

CHAPTER 1

INTRODUCTION

Pharmacy management is a crucial component of the healthcare industry, responsible for the safe and efficient handling of pharmaceutical products and patient-related data. In an era marked by technological advancements and evolving patient expectations, the development of a robust pharmacy management system becomes imperative.

This report documents the design, development, and implementation of a comprehensive Pharmacy Management System, a software application tailored to streamline operations within a pharmacy setting. The primary goal of this project is to create a user-friendly, secure, and efficient system that helps pharmacists, healthcare professionals, and administrators manage various aspects of pharmacy operations, including the tracking of medicines, sales, suppliers, prescriptions, and staff management.

In this report, we provide a detailed account of the system's architecture, features, and functionality. We also discuss the database design, data modelling, and user interactions. Throughout the project, we have adhered to best practices in software development to ensure data integrity, security, and an intuitive user experience.

Our Pharmacy Management System offers several key benefits, including:

- **Inventory Management:** Efficiently manage medicine inventory, reducing the risk of overstocking or understocking essential pharmaceuticals.
- **Sales Tracking:** Record and analyze sales data to facilitate better decision-making, from sales trends to profit margins.
- **Supplier Management:** Simplify interactions with suppliers by maintaining detailed records and facilitating communication.
- **Prescription Management :**Enable healthcare professionals to manage patient prescriptions with ease and precision.
- **Staff Administration:** Streamline staff management, from hiring to role assignments and contact information.

As the pharmaceutical industry continues to evolve, the role of pharmacy management systems becomes increasingly prominent. This report serves as a comprehensive guide to our Pharmacy Management System, detailing its various components, functionality, and the advantages it offers to the healthcare sector. It is our hope that this system will contribute to the enhancement of pharmacy operations and, ultimately, the well-being of patients in our care. This introduction provides an overview of the project's objectives and the importance of a pharmacy management system in the modern healthcare landscape. It sets the tone for the rest of the report, which can delve into technical details, implementation, and other aspects of the project.

CHAPTER2

SCOPE

It may help collecting perfect management in details. In a very short time, the collection will be obvious, simple and sensible. It will help a person to know the management of passed year perfectly and vividly. It also helps in current all works relative to Pharmacy Management System. It will be also reduced the cost of collecting the management & collection procedure will go on smoothly.

- In computer system the person has to fill the various forms & number of copies of the forms can be easily generated at a time.
- In computer system, it is not necessary to create the manifest but we can directly print it, which saves our time.
- To assist the staff in capturing the effort spent on their respective working areas.
- To utilize resources in an efficient manner by increasing their productivity through automation

CHAPTER 3

REQUIREMENTS

Software Requirements

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. The minimal software requirements are as follows,

H/w Requirement :

1. Core i5 processor.
2. 4GB Ram.
3. 20GB of hard disk space in terminal machines.
4. 1TB hard disk space in Server Machine.

S/w Requirement :

1. Front end : Python IDLE / Pycharm
2. Back end : MySql Workbench.

SOFTWARE USED

Python IDLE

Every Python installation comes with an **Integrated Development and Learning Environment**, which you'll see shortened to IDLE or even IDE. These are a class of applications that help you write code more efficiently. While there are many [IDEs](#) for you to choose from, Python IDLE is very bare-bones, which makes it the perfect tool for a beginning programmer.

Python IDLE offers a full-fledged file editor, which gives you the ability to write and execute Python programs from within this program. The built-in file editor also includes several features, like code completion and automatic indentation, that will speed up your coding workflow.

MySql Workbench

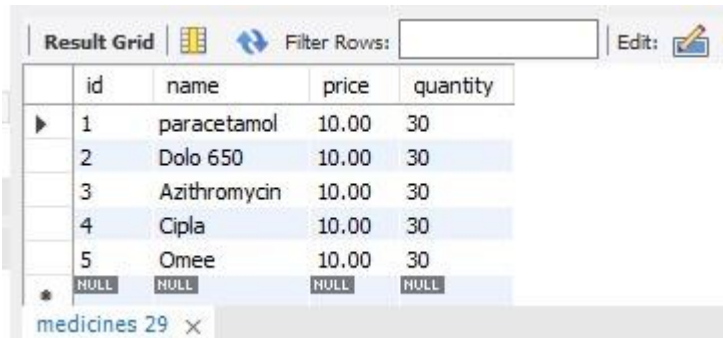
MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases. It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort.

CHAPTER 4

DATA MODELLING

Medicines Table:

- Stores information about available medicines in the pharmacy.
- Primary Key: Medicine_ID



id	name	price	quantity
1	paracetamol	10.00	30
2	Dolo 650	10.00	30
3	Azithromycin	10.00	30
4	Cipla	10.00	30
5	Omee	10.00	30
NULL	NULL	NULL	NULL

medicines 29 x

Fig 4.1 medicines table

2. Sales Table:

- Records sales transactions, including the medicines sold and quantities.
- Primary Key: Sale_ID
- Foreign Key: Medicine_ID (references Medicines)



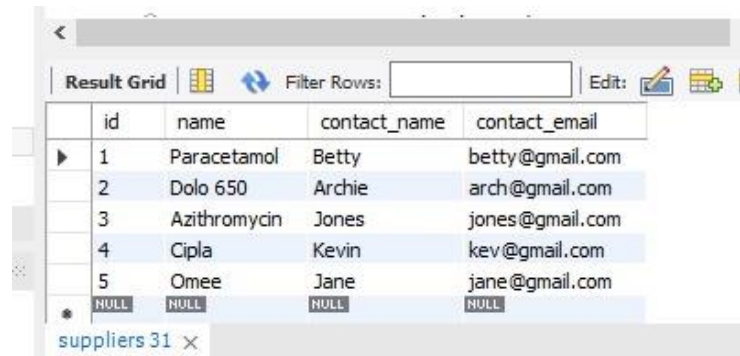
id	medicine_id	quantity_sold	sale_date
1	2	15	2023-10-12
2	1	23	2023-10-13
3	4	12	2023-10-10
4	3	25	2023-10-11
5	5	5	2023-10-11
NULL	NULL	NULL	NULL

sales 30 x

Fig 4.2 Sales table

3. Suppliers Table:

- Contains details about the suppliers of medicines.
- Primary Key: Supplier_ID

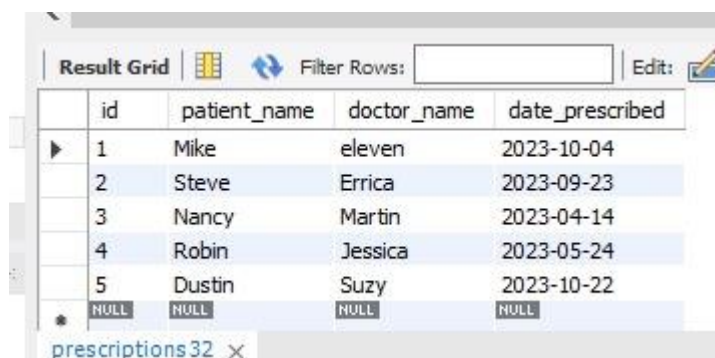


id	name	contact_name	contact_email
1	Paracetamol	Betty	betty@gmail.com
2	Dolo 650	Archie	arch@gmail.com
3	Azithromycin	Jones	jones@gmail.com
4	Cipla	Kevin	kev@gmail.com
5	Omee	Jane	jane@gmail.com
NULL	NULL	NULL	NULL

Fig 4.3 Suppliers table

4. Prescriptions Table:

- Stores prescription records, including patient information and prescribed medicines.
- Primary Key: Prescription_ID




id	patient_name	doctor_name	date_prescribed
1	Mike	eleven	2023-10-04
2	Steve	Errica	2023-09-23
3	Nancy	Martin	2023-04-14
4	Robin	Jessica	2023-05-24
5	Dustin	Suzy	2023-10-22
NULL	NULL	NULL	NULL

Fig 4.4 Prescriptions table

5. Staff Table:

- Stores information about pharmacy staff members.
- Primary Key: Staff_ID



id	name	position	contact_email
10	Moose	Manager	m@gmail.com
20	Eric	Pharmacist	e@gmail.com
30	Xavier	assistant	x@gmail.com
40	Wednesday	Clinical Research Associate	w@gmail.com
50	Tyler	Associate	t@gmail.com
NULL	NULL	NULL	NULL

Fig 4.5 Staff table

CHAPTER 5

DATA DICTIONARY

In a pharmacy management system project, you'll typically work with data stored in dictionaries or data structures. Below is the of list of data dictionaries that are used in a project. These dictionaries represent data for different aspects of the system:

1. *Medicine Data Dictionary:*

- Each medicine in the inventory can be represented as a dictionary.
- Keys: 'ID', 'Name', 'Price', 'Quantity'
- Example:

python

```
medicine = {'ID': 1, 'Name': 'Aspirin', 'Price': 5.99, 'Quantity': 100}
```

2. *Sales Data Dictionary:*

- Each sale transaction can be represented as a dictionary.
- Keys: 'ID', 'Medicine_ID', 'Quantity Sold', 'Sale Date'
- Example:

python

```
sale = {'ID': 1, 'Medicine_ID': 1, 'Quantity Sold': 10, 'Sale Date': '2023-10-15'}
```

3. *Supplier Data Dictionary:*

- Each supplier's information can be stored in a dictionary.
- Keys: 'ID', 'Name', 'Contact Name', 'Contact Email'
- Example:

python

```
supplier = {'ID': 1, 'Name': 'ABC Pharmaceuticals', 'Contact Name': 'John Doe', 'Contact Email': 'john@example.com'}
```

4. *Prescription Data Dictionary:*

- Each prescription can be represented as a dictionary.
- Keys: 'ID', 'Patient Name', 'Doctor Name', 'Date Prescribed'
- Example:

python

```
prescription = {'ID': 1, 'Patient Name': 'Alice', 'Doctor Name': 'Dr. Smith', 'Date Prescribed':  
'2023-10-10'}
```

5. *Staff Data Dictionary:*

- Information about each staff member can be stored in a dictionary.
- Keys: 'ID', 'Name', 'Position', 'Contact Email'
- Example:

python

```
staff = {'ID': 1, 'Name': 'Jane', 'Position': 'Pharmacist', 'Contact Email': 'jane@example.com'}
```

These data dictionaries can be used to represent and manage information in the pharmacy management system. Depending on your specific requirements, you may have additional data dictionaries or more complex data structures for managing relationships between different data elements in the system.

CHAPTER 6

RELATIONAL DATABASE DESIGN

Designing a relational database for a pharmacy management system involves defining the tables, their relationships, and the attributes (columns) within those tables. Below is an example of a relational database design of project. The Sales table is related to the Medicines table through the Medicine_ID foreign key, which links each sale to a specific medicine. The Sales table may also have a Staff_ID foreign key to associate the sales with the staff member who made the sale.

Tables and Relationships:

1. Medicines Table:

- Stores information about available medicines in the pharmacy.
- Primary Key: Medicine_ID

Attribute:

Medicine
Medicine_ID (Primary Key)
Name
Price
Quantity

Fig 6.1 medicine table

2. Sales Table:

- Records sales transactions, including the medicines sold and quantities.
- Primary Key: Sale_ID
- Foreign Key: Medicine_ID (references Medicines)

Attributes

Sales
Sale_ID (Primary Key)
Medicine_ID (Foreign Key)
Quantity Sold
Sale Date

Fig 6.2 Sales tables

3. Suppliers Table:

- Contains details about the suppliers of medicines.
- Primary Key: Supplier_ID

Attributes:

Suppliers
Supplier_ID (Primary Key)
Name
Contact Name
Contact Email

Fig 6.3 Supplier table

4. Prescriptions Table:

- Stores prescription records, including patient information and prescribed medicines.
- Primary Key: Prescription_ID

Attributes:

Prescriptions
Prescription_ID (Primary Key)
Patient Name
Doctor Name
Date Prescribed

Fig 6.4 Prescriptions table

5. *Staff Table:*

- Stores information about pharmacy staff members.
- Primary Key: Staff_ID

Attributes:

Staff
Staff_ID (Primary Key)
Name
Position
Contact Email

Fig 6.5 Staff table

CHAPTER 7

E-R DIAGRAM

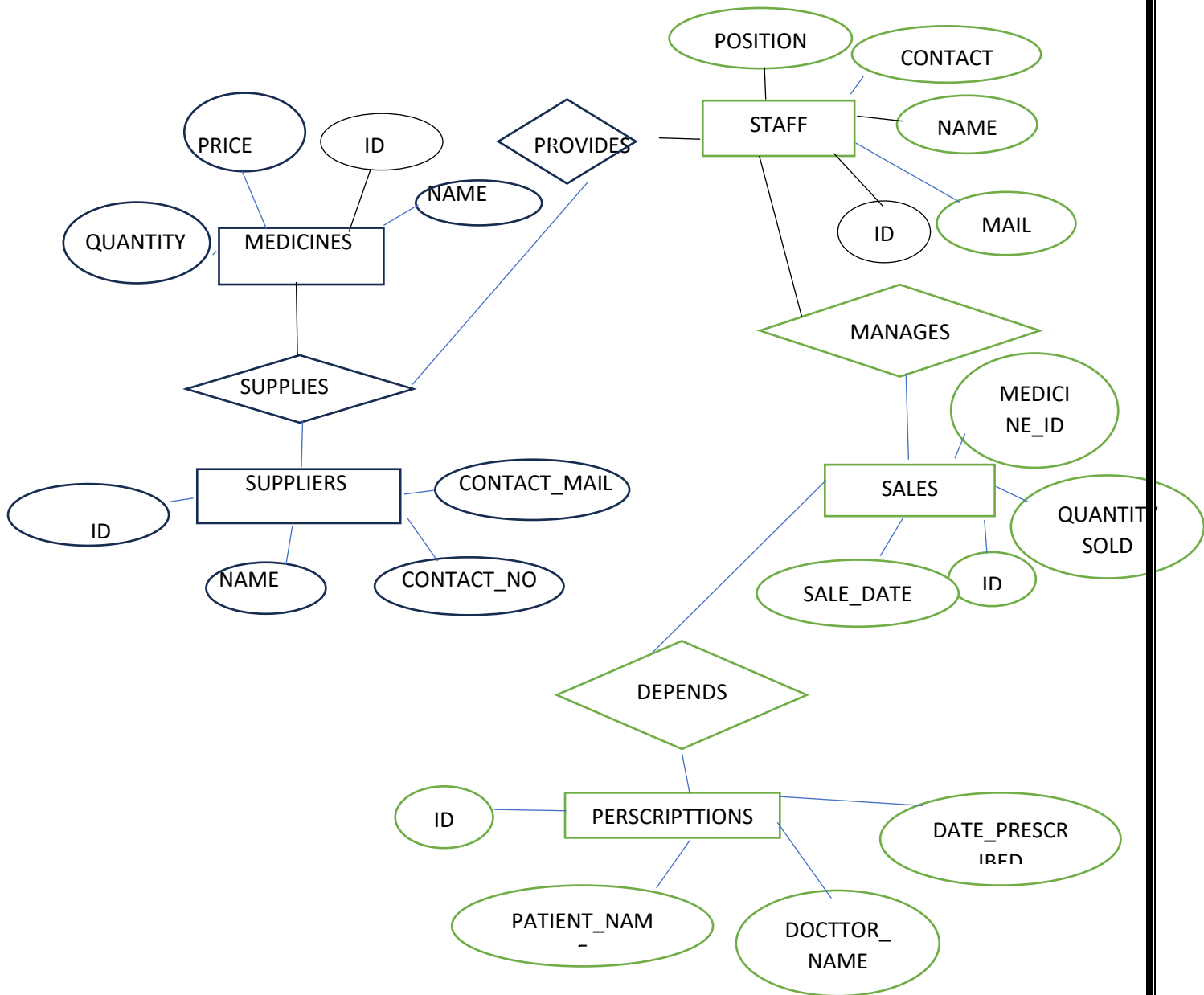


Fig 7.1 E-R Diagram

CHAPTER 8

SOURCE CODE

Program to create

```
import mysql.connector
# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host="localhost", # Replace with your MySQL server's host
    user="root", # Replace with your MySQL username
    password="mysql@2023", # Replace with your MySQL password
    database="pharmacy_db" # Specify the database you want to connect to
)

# Create a cursor object to execute SQL queries
cursor = connection.cursor()

# Function to create a new column in a table
def create_new_column():
    table_name = input("Enter the name of the table where you want to create a new column: ")
    new_column_name = input("Enter the name of the new column: ")
    new_column_data_type = input("Enter the data type of the new column (e.g., INT, VARCHAR(255), DECIMAL(10, 2)): ")

    # Check if the table exists
    cursor.execute("SHOW TABLES LIKE %s", (table_name,))
    table_exists = cursor.fetchone()

    if table_exists:
        # Create the new column
        alter_table_query = f"ALTER TABLE {table_name} ADD COLUMN {new_column_name} {new_column_data_type}"
        cursor.execute(alter_table_query)
        connection.commit()
        print(f"New column {new_column_name} with data type {new_column_data_type} added to table {table_name}.")
    else:
        print(f"Table {table_name} does not exist.")

    # Dictionary to handle different cases
    options = {
        '1': create_new_column,
    }

    while True:
        print("Choose an option:")
        print("1. Create a new column in a table")
        print("2. Exit")

        choice = input("Enter your choice: ")

        if choice == '2':
            break

        selected_option = options.get(choice)
        if selected_option:
            selected_option()
        else:
            print("Invalid choice. Please select a valid option.")

    # Close the cursor and connection when done
    cursor.close()
    connection.close()
```

Program to delete

```
import mysql.connector

# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host="localhost", # Replace with your MySQL server's host
    user="root", # Replace with your MySQL username
    password="mysql@2023", # Replace with your MySQL password
    database="pharmacy_db" # Specify the database you want to connect to
)
```

```

# Create a cursor object to execute SQL queries
cursor = connection.cursor()

# Function to delete data from the medicines table
def delete_medicine():
    medicine_id = int(input("Enter the ID of the medicine to delete: "))
    cursor.execute("SELECT * FROM medicines WHERE id = %s", (medicine_id,))
    existing_medicine = cursor.fetchone()

    if existing_medicine:
        delete_medicine_query = "DELETE FROM medicines WHERE id = %s"
        cursor.execute(delete_medicine_query, (medicine_id,))
        connection.commit()
        print("Medicine data deleted.")
    else:
        print("Medicine with ID", medicine_id, "does not exist.")

# Function to delete data from the sales table
def delete_sale():
    sale_id = int(input("Enter the ID of the sale to delete: "))

    cursor.execute("SELECT * FROM sales WHERE id = %s", (sale_id,))
    existing_sale = cursor.fetchone()

    if existing_sale:
        delete_sale_query = "DELETE FROM sales WHERE id = %s"
        cursor.execute(delete_sale_query, (sale_id,))
        connection.commit()
        print("Sale data deleted.")
    else:
        print("Sale with ID", sale_id, "does not exist.")

# Function to delete data from the suppliers table
def delete_supplier():
    supplier_id = int(input("Enter the ID of the supplier to delete: "))

    cursor.execute("SELECT * FROM suppliers WHERE id = %s", (supplier_id,))
    existing_supplier = cursor.fetchone()

    if existing_supplier:
        delete_supplier_query = "DELETE FROM suppliers WHERE id = %s"
        cursor.execute(delete_supplier_query, (supplier_id,))
        connection.commit()
        print("Supplier data deleted.")
    else:
        print("Supplier with ID", supplier_id, "does not exist.")

# Function to delete data from the prescriptions table
def delete_prescription():
    prescription_id = int(input("Enter the ID of the prescription to delete: "))

    cursor.execute("SELECT * FROM prescriptions WHERE id = %s", (prescription_id,))
    existing_prescription = cursor.fetchone()

    if existing_prescription:
        delete_prescription_query = "DELETE FROM prescriptions WHERE id = %s"
        cursor.execute(delete_prescription_query, (prescription_id,))
        connection.commit()
        print("Prescription data deleted.")
    else:
        print("Prescription with ID", prescription_id, "does not exist.")

# Function to delete data from the staff table
def delete_staff():
    staff_id = int(input("Enter the ID of the staff member to delete: "))

    cursor.execute("SELECT * FROM staff WHERE id = %s", (staff_id,))
    existing_staff = cursor.fetchone()

    if existing_staff:
        delete_staff_query = "DELETE FROM staff WHERE id = %s"
        cursor.execute(delete_staff_query, (staff_id,))
        connection.commit()
        print("Staff data deleted.")

```

```

else:
    print("staff with ID", staff_id, "does not exist.")

# Dictionary to handle different cases
options = {
    '1': delete_medicine,
    '2': delete_sale,
    '3': delete_supplier,
    '4': delete_prescription,
    '5': delete_staff
}

while True:
    print("Choose a table to delete data from:")
    print("1. Medicines")
    print("2. Sales")
    print("3. Suppliers")
    print("4. Prescriptions")
    print("5. Staff")
    print("6. Exit")

    choice = input("Enter your choice: ")

    if choice == '6':
        break

    selected_option = options.get(choice)
    if selected_option:
        selected_option()
    else:
        print("Invalid choice. Please select a valid option.")

# Close the cursor and connection when done
cursor.close()
connection.close()

```

Program to insert

```

import mysql.connector
# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host="localhost", # Replace with your MySQL server's host
    user="root", # Replace with your MySQL username
    password="mysql@2023", # Replace with your MySQL password
    database="pharmacy_db" # Specify the database you want to connect to
)
# Create a cursor object to execute SQL queries
cursor = connection.cursor()

# Function to insert data into the medicines table
def insert_medicine():
    medicine_name = input("Enter medicine name: ")
    medicine_price = float(input("Enter medicine price: "))
    medicine_quantity = int(input("Enter medicine quantity: "))

    new_medicine = (medicine_name, medicine_price, medicine_quantity)
    insert_medicine_query = "INSERT INTO medicines (name, price, quantity) VALUES (%s, %s, %s)"
    cursor.execute(insert_medicine_query, new_medicine)
    connection.commit()
    print("Medicine data inserted.")

# Function to insert data into the sales table
def insert_sale():
    medicine_id = int(input("Enter medicine ID for the sale: "))
    quantity_sold = int(input("Enter quantity sold: "))
    sale_date = input("Enter sale date (YYYY-MM-DD): ")

    new_sale = (medicine_id, quantity_sold, sale_date)
    insert_sale_query = "INSERT INTO sales (medicine_id, quantity_sold, sale_date) VALUES (%s, %s, %s)"
    cursor.execute(insert_sale_query, new_sale)
    connection.commit()
    print("Sale data inserted.")

# Function to insert data into the suppliers table
def insert_supplier():

```

```

supplier_name = input("Enter supplier name: ")
contact_name = input("Enter contact name: ")
contact_email = input("Enter contact email: ")

new_supplier = (supplier_name, contact_name, contact_email)
insert_supplier_query = "INSERT INTO suppliers (name, contact_name, contact_email) VALUES (%s, %s, %s)"
cursor.execute(insert_supplier_query, new_supplier)
connection.commit()
print("Supplier data inserted.")

# Function to insert data into the prescriptions table
def insert_prescription():
    patient_name = input("Enter patient name: ")
    doctor_name = input("Enter doctor name: ")
    date_prescribed = input("Enter date prescribed (YYYY-MM-DD): ")

    new_prescription = (patient_name, doctor_name, date_prescribed)
    insert_prescription_query = "INSERT INTO prescriptions (patient_name, doctor_name, date_prescribed) VALUES (%s, %s, %s)"
    cursor.execute(insert_prescription_query, new_prescription)
    connection.commit()
    print("Prescription data inserted.")

# Function to insert data into the staff table
def insert_staff():
    staff_name = input("Enter staff name: ")
    position = input("Enter position: ")
    contact_email = input("Enter contact email: ")

    new_staff = (staff_name, position, contact_email)
    insert_staff_query = "INSERT INTO staff (name, position, contact_email) VALUES (%s, %s, %s)"
    cursor.execute(insert_staff_query, new_staff)
    connection.commit()
    print("Staff data inserted.")

# Dictionary to handle different cases
options = {
    '1': insert_medicine,
    '2': insert_sale,
    '3': insert_supplier,
    '4': insert_prescription,
    '5': insert_staff
}

while True:
    print("Choose a table to insert data:")
    print("1. Medicines")
    print("2. Sales")
    print("3. Suppliers")
    print("4. Prescriptions")
    print("5. Staff")
    print("6. Exit")

    choice = input("Enter your choice: ")

    if choice == '6':
        break

    selected_option = options.get(choice)
    if selected_option:
        selected_option()
    else:
        print("Invalid choice. Please select a valid option.")

# Close the cursor and connection when done
cursor.close()
connection.close()

```

Main Program

```

while True:
    print("Choose an operation:")
    print("1. Insert")
    print("2. Delete")
    print("3. Update")

```

```

print("4. Read")
print("5. Create")
print("6. Exit")

choice = input("Enter your choice: ")
if choice == '1':
    import insert
elif choice == '2':
    import delete
elif choice == '3':
    import update
elif choice == '4':
    import read
elif choice == '5':
    import create
elif choice == '6':
    break;
else:
    print("Invalid choice. Please select a valid option.")

```

Program to read

```

import mysql.connector

# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host="localhost", # Replace with your MySQL server's host
    user="root", # Replace with your MySQL username
    password="mysql@2023", # Replace with your MySQL password
    database="pharmacy_db" # Specify the database you want to connect to
)

# Create a cursor object to execute SQL queries
cursor = connection.cursor()

# Function to read data from the medicines table
def read_medicine():
    cursor.execute("SELECT * FROM medicines")
    medicines = cursor.fetchall()

    if medicines:
        print("Medicines table:")
        for medicine in medicines:
            print(f" ID: {medicine[0]}\n Name: {medicine[1]}\n Price: {medicine[2]}\n Quantity: {medicine[3]}")
    else:
        print("Medicines table is empty.")

# Function to read data from the medicines table
def read_sale():
    cursor.execute("SELECT * FROM sales")
    sales = cursor.fetchall()

    if sales:
        print("sales table:")
        for sales in sales:
            print(f" ID: {sales[0]}\n medicine_id: {sales[1]}\n Price: {sales[2]}\n quantity_sold: {sales[3]}\n sale_date: {sales[4]}")
    else:
        print("sales table is empty.")

# Function to read data from the suppliers table
def read_suppliers():
    cursor.execute("SELECT * FROM suppliers")
    suppliers = cursor.fetchall()

    if suppliers:
        print("suppliers table:")
        for suppliers in suppliers:
            print(f" ID: {suppliers[0]}\n contact_name: {suppliers[1]}\n contact_email: {suppliers[2]}")
    else:
        print("suppliers table is empty.")

# Function to read data from the prescriptions table
def read_prescriptions():
    cursor.execute("SELECT * FROM prescriptions")
    prescriptions = cursor.fetchall()

    if prescriptions:

```

```

        print("prescriptions table:")
        for prescriptions in prescriptions:
            print(f" ID: {prescriptions[0]}\n patient_name: {prescriptions[1]}\n doctor_name: {prescriptions[2]}\n date_prescribed: {prescriptions[3]}")
        else:
            print("prescriptions table is empty.")

# Function to read data from the prescriptions table
def read_staff():
    cursor.execute("SELECT * FROM staff")
    staff = cursor.fetchall()

    if staff:
        print("staff table:")
        for staff in staff:
            print(f" ID: {staff[0]}\n name: {staff[1]}\n position: {staff[2]}\n contact_email: {staff[3]}")
    else:
        print("staff table is empty.")

# Dictionary to handle different cases
options = {
    '1': read_medicine,
    '2': read_sale,
    '3': read_suppliers,
    '4': read_prescriptions,
    '5': read_staff
}

while True:
    print("Choose a table to delete data from:")
    print("1. Medicines")
    print("2. Sales")
    print("3. Suppliers")
    print("4. Prescriptions")
    print("5. Staff")
    print("6. Exit")

    choice = input("Enter your choice: ")

    if choice == '6':
        break

    selected_option = options.get(choice)
    if selected_option:
        selected_option()
    else:
        print("Invalid choice. Please select a valid option.")

# Close the cursor and connection when done
cursor.close()
connection.close()

```

Program to Update

```

import mysql.connector

# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host="localhost", # Replace with your MySQL server's host
    user="root", # Replace with your MySQL username
    password="mysql@2023", # Replace with your MySQL password
    database="pharmacy_db" # Specify the database you want to connect to
)

# Create a cursor object to execute SQL queries
cursor = connection.cursor()

# Function to update data in the medicines table
def update_medicine():
    medicine_id = int(input("Enter the ID of the medicine to update: "))
    column_to_update = input("Enter the column to update (e.g., name, price, quantity): ").strip()
    new_value = input(f"Enter the new value for {column_to_update}: ")

    # Check if the medicine with the given ID exists
    cursor.execute("SELECT * FROM medicines WHERE id = %s", (medicine_id,))
    existing_medicine = cursor.fetchone()

```



```

if existing_medicine:
    update_medicine_query = f"UPDATE medicines SET {column_to_update} = %s WHERE id = %s"
    cursor.execute(update_medicine_query, (new_value, medicine_id))
    connection.commit()
    print("Medicine data updated.")
else:
    print("Medicine with ID", medicine_id, "does not exist.")

# Function to update data from the sales table
def update_sale():
    sale_id = int(input("Enter the ID of the sale to update: "))
    column_to_update1 = input("Enter the column to update (e.g., id, medicine_id, quantity_sold,sale_date): ").strip()
    new_value1 = input(f"Enter the new value for {column_to_update1}: ")

    # Check if the sale with the given ID exists
    cursor.execute("SELECT * FROM sales WHERE id = %s", (sale_id,))
    existing_sale = cursor.fetchone()

    if existing_sale:
        update_sale_query = f"UPDATE sales SET {column_to_update1} = %s WHERE id = %s"
        cursor.execute(update_sale_query, (new_value1, sale_id))
        connection.commit()
        print("Sale data updated.")
    else:
        print("Sale with ID", sale_id, "does not exist.")

# Function to update data from the suppliers table
def update_supplier():
    supplier_id = int(input("Enter the ID of the supplier to update: "))
    column_to_update2 = input("Enter the column to update (e.g., id, contact_name, contact_email): ").strip()
    new_value2 = input(f"Enter the new value for {column_to_update2}: ")

    # Check if the sale with the given ID exists

    cursor.execute("SELECT * FROM suppliers WHERE id = %s", (supplier_id,))
    existing_supplier = cursor.fetchone()

    if existing_supplier:
        update_supplier_query = f"UPDATE suppliers SET {column_to_update2} = %s WHERE id = %s"
        cursor.execute(update_supplier_query, (new_value2,supplier_id))
        connection.commit()
        print("Supplier data updated.")
    else:
        print("Supplier with ID", supplier_id, "does not exist.")

# Function to update data from the prescriptions table
def update_prescription():
    prescription_id = int(input("Enter the ID of the prescription to update: "))
    column_to_update3 = input("Enter the column to update (e.g., id, patient_name, doctor_name,date_prescribed): ").strip()
    new_value3= input(f"Enter the new value for {column_to_update3}: ")

    # Check if the sale with the given ID exists
    cursor.execute("SELECT * FROM prescriptions WHERE id = %s", (prescription_id,))
    existing_prescription = cursor.fetchone()

    if existing_prescription:
        update_prescription_query = f"UPDATE prescriptions SET {column_to_update3} = %s WHERE id = %s"
        cursor.execute(update_prescription_query, (new_value3,prescription_id))
        connection.commit()
        print("Prescription data updated.")
    else:
        print("Prescription with ID", prescription_id, "does not exist.")

# Function to delete data from the staff table
def update_staff():
    staff_id = int(input("Enter the ID of the staff member to update: "))
    column_to_update4 = input("Enter the column to update (e.g., id, name, position,contact_email): ").strip()
    new_value4= input(f"Enter the new value for {column_to_update4}: ")

    # Check if the sale with the given ID exists
    cursor.execute("SELECT * FROM staff WHERE id = %s", (staff_id,))
    existing_staff = cursor.fetchone()

    if existing_staff:

```

```
update_staff_query = f"UPDATE staff SET {column_to_update4} = %s WHERE id = %s"
cursor.execute(update_staff_query, (new_value4, staff_id))
connection.commit()
print("Staff data updated.")
else:
    print("staff with ID", staff_id, "does not exist.")

# Dictionary to handle different cases
options = {
    '1': update_medicine,
    '2': update_sale,
    '3': update_supplier,
    '4': update_prescription,
    '5': update_staff
}

while True:
    print("Choose a table to update data from:")
    print("1. Medicines")
    print("2. Sales")
    print("3. Suppliers")
    print("4. Prescriptions")
    print("5. Staff")
    print("6. Exit")

    choice = input("Enter your choice: ")

    if choice == '6':
        break

    selected_option = options.get(choice)
    if selected_option:
        selected_option()
    else:
        print("Invalid choice. Please select a valid option.")

# Close the cursor and connection when done
cursor.close()
connection.close()
```

CHAPTER 9

RESULTS

Choose an operation:

1. Insert
2. Delete
3. Update
4. Read
5. Create
6. Exit

Enter your choice:

Enter your choice: 1

Choose a table to insert data:

1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit

Enter your choice: 1

Enter medicine name: omiprazol

Enter medicine price: 99

Enter medicine quantity: 9

Medicine data inserted.

Choose a table to insert data:

1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit

Enter your choice:

Result Grid				
Filter Rows: <input type="text"/>				
	id	name	price	quantity
▶	1	paracetamol	77.00	30
	2	Dolo 650	10.00	30
	3	Azithromycin	10.00	30
	4	Cipla	10.00	30
	5	Omee	10.00	30
	7	omiprazol	99.00	9

medicines 41 x

Fig 12.1

```

Enter your choice: 2
Choose an operation:
1. Insert
2. Delete
3. Update
4. Read
5. Create
6. Exit
Enter your choice: 2
Choose a table to delete data from:
1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit
Enter your choice: 1
Enter the ID of the medicine to delete: 7
Medicine with ID 7 deleted.

```

Result Grid				
Filter Rows: <input type="text"/>				
	id	name	price	quantity
▶	1	paracetamol	77.00	30
▼	2	Dolo 650	10.00	30
⋮	3	Azithromycin	10.00	30
	4	Cipla	10.00	30
	5	Omee	10.00	30
*	NULL	NULL	NULL	NULL

medicines 42 x

Fig 12.2

```

Choose an operation:
1. Insert
2. Delete
3. Update
4. Read
5. Create
6. Exit
Enter your choice: 3
Choose a table to update data from:
1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit
Enter your choice: 3
Enter the ID of the supplier to update: 5
Enter the column to update (e.g., id, contact_name, contact_email): name
Enter the new value for name: gayatri
Supplier data updated.
Choose a table to update data from:
1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit
Enter your choice: _

```

Result Grid				
Filter Rows: <input type="text"/>				
	id	name	contact_name	contact_email
▶	1	Paracetamol	Betty	betty@gmail.com
	2	Dolo 650	Archie	arch@gmail.com
	3	Azithromycin	Jones	jones@gmail.com
	4	Cipla	Kevin	kev@gmail.com
	5	gayatri	Jane	jane@gmail.com
*	NULL	NULL	NULL	NULL

Fig 12.3

```
Choose a table to read data from:
```

1. Medicines
2. Sales
3. Suppliers
4. Prescriptions
5. Staff
6. Exit

```
Enter your choice: 4
```

```
prescriptions table:
```

```
ID: 1
patient_name: Mike
doctor_name: eleven
date_prescribed: 2023-10-04
ID: 2
patient_name: Steve
doctor_name: Errica
date_prescribed: 2023-09-23
ID: 3
patient_name: Nancy
doctor_name: Martin
date_prescribed: 2023-04-14
ID: 4
patient_name: Robin
doctor_name: Jessica
date_prescribed: 2023-05-24
ID: 5
patient_name: Dustin
doctor_name: Suzy
date_prescribed: 2023-10-22
```

```
% C:\WINDOWS\py.exe
```

```
Choose an operation:
```

1. Insert
2. Delete
3. Update
4. Read
5. Create
6. Exit

```
Enter your choice: 5
```

```
Choose an option:
```

1. Create a new column in a table
2. Exit

```
Enter your choice: 1
```

```
Enter the name of the table where you want to create a new column: prescriptions
```






```
Enter the name of the new column: medicines
```

```
Enter the data type of the new column (e.g., INT, VARCHAR(255), DECIMAL(10, 2)): varchar(20)
New column medicines with data type varchar(20) added to table prescriptions.
```

```
Choose an option:
```

1. Create a new column in a table
2. Exit

```
Enter your choice:
```

Result Grid   Filter Rows: Edit:    E

	id	patient_name	doctor_name	date_prescribed	medicines
▶	1	Mike	eleven	2023-10-04	NULL
	2	Steve	Errica	2023-09-23	NULL
	3	Nancy	Martin	2023-04-14	NULL
	4	Robin	Jessica	2023-05-24	NULL
	5	Dustin	Suzy	2023-10-22	NULL
*	NULL	NULL	NULL	NULL	NULL

prescriptions46 x

Fig 12.4

CHAPTER 10

CONCLUSION

SQL database management application which is very well used in the modern world in organising and manipulating a database. Through SQL doesn't have GUI interface like Microsoft Access is having and they all manage the database comfortably. Depending on the use of users, if an organisation has multiple users then they should go for SQL server based application. This project show how to create tables SQL and how to create a simple data manipulation language and data definition language and also how to execute them. It also shows how relationships are established with concept of primary and Foreign key within a table. Lastly, the project shows how queries are created in SQL server, queries like Insert, Delete, Update and view, etc .One can create many test cases for each text file and each option's button. It can develop the scenarios which can help to form as many test cases as possible.

CHAPTER

REFERENCES

1. https://en.wikipedia.org/wiki/Pharmacy_management_system#:~:text=The%20pharmacy%20management%20system%2C%20also,medication%20use%20process%20with%20pharmacies.

2. <https://arirms.com/features-of-pharmacy-management-system>
3. <https://www.linkedin.com/pulse/benefits-pharmacy-management-system-pharma-industry-revolution/>
4. <https://tateeda.com/blog/pharmacy-management-software-development>
- 5.