8.3.0 – Purchasing & Receiving

Introduction

You were shown in the previous lesson, how to create a complex transaction. There are two other types of transactions: Purchasing (ordering products) and Receiving (adding products to inventory).

Supporting Files

In the **Code Files (8.3.0)** folder on Moodle, you will see the following files, which are needed for this lesson:

- eStoreSystem_BLL_Controllers2.zip: contains the new PuchaseOrderController.cs and the new ReceiveOrderController.cs
- PurchaseOrder_aspx.txt: Content block of the web page for PurchaseOrder.aspx
- PurchaseOrder_aspx_cs.txt: the starting code to be added to PurchaseOrder.aspx.cs
- ReceiveOrder_aspx.txt: Content block of the web page ReceiveOrder.aspx
- ReceiveOrder_aspx_cs.txt: the starting code to be added to ReceiveOrder.aspx.cs

Purchasing

Purchasing, or ordering products, uses the following steps:

1. Employee logs in
2. Select a supplier (or a vendor; some company you order products from)
3. Get a list of suggested orders for the supplier
4. Edit the suggested order quantity (optional)
5. Add additional product items to the order (optional)
6. Submit the order → creates a new Purchase Order

BLL – PurchaseOrderController

In this class file, there is only one method, **CreatePurchaseOrder**. The code we will put in this method is like the code in **CreateSale** in the **SalesOrderController**. The code is:

Listing 1: CreatePurchsaeOrder

```
public int CreatePurchaseOrder(int employeeID, int supplierID, List<ProductForOrder>
products)
{
    int purchaseOrderNumber = 0;
    PurchaseOrder order = null;
    // Setup transaction area
    using (var context = new eStoreContext())
    {
        // Create a new PurchaseOrder
        order = new PurchaseOrder();
        order.PO_Date = DateTime.Now;
        order.EmployeeID = employeeID;
        order.SupplierID = supplierID;
        order.CompletedDate = null;
        order = context.PurchaseOrder.Add(order);
        purchaseOrderNumber = context.PurchaseOrder.Count() + 1000; // Purchase Orders
start at 1000
        // AddToOrder
        foreach(ProductForOrder item in products)
        {
            PurchaseOrderDetail detail = new PurchaseOrderDetail();
            detail.PurchaseOrderNumber = purchaseOrderNumber;
            detail.ProductID = item.ID;
            detail.OrderQuantity = item.Ordered;
            // find the Product to update (OnOrder)
            Product product = context.Product.Find(item.ID);
            // update the OnOrder
            product.OnOrder += item.Ordered;
            // tell the context the Product has been modified
            context.Entry(product).State = System.Data.Entity.EntityState.Modified;
            order.PurchaseOrderDetails.Add(detail);
        }// end for
        context.SaveChanges();
    }//end using
    return purchaseOrderNumber;
}//eom
```

BLL - ProductController

We need to add one method to this class; the method will return the suggested order quantity (SOQ) for a given supplier. Using the code form the SOQ.linq file as a template to get the SOQ, the code for this method is:

Listing 2: SuggestedOrderQuantity

```
[DataObjectMethod(DataObjectMethodType.Select, false)]
public List<SOQ> SuggestedOrderQuantity(int supplierID)
{
    using (var context = new eStoreContext())
    {
        var results = from x in context.Product
                      where x.SupplierID == supplierID
                      select new SOQ
                      {
                          ProductID = x.ProductID,
                          ProductName = x.ProductName,
                          ProductSKU = x.ProductSKU,
                          ProductDescription = x.ProductDescription,
                          OrderCost = x.OrderCost,
                          QuantityOnHand = x.QuantityOnHand,
                          ReOrderLevel = x.ReOrderLevel,
                          OnOrder = x.OnOrder,
                          OrderQuantity = (x.ReOrderLevel - x.QuantityOnHand - x.OnOrder)
+ (int)(x.ReOrderLevel * 0.2) < 0 ? 0 : (x.ReOrderLevel - x.QuantityOnHand - x.OnOrder) +
(int)(x.ReOrderLevel * 0.2) // Get SOQ
                      };
        return results.ToList();
    }//end using
}//eom
```

This uses an IIF to check the data

```
(x.ReOrderLevel - x.QuantityOnHand - x.OnOrder) + (int)(x.ReOrderLevel * 0.2)
```

The IIF (Immediate If) used above does the following:

- Checks if the Suggested quantity < 0
- If < 0 then set OrderQuantity to 0

- Otherwise, set OrderQuantity to the suggested quantity

Web Form PurchaseOrder.aspx

We created this starting form previously, but it needs to be modified to meet the requirements of creating a Purchase Order. Replace the Content section of the existing web form with the contents of the PurchaseOrder_aspx.txt file. Also, replace the contents of the PurchaseOrder.aspx.cs file with the contents of the PurchaseOrder_aspx_cs.txt file.

On the web form:

- Employee login is done the same way we did the Customer login on the ShoppingCart.aspx page
- Supplier list will be in a DropDownList control that uses an ODS control
- The ListView will display all the products for the selected supplier, and it will have a TextBox for the order quantity (the value comes from the SOQ calculation)
- Two buttons: Create and Cancel

The code for the Content block of the web form is:

Listing 3: PurchaseOrcer.aspx

```
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="Server">
    <div class="jumbotron">
        <h1>Create Purchase Order</h1>
    </div>
    <div class="row">
        <uc1:MessageUserControl runat="server" ID="MessageUserControl" />
    </div>
    <div class="row">
        <asp:Label ID="EmployeeLabel" runat="server" Text="Emploee:" />&nbsp;
        <asp:DropDownList ID="EmployeeListDDL" runat="server"
            DataSourceID="EmployeeListODS"
            DataTextField="DisplayText"
            DataValueField="IDValue"
            AppendDataBoundItems="true">
            <asp:ListItem Value="0">Select Employee</asp:ListItem>
        </asp:DropDownList>
        &nbsp;<asp:Button ID="LoginButton" runat="server" Class="btn btn-primary"
Text="Login" OnClick="LoginButton_Click"/><br /><br />
    </div>
    <div class="row">
        <asp:Label ID="SupplierLabel" runat="server" Text="Album:" />
        &nbsp;&nbsp;
        <asp:DropDownList ID="SupplierListDDL" runat="server"
            DataSourceID="SupplierListODS"
            DataTextField="DisplayText"
            DataValueField="IDValue"
            AppendDataBoundItems="true">
            <asp:ListItem Value="0">Select Supplier</asp:ListItem>
        </asp:DropDownList>
        &nbsp;&nbsp;
        <asp:Button ID="FetchButton" runat="server" Class="btn btn-primary" Text="Fetch"
OnClick="FetchButton_Click" Enabled="false"/>
    </div>
    <div class="row">
```

```
        <asp:ListView ID="SOQList_LV" runat="server">
           <EmptyDataTemplate>
                <table runat="server" style="">
                    <tr>
                        <td>No data was returned.</td>
                    </tr>
                </table>
            </EmptyDataTemplate>
            <ItemTemplate>
                <tr style="">
                    <td>
                        <asp:Label Text='<%# Eval("ProductID") %>' runat="server"
ID="ProductIDLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("ProductName") %>' runat="server"
ID="ProductNameLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("ProductSKU") %>' runat="server"
ID="ProductSKULabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("ProductDescription") %>'
runat="server" ID="ProductDescriptionLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("OrderCost") %>' runat="server"
ID="OrderCostLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("QuantityOnHand") %>' runat="server"
ID="QuantityOnHandLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("ReOrderLevel") %>' runat="server"
ID="ReOrderLevelLabel" /></td>
                    <td>
                        <asp:Label Text='<%# Eval("OnOrder") %>' runat="server"
ID="OnOrderLabel" /></td>
                    <td>
                        <asp:TextBox Text='<%# Eval("OrderQuantity") %>' runat="server"
ID="OrderQuantityTextBox" Width="50"/>&nbsp;&nbsp;</td>
                </tr>
            </ItemTemplate>
            <LayoutTemplate>
                <table runat="server">
                    <tr runat="server">
                        <td runat="server">
                            <table runat="server" id="itemPlaceholderContainer" style=""
border="0">
                                <tr runat="server" style="">
                                    <th runat="server">ID&nbsp;&nbsp;</th>
                                    <th runat="server">Name&nbsp;&nbsp;</th>
                                    <th runat="server">SKU&nbsp;&nbsp;</th>
                                    <th runat="server">Description&nbsp;&nbsp;</th>
                                    <th runat="server">Cost&nbsp;&nbsp;</th>
                                    <th runat="server">QOH&nbsp;&nbsp;</th>
                                    <th runat="server">ROL&nbsp;&nbsp;</th>
                                    <th runat="server">On Order&nbsp;&nbsp;</th>
                                    <th runat="server">SOQ&nbsp;&nbsp;</th>
                                </tr>
                                <tr runat="server" id="itemPlaceholder"></tr>
                            </table>
                        </td>
                    </tr>
                    <tr runat="server">
                        <td runat="server" style=""></td>
                    </tr>
                </table>
            </LayoutTemplate>
        </asp:ListView>
        <br /><br />
        <asp:Button ID="CreateButton" runat="server" Class="btn btn-primary"
Text="Create" OnClick="CreateButton_Click"  Enabled="false" />&nbsp;&nbsp;
        <asp:Button ID="CancelButton" runat="server" Class="btn btn-primary"
Text="Cancel" OnClick="CancelButton_Click" Enabled="false" />
        <br /><br />
    </div>
    <asp:ObjectDataSource ID="EmployeeListODS" runat="server"
        OldValuesParameterFormatString="original_{0}"
        SelectMethod="GetOrderEmployees"
        TypeName="eStoreSystem.BLL.EmployeeController">
    </asp:ObjectDataSource>
    <asp:ObjectDataSource ID="SupplierListODS" runat="server"
        OldValuesParameterFormatString="original_{0}"
        SelectMethod="GetAllSuppliers"
        TypeName="eStoreSystem.BLL.SupplierController">
    </asp:ObjectDataSource>
</asp:Content>
```

On the code behind file, PurchaseOrder.aspx.cs, the following methods were added:

Listing 4: PurchaseOrder.aspx.cs Added Methods

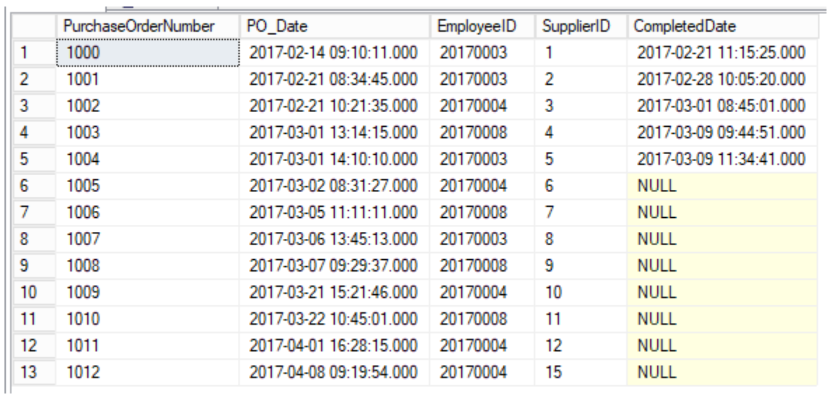
```
// Simulate Login of Employees who can create a Purchase Order
protected void LoginButton_Click(object sender, EventArgs e)
{
    int employeeID = int.Parse(EmployeeListDDL.SelectedValue);
    if (employeeID > 0)
    {
        FetchButton.Enabled = true;
        EmployeeListDDL.Enabled = false;
    }
    else
    {
        MessageUserControl.ShowInfo("LOGIN ERROR", "NO EMPLOYEE SELECTED!");
    }
}//eom

// After Login, the click of this button loads the ListView and
//     turns on the Create and Cancel buttons
protected void FetchButton_Click(object sender, EventArgs e)
{
    int supplierID = int.Parse(SupplierListDDL.SelectedValue);
    if(supplierID > 0)
    {
        ProductController controller = new ProductController();
        List<SOQ> products = controller.SuggestedOrderQuantity(supplierID);
        SOQList_LV.DataSource = products;
        SOQList_LV.DataBind();
        CreateButton.Enabled = true;
        CancelButton.Enabled = true;
    }//end if
}//eom

// Create a new Purchase Order
protected void CreateButton_Click(object sender, EventArgs e)
{
    // As there could be errors, use the TryRun()
    MessageUserControl.TryRun(() =>
    {
        // Get the values from the 2 DDL controls
        int employeeID = int.Parse(EmployeeListDDL.SelectedValue);
        int supplierID = int.Parse(SupplierListDDL.SelectedValue);
        // Create an empty List<>
        List<ProductForOrder> products = new List<ProductForOrder>();
        // Loop through the items on the List view
        foreach(ListViewDataItem lvRow in SOQList_LV.Items)
        {
            ProductForOrder product = new ProductForOrder();
            product.ID = int.Parse((lvRow.FindControl("ProductIDLabel") as Label).Text);
            product.Name = (lvRow.FindControl("ProductNameLabel") as Label).Text;
            product.SKU = (lvRow.FindControl("ProductSKULabel") as Label).Text;
            product.Cost = decimal.Parse((lvRow.FindControl("OrderCostLabel") as
Label).Text);
            product.ROL = int.Parse((lvRow.FindControl("ReOrderLevelLabel") as
Label).Text);
            product.QOH = int.Parse((lvRow.FindControl("QuantityOnHandLabel") as
Label).Text);
            product.Ordered = int.Parse((lvRow.FindControl("OrderQuantityTextBox") as
TextBox).Text);
            // if the TextBox does not have a 0, then add to the List<>
            if (product.Ordered > 0)
            {
                products.Add(product);
            }//end if
        }//end for
        // Call the CreatePurchaseOrder method of the PurchaseOrderController
        PurchaseOrderController controller = new PurchaseOrderController();
        int orderNumber = controller.CreatePurchaseOrder(employeeID, supplierID,
products);
        // Display the new PurchaseOrderNumber
        MessageUserControl.ShowInfo("Purchase Order Created", "Purchase Order Number = "
+ orderNumber);
        CancelButton_Click(sender, e);
    });
}//eom

// Reset the form to its default
protected void CancelButton_Click(object sender, EventArgs e)
{
    EmployeeListDDL.Enabled = true;
    EmployeeListDDL.SelectedValue = "0";
    SupplierListDDL.SelectedValue = "0";
    LoginButton.Enabled = true;
    FetchButton.Enabled = false;
    CreateButton.Enabled = false;
    CancelButton.Enabled = false;
    SOQList_LV.DataSource = null;
    SOQList_LV.DataBind();
```

```
}//eom
```

Test

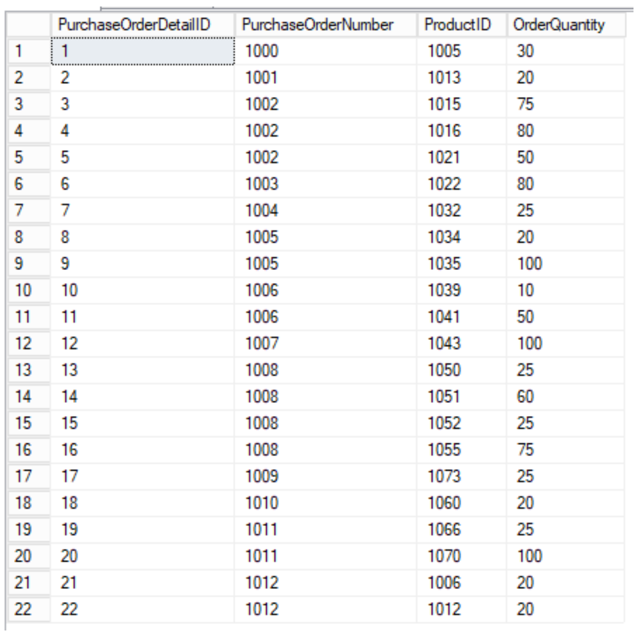
Once again, to properly test this code, use a fresh copy of the database.

Original Database Data



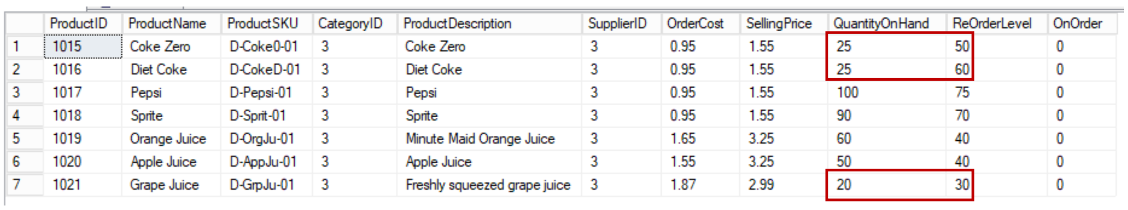
	PurchaseOrderNumber	PO_Date	EmployeeID	SupplierID	CompletedDate
1	1000	2017-02-14 09:10:11.000	20170003	1	2017-02-21 11:15:25.000
2	1001	2017-02-21 08:34:45.000	20170003	2	2017-02-28 10:05:20.000
3	1002	2017-02-21 10:21:35.000	20170004	3	2017-03-01 08:45:01.000
4	1003	2017-03-01 13:14:15.000	20170008	4	2017-03-09 09:44:51.000
5	1004	2017-03-01 14:10:10.000	20170003	5	2017-03-09 11:34:41.000
6	1005	2017-03-02 08:31:27.000	20170004	6	NULL
7	1006	2017-03-05 11:11:11.000	20170008	7	NULL
8	1007	2017-03-06 13:45:13.000	20170003	8	NULL
9	1008	2017-03-07 09:29:37.000	20170008	9	NULL
10	1009	2017-03-21 15:21:46.000	20170004	10	NULL
11	1010	2017-03-22 10:45:01.000	20170008	11	NULL
12	1011	2017-04-01 16:28:15.000	20170004	12	NULL
13	1012	2017-04-08 09:19:54.000	20170004	15	NULL

Figure 1: Data - PurchaseOrder



	PurchaseOrderDetailID	PurchaseOrderNumber	ProductID	OrderQuantity
1	1	1000	1005	30
2	2	1001	1013	20
3	3	1002	1015	75
4	4	1002	1016	80
5	5	1002	1021	50
6	6	1003	1022	80
7	7	1004	1032	25
8	8	1005	1034	20
9	9	1005	1035	100
10	10	1006	1039	10
11	11	1006	1041	50
12	12	1007	1043	100
13	13	1008	1050	25
14	14	1008	1051	60
15	15	1008	1052	25
16	16	1008	1055	75
17	17	1009	1073	25
18	18	1010	1060	20
19	19	1011	1066	25
20	20	1011	1070	100
21	21	1012	1006	20
22	22	1012	1012	20

Figure 2: Data - PurchaseOrderDetail



	ProductID	ProductName	ProductSKU	CategoryID	ProductDescription	SupplierID	OrderCost	SellingPrice	QuantityOnHand	ReOrderLevel	OnOrder
1	1015	Coke Zero	D-Coke0-01	3	Coke Zero	3	0.95	1.55	25	50	0
2	1016	Diet Coke	D-CokeD-01	3	Diet Coke	3	0.95	1.55	25	60	0
3	1017	Pepsi	D-Pepsi-01	3	Pepsi	3	0.95	1.55	100	75	0
4	1018	Sprite	D-Sprit-01	3	Sprite	3	0.95	1.55	90	70	0
5	1019	Orange Juice	D-OrgJu-01	3	Minute Maid Orange Juice	3	1.65	3.25	60	40	0
6	1020	Apple Juice	D-AppJu-01	3	Apple Juice	3	1.55	3.25	50	40	0
7	1021	Grape Juice	D-GrpJu-01	3	Freshly squeezed grape juice	3	1.87	2.99	20	30	0

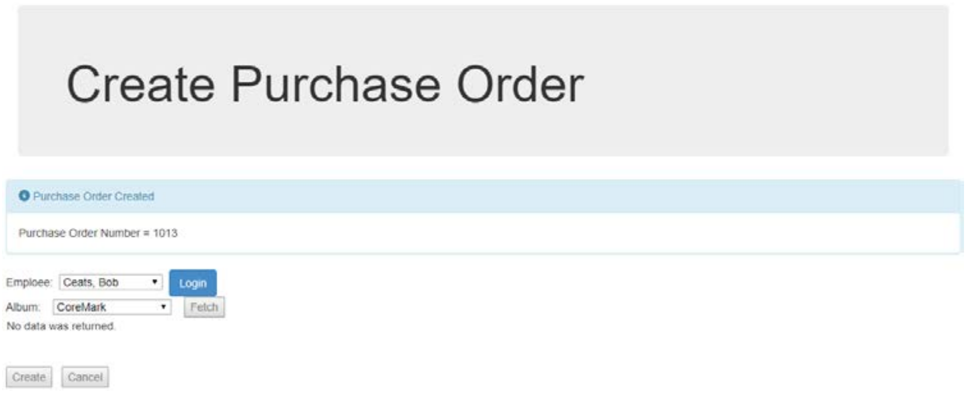
Figure 3: Data - Products for Supplier (ID=3) CoreMark

Test Scenario

The test scenario is:

- Use any Employee listed in the DropDownList, and Login
- Select **CoreMark** from the list of suppliers
- In Figure 3, the products that need ordering are highlighted with a and we will order all the required Products, thus press the **Create** button

After the running the steps above, the output in the browser should be:



Create Purchase Order

Purchase Order Created

Purchase Order Number = 1013

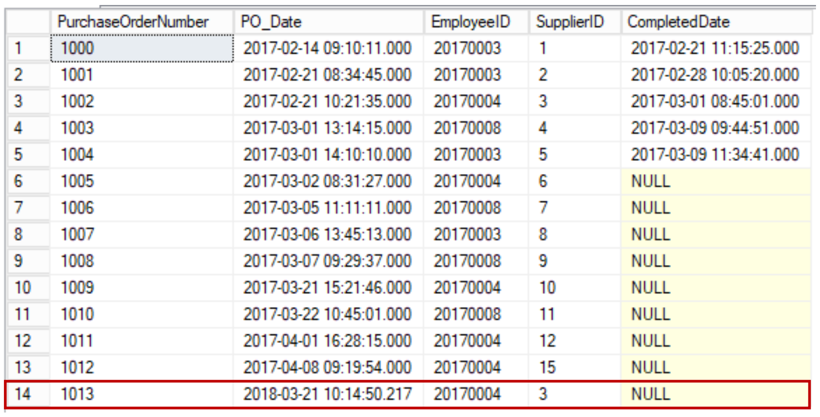
Emploee: Ceats, Bob

Album: CoreMark

No data was returned.

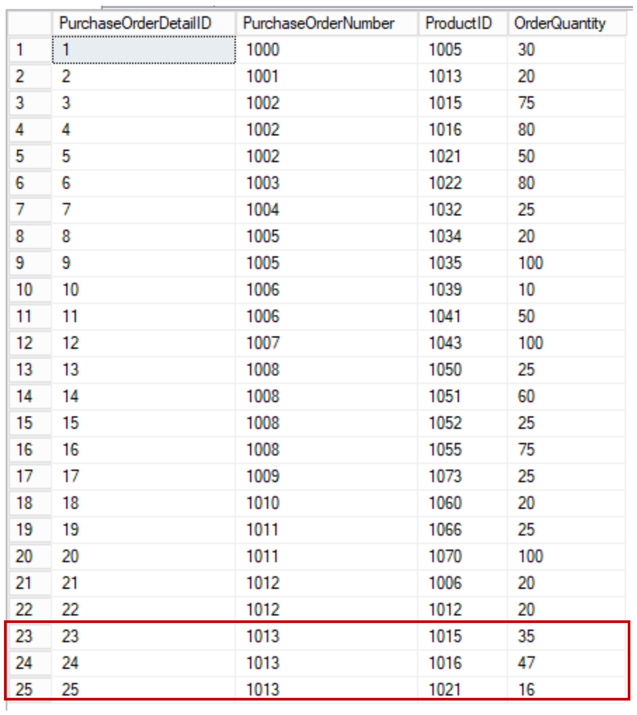
Figure 4: Test Scenario Browser Output

Data After Test



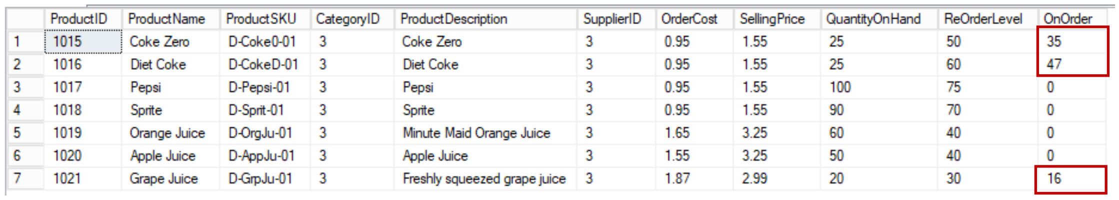
	PurchaseOrderNumber	PO_Date	EmployeeID	SupplierID	CompletedDate
1	1000	2017-02-14 09:10:11.000	20170003	1	2017-02-21 11:15:25.000
2	1001	2017-02-21 08:34:45.000	20170003	2	2017-02-28 10:05:20.000
3	1002	2017-02-21 10:21:35.000	20170004	3	2017-03-01 08:45:01.000
4	1003	2017-03-01 13:14:15.000	20170008	4	2017-03-09 09:44:51.000
5	1004	2017-03-01 14:10:10.000	20170003	5	2017-03-09 11:34:41.000
6	1005	2017-03-02 08:31:27.000	20170004	6	NULL
7	1006	2017-03-05 11:11:11.000	20170008	7	NULL
8	1007	2017-03-06 13:45:13.000	20170003	8	NULL
9	1008	2017-03-07 09:29:37.000	20170008	9	NULL
10	1009	2017-03-21 15:21:46.000	20170004	10	NULL
11	1010	2017-03-22 10:45:01.000	20170008	11	NULL
12	1011	2017-04-01 16:28:15.000	20170004	12	NULL
13	1012	2017-04-08 09:19:54.000	20170004	15	NULL
14	1013	2018-03-21 10:14:50.217	20170004	3	NULL

Figure 5: Data After Test - Purchase Order



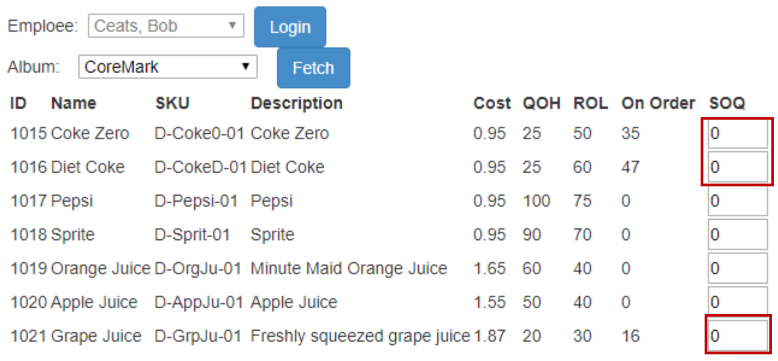
	PurchaseOrderDetailID	PurchaseOrderNumber	ProductID	OrderQuantity
1	1	1000	1005	30
2	2	1001	1013	20
3	3	1002	1015	75
4	4	1002	1016	80
5	5	1002	1021	50
6	6	1003	1022	80
7	7	1004	1032	25
8	8	1005	1034	20
9	9	1005	1035	100
10	10	1006	1039	10
11	11	1006	1041	50
12	12	1007	1043	100
13	13	1008	1050	25
14	14	1008	1051	60
15	15	1008	1052	25
16	16	1008	1055	75
17	17	1009	1073	25
18	18	1010	1060	20
19	19	1011	1066	25
20	20	1011	1070	100
21	21	1012	1006	20
22	22	1012	1012	20
23	23	1013	1015	35
24	24	1013	1016	47
25	25	1013	1021	16

Figure 6: Data After Test - PurchaseOrderDetail



	ProductID	ProductName	ProductSKU	CategoryID	ProductDescription	SupplierID	OrderCost	SellingPrice	QuantityOnHand	ReOrderLevel	OnOrder
1	1015	Coke Zero	D-Coke0-01	3	Coke Zero	3	0.95	1.55	25	50	35
2	1016	Diet Coke	D-CokeD-01	3	Diet Coke	3	0.95	1.55	25	60	47
3	1017	Pepsi	D-Pepsi-01	3	Pepsi	3	0.95	1.55	100	75	0
4	1018	Sprite	D-Sprit-01	3	Sprite	3	0.95	1.55	90	70	0
5	1019	Orange Juice	D-OrgJu-01	3	Minute Maid Orange Juice	3	1.65	3.25	60	40	0
6	1020	Apple Juice	D-AppJu-01	3	Apple Juice	3	1.55	3.25	50	40	0
7	1021	Grape Juice	D-GrpJu-01	3	Freshly squeezed grape juice	3	1.87	2.99	20	30	16

Figure 7: Data After Test - Products for Supplier (ID=3) CoreMark



Emploee: Ceats, Bob Login

Album: CoreMark Fetch

ID	Name	SKU	Description	Cost	QOH	ROL	On Order	SOQ
1015	Coke Zero	D-Coke0-01	Coke Zero	0.95	25	50	35	0
1016	Diet Coke	D-CokeD-01	Diet Coke	0.95	25	60	47	0
1017	Pepsi	D-Pepsi-01	Pepsi	0.95	100	75	0	0
1018	Sprite	D-Sprit-01	Sprite	0.95	90	70	0	0
1019	Orange Juice	D-OrgJu-01	Minute Maid Orange Juice	1.65	60	40	0	0
1020	Apple Juice	D-AppJu-01	Apple Juice	1.55	50	40	0	0
1021	Grape Juice	D-GrpJu-01	Freshly squeezed grape juice	1.87	20	30	16	0

Figure 8: Browser View After Creating Purchase Order

Receiving

Receiving, or adding products to the inventory, uses the following steps:

1. Employee logs in (same login as in Purchasing)
2. Selects a ReceiveOrder from the list (the Receive Order will have a Completed Date of null)

3. A list of Items on the Receive Order items is displayed, which includes the OnOrder of the product.
4. Receive items into inventory:
    a. Increase quantity on hand by received quantity
    b. Decrease on order value by received quantity
    c. If all the items from the original purchase order are received, close the purchase order
    d. Close the Receive order

BLL - ReceiveOrderController

GetOpenReceiveOrders

This method returns only the ReceiveOrders where the ReceivedDate is null. The code is:

Listing 5: GetOpenReceiveOrders

```
public List<SelectionList> GetOpenReceiveOrders()
{
    // Setup transaction area
    using (var context = new eStoreContext())
    {
        var results = from x in context.ReceiveOrder
                      where x.ReceivedDate == null
                      select new SelectionList
                      {
                          IDValue = x.PurchaseOrderNumber,
                          DisplayText = x.ReceiveOrderNumber.ToString()
                      };
        return results.ToList(); // change this once the method is coded
    }//end using
}//eom
```

Notice that the `SelectionList` uses the PurchaseOrderNumber as the value field; we need to display the ReceiveOrderNumber.

GetOrderDetails

This method returns the details, ReceiveOrderDetail table information, related to the ReceiveOrder. It uses a POCO class to make the display user friendly. The code is:

Listing 6: GetOrderDetails

```
public List<ReceivedProducts> GetOrderDetails(int receiveOrderNumber)
{
    // Setup transaction area
    using (var context = new eStoreContext())
    {
        var results = from x in context.ReceiveOrderDetail
                      where x.ReceiveOrderNumber == receiveOrderNumber
                      select new ReceivedProducts
                      {
                          ProductID = x.PurchaseOrderDetail.ProductID,
                          SKU = x.PurchaseOrderDetail.Product.ProductSKU,
                          Name = x.PurchaseOrderDetail.Product.ProductName,
                          ReceivedQty = x.QuantityReceived,
                          OrderedQty = x.PurchaseOrderDetail.Product.OnOrder
                      };
        return results.ToList();
    }//end using
}//eom
```

The POCO class used in this method is:

Listing 7: ReceivedProducts

```
namespace eStoreSystem.Data.POCOs
{
    public class ReceivedProducts
    {
        public int ProductID { get; set; }
        public string SKU { get; set; }
        public string Name { get; set; }
        public int ReceivedQty { get; set; }
        public int OrderedQty { get; set; }
    }//eoc
}//eon
```

ReceiveOrder

This method is a little complex. It has the following steps:

1. Set ReceiveDate to DateTime.Now
2. ReceiveProduct (i.e. add to inventory and update on order value) – using a foreach loop
3. If all items received for the original purchase order, close purchase order

The code for this method is:

Listing 8: ReceiveOrder

```
public void ReceiveOrder(int receiveOrderNumber, List<ReceivedProducts> orderDetails)
{
    // Setup transaction area
    using (var context = new eStoreContext())
    {
        // 1. Set ReceivedDate
        ReceiveOrder order = context.ReceiveOrder.Find(receiveOrderNumber);
        order.ReceivedDate = DateTime.Now;
        // 2. ReceiveProduct (i.e. add to inventory and update on order value)
        foreach(ReceivedProducts item in orderDetails)
        {
            Product product = context.Product.Find(item.ProductID);
            product.QuantityOnHand += item.ReceivedQty;
            product.OnOrder -= item.ReceivedQty;
            context.Entry(product).State = System.Data.Entity.EntityState.Modified;
        }//end for
        // 3. If all items received for the original purchase order, close Purchase Order
        PurchaseOrder po = context.PurchaseOrder.Find(order.PurchaseOrderNumber);
        List<PurchaseOrderDetail> products = po.PurchaseOrderDetails.ToList();
        bool orderCompleted = true;
        foreach(PurchaseOrderDetail detail in products)
        {
            Product p = context.Product.Find(detail.ProductID);
            if(p.OnOrder != 0)
            {
                orderCompleted = false;
            }
        }//end for
        if (orderCompleted)
        {
            po.CompletedDate = DateTime.Now;
            context.Entry(po).State = System.Data.Entity.EntityState.Modified;
        }
        context.SaveChanges();
    }//end using
}//eom
```

BUILD! Fix any errors before proceeding.

Web Form – ReceiveOrder.aspx

Create a new web form called **ReceiveOrder.aspx** in the **Purchasing** folder of your web site. Replace the code Content block with the contents of the **ReceiveOrder_aspx.txt** file. Remember to add the **MessageUserControl** to the <div> below the comment.

The supplied code has the `OnClick` events set for all the buttons. The code for these buttons is in the **ReceiveOrder_aspx_cs.txt** file. Copy these methods into the **ReceiveOrder.aspx.cs** file below the **PageLoad** method.

LoginButton_Click

On the **PurchaseOrder.aspx** web form, the drop down list of suppliers used an ODS control. On this web form, we will use code to put data on the drop down list. The reason for using code is, once the Receive Order is received (i.e. the ReceivedDate is not null), we need to update the list of Receive Orders. Using an ODS will not allow us to easily update the list.

Items are added to the drop down list once an employee logs in. The drop down list will be updated after an order is received. The code for adding items to our drop down list programmatically is:

Listing 9: LoginButton_Click

```
protected void LoginButton_Click(object sender, EventArgs e)
{
    int employeeID = int.Parse(EmployeeListDDL.SelectedValue);
    if(employeeID > 0)
    {
        EmployeeListDDL.Enabled = false;
        FetchButton.Enabled = true;
        LoadRO_DDL();
    }
    else
    {
        MessageUserControl.ShowInfo("LOGIN ERROR", "NO EMPLOYEE SELECTED!");
    }
}//eom
```

Listing 10: :LoadRO_DLL

```
protected void LoadRO_DDL()
{
    ROListDDL.Items.Clear();
    ReceiveOrderController controller = new ReceiveOrderController();
    List<SelectionList> items = controller.GetOpenReceiveOrders();
    ROListDDL.AppendDataBoundItems = true;
    ROListDDL.Items.Add(new ListItem("Select Order", "0"));
    foreach(SelectionList item in items)
    {
        ROListDDL.Items.Add(new ListItem(item.DisplayText, item.IDValue.ToString()));
    }//end for
}//eom
```

FetchButton_Click

This method uses the **MessageUserControl.TryRun** method to handle any exceptions that can happen. It calls the **GetOrderDetails** method. The code is:

Listing 11: FetchButton_Click

```
protected void FetchButton_Click(object sender, EventArgs e)
{
    MessageUserControl.TryRun(() =>
    {
        // Load the list of items on the ReceiveOrder
        ReceiveOrderController controller = new ReceiveOrderController();
        List<ReceivedProducts> products = 
controller.GetOrderDetails(int.Parse(ROListDDL.SelectedItem.Text));
        ROProducts_LV.DataSource = products;
        ROProducts_LV.DataBind();
    });
}//eom
```

ReceiveButton_Click

This method has three steps:

1. Loop through the items on the ListView to create a List<>
2. Call the ReceiveOrder method
3. Reset the form

The code for this method is:

```
protected void ReceiveButton_Click(object sender, EventArgs e)
{
    MessageUserControl.TryRun(() =>
    {
        List<ReceivedProducts> orderDetails = new List<ReceivedProducts>();
        // 1. Loop through the items on the ListView to create a List<>
        foreach (ListViewDataItem lvRow in ROProducts_LV.Items)
        {
            ReceivedProducts product = new ReceivedProducts();
            product.ProductID = int.Parse((lvRow.FindControl("ProductIDLabel") as 
Label).Text);
            product.SKU = (lvRow.FindControl("SKULabel") as Label).Text;
            product.Name = (lvRow.FindControl("NameLabel") as Label).Text;
            product.ReceivedQty = int.Parse((lvRow.FindControl("ReceivedQtyLabel") as 
Label).Text);
            product.OrderedQty = int.Parse((lvRow.FindControl("OrderedQtyLabel") as 
Label).Text);
            orderDetails.Add(product);
        }//end for
        // 2. Call the ReceiveOrder method
        ReceiveOrderController controller = new ReceiveOrderController();
        controller.ReceiveOrder(int.Parse(ROListDDL.SelectedItem.Text), orderDetails);
        // 3. Reset form and send message to MessageUserControl
        MessageUserControl.ShowInfo("Order Received", "Receive Order (" +
ROListDDL.SelectedItem.Text + ") received.");
        LoadRO_DDL();
        ROProducts_LV.DataSource = null;
        ROProducts_LV.DataBind();
    });
}//eom
```

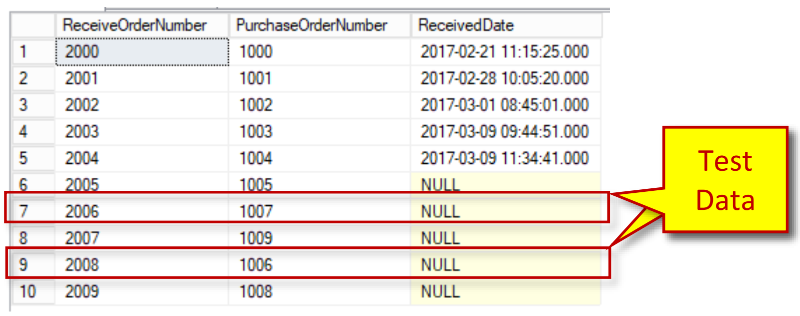
Notice the call to reload the drop down list

Reset the ListView

Test

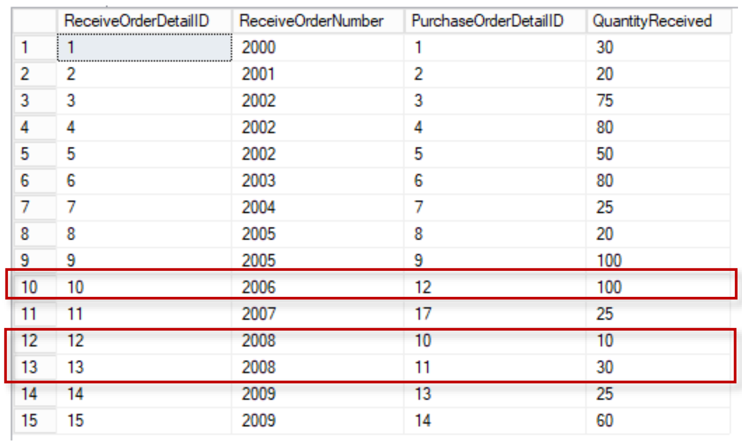
For this test, the database has been reset to its original state.

Data Before the Test



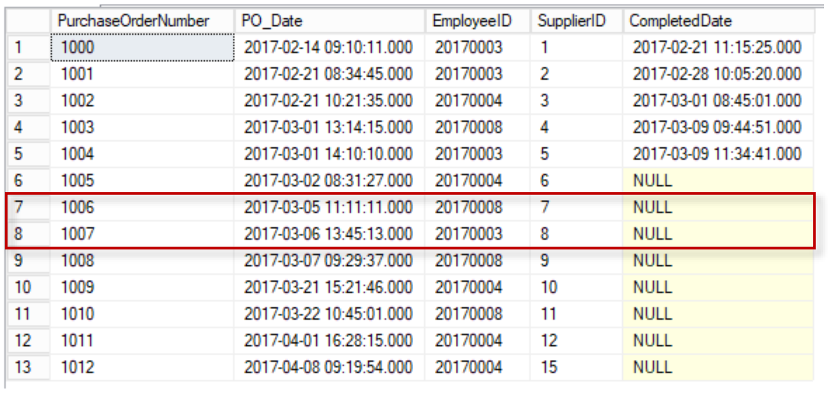
	ReceiveOrderNumber	PurchaseOrderNumber	ReceivedDate
1	2000	1000	2017-02-21 11:15:25.000
2	2001	1001	2017-02-28 10:05:20.000
3	2002	1002	2017-03-01 08:45:01.000
4	2003	1003	2017-03-09 09:44:51.000
5	2004	1004	2017-03-09 11:34:41.000
6	2005	1005	NULL
7	2006	1007	NULL
8	2007	1009	NULL
9	2008	1006	NULL
10	2009	1008	NULL

Figure 9: Data Before Test - ReceiveOrder



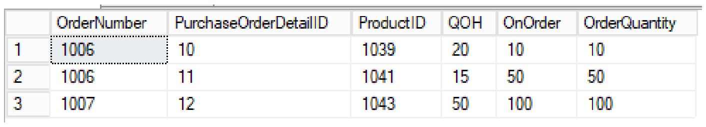
	ReceiveOrderDetailID	ReceiveOrderNumber	PurchaseOrderDetailID	QuantityReceived
1	1	2000	1	30
2	2	2001	2	20
3	3	2002	3	75
4	4	2002	4	80
5	5	2002	5	50
6	6	2003	6	80
7	7	2004	7	25
8	8	2005	8	20
9	9	2005	9	100
10	10	2006	12	100
11	11	2007	17	25
12	12	2008	10	10
13	13	2008	11	30
14	14	2009	13	25
15	15	2009	14	60

Figure 10: Data Before Test - ReceiveOrderDetail



	PurchaseOrderNumber	PO_Date	EmployeeID	SupplierID	CompletedDate
1	1000	2017-02-14 09:10:11.000	20170003	1	2017-02-21 11:15:25.000
2	1001	2017-02-21 08:34:45.000	20170003	2	2017-02-28 10:05:20.000
3	1002	2017-02-21 10:21:35.000	20170004	3	2017-03-01 08:45:01.000
4	1003	2017-03-01 13:14:15.000	20170008	4	2017-03-09 09:44:51.000
5	1004	2017-03-01 14:10:10.000	20170003	5	2017-03-09 11:34:41.000
6	1005	2017-03-02 08:31:27.000	20170004	6	NULL
7	1006	2017-03-05 11:11:11.000	20170008	7	NULL
8	1007	2017-03-06 13:45:13.000	20170003	8	NULL
9	1008	2017-03-07 09:29:37.000	20170008	9	NULL
10	1009	2017-03-21 15:21:46.000	20170004	10	NULL
11	1010	2017-03-22 10:45:01.000	20170008	11	NULL
12	1011	2017-04-01 16:28:15.000	20170004	12	NULL
13	1012	2017-04-08 09:19:54.000	20170004	15	NULL

Figure 11: Data Before Test - PurchaseOrder



	OrderNumber	PurchaseOrderDetailID	ProductID	QOH	OnOrder	OrderQuantity
1	1006	10	1039	20	10	10
2	1006	11	1041	15	50	50
3	1007	12	1043	50	100	100

Figure 12: Data Before Test - Products Ordered

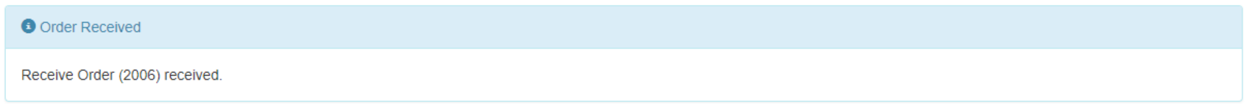
Test Scenario

Use any of the employees for the test.

We will use the two selected Receive Orders (ReceiveOrderNumber = 2006, and 2008) in separate tests from the web form. [**NOTE**: Products may be received in a different order than they were ordered. This is just timing between the supplier getting the order, the supplier filling and shipping the order, and the *store/company* receiving the order.] For ReceiveOrderNumber 2006, the corresponding PurchaseOrderNumber is 1007, and for ReceiveOrderNumber 2008, the corresponding PurchaseOrderNumber is 1006.

If we closely examine the data, we should see that PurchaseOrderNumber 1007 should be complete, and thus will have a CompletedDate set. For the other Purchase Order, the supplier sent not enough products, thus this Purchase Order will still be open. This happens quite often in business inventory transactions.

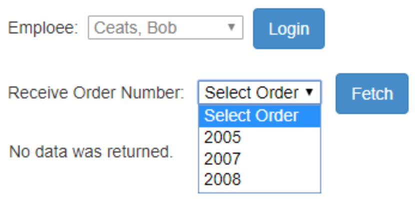
Web Browser Results



Order Received

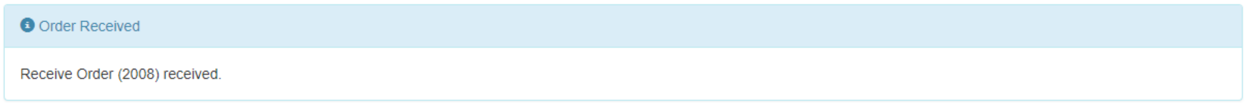
Receive Order (2006) received.

Figure 13: Receive Order 2006 Completed



No data was returned.

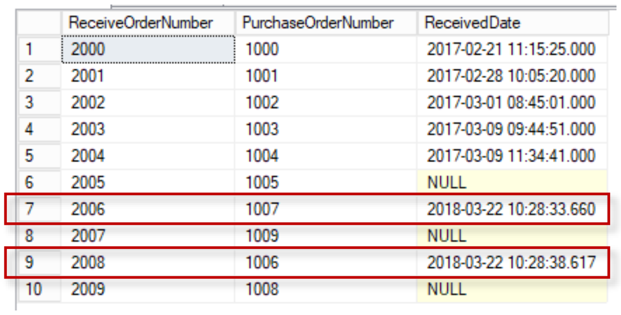
Figure 14: Receive Order 2006 Not Open


Order Received

Receive Order (2008) received.

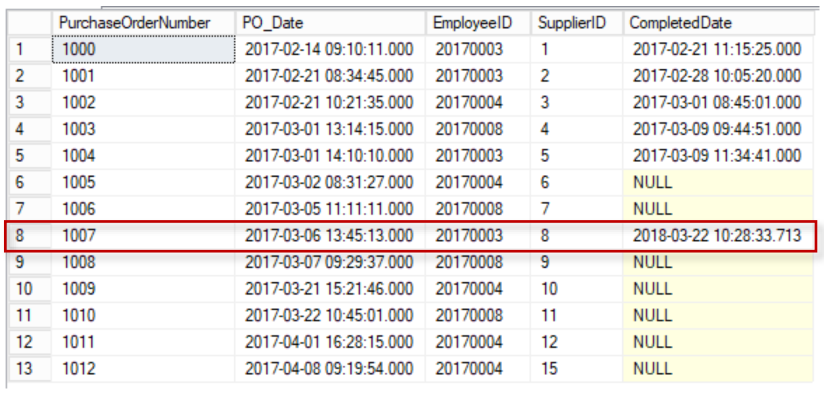
Figure 15: Receive Order 2008 Completed

Test Data after the Test



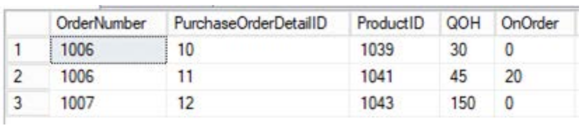
	ReceiveOrderNumber	PurchaseOrderNumber	ReceivedDate
1	2000	1000	2017-02-21 11:15:25.000
2	2001	1001	2017-02-28 10:05:20.000
3	2002	1002	2017-03-01 08:45:01.000
4	2003	1003	2017-03-09 09:44:51.000
5	2004	1004	2017-03-09 11:34:41.000
6	2005	1005	NULL
7	2006	1007	2018-03-22 10:28:33.660
8	2007	1009	NULL
9	2008	1006	2018-03-22 10:28:38.617
10	2009	1008	NULL

Figure 16: Data After Test - ReceiveOrder



	PurchaseOrderNumber	PO_Date	EmployeeID	SupplierID	CompletedDate
1	1000	2017-02-14 09:10:11.000	20170003	1	2017-02-21 11:15:25.000
2	1001	2017-02-21 08:34:45.000	20170003	2	2017-02-28 10:05:20.000
3	1002	2017-02-21 10:21:35.000	20170004	3	2017-03-01 08:45:01.000
4	1003	2017-03-01 13:14:15.000	20170008	4	2017-03-09 09:44:51.000
5	1004	2017-03-01 14:10:10.000	20170003	5	2017-03-09 11:34:41.000
6	1005	2017-03-02 08:31:27.000	20170004	6	NULL
7	1006	2017-03-05 11:11:11.000	20170008	7	NULL
8	1007	2017-03-06 13:45:13.000	20170003	8	2018-03-22 10:28:33.713
9	1008	2017-03-07 09:29:37.000	20170008	9	NULL
10	1009	2017-03-21 15:21:46.000	20170004	10	NULL
11	1010	2017-03-22 10:45:01.000	20170008	11	NULL
12	1011	2017-04-01 16:28:15.000	20170004	12	NULL
13	1012	2017-04-08 09:19:54.000	20170004	15	NULL

Figure 17: Data After Test - PurchaseOrder



	OrderNumber	PurchaseOrderDetailID	ProductID	QOH	OnOrder
1	1006	10	1039	30	0
2	1006	11	1041	45	20
3	1007	12	1043	150	0

Figure 18: Data After Test - Products Ordered

In Figure 18, notice that ProductID = 1041 still has 20 on order (Figure 12 showed that 50 were on the original order, and Figure 10 showed that 30 were received; 50 – 30 = 20). Also, notice the changes in the QOH from before the test, to after the test.