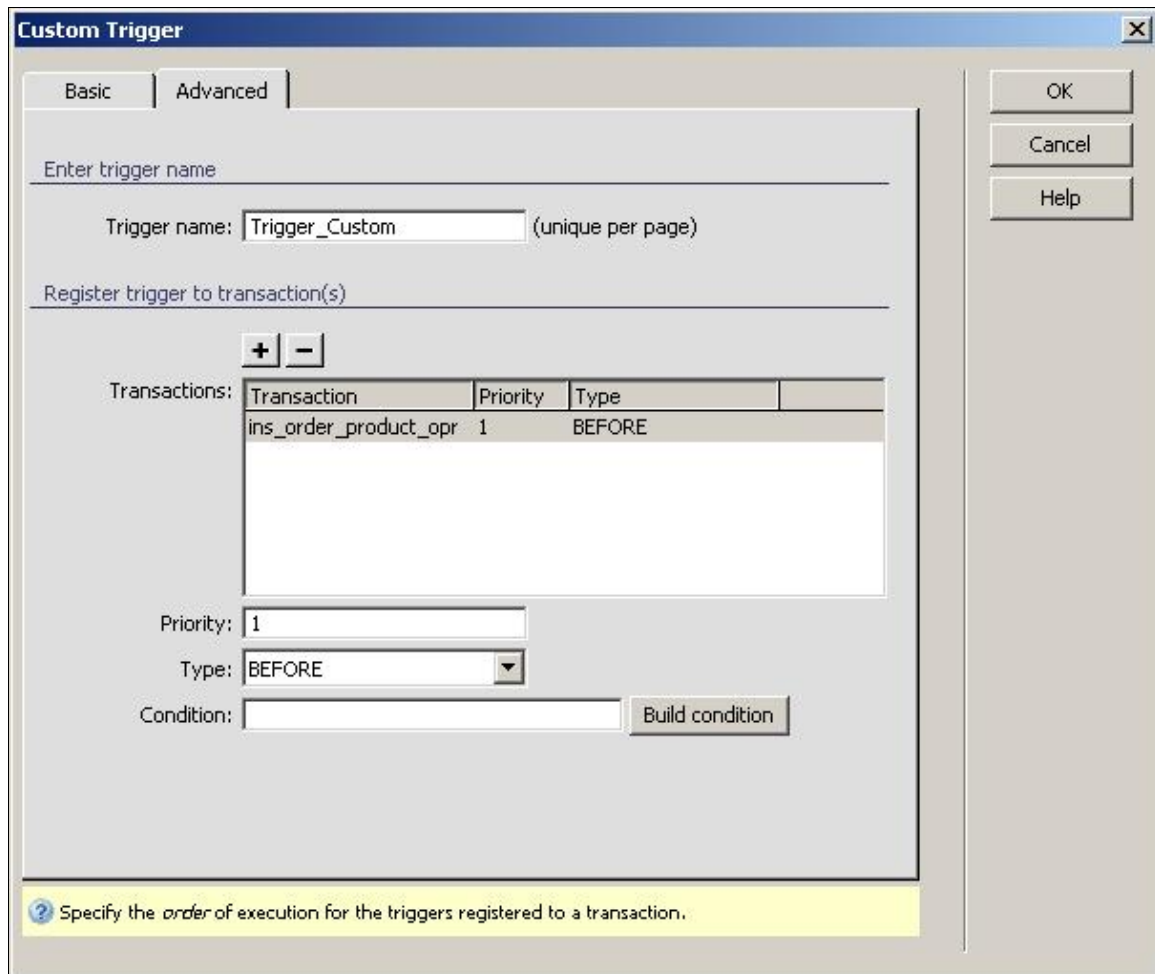
<cflock name="insertOrderOrd" type="exclusive" timeout="10">
<cfset query = "INSERT INTO order_ord (idusr_ord,date_ord,idsta_ord,
sesid_ord) VALUES (' & SESSION['kt_login_id'] & ', ' &
DateFormat(NOW(),"yyyy-mm-dd") & ',0, ' & id & ')">
<cfquery name="result" datasource="#tNG.connection#">
#PreserveSingleQuotes(query)#
</cfquery>
<cfset query ="SELECT max(id_ord) as orderid FROM order_ord">
<cfquery name="result" datasource="#tNG.connection#">
#PreserveSingleQuotes(query)#
</cfquery>
<cfset SESSION['kt_order_id'] = result.orderid>
</cflock>
<cfset tNG.setColumnValue("idord_opr", SESSION['kt_order_id'])>
</cfif>
<cfreturn Request.KT_Null>
</cffunction>
<cfset Request.AliasCustomTrigger = AliasCustomTrigger>
```

- For **ASPVBScript**

```
If Session("kt_order_id") = "" Then
id = Session.SessionID
query = "INSERT INTO order_ord (idusr_ord,date_ord,idsta_ord,
sesid_ord) VALUES (' & Session("kt_login_id") & ', ' &
Date() & ',0, ' & id & ')"
tNG.connection.Execute(query)
query = "SELECT @@IDENTITY as newid"
Set rsID = tNG.connection.Execute(query)
pkValue = Cstr(rsID.Fields.Item("newid").Value & "")
Session("kt_order_id") = pkValue
tNG.setColumnValue "idord_opr", Session("kt_order_id")
End If
Set Trigger_Custom = nothing
```



Next, you have to set the trigger to be executed before the transaction. To set trigger properties, click on the Advanced tab. Here, set the Type from the drop-down menu to BEFORE, and the priority to 1. This means that the trigger will execute before the transaction it is registered to (you can see it in the grid), with the highest priority.



Custom Trigger

Basic | Advanced

Enter trigger name

Trigger name: (unique per page)

Register trigger to transaction(s)


+ -

| Transaction | Priority | Type |
|-----------------------|----------|--------|
| ins_order_product_opr | 1 | BEFORE |

Priority:

Type:

Condition:

 Specify the *order* of execution for the triggers registered to a transaction.

OK
Cancel
Help

Now, when the page loads, the existence of an order for the current session is checked, and if it does not exist, one is initialized, before the user actually adds the product, and the transaction will go on.

After adding the possibility to browse the product list, and add any of them to the order, together with a specified quantity, you must create next the pages that allow finalizing the order. To create them, move on to the next section of the tutorial.

Complete order process

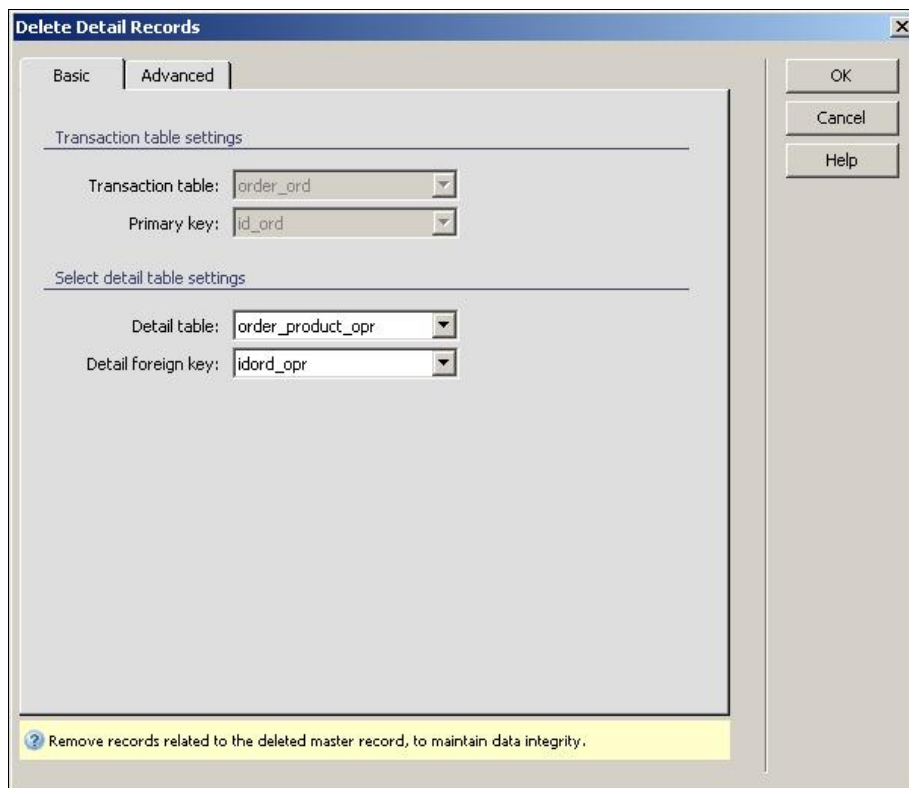
An order can have two possible ending scenarios:

1. The user decides the products he added to the order are necessary, and finalizes it - the order passes into pending, and becomes available for the site administrators. To implement this functionality, you will create the *complete_order* page, which computes the remaining stocks, forwards the order to the administrators and sends additional e-mail messages (e.g. thank you, confirmation, or if no stocks are available - but this case will be treated later on).
2. The user decides the order is no longer necessary and cancels it. At this point, all order details must be destroyed, as a new order may be created. Order details are stored in the *order_ord* and *order_product_opr* tables, as well as in the **kt_order_id** session variable. Therefore, clearing the order means deleting all records and variables. This is the *clear_order* page.

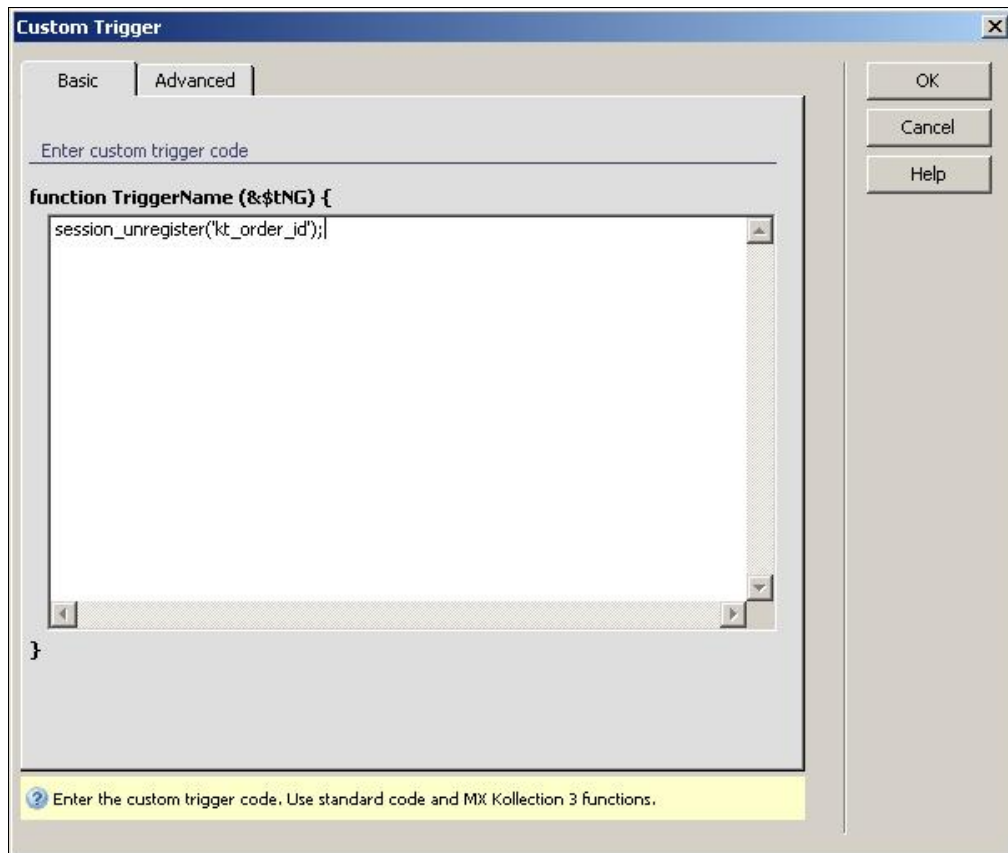
First you will implement the *clear_order* page that deletes all details of an existing order. To build this page, follow the next steps:

1. Open the *clear_order* page in **Dreamweaver**. A restrict access to page server behavior should be already on the page, added in the User authentication section of the tutorial.
2. Since the page must delete the order details, it will contain a delete transaction, that will remove the order details from the *order_ord* table (this is where the ID, date, status and user are stored for each order).
3. To remove a record, apply the **Delete Record Wizard** from the **MX Kollection tab** of the **Insert bar**. Configure the wizard as follows:
 - In the **Connection** drop-down menu select the database connection created at the beginning of this tutorial.
 - For the **Delete from table**, select the *order_ord* table, as only the main order details will be deleted at first.
 - The order ID is stored as the primary key of the column, Also, when adding a product for the first time, the custom code also saves the order ID into the **kt_order_id** session variable. The condition needed to match the record to delete is to have the primary key equal the **kt_order_id** session variable.
 - Select Session variable in the Primary key equals drop-down menu, and for the reference, enter **kt_order_id**.
 - The page to open after clearing an order is the product list, so that the user can continue shopping. Enter in the **After deleting, go to** textfield the *view_products* page.
 - The completed dialog box should look as follows:

- Click the **OK** button to apply the delete transaction on page. Only a translator will be displayed in **Dreamweaver**, as the entire transaction is transparent to the end user.
4. At this point, when you click the clear order link, order details are removed from the `order_ord` table. However, this is not the only table storing information about an order: the `order_product_opr` table stores the list and quantity of products associated to an order. As shown in the database structure description, it contains pairs of identifiers from the product and order tables, as well as the quantity for each product.
 5. To completely remove an order, all records stored in the `order_product_opr` table which relate to the order deleted at step 3 must be removed. Since the order ID is stored as a foreign key in the table, it can be viewed as a detail table of `order_ord`. MX Kollection provides a server behavior that allows removing orphaned records (detail records without a master element): the Delete Detail Records trigger.
 6. Apply the **Delete Detail Records** from the **Server Behaviors tab > + > MX Kollection > Form Validation**. The trigger will automatically recognize the existing delete transaction and will register to it. Also, the master table and primary key are automatically filled in.
 7. Configure the remaining options as follows:
 - In the Detail table drop-down menu select the `order_product_opr` table.
 - In the Detail foreign key, select the table column storing the order ID in the `order_product_opr` table (it is the ***idord_opr*** column)



- When the options are all set, click OK to add the trigger onto the page. You do not need to set any advanced options, as the trigger is by default configured to execute after the main transaction, and retrieve the order ID from it.
8. Now, all order details stored in the database will be deleted when the clear order link is used. The only remaining proof of the order existence (which also poses a problem when creating a new order) is the ID saved in the **kt_order_id** session variable. The problem appears when the same user tries to initiate a new order, after clearing an existing one. Since the ID is still in the session variable, the order is considered as existing in the database, and adding products to the order will result into invalid records (an order ID that does not exist will be entered together with the product code in the *order_product_opr* table).
 9. To remove this variable you will have to write some code of your own. To enter the code, add a new **Custom trigger** to the page. You can access this server behavior from the **Server Behaviors tab > + > MX Kollection > Forms**. By default, the trigger will execute after the transaction, with a priority of 50. These properties are OK, as the variable removal must take place after the records are deleted from the database table.
 10. In the Custom trigger dialog box's Basic tab, enter the code that clears the session variable in the textarea. The code to enter is:
 - For the **PHP_MySQL, PHP_ADODB** server model:
`session_unregister('kt_order_id');`
 - For **ColdFusion**:
`structdelete(Session,"kt_order_id");`
 - For **ASPVBScript**
`Session.Contents.Remove("kt_order_id")`
 11. The custom trigger dialog box should look like the following (the screenshot has been taken on **PHP_MySQL**, so code differences will appear):



12. Apply the server behavior, and add the code to the page by clicking the **OK** button.

The page that clears an order is now completed, as it removes all traces of the order, both from the database and from the session variables.

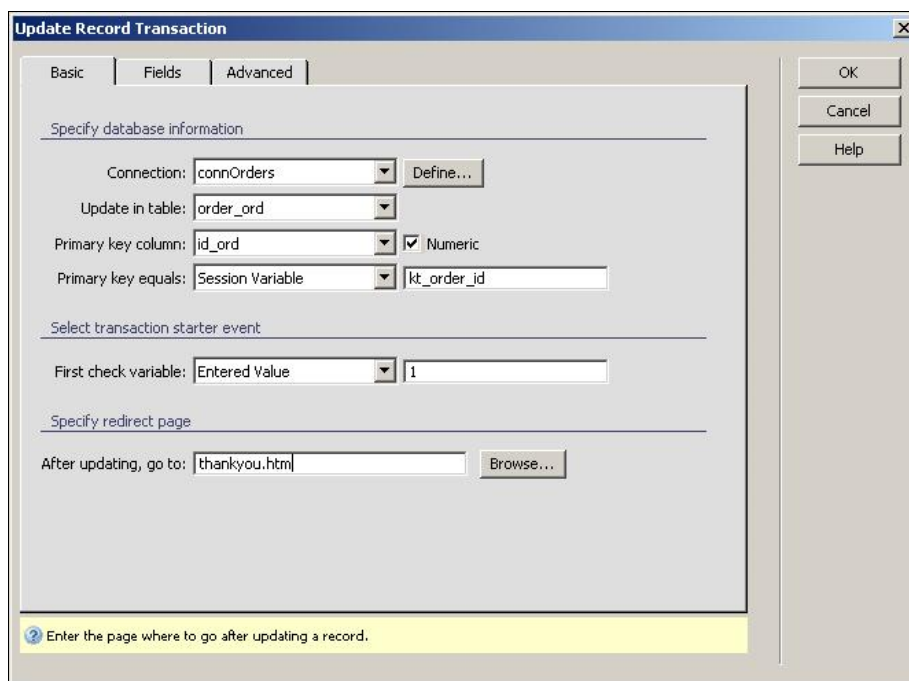
The page you will create next handles completing and confirming an order. The page that performs this action is *complete_order*. The list of actions to be performed by the complete order page should be:

1. For each product added to the order, the requested quantity must be checked against the available quantity. This is done by retrieving the ordered quantity and the product available quantity. If the ordered quantity is higher than the available one, then there are not enough in stock.
2. The order is then treated globally:
 - If at least one product cannot be provided in full quantity, the entire order is considered as being un-deliverable. If this happens, an e-mail message is sent to the purchaser, requesting action: either cancel the order, or wait until the entire order can be delivered. The order's status will be changed into "Waiting for acknowledgement", until the response from the user arrives.
 - If the desired quantity can be provided for each product, the available quantity is diminished with the amount ordered and the order's status is changed to "Pending".
3. The user is redirected to a thank you page, which contains a link to the product list page.

To implement this suite of actions, follow the next steps:

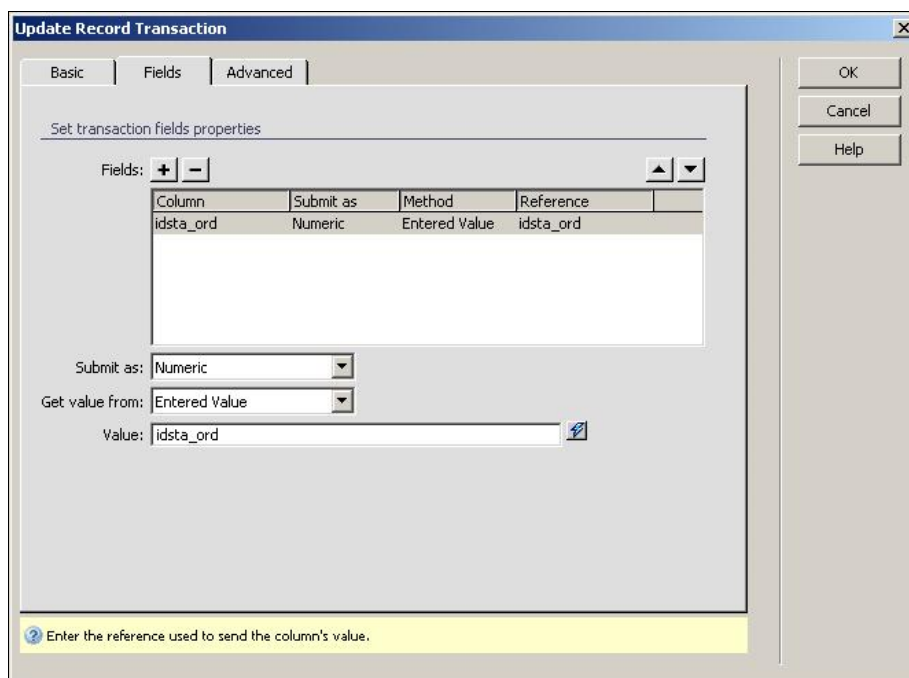
1. Open the *complete_order* page in **Dreamweaver**.

2. The quantity check and the e-mail sending are implemented using triggers. The order status update is implemented by using an update record transaction. A trigger can only be added to a page that already contains a transaction, so the update transaction will be the first one to implement.
3. Since the update operation will take place behind the scenes, without displaying any fields (the status to update to is set by the trigger that checks the quantity - either pending or waiting for acknowledgement), there is no point in using the wizard. Instead, use the **Update Record Transaction** server behavior. It can be found in the **Server Behaviors tab > + > MX Kollektion > Forms > Advanced**.
4. The dialog box is divided into three tabs, each addressing some options that can be set for the transaction. The basic tab is where you must define general information regarding the database, starting condition and redirect page:
 - In the **Connection** drop-down menu, select the database connection created at the beginning of the tutorial
 - For the table to update into, select the *order_od* - this is where the order status is stored
 - In order to update only the current order, you must apply the update operation on the record that has the same ID as the one stored in the *kt_order_id* session variable.
 - The transaction must start automatically, so you must change the starter event to an entered value. Select this option in the drop-down menu, and enter a value of choice for the reference (e.g. 1).
 - For the redirect page, browse to the *thankyou.html* file in the site root.



5. In the Fields tab, you can decide which table columns will be involved in the update transaction, and where each of them will take its value.
 - Initially, all table columns are displayed in the grid. the only one that is needed is *idsta_ord*, which stores the order status. To remove the other fields, select them one by one, and click on the Minus (-) button on top of the grid.

- For the remaining field (*idsta_ord*), set it to submit as numeric. In the Get value from drop-down select the element that provides the value to insert. Since the actual value will be determined by the quantity check trigger, set it to Entered value. Leave the value field at its default reference (*idsta_ord*)



Update Record Transaction

Basic | Fields | Advanced

Set transaction fields properties

Fields: + -

| Column | Submit as | Method | Reference |
|-----------|-----------|---------------|-----------|
| idsta_ord | Numeric | Entered Value | idsta_ord |

Submit as: Numeric

Get value from: Entered Value

Value: idsta_ord

Enter the reference used to send the column's value.

6. Click the **OK** button to close the dialog box and add all elements to the page. A translator will be displayed in **Dreamweaver**, pointing to where possible error messages will be displayed.
7. After the update transaction is added to the page, you can start creating the section that checks the stocks, updates available quantity and sets the order status. These operations cannot be done with **MX Kollection** features, so once more, a little coding is needed. The code is entered by using a **Custom trigger**. To add a Custom trigger, go to the **Server Behaviors tab >+>MX Kollection >Forms**. Before writing the actual code, switch to the advanced tab, and set the trigger type to BEFORE, and the priority to 1. This is done because the quantity check must take place first on page. To set trigger properties,
 8. The code provided below executes the following actions:
 - The order details are retrieved from the database, with a INNER JOIN between the *order_ord*, *product_prd* and *order_product_opr* tables - in order to retrieve both the order and product details
 - For each of the records retrieved by the query, the ordered quantity is checked against the available quantity in a loop. If for any of the ordered products there is not enough stock, a flag is set to prevent honoring the order.
 - After the loop is finished, the flag is evaluated. If TRUE (or 1), it means that there is not enough stock, so the status is set to "Waiting for acknowledgement" - code 2, by executing a tNG procedure.
 - If the flag is FALSE (or 0), it means the order can be delivered. A new query updates the quantities for each of the ordered products. The new quantity equals the difference between the available and ordered ones.
 - Once the quantity update loop is over, the order status is set to "Pending" - code 1, and the session variable storing the order ID is destroyed, to prevent further additions to the (now) completed order.
 9. The code to use is as follows (use the one appropriate to your server model)

- For the **PHP_MySQL** server model:


```
$query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM (order_product_opr INNER JOIN
product_prd ON product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" .$_SESSION['kt_order_id'];

$result=mysql_query($query) or die(mysql_error);
$notInStock = 0;
while($row=mysql_fetch_assoc($result)) {

if ($row['quantity_opr']>$row['quantity_prd']) {
$notInStock = 1;
}
}

if ($notInStock == 1) {
$tNG->setColumnValue("idsta_ord", 2);

} else {

$query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM (order_product_opr INNER JOIN
product_prd ON product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" .$_SESSION['kt_order_id'];

$result=mysql_query($query) or die(mysql_error);

while($row2=mysql_fetch_assoc($result)) {
$newQty = $row2['quantity_prd'] - $row2['quantity_opr'];
$query = "UPDATE product_prd SET quantity_prd = $newQty WHERE
id_prd = " . $row2['id_prd'];
$resultUpdateQty = mysql_query($query) or die (mysql_error());
}

$tNG->setColumnValue("idsta_ord", 1);
}
session_unregister('kt_order_id');
return null;
```
- For the **PHP_ADODB** server model
- For the **ColdFusion** server model

Like the previous page, the **ColdFusion** steps are little different than for **PHP**. In the **Custom Trigger**, add this line: `return Request.AliasCustomTrigger(tNG);`
 Now close the **Custom Trigger** and go into the **Code View** in **Dreamweaver**. Add the following code immediately before the Custom Trigger code begins. You will recognize the custom Trigger code because it begins with this comment: `//start Trigger_Custom trigger`. Here is the code to add:

```
<cffunction name="AliasCustomTrigger">
<cfargument name="tNG" required="true">
<cfset var query="">
<cfset var result="">
<cfset var notInStock = 0>
<cfset var resultUpdateQty = "">
<cfset query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
```

```

product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM
(order_product_opr INNER JOIN product_prd ON
product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" & SESSION['kt_order_id']>
<cfquery name="result" datasource="#tNG.connection#">
#PreserveSingleQuotes(query)#
</cfquery>

<cfloop query="result">
<cfif result.quantity_opr GT result.quantity_prd>
<cfset notInStock = 1>
</cfif>
</cfloop>
<cfif notInStock EQ 1>
<cfset tNG.setColumnValue("idsta_ord", 2)>
<cfelse>
<cfset query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM
(order_product_opr INNER JOIN product_prd ON
product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" & SESSION['kt_order_id']>

<cfquery name="result" datasource="#tNG.connection#">
#PreserveSingleQuotes(query)#
</cfquery>
<cfloop query="result">
<cfset newQty = result.quantity_prd - result.quantity_opr>
<cfset query = "UPDATE product_prd SET quantity_prd = $newQty
WHERE id_prd = " & result.id_prd>
<cfquery name="resultUpdateQty" datasource="#tNG.connection#">
#PreserveSingleQuotes(query)#
</cfquery>

</cfloop>

<cfset tNG.setColumnValue("idsta_ord", 1)>
</cfif>
<cfset StructDelete(SESSION, 'kt_order_id')>
<cfset Request.KT_Null>
</cffunction>
<cfset Request.AliasCustomTrigger = AliasCustomTrigger>
<cfscript>

```

- For the **ASP VBScript** server model

```

query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM (order_product_opr INNER JOIN
product_prd ON product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" & Session("
kt_order_id")
Set result = tNG.connection.Execute(query)notInStock = 0
while Not result.EOF
If result("quantity_opr") > result("quantity_prd") Then
notInStock = 1
End If
result.MoveNext
Wend

```

```

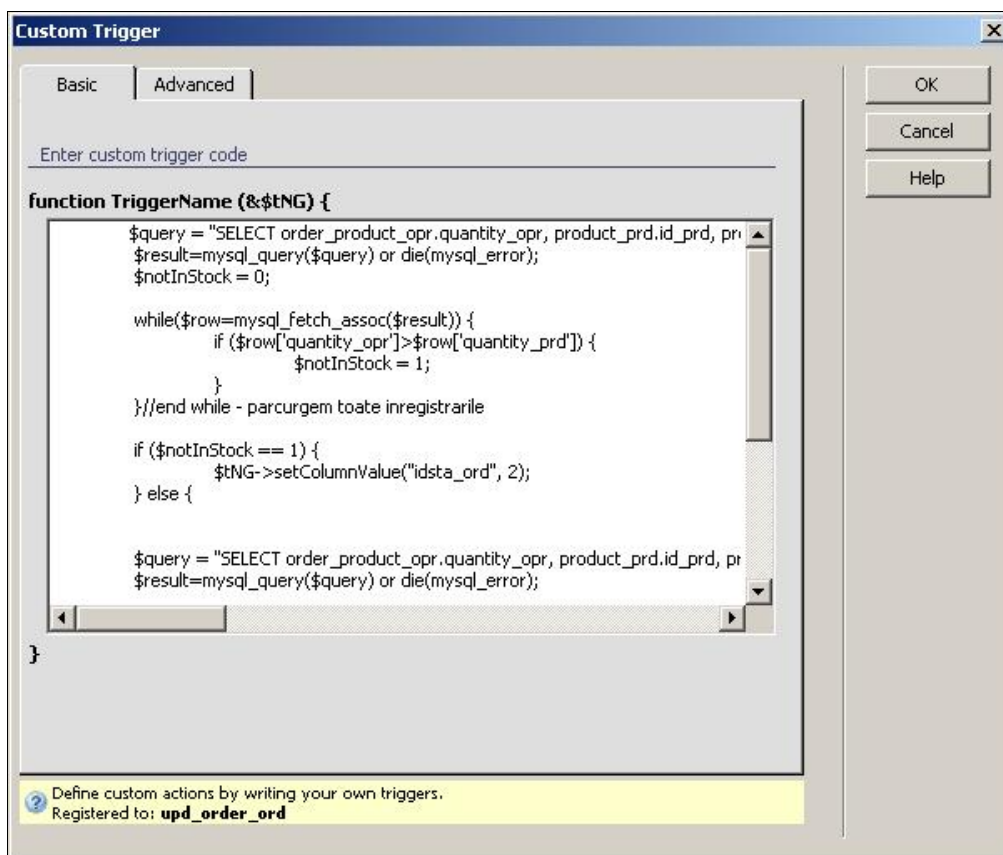
If notInStock = 1 Then
tNG.setColumnValue "idsta_ord", 2
Else
query = "SELECT order_product_opr.quantity_opr,
product_prd.id_prd, product_prd.name_prd,
product_prd.price_prd, product_prd.quantity_prd,
order_product_opr.idord_opr FROM (order_product_opr INNER JOIN
product_prd ON product_prd.id_prd=order_product_opr.idprd_opr)
WHERE order_product_opr.idord_opr=" & Session("kt_order_id")
Set result = tNG.connection.Execute(query)
While Not result.EOF
newQty = result("quantity_prd") - result("quantity_opr")
query = "UPDATE product_prd SET quantity_prd = " & newQty &
" WHERE id_prd = " & result("id_prd")
tNG.connection.Execute(query)
result.MoveNext
Wend

tNG.setColumnValue "idsta_ord", 1
End If

Session.Contents.Remove("kt_order_id")
Set Trigger_Custom = nothing

```

10. The completed dialog box (for PHP) should look as the image below:

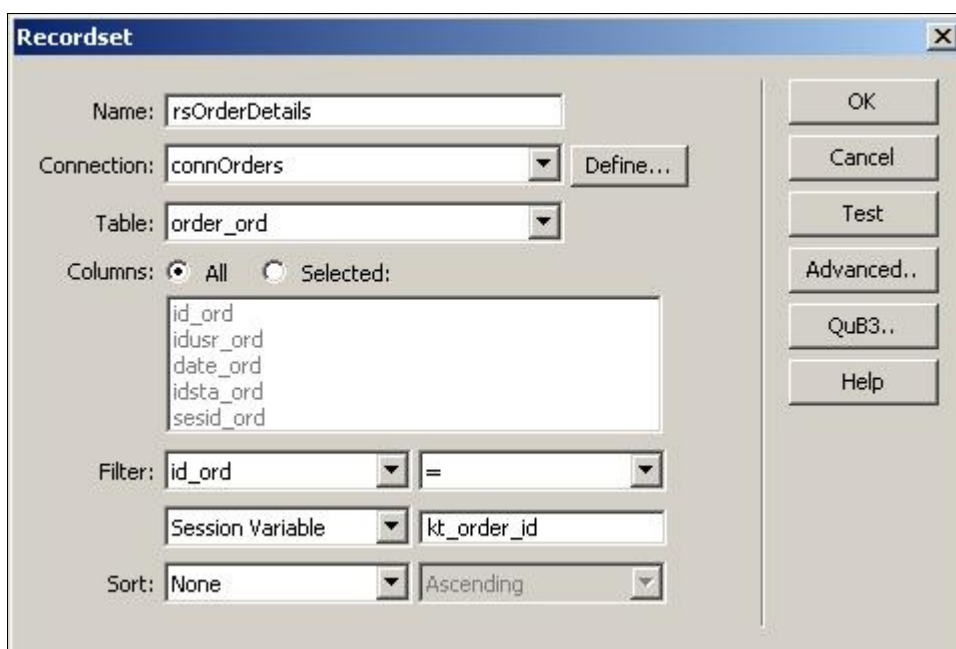


11. To close the **Custom trigger** dialog box click the **OK** button.
12. The last step to take in order to complete the page is to add the trigger that sends an e-mail message if not enough products are available. This e-mail message will contain two links: one that allows cancelling the order - by deleting it from the database tables, and one that

switches its status to acknowledged. The pages are *delete_order* and *acknowledge_order*. These pages will delete, and update a record, respectively, given the URL parameter passed from the mail message. A security issue arises, as if the primary key is passed directly, one can delete or acknowledge multiple orders, thus corrupting the database information. To ensure this does not happen, another unique identifier, that is much more difficult to guess must be used.

One solution is to use a new table column to store a unique random key that is generated with the mail sending, and to delete it once the operation is finished. The other option at hand is to use the existing *sesid_ord* column, that already stores a unique and hard to guess value: the user's session identifier. The choice is up to you, both methods will work. In this tutorial the latter option was chosen, as it does not involve altering the table.

13. The *delete_order* and *acknowledge_order* pages will be created later on. For now, the mail message will contain links to those pages, carrying the *sesid_ord* value as URL parameter.
14. In order to use the *sesid_ord* column as dynamic data in the e-mail message, you must first retrieve it from the database. To do so, create a new recordset on the *order_ord* table, filtered after the *kt_order_id* session variable, as shown below:



15. Before adding email sending capabilities to the page, you must first retrieve the user's e-mail address. Since the username is actually the email address, it is already available, as a session variable: *kt_login_user*:



16. To add the mail sending option, you will use the **Send E-mail** trigger. Apply it from **Server Behaviors > + > MX Kollection > Send E-mail**. Configure the user interface as shown below:

- For the sender address, enter your site's administrative or sales address. You can also use the default sender (the one automatically filled in from the InterAKT Control Panel)

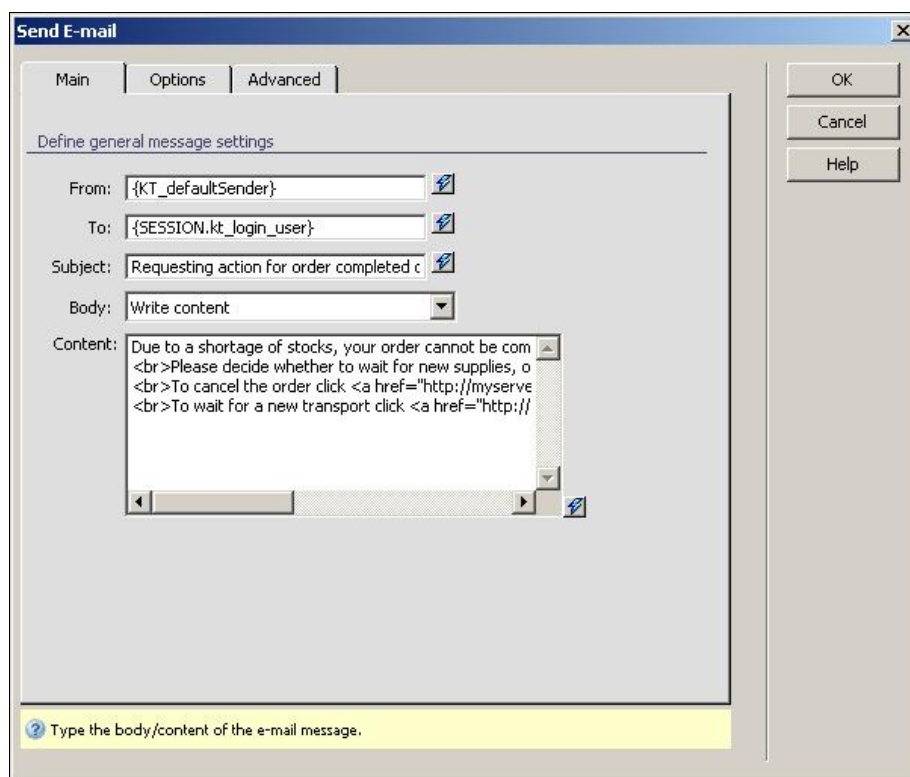
- For the recipient's address you will have to use the InterAKT Dynamic Data to generate the mark-up code that retrieves it from the kt_login_user session variable:

The screenshot shows a dialog box titled "InterAKT Dynamic Data" with a "Basic" tab. It contains the following elements:

- A section titled "Get dynamic values from the source indicated below" with a "Get values from:" dropdown menu set to "Session Variable" and a "Variable:" dropdown menu set to "kt_login_user".
- A section titled "The corresponding mark-up code" with a text field containing the code: `{SESSION.kt_login_user}`.
- A yellow tooltip at the bottom left that reads: "This is the generated mark-up code, based on your dynamic data selection."
- Buttons for "OK", "Cancel", and "Help" on the right side.

- For the Subject, you can use both entered text and dynamic data (e.g. display the order date next to the subject message - Requesting action for order completed on {rsOrderDetails.date_ord})
- Select the Write content option in the Body drop-down menu. For the content, you can use as well static and dynamic data - and HTML tags. The links for the two options must appear as well. An example content follows:

```
Due to a shortage of stocks, your order cannot be completed at
this time.
<br>Please decide whether to wait for new supplies, or cancel
the order.
<br>To cancel the order click <a
href="http://myserver.com/delete_order.php?ordid={rsOrderDetail
.sesid_ord}" >here</a>
<br>To wait for a new transport click <a
href="http://myserver.com/acknowledge_order.php?ordid={rsOrderD
etail.sesid_ord}" >here</a>
```



The screenshot shows a 'Send E-mail' dialog box with three tabs: 'Main', 'Options', and 'Advanced'. The 'Main' tab is selected. The dialog contains the following fields:

- From:** {KT_defaultSender}
- To:** {SESSION.kt_login_user}
- Subject:** Requesting action for order completed c
- Body:** Write content
- Content:** Due to a shortage of stocks, your order cannot be com

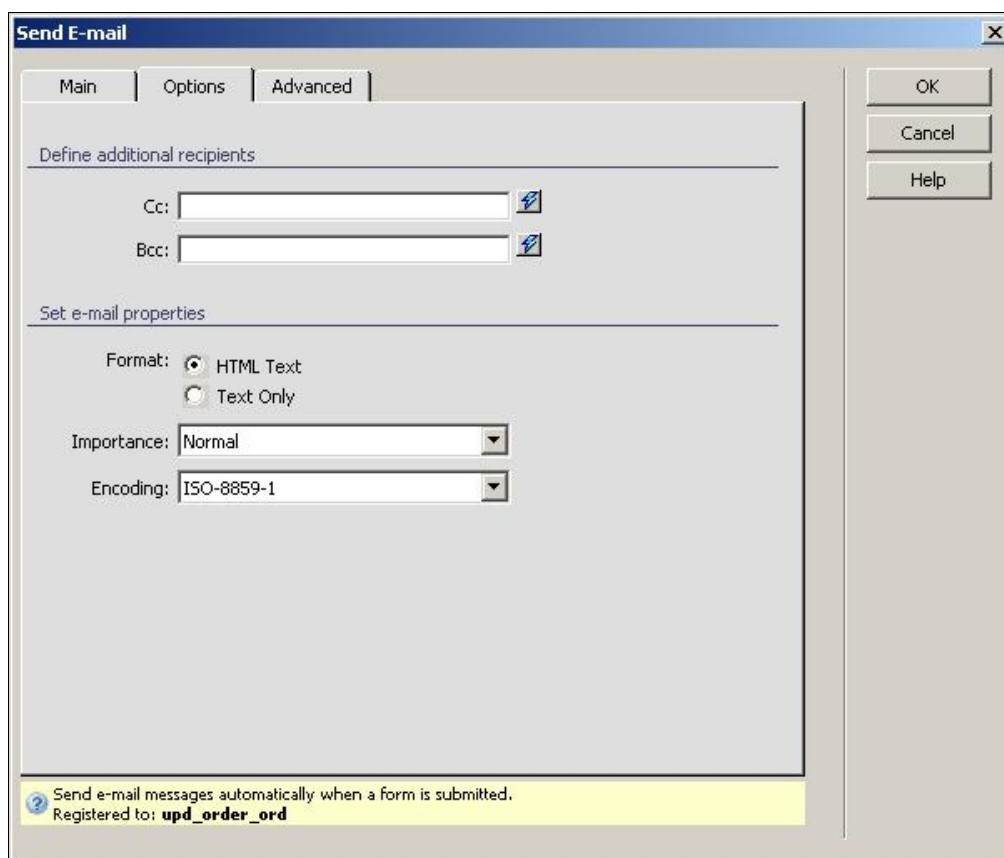
Please decide whether to wait for new supplies, o

To cancel the order click <a href="http://myserve

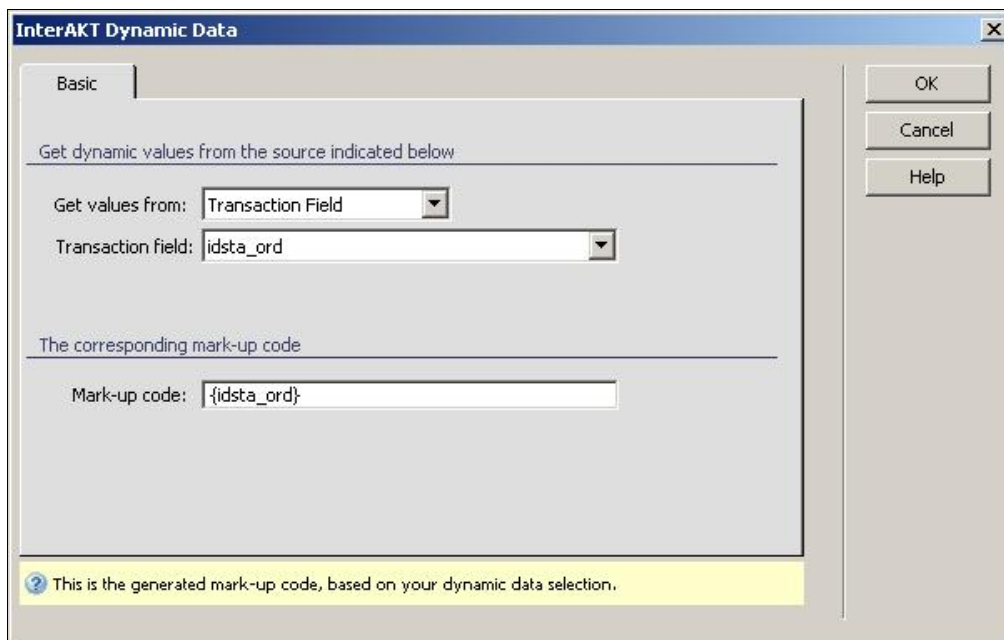
To wait for a new transport click <a href="http://

At the bottom of the dialog, there is a yellow bar with a question mark icon and the text: 'Type the body/content of the e-mail message.'

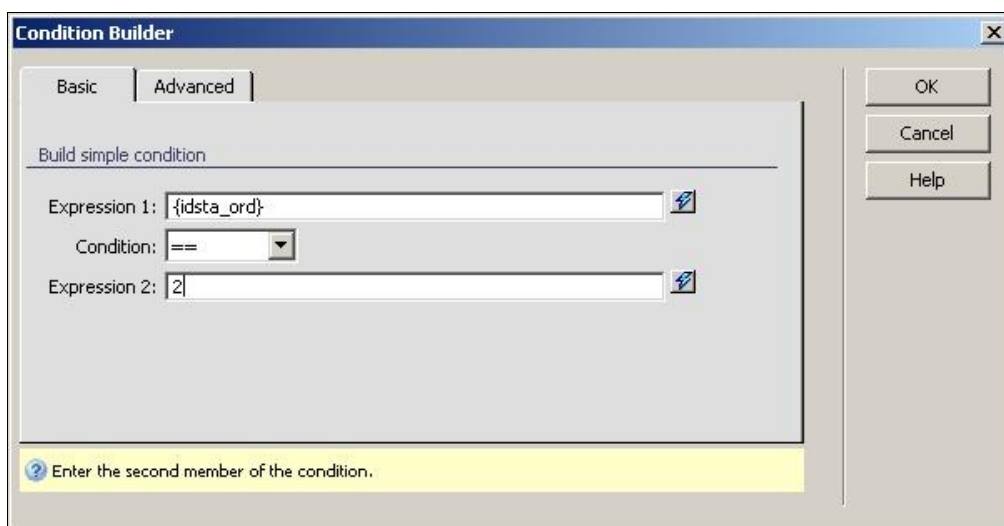
- The links in the mail body above will pass the `ord_id` URL parameter (you can name the parameter any way you want, as long as you use the same name in the page that use them) which contains the order's associated session ID from the `order_ord` table.
17. In order for the links to be displayed correctly, you must also set the message type to **HTML** instead of plain text. To do this, switch to the **Options** tab and select the **HTML** text option.



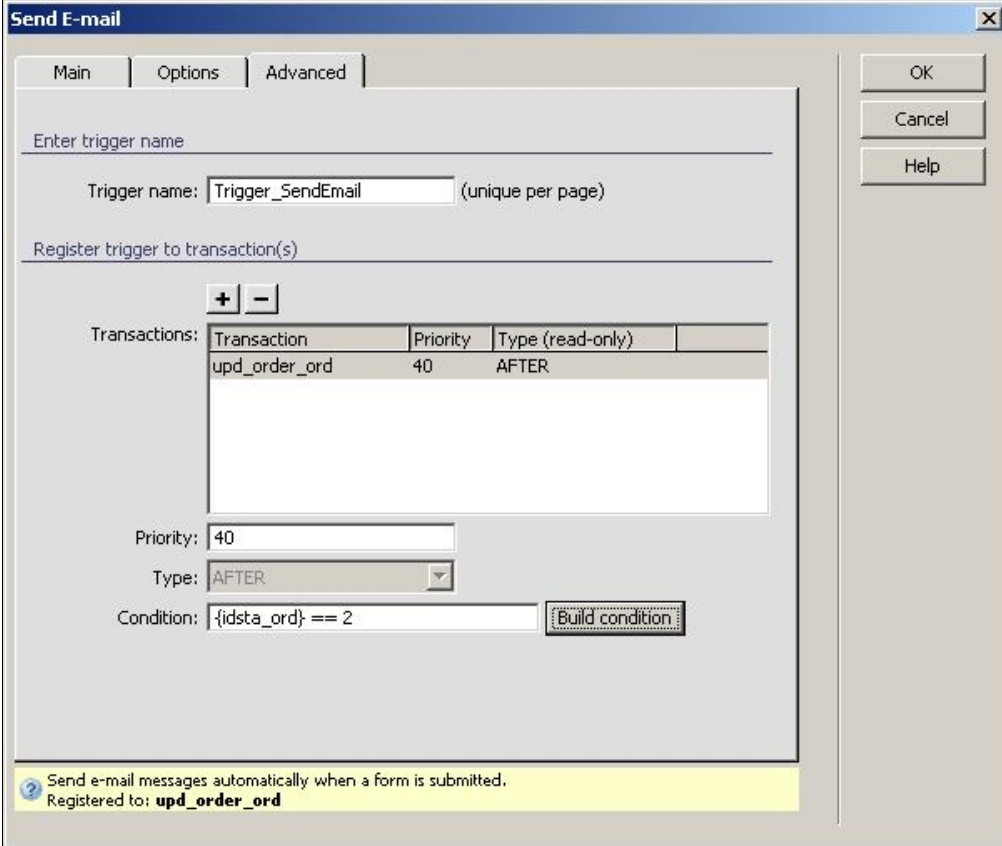
18. The last step to perform when creating the e-mail is to make its execution conditional. Do not close the **Send E-mail** user interface just yet. Before starting to work with the **Send E-mail** trigger, it has been mentioned that it must be executed only when there are not enough stock to satisfy the order. The existence of enough stock is reflected by the order status, after the transaction completes (its status is updated to either 2 - waiting for acknowledgment, or 1 - pending). The **Send E-mail** trigger is by default of the **AFTER** type, and is executed after the transaction completes, and the status is updated in the transaction field. This will be the condition to check before starting the mail trigger.
19. To add a starting condition to the mail trigger switch to the **Advanced** tab of the user interface. In the lower part of the window, a text field labeled **Condition**, and a **Build condition** button are displayed. Click on the **Build condition** button to open the **Condition builder**.
20. In the dialog box that opens, you can select the two expressions to compare, as well as the operator between them. For each you can use static and dynamic data.
21. For the first expression, click on the **InterAKT Dynamic Data** icon. In the new dialog box, select the **idsta_ord** transaction field (it is the one used in the update transaction, that now stores the new status):



22. Click **OK** to close the window, and place the generated mark-up code into the first expression's text field.
23. Next comes the operator. The condition states that the order status must be equal to the value 2 (waiting) to send the e-mail. Therefore, select the equal sign (==) from the drop-down menu.
24. For the second expression, you will use a static, entered value. Since the status must be waiting for acknowledgment, the status code must equal 2. Enter the value 2 in the text field.



25. At this point, the trigger's starting condition is completely defined (both expressions and operator) so you can click on the **OK** button to close the dialog box and return to the **Send E-mail** trigger user interface. The condition you defined will be also displayed in the trigger's **Condition** text field.



The image shows a 'Send E-mail' dialog box with three tabs: 'Main', 'Options', and 'Advanced'. The 'Main' tab is selected. It contains the following fields and controls:

- 'Enter trigger name' section: A text box containing 'Trigger_SendEmail' with '(unique per page)' to its right.
- 'Register trigger to transaction(s)' section: A table with columns 'Transaction', 'Priority', and 'Type (read-only)'. It contains one row with 'upd_order_ord', '40', and 'AFTER'. Above the table are '+' and '-' buttons.
- 'Priority' section: A text box containing '40'.
- 'Type' section: A dropdown menu showing 'AFTER'.
- 'Condition' section: A text box containing '{idsta_ord} == 2' and a 'Build condition' button.

At the bottom, a yellow banner contains the text: 'Send e-mail messages automatically when a form is submitted. Registered to: **upd_order_ord**'. On the right side of the dialog, there are three buttons: 'OK', 'Cancel', and 'Help'.

To apply the Send E-mail server behavior click on the OK button. This will close the dialog box, and add a new server behavior to the page. The order can now be completed. In the next section of the tutorial, you will create the pages that process the user's answer to the low stock e-mail message: pages that allow canceling or acknowledging the order.

Process User Reply for Order

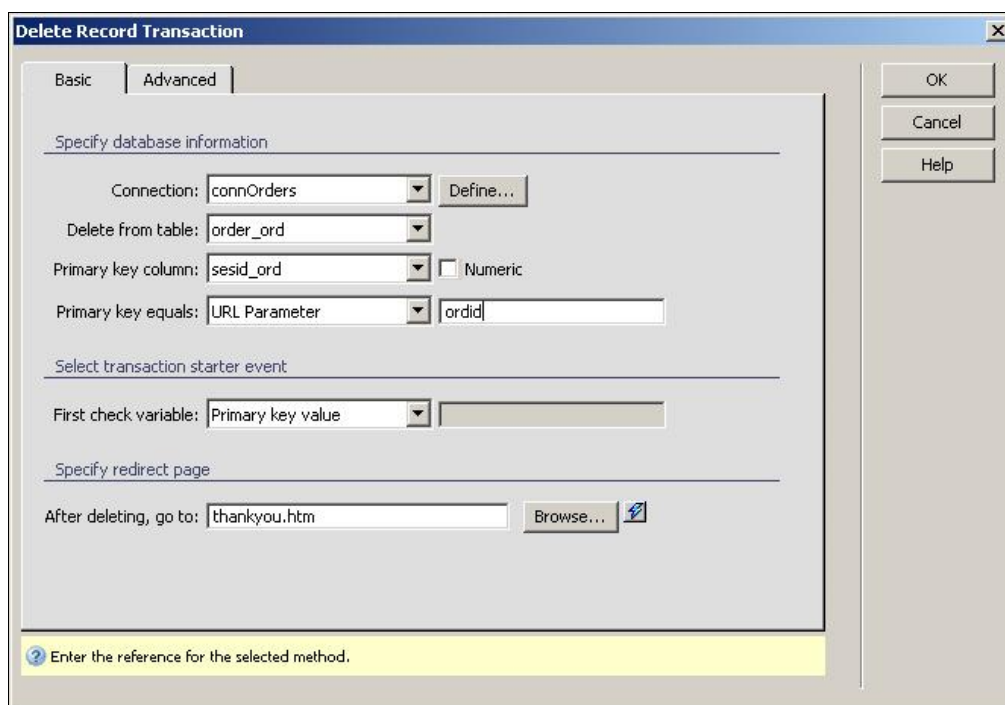
When the order cannot be delivered, because of insufficient stocks, the user receives an e-mail message offering two options: to cancel the order or to wait until resupply. Each of these options is materialized by a page: one that deletes the order - for cancel, and one that updates the order status - for the waiting option. In what follows, you will create these pages in the site root:

- *delete_order*
- *acknowledge_order*

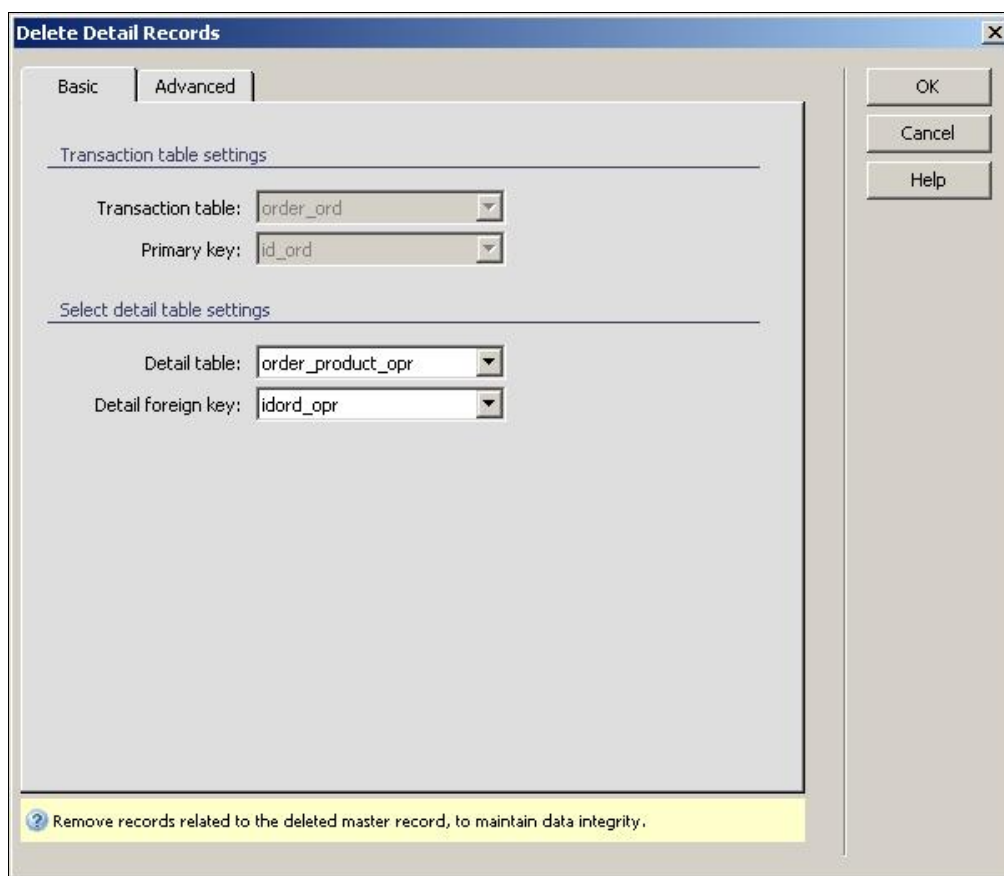
The delete order page is called when the user wants to cancel its order, and receives as an URL parameter the order's associated session ID. Similar to the *clear_order* page, not only direct order details from the *order_ord* page are to be deleted, but also the detail records in the *order_product_opr* table, storing pairs of orders - products. What is unlike the *clear_order* page is the fact that the session variable no longer exists, and must not be deleted.

To build the page, follow the next steps:

1. Open the *delete_order* page in **Dreamweaver**.
2. Apply the **Delete Record Wizard** from the **MX Kollection** tab of the Insert bar. Configure it to use the database connection defined at the beginning of this tutorial (*connOrders*) and to delete records from the *order_ord* table. For the record primary key use the *sesid_ord* column, and set it to equal the *ordid* URL parameter. The *ordid* parameter is passed from the e-mail message created in the previous section. After delete, the *thankyou* page will be opened.



3. Close the wizard by clicking on the OK button. The delete transaction will be added to the page.
4. Next apply the **Delete Detail Records** server behavior to remove the related records from the *order_product_opr*. You can access it from the **Server Behaviors tab > + >MX Kollection > Form Validation**.
5. The master table and primary key is automatically selected from the delete transaction on page, and all that remains is to select the detail table (*order_product_opr*) and the detail foreign key (the column storing the order ID - *idord_opr*).



6. Click on **OK** to add the server behavior on page. Then save and upload the page to the server.

The next page to create is the one handling the status change for the order, when the user decides to keep the order, and wait for new products. This page performs a transparent update, changing the status from "Waiting for acknowledgment" - code 2 into "Acknowledged" - code 3. Similar to the *delete_order* page, it will receive the session ID associated to the order as an URL parameter.

To build the acknowledgment page, follow the next steps:

1. Open the *acknowledge_order* page in **Dreamweaver**.
2. Apply the Update Transaction server behavior on the page. You can access this server behavior from the **Server Behaviors > + > MX Kollection > Forms > Advanced**. Configure the user interface options as follows:
 - In the Basic tab set the database connection defined at the beginning of the tutorial, the table to update records into - *order_ord*. For the starting and primary key conditions, use the same settings as for the Delete transaction above. To make the transaction start automatically, set the starter event to Entered value, and for the reference enter a value (e.g. 1). The redirect is still to the *thankyou* page

- In the **Fields** tab, thanks to the user interface persistence, only the *idsta_ord* field is left. It is already set to use entered value, so simply add the correct value in the corresponding text-field: 3 (the code for acknowledged). Then close the user interface by clicking on the **OK** button:

3. Now the acknowledgement page is completed. Save it and upload it to the server.

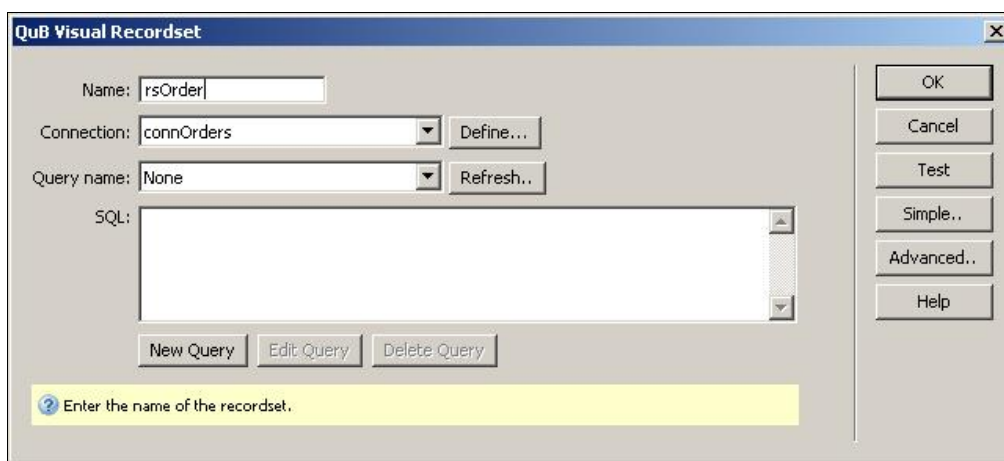
With this, the main front-end has been created. It allows users add products to their orders, clear or complete the order, and even take action if the order cannot be met. What is missing is a way for the user to view its current order, and that will be implemented next.

Create page to display current order

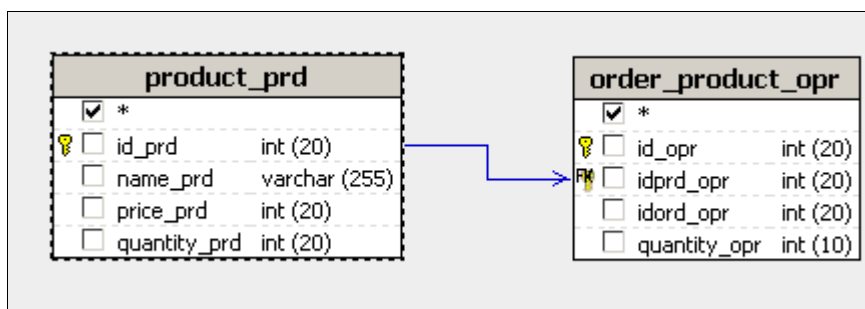
In this last section dedicated to building the application front-end you will create a page that lists the order details for the currently opened order. The page will use a filtered recordset and a dynamic table to display all products in the current order.

To create this page, follow the steps below:

1. Open the *view_order* page in **Dreamweaver**.
2. Add a new recordset, by clicking on the **Plus (+)** button of the **Bindings** tab. Then click on the **QuB3** button to open the **MX Query Builder Dreamweaver** user interface. **MX Query Builder** is a tool that allows creating queries in a visual manner, using only simple operations. For more information regarding the MX Query Builder, consult its user manual.
3. In the user interface that opened, select the database connection defined at the beginning, and enter a name for the recordset.



4. After you have defined the recordset name and database connection. click the **New Query** button. Enter its name in the dialog box that loads and click **OK**. The main **MX Query Builder** user interface will load next.
5. The user interface displays the tables in the left lower side, and the **SQL** query, results and columns in the center - bottom of the screen. To select the tables to use, simply check their check boxes in the left side of the interface. Do this for the *product_prd* and *order_product_opr* tables. After they are clicked, the tables appear in the center of the screen, with a list of fields. Between the *product_prd* and the *order_product_opr* table there is a one-to-many relation. To define the relation and create the JOIN code at the same time, drag and drop the **id_prd** field from the *product_prd* table onto the **idprd_opr** field. A blue arrow will display the relation visually:



6. Now the query will retrieve all products, and only details from the *order_product_opr* table for the ones that have associated fields. To restrict the results only to products that are actually involved in the order, right-click the arrow, and select **Edit**. In the dialog box that opens, select the first option - **Only include rows where joined fields are equal**, and click **OK**.

QuB :: Edit Relation -- Web Page Dialog

Edit Relation

Left Table Name: Right Table Name:

Left Field Name: Right Field Name:

Relation Type

Only include rows where joined fields are equal

Include all records from the **product_prd** table and only those from **order_product_opr** table where joined fields are equal

Include all records from the **order_product_opr** table and only those from **product_prd** table where joined fields are equal

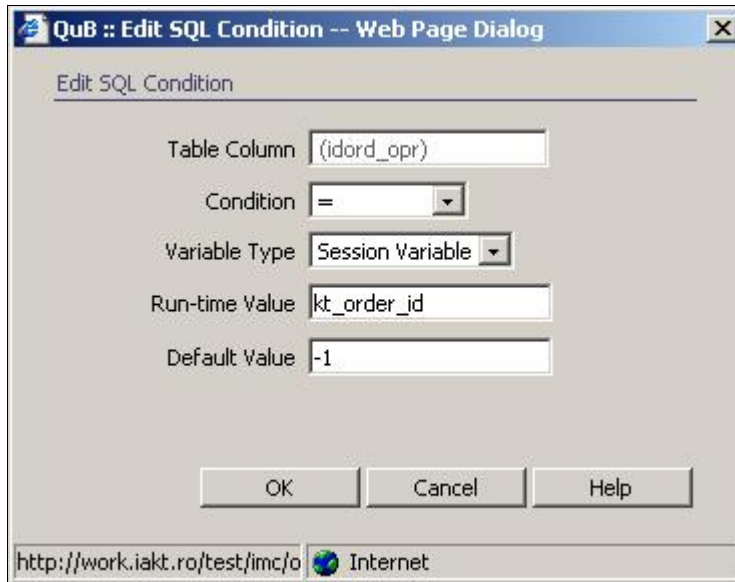
http://work.iakt.ro/test/imc/order_mngmt_final/includes/ Internet

7. Next check all fields you want the query to retrieve: the **name_prd** and **price_prd** fields from the *product_prd* table, as well as the **quantity_opr** and **idord_opr** fields from the *order_product_opr* table.

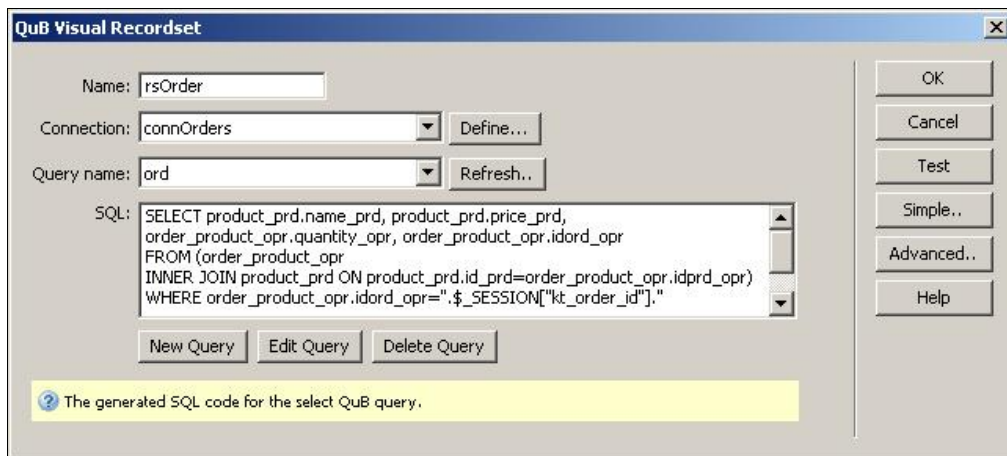
| product_prd | |
|-------------------------------------|------------------------|
| <input type="checkbox"/> | * |
| <input type="checkbox"/> | id_prd int (20) |
| <input checked="" type="checkbox"/> | name_prd varchar (255) |
| <input checked="" type="checkbox"/> | price_prd int (20) |
| <input type="checkbox"/> | quantity_prd int (20) |

| order_product_opr | |
|-------------------------------------|-----------------------|
| <input type="checkbox"/> | * |
| <input type="checkbox"/> | id_opr int (20) |
| <input checked="" type="checkbox"/> | idprd_opr int (20) |
| <input checked="" type="checkbox"/> | idord_opr int (20) |
| <input checked="" type="checkbox"/> | quantity_opr int (10) |

8. Now to filter the results, so that only the current order is retrieved: in the columns area, click the ... button next to the **idord_opr** column. In the dialog box that opens, select the **Condition** (equals), the **Variable Type** (session) and the **Run-time Value** (the reference for the session variable storing the order ID - **kt_order_id**).



9. Click the **OK** button to close the dialog box. Then click the save icon to save the query for later use, and close the **MX Query Builder** user interface (**Query > Close**)
10. Back in **Dreamweaver**, you have to hit the Refresh button in order for the query to be displayed in the **SQL** area:



11. Click **OK** once more to finish with the recordset.
12. To display data retrieved by the recordset, you must use a dynamic table. Add one from the **Application** tab of the **Insert** bar. Configure it to use the *rsOrders* recordset created earlier, and to display all records.



13. From the page, remove the unwanted fields (the *idord_opr* one). Then change the header of the table (*name_prd* > Product name, *price_prd* > Price, *quantity_opr* > Quantity). Save the page and upload it to the server. After adding a few products to the order, click the `view_order` link. It will display a table with all added products:

| Product name | Price | Quantity |
|---------------------|--------------|-----------------|
| Keyboard | 30 | 1 |
| Monitor 17" | 150 | 2 |
| Monitor 17" | 150 | 14 |

[Product list](#) [Complete order](#) [Clear order](#)

Now the site front-end is finished, and you can move on and create the administrative section, which will allow managing the orders.

Manage Orders

In this part of the tutorial you will build the administrative section for the order management application, the section where administrators can view orders, and forward them to the other departments. The product and user administrative sections will not be built, as they are not in the scope of this tutorial. However, they can be done in a similar manner as the one you will build.

1. The administrative section contains several pages, that implement a master-detail display of existing orders:
2. The first list displays the basic order details - like the user name, date of order and the order status. For each entry, a details button allows listing the products that were ordered, with name, price and quantity.
3. At last, for each order, besides the list, a form will be created that will allow changing the status. A button next to each order will allow access.

The product used to implement the entire administrative section is **NeXTensio** - the list for the first two pages, and the form for the last.

List, sort and filter orders

In this section of the tutorial you will build the pages that will display the order list, and details. Before continuing with the page creation, it is recommended that you read and understand the way a master-detail relation can be implemented using **NeXTensio** in the Master-Detail How to.

The first page to create is the master **NeXTensio** list, displaying the basic order details from the `order_ord` table. To complete it, follow the next steps:

1. Open the `admin/index` page in **Dreamweaver**.
2. Start the **NeXTensio List Wizard** from the **MX Kollection** tab of the **Insert** bar. The wizard is divided into several steps which you have to configure in order to create the list.
3. In the first step of the wizard, you have to configure the data source information:
 - The list will be generated using a table, and not a recordset
 - Select the connection defined at the beginning of this tutorial
 - Set the table used to retrieve data from as `order_ord`
 - The primary key column is `id_ord`. If you've followed the tutorial so far, the `sesid_ord` column might be selected due to the user interface persistence
 - Also in the first step you have to specify which file will store the **NeXTensio** form, as well as the number of records to display on page.

The screenshot shows the 'Create NeXTensio List Wizard' dialog box at Step 1/4: 'Select the table to display'. The wizard is configured with the following settings:

- Specify the method to retrieve data:** Get data from: Table
- Specify data source information:** Connection: connOrders, Table: order_ord, Primary key column: id_ord (checked as Numeric)
- Specify the detail page:** Detail page: form.php, Number of records: 10

A yellow tooltip at the bottom reads: 'Select the column that uniquely identifies each record.' Navigation buttons include < Back, Next >, Finish, Cancel, and Help.

4. In the second step of the wizard you have to define which columns will be displayed from the selected table, and which values to display. If you need to display the actual label for a foreign key column (e.g. the `idusr_ord` column), simply use the look-up table option, to automatically create the JOIN operation, and display the correct label.
 - Remove the `sesid_ord` field from the grid by selecting it and clicking on the **Minus (-)** button.

- Set the **idusr_ord** field to use a look-up table. In the table drop-down menu select the *user_usr*, and in the label column the **name_usr** column. By doing this, the user name is displayed instead of the ID
- Do the same for the **idsta_ord** column, but use the *status_sta* table, and the **name_sta** column for the label.
- Also change the labels for the **idusr_ord** and **idstaOrd** fields, and replace the existing ones with User and Status.

Step 2/4: Select the list columns

Specify list columns

List columns: + -

| Column | Header | Char width |
|-----------|---------|------------|
| idusr_ord | User: | 20 |
| date_ord | Date: | 20 |
| idsta_ord | Status: | 20 |

Header: Status:

Char width: 20

Get value from: Current table Look-up table

Database table: status_sta

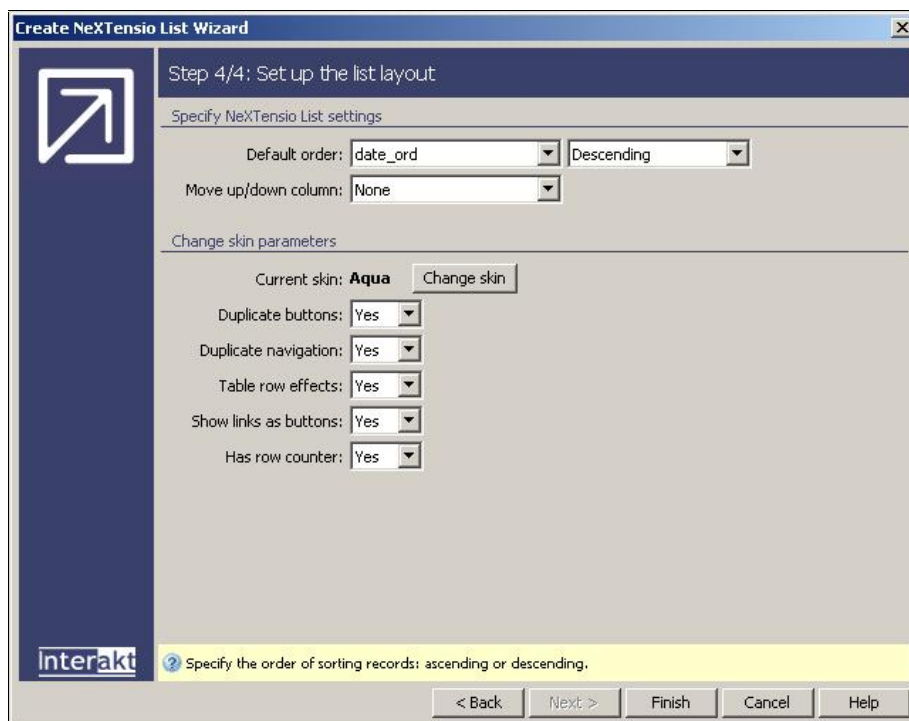
Label column: name_sta

Value column: id_sta

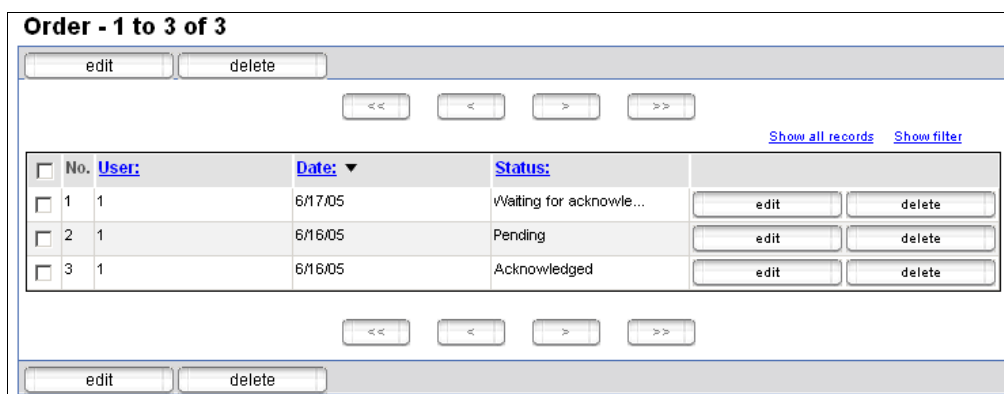
Interakt Enter the label for the selected field.

< Back Next > Finish Cancel Help

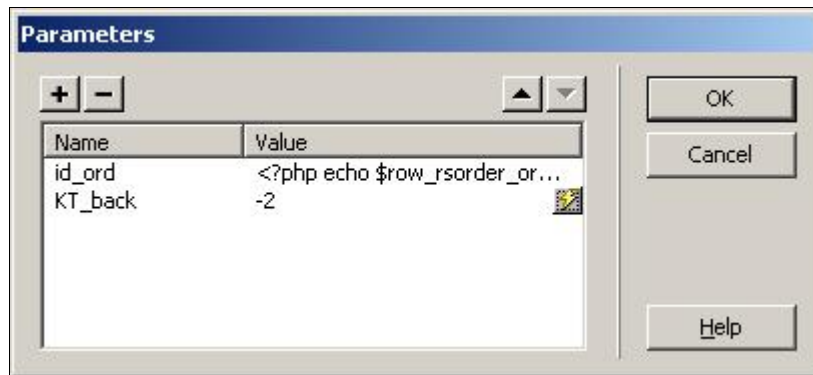
5. The third step of the wizard allows defining the filter elements to use for each of the list columns. For columns using look-up tables, the element is automatically set to menu, and the dynamic content is based on the same table as the one selected in the wizard's second step. You do not need to modify anything in this step, so simply skip over it.
6. The last wizard step offers the possibility to set list settings: the ordering column, skin, navigation and button options.
 - In the Default order drop-down menu select the *date_ord* column, and set it to be ordered Descending. this way, new orders will appear first in the list.
 - Set the skin of your choosing by clicking on the change skin button and selecting from the drop-down menu the desired one. In the images below, the Aqua skin has been used.
 - Leave the rest of the options at their default values, and click Finish to close the wizard.



7. When you preview the page in the browser, it will display the first 10 orders, and some additional action buttons: add new, edit and delete. Only edit and delete are of use to administrators, as they should be able to edit (change the status) or delete (remove) an order. An administrator must not be able to create a new order. Therefore, in **Dreamweaver**, select the add new dynamic link, and the associated drop-down and delete them.



8. Back in **Dreamweaver**, before the edit link, enter the details text. Select it and right-click on it. From the pop-up menu select the Make link option. Set it to point to the *detail_order*. The link will pass two URL parameters:
- The *id_ord* parameter, with the value retrieved from the *id_ord* field of the List recordset
 - The *KT_back* parameter, with the value -2. The *KT_back* parameter is used by the NeXTensio list and form to ease navigation between the two. It is in fact an array of pages that is passed through a session variable. the -2 value indicates the place in the array that stores the page of the original list.



9. To make the link use the same properties as the edit and delete (show as buttons and skin properties) set the new link's class to **KT_link**. To do so, select the details link, and switch to code view. Enter the following code after the last URL parameter of the corresponding `<a>` tag- outside the quotes - `class = "KT_link"`.
10. Save the page and upload it to the server. If you preview the page, the details link will appear next to edit and delete, and will look the same - it uses the same properties.

The next page to create is the detail list for the orders. This will also be implemented using NeXTensio. To create the page, follow the next steps:

1. Open the *detail_order* page in **Dreamweaver**.
2. Start the **NeXTensio List Wizard** from the **MX Kollection** tab of the **Insert** bar. Configure each step of the wizard as shown below:
3. In the first step, select the database connection defined at the beginning of this tutorial, and the database to retrieve records from - *order_product_opr*. Also enter a name for the file containing the form. The actual form will not be created, as an order's contents cannot be modified.

Step 1/4: Select the table to display

Specify the method to retrieve data

Get data from: Table

Specify data source information

Connection: connOrders Define...

Table: order_product_opr

Primary key column: id_opr Numeric

Specify the detail page

Detail page: detail_form.php Browse...

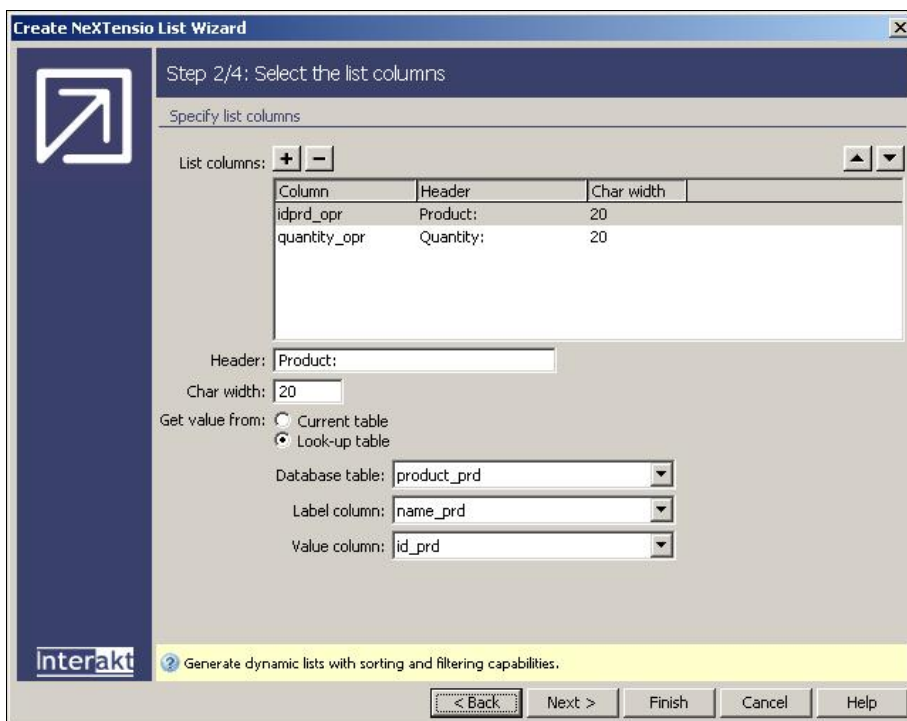
Number of records: 10

Interakt Specify the page that contains the NeXTensio form.

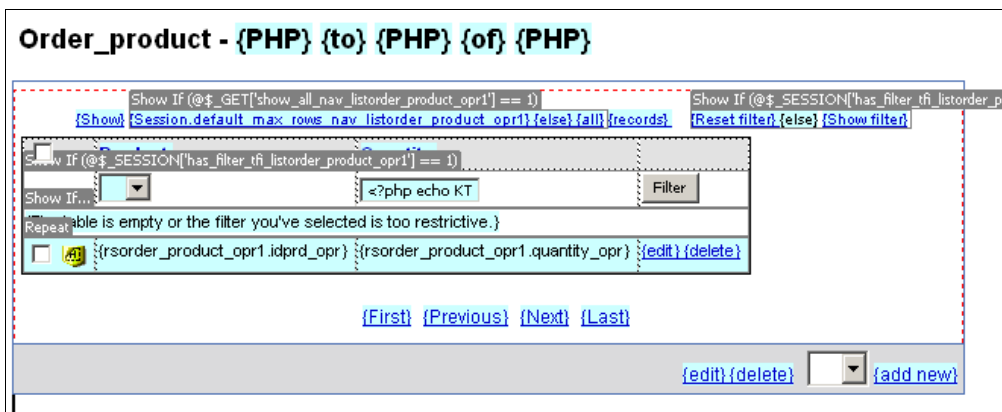
< Back Next > Finish Cancel Help

4. In the second step you can define the table columns to retrieve and display in the **NeXTensio** list. There are three available fields: **idord_opr**, **idprd_opr** and **quantity_opr**. Configure each field as follows:
 - Remove the idord_opr field. It will not be used in the list.
 - For the idprd_opr field, select the Look-up table option. In the table drop-down menu, select the product_prd table, and in the label drop-down select name_prd. Also change the label to Product.

- The quantity_opr field can remain at its default options.



5. The filter is automatically configured in step three - with the idprd_opr field set as menu, and the quantity field as a text field. So you can move right to the last step.
6. In the last step, select the Default order column as idprd_opr, to order the list after the product ID. Leave all other options at their default values. Click the Finish button to close the NeXTensio List Wizard and add all elements to the page, both HTML and code.



As you can see in the image above, the list displays the edit, delete and add new buttons. For the order detail, none of these are relevant, so you can delete them all. Also delete the drop-down menu next to the add new link, in the form footer.

In place of the form footer elements, add a link that will allow administrators return to the master list. Since you are using a master-detail construct with NeXTensio, the back list must be created in the correct manner: by using the NeXTensio back page.

To create the link, type its text in the page (e.g. Back to master list). Select the text and right-click it. From the pop-up menu select the Make link option, and set it to point towards the includes/nxt/back.php page. (if using a different server model, the file extension is different). Also set it to pass the KT_back URL parameter with the value -2 (similar to step 8 above).

As shown in the Master-Detail How to, using the file for link is not the only step you must take. The class that handles links using the KT_back parameter must be available to the page. To make it

available, you have to include in your page the includes/nxt/KT_back class file.

For **PHP_MYSQL**, the code would look similar to the following:

```
// Load the KT_back class
require_once('includes/nxt/KT_back.php');
```

For **ColdFusion**:

```
<cfinclude template="../includes/nxt/KT_back.php">
```

For ASPVBScript:

```
<!--#include file="../includes/nxt/KT_back.asp" -->
```

Now you have to filter the list. By default, it will display all details for all orders, and not only for the one selected in the master list. To filter the list, you must filter its recordset:

1. Double-click the *ListRecordset* from the Bindings panel. This will open the Recordset dialog box, in Advanced view.
2. The **SQL** code until now should look similar to the following:

```
SELECT product_prd.name_prd AS idprd_opr,
order_product_opr.quantity_opr, order_product_opr.id_opr
FROM order_product_opr LEFT JOIN product_prd ON
order_product_opr.idprd_opr = product_prd.id_prd
WHERE NXTFilter
ORDER BY NXTSort
```

3. To display only the relevant records, the recordset must be filtered after the *id_ord* URL parameter passed by the details link. To add the filter, first define a new variable. Click the Plus (+) button on top of the Variables grid, and configure it as follows:
Note: For **ColdFusion**, you can ignore this step for now. The same procedure is easier to do as part of step 4.

4. Now you will have to edit the **SQL** query. It already contains a condition - WHERE NXTFilter - that allows the **NeXTensio** Filter to function. To add a new condition that will be evaluated on the same level as the first one, use the AND operator. The condition to add is: `AND idord_opr = orderid`. Add this after the existing condition and the recordset will be filtered. The new **SQL** query will look like:

```
SELECT product_prd.name_prd AS idprd_opr,
order_product_opr.quantity_opr, order_product_opr.id_opr
FROM order_product_opr LEFT JOIN product_prd ON
order_product_opr.idprd_opr = product_prd.id_prd
WHERE NXTFilter AND idord_opr = orderid
ORDER BY NXTSort
```

For **ColdFusion**, add `AND idord_opr = #URL.id_ord#` after the `WHERE` `#PreserveSingleQuotes(SESSION.filter_tfi_listorder_product_opr1)#` so the finished **SQL** code will look like this:

```
SELECT product_prd.name_prd AS idprd_opr,
```

```

order_product_opr.quantity_opr, order_product_opr.id_oprFROM
order_product_opr LEFT JOIN product_prd ON order_product_opr.idprd_opr
= product_prd.id_prd WHERE
#PreserveSingleQuotes(SESSION.filter_tfi_listorder_product_opr1)# AND
idord_opr = #URL.id_ord#ORDER BY
#SESSION.sorter_tso_listorder_product_opr1#

```

5. Click the **OK** button to close the recordset dialog box. Save the page and upload it to the server.

Now when you preview the main list in the browser, and click on one of the order's detail button, the detail list will only display the products that the order is composed of:

Order_product - 1 to 2 of 2

[Back to master list](#)

[Show all records](#) [Show filter](#)

| <input type="checkbox"/> | No. | Product: | Quantity: |
|--------------------------|-----|----------|-----------|
| <input type="checkbox"/> | 1 | Mouse | 1 |
| <input type="checkbox"/> | 2 | Keyboard | 2 |

[Back to master list](#)

To further improve the list, you can enter a better title (replace the *order_product* with Order details), or modify the list recordset to display the product price as well.

For the overall application improvements, you can create product and user management pages, using **NeXTensio** lists and forms.

Modify order status

In this section of the tutorial you will create the **NeXTensio** form that works with the **NeXTensio List**, and allows modifying its elements. The **NeXTensio Form** is a multi-functional form, allowing all of the regular operations: insert, update and delete. Since adding a new order is not allowed, you will have to delete some of the elements on page.

To build the form page, follow the next steps:

1. Open the form page in **Dreamweaver** (it is the page with the same name as specified in the first **NeXTensio List Wizard**).
2. Start the **NeXTensio Form Wizard** from the **MX Kollection** tab of the **Insert** bar.
3. In the wizard's first step, the connection and *order_ord* table should be automatically selected. If they are not, simply select them in the available drop-down menus. Click **Next** to proceed to the second step.
4. In the wizard's second step you define the fields that will be displayed in the form, and taken into account when executing various operations. Since the only changeable property is the status, remove all other field except the *idsta_ord* from the grid. To remove a field, select it and press the **Minus (-)** button. Also thanks to the user persistence, the *idsta_ord* field should be configured to be displayed as a menu, and its recordset is automatically generated based on the choices made when creating the list (the look-up table).

The screenshot shows the 'Create NeXTensio Form Wizard' dialog box at Step 2/4: 'Configure the form fields'. The main area is titled 'Specify form fields properties'. It features a 'Form fields:' section with '+' and '-' buttons and a table with the following data:

| Column | Label | Display as | Submit as | Show on |
|-----------|---------|------------|-----------|-------------------|
| idsta_ord | Status: | Menu | Numeric | Insert and Update |

Below the table, there are input fields for 'Label: Status:', 'Show on: Insert and Update', 'Display as: Menu', and 'Submit as: Numeric'. There are also 'Char width: 5' and 'Max chars:' fields, and buttons for 'Menu Properties...' and 'Add Recordset'. At the bottom, there is a yellow informational bar with the text: 'The list of fields associated to the NeXTensio form.' and navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

5. The third step provides options to set validation rules for each field. Since you are already using a drop-down menu with predefined options, you can safely skip this step, and move on to the last wizard step.
6. The last step allows defining general form options: the move up/down column, skin, buttons and the use of grid when editing multiple records. Also, you can specify whether to display Insert as new button. And since adding is not allowed, select No in the drop-down menu:

Create NeXTensio Form Wizard

Step 4/4: Set up the form appearance and settings

Specify NeXTensio Form settings

Move up/down column: None

Change skin parameters

Current skin: **Aqua**

Duplicate buttons: Yes

Display as grid: Yes (when editing multiple records)

Copy down value: Yes (when editing multiple records)

Add "Insert as new" button: No

Interakt ? Adds a button that allows adding a new record based on an existing one.

< Back Next > Finish Cancel Help

- Now click on **Finish** to close the wizard and add all elements to the page. This will add both **HTML** elements and application logic to the page.

{Display error message.}

Show If (@\$_GET['id_ord'] == "")

{Insert} {else} {Update} Order

Repeat

{Initialize Counter} {Increment Counter}

Show If (@\$totalRows_rsorder_ord > 1)

{Record} {Display Counter}

Status: {Error}

Show If (@\$_GET['id_ord'] == "")

Insert {else} Update Delete Cancel

- Before saving and uploading the page, also remove the **Insert** button in the image above, as it is not necessary for the page.

Now, when you click the edit button next to an order in the master **NeXTensio** list, the following form will be displayed:

Update Order

Status: ▾

Now the basic order management application is completed. You can improve it by adding administrative modules for users, or even making it more complex.

Other Resources

Other Dreamweaver Extensions from InterAKT

- KTML
- MX Kart
- MX Site Search
- MX RSS Reader-Writer
- MX Dynamic Table Sorter
- MX Coder Pack
- MX Dynamic Charts
- MX CSS Dynamic Menus

Contact Us

- **Address:**
1-11 Economu Cezarescu ST, AYASH Center, 1st floor
Sector 6, ZIP 060754, Bucharest, Romania
- **Web:** <http://www.interaktonline.com/>
- **E-mail:** contact@interaktonline.com
- **Phone:** +4031 401.68.19 or +4021 312.51.91
- **Fax:** +4021 312.53.12

Copyright

Windows is a trademark of Microsoft, Inc.

Dreamweaver MX is a trademark of Macromedia, Inc.

Redhat is a trademark of Redhat, Inc.

Copyrights and Trademarks

Copyright 2000-2005 by InterAKT Online.

All Rights Reserved. This tutorial is subject to copyright protection.

PHAkt, ImpAKT, NeXTensio, MX Query Builder, Transaction Engine, MX Includes, KHTML, MX Kommerce, MX Kollection, MX Widgets, MX Looper, MX Dynamic Charts, MX CSS Dynamic Menus, MX Tree Menu, MX Form Validation, MX File Upload, MX Send E-mail, MX User Login, MX CSV Import-Export, MX Kart, MX Site Search, MX Dynamic Table Sorter, MX RSS Reader-Writer, MX Coder Pack, MX Dynamic Charts are trademarks of InterAKT Online.

All other trademarks are acknowledged as the property of their respective owners.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation.

No part of this document or of the associated product may be reproduced in any form by any means without prior written authorization of InterAKT Online, except when presenting only a summary of the tutorial and then linking to the InterAKT website.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Send comments and suggestions to products@interaktonline.com



InterAKT Online

Web: <http://www.interaktonline.com/>

E-mail: contact@interaktonline.com

Address: 1-11 Economu Cezarescu ST, AYASH Center, 1st floor, Sector 6, ZIP 060754, Bucharest, Romania

Phone: +4021 312.51.91

Fax: +4021 312.53.12