**ELECTRICITY MANAGEMENT SYSTEM**

MINI PROJECT REPORT

for

21CSS201T - COMPUTER ORGANIZATION AND ARCHITECTURE

*Submitted by*

Ragini Rai [RA2311026011074]

Addhyan Taneja[RA2311026011075]

Shivangi Tripathi [RA2311026011081]

Venkata Rao Dechiraju [RA23110260111122]

*Under the Guidance of*

Dr. Maivizhi R

(Assistant Professor, Department of Computational Intelligence)

*In partial fulfillment of the requirements for the degree of*

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

A logo with blue text

AI-generated content may be incorrect.

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

NOVEMBER 2024

**Introduction**

The Electricity Bill System is a modern web-based application created to simplify how electricity bills are managed for both customers and the people running the system, called administrators. Electricity is something we all use daily—at home, in offices, or even in schools—and keeping track of its usage, billing, and payments can be a big task. In the past, this was done with paper records, which was slow and often full of mistakes. This project changes that by bringing everything online, making it faster, easier, and more accurate.

The system was built using simple tools: PHP for the website’s logic, MySQL to store all the data in an organized way, and CSS to give it a clean, professional look that’s easy on the eyes. Customers can log in to check their bills, pay them with a few clicks, see how much electricity they’ve used, and even tell the company if something’s wrong—like a high bill or a power issue. Administrators, on the other hand, can add meter readings, set prices for electricity (called tariffs), create bills, and respond to customer feedback or complaints—all from one place. The goal was to make a system that saves time, cuts down on errors, and improves how electricity services work for everyone.

This project started as a way to solve real problems faced by electricity companies and their customers. It’s not just about bills—it’s about making life simpler and more connected. Whether it’s a customer in a small village or an admin in a busy office, this system brings convenience and control right to their fingertips.

**Objectives**

The Electricity Bill System was designed with clear goals in mind to fix problems and make things better. Here are the main objectives explained in detail:

1. **Automate the Billing Process:**

We wanted to stop doing calculations by hand and let the system figure out bills automatically using meter readings and tariffs. This saves time and makes sure the amounts are always right.

1. **Give Customers Easy Access:**

Customers should be able to log in anytime, anywhere, to see their bills, pay them online, check their usage history, or report a problem—without needing to visit an office or wait in line.

1. **Help Admins Manage Efficiently:**

Admins need tools to handle everything—like adding how much electricity a customer used, setting prices, making bills, and fixing customer issues—all in a simple way so they can work fast and focus on helping people.

1. **Make Billing More Accurate:**

Mistakes in bills (like charging too much or too little) happen a lot with paper systems. This project uses a database to store and calculate everything, so there are fewer errors and customers trust the bills they get.

1. **Create a Simple and Friendly Design:**

Not everyone is good with computers, so we aimed to make the system easy to use for both customers and admins. The buttons, pages, and options are clear and straightforward, so anyone can use it without confusion.

1. **Keep Records Organized:**

We wanted all the data—like customer details, payments, and usage—to be stored neatly in one place. This makes it easy to find old records or check what’s happening now, without losing anything.

1. **Improve Communication:**

The system sends notifications to customers about new bills or updates and lets admins see feedback or complaints right away. This keeps everyone connected and informed.

**Problem Statement**

Before this Electricity Bill System, managing electricity bills was a big headache for both customers and companies. Here’s a detailed look at the problems we wanted to solve:

1. **Too Much Manual Work:**

People had to write down meter readings, calculate bills by hand, and keep paper records. This took hours or even days, and it was easy to make mistakes—like adding numbers wrong or forgetting a bill.

1. **Hard for Customers:**

If someone wanted to pay a bill or ask about it, they had to go to an office, wait in long lines, or call someone who might not answer. This was tough for people who live far away or have busy lives.

1. **No Way to Track History:**

Customers couldn’t easily see their old bills, how much they paid before, or how much electricity they used over time. This made it hard to plan their money or spot problems like a sudden high bill.

1. **Slow Updates and Delays:**

When prices changed (like a new tariff) or a meter was read, it took a long time to update the bills. Sometimes customers got bills late, which caused confusion and late payments.

1. **Bad Communication:**

Customers didn’t know when a new bill came or if their complaint was being fixed. Admins also couldn’t see what customers thought unless they called or wrote a letter, which wasn’t quick or easy.

1. **Lost or Messy Records:**

Paper files could get lost, torn, or mixed up. There was no single place to keep everything safe, so finding old data—like a payment from last year—was a struggle.

1. **Errors Cost Money:**

Mistakes in bills meant customers might pay too much or too little, and fixing them took extra effort. This wasted time and sometimes upset customers who felt cheated.

This project tackles all these issues by moving everything online, making it automatic, and giving both customers and admins better tools to work with.

**Scope**.

1. **Customer Features:**

This system lets customers sign up with their details, log in using an email and password, and use a personal dashboard to manage their electricity account. They can see their current and past bills, pay them online with a few clicks, check their payment history to see what they’ve paid before, and look at how much electricity they’ve used over time. They can also send complaints if something’s wrong (like a power cut), give feedback about the service, or ask for help if they have a question or problem. This makes it super easy for customers because they don’t need to leave home or wait for paper bills—they can do everything on their computer or phone whenever they want.

1. **Admin Features:**

Admins get their own login and a dashboard with tools to run the system smoothly. They can add meter readings to track how much electricity each customer uses, set tariffs (the price of electricity), and generate bills automatically for everyone. They can also see all customer details—like names and addresses—plus all bills, payments, feedback, complaints, and support requests. If a customer has an issue, admins can mark it as fixed with one click. This gives admins full control to update things fast and help customers without needing piles of paperwork or complicated steps.

1. **Database Storage:**

Everything in the system—like customer names, meter readings, bills, and payments—is kept in a database, which is like a big digital box that organizes all the information. It can store lots of data safely and pull it up quickly when someone needs to see it, like finding a customer’s bill history in seconds. This keeps everything in one place, so nothing gets lost, and it helps the system work fast and stay reliable, whether it’s for one customer or a thousand.

1. **Billing Automation:**

The system does the hard work of making bills by itself. It takes the meter readings (how much electricity was used) and multiplies them by the tariffs (the price per unit) to figure out the bill amount, then creates the bill without anyone needing to do math by hand. It also adds a penalty fee if a bill isn’t paid by the due date, so late payments are handled fairly. This saves time for admins and makes sure every bill is correct and ready to go when it’s supposed to be.

1. **Notifications:**  
   Customers get messages called notifications that tell them important things, like when a new bill is ready, when it’s due, or if their complaint has been fixed. Admins can see these updates too, so they know what’s happening with each customer. This keeps everyone connected—customers don’t have to guess about their bills, and admins can act fast if something needs attention, like responding to a customer’s question.
2. **Limits of the System:**

This system focuses on digital tasks, not physical ones, so it doesn’t handle things like fixing broken meters or setting them up in homes—that’s for electricians, not the website. It needs an internet connection to work, so if someone doesn’t have internet, they can’t use it. Right now, it only shows messages online and doesn’t send emails or text messages to phones, which could be added later. This means it’s great for managing bills and accounts but isn’t built for everything an electricity company might do.

1. **Future Possibilities:**

There’s room to make the system even better down the road. For example, we could add a feature to send bill reminders by email or text message, let customers download their bills as PDF files to save or print, or show them graphs of their electricity usage to help them save energy. These aren’t part of the system now, but they’re ideas for the future that could make it more helpful. For this project, we kept it focused on the main tasks and made sure those work really well.

**System Requirements**

**Software Requirements**

These are the tools and programs needed to build and run the system:

**Operating System:** Windows, macOS, or Linux (any system that can run a web server). The system works on any computer, so it’s flexible.

**Web Server:** XAMPP, WAMP, or any server with Apache. This runs the website on your computer or a server.

**Programming Language:** PHP (version 7 or higher). PHP makes the website work and talk to the database.

**Database:** MySQL (version 5.6 or higher). MySQL stores all the data like customer info and bills.

**Web Browser:** Chrome, Firefox, or any modern browser. Users need a browser to open the website.

**Text Editor:** VS Code, Notepad++, or any editor to write code. This is for editing the PHP and CSS files.

**Network Requirements**

These are the network needs to use the system:

**Internet Connection:** A stable connection with at least 1 Mbps speed. Users need internet to log in and use the system online.

**Local Server (Optional):** If testing locally, no internet is needed—just a local network. For development, you can run it on your own computer without internet.

**Security:** Basic HTTPS (optional for live use) to keep data safe. This protects customer details when the system goes online.

**Entities and Attributes**

Entities are the main “things” in the system, and attributes are their details. Here’s a list in simple terms:

1. **Customer**

* cust\_id
* cust\_name
* address
* email
* password

1. **Admin**

* admin\_id
* login\_id
* password

1. **Account**

* account\_id
* cust\_id
* account\_number
* status

1. **Meter**

* meter\_id
* cust\_id
* meter\_number
* installation\_date

1. **Meter\_Readings**

* reading\_id
* meter\_id
* reading\_date
* units\_consumed

1. **Tariff**

* tariff\_id
* category
* rate

1. **Bills**

* bill\_id
* cust\_id
* account\_id
* amount
* bill\_date
* due\_date
* status

1. **Payments**

* payment\_id
* cust\_id
* bill\_id
* amount
* payment\_date

1. **Payment\_History**

* history\_id
* payment\_id
* cust\_id
* amount
* payment\_date

1. **Penalty**

* penalty\_id
* bill\_id
* penalty\_amount
* penalty\_date

1. **Complaint**

* complaint\_id
* cust\_id
* complaint\_text
* complaint\_date
* status

1. **Feedback**

* feedback\_id
* cust\_id
* feedback\_text
* feedback\_date

1. **Usage\_History**

* usage\_id
* cust\_id
* meter\_id
* usage\_date
* units\_consumed

1. **Notification**

* notification\_id
* cust\_id
* message
* notification\_date
* status

1. **Invoice**

* invoice\_id
* bill\_id
* cust\_id
* amount\_due
* issue\_date

1. **Customer\_Support**

* support\_id
* cust\_id
* issue\_description
* support\_date
* resolution\_status

**Entity Relationship**

1. **A Customer can have one Account, and each Account belongs to one Customer.**

(One-to-One: Customer ↔ Account, managed through cust\_id as a foreign key in the Account table)

This means every customer gets their own account to track their electricity bills, and each account is linked to just one customer using the customer’s ID.

1. **A Customer can have one Meter, and each Meter belongs to one Customer.**

(One-to-One: Customer ↔ Meter, managed through cust\_id as a foreign key in the Meter table)

Every customer has a single meter to measure their electricity usage, and that meter is tied to them using their customer ID.

1. **A Meter can have multiple Meter\_Readings, and each Meter\_Reading belongs to one Meter.**

(One-to-Many: Meter → Meter\_Readings, managed through meter\_id as a foreign key in the Meter\_Readings table)

A meter can be checked many times to see how much electricity was used, and each reading is connected to that meter with its ID.

1. **A Customer can have multiple Bills, and each Bill belongs to one Customer.**

(One-to-Many: Customer → Bills, managed through cust\_id as a foreign key in the Bills table)

A customer gets a new bill every month or cycle, and each bill is linked to them using their customer ID.

1. **An Account can have multiple Bills, and each Bill belongs to one Account.**

(One-to-Many: Account → Bills, managed through account\_id as a foreign key in the Bills table)

An account can have many bills over time, and each bill is tied to that account with its ID.

1. **A Bill can have one Payment, and each Payment belongs to one Bill.**

(One-to-One: Bill ↔ Payments, managed through bill\_id as a foreign key in the Payments table)

When a customer pays a bill, that payment is linked to just that one bill using the bill’s ID.

1. **A Customer can have multiple Payments, and each Payment belongs to one Customer.**

(One-to-Many: Customer → Payments, managed through cust\_id as a foreign key in the Payments table)

A customer can make many payments over time, and each payment is connected to them with their customer ID.

1. **A Payment can have one Payment\_History record, and each Payment\_History record belongs to one Payment.**

(One-to-One: Payments ↔ Payment\_History, managed through payment\_id as a foreign key in the Payment\_History table)

Every payment gets saved in the history, and each history record is linked to that payment using its ID.

1. **A Customer can have multiple Payment\_History records, and each Payment\_History record belongs to one Customer.**

(One-to-Many: Customer → Payment\_History, managed through cust\_id as a foreign key in the Payment\_History table)

A customer’s payment history can grow with many records, and each one is tied to them with their customer ID.

1. **A Bill can have one Penalty, and each Penalty belongs to one Bill.**

(One-to-One: Bill ↔ Penalty, managed through bill\_id as a foreign key in the Penalty table)

If a bill is paid late, it gets one penalty fee, and that penalty is connected to the bill with its ID.

1. **A Customer can have multiple Complaints, and each Complaint belongs to one Customer.**

(One-to-Many: Customer → Complaint, managed through cust\_id as a foreign key in the Complaint table)

A customer can report many problems, and each complaint is linked to them using their customer ID.

1. **A Customer can have multiple Feedback entries, and each Feedback entry belongs to one Customer.**

(One-to-Many: Customer → Feedback, managed through cust\_id as a foreign key in the Feedback table)

A customer can give feedback many times, and each piece of feedback is tied to them with their customer ID.

1. **A Customer can have multiple Usage\_History records, and each Usage\_History record belongs to one Customer.**

(One-to-Many: Customer → Usage\_History, managed through cust\_id as a foreign key in the Usage\_History table)

A customer’s past electricity usage is tracked with many records, and each one is linked to them using their customer ID.

1. **A Meter can have multiple Usage\_History records, and each Usage\_History record belongs to one Meter.**

(One-to-Many: Meter → Usage\_History, managed through meter\_id as a foreign key in the Usage\_History table)

A meter’s usage history can have many entries, and each entry is connected to that meter with its ID.

1. **A Customer can have multiple Notifications, and each Notification belongs to one Customer.**

(One-to-Many: Customer → Notification, managed through cust\_id as a foreign key in the Notification table)

A customer can get many messages about bills or updates, and each message is linked to them with their customer ID.

1. **A Bill can have one Invoice, and each Invoice belongs to one Bill.**

(One-to-One: Bill ↔ Invoice, managed through bill\_id as a foreign key in the Invoice table)

Every bill gets one formal invoice record, and that invoice is tied to the bill using its ID.

1. **A Customer can have multiple Invoices, and each Invoice belongs to one Customer.**

(One-to-Many: Customer → Invoice, managed through cust\_id as a foreign key in the Invoice table)

A customer can have many invoices over time, and each one is linked to them with their customer ID.

1. **A Customer can have multiple Customer\_Support requests, and each Customer\_Support request belongs to one Customer.**

(One-to-Many: Customer → Customer\_Support, managed through cust\_id as a foreign key in the Customer\_Support table)

A customer can ask for help many times, and each support request is connected to them using their customer ID.

**Tables**

**1. Account**

CREATE TABLE Account (

account\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

account\_number VARCHAR(50) UNIQUE NOT NULL,

status ENUM('Active', 'Inactive') DEFAULT 'Active',

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Account (cust\_id, account\_number, status) VALUES

(1, 'ACC000001', 'Active'), (2, 'ACC000002', 'Active'), (3, 'ACC000003', 'Active'),

(4, 'ACC000004', 'Active'), (5, 'ACC000005', 'Active'), (6, 'ACC000006', 'Active'),

(7, 'ACC000007', 'Active'), (8, 'ACC000008', 'Active'), (9, 'ACC000009', 'Active'),

(10, 'ACC000010', 'Active'), (11, 'ACC000011', 'Active'), (12, 'ACC000012', 'Active'),

(13, 'ACC000013', 'Active'), (14, 'ACC000014', 'Active'), (15, 'ACC000015', 'Active');

A screen shot of a computer screen

AI-generated content may be incorrect.

**2. Admin**

CREATE TABLE Admin (

admin\_id INT PRIMARY KEY AUTO\_INCREMENT,

login\_id VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL

);  
  
INSERT INTO Admin (login\_id, password) VALUES

('admin1@example.com', 'admin123'),

('admin2@example.com', 'admin456'),

('admin3@example.com', 'admin789'),

('admin4@example.com', 'admin101'),

('admin5@example.com', 'admin202'),

('admin6@example.com', 'admin303'),

('admin7@example.com', 'admin404'),

('admin8@example.com', 'admin505'),

('admin9@example.com', 'admin606'),

('admin10@example.com', 'admin707'),

('admin11@example.com', 'admin808'),

('admin12@example.com', 'admin909'),

('admin13@example.com', 'admin111'),

('admin14@example.com', 'admin222'),

('admin15@example.com', 'admin333');



**3. Bills**

CREATE TABLE Bills (

bill\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

account\_id INT,

amount DECIMAL(10, 2) NOT NULL,

bill\_date DATE NOT NULL,

due\_date DATE NOT NULL,

status ENUM('Pending', 'Paid') DEFAULT 'Pending',

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id),

FOREIGN KEY (account\_id) REFERENCES Account(account\_id),

CONSTRAINT chk\_amount CHECK (amount > 0)

);

INSERT INTO Bills (cust\_id, account\_id, amount, bill\_date, due\_date, status) VALUES

(1, 1, 15.00, '2025-02-01', '2025-02-15', 'Pending'), (2, 2, 18.00, '2025-02-01', '2025-02-15', 'Pending'),

(3, 3, 13.50, '2025-02-01', '2025-02-15', 'Pending'), (4, 4, 16.50, '2025-02-01', '2025-02-15', 'Pending'),

(5, 5, 19.50, '2025-02-01', '2025-02-15', 'Pending'), (6, 6, 14.25, '2025-02-01', '2025-02-15', 'Pending'),

(7, 7, 17.25, '2025-02-01', '2025-02-15', 'Pending'), (8, 8, 18.75, '2025-02-01', '2025-02-15', 'Pending'),

(9, 9, 15.75, '2025-02-01', '2025-02-15', 'Pending'), (10, 10, 20.25, '2025-02-01', '2025-02-15', 'Pending'),

(11, 11, 12.75, '2025-02-01', '2025-02-15', 'Pending'), (12, 12, 21.00, '2025-02-01', '2025-02-15', 'Pending'),

(13, 13, 22.50, '2025-02-01', '2025-02-15', 'Pending'), (14, 14, 12.00, '2025-02-01', '2025-02-15', 'Pending'),

(15, 15, 21.75, '2025-02-01', '2025-02-15', 'Pending');

A black and white screen with numbers

AI-generated content may be incorrect.

**4. Complaint**

CREATE TABLE Complaint (

complaint\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

complaint\_text TEXT NOT NULL,

complaint\_date DATE NOT NULL,

status ENUM('Pending', 'Resolved') DEFAULT 'Pending',

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Complaint (cust\_id, complaint\_text, complaint\_date, status) VALUES

(1, 'Power outage', '2025-02-01', 'Pending'), (2, 'High bill', '2025-02-02', 'Resolved'),

(3, 'Meter issue', '2025-02-03', 'Pending'), (4, 'Billing error', '2025-02-04', 'Resolved'),

(5, 'No supply', '2025-02-05', 'Pending'), (6, 'Overcharge', '2025-02-06', 'Resolved'),

(7, 'Frequent outages', '2025-02-07', 'Pending'), (8, 'Meter reading error', '2025-02-08', 'Resolved'),

(9, 'Service delay', '2025-02-09', 'Pending'), (10, 'Wrong bill', '2025-02-10', 'Resolved'),

(11, 'Power surge', '2025-02-11', 'Pending'), (12, 'Billing dispute', '2025-02-12', 'Resolved'),

(13, 'No response', '2025-02-13', 'Pending'), (14, 'Meter fault', '2025-02-14', 'Resolved'),

(15, 'High rates', '2025-02-15', 'Pending');

A screenshot of a computer error

AI-generated content may be incorrect.

**5. Customer**  
  
CREATE TABLE Customer (

cust\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_name VARCHAR(100) NOT NULL,

address VARCHAR(255) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL

);  
  
INSERT INTO Customer (cust\_name, address, email, password) VALUES

('John Doe', '123 Main St', 'john@example.com', 'pass123'),

('Jane Smith', '456 Oak St', 'jane@example.com', 'pass456'),

('Alice Brown', '789 Pine St', 'alice@example.com', 'pass789'),

('Bob Johnson', '101 Elm St', 'bob@example.com', 'pass101'),

('Carol White', '202 Birch St', 'carol@example.com', 'pass202'),

('David Lee', '303 Cedar St', 'david@example.com', 'pass303'),

('Eve Black', '404 Maple St', 'eve@example.com', 'pass404'),

('Frank Green', '505 Spruce St', 'frank@example.com', 'pass505'),

('Grace Hill', '606 Willow St', 'grace@example.com', 'pass606'),

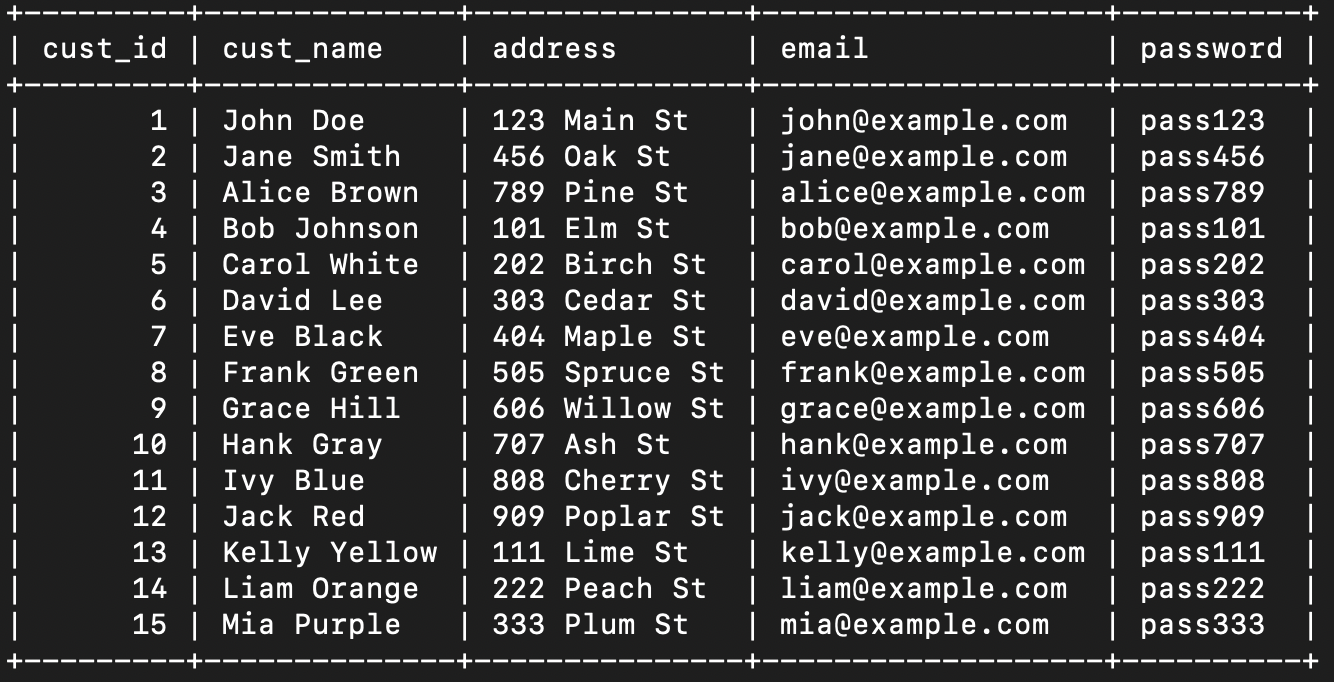
('Hank Gray', '707 Ash St', 'hank@example.com', 'pass707'),

('Ivy Blue', '808 Cherry St', 'ivy@example.com', 'pass808'),

('Jack Red', '909 Poplar St', 'jack@example.com', 'pass909'),

('Kelly Yellow', '111 Lime St', 'kelly@example.com', 'pass111'),

('Liam Orange', '222 Peach St', 'liam@example.com', 'pass222'),

('Mia Purple', '333 Plum St', 'mia@example.com', 'pass333');  
  


**6. Customer\_Support**

CREATE TABLE Customer\_Support (

support\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

issue\_description TEXT NOT NULL,

support\_date DATE NOT NULL,

resolution\_status ENUM('Open', 'Closed') DEFAULT 'Open',

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Customer\_Support (cust\_id, issue\_description, support\_date, resolution\_status) VALUES

(1, 'Billing query', '2025-02-01', 'Open'), (2, 'Meter failure', '2025-02-02', 'Closed'),

(3, 'Payment issue', '2025-02-03', 'Open'), (4, 'Service disruption', '2025-02-04', 'Closed'),

(5, 'High bill complaint', '2025-02-05', 'Open'), (6, 'Support delay', '2025-02-06', 'Closed'),

(7, 'Outage report', '2025-02-07', 'Open'), (8, 'Billing error', '2025-02-08', 'Closed'),

(9, 'Meter reading dispute', '2025-02-09', 'Open'), (10, 'Payment not updated', '2025-02-10', 'Closed'),

(11, 'Service request', '2025-02-11', 'Open'), (12, 'Penalty query', '2025-02-12', 'Closed'),

(13, 'Tariff question', '2025-02-13', 'Open'), (14, 'Outage follow-up', '2025-02-14', 'Closed'),

(15, 'General inquiry', '2025-02-15', 'Open');

A screenshot of a computer program

AI-generated content may be incorrect.

**7. Feedback**

CREATE TABLE Feedback (

feedback\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

feedback\_text TEXT NOT NULL,

feedback\_date DATE NOT NULL,

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Feedback (cust\_id, feedback\_text, feedback\_date) VALUES

(1, 'Good service', '2025-02-01'), (2, 'Slow response', '2025-02-02'),

(3, 'Friendly staff', '2025-02-03'), (4, 'Needs improvement', '2025-02-04'),

(5, 'Reliable power', '2025-02-05'), (6, 'Billing issues', '2025-02-06'),

(7, 'Great support', '2025-02-07'), (8, 'Poor communication', '2025-02-08'),

(9, 'Fast fixes', '2025-02-09'), (10, 'High costs', '2025-02-10'),

(11, 'Excellent', '2025-02-11'), (12, 'Average service', '2025-02-12'),

(13, 'Very good', '2025-02-13'), (14, 'Bad experience', '2025-02-14'),

(15, 'Satisfactory', '2025-02-15');

A screenshot of a computer program

AI-generated content may be incorrect.

**8. Invoice**

CREATE TABLE Invoice (

invoice\_id INT PRIMARY KEY AUTO\_INCREMENT,

bill\_id INT,

cust\_id INT,

amount\_due DECIMAL(10, 2) NOT NULL,

issue\_date DATE NOT NULL,

FOREIGN KEY (bill\_id) REFERENCES Bills(bill\_id),

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Invoice (bill\_id, cust\_id, amount\_due, issue\_date) VALUES

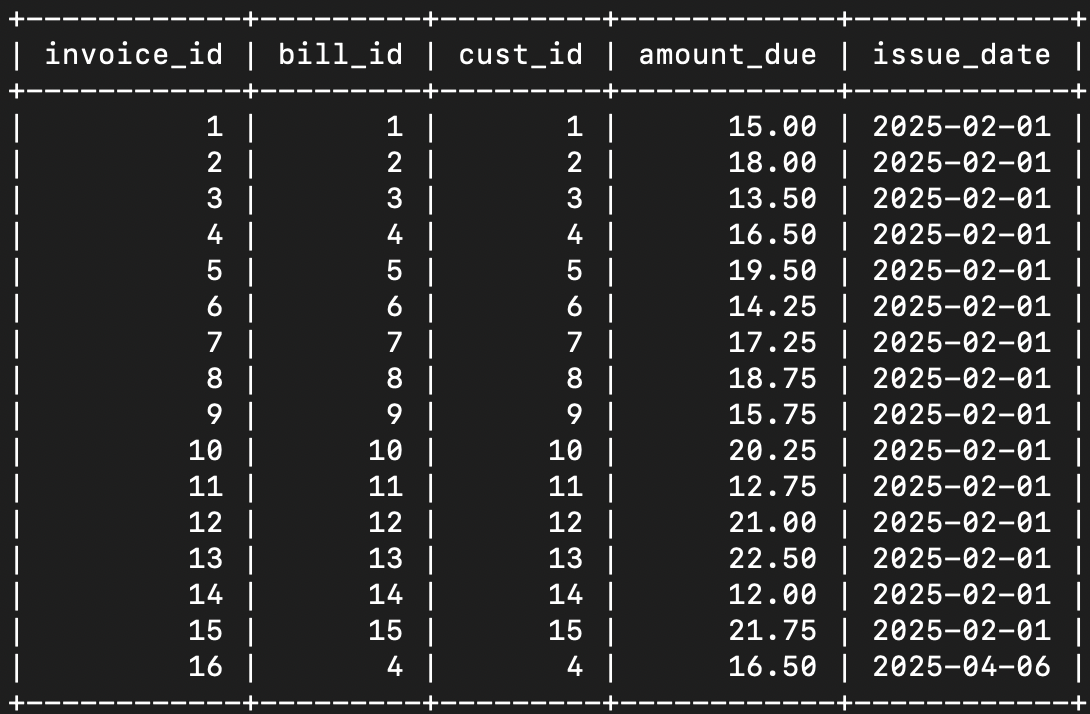
(1, 1, 15.00, '2025-02-01'), (2, 2, 18.00, '2025-02-01'), (3, 3, 13.50, '2025-02-01'),

(4, 4, 16.50, '2025-02-01'), (5, 5, 19.50, '2025-02-01'), (6, 6, 14.25, '2025-02-01'),

(7, 7, 17.25, '2025-02-01'), (8, 8, 18.75, '2025-02-01'), (9, 9, 15.75, '2025-02-01'),

(10, 10, 20.25, '2025-02-01'), (11, 11, 12.75, '2025-02-01'), (12, 12, 21.00, '2025-02-01'),

(13, 13, 22.50, '2025-02-01'), (14, 14, 12.00, '2025-02-01'), (15, 15, 21.75, '2025-02-01');



**9. Meter**

CREATE TABLE Meter (

meter\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

meter\_number VARCHAR(50) UNIQUE NOT NULL,

installation\_date DATE NOT NULL,

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

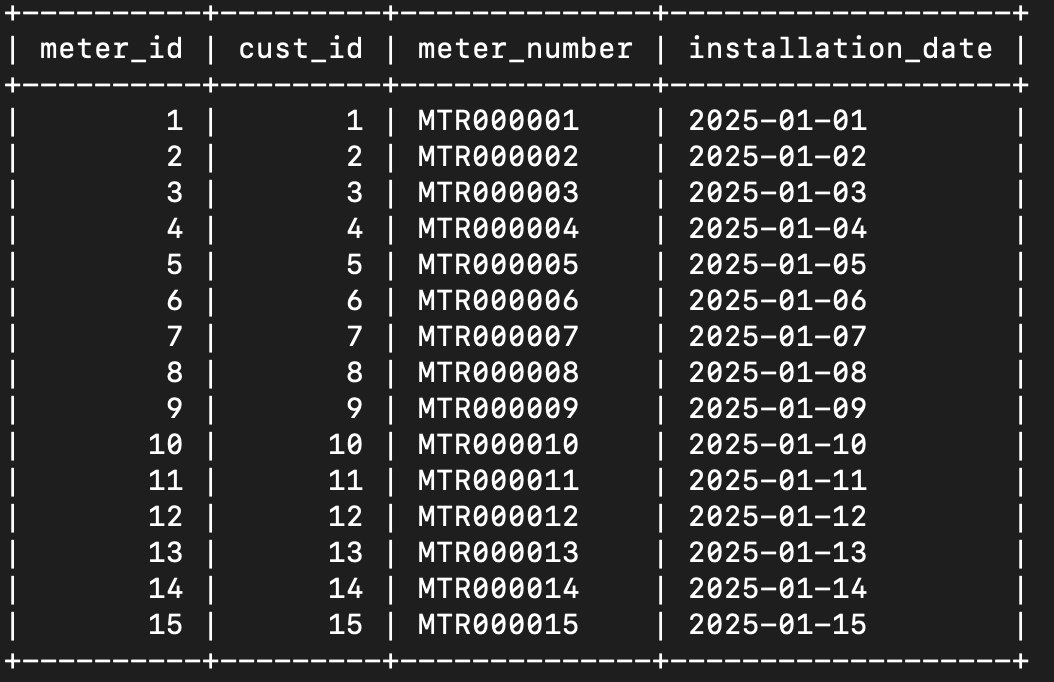
INSERT INTO Meter (cust\_id, meter\_number, installation\_date) VALUES

(1, 'MTR000001', '2025-01-01'), (2, 'MTR000002', '2025-01-02'), (3, 'MTR000003', '2025-01-03'),

(4, 'MTR000004', '2025-01-04'), (5, 'MTR000005', '2025-01-05'), (6, 'MTR000006', '2025-01-06'),

(7, 'MTR000007', '2025-01-07'), (8, 'MTR000008', '2025-01-08'), (9, 'MTR000009', '2025-01-09'),

(10, 'MTR000010', '2025-01-10'), (11, 'MTR000011', '2025-01-11'), (12, 'MTR000012', '2025-01-12'),

(13, 'MTR000013', '2025-01-13'), (14, 'MTR000014', '2025-01-14'), (15, 'MTR000015', '2025-01-15'); 

**10. Meter\_Readings**

CREATE TABLE Meter\_Readings (

reading\_id INT PRIMARY KEY AUTO\_INCREMENT,

meter\_id INT,

reading\_date DATE NOT NULL,

units\_consumed INT NOT NULL,

FOREIGN KEY (meter\_id) REFERENCES Meter(meter\_id)

);

INSERT INTO Meter\_Readings (meter\_id, reading\_date, units\_consumed) VALUES

(1, '2025-02-01', 100), (2, '2025-02-01', 120), (3, '2025-02-01', 90),

(4, '2025-02-01', 110), (5, '2025-02-01', 130), (6, '2025-02-01', 95),

(7, '2025-02-01', 115), (8, '2025-02-01', 125), (9, '2025-02-01', 105),

(10, '2025-02-01', 135), (11, '2025-02-01', 85), (12, '2025-02-01', 140),

(13, '2025-02-01', 150), (14, '2025-02-01', 80), (15, '2025-02-01', 145);

A black screen with white text

AI-generated content may be incorrect.

**11. Notification**

CREATE TABLE Notification (

notification\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

message TEXT NOT NULL,

notification\_date DATE NOT NULL,

status ENUM('Sent', 'Pending') DEFAULT 'Pending',

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

INSERT INTO Notification (cust\_id, message, notification\_date, status) VALUES

(1, 'Bill due soon', '2025-02-10', 'Pending'), (2, 'Payment received', '2025-02-11', 'Sent'),

(3, 'Meter reading scheduled', '2025-02-12', 'Pending'), (4, 'Complaint resolved', '2025-02-13', 'Sent'),

(5, 'Bill overdue', '2025-02-14', 'Pending'), (6, 'Payment confirmed', '2025-02-15', 'Sent'),

(7, 'Outage alert', '2025-02-16', 'Pending'), (8, 'Feedback received', '2025-02-17', 'Sent'),

(9, 'Service update', '2025-02-18', 'Pending'), (10, 'Bill issued', '2025-02-19', 'Sent'),

(11, 'Maintenance notice', '2025-02-20', 'Pending'), (12, 'Payment reminder', '2025-02-21', 'Sent'),

(13, 'New tariff rates', '2025-02-22', 'Pending'), (14, 'Support request', '2025-02-23', 'Sent'),

(15, 'Penalty applied', '2025-02-24', 'Pending');

A screenshot of a computer screen

AI-generated content may be incorrect.

**12. Payment\_History**CREATE TABLE Payment\_History (

history\_id INT PRIMARY KEY AUTO\_INCREMENT,

payment\_id INT,

cust\_id INT,

amount DECIMAL(10, 2) NOT NULL,

payment\_date DATE NOT NULL,

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id),

FOREIGN KEY (payment\_id) REFERENCES Payments(payment\_id)

);

INSERT INTO Payment\_History (payment\_id, cust\_id, amount, payment\_date) VALUES

(1, 1, 15.00, '2025-02-10'), (2, 2, 18.00, '2025-02-11'), (3, 3, 13.50, '2025-02-12'),

(4, 4, 16.50, '2025-02-13'), (5, 5, 19.50, '2025-02-14'), (6, 6, 14.25, '2025-02-15'),

(7, 7, 17.25, '2025-02-16'), (8, 8, 18.75, '2025-02-17'), (9, 9, 15.75, '2025-02-18'),

(10, 10, 20.25, '2025-02-19'), (11, 11, 12.75, '2025-02-20'), (12, 12, 21.00, '2025-02-21'),

(13, 13, 22.50, '2025-02-22'), (14, 14, 12.00, '2025-02-23'), (15, 15, 21.75, '2025-02-24');

A black and white chart with numbers

AI-generated content may be incorrect.

**13. Payments**

CREATE TABLE Payments (

payment\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

bill\_id INT,

amount DECIMAL(10, 2) NOT NULL,

payment\_date DATE NOT NULL,

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id),

FOREIGN KEY (bill\_id) REFERENCES Bills(bill\_id)

);

INSERT INTO Payments (cust\_id, bill\_id, amount, payment\_date) VALUES

(1, 1, 15.00, '2025-02-10'), (2, 2, 18.00, '2025-02-11'), (3, 3, 13.50, '2025-02-12'),

(4, 4, 16.50, '2025-02-13'), (5, 5, 19.50, '2025-02-14'), (6, 6, 14.25, '2025-02-15'),

(7, 7, 17.25, '2025-02-16'), (8, 8, 18.75, '2025-02-17'), (9, 9, 15.75, '2025-02-18'),

(10, 10, 20.25, '2025-02-19'), (11, 11, 12.75, '2025-02-20'), (12, 12, 21.00, '2025-02-21'),

(13, 13, 22.50, '2025-02-22'), (14, 14, 12.00, '2025-02-23'), (15, 15, 21.75, '2025-02-24');



**14. Penalty**

CREATE TABLE Penalty (

penalty\_id INT PRIMARY KEY AUTO\_INCREMENT,

bill\_id INT,

penalty\_amount DECIMAL(10, 2) NOT NULL,

penalty\_date DATE NOT NULL,

FOREIGN KEY (bill\_id) REFERENCES Bills(bill\_id)

);

INSERT INTO Penalty (bill\_id, penalty\_amount, penalty\_date) VALUES

(1, 1.50, '2025-02-16'), (2, 1.80, '2025-02-16'), (3, 1.35, '2025-02-16'),

(4, 1.65, '2025-02-16'), (5, 1.95, '2025-02-16'), (6, 1.43, '2025-02-16'),

(7, 1.73, '2025-02-16'), (8, 1.88, '2025-02-16'), (9, 1.58, '2025-02-16'),

(10, 2.03, '2025-02-16'), (11, 1.28, '2025-02-16'), (12, 2.10, '2025-02-16'),

(13, 2.25, '2025-02-16'), (14, 1.20, '2025-02-16'), (15, 2.18, '2025-02-16');

A black and white screen with numbers

AI-generated content may be incorrect.

**15. Tariff**

CREATE TABLE Tariff (

tariff\_id INT PRIMARY KEY AUTO\_INCREMENT,

category VARCHAR(50) NOT NULL,

rate DECIMAL(10, 2) NOT NULL

);

INSERT INTO Tariff (category, rate) VALUES

('Domestic Low', 0.10), ('Domestic Medium', 0.15), ('Domestic High', 0.20),

('Commercial Low', 0.25), ('Commercial Medium', 0.30), ('Commercial High', 0.35),

('Industrial Low', 0.40), ('Industrial Medium', 0.45), ('Industrial High', 0.50),

('Rural Low', 0.08), ('Rural Medium', 0.12), ('Rural High', 0.18),

('Special Low', 0.22), ('Special Medium', 0.28), ('Special High', 0.32);

A screen shot of a black screen

AI-generated content may be incorrect.

**16. Usage\_History**

CREATE TABLE Usage\_History (

usage\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

meter\_id INT,

usage\_date DATE NOT NULL,

units\_consumed INT NOT NULL,

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id),

FOREIGN KEY (meter\_id) REFERENCES Meter(meter\_id)

);

INSERT INTO Usage\_History (cust\_id, meter\_id, usage\_date, units\_consumed) VALUES

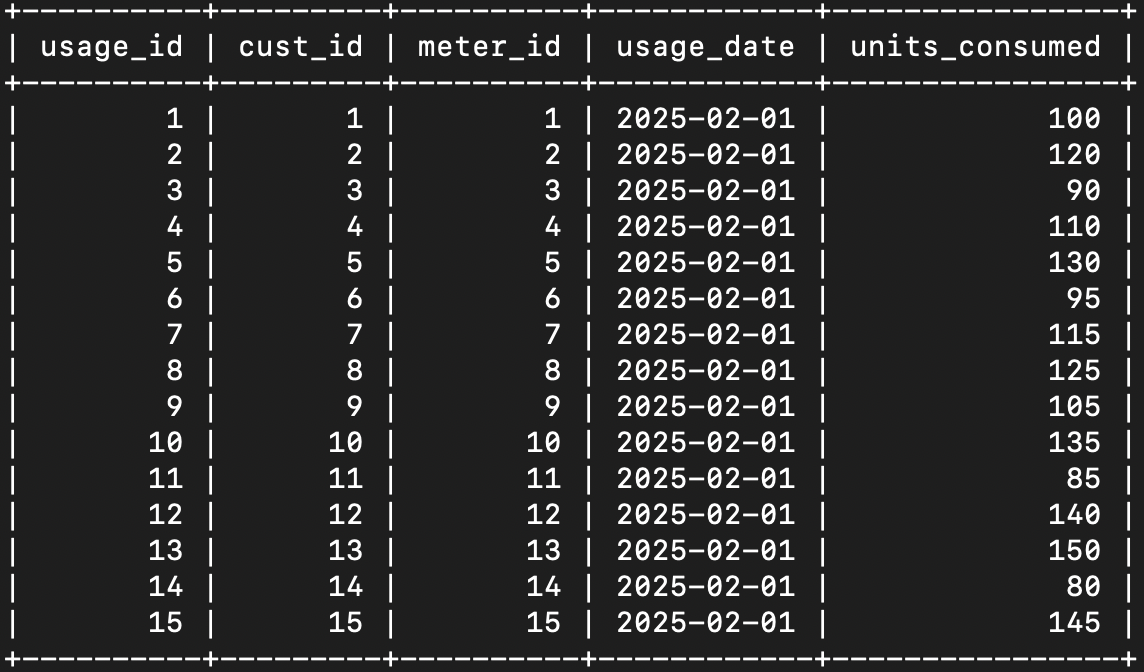
(1, 1, '2025-02-01', 100), (2, 2, '2025-02-01', 120), (3, 3, '2025-02-01', 90),

(4, 4, '2025-02-01', 110), (5, 5, '2025-02-01', 130), (6, 6, '2025-02-01', 95),

(7, 7, '2025-02-01', 115), (8, 8, '2025-02-01', 125), (9, 9, '2025-02-01', 105),

(10, 10, '2025-02-01', 135), (11, 11, '2025-02-01', 85), (12, 12, '2025-02-01', 140),

(13, 13, '2025-02-01', 150), (14, 14, '2025-02-01', 80), (15, 15, '2025-02-01', 145);



**CONSTRAINTS, JOINS, VIEWS, TRIGGERS, AND CURSORS**

**Constraints**

Constraints are rules in the database that make sure all the data stays correct and makes sense. They stop mistakes like duplicate records or missing important information. Here’s what we used:

**1. PRIMARY KEY:**

Each table has a unique ID (e.g., cust\_id, bill\_id) so no two records are the same.

Explanation: This is like giving every customer, bill, or meter its own special number. For example, if two customers had the same cust\_id, the system wouldn’t know who’s who. The primary key stops that by making sure every ID is unique. It’s the main way the database identifies each record.

Code Snippet: Here’s how we set the primary key for the Customer table:

CREATE TABLE Customer (

cust\_id INT AUTO\_INCREMENT,

cust\_name VARCHAR(100),

address VARCHAR(255),

email VARCHAR(100),

password VARCHAR(50),

PRIMARY KEY (cust\_id)

);

**2. AUTO\_INCREMENT:**

AUTO\_INCREMENT means the ID goes up by 1 for each new customer (1, 2, 3, etc.), so we don’t have to set it manually.

Code Snippet: Here’s how we use AUTO INCREMENT for the Customer table:

CREATE TABLE Customer (

cust\_id INT AUTO\_INCREMENT,

cust\_name VARCHAR(100),

address VARCHAR(255),

email VARCHAR(100),

password VARCHAR(50),

PRIMARY KEY (cust\_id)

);

**FOREIGN KEY:**

Links tables together (e.g., cust\_id in Bills links to Customer).

Explanation: This is how the system connects related information. For instance, a bill needs to know which customer it belongs to. The cust\_id in the Bills table acts like a pointer back to the Customer table. If a bill has cust\_id = 5, it’s for the customer with cust\_id = 5. This keeps everything tied together properly. Without it, bills could end up with no owner, which would be a mess.

Code Snippet: Here’s how we added a foreign key in the Bills table:

CREATE TABLE Bills (

bill\_id INT AUTO\_INCREMENT,

cust\_id INT,

account\_id INT,

amount DECIMAL(10,2),

bill\_date DATE,

due\_date DATE,

status VARCHAR(20),

PRIMARY KEY (bill\_id),

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

**3. REFERENCES** :

Customer(cust\_id) tells the database that cust\_id in Bills must match an existing cust\_id in Customer.  
CREATE TABLE Bills (

bill\_id INT AUTO\_INCREMENT,

cust\_id INT,

account\_id INT,

amount DECIMAL(10,2),

bill\_date DATE,

due\_date DATE,

status VARCHAR(20),

PRIMARY KEY (bill\_id),

FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)

);

**NOT NULL:**

Some fields (e.g., email, amount) must have a value.

Explanation: This rule makes sure important fields aren’t left blank. For example, every customer needs an email to log in—if it’s empty, they can’t use the system. Same with amount in Bills—a bill without an amount doesn’t make sense. Not Null forces these fields to always have something in them, so the system works right.

Code Snippet: Here’s how we used NOT NULL in the Customer table:

CREATE TABLE Customer (

cust\_id INT AUTO\_INCREMENT,

cust\_name VARCHAR(100) NOT NULL,

address VARCHAR(255) NOT NULL,

email VARCHAR(100) NOT NULL,

password VARCHAR(50) NOT NULL,

PRIMARY KEY (cust\_id)

);

NOT NULL after email means you can’t add a customer without an email address.

**JOINS**

Joins combine tables to pull related data together, like matching a customer’s name with their bills. They’re used in queries to show information from more than one table at once.

Example: Joining Customer and Bills using cust\_id to show a customer’s bills.

Explanation: Imagine a customer logs in and wants to see their bills. The Customer table has their name and email, but the Bills table has their bill details like amount and due date. A join links these tables using cust\_id so the system can show the customer’s name next to their bills. Without joins, we’d only see numbers like cust\_id = 5, but with a join, we see “John Doe’s bill of $50 due on April 10.” It’s like putting puzzle pieces together to get the full picture.

Code Snippet: Here’s a query to join Customer and Bills:

SELECT c.cust\_name, b.amount, b.due\_date

FROM Customer c

JOIN Bills b ON c.cust\_id = b.cust\_id

WHERE c.cust\_id = 5;

JOIN connects the tables, and ON c.cust\_id = b.cust\_id matches the customer to their bills. This shows all bills for customer ID 5 with their name.

**VIEWS**

Views are like shortcuts or saved queries that make it easier to see specific data without writing long queries every time.

Customer\_Bill\_Summary: Shows total bills, amounts, and pending amounts for a customer.

Explanation: This view is super helpful for customers who want a quick look at their billing status. Instead of searching through every bill, it adds up how many bills they have, the total amount they’ve been charged, and how much they still owe (pending bills). It’s like a report card for their account—simple and fast. Admins can use it too to check on customers. Without this view, we’d have to calculate all that manually each time, which would take longer.

Code Snippet: Here’s how we created the Customer\_Bill\_Summary view:

CREATE VIEW Customer\_Bill\_Summary AS

SELECT

c.cust\_id,

c.cust\_name,

COUNT(b.bill\_id) AS total\_bills,

SUM(b.amount) AS total\_amount,

SUM(CASE WHEN b.status = 'Pending' THEN b.amount ELSE 0 END) AS pending\_amount

FROM Customer c

LEFT JOIN Bills b ON c.cust\_id = b.cust\_id

GROUP BY c.cust\_id, c.cust\_name;

This counts all bills, adds up their amounts, and checks for pending ones, all grouped by customer. LEFT JOIN makes sure even customers with no bills show up.A screen shot of a black and white screen

AI-generated content may be incorrect.

**TRIGGERS**

Triggers are automatic actions that happen when something changes in the database, like adding a new record.

after\_payment\_insert: When a payment is added, it copies it to Payment\_History.

This trigger makes sure every payment is saved in the history table automatically. When a customer pays a bill, the Payments table gets a new record. The trigger sees that and instantly adds the same details—like the amount and date—to Payment\_History. This keeps a backup without anyone having to do extra work. It’s like a safety net to track all payments forever, even if something happens to the main payment record.

Code Snippet: Here’s how we created the trigger:

DELIMITER //

CREATE TRIGGER after\_payment\_insert

AFTER INSERT ON Payments

FOR EACH ROW

BEGIN

INSERT INTO Payment\_History (payment\_id, cust\_id, amount, payment\_date)

VALUES (NEW.payment\_id, NEW.cust\_id, NEW.amount, NEW.payment\_date);

END //

DELIMITER ;

AFTER INSERT means it runs after a payment is added. NEW.payment\_id grabs the new payment’s details and copies them to Payment\_History.

**CURSORS**

Cursors are tools to go through data one row at a time, like flipping through pages in a book. They’re useful when you need to do something complicated with each record, like updating rows one by one based on special rules. In our system, we didn’t need them because regular queries—like SELECT or INSERT—did everything we wanted. For example, generating bills for all customers at once was handled with a single query, not row-by-row. So, we kept it simple and skipped cursors since they weren’t necessary for our tasks.

Code Snippet: An example of what a cursor might look like :

DELIMITER //

CREATE PROCEDURE example\_cursor()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE curr\_cust\_id INT;

DECLARE cur CURSOR FOR SELECT cust\_id FROM Customer;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO curr\_cust\_id;

IF done THEN LEAVE read\_loop; END IF;

-- Do something with curr\_cust\_id

END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Conclusion**

The Electricity Bill System is a web-based tool that makes managing electricity bills easier and more efficient. It solves problems like slow manual billing and hard-to-access records by putting everything online. Customers can log in to check bills, pay them, and report issues, while admins can add meter readings, generate bills, and handle complaints—all from one place. Built with PHP, MySQL, and CSS, it saves time, reduces errors, and keeps data organized in a database.

This project meets its goals well. It automates billing, gives customers easy access, and helps admins work faster. The design is simple, so anyone can use it, and notifications improve communication. It shows how technology can make everyday tasks better—like turning paper bills into something quick and accurate.

There’s room to improve, though. Adding email reminders or a mobile app could make it even more helpful later on. For now, it’s a solid system that does its job: making billing smoother for customers and admins. It’s a practical solution that works today and can grow tomorrow.