**Department of Computing Sciences**

**The College at Brockport**

**CSC 423: Web Application Development**

**Term Project Description**

**Acknowledgment: This project has been conceived with the kind collaboration of Aptaris, LLC**

**Problem Description**

Retail outlets throughout a certain geographical area receive deliveries from various vendors. Let us assume that a certain retail establishment – a grocery store called Nanno’s Foods – has several outlets (stores) in the local area. Let us consider two vendors that also operate in this area – one a large Food Services corporation, called Parle’s, Inc, which supplies non-alcoholic drinks such as those called “Thums Up” and “Mirinda” to retail outlets, and the other a small operation, the Rogers Farm, which supplies items like eggs, cabbage, corn, etc. to retail outlets. Our project is intended to aid both the retail establishment – Nanno’s Foods – and the vendors who supply them operate optimally. In other words, a vendor would load up their truck and drive it to a certain store they intend to supply only if Nanno’s Foods had indicated that they wanted them to supply some items at that store that week. Also, the vendor would only load those items that Nanno’s Foods had indicated they wanted in that particular store that week into the truck. Once the truck from – say, the Rogers Farm – gets to the concerned Nanno’s store, a representative from Nanno’s would record the various items received, including information about the quantity (number of items) received that day. This becomes part of the store’s inventory. The representative from the Rogers Farm would also *take back* a certain quantity of items that they had earlier provided to the store. For example, if the farm had provided three dozen large brown eggs the previous week, but only two dozen eggs sold, then the remaining dozen eggs may be considered ‘expired’ and cannot be sold to Nanno’s customers any more. Therefore, the Rogers Farm is expected to take them back (it does NOT give credit to Nanno’s Foods for the unsold items – Nanno’s Foods was expected to order responsibly). When the farm loads these unwanted items back on to their truck, the Nanno’s representative records the fact that a dozen large brown eggs were returned to Rogers farm on that day. The system we intend to build should allow Nanno’s Foods to indicate, on a regular basis (e.g., every week), to each vendor they do business with (e.g., to both Parle’s and the Rogers Farm), the various items and associated quantities they expect at a particular store. The vendor should be able to look this information up via the Web and know which items, and in what quantities, they are expected to deliver to stores in the area. The system will also allow the Nanno’s Foods representative to record the items and quantities delivered on a particular day by a particular vendor. It should also allow the representative to record the items and quantities that were taken back by the vendor on that day. Items taken back by the vendor are considered to be taken out of inventory.

In addition, customers at the Nanno’s Foods store will buy the items supplied by the vendors. Each such customer is expected to be registered with Nanno’s Foods – i.e., they are given a “Loyalty Card”, with their own unique ID on it. The item and quantity that the purchased, together with the date of purchase, will be logged, and the purchased item will be considered as being taken out of inventory.

The system will provide the following functionality (“use cases”):

1. Register a vendor with Nanno’s Foods.

For each vendor, information recorded will include vendor code (unique to each vendor), name, address, city, state and ZIP code. In addition, Nanno’s Foods will provide each vendor with a unique id. See the Vendor’s Table schema below for details of what data must be included for each vendor. The password required by the vendor will be set as a part of this use case (at this point, we are not requiring you to implement a password change feature).

1. Modify/Delete a vendor

The user can execute either of these use cases by entering the Vendor Id.

However, if the Vendor Id is not known, then the user should be able to use (part of) the name of a vendor to locate a set of matching vendors, see a table of these, select one of the vendors and modify/delete the vendor. We are not requesting this functionality at this point, you may incorporate it for **EXTRA CREDIT** (amount of credit will be decided by the instructor).

**NOTE:** Due to referential integrity constraints in the database, you may not be able to physically delete a vendor from the Vendor table (for example, because orders placed with this vendor are still in the database, perhaps in ‘Delivered’ status). Therefore, you might consider having a field called ‘ActiveStatus’ in your vendor table, whose value is set to ‘Active’ or ‘Inactive’, depending on whether the Vendor is deleted or not.

1. Add/Modify/Delete store locations of Nanno’s Foods.

Each store location will have a unique store code (indicated by the corporate office), name, address, city, state, ZIP, telephone number and manager name. See the RetailStore table schema below for full details.

As described above, you might implement the modify/delete use cases just by asking the user to enter the RetailStore id. If you implement the search feature for these use cases, you will get **EXTRA CREDIT** (amount of credit will be decided by the instructor).

1. Add/Modify/Delete an item that Nanno’s Foods stocks in its stores

Each item will have a unique ID, a description, a size value, a division value, a category value, a price at which Nanno’s Foods buys the item from the vendor (item cost), a price at which Nanno’s Foods sells this item (retail price) and the name of an image file associated with it. See the schema of the ‘InventoryItem’ table below for further details.

**NOTE:** In the GUI screen associated with this use case, the ‘Division’ and ‘Category’ values should be chosen via a drop-down, as must the vendor information. Note that when selecting the vendor, the vendor’s names must show in the drop-down, not their internal (possibly numerical) Ids. However, as the database table schema says, it is the vendor Id that you need to store into the ‘InventoryItem’ table. Also, note that since you are requested to keep track of whether a Vendor is deleted or not with an ‘ActiveStatus’ field, you should only have Vendors whose ‘ActiveStatus’ is ‘Active’ show up in your drop-down.

Again, the ‘modify’ and ‘delete’ use cases can be implemented using just the Item Id, and a search feature may be implemented for **EXTRA CREDIT** (amount to be decided by instructor). Note that items are also referenced from other tables, so you may not be able to physically delete an item.

1. Add/Modify/Delete customer

Included in this information is a Nanno’s Foods-supplied customer ID, customer name, address, city, state, ZIP, contact number and e-mail. See the schema of the ‘Customer’ table below. Modify and delete may be implemented with the customer id, and the search feature implemented for **EXTRA CREDIT** (amount to be decided by instructor).

1. Create a new order

This use case should allow for the creation of a new order to be delivered to a particular store. See the schema of the “Order” table below. Note that the ‘store id’ expected to be stored as part of a new order should be selected by the user from a drop-down of store *names* (i.e., you have to work on the same lines as the vendor selection for the ‘Add new item’ use case above). The order should be created with the status of “Pending”.

Also, as part of this use case, you have to ensure that at least one item is ordered. In other words, there should be at least one ‘OrderDetail’ (see schema below) associated with this order. The user should be able to *select* the item that is associated with this order detail (i.e., (s)he should not have to type in the item id). Note that the set of items that the user can select from should only be those items that the vendor for whom this order is being placed supplies to Nanno’s Foods (i.e., if the vendor is “Parle’s”, items that are drinks whose description is “Thums Up” and “Mirinda” can be ordered from them, but not an item whose description is “brown eggs”).

**NOTE:** Of course, more than one item can be associated with the order as part of this use case.

1. Add items to existing order

This use case should allow for the selection of an existing order (using an order id) and add items to it. Again, you have to ensure that the items that can be added as ‘OrderDetail’ elements to the order are those that the vendor associated with the selected order provides to Nanno’s Foods.

1. View orders

This use case is only executable by a vendor, who has to log in to the system. This is the only use case a vendor can run – note that all the other use cases here are expected to be run by Nanno’s Foods personnel (and we are not requiring log in for that at this time). When the vendor logs in, they should be able to see all the orders assigned to them, with the ones that are in ‘Pending’ status above all orders with a different status. All orders with a certain status should be sorted by date of order.

Furthermore, if the user clicks on any of these orders, they should be able to see a table of all items (‘OrderDetail’) associated with this order. All data associated with each detail (item id, description and quantity ordered) should be shown in this table.

1. Process a delivery

This use case updates inventory with all items received from a vendor in a delivery. When a vendor brings all the items associated with an order into the designated store, store personnel mark the order as ‘Delivered’ and update the ‘DateTimeOfFulfilment’ value. Therefore, this use case begins with the user entering the id of an Order. Since it is expected that each of the items associated with the order are delivered to the store, the use case then loops through each of the ‘OrderDetail’ items associated with the selected order, and updates the ‘QuantityInStock’ value in the appropriate ‘Inventory’ table entry with the ‘QuantityOrdered’ value from ‘OrderDetail’. Note that selecting the appropriate ‘Inventory’ table entry requires you to keep in mind not just the item id but also the id of the store at which the order was delivered.

1. Process a return

This use case allows the user to create a new ‘ReturnToVendor’ entry (see schema below). Just after this entry is created, this use case should allow the user to add a number of ‘ReturnToVendorDetail’ entries (see schema below). **NOTE:** For each item returned from a store, the corresponding row in the ‘Inventory’ table must be decreased by the ‘QuantityReturned’ value associated with the ‘ReturnToVendorDetail’. Note that the value in the ‘QuantityInStock’ field should NOT fall below zero. If that is in ‘danger’ of occurring, the user should get an error message.

1. Handle a customer purchase

Log a customer purchase – record data as shown in the schema of the ‘CustomerPurchase’ table below. The date/time of purchase should be the current time, and the customer id should be the id supplied to the customer by Nanno’s Foods. This can be entered manually (in real life, a barcode would be scanned/mag-stripe swiped). The store name should be selected from a drop-down, and the appropriate id entered into the database table. Note that the customer can buy any item stocked by the store. Therefore, you can assume that the right item id would be manually entered, since, again, in real life, a barcode would be scanned. Also note that in order to buy the item entered, it must be available in inventory at the store (‘QuantityInStock’ must be greater than zero). As part of the purchase process, the inventory value (‘QuantityInStock’) in the appropriate database table must be reduced by the quantity the customer bought. Of course, the customer should not buy a quantity that will cause the inventory in that store for that item to fall below zero.

Various reports (e.g., search for items in inventory)

1. Obtain a report of all items in inventory. This should show the item id, description, etc. (see schema below) and indicate the ‘QuantityInStock’.
2. Obtain a report of all items in inventory that are overstocked (i.e., whose ‘QuantityInStock’ is running above a certain threshold). In order to run this report, the user should be able to enter the threshold value that they want to start with. Thus, if the user enters ‘10’, then all items whose ‘QuantityInStock’ is higher than 10 should appear in this report.
3. Report items delivered: The user should be able to obtain a report of all items delivered to the store within a certain time period – i.e., between a certain start and end date. The user should be able to select these dates, but when the screen for running this report first comes up, the dates should be populated by default – the start date should be a week before the current date, and the end date should be the current date.
4. Top ten items that are frequently returned to the vendor within a certain time period (say, the past six months): Report all items that have the largest ten total values of the ‘QuantityReturned’ amount. The user should also be able to select the time frame within which these returns occurred – i.e., select a start date and end date. By default, when the screen comes up, the start date should be six months before the current date, and the end date should be the current date.

**IMPORTANT NOTE:** A prime concern in a project such as this one is USER-FRIENDLINESS. If our “sponsor” (Aptaris) uses our project in any way, it is likely to be the GUI screens. Therefore, it is possible that they will – for example – provide us with images of items, and these images should form part of the GUI at the appropriate point(s). You might want to keep in mind other factors that promote ease of use.

**OPTIONAL USE CASES (MORE EXTRA CREDIT POSSIBLE):**

Add use cases that will allow the user to select items that must be associated with a certain Universal Product Code (UPC) and these get entered into the ‘InventoryItemCategory’ table.

**DATA MODEL**



|  |
| --- |
| **RetailStore** |
| StoreId (PK) |
| StoreCode (UNIQUE) |
| StoreName |
| Address |
| City |
| State |
| ZIP |
| Phone |
| ManagerName |

|  |
| --- |
| **Vendor** |
| VendorId (PK) |
| VendorCode (UNIQUE) |
| VendorName |
| Address |
| City |
| State |
| ZIP |
| Phone |
| ContactPersonName |
| Password |

|  |
| --- |
| **InventoryItem** |
| ItemId (PK) |
| Description |
| Size (a TEXT field, with values such as “2 liter”, “16 oz”, etc.) |
| Division (e.g., values such as “Bakery”, “Grocery”, etc.) |
| Department (e.g., “Beer”, “Bread”, etc.) |
| Category (e.g., values such as “Bread”, “Donuts”, etc.) |
| ItemCost (price at which Nanno’s Foods buys this item from vendor) |
| ItemRetail (price at which Nanno’s Foods sells this item) |
| ImageFileName |
| VendorId (FK to Vendor) |

|  |
| --- |
| **Inventory** |
| InventoryId (PK) |
| StoreId (FK to Store) |
| ItemId (FK to InventoryItem) |
| QuantityInStock |

|  |
| --- |
| **Customer** |
| CustomerId (PK) |
| Name |
| Address |
| City |
| State |
| ZIP |
| Phone |
| E-mail |

|  |
| --- |
| **CustomerPurchase** |
| CustomerPurchaseId (PK) |
| CustomerId (FK to Customer) |
| ItemId (FK to InventoryItem) |
| StoreId (FK to RetailStore) |
| QuantityPurchased |
| DateTimeOfPurchase |

|  |
| --- |
| **Order** |
| OrderId (PK) |
| VendorId (FK to Vendor) |
| StoreId (FK to RetailStore) |
| DateTimeOfOrder |
| Status (e.g., values such as “Pending”, “Delivered” or “Canceled”) |
| DateTimeOfFulfilment |

|  |
| --- |
| **OrderDetail** |
| OrderDetailId (PK) |
| OrderId (FK to Order) |
| ItemId (FK to InventoryItem) |
| QuantityOrdered |

|  |
| --- |
| **ReturnToVendor** |
| ReturnToVendorId (PK) |
| VendorId (FK to Vendor) |
| StoreId (FK to RetailStore) |
| DateTimeOfReturn |

|  |
| --- |
| **ReturnToVendorDetail** |
| ReturnToVendorDetailId (PK) |
| ReturnToVendorId (FK to ReturnToVendor) |
| ItemId (FK to InventoryItem) |
| QuantityReturned |