Formal Specifications

Deliverable 1: Identify the different kinds of users that your application will service.

Customer: A patron who purchases goods from our store. The customer should not be able to make any changes to our data but should be able to look up any product information.

Employee: An employee of the store who is not classified as a manager. Basic employees can look up information about the products, update the quantity of a product, and find, add, and update information for customers.

Manager: A manager of the store. The manager will be able to perform all of the same actions as a regular employee, but can also add, modify, or remove product and employee information.

Deliverable 2: You need to have at least one INSERT statement.

Query 1 - (products.js)

Description: Add new product to inventory
Users: Manager
Input: ProductID: Integer, PricePerCostUnit: Double, CostUnit: String, ProductName: String, DepartmentName: String
Optional Input: Brand: String, QuantityInStock: Integer, ProductionDate: String, BestBeforeDate: String, PLU: Integer, UPC: Integer, Organic: Boolean, Cut: String, Animal: String
Output: “OK” or “Error”
Basic Case: The system will add a product to the database with the supplied inputs.
Notes: If ProductID already exists, return “Error.”

Deliverable 3: You need to have at least one DELETE statement.

Query 1 - (employees.js)

Description: Delete existing employee from database
Users: Manager
Input: EmployeeID: Integer
Output: “OK” or “Error”
Basic Case: The system will remove the specified employee from the database.

Notes: If EmployeeID does not exist in records, return “Error.”

Deliverable 4: You need to have at least one UPDATE statement.

Query 1 - (products.js)

Description: Update product cost for a given ProductID
Users: Manager
Input: ProductID: Integer, PricePerCostUnit: Double, CostUnit: String
Output: “OK” or “Error”
Basic Case: The system updates the price per cost unit and cost unit fields of the item specified by the product ID.
Notes: If ProductID does not exist in records, return “Error.”

Deliverable 5: At least one query must join 3 or more tables.

Query 1 - (customer.js)

Description: Look up all products purchased by a customer
Joined tables: Products, Purchases, Customers
Users: Manager
Input: CustomerID: Integer
Basic Case: All products purchased by the specified customer are displayed.
Notes: If CustomerID does not exist in records, return “Error.”

Deliverables 6 and 7: At least 2 other queries need to join 2 or more tables.

Query 1 - (aisles.js)

Description: Look up all products located in a given aisle
Joined tables: AisleContains, Products, Aisles
Users: Manager, Employee, Customer
Input: AisleNumber: Integer
Basic Case: All products located in the given aisle are displayed.
Notes: If AisleNumber does not exist in records, return “Error.”

Query 2 - (suppliers.js)

Description: Find all products supplied by a given supplier
Joined tables: Product, ProvidedBy, Supplier
Users: Manager
Input: SupplierID: Integer
Basic Case: Display all products supplied by a given supplier
Notes: If SupplierID does not exist in records, return “Error.”

Deliverable 8: At least one query must be an interesting GROUP BY query (aggregation). Describe it.

Query 1 - (customers.js)

Description: Calculate the total amount a customer has spent in a particular department.
Grouped by: DepartmentName
Users: Customer, Employee, Manager
Input: CustomerID: Integer
Basic Case: Displays a listed breakdown of the total dollars spent by that customer in each department.
Notes: If CustomerID does not exist in records, return “Error.”

Deliverables 9-11: Describe the other queries you plan to have (these can be simpler queries), so that you have at least 10 SQL statements overall.

Query 1 - (customers.js)

Description: Add new customer to database
Users: Employee, Manager
Input: CustomerID: Integer, CustomerName: String, CustomerPhone: String, CustomerEmail: String
Output: “OK” or “Error”
Basic Case: A customer is added to the database using the provided input.
Notes: If CustomerID already exists, request valid input.

Query 2 - (customers.js)

Description: Update customer information
Users: Employee, Manager
Input: CustomerID: Integer
Optional Input: CustomerNameNew: String, CustomerPhoneNew: String, CustomerEmailNew: String
Output: “OK” or “Error”
Basic Case: An existing customer’s information in the database is updated with the provided information.
Notes: If CustomerID does not exist in records, request valid input.

Query 3 - (products.js)

Description: Update product quantity
Users: Employee, Manager
Input: ProductID: Integer, QuantityInStock: Integer

Output: “OK” or “Error”
Basic Case: The quantity in stock of an existing product in the database is updated.
Notes: If ProductID does not exist in records, request valid ProductID input.

Query 4 - (employees.js)

Description: Update employee information
Users: Manager
Input: EmployeeID: Integer
Optional Input: EmployeeNameNew: String, EmployeeDepartmentNew: String, EmployeePositionNew: String, EmployeeSINumberNew: Integer, EmployeeAddressNew: String, EmployeePhoneNew: String, EmployeeWageNew: Integer
Output: “OK” or “Error”
Basic Case: An existing employee’s information in the database is updated with the provided information.

Query 5 -

Description: Add new employee to staff
Users: Manager
Input: EmployeeID: Integer, EmployeeName: String, SIN: Integer, DepartmentName: String
Optional Input: Position: String, EmployeePhone: String, EmployeeAddress: String, Wage: Integer
Basic Case: Adds an employee to the database using the provided input.
Notes: If EmployeeID already exists, return “Error.”

Deliverable 12: You need to have at least one view for your database, created using the CREATE VIEW statement in SQL. It should be a proper subset of a table. When the user wants to display all the data from this view, they will get all the columns specified by the view, but not all of the columns in the underlying table.

Query 1 -

Description: Find all products that are low in stock
Users: Manager
Displayed Columns: ProductID: Integer, ProductName: String, QuantityInStock: Integer, DepartmentName: String
Notes: Allows manager of department to determine which products are low in stock and need to be replenished.