Rutuja Pohare
Utsav Rastogi
Shweta Singh
Paulina Viramontes Juarez

DATA 225 - Database Management Systems

December 12th, 2022

Professor Ron Mak

Term Project: “Team Database Application - DBSmart”

Table of Contents

Acknowledgement

We would like to take a moment to express our sincere gratitude to Professor Ron Mak for his guidance and support throughout the GUI development project for our Database Systems class. Professor Ron's expertise and knowledge in the field was invaluable in helping us understand the complexities of database design and implementation. His willingness to answer our questions and provide feedback on our work was greatly appreciated. We are grateful for his contributions to our learning and growth as students. Thank you, Professor, for your guidance and support.

Introduction - Application Description

This application is designed to help small businesses track their sales and create a customer reward program. The app allows businesses to register transactions made in their supermarket, storing the data gathered in a database. This data can then be used to generate reports on sales, allowing businesses to better understand their customers and their purchasing habits. The app also has a customer reward program feature, which allows businesses to offer rewards and discounts to their regular customers. This can help businesses to build customer loyalty and encourage repeat business. The app is user-friendly and can be easily integrated into a business's existing systems, making it a valuable tool for small businesses looking to improve their sales and customer relationships.

Our Operational GUI is used to register customer data in a supermarket and it includes the following features:

- A user-friendly interface with clear instructions and intuitive navigation.
- Input fields for database managing client to enter their name, email address, and home address.
- A submit button to save the customer data to a back end table.

- A search function that allows businesses to quickly look up a customer's information and purchase history.
- Integration with the supermarket's existing point-of-sale system, allowing customer data to be automatically captured and stored in the database.

By using our GUI, businesses can easily register customer data and generate rewards to encourage repeat business and build customer loyalty. The app's user-friendly design and integration with the business's existing systems make it a valuable tool for small supermarkets.

The Analytical GUI we created for tracking the performance of a supermarket includes the following features:

- A screen displaying key metrics and insights about the store's performance, such as highest selling states, highest selling dates, and top customers.
- Integration with the supermarket's back-end database, allowing the app to automatically gather and analyze data in real-time.
- Insights that can allow the user to create reports on the store's performance.

By using this Analytical GUI, supermarket owners can quickly and easily gain insights into their store's performance, allowing them to make data-driven decisions to improve sales and customer satisfaction. The app's user-friendly design and integration with the store's back-end systems make it a valuable tool for small businesses looking to improve their performance.

Data Sources:

The dataset originated from the tool Mockaroo contains information related to a supermarket, including customer data, information about the items sold, and invoices for those items. The dataset includes the following fields:

- Customer ID: A unique identifier for each customer.
- Customer Name: The customer's first and last name.

- Email Address: The customer's email address.
- Home Address: The customer's home address.
- Post code: The customer's post code.
- Item No: A unique identifier for each item sold.
- Item Type: The name of the item sold.
- Item Quantity: The number of units of the item sold.
- Item Price: The price per unit of the item sold.
- Invoice no: A unique identifier for each invoice.
- Invoice time: The time on which the invoice was generated.

This dataset was used to build tables that support both operational and analytical databases. The operational database allows the supermarket to track and manage sales transactions in real-time, while the analytical database allows the supermarket to gain insights into sales trends and customer behavior. This dataset is a valuable resource for businesses looking to improve their sales and customer relationships.

Customer Sales

Cust_id	first_name	last_name	Cust_address	Cust_state	Cust_postcode	Invoice_no	Invoice_date	Invoice_time	Line_no	Quantity	Item_no	Item_type	Item_price	Discount
X674	Nerti	Pentlow	849 Shasta Trail	Colorado	81010	ZDTR827NO	17/04/2021	10:09 AM	D52	5	FZ349	Soup - Campbellschix Stew	64.4	9
D570	Jordan	Dreier	1 Summit Avenue	Missouri	65211	BMXG722YG	27/09/2021	11:10 PM	B16	10	WG897	Bacardi Breezer - Tropical	95.3	6
J883	Ferdinanda	Vasenkov	589 Blue Bill Park Point	Georgia	30336	MQBW148KK	2/9/2021	7:33 PM	J01	8	FA271	Pasta - Spaghetti, Dry	93	12
N854	Laryssa	Lidstone	3 Eliot Terrace	Texas	75353	VPLT986VP	9/7/2021	9:01 PM	L29	1	EX622	Carrots - Jumbo	13.4	8
C465	Bobette	Philipson	29801 Hagan Place	California	92612	RMTS932JX	12/5/2021	11:02 AM	Z46	8	CN516	Hold Up Tool Storage Rack	6.4	12
L495	Sada	Stokell	8538 Westend Avenue	Ohio	43240	TVPC770BD	7/7/2021	9:08 PM	J71	3	AW555	Island Oasis - Sweet And Sour Mix	67.9	15
S705	Kendre	MacCallam	601 Homewood Street	Georgia	30195	JKLP239FP	9/8/2021	9:46 AM	F51	4	NE918	Wheat - Soft Kernal Of Wheat	33.4	7
H204	Shell	Lefley	09 Mcbride Trail	Ohio	44505	QTMQ907GK	29/08/2021	9:56 PM	D67	5	RI334	Pesto - Primerba, Paste	40.9	9

Operational database design

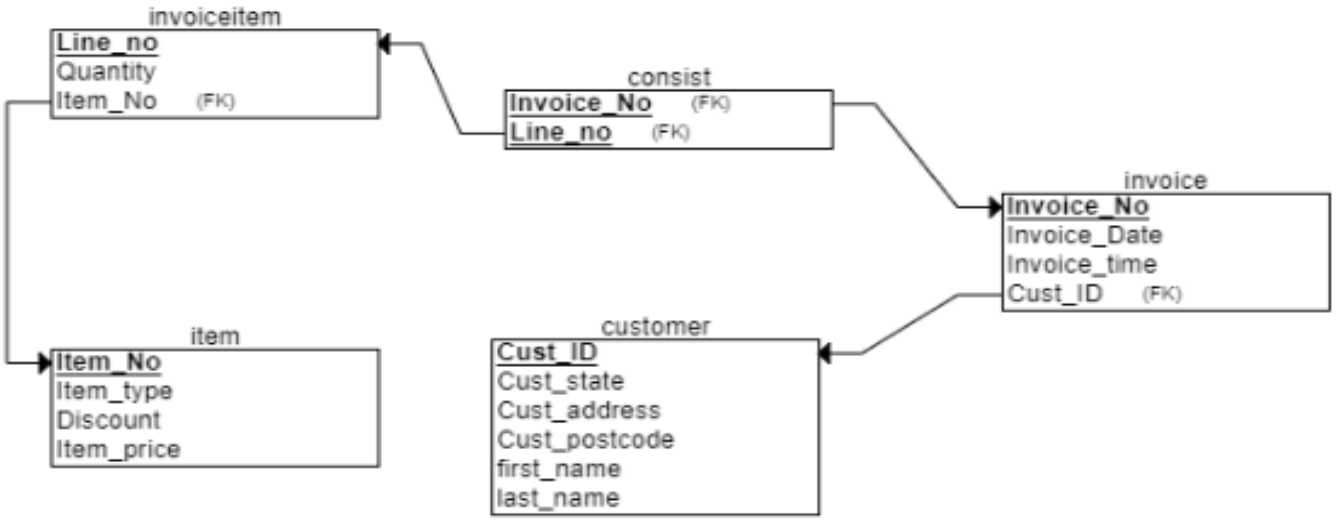
The design of our operational database for a supermarket includes the following 5 tables:

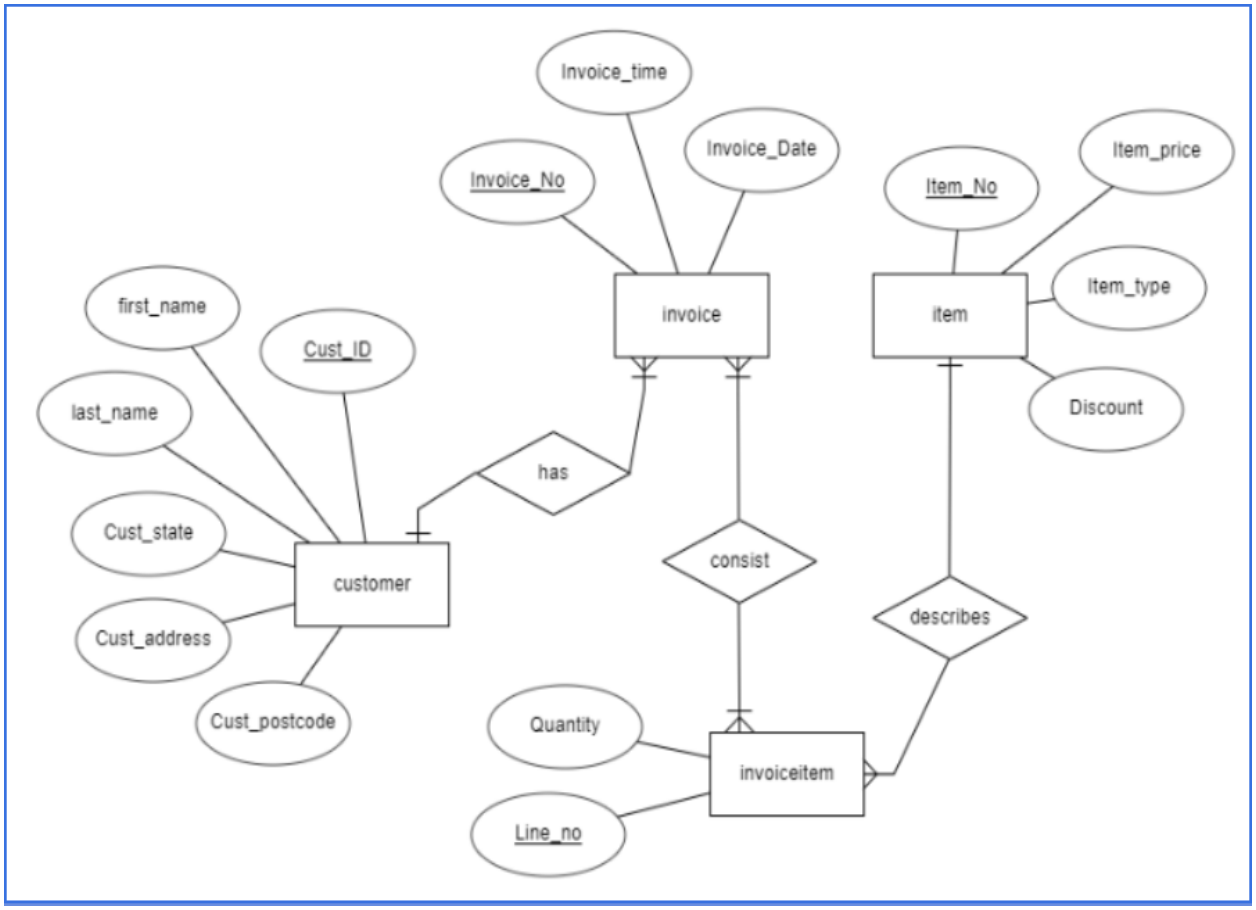
- The consist table would contain information about the line items in an invoice, including the line number and the invoice number. This table would be used to track the individual items that were sold in a transaction.
- The customer table would contain information about the customers, including their name, address, and postcode. This table would be used to store information about the customers who make purchases at the supermarket.

- The invoice table would contain information about each invoice, including the invoice number, the date the invoice was generated, and the customer who made the purchase. This table would be used to track the details of each sale made at the supermarket.
- The invoice_item table would contain information about the items that were sold in each invoice, including the item's name, quantity, and price. This table would be used to track the details of the items that were sold in each transaction.
- The item table would contain information about the items sold at the supermarket, including the item's name, price, and any discounts that were applied. This table would be used to store information about the products available for sale at the supermarket.

These tables could be populated using a CSV file generated by Mockaroo; the process we followed will be described during the ETL part of this document. This would allow the database to be quickly and easily populated with realistic data for testing and development purposes. The data in these tables would be used to support operational processes, such as tracking sales transactions and managing customer information.

Below you can see our relational schema where the 5 tables that were created are shown with its respective columns and the connection between them.





Analytical database design.

The design of an analytical database for a supermarket with 4 tables might include the following:

- The calendar_dimension table would contain information about the date and time of each invoice, including the day of the week, the month, and the year. This table would be used to track the time at which each sale was made.
- The customer_dimension table would contain information about the customers, including their name, address, and postcode. This table would be used to store information about the customers who make purchases at the supermarket.
- The item_dimension table would contain information about the items that were sold in each invoice, including the item's name, quantity, and price. This table would be used to track the details of the items that were sold in each transaction.

- The sales_fact table would contain a summary of the information from the other tables, including the date and time of the sale, the customer who made the purchase, and the items that were sold. This table would be used to store a high-level overview of each sale made at the supermarket.

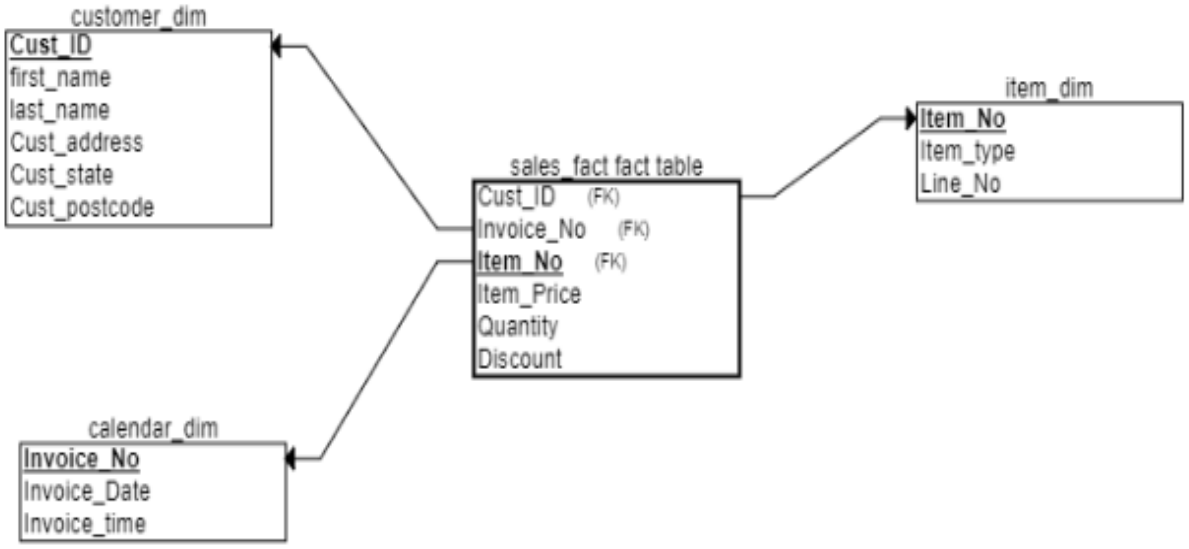
These tables were populated using INSERT SQL queries to connect the analytical database to the tables created in the operational database. This would allow the data from the operational database to be used to support analytical processes, such as generating reports and tracking trends. The data in these tables would be used to support decision-making and business analysis, allowing the supermarket to gain insights into their sales and customer behavior.

In [1]:
```
import pandas as pd
```

In [2]:
```
import mysql.connector as msql
from mysql.connector import Error
```

In [3]:
```
try:
    conn = msql.connect(host='localhost', database='sales_an', user='root',
                        password='Sanjose.9694')
    if conn.is_connected():
        cursor = conn.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()

        print("You're connected to database: ", record)
        cursor.execute('DROP TABLE IF EXISTS sales_fact;')
        print('Dropped sales fact table....')

        sql=('CREATE TABLE sales_fact'
             +'(\n'
             +'Item_Price FLOAT NOT NULL,'
             +'Quantity INT NOT NULL,'
             +'Discount INT NOT NULL,'
             +'Item_No VARCHAR(5) NOT NULL,'
             +'Cust_ID VARCHAR(4) NOT NULL,'
             +'Invoice_No VARCHAR(9) NOT NULL,'
             +'PRIMARY KEY (Item_No, Cust_ID, Invoice_No),'
             +'FOREIGN KEY (Item_No) REFERENCES item_dim(Item_No),'
             +'FOREIGN KEY (Cust_ID) REFERENCES customer_dim(Cust_ID),'
             +'FOREIGN KEY (Invoice_No) REFERENCES calendar_dim(Invoice_No)'
             +')'
             )


except Error as e:
            print("Error while connecting to MySQL", e)
```

Below you will find our star schema which shows the relationship between each table created in our analytical database, we have for tables with their respective columns shown.



ETL processes.

Operational database ETL

We performed the ETL process to create our operational database in the following steps:

- Extracted the data from the CSV file using a tool like Pandas in a Jupyter notebook. This allowed the data to be easily accessed and manipulated in a format that is convenient for data analysis.
- Transformed the data by cleaning and sorting it. This includes tasks such as removing duplicates, filling in missing values, and formatting the data in a consistent way.
- Loaded the transformed data into the operational database. Through the use of SQL queries to insert the data into the appropriate tables.
- Test the data in the operational database to ensure that it is correct and can be used by the application. We did this by running queries against the database to verify that the data is accurate and complete.

This ETL process allows the operational database to be quickly and easily populated with data from the CSV file, ensuring that the data is clean and organized in a way that

is convenient for the application to use. This process is a crucial step in ensuring that the operational database is able to support the needs of the application and the business.

Below you will find some screenshots of the jupyter notebook that was used to perform the ETL in our operational database:

Analytical database

To perform the ETL process in our Analytical Database we followed the steps shown below:

- Created the analytical database and the tables that will be used to store the data. This was through using SQL commands to define the schema of the database and the tables, and to create the tables in the database.
- Extract the data from the operational database using SQL queries, In which we had to select specific columns and rows from the operational database and stored them in our analytical database tables.
- Transform the data by cleaning and sorting it in which we removed duplicates, filled in missing values, and formatted the data in a consistent way.
- Loaded the transformed data into the analytical database using SQL INSERT commands to insert the data into the appropriate tables in the analytical database.
- Test the data in the analytical database to ensure that it is correct and can be used for the GUI development.

This ETL process allows the analytical database to be populated with data from the operational database, ensuring that the data is clean and organized in a way that it can be used to create a GUI application. This process

Below you will be able to see the queries we used to perform ETL in our analytical

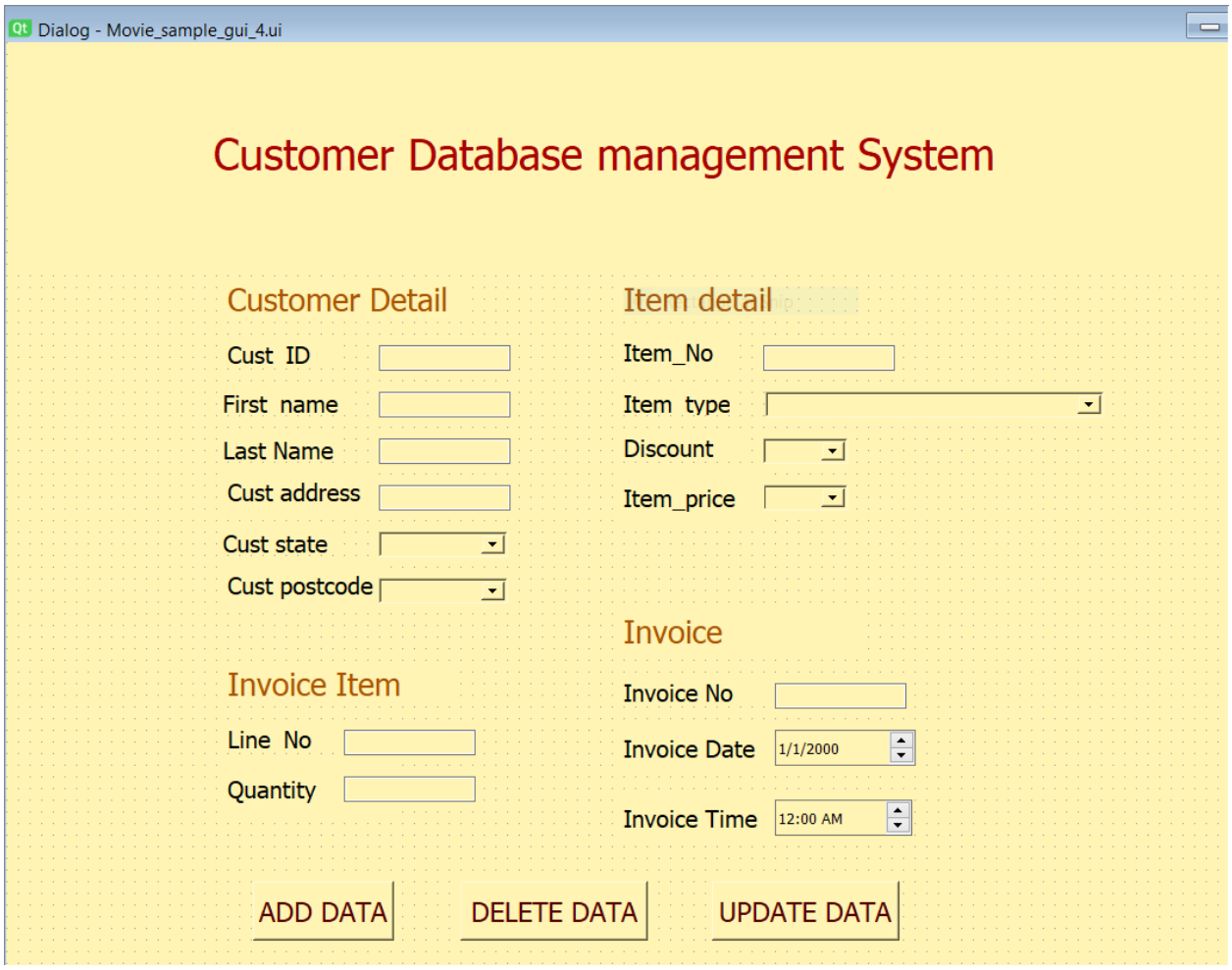
database.

In [4]:
```
try:
    conn = msql.connect(host='localhost', database='sales_an', user='root',
                        password='Sanjose.9694')
    if conn.is_connected():
        cursor = conn.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()

        print("You're connected to database: ", record)
        cursor.execute('DROP TABLE IF EXISTS customer_dim;')
        print('Creating table....')

        sql=('CREATE TABLE customer_dim'
             +'(\n'
             +'Cust_ID VARCHAR(4) NOT NULL,'
             +'first_name VARCHAR(255) NOT NULL,'
             +'last_name VARCHAR(255) NOT NULL,'
             +'Cust_state VARCHAR(35) NOT NULL,'
             +'Cust_address VARCHAR(255) NOT NULL,'
             +'Cust_postcode INT NOT NULL,'
             +'PRIMARY KEY (Cust_ID)'
             +')'
             )
        cursor.execute(sql)
        print("Table is created....")


#         for i, row in df.iterrows():
#             cursor.execute("INSERT INTO sales.customer_dim(Cust_ID,Cust_state,Cust_address,Cust_postcode,first_name,
#                            [row['Cust_ID'],row['Cust_state'], row['Cust_address'], row['Cust_postcode'],row['first_na
#         print('rows executed')
        conn.commit()

except Error as e:
            print("Error while connecting to MySQL", e)
```

In [5]:
```
try:
    conn = msql.connect(host='localhost', database='sales_an', user='root',
                        password='Sanjose.9694')
    if conn.is_connected():
        cursor = conn.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()

        print("You're connected to database: ", record)
        cursor.execute('DROP TABLE IF EXISTS calendar_dim;')
        print('Creating table....')

        sql=('CREATE TABLE calendar_dim'
             +'(\n'
             +'Invoice_No VARCHAR(9) NOT NULL,'
             +'Invoice_Date VARCHAR(15)  NOT NULL,'
             +'Invoice_time VARCHAR(9) NOT NULL,'
             +'PRIMARY KEY (Invoice_No)'
             +')'
             )
        cursor.execute(sql)
        print("Table is created....")


#         for i, row in df.iterrows():
#             cursor.execute("INSERT INTO sales.invoice(Invoice_No,Invoice_Date,Invoice_time,Cust_ID) VALUES (%s,%s,%s
#                            [row['Invoice_No'],row['Invoice_Date'], row['Invoice_time'], row['Cust_ID']])
#         print('rows executed')
#         conn.commit()

except Error as e:
            print("Error while connecting to MySQL", e)
```

```
You're connected to database:  ('sales_an',)
Creating table....
Table is created....
```

Description of Operational GUI:



This is the working model of our Customer database management system. In this application we have used multiple pushbutton, lineEdit, date, time and specially the comboBox function. All these mentioned buttons are used to insert the data into the operational database. Several functional code has been written to enable this GUI in the backend. Few of the screenshots are as follows.

- The below screenshot shows how to enable all the button used in the GUI:

```
class MainWindow(QDialog):
    def __init__(self):
        super().__init__()
        self.ui =uic.loadUi('Movie_sample_gui_4.ui',self)

        self.comboBox.activated.connect(self.selected_state_value)
        self.comboBox_2.activated.connect(self.selected_postcode_value)
        self.comboBox_3.activated.connect(self.selected_itemtype_value)
        self.comboBox_4.activated.connect(self.selected_discount_value)
        self.comboBox_5.activated.connect(self.selected_itemprice_value)

        self.combox_state()
        self.combox_postcode()
        self.combox_item_type()
        self.combox_discount()
        self.combox_price()
        self.show_dialog()
        self.show_date()
        self.show_time()

        #self.ui.pushButton_4.clicked.connect(self.Load_sale_data)
        #self.ui.pushButton_2.clicked.connect(self.delete_data)
        self.ui.pushButton_3.clicked.connect(self.update_data)


        self.ui.pushButton.clicked.connect(self.insert_data)
```

- The below screenshot shows how to insert the required data into the comboBox. So it will be easy to pick from the limited option which is available in the database.

```
def combox_state(self):
    try:
        cursor = conn.cursor()
        cursor.execute("select  distinct Cust_state from customer")
        k = cursor.fetchall()
        for row in k:
            self.comboBox.addItem(''.join(row))
            #print(row)
    except Exception  as e:
        print('error ' + str(e))


def combox_postcode(self):
    try:
        cursor = conn.cursor()
        cursor.execute("select   Cust_postcode from customer")
        l = cursor.fetchall()
        for row in l:
            #print(row[0])
            self.comboBox_2.addItem(str(row[0]))
    except Exception  as e:
        print('error ' + str(e) + "\n\n")
```

- This is a function which is written to insert the date and time from the operational GUI into the operational database in the same dd/mm/yyyy format and hh:mm format so that the translation of dte and time is easy.

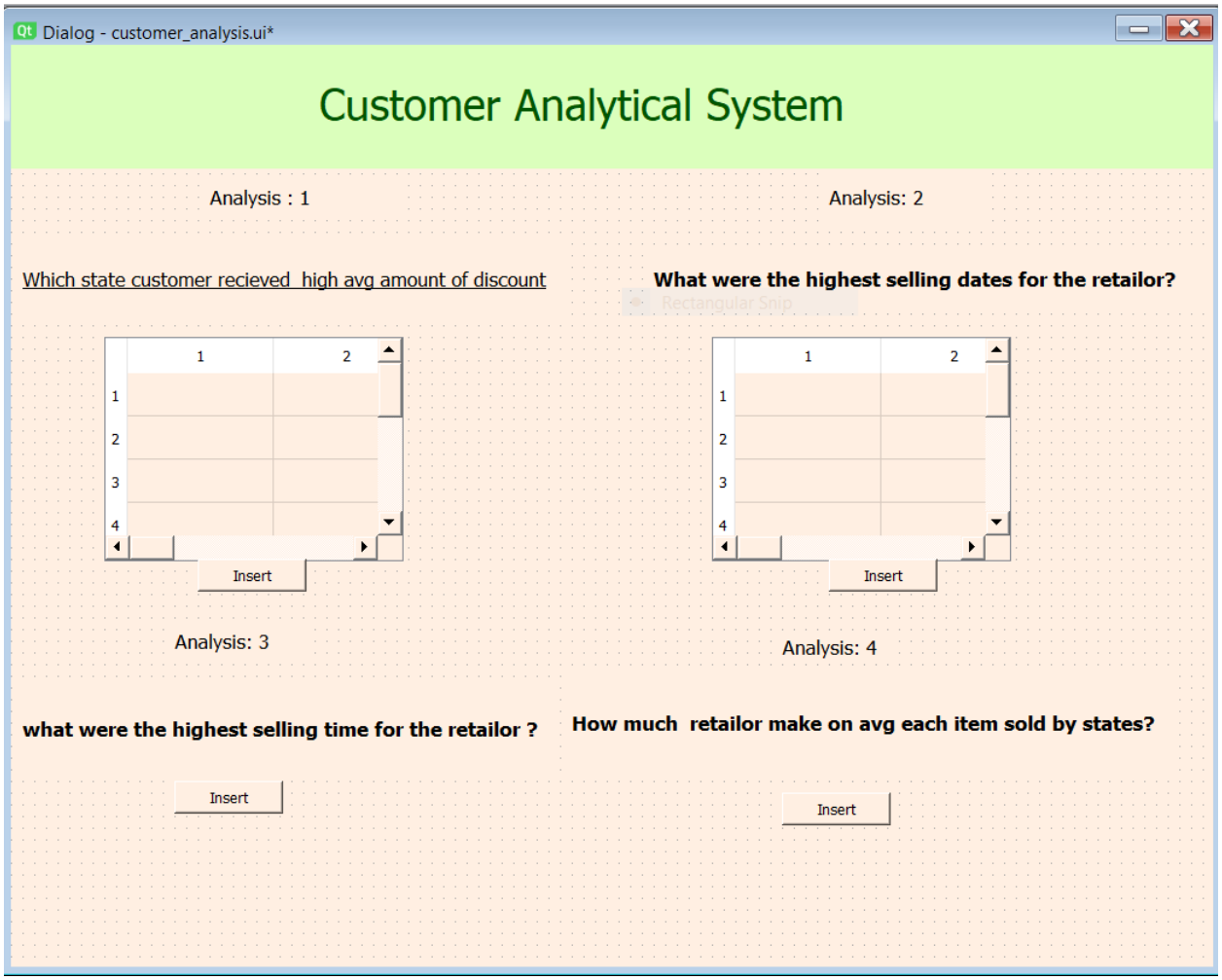
```
Invoice_date = QDate.toString(self.dateEdit.date(), "dd/MM/yyyy")
try:
    Invoice_time = self.timeEdit.time().toString("hh:mm")
    h = Invoice_time.split(":")[0]
    m = Invoice_time.split(":")[1]
    if (int(h) > 12):
        x = int(h) - 12
        Invoice_time = str(x) + ":" + str(m) + " PM"
    else:
        Invoice_time = Invoice_time + " AM"
except Exception as e:
    print("Fail: " + str(e))
```

Use cases for operational GUI:

Some of the use cases for our operational GUIthat gathers customer information from their purchases at a supermarket are the following:

- The application can be used to track customer loyalty and reward program membership, allowing the store to offer special discounts and promotions to their most loyal customers.
- The application can be used to manage customer accounts and preferences, allowing customers to view their purchase history, update their contact information, and set their preferences for receiving marketing communications.
- The application can be used to track customer purchasing behavior and preferences, allowing the store to personalize their product recommendations and offers based on each customer's individual preferences and interests.

Description of Analytical GUI:



This GUI will analyze the questions which are highlighted. By clicking on the insert button the desired output will be fetched from the analytical database. We have used the table widget to insert the analytical data into a tabular form.

In this analytical GUI some effective backend coding has been implemented to enable the GUI.

- The screenshots are shown below.

```
#adding pushbutton
self.ui.pushButton.clicked.connect(self._enter_Avg_discount_data)
self.ui.pushButton_2.clicked.connect(self._enter_sum_Quantity_data)
self.ui.pushButton_3.clicked.connect(self._enter_Avg_discount_time_data)
self.ui.pushButton_4.clicked.connect(self._enter_Avg_price_data)
```

- The Query below is one example of the OLAP functions. And all the other queries which are analyzed in our Analytical GUI are performing the OLAP operations like slice,dice, rollup and drill down.

```
#query 1 with tablewidget
    def _enter_Avg_discount_data(self):
        """
        Enter passanger data from the query into
        the new tables.
        """


        sql_1 = ( """
            select Cust_state, avg(discount)
            from customer_dim, sales_fact
            where customer_dim.Cust_ID=sales_fact.Cust_ID
            Group by Cust_state
            order by avg(discount)DESC
            limit 10;
            """
              )
        cursor.execute(sql_1)
        k = cursor.fetchall()
        print(k)

        # Set the class data into the table cells.
        row_index = 0
        for row in k:
            column_index = 0

            for data in row:
                item = QTableWidgetItem(str(data))
                self.ui.tableWidget.setItem(row_index, column_index, item)
                column_index += 1

            row_index += 1

        self._adjust_column_widths()
        self._adjust_column_widths()
```

- The query written below shows another OLAP function and fetches the output to analyze the average item price for each customer and groups by the customer living in that particular state.

```
#query 4
    def _enter_Avg_price_data(self):


        sql_1 = ( """
           Select Cust_state,avg(Item_price)
           from customer_dim,sales_fact
           where sales_fact.Cust_ID=customer_dim.Cust_ID
           Group by Cust_state
         order by avg(Item_Price) desc
         limit 10;
            """
              )
        cursor.execute(sql_1)
        m = cursor.fetchall()

        for row in m:
            print(row)
```

Use case for analytical GUI:

Some potential use cases for our analytical database GUI are the following:

- The application can be used to track sales trends over time, allowing the owner to see how their sales are changing over the course of a week, month, or year. This could be useful for identifying seasonal trends and making predictions about future sales.
- The application can be used to analyze customer demographics and purchasing habits, allowing the owner to understand who their customers are and what they are buying. This could be useful for targeting marketing campaigns and tailoring the store's product offerings to meet customer needs.
- The application can be used to track the performance of individual products, allowing the owner to see which products are selling well and which are not. This

could be useful for identifying underperforming products and making decisions about which products to stock in the future.

- The application can be used to analyze the store's profitability, allowing the owner to see how much profit they are making and where they are spending the most money. This could be useful for identifying areas where the store could be more efficient and making decisions about how to reduce costs.

Overall, this application has the purpose of providing valuable insights to small supermarket owners, helping them to make data-driven decisions about their business and improve their performance over time as well as creating rewards programs for their frequent buyers to embrace customer retention in the long term.

This application provides valuable information to the store about their customers and helps them to improve their customer experience and build stronger relationships with their customers.

Conclusion

In conclusion, the development of a graphical user interface(GUI) for our database systems class was a successful and rewarding project. By creating both an operational and analytical database, we were able to demonstrate our understanding of database concepts and the importance of data management. Through the process of designing and implementing the database and user interface, we gained valuable experience in working with databases and programming languages. We also learned about the importance of data quality and the role of data analysis.

Overall, this project provided us a valuable learning experience and helped us to develop the skills and knowledge needed to work with databases in a professional setting. We are confident that the skills and knowledge we gained through this project will be useful in our future careers and continue to benefit us as we move forward in our studies and professional lives.

References

We based this project on the information shared by professor Ron Mak in class as well as the following softwares.

- MySQL Workbench , Python IDLE, Jupyter Notebook
- PyQt5 designer.
- ERD plus