Name:

**Table of Contents**

[Project Direction Overview 2](#_Toc121571494)

[Use Cases and Fields 3](#_Toc121571495)

[Structural Database Rules 7](#_Toc121571496)

[Conceptual Entity-Relationship Diagram 9](#_Toc121571497)

[Initial DBMS Physical ERD 10](#_Toc121571498)

[Stored Procedure Execution and Explanations 13](#_Toc121571499)

[Question Identification and Explanations 17](#_Toc121571500)

[Query Executions and Explanations 18](#_Toc121571501)

[Index Identification and Creations 20](#_Toc121571502)

[History Table Demonstration 21](#_Toc121571503)

[Data Visualizations 23](#_Toc121571504)

[Summary and Reflection 25](#_Toc121571505)

# Project Direction Overview

As an individual with hospitality background, I have extensive experience working with Self Pay Kiosk System (SPKS) and I would like to develop a simplified version of a SPKS that is commonly used in restaurants and corporate cafeterias (refer to Stores in the rest of the project).

For a quick overview of my vision of SPKS, I created the following scenario to illustrate how the SPKS works:

|  |
| --- |
| *A customer walked into a Store and wants to buy some lunch. The customer walks up to a kiosk screen and browses for today’s menu offering. The Store offers stations like Chef’s Special, Tossed Salad Bar, Grille, and Fusion. Within each station, the customer can find the menu items on the kiosk screen and can either select a single item or build an item step by step like burgers or tossed salads and added to the cart. By design, modifiers can be added to some customizable items such as amount of dressing on a salad or toasted/untoasted bread on a deli sandwich.*  *Once finished selecting, the customer can proceed to check out on the kiosk screen and a detailed receipt with order number is printed. Consequently, the corresponded Station printers will receive a ticket with the name of the dish as well as the order number. Another core feature of a SPKS is that an administrative user has access to view all sales history to fulfill their financial duties.* |

Given that SPKS are highly customizable at each Store, the onsite managers are often given with the admin access to perform operations such as create, delete, and modify the database that mentioned above. However, since this project doesn’t involve with the View and business logic aspect of the SPKS development, this database is only as beneficial as storing and retrieving data.

Lastly, it’s worth to note that foodservice company don’t want to deal with the cost of the R & D and overhead of maintaining the SPKS, therefore, the brand I was working for (and pretty much the entire hospitality industry) are contracted with third party SPKS company to help us setup the hardware (kiosks, screens, printers, and servers), create the Store account, provide training, and provide regular maintenance, etc.

# 

# Use Cases and Fields

1. Initial Store Setup for a new Location
   1. The sales team signed a contract with a Store to expand the SPKS to a new location.
   2. Each new Store needs to be setup by an IT staff of a SPKS company.
   3. IT staff creates store\_id, store\_name, and tax\_rate for the new Store.
   4. Once created, the new Store is ready to be customized by the Admin User(s).

Store

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| store\_id decimal(12) | The ID of a store | This is necessary for identifying each store |
| store\_name varchar(64) | The name of the store | This is useful to identify the store |
| tax\_rate decimal(2,3) | The local sales tax rate | This is necessary for calculating the local tax rate |
| system\_id decimal(12) | The name of the SPKS System | This is necessary for associating System users to the SPKS |

1. Setup Admin Users/System Users
   1. The SPKS company will determine and choose whether the user will be Admin User or System Users based on the scenario.
   2. If the company hires a new IT, then they create a System User.
   3. If the company’s goal is to setup Admin Users for the Stores, the Store will tell the IT staff how many Admin Users they will need.
   4. Once the numbers of Admin Users have been determined, the IT staff creates admin user accounts so that Admin User haves the access to the Store.

Users

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| user\_id decimal(12) | the id of a user | This is necessary for identifying each user |
| user\_name varchar(64) | name of the store | This is useful to identify the user |
| system\_id decimal(12) | the name of the spks system | This is necessary for associating System users to the SPKS |
| store\_id decimal(12) | the store id of user’s location | This is necessary to grant admin access to their locations. |
| user\_type char(1) | boolean value of user status | This is necessary for identifying each user |

1. View Stations
   1. During Store service time, customer can walk up to the kiosk to browse the menu.
   2. First thing the customer see on the kiosk are the Stations this Store has.
   3. If the customer finds the Station to be intriguing, the customer selects the Station to view that station’s menu items.

Station

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| station\_id DECIMAL(6) | ID of a station | This is necessary for uniquely identify each station |
| station\_name varchar(16) | Name of stations | This is necessary for displaying Stations on the kiosk screen |
| store\_id decimal(12) | Store ID | This is necessary for identifying each store |
| printer\_id varchar(32) | The ID of the printer | This is necessary to associating corresponding printer |

1. Browse and Add Menu Items to the Cart

Single\_Item

* 1. Once the customer selects the Station, the customer can see all the Menu Items from that Station.
  2. Menu items are associate with information such as price, calories, and item descriptions.
  3. There are two types of Menu Items, a Single Item or a Build Your Own Item (BYO Item).
  4. If a customer selects a Single Item, the selection process is finished and can proceed to select more Menu Items or check out.

Modifier\_Item

1. Modifier\_Item behalf like a menu item but instead being a food item, modifiers are the instructions that describe how a menu item should be prepared.
2. For example, the kiosk asks the customer for the salad dressing level or how well-cooked they want their steak to be cooked. The customer can select the Modifier\_Item such as “Light”, “Normal”, and “Extra” for the dressing level and “Rare”, “Medium”, and “Well Done” for the wellness of the steak.
3. Modifier\_Item may contain calories information for places like hospitals or gyms or may associate with up-charge if “Extra” dressing was selected.

BYO\_Item

1. A customer selects a BYO menu item on a Station.
2. Depends on how the BYO item were designed, the SPKS prompts number of steps to asks the customer to select desired ingredients to be included in the BYO menu item.
3. For example, in order to build a burger, first step is to select type of buns (brioche, wheat, multigrain), second step is to select type of patties (beef patties, impossible), third step is to select type of cheese (pepper jack, American, Swiss, cheddar), forth step is to select type of condiments (mayo, mustard, ketchup), and fifth step is to select type of sides (French fries, sweet potato fries, onion rings).
4. Once the BYO menu item has been built, it is added to the cart automatically.

Menu Item

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| item\_id decimal(12) | ID of a menu item | This is necessary for uniquely identify each menu items |
| item\_name varchar(32) | Name of menu items | This is necessary for displaying the menu item on the kiosk screen |
| price decimal(5,3) | Price of an item | This is necessary for displaying the menu price on the kiosk screen |
| description varchar(256) | Item descriptions | This is useful for displaying the menu descriptions on the kiosk screen |
| calories decimal(6) | Item calories | This is useful for displaying the menu calories on the kiosk screen |
| Item\_type char(1) | True or False if this is a Build Your own Item | This is necessary for identifying the subtype of a Menu Item |

1. Browse and add a built-your-own (BYO) Menu item to cart

Step

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| Step\_id | The id of each step | This is necessary for uniquely identify each step |
| Step\_name | The name of each step | This is useful for naming each step |
| Item\_id | The id of associated item | This is necessary to associate each step to its BYO\_item |

1. Checkout order
   1. The customer can add multiple desired Menu Items to the cart.
   2. Once all items have been added, the customer can proceed to checkout.
   3. After the order has been processed, a detail receipt will be printed for the customer which contains order number, timestamp, store ID, store name, line items, prices, tax, and sales total.

Receipt

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| receipt\_id DECIMAL(12) | The ID of a receipt | This is necessary for displaying the id of receipts |
| order\_num DECIMAL(4) | Order number | This is necessary for displaying the menu item on receipts |
| Timestamp DATE | Time when transaction took place | This is necessary for displaying the timestamp on receipts |
| store\_id DECIMAL(12) | Store ID | This is necessary for displaying store id |
| pretax\_sales | Sales before tax | This is necessary for displaying sales before tax |
| tax\_amount | Tax amount | This is necessary for displaying tax amount |
| aftertax\_amount | Sales after tax | This is necessary for displaying the sales amount on receipts |

1. Print Kitchen Orders
   1. The processed order will be printed at adequate printers at the stations depending on what food the customer ordered.

Printer

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| printer\_model | Model of the printer | This is useful to keep track of the printer models |
| printer\_id | ID of the printer | This is necessary for uniquely identify each printer |
| associated\_station\_id | Name of stations | This is necessary for sending the right orders to the right printer |

1. Customer Refund
   1. A customer may ask for a refund for their unsatisfied meals.
   2. The admin user (café manager) may issue a refund through SPKS terminal by identifying customer’s order\_number that was printed on their receipts.

Receipt\_Refund\_History

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| Refund\_id | The refund\_id | This is necessary to identify the refund transaction |
| Receipt\_id | The receipt\_id | This is necessary for associate the receipt\_id |
| Old\_pretax\_amount | The old amount | This is necessary for tracking the old amount |
| New\_pretax\_amount | The new amount | This is necessary for tracking the new amount |
| Change\_date | The time when refund too place | This is necessary for tracking the time of refund |

# 

# Structural Database Rules

1. The SPKS associate with one or many System Users, each System User is associated by a SPKS.

(Mandatory, singular/plural), (Mandatory, singular)

The SPKS needs a System User(s) to perform daily operations and maintenance to the system at a corporate level. Therefore, one to many System Users are necessary for this SPKS that I am designing.

1. A User is a System User, or an Admin User.

(Mandatory, or)

The supertype User, can either be a subtype System User, or a subtype Admin User.

1. The SPKS may operate on zero or many Stores, each Store is operated by one SPKS.

(Optional, plural), (Mandatory, singular)

The SPKS may expand business from zero Store to multiple Stores. In the case of expansion, a System User will create a unique Store profile for each new location.

1. Each Store has one or many Admin Users, each Admin User has one Store.

(Mandatory, singular/ plural), (Mandatory, singular)

Once a Store profile has been created, Admin User(s) can be added to that Store so that those Admin User(s) can start design the menu interface. (Typically, Admin Users are the café managers)

1. Each Store have one to many Stations, each Station has one Store.

(Mandatory, singular/ plural), (Mandatory, singular)

Depends on the sales volume, ranging from 3 to 10 Stations are possible to be found at each Store to divert the sales traffic.

1. Each Station serve one or many Menu Items, each Menu Item may be served by zero to many Stations.

(Mandatory, singular/ plural), (optional, singular/ plural)

Typically, Admin Users will create Menu Items based on daily offering. Unused Menu Items may be taken out of the current menu for future use. In this case, a Menu item may have zero association with any Stations. On the other hand, a popular Menu Item such as “rice” or “mashed potato” may be found at multiple Stations on the same day. In this scenario, each Menu Item may be associated with many Stations.

1. A Menu Item can be either a Modifier\_Item, Single\_Item, or a BYO\_Item.

(Mandatory, or)

A Menu Item have three subtypes, a Modifier, a build-your-own item, or a single item.

1. Each Menu Item may have zero or many Modifiers, each Modifier can be associated with zero to many Menu Items.

(Optional, singular/ plural), (Optional, singular/ plural)

Typically, Modifiers are applied where static menus are being used such as Grill Station, Salad Stations, and Daily Stations. While daily specials are less likely involved with Modifiers.

1. Each BYO Menu Item contains one or many Steps, each Step is used by one BYO Menu Item

(Mandatory, singular/plural), (Mandatory, singular)

A build-your-own Menu item must contain at least one Step and every Step must be unique and used once by its associated item.

1. Each Step contains one or many Menu Item, each Menu Item may be used in zero or many Steps.

(Mandatory, singular/plural), (Optional, singular/plural)

In each Step of building a BYO Menu Item, it contains at least one ingredient (Menu Item) and one ingredient (Menu Item) may be used in different Steps at other BYO Menu Items

1. Each Store may generate zero to many Receipts, each Receipt is generated by one Store.

(Optional, singular/ plural), (Mandatory, singular)

A Store may generate large amounts of transactions per day, or zero if it hasn’t opened.

1. Each Receipt contain one or many Menu Items, each Menu Item may be contained by zero to many Receipts.

(Mandatory, singular/ plural), (Optional, singular/ plural)

When a Menu Item is unpopular, customer may not buy it on that day.

1. Each Station own a Printer, each Printer may be owned by multiple Stations

(Mandatory, Single), (Optional, Plural)

Each Station needs one Printer, but sometimes due to space limitations, multiple Stations can share one printer. Further, in some cases during a printer failure, an Admin user can reroute the Station-Printer relationship so that one Printer is capable to print multiple Station’s orders.

1. Each Receipt may have one or many Refunds, each Refund can only be applied to one Receipt.

(Optional, plural), (Mandatory, singular)

A customer may request several refunds on their order and each Refund can only applied to one Receipt.

# Conceptual Entity-Relationship Diagram

Diagram, schematic

Description automatically generated

This diagram captures the key relationship of my SPKS as well as specialization generalization.

In addition, I want to point out that Modifier, Single Item, and BYO Item all behalf identically in the relational database however, they all have unique definition and usage. Therefore, it is a good practice to differentiate all three of them now for the better clarity when we decided to extend this project to a production environment in the future because there will be attributes that pertain to of them which are not needed at database level.

# Initial DBMS Physical ERD

Diagram, schematic

Description automatically generated

Adding Attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Attribute | Datatype | Reasoning | Example Data |
| Store | store\_name | VARCHAR(64) | It is reasonable to allow 64 digits of numbers and letters for store name. | Café 126 |
| Store | tax\_rate | DECIMAL(2,3) | Tax is ranging from 4 to 13 percent. | 10.00 |
| Users | user\_name | VARCHAR(64) | VARCHAR(64) is used to ensure it captured all possible names. | Hailey Nane |
| Users | user\_type | CHAR(1) | Two subtypes of User table so CHAR(1) is enough | 0 |
| Station | station\_name | VARCHAR(16) | VARCHAR(16) is used to ensure it captured all possible station names. | SEASONAL |
| Printer | printer\_model | VARCHAR(32) | VARCHAR(32) is used to ensure it captured all possible models. | X-Model |
| Receipt | order\_num | DECIMAL(3) | Numbers are ranging from 1 to 999. It repeats once 999 has met. | 135 |
| Receipt | TRAN\_TIME | TIMESTAMP | Receipt needs to capture the date and time of each transaction. | 2022-11-28 15:36:15 |
| Receipt | pretax\_amount | DECIMAL(6,2) | In case the Store are selling high price items or serving large groups, DECIMAL(6,2) is used. | 12.00 |
| Receipt | tax\_amount | DECIMAL(5,2) | Tax is often a fraction of pretax\_sales, (5,2) is sufficient | 1.20 |
| Receipt | aftertax\_amount | DECIMAL(6,2) | In case the Store are selling high price items or serving large groups, DECIMAL(6,2) are used. | 13.20 |
| Menu\_Item | item\_name | VARCHAR(32) | In case we need numbers to show on the item\_name, VARCHAR is used instead CHAR | Bacon (2) |
| Menu\_Item | price | DECIMAL(5,2) | In case the Store are selling high price items, DECIMAL(5,2) is used. | 13.99 |
| Menu\_Item | calories | DECIMAL(4) | No single item will exceed 9999 calories. | 550 |
| Menu\_Item | item\_type | CHAR(1) | Menu\_Item has 3 subtypes, CHAR(1) is sufficient. | 2 |
| Menu\_Item | description | VARCHAR(256) | 256 letters are more than enough to describe a dish as we usually try to shorten it for a more concise information | Atlantic Wild Salmon with Local Honey Glaze |
| Step | Step\_name | VARCHAR(64) | Steps are usually very simple, VARCHAR(64) is sufficient | Pick Your Bun |
| Step | step\_seq | DECIMAL(2) | Typically, an item takes less than 5 steps to complete, I gave DECIMAL(2) in case of exception | 4 |
| Receipt\_Refund\_History | Old\_ Pretax\_amount | DECIMAL(6,2) | In case the Store are selling high price items or serving large groups, DECIMAL(6,2) is used. | 13.30 |
| Receipt\_Refund\_History | New\_ Pretax\_amount | DECIMAL(6,2) | In case the Store are selling high price items or serving large groups, DECIMAL(6,2) is used. | 12.20 |
| Receipt\_Refund\_History | Change\_date | TIMESTAMP | History needs to capture the date and time of each refund. | 2022-11-28 15:36:15 |

All tables are valid and legally represented with constraints, and attributes so they all qualify for 1NF. After further review, there are no partial dependencies so therefore, it also qualifies for 2NF.

When reviewing for transitive dependencies, I noticed that I originally have store\_name attribute under the Receipt table, which is a transitive dependency because receipt\_id -> store\_id -> store\_name. Once I removed the store\_name attribute under Receipt table, I now qualify for 3NF.

By following the rule ‘every determinant must be a candidate key’, I have reviewed and confirmed that no two candidate keys exist within any given entity, therefore, my DBMS physical ERS is compliant with BCNF.

# Stored Procedure Execution and Explanations

Part 1 of the stored procedures is straigtforward as it just passes data into the entities.

Table

Description automatically generated with medium confidence

Part 2 involves stored procedures for the bridge entities and I had to hard coded two procedures because creating modifiers and BYO steps are very subjective to each Store and I don’t have the knowledge yet to dynamically create a stored procedures that invloves unknown number of data rows to be inserted.

Text

Description automatically generated with low confidence

Part 3 involves hard coding a few receipt templates that simulate a few most common combinations that would occurred during real usage. As I mentioned earlier, since my Receipt Entity takes the advantage of using many to many relationship, I can’t dynamically create a stored procedure that captures all the menu items, modifiers, and BYO items with my current knowledge.

Graphical user interface, text, application

Description automatically generated

Executing stored procedures are straightforward except Receipt and Has Entities. They were a bit trickier because without the aid of computer automation from the Controllers and View, it required some plannings for the combinations of different menu types and pre-calculations on the sales price.

Graphical user interface, text, application

Description automatically generated

# Question Identification and Explanations

While a SPKS is targeted to facilitate front of house operation at a restaurant, the potential advantage of using a SPKS is that it also serves as a Point-of-Sale System which it tracks all transactions information such as sales and items for various purposes such as financial reports, sales forecast, or waste management.

With that being said, the most common questions that the restaurant operators ask are:

1. Sales volume within a certain period
2. Most popular items sold within a certain period
3. Most popular modifier by number of modifiers sold within a certain period

# Query Executions and Explanations

1. This is a simple yet powerful query for the operators to retrieve sales volume by timeframe. I used the SUM operator to sum up the aftertax\_amount.

Graphical user interface, text, application

Description automatically generated

1. In order to obtain the item names, I had to join two tables to get to the final answer. With a simple GROUP BY and ORDER BY, I was able to obtain the QTY\_SOLD per item\_name.

Graphical user interface

Description automatically generated with low confidence

1. Very similar to query 2, the only difference is an additional AND with item\_type = 1 where 1 means the modifier.

Graphical user interface, text, application

Description automatically generated

# Index Identification and Creations

Based on the rules identifying columns that need to be indexed, other than all the primary key and foreign key, I have determined that the item calories, name, and type should be nonunique indexes since they have higher chance to be filtered. Although TRAN\_TIME are more heavily used than the item\_type, due to small number of distinctive values, it is not practical to add index on TRAN\_TIME.

Text

Description automatically generated

# History Table Demonstration

Based on the characteristics of a history table, I realized it works like a refund table that automatically stores a refund transaction of an existing receipt. For that reason, I decided to design it into a Receipt\_Refund\_History table and structural database rule and EDRs were updated accordingly.

In real world, the Store managers interact with the View to initial a refund (whole or partial) of an existing receipt, then update both Receipt and Has tables through business logics. Ultimately, a change on Receipt table triggers the SQL trigger function and update itself and store the refund information.

1. To avoid any repetitive and transitive dependencies, this table only contains the old and new pretax amount and the time it was changed other than the primary and foreign keys.

Graphical user interface, text, application

Description automatically generated

1. I first added the sequence and index for the history table. For the trigger, I firstly initiate a sequence for the refund table then I use “AFTER UPDATE” to ensure that I only capture the refunds history after transaction has completed. Secondly, I included a comparison between the old and new pretax\_amount to trigger the insertion of the receipt\_refund\_history table.

Graphical user interface, text, application, email

Description automatically generated

1. In this step, I am testing and validating the trigger by updating the pretax\_amount on one of the receipts. Then I immediately select the receipt\_refund\_history table to ensure it captures all columns as it should.

Graphical user interface, text, application, email

Description automatically generated

# Data Visualizations

Include and explain data visualizations and stories that tell effective stories about the information in a way that people quickly and accurately understand it.

One of the most common visualizations that the operators want to see is the weekly sales. For this reason, I have created a query that searches for all receipts within a week and aggregates the sales by grouping the day of the week of the transaction. The weekly sales report is a powerful tool for not just realizing which days has more sales than the others, but also a forecast report to minimize the operation expense.

Graphical user interface, application

Description automatically generated

Chart, bar chart

Description automatically generated

The second most useful sales report for the operators is the sales by hours report. This report gives the operator a good understanding of customer behavior in order to better staff the Store. In this example, we only have sales history in the span of 4 hours which 13 hours being the busiest followed by 12 hours, but in real world, this report may span across 12 hours and should see multiple peaks.

Graphical user interface, text, application

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

# Summary and Reflection

The purpose of a SPKS is to reduce the labor expense by eliminating the cashier roles. The system must be reliable and accurate at all times. There are many SPKS in the market but only the ones that are secure, reliable, versatile, and the best user experience shines.

The heart of a SPKS is the relationship between the subtypes of a menu item as well as how receipts are interacting with the menu items. I have captured the differences between a modifier, single item, and a build-your-own item and constructed their relationships to the sales receipts through a bridge entity. The use cases and structural database rules provides the main objectives of a SPKS while the conceptual ERD and DBMS physical EDR contains essential relationships and attributes between each entity within a SPKS. To make the database more efficient, I created a few indexes out of the foreign keys and an attribute of some heavy traffic entities while executing common queries.

In this SPKS design, I also utilized the trigger function as my business logic to update my receipt\_refund\_history table since it is a direct built-in feature of SQL, and I don’t have to depend on another service to do so. To better interact with the database, I also outlined some of the common usages of stored procedures to insert data rows to the SPKS and I executed those stored procedures to populate a sample dataset that are used for the system testing and testing. By design, a SPKS should be capable of maintaining sales history and providing various types of sales reports. Therefore, I conducted a few sales reports such as hourly report and weekday reports and I visualized the results by exporting the query result into Microsoft Excel.

Even though this SPKS is overly simplified because the Controller, the View, and the hardware communications are left out in this design stage, but based on my experience with the SPKS, I am still very confident that this database design is robust and can continuously expand without a major reconstruction.