

Supermarket Management System - KwikMart

Rachana Anil, Bishal Agrawal

Northeastern University

DS 5110 - Introduction to Data Management & Processing (Fall 2022)

Overview

We walk in and out of a supermarket regularly but never stop to think how it is working so efficiently. A supermarket would not be able to work so efficiently if not for the software systems they use. From employee management to inventory management, a software application is used for each of the tasks.

In this project we have built a Supermarket Management System, which manages the employees, inventory, various stores, warehouses and customer data.

We have used MySQL and jupyter notebook for the implementation. Various functions, stored procedures, views and triggers were defined to make the system as robust as possible. Each of these were designed to help solve a real use case an employee would face while working at the supermarket. Finally, we also have few data insights that would give us an in depth picture of our data.

Database Design

We have designed our database by keeping all the major aspects and design principles in mind. Some of the key steps toward our database design:

- Defined objective and research
- Identify the major tables and relationships
- Map primary and foreign keys to ensure data consistency
- Identify the relationship constraints between the tables(e.g.: (1:1), (1:M), etc.)
- Data Normalization to 3NF

Relational Schema

Table	Fields
role	<u>role_id</u> , role_name, role_desc
registered_customer	<u>phone_no</u> , first_name, last_name, reward_points
category	<u>category_id</u> , category_name, category_description
store	<u>store_id</u> , street_name, city, state, zip_code, store_area, email
employee	<u>emp_id</u> , <u>role_id</u> , <u>store_id</u> , emp_phone, emp_ssn, first_name, last_name, emp_address, hourly_wage, is_working
item	<u>item_id</u> , <u>category_id</u> , name, description, brand
store_item	<u>store_id</u> , <u>item_id</u> , qty_in_stock, unit_price
discount	<u>store_id</u> , <u>item_id</u> , percent_off, is_active
bill	<u>bill_id</u> , <u>store_id</u> , registered_phone, bill_date
bill_items	<u>bill_id</u> , <u>item_id</u> , quantity, unit_price, discount_amount, net_price
warehouse	<u>warehouse_id</u> , name, address, zip_code, email_id
supply	<u>supply_id</u> , <u>warehouse_id</u> , <u>store_id</u> , supply_date
supply_item	<u>supply_id</u> , <u>item_id</u> , quantity, unit_price
Primary key	<u>underlined fields</u>
Foreign key	<u>red_fields</u>

ERD

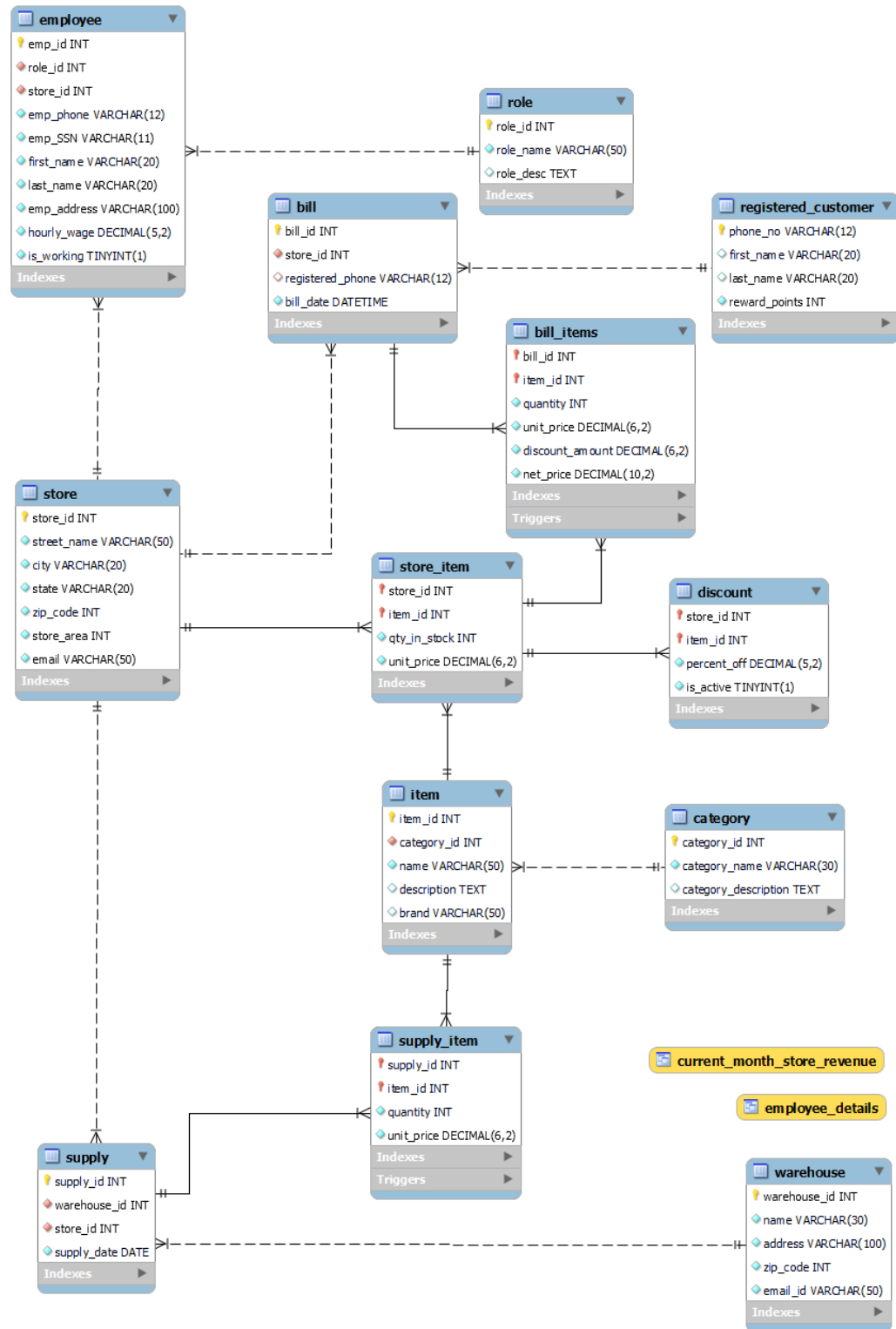
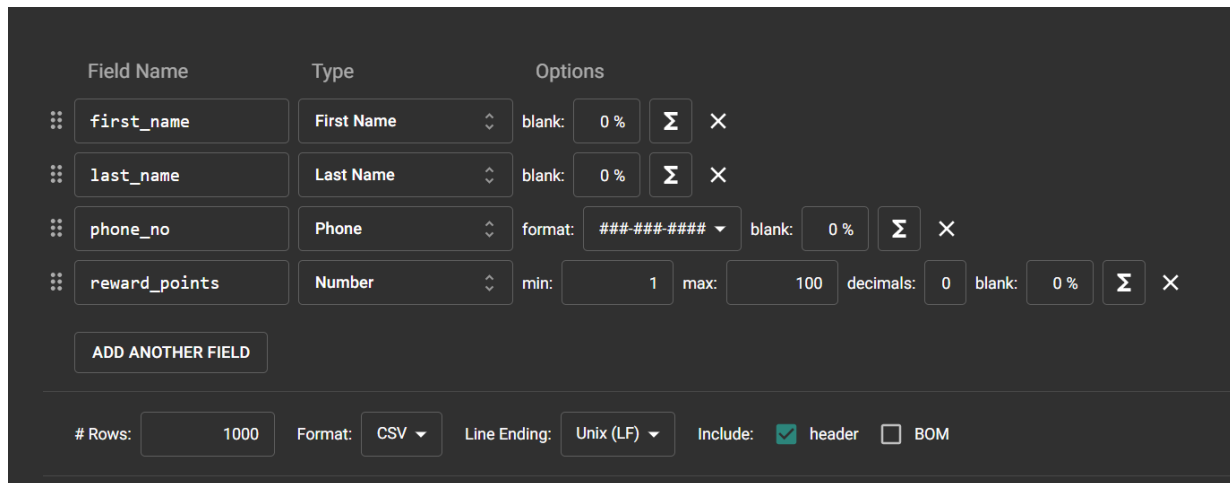


Fig 1: ERD Diagram

Data

Data Generation

We used [Mockaro](#) (a data generation tool) to generate most of our data.

The image shows the Mockaroo web application interface for generating data. It features a table with columns for Field Name, Type, and Options. Four fields are defined: first_name (First Name), last_name (Last Name), phone_no (Phone), and reward_points (Number). Each field has specific options like blank percentage, format, min/max values, and decimals. At the bottom, there are settings for the number of rows (1000), format (CSV), line ending (Unix (LF)), and whether to include a header (checked) or BOM (unchecked).

Field Name	Type	Options
first_name	First Name	blank: 0 %
last_name	Last Name	blank: 0 %
phone_no	Phone	format: ###-###-#### blank: 0 %
reward_points	Number	min: 1 max: 100 decimals: 0 blank: 0 %

ADD ANOTHER FIELD

Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: ☒ header ☐ BOM

Fig 2: Mockaroo Data Generation tool

Enter fields of a table/relation we want to generate and choose a type that is suitable for the field. One can then generate as many records as they would like in a csv format. The generated file can be imported into MySQL using an import option available.

Most of our tables were generated as described.

Data Manipulation

Certain tables like bill_items could not be generated. This is because we had to make sure that our bill_items would contain only those items that were actually in store_items. We used a combination of SQL select statements and excel formatting to generate data for the tables.

Online Resources

We got data for items table and categories table from an online resource added in the reference.

Application Description

Triggers

Our triggers had to be defined before any data insertion to keep our DB consistent. We have defined 2 triggers in this project.

Supply_item to Store_Item

This is an **after insert trigger**. When items are inserted in *supply_item* (this table has all supplies from warehouses to the stores) the same items also need to be added into *store_item* as part of its inventory now.

Bill_items to Store_item

This is a **before insert trigger**. When items are inserted in *bill_items* (items purchased by customers), the same item also needs to be subtracted from the store inventory *store_item*.

store_id	item_id	qty_in_stock	store_id	item_id	qty_in_stock
0	1	5	0	1	93
1	1	58	1	1	141
2	1	99	2	1	76
3	1	205	3	1	79
4	1	302	4	1	172

Fig 3: List of items in stock before and after billing

Stored Procedures

A stored procedure is a SQL code that one can save, so the code can be reused over and over again. We have identified use cases that will be repeated in a supermarket and made a stored procedure for employees to easily use it.

Add item to bill

While getting your items billed at a supermarket, a cashier generally scans an item, this automatically adds it to a bill.

This stored procedure does something similar, the only difference is we need to provide bill_id, item_id and quantity as input to the stored procedure. The procedure adds these items to the provided bill_id.

Generate Bill

This stored procedure is used once all the items are added to the cart and the cashier at the counter wants to generate a bill.. One must provide the required bill_id as an input. The stored procedure returns items in the bill, quantity on each item, discount for each item, total price, total discount and total net price. The figure below, shows a sample bill generated.

KwikMart Supermarket Kiehn Well, Romabury, 78377 2022-12-04 16:19:38.349648 Bill Number: 196		
Item	Quantity	Amount
Vitamins / Supplements	1	23.11
	Discount:	3.47
Insect repellent	2	23.88
Vinegar	2	21.95
	Discount:	2.20
Buns / Rolls	3	17.69
Burger night	1	17.00
Total		184.84
Discount		7.87
Net		176.99

Fig 4: Bill generated using stored procedure

Item Checker

When a particular item is not available in a store, the customer reaches out to an employee to check where the product is available. This procedure comes in handy in such situations. One must enter the current store_id and item_id as input. The procedure returns a list of the nearest stores (in terms of zip), their address and quantity of the item available. The figure below shows nearest stores to store_id =1 having item_id = 100.

Store Address: Erdman Road, East Lorainefort, 68136 Quantity: 36

Store Address: Sauer Motorway, East Korbinmouth, 56259 Quantity: 133

Store Address: Jaycee Glens, Lake Jaydeland, 17644 Quantity: 17

Fig 5: List of closest stores where item is available

Functions

Create Bill

This function is used by a cashier to create a new bill when a customer lines up at a billing counter. The bill can be generated with a mobile number for a registered customer and without a mobile number for unregistered customers. This function returns the generated bill_id.

Overall Revenue

This function can be used by the board of directors or upper management to look at revenues across all stores in a given time frame. This would help them strategize and make important decisions. The function takes start date and end date as input and returns the revenue generated in that time frame.

Store Revenue

This function can be used by the store manager to track revenue generated by the store in a given time frame. The function takes store_id, start date and end date as input and returns the revenue generated in that time frame.

Update Discounts

It's common to add and remove discounts for various items in the store. This function can be used by an employee to update discounts for an item in the store. It takes store_id, item_id, discount percent and is_active(1 for active, 0 for non-active) as input. Returns 1 if the changes were successfully made.

```
Please enter store id: 1
Please enter item id: 99
The discount for item id 99 does not exist. Please add the discount in next step.
Please enter discount %: 10
Please enter 1 to make discount live and 0 to remove discount: 1
(1,)
```

	store_id	item_id	percent_off	is_active
0	1	99	10.0	1

Fig 5: Function to add or update discount for an item at store

Views

Employee details

This view has only the basic details of all the active employees across all the stores. All the sensitive information such as salary, SSN number, and higher level employees (CEO's) information has been hidden.

Current month's store revenue

This view would be useful for upper management roles such as store managers, directors, etc. It has information such as store id, address, store email, net revenue till date along with the previous day's revenue and the total number of bills generated across the stores. This view automatically returns the current month's data without the need of changing the date.

Data analysis

Category wise sales

Variety of products are sold across different stores and it plays an integral part in understanding the sales behavior. This bar plot helps with the visualization with revenue for each category within a given date range.

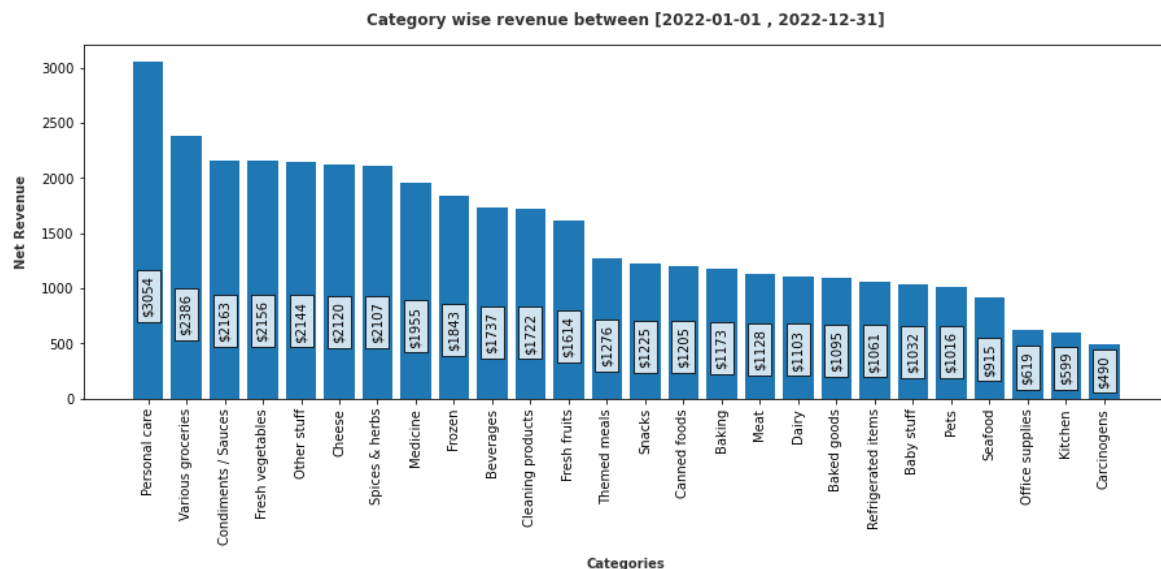


Fig 6: Category wise revenue between 2022-01-01 and 2022-12-31

Number of stores

It is important to develop brand awareness for any organization and therefore it matters for a business to showcase their presence to gain customers' trust. This is a geoplots which basically shows the number of stores in each state.

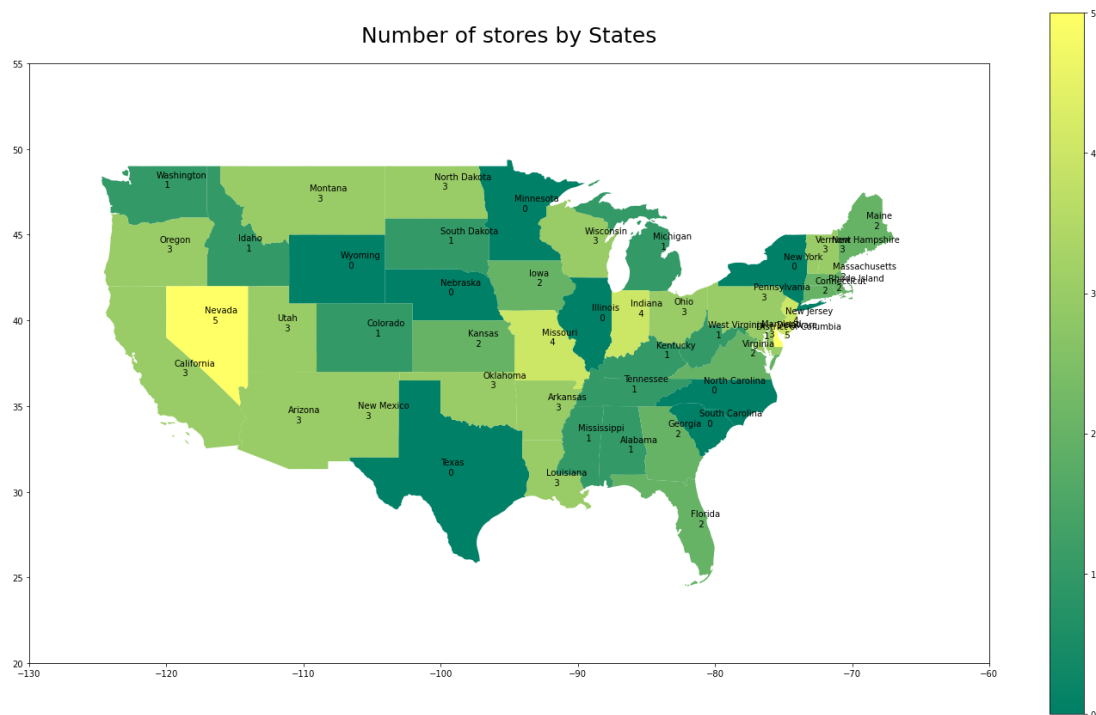


Fig 7: Number of stores in each state

Application Summary

We implemented various functions, stored functions and triggers to perform operations on the database using SELECT, UPDATE, INSERT, etc. We validated the working of all the implementations by thoroughly testing each aspect manually and are confident in the database application we have made. We used Python to connect to the database and simulate a working environment, there could be a few improvements such as building transaction for billing and web page UI can be improved by using alternative frameworks like Angular JS/React that provide us with enhanced functionalities. A mobile application can also be developed using the same database. A fully functional and reliable database system acts as a backbone for any business.

Future Scope

1. Give a feature for customers to use reward points while purchasing.
2. Add features to track profit.
3. Use weight as one of the units of measure. Currently it only supports count.
4. A well designed UI.
5. Integration with other hardwares such as scanners to scan item_id.

Conclusion

Running a supermarket chain is definitely not as simple as it seems from a consumer perspective. The chain needs to manage its warehouses, stores, employees and consumer data.

These tables need triggers to keep the data consistent since many tables like store_item, supply_item and bill_items are interrelated. Functions and stored procedures help non-technical staff to use the system at ease. All the stored data can be used to draw insights about the supermarket functioning and strategize new ventures/ideas.

Finally, no supermarket chain can function smoothly without a robust underlying database management system.

Advice to future DS5110 students

Advice to future DS5110 students: Knowing about database management and having a working knowledge in scripting languages like SQL, MongoDB are great resources that would benefit them. Project is a key aspect of this course and plays an integral role, they should start and think of project ideas as early as possible and work on it as the course progresses. It gives a practical exposure and will help familiarize with the tools involved.

References

1. [Mockaroo](#)
2. [Git](#)