VARK Finnish(ed?) Furniture

Business Document
Database Design

Developers:
Ana Carolina de Souza Mendes
Katie Reynolds
Robert Holt
Vinh Nguyen

March, 2019

VARK Finnish(ed?) Furniture

Business Requirement:

Ready-to-assemble Furniture Store will manage inventory information of products and hardware. It will keep track of products' id number, number of parts, type of part, type of hardware used to assemble, orders, inventory. Besides the product side, the company is interested in organizing information related to employees and departments. On the sales side, records pertaining Orders, Order Details, and Discounts will also be organized and properly stored.

Assumptions/Features:

- Every furniture will come with the hardware and parts appropriate for assembly of the products.
- Customer can buy spare parts and hardware.
- Since it's a new store, there is no need to store customer information yet.
- Employees who work in the company for one year will get a 5% raise.
- Employees receive a 15% discount on any purchase.
- The owner of the company has full access of all data.
- Managers will only be allowed to change data related to their own department, but they can view any information in the database.
- Department employees will only be allowed to change data related to their own department, but they cannot view any other information in the database.
- All data needs a Record Log to store changes on each table.
- For simplicity purpose, when records are deleted and they are also foreign keys in other tables, the value for the deleted record will be changed to NULL on the other tables. For example, if a discount is deleted from the Discounts table, all the information in OrderDetails should stay in record, so the discount will simply be changed to NULL.

Rules/Constraints:

- Each table must to have at least an Insert, Update, and Delete procedures.
- Name convention for Tables:
    - SchemaName.DescriptiveName
- Name convention for RecordsLog Tables:
    - RecordsLog.TableNameHistory
- Name convention for Views:
    - TableNameView
- Name convention for Stored Procedures:
    - InsertTableName

    - UpdateTableName
    - DeleteTableName
    - CheckFieldNameExists
- Name convention for Triggers:
    - DescriptiveNameTrigger
- Hardware and Parts hold quantity information for spare parts.
- ID identifiers: Products will have id range from 1 - 49, Parts will have id range from 50 - 99, Hardware will have id range from 100 - 149.
- When order is processed, the quantity in tables Products, Parts, and Hardware will be updated.
- A manager can only manage one department, ie, manager_id in Departments table will be unique.
- When there is a management change in the department, the employees working in that department will also have the management id updated.
- When a employee transfers to another department, the manager id for that employee will be updated to the manager of that new department.
- Any purchase over $100 will receive an automatic discount of 10%.
- A discount can only be lower than 20%, unless it's a "discontinued" discount, which doesn't have a limit.

Stored Procedures

Database System will have answer to the questions like:
- Is employee 'A' eligible for a 5% raise?
- How many employees have salary greater than amount 'X'?
- What is the department with the most number of employees?

Summary of all Stored Procedures:

- **Table Employees:**
    - InsertNewEmployee(@fname varchar(20), @lname varchar(25), @email varchar(100), @phone_num varchar(20), @hire_date DATE, @salary Decimal(8, 2), @manager_id int, @department_id int): Procedure to insert a new employee.
    - RemoveEmployeeFromDepartment(@employee_id int): Procedure to change the manager_id in the department to NULL before deleting employee.
    - RemoveManagerFromEmployee(@manager_id int): Procedure to change the manager_id for the employee to NULL before deleting employee.
    - DeleteEmployee(@employee_id int): Deletes the employee from table with the id. It will also check if employee is manager of a department and/or has records in orders. If yes for any of those, change field employee_id to NULL.

- UpdateEmployee (@employee_id int, @first_name varchar(20), @last_name varchar(25), @email varchar(100), @phone varchar(20), @hire_date date, @salary decimal(8,2), @manager_id int, @department int): Updates an employee's information. Adding in either new data or null as values.
- CheckEmployeeExists(@employee_id int, @result int output): Checks if an ID matches a current employee. Takes in an ID and checks if it matches an ID in the table.
- CheckRaiseEligible(@employee_id int, @input_date date): Checks employees getting 5% raise for one year of service. This query does not all employees that are eligible. This only returns whether a given employee (one) if they will be eligible on a given day. Such as if we want to see if an employee who may not be eligible currently, will be when the raise goes into effect, like the beginning of the fiscal year. Or you could enter current date.
- CheckGreaterSalary(@salary_input decimal(8,2)): Counts employees with salary greater than input_salary.

- **Table Departments:**
  - InsertNewDepartment(@department_name varchar(20), @num_of_employees int, @manager_id int): Procedure to insert a new department.
  - RemoveDepartmentFromEmployee(@department_id int): Procedure to change department id for employees to NULL before deleting department.
  - DeleteDepartment(@department_id int): deletes the department from table with the id.
  - UpdateDepartment (@department_id int, @department_name varchar(20)): Updates an department's name information.
  - CheckDepartmentExists(@department_id int, @result int output): Checks if an ID matches a current department. Takes in an ID and checks if it matches an ID in the table.
  - UpdateDepartmentEmpNum(@department_id int): Updates number of employees but getting a new count and setting that to the value at that department id.
  - MostEmployees(no input): Checks which department has more employees, by declaring and setting a value to the department ID with the max employee number and printing the id.
  - UpdateEmployeeManager(@employee_id int, @department_id int): Updates an employees manager in the employee table when they are moved to a new department.
  - UpdateManagerInDepartment(@department_id int, @manager_id int): Updates the manager id of employees in a department for to a trigger to call when the department's manager_id changes.
  - CheckManagerExists(@department_id int, @result int output): Checks if department has a manager.

- **Table Orders:**
  - CheckOrderExists(@order_id int, @result int output): Procedure to check if order exists.
  - InsertOrder(@order_id int, @order_date date, @employee_id int): Procedure to insert new Order.
  - UpdateOrder(@order_id int, @order_date datetime, @employee_id int): Procedure to update existing order and it saves old information into RecordsLog.OrdersHistory.
  - CascadeDeleteOrderDetailsByOrder(@order_id int): Procedure to delete OrderDetails records containing specific order_id before deleting the order_id in the Orders table.
  - DeleteOrder(@order_id int): Procedure to delete existing order and it saves old information into RecordsLog.OrdersHistory.
  - OrderSummary(@order_id int): Procedure to generate an order summary based on the order id.
  - UpdateProductQuantity(@product_id int, @quantity int): Procedure to check if product belongs to Produtc, Parts, or Hardware. Later, the procedure updates the quantity in the specific table. This procedure will be used in "ProcessOrder" procedure.
  - ProcessOrder(@order_id int): Procedure to finalize order. It calls the procedure "OrderSummary" to display a summary of the order and order details. It also calls "UpdateProductQuantity" to update the quantity in the respective product table.

- **Table OrderDetails:**
  - CheckOrderDetailsExists(@item_id int, @result int output): Procedure to check if order detail exists.
  - InsertOrderDetail(@order_id int, @product_id int, @quantity int, @discount_id int): Procedure to insert new order detail.
  - UpdateOrderDetail(@item_id int, @order_id int, @product_id int, @quantity int, @list_price decimal(10, 2), @discount_id int): Procedure to update existing order detail and it saves old information into RecordsLog.OrderDetailsHistory.
  - RemoveDiscountFromOrderDetails(@discount_id int): Procedure to change discount_id in OrderDetails table to NULL before deleting discount record.
  - DeleteOrderDetail(@item_id int): Procedure to delete existing order detail and it saves old information into RecordsLog.OrderDetailsHistory.

- **Table Discounts:**
  - CheckDiscountExists(@discount_id int, @result int output): Procedure to check if discount exists.
  - InsertDiscount(@disc_description varchar(50), @amount decimal(4, 2)): Procedure to insert new Discount.
  - UpdateDiscount(@discount_id int, @disc_description varchar(50), @amount decimal(4, 2)): Procedure to update existing discount and it saves old information into RecordsLog.DiscountsHistory.

- DeleteDiscount(@discount_id int): Procedure to delete existing discount and it saves old information into RecordsLog.DiscountsHistory.

- **Table Products:**
  - CheckProductExists(@product_id int, @result int output): Procedure to check if product exists.
  - InsertNewProduct(@product_name varchar(50), @product_price decimal(10, 0), @product_qty int, @product_reorderpoint int, @product_unitissue varChar(25)): Procedure to insert new product.
  - UpdateProduct (@product_id int, @product_name varchar(50), @product_price decimal(10, 0), @product_qty int, @product_reorderpoint int, @product_unitissue varChar(25)): Procedure to update product information.
  - DeleteProduct(@product_id int): Procedure to remove product.

- **Table Parts:**
  - CheckPartExists(@part_id int, @result int output): Procedure to check if part exists.
  - InsertPart(@part_name varchar(50), @part_description varchar(200), @part_price decimal(8, 2), @part_qty int, @part_reorderpoint int, @part_unitissue varChar(25)): Procedure to add new part.
  - DeletePart (@part_id int): Procedure to delete part by using part_id.
  - UpdatePart (@part_id int, @part_name varchar(50), @part_description varchar(50), @part_price decimal(8, 2), @part_qty int, @part_reorderpoint int, @part_unitissue varChar(25)): Procedure to update existing part.
  - CheckProductPart(@part_id int, @result int output): Procedure to check which product and part list line the part belong to.

- **Table PartsList:**
  - CheckPartListExists(@partlist_id int, @result int output): Procedure to check if partlist_id exists.
  - InsertPartList(@product_id int, @hardware_id int, @part_id int, @partlist_qty int, @hardwarelist_qty int): Procedure to insert new item into partlist.
  - DeleteProductPartlist (@partlist_id int): Procedure to delete partlist by partlist_id and insert into recordlog.partlisthistory table.
  - UpdatePartList (@partlist_id int, @product_id int, @hardware_id int, @part_id int, @partlist_qty int, @hardwarelist_qty int): Procedure to update existing partlist.

- **Table Hardware:**
  - CheckHardwareExists(@hardware_id int, @result int output): Procedure to check if hardware exists.
  - InsertNewHardware(@hardware_name varchar(50), @hardware_description varchar(100), @hardware_price decimal(6, 2), @hardware_qty int, @hardware_reorderpoint int, @hardware_unitissue varchar(25)): Procedure to insert new hardware.
  - DeleteHardware(@hardware_id int): Procedure to delete existing hardware.
  - UpdateHardware (@hardware_id int, @hardware_name varchar(20), @hardware_description varchar(100), @hardware_price decimal(8,2), @hardware_qty int, @hardware_reorderpoint int, @hardware_unitissue varchar(25)): Procedure to update information about existing hardware.
  - CheckHardwarePart(@hardware_id int, @result int output): Procedure to check which hardware belong to which product in partlist.

Triggers:

Summary of all Triggers:

- **Table Employees:**
  - UpdateDepInEmployeeTrigger: When employee is updated, check to see if the department changed. If department_id is updated, execute the UpdateDepartmentEmpNum with the @department_id.

- **Table Departments:**
  - UpdateManagerTrigger: When manager in Department is update manager, trigger calls update managers in employees table.

- **Table Products:**
  - CheckProductReorderPointTrigger: When a record in Products is updated, triggers checks if quantity is less than reorder point quantity to inform user that there is a need to order more products.

- **Table Parts:**
  - CheckPartReorderPointTrigger: When a record in Parts is updated, triggers checks if quantity is less than reorder point quantity to inform user that there is a need to order more parts.

- **Table Hardware:**
    - CheckHardwareReorderPointTrigger: When a record in Hardware is updated, triggers checks if quantity is less than reorder point quantity to inform user that there is a need to order more hardware.

Users:

Admin: will have permission to access/modify all database.
Manager: will have access to view all the objects of database. Will be able to edit tables related to their department. For example, HR manager can edit employees and departments table.
Cashier: will have access to view information about Products, Parts, Hardware, and Partlist tables to consult inventory. Allowed to edit Orders and OrdersDetails tables. No other rights will be granted to this user set.
Customer: will have access to Products, Parts, and Hardware tables to view the furniture quantity, id, price and description available for purchasing. Any other information will not be accessible.

Design Decision:

- Create partial views of Products, Parts, and Hardware table to allow customer to look up information of products before buying them.
- It is necessary to create a bridge table called PartList to relate information about products, information about the parts it contains, and information about the hardware it contains.

Schema: Sales

Tables:

- Employees: This table will store information about employees, their managers, and the department they belong to.
- Departments: This table will store information about the departments in the company.
- Orders: This table will store information pertaining the header of the orders. The order_id on this table will be used on OrderDetails.
- OrderDetails: This table will store detailed information about an order, ie, it will record the lines of products of each order.
- Discount: This table will store information about different types of discounts that can be applied to an order.

Table: Employees

Employee ID	First Name	Last Name	Email	Phone Number	Hire Date	Salary	Manager ID	Departm ent ID
-	-	-	-					
-	-	-	-					
-	-	-	-					
-	-	-	-					
Small Org: 500 Rows								

Table: Departments

Department ID	Department Name	Manager ID	Number of Employee
Small Org: 15 Rows			

Table: Orders

Order ID	Order Date	Employee ID
-	-	-
-	-	-
-	-	-
-	-	-
Small Org: 1000 Rows		

Table: OrderDetails

Order ID	Item ID	Product ID	Quantity	List Price	Discount
-	-	-			
-	-	-			
-	-	-			
-	-	-			
Small Org: 5000 Rows					

Table: Discounts

Discount ID	Description	Amount (%)
-		-
-		-
-		-
Small Org: 5 Rows		

Schema: ProductSupport

Tables:

- Products: The Products table stores the information for the products available and being offered by VARK Finnished Furniture.
- Parts: The part table stores the information for the larger bits and pieces that make up a portion of the total product and can be requested individually if a customer needs replacement parts.
- Hardware: The hardware table stores the information for the smaller bits and pieces that make up a portion of the total product, and can be requested individually if a customer needs replacement parts.
- PartList: This is a static bridge table, meaning that the table is a link between the product table and the part and hardware tables. The quantities given on this table are static, as they define the quantity of the item necessary for the part list as the partlist pertains to a specific product.

Views:

- ProductsView: This view shows partial information on the Products table. It only displays product_id, product_name, product_qty, and product_price. Accessible by customer.
- PartsView: This view shows partial information on the Parts table. It only displays part_id, part_name, part_description, part_qty, and part_price. Accessible by customer.
- HardwareView: This view shows partial information on the Hardware table. It only displays hardware_id, hardware_name, hardware_description, hardware_qty, and hardware_price. Accessible by customer.

Table:Products

Product ID	Product Price	Product Qty	Product Reorder Point	Product Unit Issue
-				
-				
-				
-				
Small Org: 200 Rows				

Table:Parts

Part ID	Part Name	Part Description	Part Price	Part Qty	Part Reorder Point	Part Unit Issue
-						
-						
-						
-						
Small Org: 200 Rows						

Table:PartList

Partlist ID	Product ID	Part ID	Hardware ID	Hardware Qty	Part Qty
-					
-					
-					
-					
Small Org: 200 Rows					

Table:Hardware

Hardware ID	Hardware Name	Hardware Description	Hardware Price	Hardware Qty	Hardware Reorder Point	Hardware Unit Issue
-						
-						
-						
-						
Small Org: 200 Rows						

Schema: RecordsLog

Tables:

- EmployeesHistory: This table will store information on changes made to the Sales.Employees table.
- DepartmentsHistory: This table will store information on changes made to the Sales.Departments table.
- OrdersHistory: This table will store information on changes made to the Sales.Orders table.
- OrderDetailsHistory: This table will store information on changes made to the Sales.OrderDetails table.
- DiscountHistory: This table will store information on changes made to the Sales.Discount table.
- ProductsHistory: This table will store information on changes made to the ProductSupport.Products table.
- PartsHistory: This table will store information on changes made to the ProductSupport.Parts table.
- HardwareHistory: This table will store information on changes made to the ProductSupport.Hardware table.
- PartListHistory: This table will store information on changes made to the ProductSupport.PartList table.

Model for RecordsLog Tables:

Record ID	Field from table	Field from table	Field from table	Field from table	Field from table	Delete Time
-	-	-	-	-		
-	-	-	-	-		
-	-	-	-	-		
-	-	-	-	-		