Mini project report on

ELECTRICITY MANAGEMENT SYSTEM

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE21CS351A – DBMS Project

Submitted by:

Melvin Jojee Joseph	**PES2UG21CS294**
L Sai Tejas	**PES2UG21CS250**

Under the guidance of

Prof. Mannan J Mannan

Associate Professor,

CSE

PES University

AUG - DEC 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Electricity Management System

is a bonafide work carried out by

Melvin Jojee Joseph	**PES2UG21CS294**
L Sai Tejas	**PES2UG21CS250**

In partial fulfilment for the completion of fifth semester DBMS Project (UE20CSS301) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2022 – DEC. 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5^{th} semester academic requirements in respect of project work.

Signature
Prof. Mannan J Mannar
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Electricity management system** has been carried out by us under the guidance of **Prof. Mannan J Mannar , Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2023.

Melvin Jojee Joseph	**PES2UG21CS294**
L Sai Tejas	**PES2UG21CS250**

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Mannan J Mannar, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE21CS351 - DBMS Project.

I take this opportunity to thank Dr. Sandesh B J, C, Professor, Chair Person, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr.Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

The constant evolution of technology and the increasing demand for energy necessitate efficient and intelligent systems for managing electricity consumption. This abstract introduces an innovative Electricity Management System (EMS) designed for seamless integration with Database Management Systems (DBMS). The proposed system aims to provide a user-friendly platform for managing and tracking electricity consumption, billing, and payment processes efficiently. This system aims to streamline the billing process enhance user experience by providing seamless access to the users to their monthly consumption.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In the dynamic landscape of modern energy consumption, efficient and organized management of electricity is paramount. To address the complexities and demands of electricity distribution and billing, the Electricity Management System (EMS) emerges as a robust solution. This system leverages the power of a well-designed Database Management System (DBMS) to streamline operations, enhance data accuracy, and improve overall efficiency.

The key entities within the Electricity Management System include customers, administrators, bills, tariff structures and the electricity board. Each entity plays a crucial role in maintaining a seamless flow of information and services throughout the electricity distribution network.

1. **Customer:**
    - Represents the end-users who consume electricity.
    - Captures and manages customer details such as contact information, meter readings, and consumption history.
2. **Admin:**
    - Empowers system administrators to oversee and control the entire EMS.
    - Manages user accounts, system configurations, and ensures the security and integrity of the database.
3. **Bills:**
    - Encompasses the billing information associated with each customer.
    - Stores details like billing period, meter readings, and calculates the total consumption cost.
4. **Tariff:**
    - Defines the pricing structures for electricity consumption.
    - Encompasses different rates based on factors such as usage patterns, time of day, and any applicable discounts.
5. **Electricity Board:**

- Represents the governing body responsible for the overall management and regulation of electricity distribution.
- Monitors system performance, implements policies, and ensures compliance with regulatory standards.

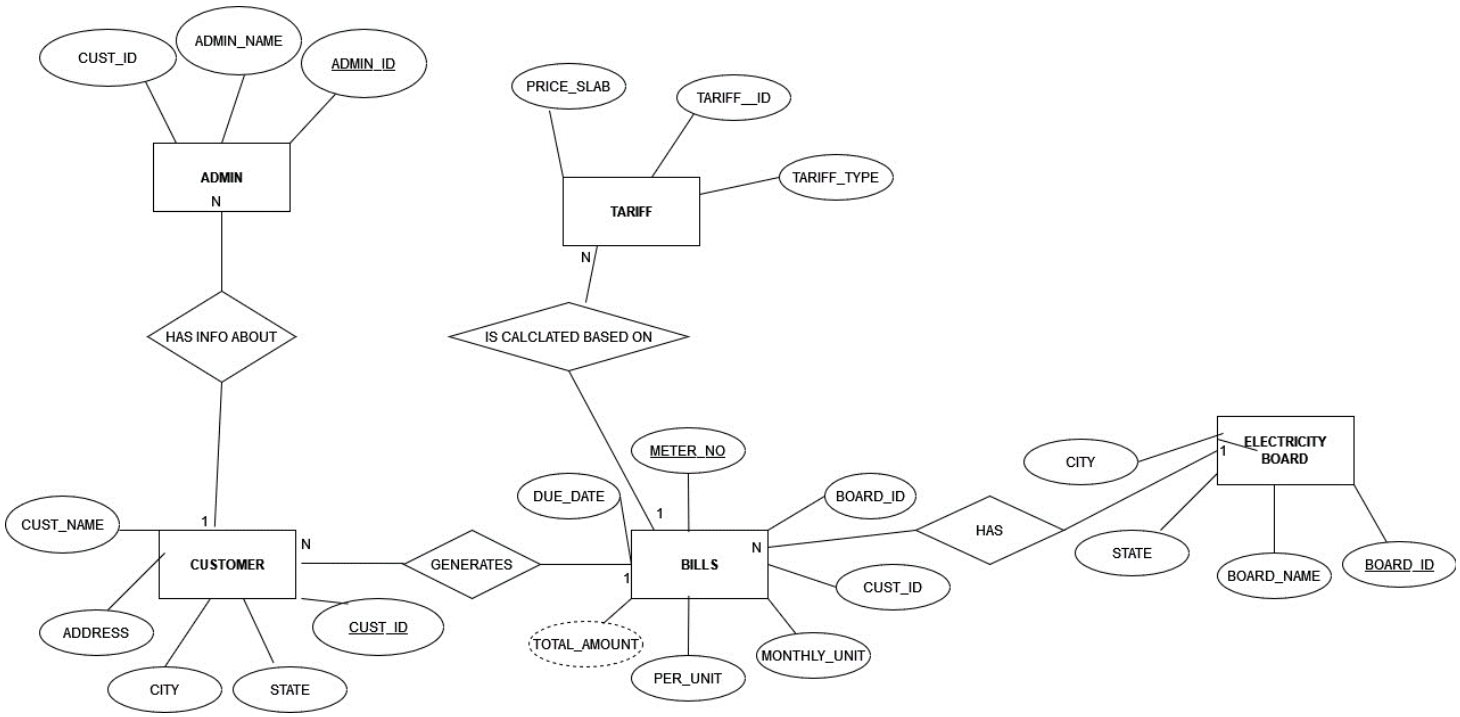
The integration of a DBMS into the Electricity Management System not only enhances data organization but also facilitates real-time data retrieval and analysis. This system provides a user-friendly interface for both administrators and customers, allowing for seamless interaction and efficient decision-making.

By centralizing and optimizing the management of entities like customers, admins, bills, tariffs, and the electricity board, the Electricity Management System becomes a pivotal tool in fostering a reliable, transparent, and economically viable electricity distribution network. This introduction sets the stage for exploring the various functionalities and benefits offered by the EMS powered by a robust DBMS.

2. PROBLEM DEFINITION:

The Electricity Management System using a Database Management System aims to address these challenges by providing a structured and efficient platform. It centralizes data, streamlines operations, enhances user interaction, ensures accurate billing, facilitates financial transaction management, and promotes regulatory compliance. By identifying and articulating these challenges, the problem definition sets the stage for the development and implementation of a comprehensive solution to improve the overall electricity management process.

3. ER MODEL:

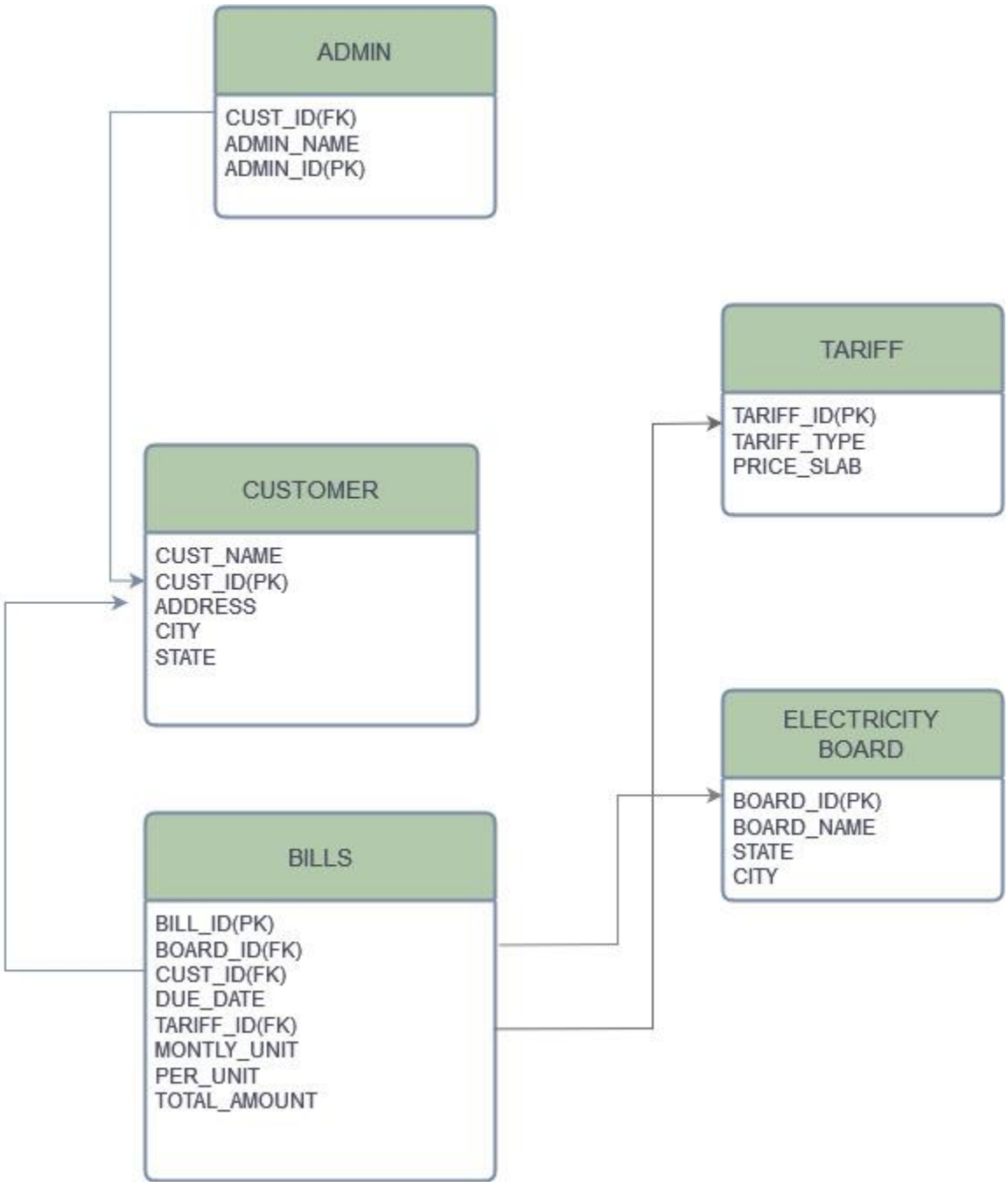

CUST_ID
ADMIN_NAME
ADMIN_ID
ADMIN
N
HAS INFO ABOUT
PRICE_SLAB
TARIFF__ID
TARIFF_TYPE
TARIFF
N
IS CALCLATED BASED ON
METER_NO
DUE_DATE
1
BOARD_ID
CUST_NAME
1
CUSTOMER
N
GENERATES
1
BILLS
N
HAS
CITY
1
ELECTRICITY BOARD
STATE
BOARD_NAME
BOARD_ID
CUST_ID
ADDRESS
CUST_ID
TOTAL_AMOUNT
MONTHLY_UNIT
PER_UNIT
CITY
STATE

4. ER TO RELATIONAL MAPPING:

4.1 STEPS OF ALGORITHM FOR CHOOSEN PROBLEM :

1. **Identify Entities:**
   - Entities in the ER diagram are translated into tables in the relational model.
   - In the provided script, tables such as `customer`, `admin`, `electricity_board`, `tariff`, and `bill` represent the entities.
2. **Define Attributes:**
   - Attributes of entities become columns in the corresponding tables.
   - For example, in the `customer` table, attributes like `cust_id`, `name`, `address`, `city`, and `state` are columns.
3. **Identify Primary Keys:**
   - Primary keys in the ER diagram become primary key constraints in the relational model.
   - In the script, the `PRIMARY KEY` constraints are applied to columns like `cust_id`, `admin_id`, `board_id`, `tariff_id`, and `bill_id`.
4. **Handle Relationships:**
   - Relationships between entities are represented using foreign keys.
   - For instance, in the `admin` table, `cust_id` is a foreign key referencing the `customer` table's `cust_id`.
   - In the `bill` table, `board_id` and `cust_id` are foreign keys referencing the `electricity_board` and `customer` tables, respectively.
5. **Handle Cardinality:**
   - If there are one-to-one, one-to-many, or many-to-many relationships, ensure that foreign keys are appropriately placed to maintain referential integrity.
6. **Translate Weak Entities:**
   - If there are weak entities, they might not have a primary key of their own. In such cases, a composite primary key involving the owner entity's primary key may be used.
7. **Create Indexes:**
   - Depending on the requirements and query patterns, you may need to create indexes on certain columns for better performance.
8. **Data Types and Constraints:**
   - Choose appropriate data types for each column (e.g., `INT`, `VARCHAR(255)`, `DATE`) based on the nature of the data.
   - Apply constraints such as `NOT NULL` where necessary.

4.2 COMPLETE DIAGRAM OF RELATIONAL MAPPING:



5. DDL STATEMENTS:

Creation of table “customer”:

```
CREATE TABLE customer
(
cust_id INT AUTO_INCREMENT,
name VARCHAR(255) NOT NULL,
address VARCHAR(255) NOT NULL,
city VARCHAR(255) NOT NULL,
state VARCHAR(255) NOT NULL,
PRIMARY KEY (cust_id)
);
```

```
pes2ug21cs294_250>CREATE TABLE customer
    -> (
    -> cust_id INT AUTO_INCREMENT,
    -> name VARCHAR(255),
    -> address VARCHAR(255),
    -> city VARCHAR(255),
    -> state VARCHAR(255),
    -> PRIMARY KEY (cust_id)
    -> );
Query OK, 0 rows affected (0.14 sec)
```

Creation of table “admin”:

```
CREATE TABLE admin
(
admin_id INT AUTO_INCREMENT NOT NULL,
admin_name VARCHAR(255) NOT NULL,
cust_id INT NOT NULL,
PRIMARY KEY (admin_id),
FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
);
```

```
pes2ug21cs294_250>CREATE TABLE admin
    -> (
    -> admin_id INT AUTO_INCREMENT,
    -> admin_name VARCHAR(255),
    -> cust_id INT,
    -> PRIMARY KEY (admin_id),
    -> FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
    -> );
Query OK, 0 rows affected (0.23 sec)
```

Creation of table “electricity_board”:

```
CREATE TABLE electricity_board
(
board_id INT AUTO_INCREMENT NOT NULL,
board_name VARCHAR(255) NOT NULL,
state VARCHAR(255)  NOT NULL,
city VARCHAR(255) NOT NULL,
PRIMARY KEY (board_id)
);
```

```
pes2ug21cs294_250>CREATE TABLE electricity_board
    -> (
    -> board_id INT AUTO_INCREMENT,
    -> board_name VARCHAR(255),
    -> state VARCHAR(255),
    -> city VARCHAR(255),
    -> PRIMARY KEY (board_id)
    -> );
Query OK, 0 rows affected (0.40 sec)
```

Creation of the table “tariff”:

```
CREATE TABLE tariff
(
tariff_id INT AUTO_INCREMENT NOT NULL,
tariff_type VARCHAR(255) NOT NULL,
price_slab INT NOT NULL,
PRIMARY KEY (tariff_id),
CONSTRAINT test_column_positive CHECK (price_slab > 0)
);
```

```
pes2ug21cs294_250>CREATE TABLE tariff
    -> (
    -> tariff_id INT AUTO_INCREMENT NOT NULL,
    -> tariff_type VARCHAR(255) NOT NULL,
    -> price_slab INT NOT NULL,
    -> PRIMARY KEY (tariff_id),
    -> CONSTRAINT test_column_positive CHECK (price_slab > 0)
    -> );
Query OK, 0 rows affected (0.10 sec)
```

Creation of the table “bill”:

```
CREATE TABLE bill
(
bill_id INT AUTO_INCREMENT NOT NULL,
board_id INT NOT NULL,
cust_id INT NOT NULL,
meter_number VARCHAR(255) NOT NULL,
monthly_units INT NOT NULL,
amount_per_unit INT NOT NULL ,
total_amount INT NOT NULL,
due_date DATE NOT NULL,
PRIMARY KEY (bill_id),
FOREIGN KEY (board_id) REFERENCES electricity_board(board_id),
FOREIGN KEY (cust_id) REFERENCES customer(cust_id),
CONSTRAINT amount_per_unit_positive CHECK (amount_per_unit > 0),
CONSTRAINT monthly_units_positive CHECK (monthly_units > 0),
CONSTRAINT total_amount_positive CHECK (total_amount > 0)
);
```

```
pes2ug21cs294_250>CREATE TABLE bill
    -> (
    -> bill_id INT AUTO_INCREMENT NOT NULL,
    -> board_id INT NOT NULL,
    -> cust_id INT NOT NULL,
    -> meter_number VARCHAR(255) NOT NULL,
    -> monthly_units INT NOT NULL,
    -> amount_per_unit INT NOT NULL ,
    -> total_amount INT NOT NULL,
    -> due_date DATE NOT NULL,
    -> PRIMARY KEY (bill_id),
    -> FOREIGN KEY (board_id) REFERENCES electricity_board(board_id),
    -> FOREIGN KEY (cust_id) REFERENCES customer(cust_id),
    -> CONSTRAINT amount_per_unit_positive CHECK (amount_per_unit > 0),
    -> CONSTRAINT monthly_units_positive CHECK (monthly_units > 0),
    -> CONSTRAINT total_amount_positive CHECK (total_amount > 0)
    -> );
Query OK, 0 rows affected (0.24 sec)
```

6. DML STATEMENTS

```
INSERT INTO customer (cust_id,name,address,city,state) VALUES ('100','Abhay','MG Road','Mysore','Karnataka');
INSERT INTO customer (name,address,city,state) VALUES ('Vishnu','Basaveshwara Nagar','Bangalore','Karnataka');
INSERT INTO customer (name,address,city,state) VALUES ('Anant','HD Kote Road','Mysore','Karnataka');
INSERT INTO customer (name,address,city,state) VALUES ('Vijay','KRS Road','Pune','Maharashtra');
INSERT INTO customer (name,address,city,state) VALUES ('Deekshith','RK Block','Chennai','Tamil Nadu');
INSERT INTO customer (name,address,city,state) VALUES ('Farhaan','Auromira','Ahmedabad','Gujarat');
INSERT INTO customer (name,address,city,state) VALUES ('Ajay','Pamban Bridge Road','Rameshwaram','Tamil Nadu');
INSERT INTO customer (name,address,city,state) VALUES ('Nikhil','HSR Layout','Bangalore','Karnataka');
INSERT INTO customer (name,address,city,state) VALUES ('Tushar','MS Raod','Lucknow','Uttar Pradesh');
INSERT INTO customer (name,address,city,state) VALUES ('Ayushman','Kanakapura Road','Bangalore','Karnataka');
INSERT INTO customer (name,address,city,state) VALUES ('Rohanjit','Bandra','Mumbai','Maharashtra');
INSERT INTO customer (name,address,city,state) VALUES ('Anwesh','DFG Layout','Indore','Madhya Pradesh');
INSERT INTO customer (name,address,city,state) VALUES ('Devash','Edapalli','Kochi','Kerala');
INSERT INTO customer (name,address,city,state) VALUES ('Preetham','AB Block','Ayodhya','Uttar Pradesh');
INSERT INTO customer (name,address,city,state) VALUES ('Sridhar','Gwalior Road','Gwalior','Madhya Pradesh');
INSERT INTO customer (name,address,city,state) VALUES ('Sahil','MG Road','New Delhi','Delhi');
```

```
pes2ug21cs294_250>SELECT * from customer;
+---------+-----------+---------------------+-------------+----------------+
| cust_id | name      | address             | city        | state          |
+---------+-----------+---------------------+-------------+----------------+
|     100 | Abhay     | MG Road             | Mysore      | Karnataka      |
|     101 | Vishnu    | Basaveshwara Nagar  | Bangalore   | Karnataka      |
|     102 | Anant     | HD Kote Road        | Mysore      | Karnataka      |
|     103 | Vijay     | KRS Road            | Pune        | Maharashtra    |
|     104 | Deekshith | RK Block            | Chennai     | Tamil Nadu     |
|     105 | Farhaan   | Auromira            | Ahmedabad   | Gujarat        |
|     106 | Ajay      | Pamban Bridge Road  | Rameshwaram | Tamil Nadu     |
|     107 | Nikhil    | HSR Layout          | Bangalore   | Karnataka      |
|     108 | Tushar    | MS Raod             | Lucknow     | Uttar Pradesh  |
|     109 | Ayushman  | Kanakapura Road     | Bangalore   | Karnataka      |
|     110 | Rohanjit  | Bandra              | Mumbai      | Maharashtra    |
|     111 | Anwesh    | DFG Layout          | Indore      | Madhya Pradesh |
|     112 | Devash    | Edapalli            | Kochi       | Kerala         |
|     113 | Preetham  | AB Block            | Ayodhya     | Uttar Pradesh  |
|     114 | Sridhar   | Gwalior Road        | Gwalior     | Madhya Pradesh |
|     115 | Sahil     | MG Road             | New Delhi   | Delhi          |
+---------+-----------+---------------------+-------------+----------------+
16 rows in set (0.00 sec)
```

```
INSERT INTO admin (admin_id,admin_name,cust_id) VALUES ('200','Sahil','100');
INSERT INTO admin (admin_name,cust_id) VALUES ('Karan','101');
INSERT INTO admin (admin_name,cust_id) VALUES ('Rahul','102');
INSERT INTO admin (admin_name,cust_id) VALUES ('Nikhil','103');
```

```
+----------+------------+---------+
| admin_id | admin_name | cust_id |
+----------+------------+---------+
|      200 | Sahil      |     100 |
|      201 | Karan      |     101 |
|      202 | Rahul      |     102 |
|      203 | Nikhil     |     103 |
+----------+------------+---------+
4 rows in set (0.00 sec)
```

```sql
--inserting values into electricity board table
INSERT INTO electricity_board (board_id,board_name,state,city) VALUES ('300','Chamundeshwari Power Corporation','Karnataka','Mysore');
INSERT INTO electricity_board (board_name,state,city) VALUES ('Karnataka Power Corporation','Karnataka','Bangalore');
INSERT INTO electricity_board (board_name,state,city) VALUES ('BESCOM','Karnataka','Bangalore');
INSERT INTO electricity_board (board_name,state,city) VALUES ('Tamil Nadu Power Corporation','Tamil Nadu','Chennai');
INSERT INTO electricity_board (board_name,state,city) VALUES ('Uttar Pradesh Power Corporation','Uttar Pradesh','Lucknow');
INSERT INTO electricity_board (board_name,state,city) VALUES ('Madhya Pradesh Power Corporation','Madhya Pradesh','Indore');
```

```
+----------+-----------------------------------+----------------+-----------+
| board_id | board_name                        | state          | city      |
+----------+-----------------------------------+----------------+-----------+
|      300 | Chamundeshwari Power Corporation | Karnataka      | Mysore    |
|      301 | Karnataka Power Corporation       | Karnataka      | Bangalore |
|      302 | BESCOM                            | Karnataka      | Bangalore |
|      303 | Tamil Nadu Power Corporation      | Tamil Nadu     | Chennai   |
|      304 | Uttar Pradesh Power Corporation   | Uttar Pradesh  | Lucknow   |
|      305 | Madhya Pradesh Power Corporation  | Madhya Pradesh | Indore    |
+----------+-----------------------------------+----------------+-----------+
5 rows in set (0.00 sec)
```

```sql
--inserting values into tariff table
INSERT INTO tariff (tariff_id,tariff_type,price_slab) VALUES ('400','Power factor tariff','10');
INSERT INTO tariff (tariff_type,price_slab) VALUES ('Peak Load tariff','40');
INSERT INTO tariff (tariff_type,price_slab) VALUES ('Two part tariff','18');
INSERT INTO tariff (tariff_type,price_slab) VALUES ('Three part tariff','36');
```

```
pes2ug21cs294_250>SELECT * from tariff;
+-----------+---------------------+------------+
| tariff_id | tariff_type         | price_slab |
+-----------+---------------------+------------+
|       400 | Power factor tariff |         10 |
|       401 | Peak Load tariff    |         40 |
|       402 | Two part tariff     |         18 |
|       403 | Three part tariff   |         36 |
+-----------+---------------------+------------+
4 rows in set (0.00 sec)
```

```
pes2ug21cs294_250>SELECT * from bill;
+---------+----------+---------+--------------+---------------+-----------------+--------------+------------+
| bill_id | board_id | cust_id | meter_number | monthly_units | amount_per_unit | total_amount | due_date   |
+---------+----------+---------+--------------+---------------+-----------------+--------------+------------+
|     500 |      300 |     100 | 37713        |           105 |              10 |         1050 | 2023-06-30 |
|     504 |      301 |     101 | 22849        |           187 |              18 |         3366 | 2023-12-16 |
|     505 |      303 |     102 | 94853        |            23 |              15 |          345 | 2023-11-09 |
|     506 |      302 |     103 | 36274        |            43 |              22 |          946 | 2023-10-12 |
|     507 |      303 |     104 | 47232        |            57 |              28 |         1596 | 2023-09-15 |
+---------+----------+---------+--------------+---------------+-----------------+--------------+------------+
5 rows in set (0.00 sec)
```

7. QUERIES

7.1 SIMPLE QUERY WITH GROUP BY, AGRREGATE

Query to find the total monthly units consumed by customers in each city:

```
SELECT c.city, SUM(b.monthly_units) AS total_monthly_units
FROM customer c
JOIN bill b ON c.cust_id = b.cust_id
GROUP BY c.city;
```

```
+-------------+----------------------+
| city        | total_monthly_units  |
+-------------+----------------------+
| Mysore      |                  128 |
| Bangalore   |                  384 |
| Pune        |                   43 |
| Chennai     |                   57 |
| Ahmedabad   |                   67 |
| Rameshwaram |                   78 |
| Lucknow     |                   98 |
| Mumbai      |                  118 |
| Indore      |                  128 |
| Kochi       |                  138 |
| Ayodhya     |                  148 |
+-------------+----------------------+
11 rows in set (0.25 sec)
```

```
--query to find the total amount in each state
SELECT c.state, SUM(b.monthly_units) AS total_monthly_units
FROM customer c
JOIN bill b ON c.cust_id = b.cust_id
GROUP BY c.state;
```

```
+----------------+---------------------+
| state          | total_monthly_units |
+----------------+---------------------+
| Karnataka      |                 512 |
| Maharashtra    |                 161 |
| Tamil Nadu     |                 135 |
| Gujarat        |                  67 |
| Uttar Pradesh  |                 246 |
| Madhya Pradesh |                 128 |
| Kerala         |                 138 |
+----------------+---------------------+
7 rows in set (0.10 sec)
```

Query to find the bill amount collected by each electricity board

```sql
SELECT eb.board_id, eb.board_name, SUM(b.total_amount) AS total_bill_amount
FROM electricity_board eb
JOIN bill b ON eb.board_id = b.board_id
GROUP BY eb.board_id, eb.board_name;
```

```
+----------+-----------------------------------+--------------------+
| board_id | board_name                        | total_bill_amount |
+----------+-----------------------------------+--------------------+
|      300 | Chamundeshwari Power Corporation |               8012 |
|      301 | Karnataka Power Corporation      |              13769 |
|      302 | BESCOM                           |              13510 |
|      303 | Tamil Nadu Power Corporation     |              16813 |
|      304 | Uttar Pradesh Power Corporation  |               4802 |
|      305 | Madhya Pradesh Power Corporation |               5832 |
+----------+-----------------------------------+--------------------+
6 rows in set (0.36 sec)
```

7.2 UPDATE OPERATION

```sql
--query to update the address of a customer
UPDATE customer
SET address = 'New Address'
WHERE cust_id = 101;
```

7.3 DELETE OPERATION

```
--query to delete a customer
DELETE FROM customer
WHERE cust_id = 101;
```

7.4 CORRELATED QUERY

```
--correlated query to find customers with bill amount greater than average bill amount for their city
SELECT c.*, b.monthly_units
FROM customer c
JOIN bill b ON c.cust_id = b.cust_id
WHERE b.monthly_units > (
    SELECT AVG(b2.monthly_units)
    FROM bill b2
    JOIN customer c2 ON c2.cust_id = b2.cust_id
    WHERE c2.city = c.city
);
```

```
+---------+--------+--------------------+-----------+-----------+---------------+
| cust_id | name   | address            | city      | state     | monthly_units |
+---------+--------+--------------------+-----------+-----------+---------------+
|     100 | Abhay  | MG Road            | Mysore    | Karnataka |           105 |
|     101 | Vishnu | Basaveshwara Nagar | Bangalore | Karnataka |           187 |
+---------+--------+--------------------+-----------+-----------+---------------+
2 rows in set (0.06 sec)
```

7.5 NESTED QUERY

```
--nested query to find customer with highest bill amount
SELECT cust_id, name, address, city, state
FROM customer
WHERE cust_id = (
    SELECT cust_id
    FROM bill
    GROUP BY cust_id
    ORDER BY SUM(total_amount) DESC
    LIMIT 1
);
```

```
+---------+----------+----------+---------+---------------+
| cust_id | name     | address  | city    | state         |
+---------+----------+----------+---------+---------------+
|     113 | Preetham | AB Block | Ayodhya | Uttar Pradesh |
+---------+----------+----------+---------+---------------+
1 row in set (0.01 sec)
```

8. STORED PROCEDURES, FUCNTIONS AND TRIGGERS

8.1 STORED PROCEDURES OR FUNCTIONS

```
--function to find average monthly units for a city
CREATE FUNCTION GetAverageMonthlyUnitsForCity(cityNameParam VARCHAR(255))
RETURNS DECIMAL(10, 2)
READS SQL DATA
BEGIN
    DECLARE avgMonthlyUnits DECIMAL(10, 2);
    SELECT AVG(b.monthly_units) INTO avgMonthlyUnits
    FROM bill b
    JOIN customer c ON b.cust_id = c.cust_id
    WHERE c.city = cityNameParam;
    RETURN avgMonthlyUnits;
END //
```

```
mysql> SELECT GetAverageMonthlyUnitsForCity('Mysore') AS avg_monthly_units;
+--------------------+
| avg_monthly_units |
+--------------------+
|              64.00 |
+--------------------+
1 row in set (0.41 sec)
```

```
--procedure to find bill details for a customer
DELIMITER //
CREATE PROCEDURE GetBillDetailsByCustomerID(IN custID INT)
BEGIN
    SELECT * FROM bill WHERE cust_id = custID;
END //
DELIMITER ;
```

```
mysql> CALL GetBillDetailsByCustomerID(107);
+---------+----------+---------+--------------+--------------+----------------+--------------+------------+
| bill_id | board_id | cust_id | meter_number | monthly_units | amount_per_unit | total_amount | due_date   |
+---------+----------+---------+--------------+--------------+----------------+--------------+------------+
|     510 |      303 |     107 | 12753        |           89 |             44 |         3920 | 2023-11-24 |
+---------+----------+---------+--------------+--------------+----------------+--------------+------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

8.2 TRIGGERS

```
--trigger to insert bill details into due_bills table when a bill is updated
CREATE TRIGGER IF NOT EXIST due_bills_trigger
AFTER UPDATE ON bill
FOR EACH ROW
BEGIN
IF NEW.due_date < CURDATE()
THEN INSERT INTO due_bills (cust_id, meter_no, units, cost_per_unit, amount, due_date, board_id)
VALUES (NEW.cust_id, NEW.meter_no, NEW.units, NEW.cost_per_unit, NEW.amount, NEW.due_date, NEW.board.id);
END IF;
END;
```

bill_id	board_id	cust_id	meter_number	monthly_units	amount_per_unit	total_amount	due_date
500	300	100	37713	105	10	1050	2023-06-30
505	303	102	94853	23	15	345	2023-11-09
506	302	103	36274	43	22	946	2023-10-12
507	303	104	47232	57	28	1596	2023-09-15
508	301	105	12975	67	33	2211	2023-08-18
511	304	108	75535	98	49	4802	2023-10-27
512	305	109	15821	108	54	5832	2023-09-30

9. FRONT END DEVELOPMENT

The frontend for this electricity bill management system has been created with streamlit. Some parts of the frontend code have been added since the entire code cannot be added here due to its size. The entire source code for this project can be found on the Project's Github Repository.

main.py

```
import streamlit as st
import mysql.connector
from database import *
from customer import *
from admin import *
from bill import *
from tariff import *
from board import *


def main():
    st.title("Electricity management system")

    choose=st.sidebar.radio("Select whether you are customer or admin",["Cus-
tomer","Admin"])
    if choose=="Customer":
        menu=["Home","Customer", "Billing"]
        choice=st.sidebar.selectbox("Menu",menu)
    if choose=="Admin":
        menu=["Home","Customer","Admin", "Billing", "Tariff", "Electricity Boards",
"Show Due Bills", "Custom Query"]
        choice=st.sidebar.selectbox("Menu",menu)


    if choice=="Home":
        st.subheader("Home")
        st.header("Welcome to the electricity bill management system")
        st.write("This is a simple electricity bill management system")
        st.write("Please select a menu option from the sidebar")

    if choice == "Customer":

        st.subheader("Customer details")
        customer_menu=["Add","View","Update","Delete"]
```

```
        customer_choice=st.selectbox("Menu",customer_menu)
        if customer_choice=="Add":
            st.subheader("Enter details")
            create_customer()
        elif customer_choice=="View" :
            view_customer()
        elif customer_choice=="Update" :
            update_customer()
        elif customer_choice=="Delete" :
            delete_customer()

    if choice == "Admin":
        st.subheader("Admin")
        admin_menu=["Add","View","Update","Delete"]
        admin_choice=st.selectbox("Menu",admin_menu)
        if admin_choice=="Add":
            st.subheader("Enter details")
            create_admin()
        elif admin_choice=="View" :
            read_admin()
        elif admin_choice=="Update" :
            update_admin()
        elif admin_choice=="Delete" :
            delete_admin()

    if choice == "Billing":
        st.subheader("Billing")
        billing_menu=["Add Bill","View Bills","Update Bills","Delete Bills"]
        billing_choice=st.selectbox("Menu",billing_menu)
        if billing_choice=="Add Bill":
            create_bill()
        elif billing_choice=="View Bills" :
            read_bill()
        elif billing_choice=="Update Bills" :
            update_bill()
        elif billing_choice=="Delete Bills" :
            delete_bill()

    if choice == "Tariff":
        st.subheader("Tariff")
        tariff_menu=["Add","View","Update","Delete"]
        tariff_choice=st.selectbox("Menu",tariff_menu)
        if tariff_choice=="Add":
            create_tariff()
        elif tariff_choice=="View" :
            read_tariff()
        elif tariff_choice=="Update" :
```

```
            update_tariff()
        elif tariff_choice=="Delete" :
            delete_tariff()

    if choice == "Electricity Boards":
        st.subheader("Electricity Boards")
        eb_menu=["Add","View","Update","Delete"]
        eb_choice=st.selectbox("Menu",eb_menu)
        if eb_choice=="Add":
            create_eb()
        elif eb_choice=="View" :
            read_eb()
        elif eb_choice=="Update" :
            update_eb()
        elif eb_choice=="Delete" :
            delete_eb()

    if choice =="Show Due Bills":
        st.subheader("Due Bills")
        due_bills()

    if choice == "Custom Query":
        st.subheader("Custom Query")
        query=st.text_input("Enter query")
        submit=st.button("Submit")
        if submit:
            mycursor.execute(query)
            myresult = mycursor.fetchall()
            for x in myresult:
                st.write(x)


main()
```

database.py

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

mydb = mysql.connector.connect(user="root", password="246810", host="localhost")
mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE IF NOT EXISTS electricity")
mycursor.execute("USE electricity")

def add_customer(name,address,city,state):
```

```
    mycursor.execute("CREATE TABLE IF NOT EXISTS customer (cust_id INT AUTO_INCREMENT
PRIMARY KEY, name VARCHAR(255), address VARCHAR(255), city VARCHAR(255), state VAR-
CHAR(255))")
    sql = "INSERT INTO customer (name, address, city, state) VALUES (%s, %s, %s, %s)"
    val = (name, address, city, state)
    mycursor.execute(sql, val)
    mydb.commit()

    return mycursor.lastrowid

def read_customer():
    mycursor.execute("SELECT * FROM customer")
    myresult = mycursor.fetchall()
    st.write("Customer details")
    df=pd.DataFrame(myresult,columns=['cust_id','name','address','city','state'])
    st.dataframe(df)


def delete_customer_db(cust_id):
    #check if cust_id exists
    mycursor.execute("SELECT * FROM customer WHERE cust_id = %s",(cust_id,))
    myresult = mycursor.fetchall()
    if not myresult:
        st.write("cust_id does not exist")
        return
    sql = "DELETE FROM customer WHERE cust_id = %s"
    val = (cust_id,)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Customer deleted successfully")

def update_customer_db(cust_id,name,address,city,state):
    sql = "UPDATE customer SET name = %s, address = %s, city = %s, state = %s WHERE
cust_id = %s"
    val = (name, address, city, state, cust_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Customer updated successfully")

def add_admin(name,Customer_id):
    mycursor.execute("CREATE TABLE IF NOT EXISTS admin (admin_id INT AUTO_INCREMENT
PRIMARY KEY, name VARCHAR(255), Customer_id INT NOT NULL, FOREIGN KEY (Customer_id)
REFERENCES customer(cust_id))")
    sql = "INSERT INTO admin (name, Customer_id) VALUES (%s, %s)"
    val = (name, Customer_id)
    mycursor.execute(sql, val)
    mydb.commit()
```

```
    return mycursor.lastrowid

def read_admin():
    mycursor.execute("SELECT * FROM admin")
    myresult = mycursor.fetchall()
    st.write("Admin details")
    df=pd.DataFrame(myresult,columns=['admin_id','name','Customer_id'])
    st.dataframe(df)


def update_admin_db(admin_id,name,Customer_id):
    sql = "UPDATE admin SET name = %s, Customer_id = %s WHERE admin_id = %s"
    val = (name, Customer_id, admin_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Admin updated successfully")

def delete_admin_db(admin_id):
    #check if admin_id exists
    mycursor.execute("SELECT * FROM admin WHERE admin_id = %s",(admin_id,))
    myresult = mycursor.fetchall()
    if not myresult:
        st.write("admin_id does not exist")
        return
    sql = "DELETE FROM admin WHERE admin_id = %s"
    val = (admin_id,)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Admin deleted successfully")

def add_bill(cust_id,meter_no,units,cost_per_unit,due_date,board_id):
    mycursor.execute("CREATE TABLE IF NOT EXISTS bill (bill_id INT AUTO_INCREMENT PRI-
MARY KEY, cust_id INT NOT NULL, meter_no INT NOT NULL, units INT NOT NULL,
cost_per_unit INT NOT NULL, amount INT NOT NULL, due_date DATE NOT NULL board_id INT
NOT NULL, FOREIGN KEY (cust_id) REFERENCES customer(cust_id), FOREIGN KEY (board_id)
REFERENCES Board(eb_id))")
    sql = "INSERT INTO bill (cust_id, meter_no, units, cost_per_unit, amount,
due_date) VALUES (%s, %s, %s, %s, %s, %s)"
    val = (cust_id, meter_no, units, cost_per_unit, int(units)*int(cost_per_unit),
due_date, board_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Bill added successfully")
    return mycursor.lastrowid

def read_bill():
    mycursor.execute("SELECT * FROM bill")
```

```
    myresult = mycursor.fetchall()
    st.write("Bill details")
    df=pd.DataFrame(myresult,columns=['bill_id','cust_id','me-
ter_no','units','cost_per_unit','amount','due_date', 'board_id'])
    st.dataframe(df)


def update_bill_db(bill_id,cust_id,meter_no,units,cost_per_unit, due_date, board_id):
    sql = "UPDATE bill SET cust_id = %s, meter_no = %s, units = %s, cost_per_unit =
%s, amount = %s, due_date=%s, board_id=%s WHERE bill_id = %s"
    val = (cust_id, meter_no, units, cost_per_unit, int(units)*int(cost_per_unit),
due_date, board_id, bill_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Bill updated successfully")

def delete_bill_db(bill_id):
    #check if bill_id exists
    mycursor.execute("SELECT * FROM bill WHERE bill_id = %s",(bill_id,))
    myresult = mycursor.fetchall()
    if not myresult:
        st.write("bill_id does not exist")
        return
    sql = "DELETE FROM bill WHERE bill_id = %s"
    val = (bill_id,)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Bill deleted successfully")

def add_tariff(tariff_type,tariff_cost):
    mycursor.execute("CREATE TABLE IF NOT EXISTS tariff (tariff_id INT AUTO_INCREMENT
PRIMARY KEY, tariff_type VARCHAR(255), tariff_cost INT NOT NULL)")
    sql = "INSERT INTO tariff (tariff_type, tariff_cost) VALUES (%s, %s)"
    val = (tariff_type, tariff_cost)
    mycursor.execute(sql, val)
    mydb.commit()
    return mycursor.lastrowid

def read_tariff():
    mycursor.execute("SELECT * FROM tariff")
    myresult = mycursor.fetchall()
    st.write("Tariff details")
    df=pd.DataFrame(myresult,columns=['tariff_id','tariff_type','tariff_cost'])
    st.dataframe(df)

def update_tariff_db(tariff_id,tariff_type,tariff_cost):
    sql = "UPDATE tariff SET tariff_type = %s, tariff_cost = %s WHERE tariff_id = %s"
```

```
    val = (tariff_type, tariff_cost, tariff_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Tariff updated successfully")

def delete_tariff_db(tariff_id):
    sql = "DELETE FROM tariff WHERE tariff_id = %s"
    val = (tariff_id,)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Tariff deleted successfully")

def add_eb(name,city,state):
    mycursor.execute("CREATE TABLE IF NOT EXISTS Board (eb_id INT AUTO_INCREMENT PRI-
MARY KEY, name VARCHAR(255), city VARCHAR(255), state VARCHAR(255))")
    # mycursor.execute("INSERT INTO Board (eb_id, name, city, state) VALUES
('100','Karnataka Power Transmission Corporation Limited', 'Bengaluru', 'Karnataka')")
    sql = "INSERT INTO Board (name, city, state) VALUES (%s, %s, %s)"
    val = (name, city, state)
    mycursor.execute(sql, val)
    mydb.commit()
    return mycursor.lastrowid

def read_eb():
    mycursor.execute("SELECT * FROM Board")
    myresult = mycursor.fetchall()
    st.write("Electricity board details")
    df=pd.DataFrame(myresult,columns=['eb_id','name','city','state'])
    st.dataframe(df)

def update_eb_db(eb_id,name,city,state):
    sql = "UPDATE Board SET name = %s, city = %s, state = %s WHERE eb_id = %s"
    val = (name, city, state, eb_id)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Electricity board updated successfully")

def delete_eb_db(eb_id):
    sql = "DELETE FROM Bpard WHERE eb_id = %s"
    val = (eb_id,)
    mycursor.execute(sql, val)
    mydb.commit()
    st.write("Electricity board deleted successfully")
```

10. REFERENCES

[1] https://docs.streamlit.io/
[2] https://dev.mysql.com/doc/connector-python/en/