Laynes Database Scope Statement Documentation

Aggie Automotive Solutions, LLC.
Zaid Alhilo Carter Ho Dexter Law Jr. Daniel Link

Executive Summary

Our cutting edge Point of Sales(POS) and Inventory database software(DBMS) will serve to assist both store managers and crew members by creating a cost efficient front-end and back-end database management system and point of sales system for analyzing order trends. Our extremely efficient and preemptive digital solution is able to decide what amounts to restocks for the store and provide immersive and yet coherent and easy-to-understand graphical user interfaces(GUI).

1. Our digital solutions will assist crew members during service times by providing a graphical user interface that will allow them to enter customer order information and the system will update the restaurant's order history data, inventory, and sales data.
2. After hours, managers will be able to analyze trends in the order history data collected by the POS over user-specified time intervals and that will allow them to create restocking orders to replenish their on-hand inventory by providing store fill levels for every item and creating supply reorder requests.

We will provide the following three basic services.

1. Designing and building a **computerized central DBMS.**
2. Create the **immersive, coherent, and well-designed GUI(s)**
3. Serve as a liaison where you will be provided with a **dedicated team of account manager and technology consultants** who will work with you during the **construction of the connection** between your the **restuarants local application** on the physical devices and **data housed in** the cloud via **AWS**.

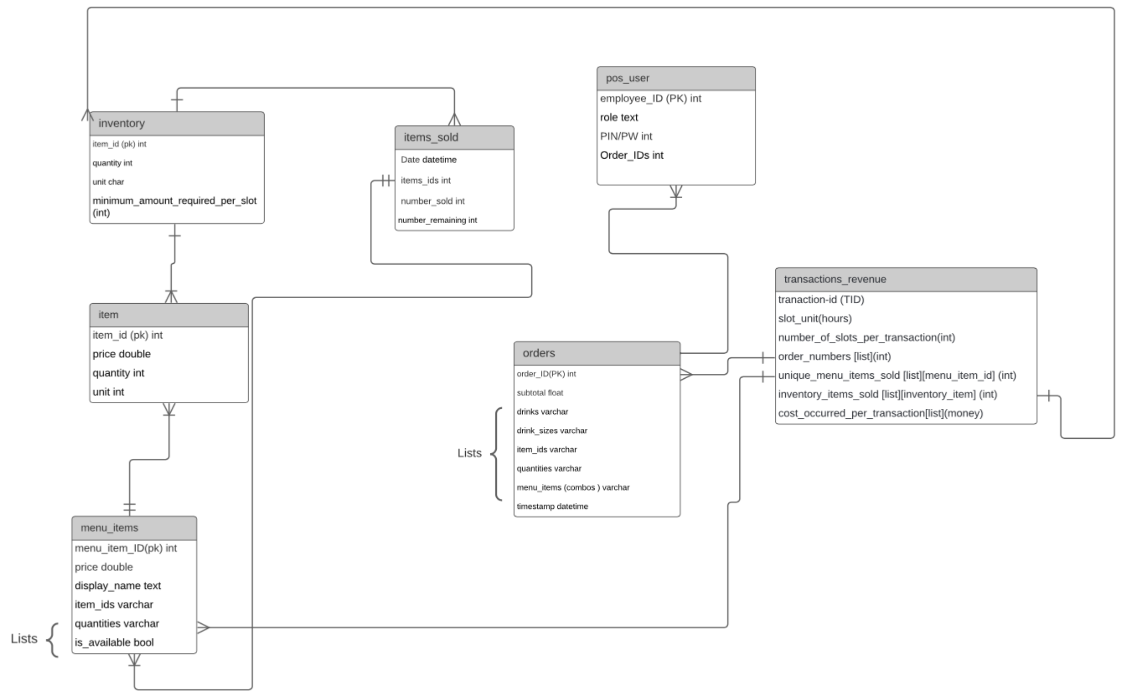
Purpose

Layne's needs a new Point of Sale (POS) system that can track inventory and order history. Employees need a way to put orders into the order history. Management needs accurate tracking to make informed decisions in regards to inventory.

The target audience is employees at Layne's restaurant, specifically servers and managers. They are the focus rather than any other group due to the problem focusing on the sales made and the supplies in the inventory. The servers are the main workers concerning the sales and orders, which will connect to the inventory to keep track of what is available at the restaurant. Furthermore, the system needs to be accessed by managers so they are able to properly restock inventory.

The proposed solution will give the restaurant a point of sale system that will help the business make sales. It will also help the management team analyze the sales of their restaurants by looking at the orders history. The system design will allow the employee to place an order and then collect that order and how much was made out of this order and update the inventory in the moment of the order creation. It will also give the restaurant's management the ability to see how much inventory they have left and how much money they made which will allow them to graph the data or visualize it in different ways.

High-Level Entity Design



Low-Level Entity Design

The UML diagram below describe the design of our system when the employee will use **pos_user** entity to generate an order in the **orders** entity. Now the **orders** entity will use the **menu_items** entity to make sure how many items were sold out of this menu item in the order. The **menu_items** entity gets it's data from the **item** and **inventory** entities. Finally, after the order has been made, the **orders** entity update both **revenue** and **items_sold** entity and the **items_sold** will also update the **inventory** since it needs to subtract the number of items sold.

Entity Attributes

Table Name: *Name: inventory—-*
Purpose: Where we will house all the quantities and items that the store uses.
Justification: There will be one item_id that will be the primary key that is used to look up the current quantity in-stock, when the item was first brought into the store, and the last time that the item was updated.
Table Name: menu_items—-
Purpose: Table will be where all information about available menu items are.
Menu items will be anything that a customer can buy on the menu. Every menu item will have an ID as well as the subtotal for that item.
Justification: The display_name will be what it's labeled on the POS software and on the physical menu. The item_ids will be a comma delimited list of item_ids that make up menu items. The quantities will be a similar list of numbers that tell the system how many of each item is in that menu item. The is_available attribute will basically determine whether or not the store has all the necessary pieces of the menu_item in inventory. So if the box meal requires 5 pieces of chicken, and inventory only shows that the store has 4 tenders in the back, is_available will be false.
Table Name: order_history. —-
Purpose: This will include a unique order_ID and the subtotal of the order.
Justification: The drinks and drink_sizes as well as the item_ids and quantities will be lists similar to those described in the menu_items table. Additionally the menu_items will be a list that we just keep menu_item_IDs in when the customer orders them. Lastly, the orders will include a timestamp that just stores when the order was actually made for statistical purposes and store records.

Table Name: item_table —-
Purpose: Contains information on each inventory item needed in our store, edible and inedible, as well as, what food items they are composed of.
Justification: The table's attributes are item_id, price, quantity and unit. Item_id serves as a primary key as is of data type int to identify the specific item in an order. Price is the value of the individual items, and quantity is the number of items.
Table Name: Items_sold —-
Purpose: Is a table which records the inventory changes of items sold, and is used to keep track of the number of supplies available in the inventory.
Justification: The date attribute records when the amount in the inventory changed. The item_id communicates to the inventory which item to change, the number_sold tells the inventory how much to subtract from the current inventory, and the number_remaining is a value received from the inventory to know how much food/supplies is remaining.
Table Name: pos_user—-
Purpose: Keeps track of which employees create which orders. Its attributes include employee_ID, which differentiates employees, while role differentiates what level they are and what data is accessible.
Justification: A manager is allowed to access the sales, while a server is not. THen there is a PIN (or password) that allows the employee to login. Lastly are the Order_IDs which keep track of the orders made by the currently logged-in employee.
Table Name: revenue_table —-
Purpose: Keeps track of revenue values for each business day, week, or hour, depending on the information, and time frame the manager would like to analyze either inventory, sales, or employee, or customer information.
Justification: The manager can how there data is formatted, depending on the time period in seconds, minutes, hours, weeks, or seasons, and analyze a lot of information including a list of order numbers during the transaction time period, a list of distinct menu items sold that day, a list of distinct order items sold that day, and the total cost occurred that day.

Entity Relationships

Pos_user is only involved with making orders, and each user can make many orders and there can be many users, so it is a many-to-many relationship. There are many orders so any relationship stemming from orders has many on its end. For the connection from orders to revenue, there is only one revenu total, so orders to revenu is many-to-one. Items_sold is a singular instance unique to each order, so orders to items_sold is many-to-one. For each order there can be multiple menu_items so it is one-to-many. Then for each menu_item, it is composed of multiple items, so it is one-to-many. Lastly, concerning the inventory's relationships, one inventory is responsible for all the multiple items sold and items that make up each order so both of the inventory's relations from inventory to its connection is one-to-many.

Interactions

We have designed the system in a way that some entities need to interact with others to fill the required data. The first one we have in the design is the **orders** as it interacts with **revenue** once order is created, it will add it to the **revenue** at the moment. It is doing this computation to provide the user of the database with total revenue. The second interaction we have in our design is the **pos_user** with **orders** as it will provide the **orders** with the order placed by the user at that moment. This query type computation is important as it transfer the order with it's details from one entity to another. The third interaction we have in the design is the

Assumptions and Risk

We assume that the front-end we will develop will be deployed on a device like a tablet or other standard POS device. We're also operating under the assumption that employees will need a set of login credentials that will be needed to access the POS system. As per the project guidelines we also assume that the employees will be the only group adding orders, managers will only login to the system to see the sales statistics.

Our proposed implementation of this system does have some risks involved. We're using a large amount of attributes that are to be lists of items (data type VARCHAR). For the orders table, the item_IDs and quantities will need to be carefully manipulated. Our plan is to have item_IDs and quantities set up such that the item_ID[x] will have the quantity of quantities[x]. This should be easy to implement thanks to javascript's string_split() function. Under this model grabbing data to display to the manager can be slow which is why we're dedicating two other tables to collecting the data to display so that we don't have to parse the order history and all the lists in each order to find that data.