# Data Integrity and More Advanced Microflows

This case is the third in the series of introduction cases. In this case, you will add data integrity to the Domain Model and create more advanced microflows.

### Prerequisites

This case was prepared using version 7.21.0 of the desktop Mendix Modeler. You must have completed Introduction to Mendix case 2- Master/Detail Page and Microflows before starting this case.

## Open the Introductory Project

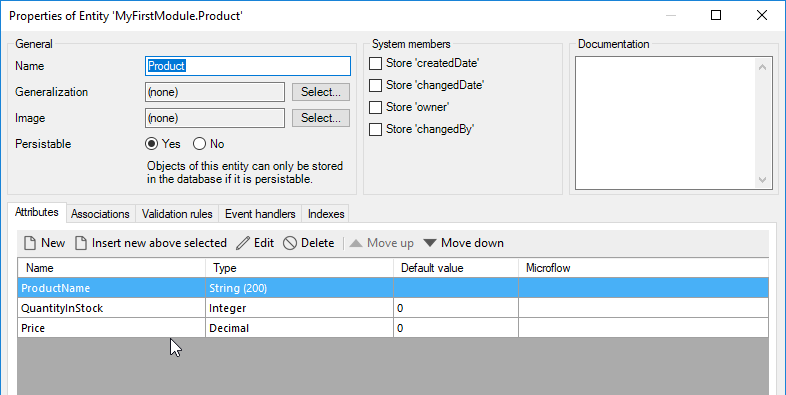
Open the project you created in case 1 and continued in case 2 of the introductory series in the desktop modeler.

## Configure Data Integrity

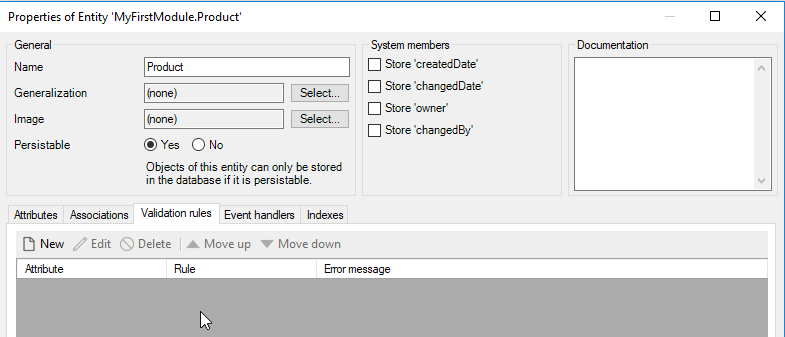
Data integrity can be managed by adding data validation and referential integrity constraints.

### Add Validation Rules

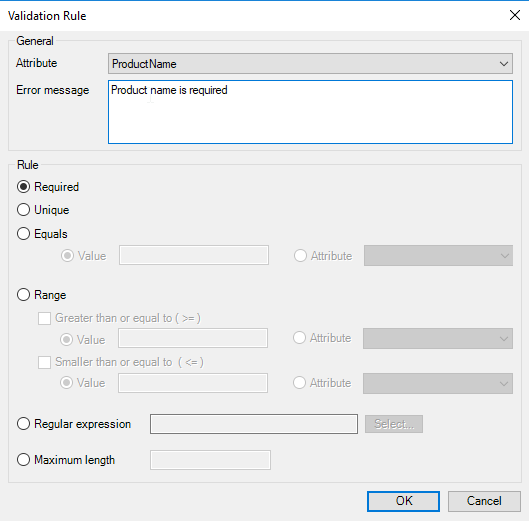
Open the Domain Model in the MyFirstModule and open the properties of the Product entity. The ProductName should be required and unique and the QuantityInStock and Price should be greater than or equal to zero.



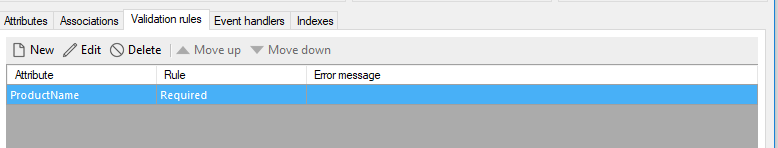
Click the Validation rules tab and click New to add a rule.



Select the ProductName attribute. The Required rule is selected by default. Add a message the user will see if the rule is violated. Click OK.

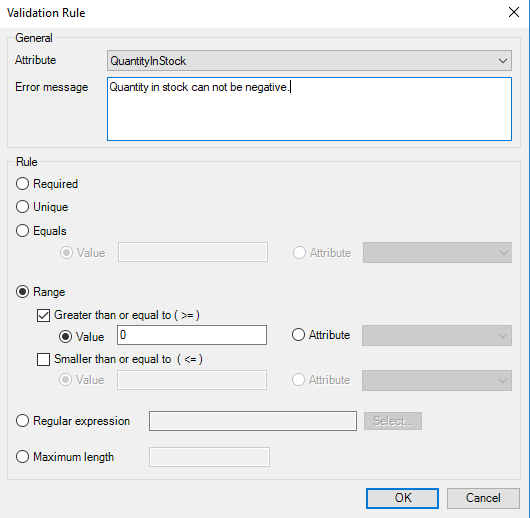


The rule is created.

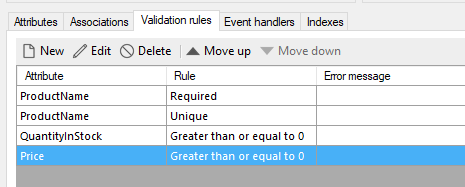


Add a second rule for ProductName to ensure it is unique.

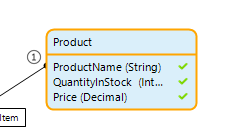
Next add rules to ensure that QuantityInStock and Price are greater than or equal to zero.



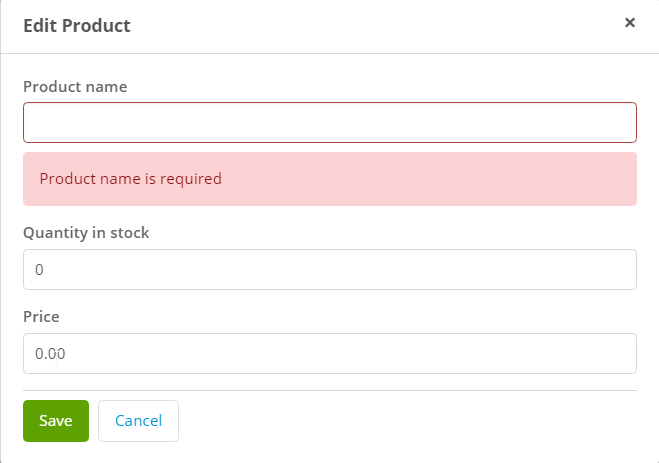
The final rules look like this:



Note you can see which attributes have validation rules by the green checks on the entity.



Now if a user attempts to save a product without a name, they will be prompted with the message.



Add the following validation rules:

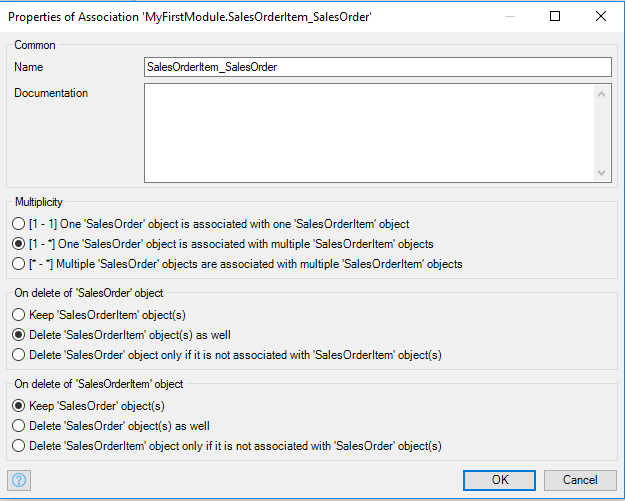
**Customer**

CustomerName Required  
CustomerAddress Required  
CustomerEmail Required

### Add Referential Integrity Rules

If a sales order is deleted, we must decide what should happen to the associated sales order items. The options are to leave them, to delete them or to not allow a sales order to be deleted if it has sales order items.

Double-click the association between SalesOrder and SalesOrderItem. In the On delete of ‘SalesOrder’ object section choose to Delete ‘SalesOrderItem object(s) as well.



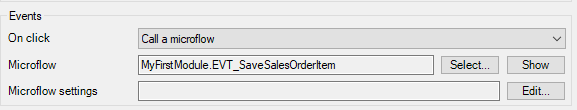
For the Product\_SalesOrderItem association, make sure that a product cannot be deleted if an associated sales order item exists.

For the SalesOrder\_Customer association, make sure that a customer cannot be deleted if an associated sales order exists.

## Add Data Validation to a Save Microflow

Another way to handle data validation is to add it to a microflow. The difference between this method and adding a validation rule to the entity is that they validation rule on the entity is always applied whereas the microflow can be more targeted. In this example, we’ll create a microflow that will ensure that the quantity field on a sales order item is greater than 0.

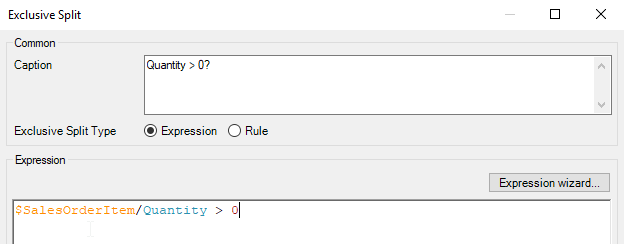
Open the SalesOrderItem\_NewEdit page. Open the properties of the Save button and configure the Events section as shown below. Create the EVT\_SaveSalesOrderItem microflow in the Microflows folder.



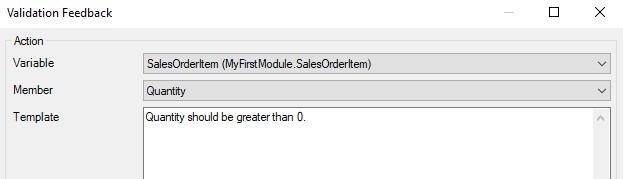
Open the microflow. Note the SalesOrderItem parameter is equal to the object being edited on the form.



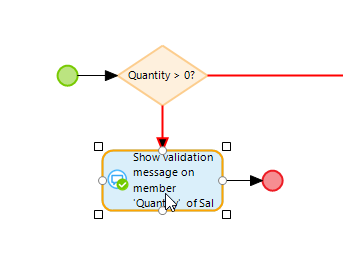
Add an Exclusive split to the right of the Start event and configure it as shown.



Drag down from the bottom point of the Exclusive split and add an activity. Configure the type as Validation feedback and configure it as shown.

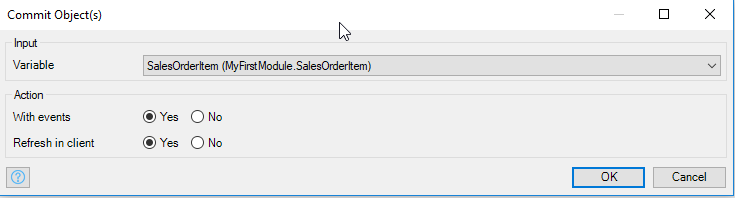


Drag out from the edge of the Validation feedback activity and add an End event.

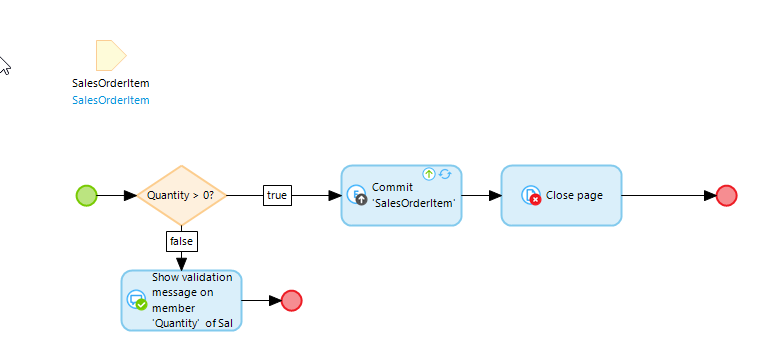


Right-click the line between the Exclusive split and the Validation feedback activity and select the false condition.

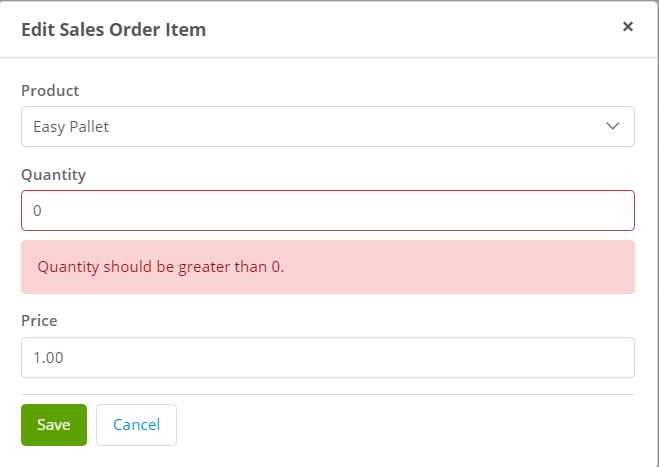
Add an activity on the True branch of the Exclusive split and make it a Commit activity and configure it to commit the SalesOrderItem object. Make sure you select Refresh in client.



Finally, add an activity to close the page.



The result looks the same as the validation rule.

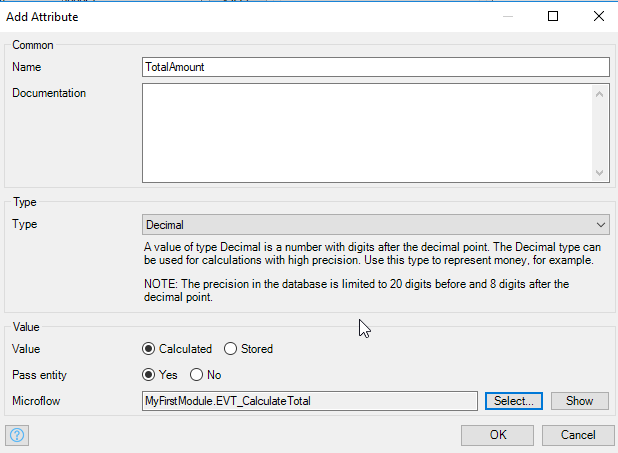


## Add a Total to the SalesOrder Entity

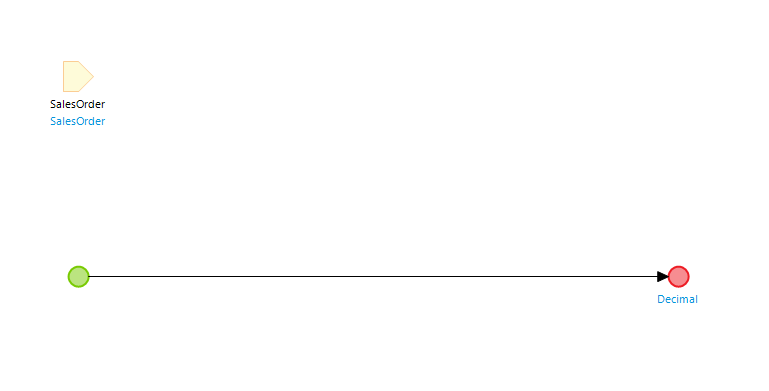
Next we’ll add a TotalAmount attribute to the SalesOrder entity that is equal to the sum of the Value attribute of the sales order items. Calculating this value will require a microflow.

### Create the Attribute

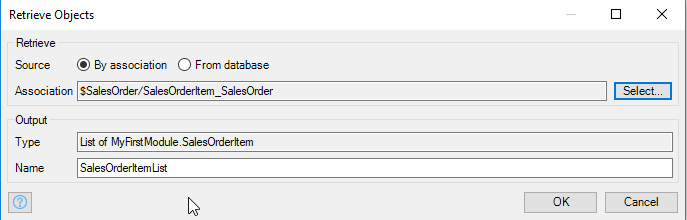
Create the attribute in the SalesOrder entity and configure it as shown below. Create the EVT\_CalculateTotal microflow in the Microflows folder.



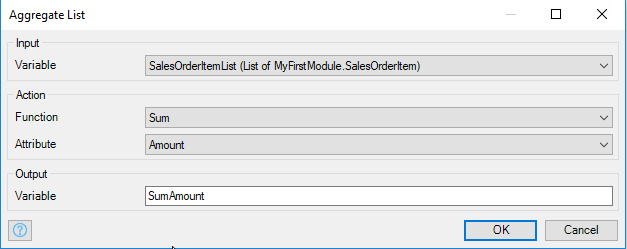
Open the microflow.



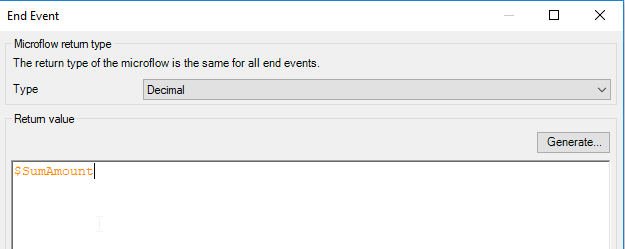
The first thing we must do is to retrieve the sales order items associated with the SalesOrder. Add a Retrieve activity and use the SalesOrder\_SalesOrderItem association to retrieve the items.



Now add an Aggregate list activity and sum the Amount attribute of the items on the SalesOrderItemList.

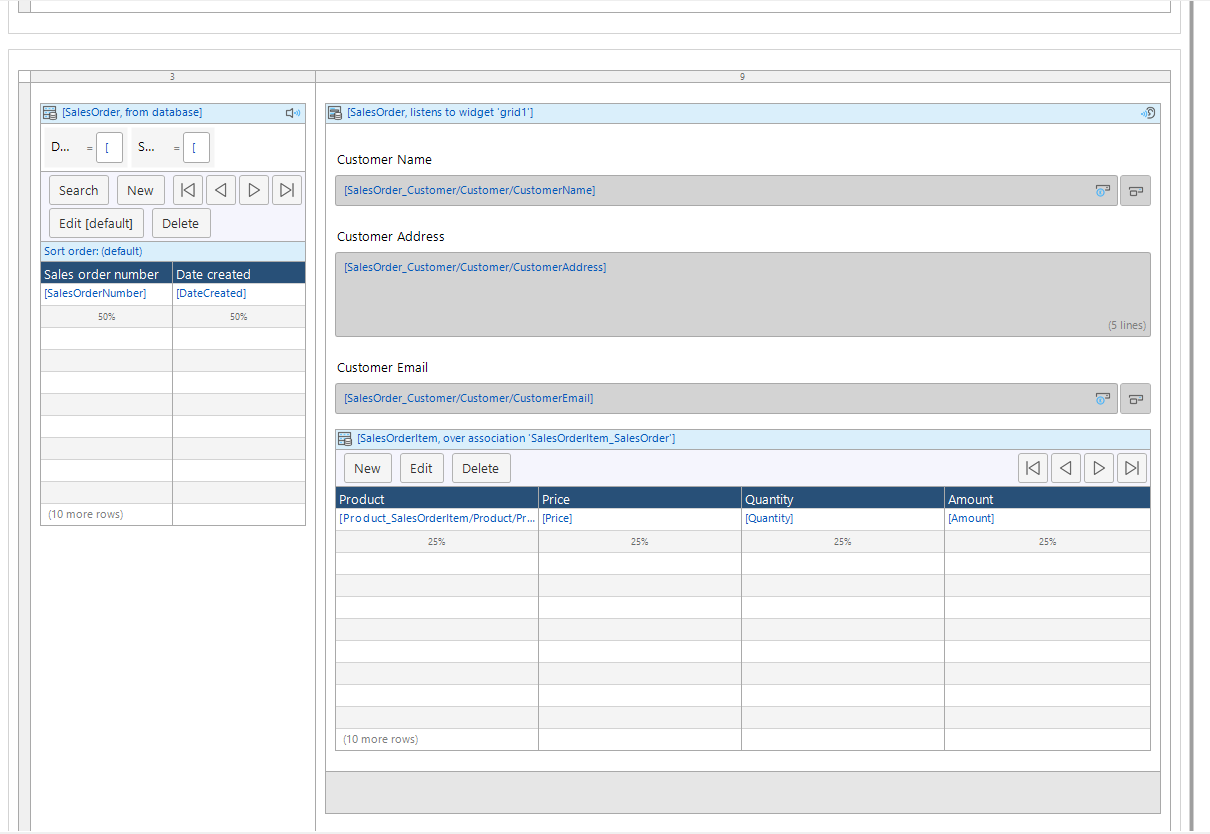


Finally, configure the End event to return the SumAmount variable.

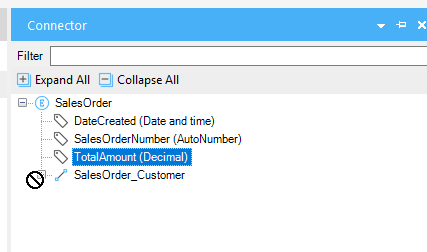


### Add the Attribute to the Interface

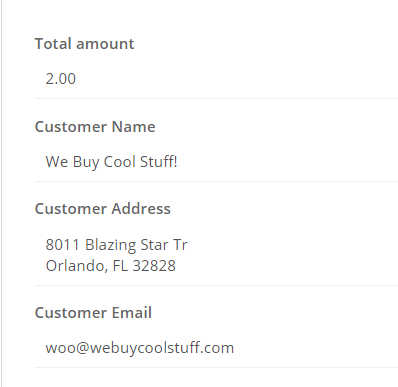
Open the SalesOrderOverview page.



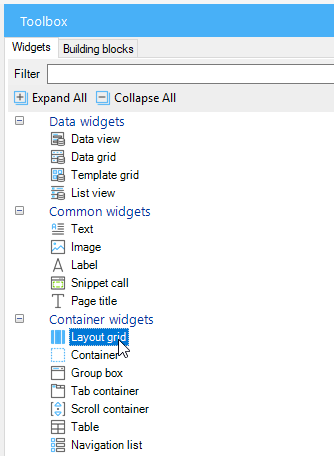
Click in the Data view on the detail side of the page and open the Connector tab. Drag TotalAmount to above the Customer widgets.



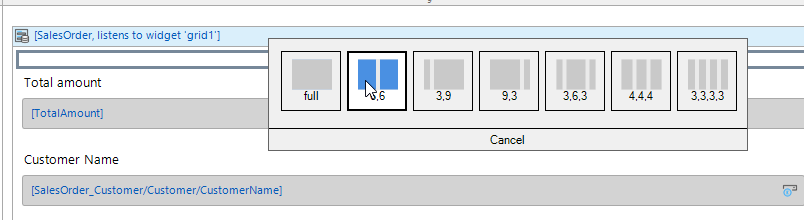
The total now shows but we can rearrange this a bit to make it look a little better.



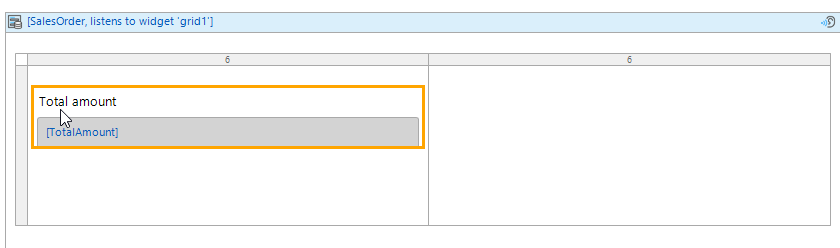
Drag a Layout grid from the Toolbox…



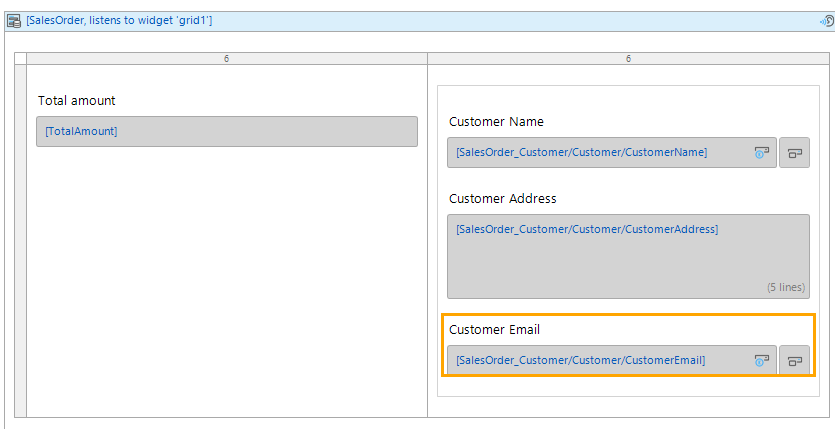
…to just above the TotalAmount widget. Select the two even columns layout.



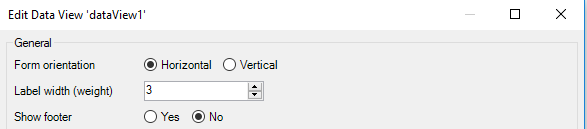
Drag the TotalAmount widget into the left column.



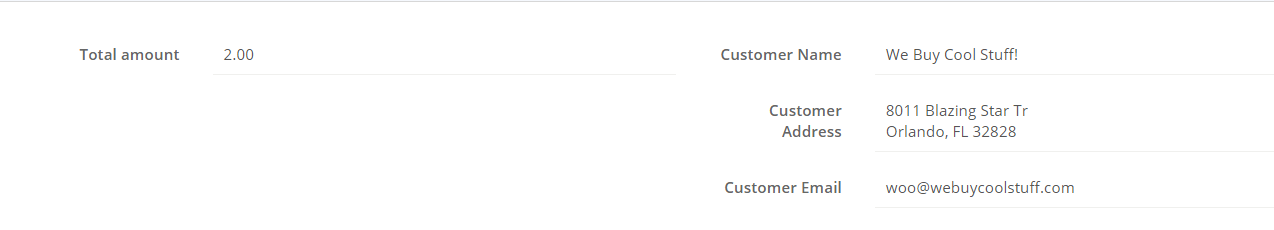
Drag the customer widgets into the right column.



Double-click the header of the Data view and change the Form orientation to Horizontal.



That’s a little better.

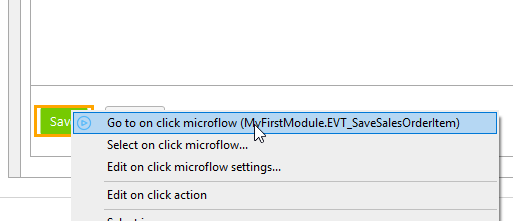


## Check QuantityInStock

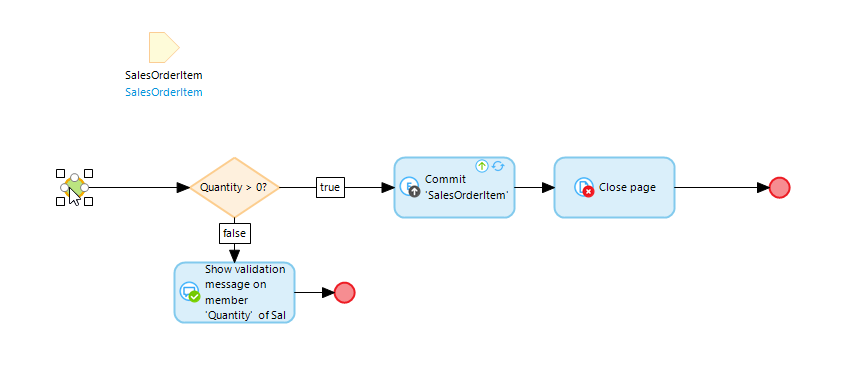
When a user enters a sales order item, we should check to make sure there is sufficient quantity of the product in stock. Since we already have a microflow for the save sales order item operation, we can add the check there.

### Rearrange the Form

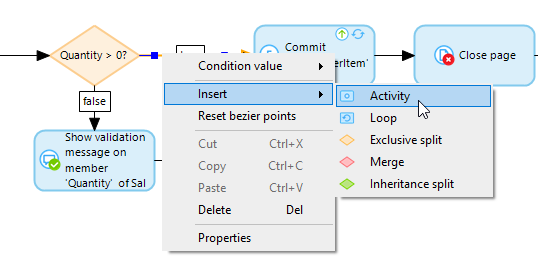
Open the SalesOrderItem\_NewEdit page and right-click the Save button and select Go to on click microflow.



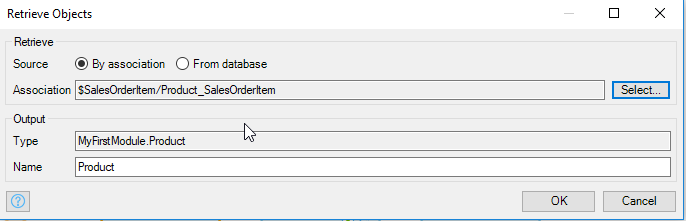
This is the microflow so far:



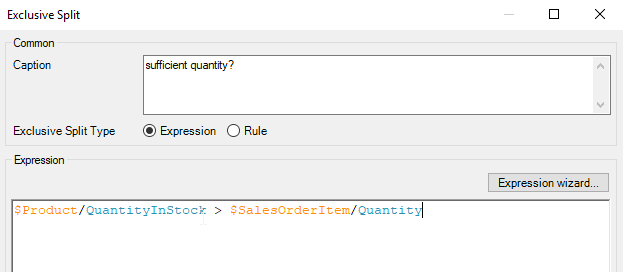
The first thing we must do is to retrieve the product associated with the sales order item. Let’s add a Retrieve activity to the right of the Exclusive split.



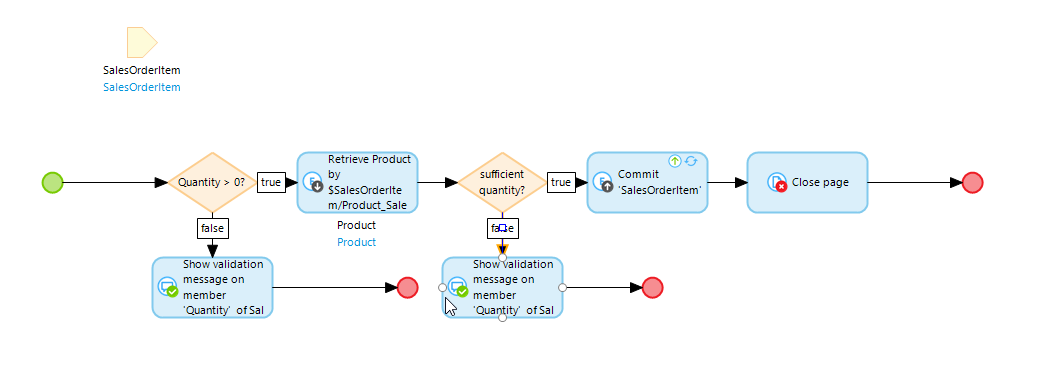
Configure it as shown:

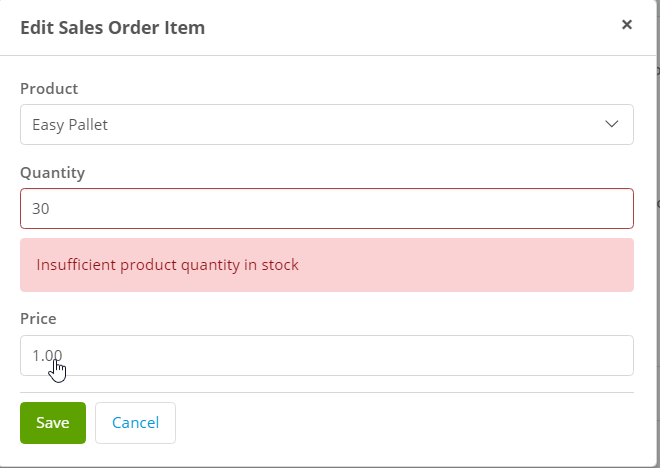


Next, add an Exclusive split to make sure the sales order item quantity is less than the product quantity in stock.



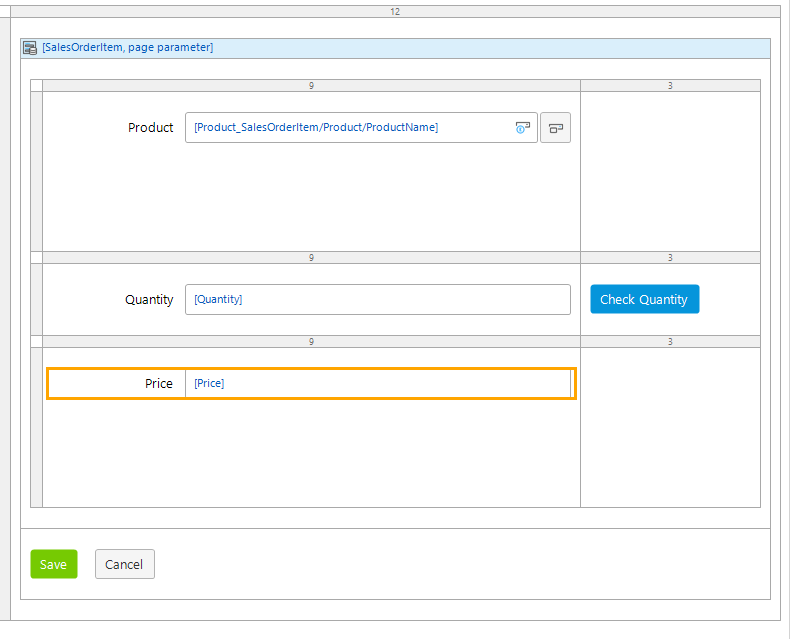
Finally, configure the false branch to use a Validation Feedback activity to let the user know the error. The final microflow looks like this:



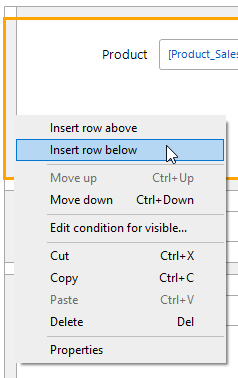


Another nice option would be to add a button the SalesOrderItem\_NewEdit form to allow the user to check the quantity of the product.

Open the SalesOrderItem\_NewEdit page. The final configuration will look like this.



Drag a Layout grid inside the Data view and choose the 9,3 column options. Right-click the outline of the grid and select Insert row below twice so that you have three rows.



Drag the three widgets into the left column cells.

Drag a Call a microflow button into the cell to the right of the quantity field. Configure it to call a microflow called EVT\_CheckQuantityInStock.

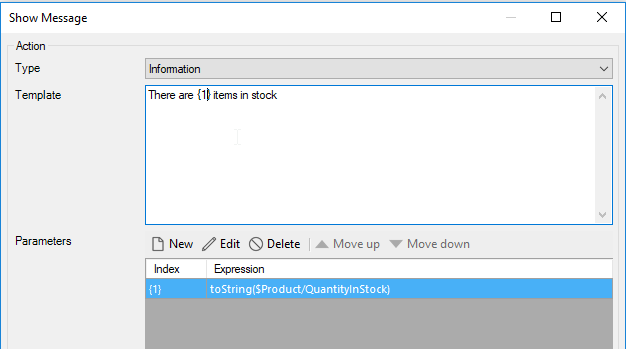
Finally, change the Form orientation of the Data view to Horizontal.

### Create the Microflow

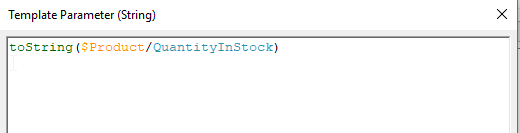
Open the microflow.



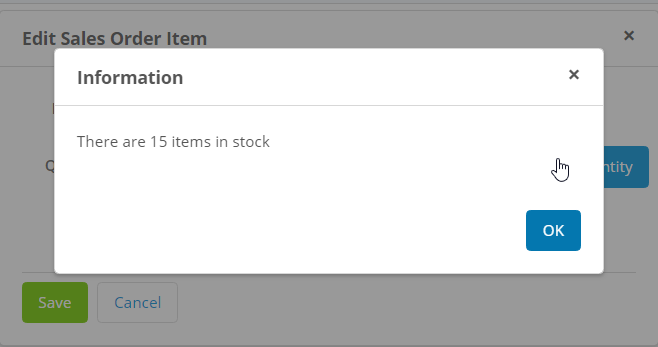
We must retrieve the product associated with the sales order item and display the quantity to the user. Add a Retrieve activity to retrieve the product. Next add a Show message activity and configure it as shown:



The {1} in the Template represents a parameter that will be replaced by the value created in the Parameters table. The formula for the parameter is shown below. Since QuantityInStock is an integer, we have to convert it to a string using the toString() function.



Now the user can check the quantity.

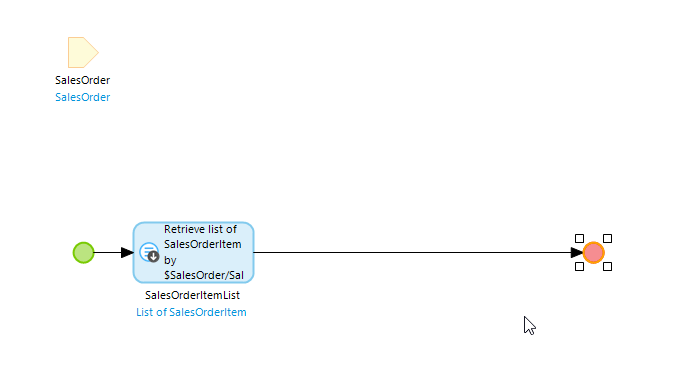


As an exercise, add the product name to the message.

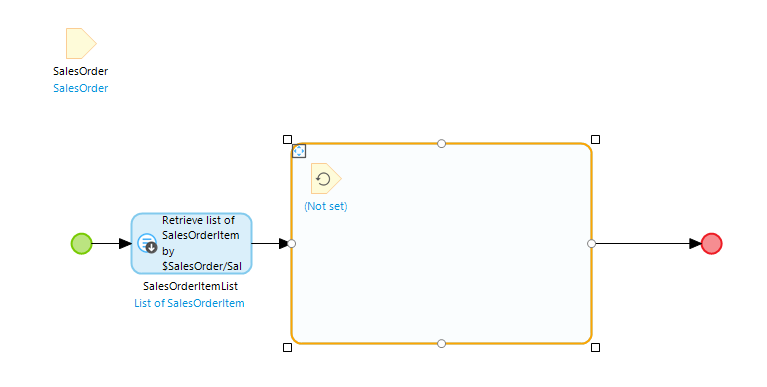
## Update QuantityInStock When a Sales Order is Created

The final microflow we will create is one that will update the product quantity in stock when we save a sales order. This will require us to attach a microflow to the save button of the SalesOrder\_NewEdit page as we did previously with the SalesOrderItem\_NewEdit page.

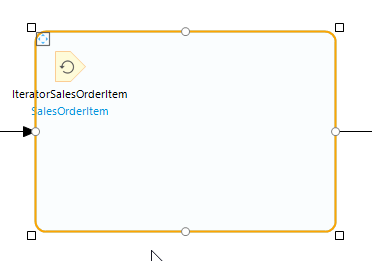
Configure the Save button on the SalesOrder\_NewEdit page to call a microflow called EVT\_SaveSalesOrder. The first activity should retrieve the sales order items associated with the sales order.



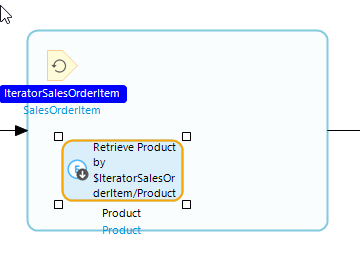
Next, insert a loop.



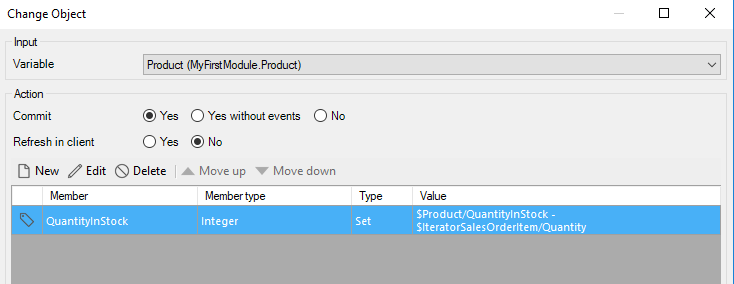
This loop will iterate over the sales order items, retrieve the associated product, update its quantity in stock and save it. Set the iterator (top left inside the loop that has Not set below it) equal to the SalesOrderItemList list.



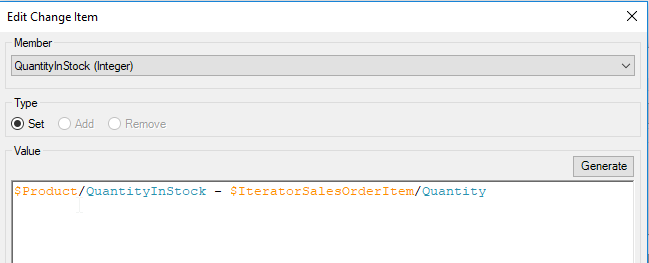
The IteratorSalesOrderItem variable will contain the current SalesOrderItem object. Insert a Retrieve activity to retrieve the product associated with IteratorSalesOrderItem.



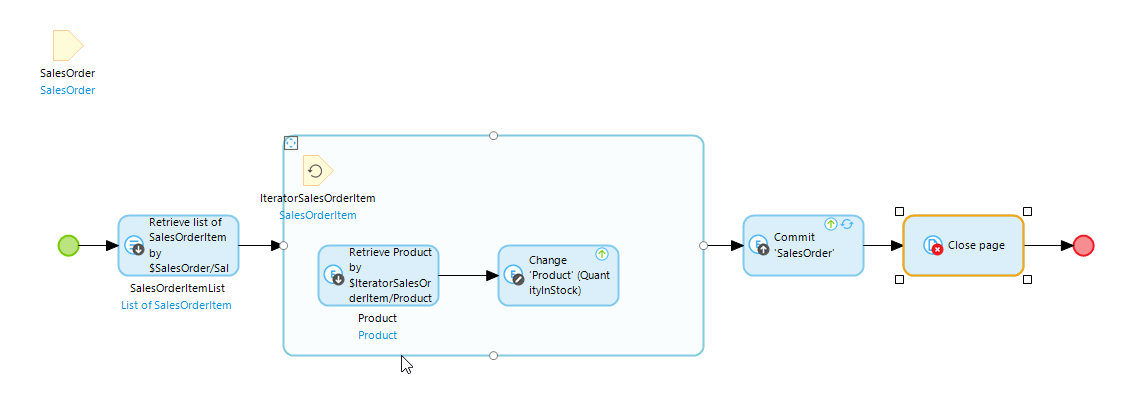
Drag out from the edge of the Retrieve activity and create a new activity (you will have to adjust the size and position of the loop boundary to make it fit). Configure the activity as a Change object type as shown below. Make sure you select Yes for Commit.



The formula to update QuantityInStock is shown below.



After the loop, commit the SalesOrder and close the page.

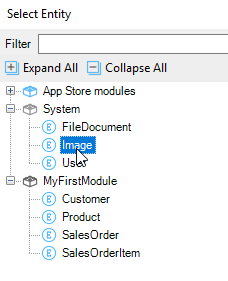


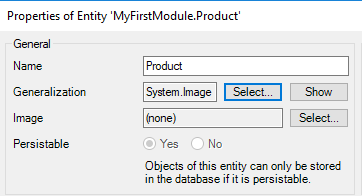
## Add a Product Image

The final change to the app is to add an image for the product.

### Update the Entity

Open the properties of the Product entity. Use the Select button next to Generalization to select the System Image.

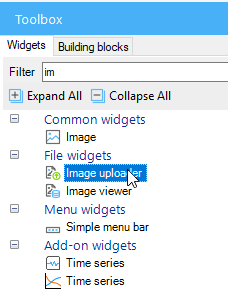


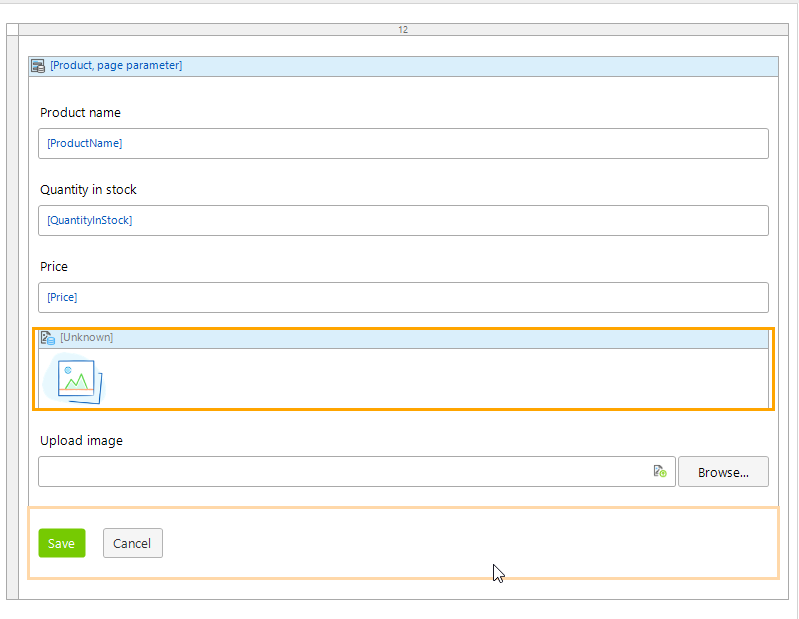


This adds several attributes in the background required to handle images.

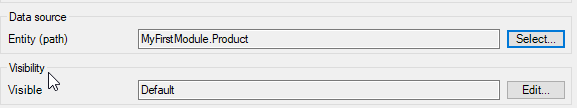
### Update the Product\_NewEdit Page

Drag an Image viewer and Image uploader onto the page below the price widget.

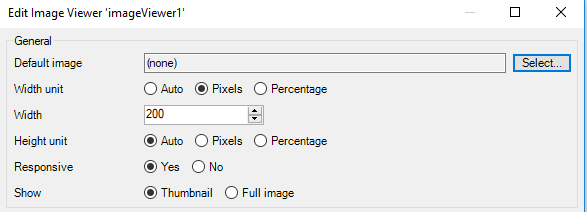




Set the data source for the Image viewer to the Product entity.



Configure the size of the image to something that will fit nicely in the form.



Now you can add images. Note that the image won’t show up when you first upload it, only when you edit the product after uploading an image.

