VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnanasangama," Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V.Puram , Bangalore-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DATABASE MANAGEMENT SYSTEM MINI PROJECT

18CSL58

"PETROL PUMP MANAGEMENT"

Submitted By

Divyansh Nama

1BI20CS060

for the academic year 2022-23

Department of Computer Science & Engineering

Bangalore Institute of Technology

K.R. Road, V.V.Puram, Bangalore-560 002

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnanasangama,” Belagavi-590018, Karnataka

BANGALOREINSTITUTEOFTECHNOLOGY
K.R.Road,V.V.Puram, Bangalore-560004

DepartmentofComputerScience&Engineering

Certificate

This is to certify that the implementation of **DBMS MINI PROJECT** entitled “**TIME TABLE GENERATOR**”has been successfully completed by

USN	**NAME**
1BI20CS060	**Divyansh Nama**

of V semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in Computer Science & Engineering of the Visvesvaraya Technological University during the academic year 2021-2022.

LabIn-charge:

Prof. Pratima Mg	**Dr.J.Girija**
Assistant Professor Dept. of CS&E Bangalore Institute of Technology Technology Bangalore	Professor and Head Department of CS&E Bangalore Institute of Bangalore

Examiners: 1) 2)

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my thanks **Dr. J.Girija.** HoD, Dept. Of CS&E for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the Faculty in-charge **Prof. Prathima Mg** Assistant Professor, Dept. Of CS&E on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I am very pleasured ton express my sincere gratitude to the friendly-operation showed by all the members of **Computer Science & Engineering Department, BIT.**

Divyansh Nama

1BI20CS060

CONTENTS

LISTOF FIGURES

CHAPTER 1

INTRODUCTION

INTRODUCTION

1. Overview :

The petrol pump management system helps to manage all the employees , owners, inventory and other resources in a very effective manner.

With an effective Petrol pump management system we can keep a track of the availability of fuel and thereby reducing the risk of theft and dishonesty also the logging is only with the admin so the database is very secure.

Billing and accounting becomes a systematic task and updation of new records or deletion of old ones is also very easily managed.

In our Petrol Pump management system we have implemented a query option where we can directly type any SQL command and get the output.

2. Problem Statement :

The project deals with following concerns:

- Generating and Updating information by just a click of a button, by authorized person.
- Viewing the updated info of employees, owner, stations ,tanker, invoice etc.

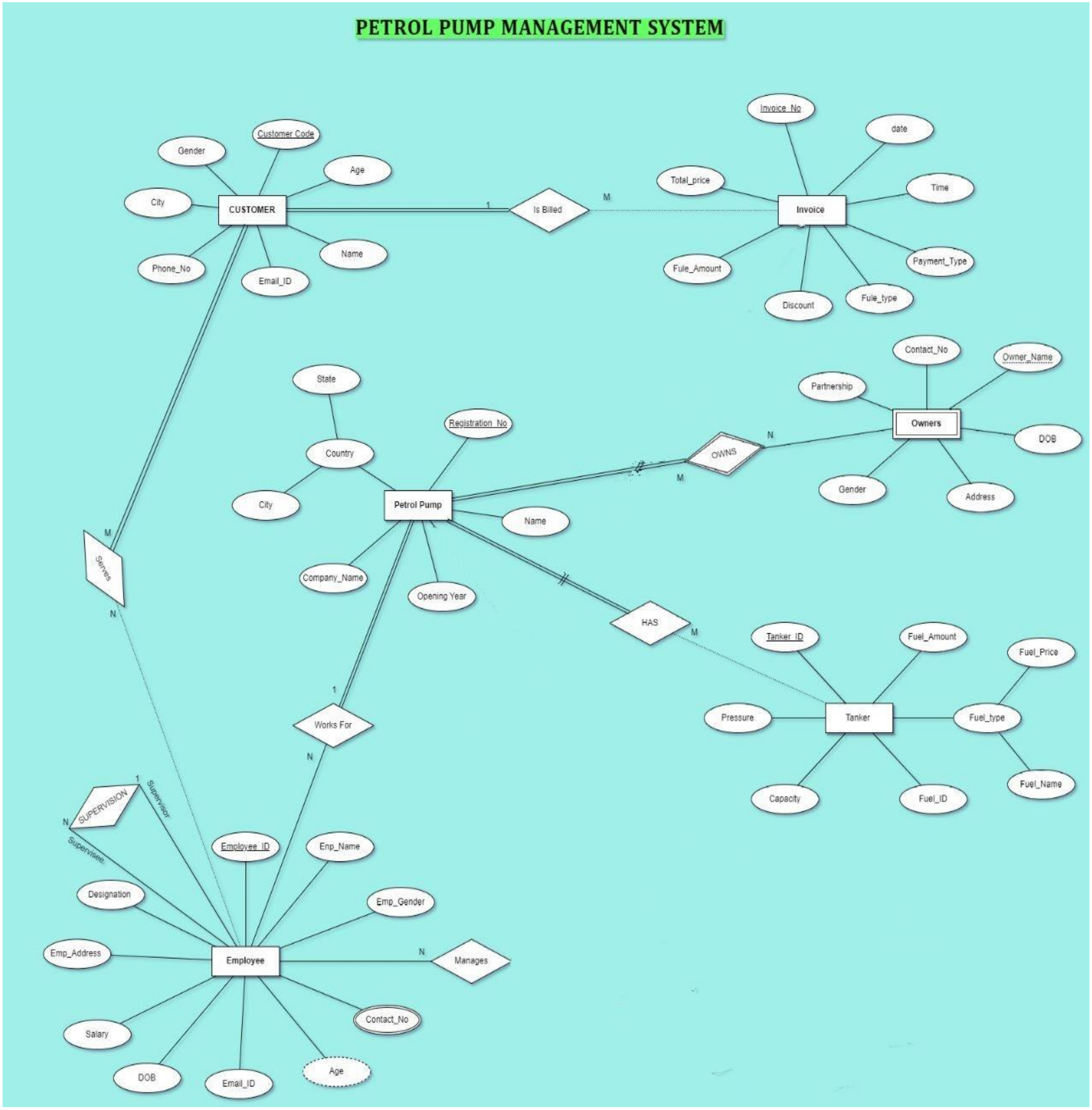
CHAPTER 2

BACK END DESIGN

2.1 CONCEPTUAL DATABASE DIAGRAM (ER DIAGRAM)

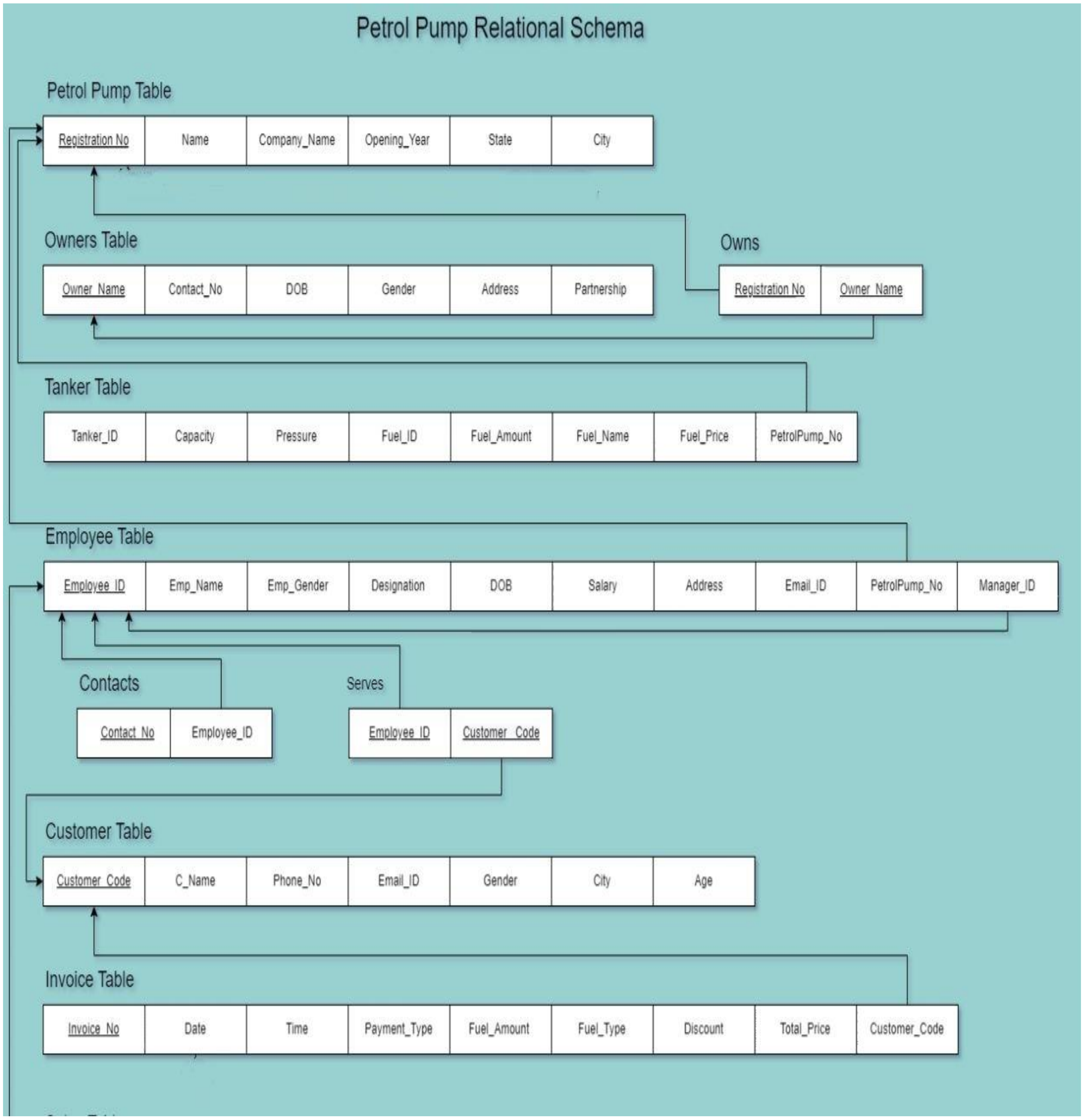
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

Fig 2.1: E R Diagram for Food Ordering System



2.2 Relational Schema:

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model.
A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram.



2.3 NORMALIZATION :

1. Petrol_Pump Table

Registration_No	Name	Company_Name	Opening_Year	State	City

Functional Depedencies –

Registration_No→ {Name,Company_Name,Opening_Year,State,City}

Candidate Key =Registration_No

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

2. Owners Table

Owner_Name	Contact_No	DOB	Gender	Address	Partnership

Functional Depedencies –

Owner_Name→ {Contact_No,DOB,Gender,Address,Partnership}

Candidate Key =Owner_Name

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.

- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

3. Tanker_Table

Tanker_ID	Capacity	Pressure	Fuel_ID	Fuel_Amount	Fuel_Name	Fuel_Price	Petrol_Pump_No

Functional Depedencies –

All are independent, as this is used to give structure to the Petrol_Pump management

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

4. Employee Table

Employee_ID	Emp_Name	Emp_Gender	Designation	DOB	Salary	Address	Email_ID	PetrolPump_no	Manage_ID

Functional Dependencies –

Employee_ID→Emp_Name,Emp_Gender,esignation,DOB,Salary,Address,Email_ID,PetrolPump_No, Manager_ID

Candidate Key =Employee_ID

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

5. **Customer Table**

<u>Customer_Code</u>	C_Name	Phone_No	Email_ID	Gender	City	Age

Functional Depedencies –

Customer_Code→ {C_Name,Phone_No,Email_ID,Gender,City,Age}

Candidate Key =Customer_Code

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

6. **Contacts**

<u>Conatct_No</u>	Employee_ID

Functional Depedencies –

Contact_No→Employee_ID

Candidate Key =Contact_No

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

7. **Serves**

<u>Employee_ID</u>	<u>Customer_Code</u>

Functional Depedencies –

Both are Candidate Key ,so both can derive each other

Candidate_Key =Employee_ID,Customer_Code

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

8. **Owns**

<u>Registration_No</u>	<u>Owner_Name</u>

Functional Depedencies –

Both are Candidate Key ,so both can derive each other

Candidate Key =Registration_No, Owner_Name

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

9. Invoice Table

Invoice_No	Date	Time	Payment_Type	Fuel_Amount	Fuel_Type	Discount	Total_Price	Customer_Code

Functional Depedencies –

Invoice_No→ {Date,Time,Payment_Type,Fuel_Amount,Fuel_Type,Discount,Total_Price,Customer_Code}

Candidate Key =Invoice_No

Justification –

- Here , in the relation above, all the attributes are indivisible and atomic in nature , thus it is in 1NF form.
- All the non-prime attributes are fully functionally dependent on the prime attributes and there is no partial dependency, this it is in 2NF form.
- Since there exists no transitive dependency in the relation between prime and on-prime attributes , so the relation is in 3NF form.

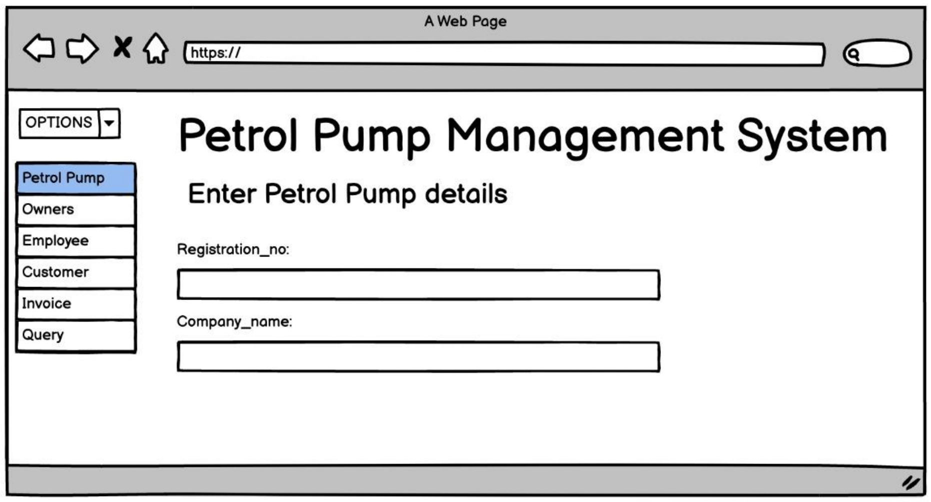
CHAPTER 3

FRONT END DESIGN

FrontEndDesign

3.1 Screen Layout Design:

One of the most important for any application is the design of the graphical user interface (GUI) and the layout of the screen.



The first login page is divided into sections as shown below: - Title , Navigation ,Logo and space for entering employee credentials.

Title Text	
Side bar with crud operations	Space for implementation of buttons along with their description.

3.2 StreamLit

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps.

3.3 MYSQL.connector:Front end and back end connector

MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specification v2.0 (PEP 249).

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="30Divyansh@"
)
```

3.4 Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

1. System Requirements and Specifications :

Software requirements

Minimum software requirements are:

- Tool: SQL Server
- Operating System :WindowsXP/7,8,10
- Scripting Language :Python

Hardwarerequirements

Minimum Hardware requirements are:

- Processor: PentiumIV
- Ram:1GBRAM
- HardDisk:20GB

CHAPTER 4

MAJOR MODULES

4.1 Login Module :

Here the person intending to use the service has to login in using an appropriate username and password. This is the basis step as further on services accessible by the user depends on this only.

4.2 Menu Module

This module has basically divided into section

A. Tables

B. CRUD operations

C. Query

A. The tables has -

1. **Petrol Pump Management** –This section is about petrol pump details.
2. **Employee** – On clicking this module we get a employee details form
3. **Customer** – On clicking it prompts a form for customer details.
4. **Invoice** – On clicking it prompts a bill for customer.
5. **Tanker** – On clicking it prompts a form where admin can fill the details of tanker.

B. CRUD Operations

All the above five has these given below

1. **Add –** In this we can add details of add petrol pump , add employee etc.
2. **View –** All the details that we added we can view by this section.
3. **Update-** In this section we can update details of added data
4. **Remove-**From this section we can remove any added details.

C. Query

1.**Custom query-** In this section we can execute any MYSQL query

2.**Function-**In this section when we put any tanker ID and Run function it shows a total amount.

CHAPTER 5

IMPLEMENTATION

5.1 Database - Creating Tables (MySQL):/Backend

```
import mysql.connector

mydb = mysql.connector.connect(

  host="localhost",

  user="root",

  password="",

  database="Petrolpump_Management"

)


c = mydb.cursor()


def create_table():

    c.execute('CREATE TABLE IF NOT EXISTS Petrolpump (Registration_No varchar(10) NOT NULL, Petrolpump_Name varchar(50) NOT NULL,  Company_Name varchar(30) DEFAULT NULL, Opening_Year int(5) DEFAULT NULL, State varchar(30) DEFAULT NULL,City varchar(40) NOT NULL, PRIMARY KEY(Registration_No))')

    c.execute('CREATE TABLE IF NOT EXISTS Owners( Owner_Name varchar(20) NOT NULL, Contact_NO char(10) NOT NULL, DOB date DEFAULT NULL, Gender char DEFAULT NULL, Address varchar(255) DEFAULT NULL, Partnership int(5) NOT NULL, PRIMARY KEY(Owner_Name))')

    c.execute('CREATE TABLE IF NOT EXISTS Employee( Employee_ID varchar(10) NOT NULL, Emp_Name varchar(30) NOT NULL,  Emp_Gender char DEFAULT NULL,  Designation varchar(10) DEFAULT NULL, DOB date DEFAULT NULL,  Salary int(20) DEFAULT NULL, Emp_Address varchar(255) NOT NULL, Email_ID varchar(100) NOT NULL, Petrolpump_No varchar(10) DEFAULT NULL, Manager_ID varchar(10) DEFAULT NULL, PRIMARY KEY(Employee_ID)) ')

    c.execute('CREATE TABLE IF NOT EXISTS Customer( Customer_Code varchar(10) NOT NULL, C_Name varchar(30) NOT NULL, Phone_No char(10) DEFAULT NULL,  Email_ID varchar(100) DEFAULT NULL, Gender char DEFAULT NULL,  City varchar(50) DEFAULT NULL, Age int(3) DEFAULT NULL, PRIMARY KEY(Customer_Code))')

    c.execute('CREATE TABLE IF NOT EXISTS Invoice( Invoice_No varchar(10) NOT NULL, Date date NOT NULL, Payment_Type varchar(20) NOT NULL, Fuel_Amount float(15) DEFAULT NULL, Fuel_Type varchar(15) DEFAULT NULL, Discount int(5) DEFAULT NULL, Total_Price float(10) NOT NULL, Customer_Code varchar(10) NULL, PRIMARY KEY(Invoice_No))')

    c.execute('CREATE TABLE IF NOT EXISTS Tanker( Tanker_ID varchar(10) NOT NULL, Capacity float(10) DEFAULT NULL,  pressure float(10) DEFAULT NULL,  Fuel_ID varchar(10) NOT NULL, Fuel_Amount float(15) DEFAULT NULL, Fuel_Name varchar(20) DEFAULT NULL, Fuel_Price float(5) NOT NULL, Petrolpump_No varchar(10) DEFAULT NULL, PRIMARY KEY(Tanker_ID))')
```

```
def add_Petrolpump_data(Registration_No,Petrolpump_Name,Company_Name,Opening_Year,State,City):

c.execute('insert into Petrolpump (Registration_No,Petrolpump_Name,Company_Name,Opening_Year,State,City) values (%s,%s,%s,%s,%s,%s)',(Registration_No,Petrolpump_Name,Company_Name,Opening_Year,State,City) )

mydb.commit()


def add_Owners_data(Owner_Name, Contact_NO, DOB, Gender, Address, Partnership):

c.execute('INSERT INTO Owners  (Owner_Name, Contact_NO, DOB, Gender, Address, Partnership) VALUES (%s,%s,%s,%s,%s,%s)',(Owner_Name, Contact_NO, DOB, Gender, Address, Partnership))

mydb.commit()


def add_Employee_data(Employee_ID, Emp_Name,  Emp_Gender,   Designation,  DOB, Salary, Emp_Address, Email_ID , Petrolpump_No, Manager_ID):

c.execute('insert into Employee  (Employee_ID, Emp_Name,  Emp_Gender,   Designation,  DOB, Salary, Emp_Address, Email_ID , Petrolpump_No, Manager_ID) values (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)',(Employee_ID, Emp_Name,  Emp_Gender,   Designation,  DOB, Salary,  Emp_Address, Email_ID , Petrolpump_No, Manager_ID))

mydb.commit()


def add_Customer_data(Customer_Code, C_Name , Phone_No  , Email_ID , Gender,  City , Age):

c.execute('INSERT INTO Customer  (Customer_Code , C_Name , Phone_No  , Email_ID , Gender,  City , Age) VALUES (%s,%s,%s,%s,%s,%s,%s)',(Customer_Code , C_Name , Phone_No  , Email_ID , Gender,  City , Age))

mydb.commit()


def add_Invoice_data(Invoice_No , Date  , Payment_Type , Fuel_Amount , Fuel_Type , Discount  , Total_Price , Customer_Code):

c.execute('INSERT INTO Invoice  (Invoice_No , Date  , Payment_Type , Fuel_Amount , Fuel_Type , Discount , Total_Price , Customer_Code) Values (%s,%s,%s,%s,%s,%s,%s,%s)',(Invoice_No , Date  , Payment_Type , Fuel_Amount , Fuel_Type , Discount  , Total_Price , Customer_Code))

mydb.commit()


def add_Tanker_data(Tanker_ID  , Capacity,  pressure,  Fuel_ID , Fuel_Amount, Fuel_Name , Fuel_Price , Petrolpump_No):
```

```
c.execute('INSERT INTO Tanker  (Tanker_ID  , Capacity,  pressure,  Fuel_ID , Fuel_Amount, Fuel_Name , Fuel_Price , Petrolpump_No) Values (%s,%s,%s,%s,%s,%s,%s)',(Tanker_ID  , Capacity,  pressure,  Fuel_ID , Fuel_Amount, Fuel_Name , Fuel_Price , Petrolpump_No))

mydb.commit()



def view_all_Petrolpump_data():

c.execute('SELECT * FROM Petrolpump')

    data = c.fetchall()

    return data


def view_all_Owners_data():

c.execute('SELECT * FROM Owners')

    data = c.fetchall()

    return data


def view_all_Employee_data():

c.execute('SELECT * FROM Employee')

    data = c.fetchall()

    return data


def view_all_Customer_data():

c.execute('SELECT * FROM Customer')

    data = c.fetchall()

    return data


def view_all_Invoice_data():

c.execute('SELECT * FROM Invoice')

    data = c.fetchall()

    return data
```

```
def view_all_Tanker_data():

c.execute('SELECT * FROM Tanker')

    data = c.fetchall()

    return data



def view_only_Registration_No():

c.execute("select Registration_No from Petrolpump")

    data = c.fetchall()

    return data


def view_only_Owner_Name():

c.execute("select Owner_Name from Owners")

    data = c.fetchall()

    return data


def view_only_Employee_ID():

c.execute("select Employee_ID from Employee")

    data = c.fetchall()

    return data


def view_only_Customer_Code():

c.execute("select Customer_Code from Customer")

    data = c.fetchall()

    return data


def view_only_Invoice_No():

c.execute("select Invoice_No from Invoice")

    data = c.fetchall()
```

```
  return data


def view_only_Tanker_ID():
c.execute("select Tanker_ID from Tanker")
    data = c.fetchall()
    return data



def get_all_info_Petrolpump(selected_Petrolpump):
c.execute('select * from Petrolpump where Registration_No="{}"'.format(selected_Petrolpump))
    data = c.fetchall()
    return data


def get_all_info_Owners(selected_Owners):
c.execute('select * from  Owners where Owner_Name="{}"'.format(selected_Owners))
    data = c.fetchall()
    return data


def get_all_info_Employee(selected_Employee):
c.execute('select * from Employee where Employee_ID="{}"'.format(selected_Employee))
    data = c.fetchall()
    return data


def get_all_info_Customer(selected_Customer):
c.execute('select * from Customer where Customer_Code="{}"'.format(selected_Customer))
    data = c.fetchall()
    return data


def get_all_info_Invoice(selected_Invoice):
```

```
c.execute('select * from Invoice where Invoice_No="{}"'.format(selected_Invoice))

 data = c.fetchall()
    return data

def get_all_info_Tanker(selected_Tanker):
c.execute('select * from Tanker where Tanker_ID="{}"'.format(selected_Tanker))
    data = c.fetchall()
    return data

def edit_Petrolpump_data(new_Petrolpump_Name, new_Company_Name, new_Opening_Year, new_State,
new_City, Registration_No):
c.execute("update Petrolpump set
Petrolpump_Name=%s,Company_Name=%s,Opening_Year=%s,State=%s,City=%s where Registration_No=%s",
(new_Petrolpump_Name, new_Company_Name, new_Opening_Year, new_State, new_City, Registration_No))
mydb.commit()
    data = view_all_Petrolpump_data()
    return data

def edit_Owners_data(new_Contact_NO, new_DOB, new_Gender,new_Address, new_Partnership,
Owner_Name):
c.execute("update Owners set Contact_NO=%s, DOB=%s, Gender=%s, Address=%s, Partnership=%s where
Owner_Name=%s", (new_Contact_NO, new_DOB, new_Gender,new_Address, new_Partnership, Owner_Name))
mydb.commit()
    data = view_all_Owners_data()
    return data

def edit_Employee_data(new_Emp_Name, new_Emp_Gender,  new_Designation, new_DOB,new_Salary,
new_Emp_Address, new_Email_ID , new_Petrolpump_No, new_Manager_ID, Employee_ID):
c.execute("update Employee set Emp_Name=%s,  Emp_Gender=%s,   Designation=%s,  DOB=%s, Salary=%s,
Emp_Address=%s, Email_ID=%s, Petrolpump_No=%s, Manager_ID=%s where Employee_ID=%s",
        (new_Emp_Name, new_Emp_Gender, new_Designation, new_DOB,new_Salary, new_Emp_Address,
new_Email_ID ,new_Petrolpump_No, new_Manager_ID, Employee_ID))

        mydb.commit()

          data = view_all_Employee_data()

          return data



        def edit_Customer_data(new_C_Name , new_Phone_No  , new_Email_ID , new_Gender,  new_City ,
new_Age, Customer_Code):

        c.execute("update Customer set C_Name=%s , Phone_No=%s , Email_ID=%s , Gender=%s,  City=%s ,
Age=%s where Customer_Code=%s", (new_C_Name , new_Phone_No , new_Email_ID , new_Gender, new_City ,
new_Age, Customer_Code))

        mydb.commit()

          data = view_all_Customer_data()

          return data




        def edit_Invoice_data( new_Date , new_Payment_Type , new_Fuel_Amount , new_Fuel_Type , new_Discount
,new_Total_Price , new_Customer_Code, Invoice_No):

        c.execute("update Invoice set Date=%s ,  Payment_Type=%s , Fuel_Amount=%s , Fuel_Type=%s ,
Discount=%s  , Total_Price=%s , Customer_Code=%s where Invoice_No=%s", (new_Date , new_Payment_Type ,
new_Fuel_Amount , new_Fuel_Type , new_Discount  ,new_Total_Price , new_Customer_Code, Invoice_No))

        mydb.commit()

          data = view_all_Invoice_data()

          return data



        def edit_Tanker_data(new_Capacity,  new_pressure,  new_Fuel_ID , new_Fuel_Amount, new_Fuel_Name ,
new_Fuel_Price ,new_Petrolpump_No, Tanker_ID):

        c.execute("update Tanker set Capacity=%s,  pressure=%s,  Fuel_ID=%s, Fuel_Amount=%s, Fuel_Name=%s,
Fuel_Price=%s, Petrolpump_No=%s where Tanker_ID=%s", (new_Capacity,  new_pressure,  new_Fuel_ID ,
new_Fuel_Amount, new_Fuel_Name , new_Fuel_Price ,new_Petrolpump_No, Tanker_ID))

        mydb.commit()

          data = view_all_Tanker_data()

          return data
```

```
def delete_data_Petrolpump(selected_Petrolpump):

c.execute('DELETE FROM Petrolpump WHERE Registration_No="{}"'.format(selected_Petrolpump))

mydb.commit()


def delete_data_Owners(selected_Owners):

c.execute('DELETE FROM Owners WHERE Owner_Name="{}"'.format(selected_Owners))

mydb.commit()


def delete_data_Employee(selected_Employee):

c.execute('DELETE FROM Employee WHERE Employee_ID="{}"'.format(selected_Employee))

mydb.commit()


def delete_data_Customer(selected_Customer):

c.execute('DELETE FROM Customer WHERE Customer_Code="{}"'.format(selected_Customer))


mydb.commit()


def delete_data_Invoice(selected_Invoice):

c.execute('DELETE FROM Invoice WHERE Invoice_No="{}"'.format(selected_Invoice))

mydb.commit()


def delete_data_Tanker(selected_Tanker):

c.execute('DELETE FROM Tanker WHERE Tanker_ID="{}"'.format(selected_Tanker))

mydb.commit()



def TOTAL_Amount(tanker_id):

    query = "SET @p0='{}';".format(tanker_id)
```

```
        c.execute(query)
            print(query)


            query = "SELECT `TOTAL_AMOUNT`(@p0) AS `TOTAL_AMOUNT`;"


        c.execute(query)
            print(query)
            result = c.fetchall()
            print(result)
            return result
```

5.2 Front-end Modules :

```
import streamlit as st
import mysql.connector
import pandas as pd


from create import *
from database import *
from delete import *
from read import *
from update import *


def main():
    st.title("Petrol Pump Management System")
    menu = ["PetrolPump", "Owners", "Employee", "Customer","Invoice", "Tanker","Query"]
    choice = st.sidebar.selectbox("Tables", menu)
```

```
create_table()


 if choice == "PetrolPump":
    menu = ["Add", "View", "Update", "remove"]
    choice2 = st.sidebar.selectbox("CRUD Operations", menu)
    if choice2 == "Add":
      st.subheader("Enter Petrolpump Details:")
      create_for_Petrolpump()
    elif choice2 == "View":
      st.subheader("View the Petrolpump details:")
      read_for_Petrolpump()
    elif choice2 == "Update":
      st.subheader("Updated petrolpump  tasks")
      update_for_Petrolpump()
    elif choice2 == "remove":
      st.subheader("Deleted petrolpump  tasks")
      delete_for_Petrolpump()


 elif choice == "Owners":
    menu = ["Add", "View", "Update", "Remove"]
    choice2 = st.sidebar.selectbox("CRUD Operations", menu)
    if choice2 == "Add":
        st.subheader("Enter Owners Details:")
        create_for_Owners()
    elif choice2 == "View":
        st.subheader("View Owners details:")
        read_for_Owners()
```

```
  elif choice2 == "Update":

      st.subheader("Update created tasks")

      update_for_Owners()

  elif choice2 == "Remove":

      st.subheader("Delete created tasks")

      delete_for_Owners()


elif choice == "Employee":

  menu = ["Add", "View", "Update", "Remove"]

  choice2 = st.sidebar.selectbox("CRUD Operations", menu)

  if choice2 == "Add":

    st.subheader("Enter Employee Details:")

    create_for_Employee()

  elif choice2 == "View":

    st.subheader("View the Employee details:")

    read_for_Employee()

  elif choice2 == "Update":

    st.subheader("Update created tasks")

    update_for_Employee()

  elif choice2 == "Remove":

    st.subheader("Delete created tasks")

    delete_for_Employee()


elif choice == "Customer":

  menu = ["Add", "View", "Update", "Remove"]

  choice2 = st.sidebar.selectbox("CRUD Operations", menu)

  if choice2 == "Add":

    st.subheader("Enter trainer Details:")
```

```
        create_for_Customer()

    elif choice2 == "View":

        st.subheader("View the trainer details:")

        read_for_Customer()

    elif choice2 == "Update":

        st.subheader("Update created tasks")

        update_for_Customer()

    elif choice2 == "Remove":

        st.subheader("Delete created tasks")

        delete_for_Customer()


elif choice == "Invoice":

    menu = ["Add", "View", "Update", "Remove"]

    choice2 = st.sidebar.selectbox("CRUD Operations", menu)

    if choice2 == "Add":

        st.subheader("Enter Invoice Details:")

        create_for_Invoice()

    elif choice2 == "View":

        st.subheader("View the Invoice details:")

        read_for_Invoice()

    elif choice2 == "Update":

        st.subheader("Update created tasks")

        update_for_Invoice()

    elif choice2 == "Remove":

        st.subheader("Delete created tasks")

        delete_for_Invoice()


elif choice == "Tanker":
  menu = ["Add", "View", "Update", "Remove"]

  choice2 = st.sidebar.selectbox("CRUD Operations", menu)

  if choice2 == "Add":

     st.subheader("Enter Tanker Details:")

     create_for_Tanker()

  elif choice2 == "View":

     st.subheader("View the Tanker details:")

     read_for_Tanker()

  elif choice2 == "Update":

     st.subheader("Update created tasks")

     update_for_Tanker()

  elif choice2 == "Remove":

     st.subheader("Delete created tasks")

     delete_for_Tanker()



elif choice == "Query":

   menu = ["Custom Query","Function"]

   choice2 = st.sidebar.selectbox("Query", menu)

   if choice2 == "Custom Query":

     query = st.text_input("Enter Your Query:")

     if st.button("Run Query"):

        c.execute(query)

        data = c.fetchall()

        st.dataframe(data)

   elif choice2 == "Function":

     net_value()
  else:


 st.subheader("About tasks")


def net_value():

  tanker_id = st.text_input("Enter Tanker ID:")

  result = TOTAL_Amount(tanker_id)

  if st.button("RUN Function"):

    df2=pd.DataFrame(result, columns = ["Total Amount"])

    st.dataframe(df2)


if __name__ == '__main__':

  main()
```

Create file code(functions)

```
import streamlit as st

from database import *




def create_for_Petrolpump():

   with st.container():

      Registration_No = st.text_input("Registration_No:")

      Petrolpump_Name = st.text_input("Petrolpump_Name:")

      Company_Name = st.text_input("Company_Name:")

      Opening_Year = st.number_input("Opening_Year:")

      State = st.text_input("State:")

      City = st.text_input("City:")
```

```
    if st.button("Add Petrolpump Details"):

  add_Petrolpump_data(Registration_No,Petrolpump_Name,Company_Name,Opening_Year,State,City)
        st.success("Successfully added Petrolpump details: {}".format(Registration_No))



def create_for_Owners():
    with st.container():
        Owner_Name = st.text_input("Owner_Name:")
        Contact_NO = st.text_input("Contact_NO:")
        DOB = st.date_input("DOB:")
        Gender = st.text_input("Gender:")
        Address = st.text_input("Enter Address")
        Partnership = st.number_input("Your Partership")

    if st.button("Add Owners Details"):
        add_Owners_data(Owner_Name, Contact_NO, DOB, Gender, Address, Partnership)
        st.success("Successfully added Owners details: {}".format(Owner_Name))



def create_for_Employee():
    with st.container():
        Employee_ID = st.text_input("Employee_ID")
        Emp_Name = st.text_input("Emp_Name:")
        Emp_Gender = st.text_input("Emp_Gender:")
        Designation = st.text_input(" Designation:")
        DOB= st.date_input("DOB:")
        Salary = st.number_input("Salary:")
        Emp_Address=st.text_input("Emp_Address:")


  Email_ID=st.text_input("Email_ID:")

        Petrolpump_No=st.text_input("Petrolpump_No:")

        Manager_ID=st.text_input("Manager_ID:")


    if st.button("Add Employee Details"):

        add_Employee_data(Employee_ID, Emp_Name,  Emp_Gender,   Designation,  DOB, Salary,
Emp_Address, Email_ID , Petrolpump_No, Manager_ID)

        st.success("Successfully added Employee details: {}".format(Employee_ID))




def create_for_Customer():

    with st.container():

        Customer_Code = st.text_input("Customer_Code")

        C_Name = st.text_input("C_Name:")

        Phone_No = st.text_input("Phone_No:")

        Email_ID=st.text_input("Email_ID")

        Gender = st.text_input("Gender:")

        City = st.text_input("City:")

        Age = st.number_input("Age")


    if st.button("Add Customer Details"):

        add_Customer_data(Customer_Code , C_Name , Phone_No  , Email_ID , Gender,  City , Age)

        st.success("Successfully added Customer details: {}".format(Customer_Code))




def create_for_Invoice():
    with st.container():

        Invoice_No=st.text_input(" Invoice_No:")

        Date=st.date_input("Date:")

        Payment_Type=st.text_input("Payment_Type:")

        Fuel_Amount=st.number_input("Fuel_Amount:")

        Fuel_Type=st.text_input("Fuel_Type:")

        Discount=st.number_input("Discount:")

        Total_Price=st.number_input("Total_Price:")

        Customer_Code=st.text_input("Customer_Code:")



    if st.button("Add Invoice Details"):

        add_Invoice_data(Invoice_No , Date , Payment_Type , Fuel_Amount , Fuel_Type , Discount  ,
Total_Price , Customer_Code)

        st.success("Successfully added Invoice details: {}".format(Invoice_No))


def create_for_Tanker():

    with st.container():

        Tanker_ID = st.text_input("Tanker_ID:")

        Capacity = st.number_input("Capacity:")

        pressure = st.number_input("pressure:")

        Fuel_ID = st.text_input("Fuel_ID")

        Fuel_Amount = st.number_input("Fuel_Amount")

        Fuel_Name= st.text_input("Fuel_Name:")

        Fuel_Price= st.number_input("Fuel_Price:")

        Petrolpump_No=st.text_input("Petrolpump_No:")


    if st.button("Add Tanker Details"):
```

```
    add_Tanker_data(Tanker_ID  , Capacity,  pressure,  Fuel_ID , Fuel_Amount, Fuel_Name , Fuel_Price , 
Petrolpump_No)


 st.success("Successfully added Tanker details: {}".format(Tanker_ID))
```

Delete file code

```
import pandas as pd

import streamlit as st

from database import *


def delete_for_Petrolpump():

    result = view_all_Petrolpump_data()

    df = pd.DataFrame(result, 
columns=['Registration_No','Petrolpump_Name','Company_Name','Opening_Year','State','City'])

    with st.expander("View all Petrolpump"):

        st.dataframe(df)


    list_of_Petrolpump = [i[0] for i in view_only_Registration_No()]

    selected_Petrolpump = st.selectbox("Petrolpump to delete", list_of_Petrolpump)

    st.warning("Do you want to delete ::{}".format(selected_Petrolpump))

    if st.button("Delete Petrolpump"):

        delete_data_Petrolpump(selected_Petrolpump)

        st.success("Petrolpump has been deleted successfully")

    result2 = view_all_Petrolpump_data()

    df2 = pd.DataFrame(result2, 
columns=['Registration_No','Petrolpump_Name','Company_Name','Opening_Year','State','City'])

    with st.expander("Updated data"):

        st.dataframe(df2)
```

```
def delete_for_Owners():

    result = view_all_Owners_data()

    df = pd.DataFrame(result, columns=['Owner_Name', 'Contact_NO', 'DOB', 'Gender', 'Address',
'Partnership'])

    with st.expander("View all Owners"):

        st.dataframe(df)


    list_of_Owners = [i[0] for i in view_only_Owner_Name()]

    selected_Owners = st.selectbox("Owners to delete", list_of_Owners)

    st.warning("Do you want to delete ::{}".format(selected_Owners))

    if st.button("Delete Owners"):

        delete_data_Owners(selected_Owners)

        st.success("Owners has been deleted successfully")


    result2 = view_all_Owners_data()

    df2 = pd.DataFrame(result2, columns=['Owner_Name', 'Contact_NO', 'DOB', 'Gender', 'Address',
'Partnership'])

    with st.expander("Updated data"):

        st.dataframe(df2)




def delete_for_Employee():

    result = view_all_Employee_data()

    df = pd.DataFrame(result, columns=['Employee_ID', 'Emp_Name', 'Emp_Gender', 'Designation','DOB',
'Salary', 'Emp_Address', 'Email_ID' , 'Petrolpump_No', 'Manager_ID'])

    with st.expander("View all Employee"):

        st.dataframe(df)


    list_of_Employee = [i[0] for i in view_only_Employee_ID()]
    selected_Employee = st.selectbox("Employee to delete", list_of_Employee)

    st.warning("Do you want to delete ::{}".format(selected_Employee))


‘  if st.button("Delete Employee"):

        delete_data_Employee(selected_Employee)

        st.success("Employee has been deleted successfully")

    result2 = view_all_Employee_data()

    df2 = pd.DataFrame(result2, columns=['Employee_ID', 'Emp_Name', 'Emp_Gender', 'Designation','DOB',
'Salary', 'Emp_Address', 'Email_ID' , 'Petrolpump_No', 'Manager_ID'])

    with st.expander("Updated data"):

        st.dataframe(df2)



def delete_for_Customer():

    result = view_all_Customer_data()

    df = pd.DataFrame(result, columns=['Customer_Code', 'C_Name' , 'Phone_No', 'Email_ID' , 'Gender', 'City'
, 'Age'])

    with st.expander("View all Customer"):

        st.dataframe(df)


    list_of_Customer = [i[0] for i in view_only_Customer_Code()]

    selected_Customer = st.selectbox("Customer to delete", list_of_Customer)

    st.warning("Do you want to delete ::{}".format(selected_Customer))

    if st.button("Delete customer"):

        delete_data_Customer(selected_Customer)

        st.success("customer has been deleted successfully")

    result2 = view_all_Customer_data()

    df2= pd.DataFrame(result2, columns=['Customer_Code', 'C_Name' , 'Phone_No', 'Email_ID' , 'Gender',
'City' , 'Age'])

    with st.expander("Updated data"):
        st.dataframe(df2)



def delete_for_Invoice():

    result = view_all_Invoice_data()

    df = pd.DataFrame(result, columns=['Invoice_No' , 'Date' , 'Invoice_Type' , 'Fuel_Amount' , 'Fuel_Type' , 
'Discount' , 'Total_Price' , 'Customer_Code'])

    with st.expander("View all Invoices"):

        st.dataframe(df)


    list_of_Invoices = [i[0] for i in view_only_Invoice_No()]

    selected_Invoice = st.selectbox("Invoices to delete", list_of_Invoices)

    st.warning("Do you want to delete ::{}".format(selected_Invoice))

    if st.button("Delete Invoice"):

        delete_data_Invoice(selected_Invoice)

        st.success("transaction has been deleted successfully")

    result2 = view_all_Invoice_data()

    df2 = pd.DataFrame(result2, columns=['Invoice_No' , 'Date' , 'Invoice_Type' , 'Fuel_Amount' , 'Fuel_Type' , 
'Discount' , 'Total_Price' , 'Customer_Code'])

    with st.expander("Updated data"):

        st.dataframe(df2)



def delete_for_Tanker():

    result = view_all_Tanker_data()

    df = pd.DataFrame(result, columns=['Tanker_ID', 'Capacity', 'pressure', 'Fuel_ID' , 'Fuel_Amount', 
'Fuel_Name' , 'Fuel_Price' , 'Petrolpump_No'])

    with st.expander("View all Tankers"):

        st.dataframe(df)
    list_of_Tanker = [i[0] for i in view_only_Tanker_ID()]


    selected_Tanker = st.selectbox("Tanker to delete", list_of_Tanker)


   st.warning("Do you want to delete ::{}".format(selected_Tanker))
    if st.button("Delete Tanker"):
        delete_data_Tanker(selected_Tanker)
        st.success("Tanker has been deleted successfully")
    result2 = view_all_Tanker_data()
    df2 = pd.DataFrame(result2, columns=['Tanker_ID', 'Capacity', 'pressure', 'Fuel_ID' , 'Fuel_Amount',
'Fuel_Name' , 'Fuel_Price' , 'Petrolpump_No'])
    with st.expander("Updated data"):
        st.dataframe(df2)
```

View File Code

```
import pandas as pd
import streamlit as st
from database import *



def read_for_Petrolpump():
    result = view_all_Petrolpump_data()
    df = pd.DataFrame(result,
columns=['Registration_No','Petrolpump_Name','Company_Name','Opening_Year','State','City'])
    with st.expander("View all Petrolpumps"):
        st.dataframe(df)


def read_for_Owners():
    result = view_all_Owners_data()
    df = pd.DataFrame(result, columns=['Owner_Name', 'Contact_NO', 'DOB', 'Gender', 'Address',
'Partnership'])

    with st.expander("View all Owners"):

        st.dataframe(df)



def read_for_Employee():

    result = view_all_Employee_data()

    df = pd.DataFrame(result, columns=['Employee_ID', 'Emp_Name', 'Emp_Gender', 'Designation', 'DOB',
'Salary',  'Emp_Address', 'Email_ID' , 'Petrolpump_No', 'Manager_ID'])

    with st.expander("View all Employees"):

        st.dataframe(df)



def read_for_Customer():

    result = view_all_Customer_data()

    df = pd.DataFrame(result, columns=['Customer_Code', 'C_Name' , 'Phone_No' , 'Email_ID' , 'Gender', 'City'
, 'Age'])

    with st.expander("View all Customers"):

        st.dataframe(df)



def read_for_Invoice():

    result = view_all_Invoice_data()

    df = pd.DataFrame(result, columns=['Invoice_No' , 'Date','Payment_Type', 'Fuel_Amount' , 'Fuel_Type' ,
'Discount' , 'Total_Price' , 'Customer_Code'])

    with st.expander("View all Invoices"):

        st.dataframe(df)



def read_for_Tanker():

    result = view_all_Tanker_data()

    df = pd.DataFrame(result, columns=['Tanker_ID', 'Capacity', 'pressure', 'Fuel_ID' , 'Fuel_Amount',
'Fuel_Name' , 'Fuel_Price' , 'Petrolpump_No'])
```

```
    with st.expander("View all Tankers"):

        st.dataframe(df)
```

Update file Code

```
import pandas as pd

import streamlit as st

from database import *




def update_for_Petrolpump():

    result = view_all_Petrolpump_data()

    df = pd.DataFrame(result,
columns=['Registration_No','Petrolpump_Name','Company_Name','Opening_Year','State','City'])

    with st.expander("Current Petrolpump details"):

        st.dataframe(df)

    list_of_Petrolpump = [i[0] for i in view_only_Registration_No()]


    selected_Petrolpump = st.selectbox("Petrolpumps to Edit", list_of_Petrolpump)


    selected_result = get_all_info_Petrolpump(selected_Petrolpump)


    if selected_result:

        Registration_No = selected_result[0][0]

        Petrolpump_Name = selected_result[0][1]

        Company_Name = selected_result[0][2]

        Opening_Year = selected_result[0][3]

        State = selected_result[0][4]

        City = selected_result[0][5]

        with st.container():
            new_Petrolpump_Name = st.text_input("Petrolpump_Name:", Petrolpump_Name)

            new_Company_Name = st.text_input("Company_Name:", Company_Name)

            new_Opening_Year = st.number_input("Opening_Year",Opening_Year)

            new_State = st.text_input("State",State)

            new_City = st.text_input("City",City)


        if st.button("Update Petrolpump"):

            edit_Petrolpump_data(new_Petrolpump_Name, new_Company_Name, new_Opening_Year,
new_State, new_City, Registration_No)

            st.success("Successfully updated")


    result2 = view_all_Petrolpump_data()

    df2 = pd.DataFrame(result2,
columns=['Registration_No','Petrolpump_Name','Company_Name','Opening_Year','State','City'])

    with st.expander("Updated data"):

        st.dataframe(df2)




def update_for_Owners():

    result = view_all_Owners_data()


    df = pd.DataFrame(result, columns=['Owner_Name', 'Contact_NO', 'DOB', 'Gender', 'Address',
'Partnership'])

    with st.expander("Current Owners details"):

        st.dataframe(df)

    list_of_Owners = [i[0] for i in view_only_Owner_Name()]


    selected_Owners = st.selectbox("Owners to Edit", list_of_Owners)


    selected_result = get_all_info_Owners(selected_Owners)
    if selected_result:
        Owner_Name = selected_result[0][0]
        Contact_NO = selected_result[0][1]
        DOB = selected_result[0][2]
        Gender = selected_result[0][3]
        Address = selected_result[0][4]
        Partnership = selected_result[0][5]
        with st.container():
            new_Contact_NO = st.text_input("Contact_NO:",Contact_NO )
            new_DOB = st.date_input("DOB:",DOB)
            new_Gender = st.text_input("Gender:",Gender )
            new_Address = st.text_input("Address:", Address)
            new_Partnership = st.number_input("Partnership:", Partnership)
        if st.button("Update Owners"):
            edit_Owners_data(new_Contact_NO, new_DOB, new_Gender,new_Address, new_Partnership,
Owner_Name)
            st.success("Successfully updated")


    result2 = view_all_Owners_data()
    df2 = pd.DataFrame(result2, columns=['Owner_Name', 'Contact_NO', 'DOB', 'Gender', 'Address',
'Partnership'])
    with st.expander("Updated data"):
        st.dataframe(df2)




def update_for_Employee():
    result = view_all_Employee_data()
    df = pd.DataFrame(result, columns=['Employee_ID', 'Emp_Name', 'Emp_Gender', 'Designation','DOB',
'Salary', 'Emp_Address', 'Email_ID' , 'Petrolpump_No', 'Manager_ID'])
    with st.expander("Current Employee details"):


   st.dataframe(df)
    list_of_Employee = [i[0] for i in view_only_Employee_ID()]


    selected_Employee = st.selectbox("Employee to Edit", list_of_Employee)


    selected_result = get_all_info_Employee(selected_Employee)


    if selected_result:
        Employee_ID = selected_result[0][0]
        Emp_Name = selected_result[0][1]
        Emp_Gender = selected_result[0][2]
        Designation = selected_result[0][3]
        DOB = selected_result[0][4]
        Salary = selected_result[0][5]
        Emp_Address = selected_result[0][6]
        Email_ID = selected_result[0][7]
        Petrolpump_No = selected_result[0][8]
        Manager_ID = selected_result[0][9]
        with st.container():
            new_Emp_Name= st.text_input("Emp_Name:", Emp_Name)
            new_Emp_Gender= st.text_input("Emp_Gender:", Emp_Gender)
            new_Designation= st.text_input("Designation:", Designation)
            new_DOB= st.date_input("DOB:", DOB)
            new_Salary= st.number_input("Salary:", Salary)
            new_Emp_Address= st.text_input("Emp_Address:", Emp_Address)
        new_Email_ID= st.text_input("Email_ID:", Email_ID)

        new_Petrolpump_No= st.text_input("Petrolpump_No:", Petrolpump_No)

        new_Manager_ID= st.text_input("Manager_ID:", Manager_ID)


    if st.button("Update Employee"):

        try:

            edit_Employee_data(new_Emp_Name, new_Emp_Gender,  new_Designation, new_DOB, 
new_Salary,  new_Emp_Address, new_Email_ID , new_Petrolpump_No, new_Manager_ID, Employee_ID)

            st.success("Successfully updated")

        except Exception as err:

            st.exception(err)


  result2 = view_all_Employee_data()

  df2 = pd.DataFrame(result2, columns=['Employee_ID', 'Emp_Name', 'Emp_Gender', 'Designation','DOB', 
'Salary', 'Emp_Address', 'Email_ID' , 'Petrolpump_No', 'Manager_ID'])

  with st.expander("Updated data"):

    st.dataframe(df2)




def update_for_Customer():

  result = view_all_Customer_data()

  df = pd.DataFrame(result, columns=['Customer_Code', 'C_Name' , 'Phone_No', 'Email_ID' , 'Gender', 'City'
, 'Age'])

  with st.expander("Current Customer details"):

    st.dataframe(df)

  list_of_Customer = [i[0] for i in view_only_Customer_Code()]

  selected_Customer = st.selectbox("Customer to Edit", list_of_Customer)

  selected_result = get_all_info_Customer(selected_Customer)

  if selected_result:

    Customer_Code = selected_result[0][0]
```

```
        C_Name = selected_result[0][1]

        Phone_No = selected_result[0][2]

        Email_ID = selected_result[0][3]

        Gender = selected_result[0][4]

        City = selected_result[0][5]

        Age = selected_result[0][6]


        with st.container():

            new_C_Name = st.text_input("customer name:", C_Name)

            new_Phone_No = st.text_input("Phone_No:", Phone_No)

            new_Email_ID = st.text_input("Email_ID:", Email_ID)

            new_Gender = st.text_input("Gender:", Gender)

            new_City = st.text_input("City:", City)

            new_Age = st.number_input("Age:", Age)

        if st.button("Update Customer"):

            edit_Customer_data(new_C_Name , new_Phone_No  , new_Email_ID , new_Gender,  new_City ,
new_Age, Customer_Code)

            st.success("Successfully updated")


    result2 = view_all_Customer_data()

    df2 = pd.DataFrame(result2, columns=['Customer_Code', 'C_Name' , 'Phone_No', 'Email_ID' , 'Gender',
'City' , 'Age'])

    with st.expander("Updated data"):

        st.dataframe(df2)




def update_for_Invoice():

    result = view_all_Invoice_data()

    df = pd.DataFrame(result, columns=['Invoice_No', 'Date' , 'Payment_Type', 'Fuel_Amount', 'Fuel_Type',
'Discount', 'Total_Price', 'Customer_Code'])
```

```
    with st.expander("Current Invoice details"):

        st.dataframe(df)

    list_of_Invoice = [i[0] for i in view_only_Invoice_No()]


    selected_Invoice = st.selectbox("Invoice to Edit", list_of_Invoice)

    selected_result = get_all_info_Invoice(selected_Invoice)

    if selected_result:

        Invoice_No = selected_result[0][0]

        Date = selected_result[0][1]

        Payment_Type = selected_result[0][2]

        Fuel_Amount = selected_result[0][3]

        Fuel_Type = selected_result[0][4]

        Discount = selected_result[0][5]

        Total_Price = selected_result[0][6]

        Customer_Code = selected_result[0][7]

        with st.container():

            new_Date = st.date_input("Date:", Date)

            new_Payment_Type = st.text_input("Payment_Type:", Payment_Type)

            new_Fuel_Amount = st.number_input("Fuel_Amount:", Fuel_Amount)

            new_Fuel_Type = st.text_input("Fuel_Type:", Fuel_Type)

            new_Discount = st.number_input("Discount:", Discount)

            new_Total_Price = st.number_input("Total_Price:", Total_Price)

            new_Customer_Code = st.text_input("Customer_Code:", Customer_Code)


        if st.button("Update Invoice"):

            edit_Invoice_data(new_Date , new_Payment_Type , new_Fuel_Amount , new_Fuel_Type,
new_Discount, new_Total_Price , new_Customer_Code, Invoice_No)

            st.success("Successfully updated")
    result2 = view_all_Invoice_data()

    df2 = pd.DataFrame(result2, columns=['Invoice_No' , 'Date', 'Payment_Type' , 'Fuel_Amount' , 'Fuel_Type'
, 'Discount' , 'Total_Price' , 'Customer_Code'])


 with st.expander("Updated data"):

        st.dataframe(df2)




def update_for_Tanker():

    result = view_all_Tanker_data()

    df = pd.DataFrame(result, columns=['Tanker_ID', 'Capacity', 'pressure', 'Fuel_ID' , 'Fuel_Amount',
'Fuel_Name' , 'Fuel_Price' , 'Petrolpump_No'])

    with st.expander("Current Tanker details"):

        st.dataframe(df)

    list_of_Tankers = [i[0] for i in view_only_Tanker_ID()]

    selected_Tanker = st.selectbox("Tankers to Edit", list_of_Tankers)

    selected_result = get_all_info_Tanker(selected_Tanker)


    if selected_result:

        Tanker_ID = selected_result[0][0]

        Capacity = selected_result[0][1]

        pressure = selected_result[0][2]

        Fuel_ID = selected_result[0][3]

        Fuel_Amount = selected_result[0][4]

        Fuel_Name = selected_result[0][5]

        Fuel_Price = selected_result[0][6]

        Petrolpump_No = selected_result[0][7]


        with st.container():
```

```
        new_Capacity = st.text_input("Tanker Capacity:", Capacity)

        new_pressure = st.number_input("pressure:", pressure)

        new_Fuel_ID = st.text_input("Fuel_ID:", Fuel_ID)

        new_Fuel_Amount = st.number_input("Fuel_Amount:", Fuel_Amount)

        new_Fuel_Name = st.text_input("Fuel_Name:", Fuel_Name)


  new_Fuel_Price = st.number_input("Fuel_Price:", Fuel_Price)

        new_Petrolpump_No = st.text_input("Petrolpump No:", Petrolpump_No)

      if st.button("Update Tankers"):

        edit_Tanker_data(new_Capacity,  new_pressure,  new_Fuel_ID , new_Fuel_Amount, new_Fuel_Name
, new_Fuel_Price ,new_Petrolpump_No, Tanker_ID)

        st.success("Successfully updated")


   result2 = view_all_Tanker_data()

   df2 = pd.DataFrame(result2, columns=['Tanker_ID', 'Capacity', 'pressure', 'Fuel_ID' , 'Fuel_Amount',
'Fuel_Name' , 'Fuel_Price' , 'Petrolpump_No'])

   with st.expander("Updated data"):

      st.dataframe(df2)
```

5.3 Trigger in database :

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by INSERT or LOAD DATA statements, and an insert trigger activates for each inserted row.

```
--  Trigger --
DELIMITER $$
CREATE TRIGGER salary_check
BEFORE UPDATE
ON Employee FOR EACH ROW
BEGIN
declare error_msgvarchar(225);
set error_msg = ("Error: Insufficient Salary For Living");
if new.Salary< 30000 then
```

```
    signal sqlstate '45000'


    set MESSAGE_TEXT = error_msg;
    end if;
    END $$
```

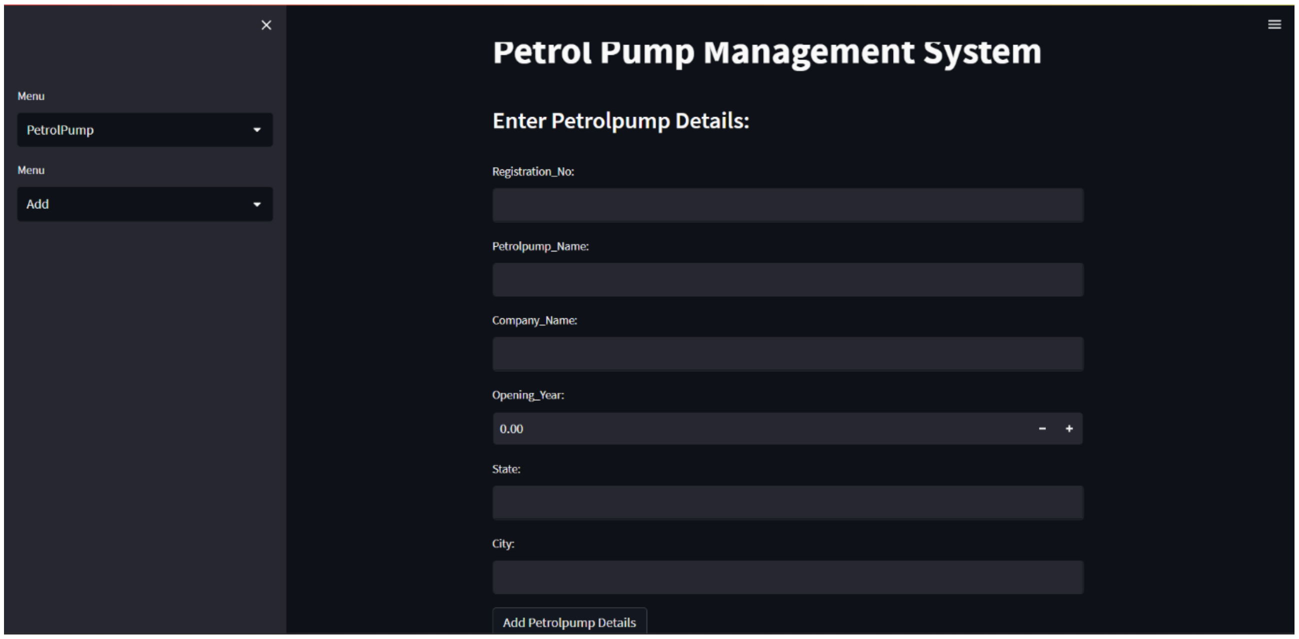
5.3 Functions in database :

```
-- Function

DELIMITER $$

CREATE FUNCTION `TOTAL_AMOUNT`(`TID` VARCHAR(10)) RETURNS float

DETERMINISTIC

BEGIN


DECLARE BILL FLOAT;

DECLARE RATE FLOAT;

DECLARE VOL FLOAT;


SET RATE = (SELECT FUEL_PRICE FROM TANKER WHERE TANKER_ID = TID);

SET VOL = (SELECT FUEL_AMOUNT FROM TANKER WHERE TANKER_ID = TID);


SET BILL = RATE * VOL;


RETURN BILL;


END$$

DELIMITER ;
```

CHAPTER 6

SNAPSHOTS

6.1. Front Page



Petrol Pump Management System

Enter Petrolpump Details:

Menu: PetrolPump

Menu: Add

Registration_No:

Petrolpump_Name:

Company_Name:

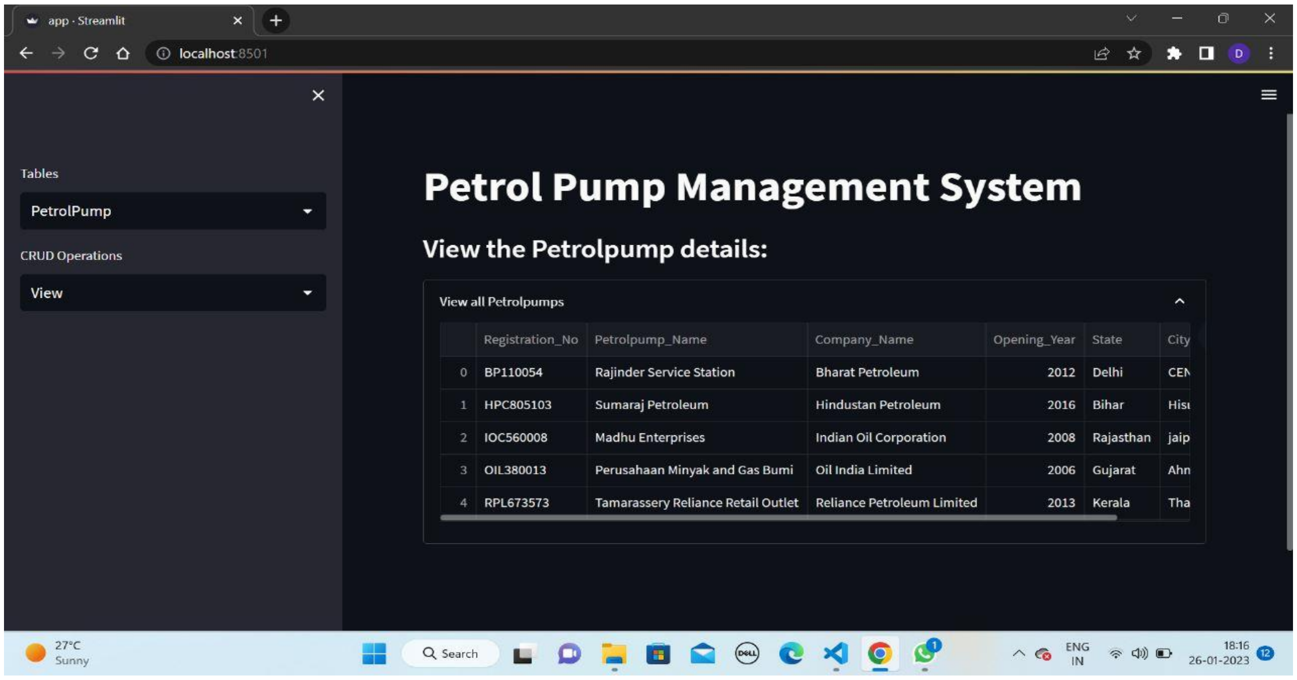
Opening_Year: 0.00

State:

City:

Add Petrolpump Details

6.2. View Page



app · Streamlit

localhost:8501

Tables: PetrolPump

CRUD Operations: View

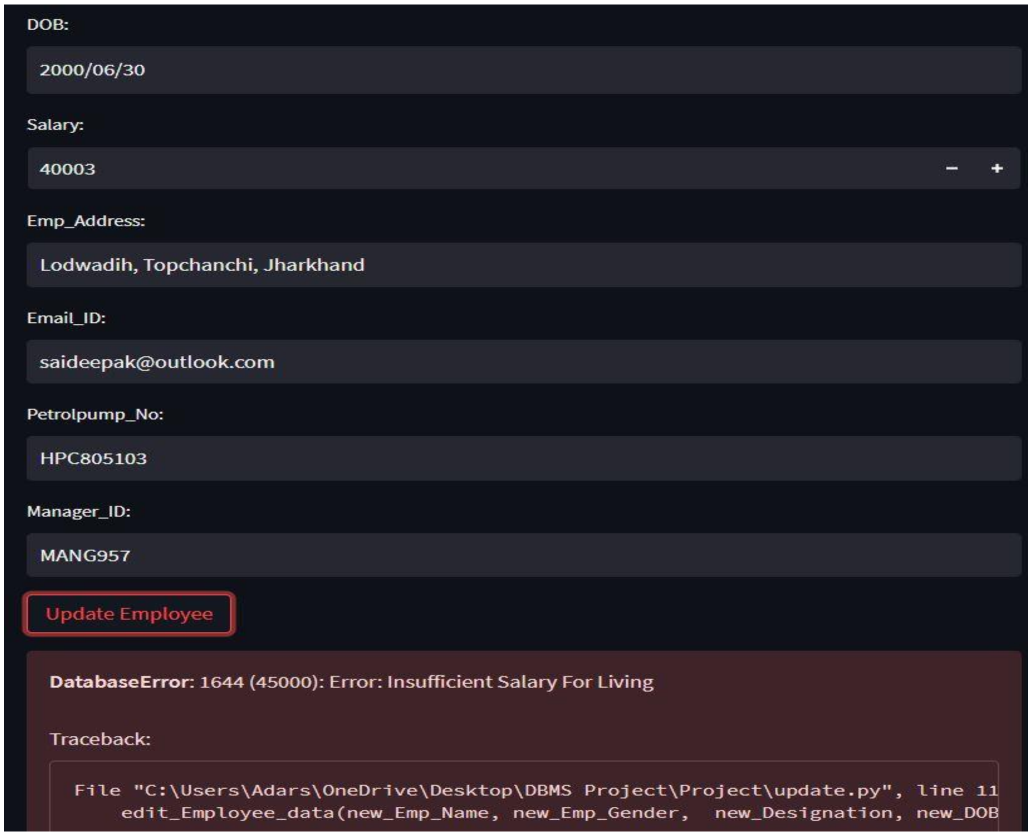
Petrol Pump Management System

View the Petrolpump details:

View all Petrolpumps

	Registration_No	Petrolpump_Name	Company_Name	Opening_Year	State	City
0	BP110054	Rajinder Service Station	Bharat Petroleum	2012	Delhi	CEN
1	HPC805103	Sumaraj Petroleum	Hindustan Petroleum	2016	Bihar	Hisu
2	IOC560008	Madhu Enterprises	Indian Oil Corporation	2008	Rajasthan	jaip
3	OIL380013	Perusahaan Minyak and Gas Bumi	Oil India Limited	2006	Gujarat	Ahn
4	RPL673573	Tamarassery Reliance Retail Outlet	Reliance Petroleum Limited	2013	Kerala	Tha

6.3.Trigger Implementation



DOB:

2000/06/30

Salary:

40003

Emp_Address:

Lodwadih, Topchanchi, Jharkhand

Email_ID:

saideepak@outlook.com

Petrolpump_No:

HPC805103

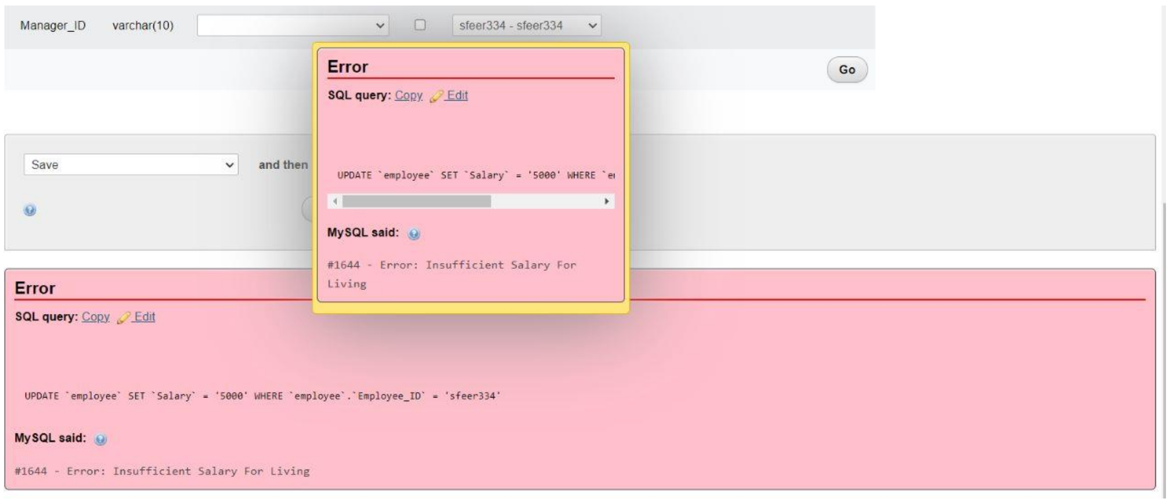
Manager_ID:

MANG957

DatabaseError: 1644 (45000): Error: Insufficient Salary For Living

Traceback:

```
File "C:\Users\Adars\OneDrive\Desktop\DBMS Project\Project\update.py", line 11
    edit_Employee_data(new_Emp_Name, new_Emp_Gender,  new_Designation, new_DOB
```



Manager_ID varchar(10) sfeer334 - sfeer334

Error

SQL query: Copy Edit

```
UPDATE `employee` SET `Salary` = '5000' WHERE `e
```

MySQL said:

```
#1644 - Error: Insufficient Salary For
Living
```

Error

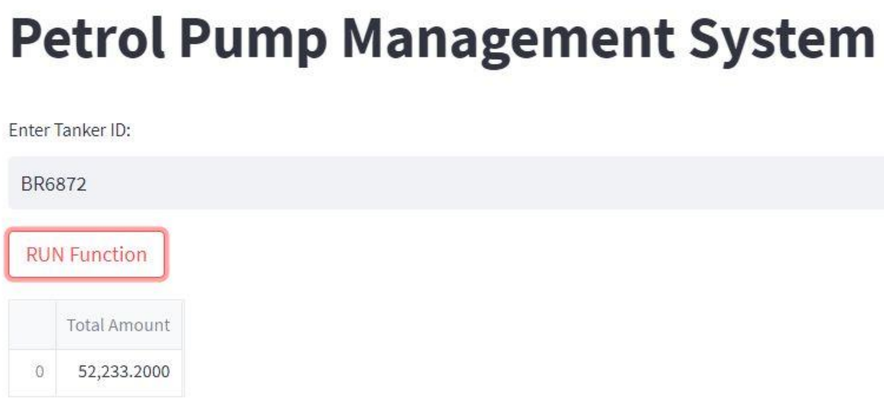
SQL query: Copy Edit

```
UPDATE `employee` SET `Salary` = '5000' WHERE `employee`.`Employee_ID` = 'sfeer334'
```

MySQL said:

```
#1644 - Error: Insufficient Salary For Living
```

6.4.Tanker Function for Total amount



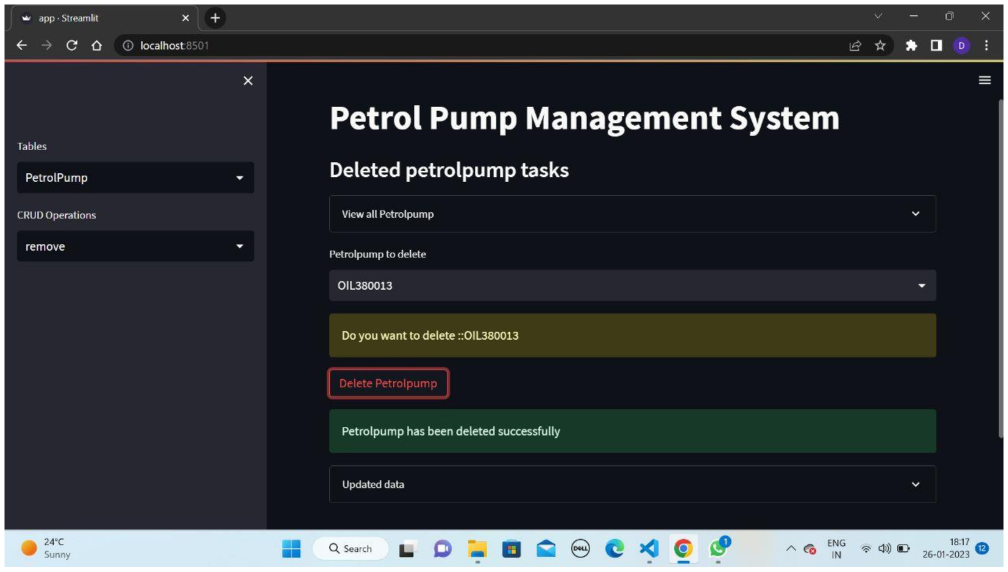
Petrol Pump Management System

Enter Tanker ID:

BR6872

	Total Amount
0	52,233.2000

6.5.Delete Operation



Tables

PetrolPump

CRUD Operations

remove

Petrol Pump Management System

Deleted petrolpump tasks

Petrolpump to delete

OIL380013

Do you want to delete ::OIL380013

Petrolpump has been deleted successfully

CHAPTER 7

APPLICATION

APPLICATION

- This project will maintain data about
    - Petrol Pumps in an area, their owners, Employees details working in that petrol.
    - Customer detail so that a regular customer will get Goodies & Discount.
    - Tanker details as well as Sales of a particular Petrol Pump.
- This project ensure data accuracy
- Minimize manual data entry.
- Provide security

7.1 TEST CASES

Module	**Given Input**	**Expected Output**	**Actual Output**	**Remark**
admin login	enter valid usernameand password	Login successful	Logged in successfully	Tested Ok
admin login	Invalid username orpassword	Login Failure	Log In failure	Tested Ok
admin	Click on menu option	Display all the tables with CRUD operation	Display all the tables with CRUD operation	Tested Ok
admin	Click on given tables	Viewed forms associated with tables	Viewed successfully	Tested Ok
admin	Fill the form linked with tables	Filled successfully with successful message	Filled successfully	Tested Ok
admin	Create, Delete,Update, view option	Open the given data associate with this	CRUD operation done	Tested Ok

CHAPTER 8

CONCLUSION

CONCLUSION

- An application has been created for managing Petrol Pumps.
- Petrol Pump Management system can be easily maintained and updated by the administrator.
- Administrator can easily maintain employee, owner, tanker, inventory list and make SQL queries .
- Administrator can delete and also view the updated results easily.

CHAPTER 8

REFERENCES

REFERENCES

- https://youtube.com/playlist?list=PLu0W_9lII9aikXkRE0WxDt1vozo3hnmtR
- Fundamentals of database Systems ,Ramez Elmasri and Shamkant B. Navathe 7th edition
- https://wikipedia.com
- https://app.diagrams.net/
- https://www.geeksforgeeks.org/
- https://stackoverflow.com/
- https://www.w3schools.com/