

A Comprehensive Study on Database Design of Smart Home Energy Management Systems (SHEMS)

Rohan Chopra^{1,2}, Utsav Sharma^{1,3}

¹ New York University, New York, United States

² rc4920@nyu.edu

³ us2143@nyu.edu

Abstract. This study develops a model and methodology for describing the information objects in an energy management system called SHEMS and how such objects flow among the database components of such a system. This study builds a relational schema based on a literature review and intuition of designing a database system and then presents the Entity-Relation Diagram for the generated schema. The model and methodology support the specification of information objects, tested by running 6 SQL queries (given in question). An interactive prototype design tool based on the methodology and model will have to be designed and experimentally implemented (Second part of the project). PostgreSQL is used for the implementation of queries.

Keywords: Database design, SQL, Entity-relation diagram.

1. Introduction

The goal of this study is to provide a systematic and unifying relational database theory for the design of the SHEMS model. The first part covers the two basic components of the relational data model: its specification component, that is, the ER diagram, the database schema with dependencies, and its operational component, that is, the relational SQL queries. The choice of basic constructs, for specifying the semantically meaningful databases and for querying them, is justified through an in-depth investigation of their properties [1]. Some important themes are reviewed in this context: the analysis of the database schema and the implementation of the query language using intuition and universal relation assumptions. The subsequent parts of the study are structured around the two fundamental concepts illustrated in the first part, namely dependencies using an E-R diagram and queries[2]. The main themes of dependency theory are implication problems and applications to database schema design. Queries are solved with emphasis on the clear expressibility of query languages with the intuition learned in the class and assumptions taken for this study. We assume that the SHEMS database is deployed by an energy provider such as Con Edison, and its customers can then connect to the system. Customers need to be able to sign up for the system, by providing certain information such as their name and billing address. For this

purpose, the database stores relevant information about the customer for secure login of only the authenticated customer.

The practical need for efficient manipulation of large amounts of structured information and the basic insight that "data should be treated as an integrated resource, independent of application programs" led to the development of database management like in our case, SHEMS database management system. Database research has had a major impact on software systems and computer science in general; it has provided one of the few paradigms of man-machine interaction, which is both of a very high level, akin to programming in logic, and computationally efficient. Database concepts grew as the theory corresponded to and directly influenced several database management system (DBMS) implementations.

In this study, the SHEMS database integrates with various smart electrical devices, such as ACs, washers, dryers, refrigerators, ovens, fans, and lights, to monitor and control energy usage. We focus on the part of the system that stores past energy usage, thus allowing a user to understand their past energy usage and costs, and how they relate to their various appliances and appliance settings. However, the scope of this study does not try to model how the user can control future energy use and cost by, for example, setting schedules for their appliances, or avoiding running appliances at times when energy prices peak.

We test the created SHEMS database design by running 6 queries. The theory of queries is very much related to research on database logic programs, which are an elegant formalism for the study of the principles of knowledge base systems. The optimization of such programs involves both techniques developed for the relational data model and new methods for analyzing recursive definitions. The exposition closes with a discussion of how relational database theory deals with the problems of complex objects, incomplete information, and database updates.

The remainder of this paper includes a Literature review section which describes various database concepts used to accomplish this study, followed by the methodology used, experimental results (Implementation of queries), and finally conclusion and future scope of this paper.

2. Literature Review

In this section, we give an introduction to the concepts of database management systems we have used in this study and give examples for the same.

2.1. Structured Query Language (SQL)

Structured Query Language is a standard Database language that is used to create, maintain, and retrieve the relational database. It provides a set of commands that allow users to interact with databases, perform various operations, and retrieve or modify data[3].

Key aspects of SQL include:

1. **Database Definition and Creation:** Involves defining and creating relational databases, specifying the structure of tables, the types of data they will store, and the relationships between them.
2. **Data Retrieval (Querying):** Users can retrieve specific data from a database using SQL queries. SELECT statements are a fundamental part of SQL and allow for the retrieval of data based on specified criteria.
3. **Data Manipulation:** SQL provides commands for inserting, updating, and deleting records in a database. This allows users to modify the data stored in the database tables.
4. **Data Integrity:** SQL includes features that ensure the integrity of the data in a database. This includes constraints, such as primary keys and foreign keys, which help maintain consistency and relationships between tables.
5. **Transaction Control:** SQL supports transactions, which are sequences of one or more SQL statements that are executed as a single unit. This ensures data consistency and helps in managing database operations effectively.
6. **Data Security:** SQL includes features for managing access to databases, allowing administrators to grant or revoke permissions to users and control who can perform specific actions on the data.
7. **Aggregation and Grouping:** SQL supports functions for aggregating and summarizing data, such as COUNT, SUM, AVG, MIN, and MAX. This enables users to perform calculations on sets of data.

SQL is not tied to a specific database management system (DBMS), but different database systems that might implement SQL with slight variations. Although there is an ISO standard for SQL, most of the implementations slightly vary in syntax. Common relational database management systems that use SQL include MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, and SQLite. In this study, we have used PostgreSQL for the implementation of queries.

2.2. Relational database/ Tables

A **relational database** means the data is stored as well as retrieved in the form of relations (tables). In a relational database, a **table** is a structured collection of data organized into rows and columns. Tables are fundamental components of a relational database management system (RDBMS) and play a central role in storing and organizing data. Each table in a relational database represents a specific entity or concept, and the rows in the table represent individual instances or records of that entity.

Key characteristics and components of tables in a relational database:

1. **Columns (Attributes):** Tables are composed of columns, also known as attributes or fields. Each column represents a specific piece of information related to the entity the table represents.
2. **Rows (Records):** Each row in a table represents a single record or instance of the entity. It contains data values for each of the table's columns.

3. **Primary Key:** Tables typically have one or more columns designated as the primary key. The primary key uniquely identifies each row in the table. It ensures that there are no duplicate records and provides a way to establish relationships between tables.
4. **Relationships:** In a relational database, tables can be related to each other based on common values in their columns. This relationship is established using foreign keys, which are columns that reference the primary key of another table. This enables the creation of complex data structures that reflect the real-world relationships between different entities.
5. **Data Types:** Each column in a table has a specific data type, defining the kind of data it can store. Common data types include integers, strings, dates, and floating-point numbers.
6. **Constraints:** Tables can have constraints that enforce rules on the data they contain. Common constraints include uniqueness constraints, which ensure that values in a column are unique, and referential integrity constraints, which maintain the integrity of relationships between tables.
7. **Normalization:** Relational database design often involves normalization, a process that organizes data to reduce redundancy and improve data integrity. This includes breaking down large tables into smaller, related tables to avoid data duplication.
8. **Indexing:** Indexes can be created on one or more columns of a table to improve query performance. Indexes provide a quick lookup mechanism, making data retrieval faster for specific columns.

2.3. Constraints

Constraints in a relational database are rules or conditions applied to tables and columns to maintain the integrity, accuracy, and consistency of the data stored in the database. Constraints help ensure that the data conforms to a certain set of rules, preventing the introduction of inconsistencies or errors.

The following are common types of constraints in a relational database:

1. **Primary Key Constraint:** Ensures that a column or a set of columns uniquely identifies each record in a table.
2. **Foreign Key Constraint:** Defines a relationship between two tables by linking the foreign key column(s) in one table to the primary key column(s) in another table.
3. **Unique Constraint:** Ensures that all values in a column or a set of columns are unique.
4. **Check Constraint:** Specifies a condition that must be true for each row in a table.
5. **Not Null Constraint:** Ensures that a column does not contain NULL values.

These constraints collectively contribute to the data integrity and reliability of a relational database. By enforcing these rules, the database management system (DBMS) helps maintain a consistent and accurate representation of the real-world entities and relationships within the database.

2.4. Aggregation Functions

Aggregation functions are used to perform mathematical operations on data values of a relation. Some of the common aggregation functions used in SQL are:

1. **COUNT:** Count function is used to count the number of rows in a relation.
2. **SUM:** SUM function is used to add the values of an attribute in a relation.
3. In the same way, **MIN, MAX and AVG** are also aggregate functions commonly used in SQL.

GROUP BY: Group by is used to group the tuples of a relation based on an attribute or group of attributes. It is always combined with aggregation function which is computed on group.

2.5. Relational Schema

A relational schema is a blueprint or structural representation that defines the organization and structure of a relational database. It outlines the tables, their attributes (columns), and the relationships between them. In other words, a relational schema provides a visual and logical representation of how data is organized and stored in a relational database.

2.6. Entity-relationship (ER) Diagrams

An ER model is usually the result of systematic analysis to define and describe what data is created and needed by processes in an area of a business. Typically, it represents records of entities and events monitored and directed by business processes, rather than the processes themselves. It is usually drawn in a graphical form as boxes (entities) that are connected by lines (relationships) which express the associations and dependencies between entities. It can also be expressed in a verbal form, for example: one building may be divided into zero or more apartments, but one apartment can only be located in one building.

An ER model is typically implemented as a database. In a simple relational database implementation, each row of a table represents one instance of an entity type, and each field in a table represents an attribute type. In a relational database a relationship between entities is implemented by storing the primary key of one entity as a pointer or "foreign key" in the table of another entity.

There is a tradition for ER/data models to be built at two or three levels of abstraction. Note that the conceptual-logical-physical hierarchy below is used in other kinds of specification and is different from the three schema approach to software engineering.

An ER (Entity-Relationship) diagram is a type of conceptual modeling tool used in database design. Conceptual modeling involves creating an abstract representation of the data and its relationships within a system without concerning itself with the specific details of how the data will be implemented in a database management system (DBMS).

2.6.1. Importance of ER Diagram/ Conceptual Modeling

Conceptual modeling is the initial phase of database design, allowing designers to create a blueprint of the data structure before moving into the technical implementation. It serves as a communication tool between stakeholders, including developers, business analysts, and end-users, fostering a shared understanding of the data requirements.

2.6.2. Components of ER Diagrams

1. **Entities:** Entities represent real-world objects, such as "Customer" or "Product," that the database aims to store information about. Each entity has attributes that describe its properties.
2. **Relationships:** Relationships define associations between entities. They highlight how entities are connected and interact with each other. For example, a "Customer" entity might have a relationship with an "Order" entity. Relationships have various types, such as : One-to-Many, Many-to-One, Many-to-Many, One-to-One.
3. **Attributes:** Attributes are the properties of entities. They provide details about the characteristics of entities. For a "Customer" entity, attributes could include "CustomerID," "Name," and "Email."







Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Figure 1: Components of E-R Diagram

2.6.3. Procedures

PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements. A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc.

Advantages:

1. They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
2. They reduce the traffic between the database and the application since the lengthy statements are already fed into the database and need not be sent again and again via the application.
3. They add to code reusability, like how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:

1. Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.

3. Methodology

This section delineates the procedure followed to accomplish this study of making the SHEMS database. The problem introduction and assumptions taken are hence mentioned.

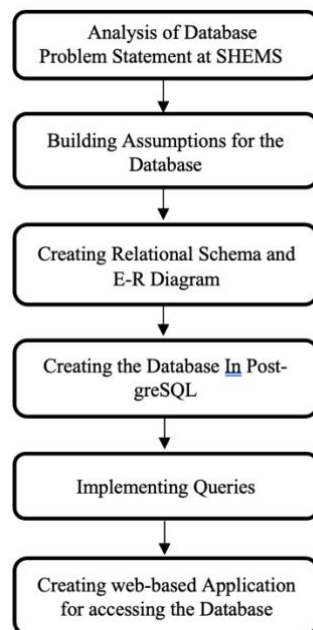


Figure 2: Flow chart of methodology

A client should be able to add service locations to their account, and each customer account may have several service locations—that is, properties that are getting electrical service. (A consumer may, for instance, have one service site at their permanent home, one at a vacation home, and

perhaps one at an owned rental property.) To enable energy usage comparisons across similar units, we need to keep track of every service location's full address, which includes the building's unit number, the date the location was taken over (for example, by moving in or purchasing it), the unit's square footage, number of bedrooms, and occupant count.

Customers may then enroll appliances like air conditioning, dryers, lighting, and refrigerators for each service location. It is assumed that all of these smart devices can transmit specific data, including their present configurations, certain events, or their energy consumption, to the SHEMS database, as detailed below. When registering new equipment, the user must indicate the model number and the kind. Customers pick their model from a list presented by the system once the database has likely restored a specific number of models and their attributes.

Be aware that a client may own many appliances of the same make and model at the same address. For example, a customer may own multiple models of smart lighting. To individually identify each enrolled device, you may thus need to give it an ID.

Subsequently, the SHEMS database will get specific data from each registered device. Events like turning on and off a device, adjusting a setting (such as the temperature on an air conditioner or the brightness on a dimmable smart light), or even opening and closing a refrigerator door are examples of this kind of data. An additional kind of data would be energy consumption data. It is anticipated that once turned on, every device will transmit data on its most recent energy usage, measured in kilowatt-hours, at regular intervals—let's say once every five minutes—as well as whenever it is turned off. For each such data item, we need to store the device ID, a time stamp, the label of the event, and a value.

Finally, energy prices per kWh can vary over time, and can also vary by location, and the system has to store this information. In particular, we assume that prices can change on an hourly basis and that prices can depend on the zip code of the service location. This allows us to compute the energy cost of, say, washing a load of clothes at 3pm on Thursday last week, based on the energy prices in the neighborhood at that time.

Assumptions taken in this SHEMS model:

1. All devices send out signals of various labels at certain intervals which are noted in the EnergyData table.
2. Devices send signals when they are switched on, switched off and also about their energy usage.
3. Each energy use signal label sent out by devices have an associated value to it.
4. The value is NULL for signal labels switched on, switched off and has energy consumption value by the device during that time interval in Kilowatts for the 'energy used' label.
5. Devices like Refrigerator send signals about door opening and closing.
6. The time interval after which the ACs send energy usage signal is 30 minutes.
7. The time interval after which the LED Lights and Fans send energy usage signal is 4 hours.
8. The time interval after which the Refrigerators send energy usage signal is 5 hours.
9. Special case: the time interval after which the Refrigerators send energy signals when door is opened is sent every 15 mins.
10. Each door of the refrigerator is closed within an interval of 1 hour.
11. Total Energy price is given in cents.
12. ConsumptionPrice contains only those zipcodes that are present in Address table.

4. Experimental Results

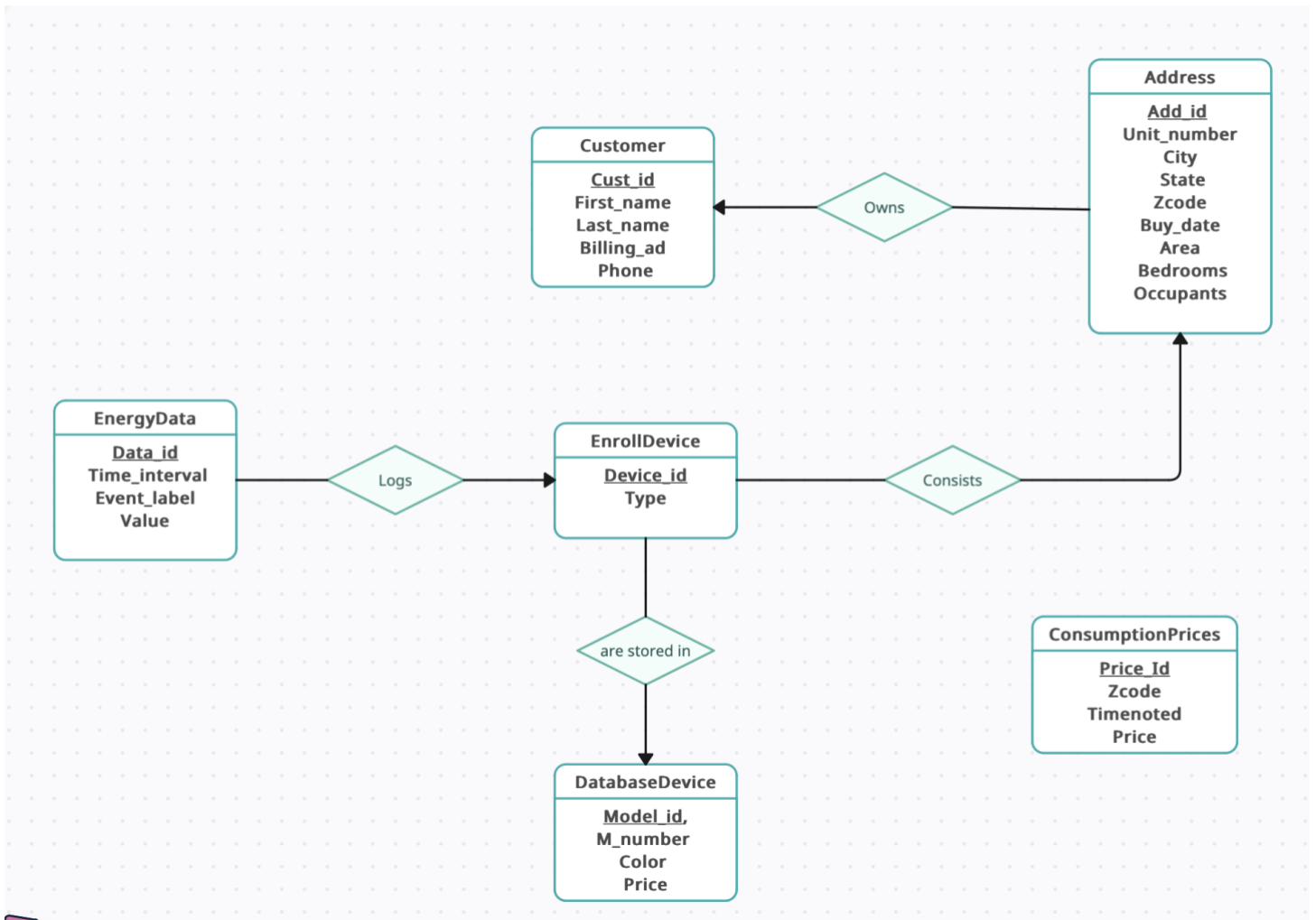


Figure 3: Entity-Relationship Diagram

The Database Schema is:

1. **Customer** (Cust_id, First_name, Last_name, Billing_add, Phone)
Primary Key – Cust_id
2. **ServiceLocation** (Add_id, Cust_id, Unit_number, City, State, Zcode, Buy_date, Area, Bedrooms, Occupants)
Primary Key – Add_id, Foreign Key – Cust_id
3. **DatabaseDevice** (Model_id, M_number, Color, Price)
Primary Key – Model_id

4. EnrollDevice (Device_id, Add_id, Model_id, Type)

Primary Key – Device_id, Foreign Key – Add_id, Model_id

5. EnergyData (Data_id, Device_id, Time_interval, Event_label, Value)

Primary Key – Data_id, Foreign Key – Device_id

6. ConsumptionPrices (Price_Id, Zcode, Timenoted, Price)

Primary Key – Price_Id

Note: Constraints are mentioned in front of attributes in below images.

This database schema created is supremely efficient because it has the following features:

1. Space efficiency
2. Suitable normalized
3. Dependency preserving
4. Lossless
5. No redundancy
6. It has improved Query Performance
7. Ease of Maintenance
8. Flexibility and Adaptability
9. Efficient Data Retrieval
10. Documentation and Understanding
11. Compatibility with Applications (will be proved in Second part of project)

Tables' Screenshots:

```
[project=# SELECT * FROM CUSTOMER;
 cust_id | first_name | last_name | billing_add | phone
-----+-----+-----+-----+-----
      1 | Alice     | Johnson  | 123 Oak St | 555-1234
      2 | Bob       | Williams | 456 Elm Ave | 555-5678
      3 | Charlie   | Smith    | 789 Cedar St | 555-9876
(3 rows)
```

[project=# SELECT * FROM CONSUMPTIONPRICES;				
price_id	zcode	timenoted		price
501	2108	2022-08-28	00:00:00	15.00
502	2108	2022-08-28	06:00:00	17.00
503	2108	2022-08-28	12:00:00	12.00
504	2108	2022-08-28	18:00:00	11.00
505	32789	2022-08-28	00:00:00	19.00
506	32789	2022-08-28	06:00:00	25.00
507	32789	2022-08-28	12:00:00	23.00
508	32789	2022-08-28	18:00:00	11.00
509	80014	2022-08-28	00:00:00	22.00
510	80014	2022-08-28	06:00:00	10.00
511	80014	2022-08-28	12:00:00	5.00
512	80014	2022-08-28	18:00:00	22.00
513	11221	2022-08-28	00:00:00	21.00
514	11221	2022-08-28	06:00:00	30.00
515	11221	2022-08-28	12:00:00	29.00
516	11221	2022-08-28	18:00:00	28.00
517	91911	2022-08-28	00:00:00	23.00
518	91911	2022-08-28	06:00:00	31.00
519	91911	2022-08-28	12:00:00	27.00
520	91911	2022-08-28	18:00:00	28.00
521	2108	2022-08-29	00:00:00	25.00
522	2108	2022-08-29	06:00:00	23.00
523	2108	2022-08-29	12:00:00	22.00
524	2108	2022-08-29	18:00:00	21.00
525	32789	2022-08-29	00:00:00	17.00
526	32789	2022-08-29	06:00:00	19.00
527	32789	2022-08-29	12:00:00	18.00
528	32789	2022-08-29	18:00:00	17.00
529	80014	2022-08-29	00:00:00	19.00
530	80014	2022-08-29	06:00:00	16.00
531	80014	2022-08-29	12:00:00	17.00
532	80014	2022-08-29	18:00:00	18.00
533	11221	2022-08-29	00:00:00	25.00
534	11221	2022-08-29	06:00:00	23.00
535	11221	2022-08-29	12:00:00	24.00
536	11221	2022-08-29	18:00:00	26.00
537	91911	2022-08-29	00:00:00	23.00
538	91911	2022-08-29	06:00:00	21.00
539	91911	2022-08-29	12:00:00	22.00
540	91911	2022-08-29	18:00:00	24.00
541	2108	2022-09-09	00:00:00	35.00
542	2108	2022-09-09	06:00:00	31.00
543	2108	2022-09-09	12:00:00	36.00
544	2108	2022-09-09	18:00:00	37.00
545	32789	2022-09-09	00:00:00	34.00
546	32789	2022-09-09	06:00:00	35.00
547	32789	2022-09-09	12:00:00	36.00
548	32789	2022-09-09	18:00:00	33.00
549	80014	2022-09-09	00:00:00	32.00
550	80014	2022-09-09	06:00:00	35.00
551	80014	2022-09-09	12:00:00	33.00
552	80014	2022-09-09	18:00:00	34.00
553	11221	2022-09-09	00:00:00	32.00
554	11221	2022-09-09	06:00:00	36.00
555	11221	2022-09-09	12:00:00	34.00
556	11221	2022-09-09	18:00:00	31.00
557	91911	2022-09-09	00:00:00	33.00
558	91911	2022-09-09	06:00:00	31.00
559	91911	2022-09-09	12:00:00	32.00
560	91911	2022-09-09	18:00:00	34.00
561	2108	2022-09-19	00:00:00	28.00

```
[project=# SELECT * FROM DATABASEDEVICE;
```

model_id	m_number	color	price
21	GE CAFE 400 REFRIGERATOR	WHITE	100.00
22	SAMSUNG QB800 ELECTRIC DRYER	BLACK	150.00
23	WHIRLPOOL XYZ WASHER	SILVER	120.00
24	LG SMART AC UNIT	RED	200.00
25	SONY LED TV	BLUE	180.00
26	APPLE SMART HOME HUB	GREEN	130.00
27	GOOGLE NEST OVEN	YELLOW	160.00
28	AMAZON ECHO DOT	PURPLE	190.00
29	DYSON V10 CORDLESS FAN	ORANGE	110.00
30	BOSE SOUNDLINK SPEAKER	BROWN	170.00
31	HP ENVY LAPTOP	GRAY	140.00
32	LENOVO YOGA TABLET	PINK	210.00
33	VOLTAS SMART AC UNIT	RED	200.00
34	LED LIGHTS	PINK	210.00

(14 rows)

```
[project=# SELECT * FROM ENROLLDEVICE;
```

device_id	add_id	model_id	type
1	1005	24	AC SYSTEM
2	1001	34	LED LIGHT
3	1001	21	REFRIGERATOR
4	1002	24	AC SYSTEM
5	1002	34	LED LIGHT
6	1002	29	FAN
7	1002	23	DISHWASHER
8	1003	27	OVEN
9	1003	21	REFRIGERATOR
10	1003	33	AC SYSTEM
11	1004	34	LED LIGHT
12	1004	21	REFRIGERATOR
13	1004	24	AC SYSTEM
14	1005	24	AC SYSTEM
15	1001	29	FAN
16	1004	23	DISHWASHER
17	1005	34	LED LIGHT
18	1005	23	DISHWASHER
19	1001	23	DISHWASHER
20	1005	27	OVEN

(20 rows)

```
[project=# SELECT * FROM ENERGYDATA;
```

data_id	device_id	time_interval	event_label	value
111	1	2022-08-28 11:00:00	SWITCH ON	
112	2	2022-08-28 11:30:00	SWITCH ON	
113	3	2022-08-28 05:00:00	SWITCH ON	
114	6	2022-08-28 11:35:00	SWITCH ON	
115	1	2022-08-28 11:40:00	TEMPERATURE LOWERED	40.00
116	1	2022-08-28 12:00:00	ENERGY USE	1.00
117	1	2022-08-28 13:00:00	ENERGY USE	0.77
118	1	2022-08-28 13:59:45	SWITCH OFF	2.50
119	2	2022-08-28 16:00:00	ENERGY USE	0.07
120	3	2022-08-28 10:10:00	DOOR OPENED	
121	3	2022-08-28 10:40:52	ENERGY USE	7.80
122	3	2022-08-28 10:50:20	DOOR CLOSED	
192	6	2022-08-28 14:35:00	ENERGY USE	0.26
204	6	2022-08-28 18:35:00	ENERGY USE	0.26
123	2	2022-08-28 19:00:00	SWITCH OFF	0.12
193	6	2022-08-28 19:05:00	TURN OFF	0.56
124	4	2022-08-29 10:00:00	SWITCH ON	
125	5	2022-08-29 10:30:00	SWITCH ON	
126	9	2022-08-29 10:35:00	SWITCH ON	
127	7	2022-08-29 10:35:00	SWITCH ON	
128	4	2022-08-29 10:40:00	TEMPERATURE INCREASED	45.00
129	4	2022-08-29 11:00:00	ENERGY USE	1.23
194	7	2022-08-29 11:35:00	ENERGY USE	1.80
195	7	2022-08-29 11:35:00	SWITCH OFF	1.80
130	4	2022-08-29 12:00:00	ENERGY USE	0.99
131	4	2022-08-29 12:59:45	SWITCH OFF	3.12
132	5	2022-08-29 15:00:00	ENERGY USE	0.08
133	9	2022-08-29 16:14:52	DOOR OPENED	
134	9	2022-08-29 16:29:52	ENERGY USE	9.45
135	9	2022-08-29 16:49:36	DOOR CLOSED	
136	5	2022-08-29 18:00:00	SWITCH OFF	0.20
137	4	2022-08-29 09:00:00	SWITCH ON	
138	11	2022-08-29 09:30:00	SWITCH ON	
139	12	2022-08-29 09:35:00	SWITCH ON	
140	8	2022-08-29 09:35:00	SWITCH ON	
196	8	2022-08-29 09:50:00	ENERGY USE	0.80
197	8	2022-08-29 09:55:00	SWITCH OFF	0.12
141	4	2022-08-29 09:40:00	TEMPERATURE INCREASED	45.00
142	4	2022-08-29 10:00:00	ENERGY USE	0.66
143	4	2022-08-29 11:00:00	ENERGY USE	0.87
144	4	2022-08-29 11:59:45	SWITCH OFF	2.32
145	11	2022-08-29 14:00:00	ENERGY USE	0.05
146	12	2022-08-29 14:35:00	DOOR OPENED	
147	12	2022-08-29 15:29:52	ENERGY USE	10.00
148	12	2022-08-29 14:43:00	DOOR CLOSED	
149	12	2022-08-29 17:00:00	SWITCH OFF	12.55
150	4	2022-09-09 10:00:00	SWITCH ON	
151	5	2022-09-09 10:30:00	SWITCH ON	
152	9	2022-09-09 10:35:00	SWITCH ON	
153	7	2022-09-09 10:35:00	SWITCH ON	
154	4	2022-09-09 10:40:00	TEMPERATURE INCREASED	45.00
155	4	2022-09-09 11:00:00	ENERGY USE	0.87
156	4	2022-09-09 12:00:00	ENERGY USE	0.65
198	7	2022-08-29 11:35:00	ENERGY USE	1.90
199	7	2022-08-29 11:35:00	SWITCH OFF	1.90
157	4	2022-09-09 12:59:45	SWITCH OFF	2.54
158	5	2022-09-09 15:00:00	ENERGY USE	0.04
159	9	2022-09-09 14:20:21	DOOR OPENED	
160	9	2022-09-09 16:29:52	ENERGY USE	9.98
161	9	2022-09-09 14:59:00	DOOR CLOSED	
162	5	2022-09-09 18:00:00	SWITCH OFF	0.13

162	5	2022-07-07	18:00:00	SWITCH OFF	0.15
163	13	2022-09-19	09:00:00	SWITCH ON	
164	11	2022-09-19	09:30:00	SWITCH ON	
165	12	2022-09-19	09:35:00	SWITCH ON	
166	19	2022-09-19	09:35:00	SWITCH ON	
167	13	2022-09-19	09:40:00	TEMPERATURE INCREASED	45.00
168	13	2022-09-19	10:00:00	ENERGY USE	1.10
200	19	2022-08-29	10:35:00	ENERGY USE	1.90
201	19	2022-08-29	10:35:00	SWITCH OFF	1.90
169	13	2022-09-19	11:00:00	ENERGY USE	0.88
170	13	2022-09-19	11:59:45	SWITCH OFF	2.72
171	11	2022-09-19	14:00:00	ENERGY USE	0.07
172	12	2022-09-19	20:40:00	DOOR OPENED	
173	12	2022-09-19	15:29:52	ENERGY USE	11.55
174	12	2022-09-19	21:25:25	DOOR CLOSED	
175	12	2022-09-19	21:30:00	SWITCH OFF	18.50
176	14	2022-09-29	08:00:00	SWITCH ON	
177	2	2022-09-29	08:30:00	SWITCH ON	
178	3	2022-09-29	08:35:00	SWITCH ON	
179	6	2022-09-29	08:35:00	SWITCH ON	
180	14	2022-09-29	08:40:00	TEMPERATURE INCREASED	45.00
181	14	2022-09-29	09:00:00	ENERGY USE	1.00
182	14	2022-09-29	10:00:00	ENERGY USE	0.91
183	14	2022-09-29	10:59:45	SWITCH OFF	2.40
184	2	2022-09-29	13:00:00	ENERGY USE	0.06
185	3	2022-09-29	12:00:00	DOOR OPENED	
202	6	2022-08-28	15:35:00	ENERGY USE	0.51
203	6	2022-08-28	16:05:00	TURN OFF	0.55
186	3	2022-09-29	14:29:52	ENERGY USE	10.20
187	3	2022-09-29	12:50:05	DOOR CLOSED	
188	3	2022-09-29	16:00:00	SWITCH OFF	12.00
189	11	2022-08-29	22:00:00	SWITCH OFF	0.12
190	9	2022-09-09	23:50:00	SWITCH OFF	13.00
191	11	2022-09-19	23:20:00	SWITCH OFF	0.21
205	2	2022-09-29	20:55:00	SWITCH OFF	0.15
(95 rows)					

Queries' Screenshots:

1. Query Number: 1

--QUERY 1:

```
SELECT C.CUST_ID, ED.DEVICE_ID, A.ADD_ID, DD.M_NUMBER AS DEVICE_MODEL, ED.TYPE AS DEVICE_TYPE,
SUM(EDATA.VALUE) AS TOTAL_ENERGY_CONSUMPTION
FROM ENROLLDEVICE ED
JOIN ENERGYDATA EDATA ON ED.DEVICE_ID = EDATA.DEVICE_ID
JOIN DATABASEDEVICE DD ON ED.MODEL_ID = DD.MODEL_ID
JOIN SERVICELOCATION A ON ED.ADD_ID = A.ADD_ID
JOIN CUSTOMER C ON A.CUST_ID = C.CUST_ID
WHERE C.CUST_ID = 1 AND EDATA.EVENT_LABEL='ENERGY USE'
AND EDATA.TIME_INTERVAL >= '2022-08-29 00:00:00'::TIMESTAMP - INTERVAL '24' HOUR
GROUP BY (C.CUST_ID,ED.DEVICE_ID, A.ADD_ID,DD.M_NUMBER, ED.TYPE);
```

```
project=# SELECT C.CUST_ID, ED.DEVICE_ID, A.ADD_ID, DD.M_NUMBER AS DEVICE_MODEL, ED.TYPE AS DEVICE_TYPE,
project=# SUM(EDATA.VALUE) AS TOTAL_ENERGY_CONSUMPTION
project=# FROM ENROLLDEVICE ED
project=# JOIN ENERGYDATA EDATA ON ED.DEVICE_ID = EDATA.DEVICE_ID
project=# JOIN DATABASEDEVICE DD ON ED.MODEL_ID = DD.MODEL_ID
project=# JOIN SERVICELOCATION A ON ED.ADD_ID = A.ADD_ID
project=# JOIN CUSTOMER C ON A.CUST_ID = C.CUST_ID
project=# WHERE C.CUST_ID = 1 AND EDATA.EVENT_LABEL='ENERGY USE'
project=# AND EDATA.TIME_INTERVAL >= '2022-08-29 00:00:00'::TIMESTAMP - INTERVAL '24' HOUR
project=# GROUP BY (C.CUST_ID,ED.DEVICE_ID, A.ADD_ID,DD.M_NUMBER, ED.TYPE);
 cust_id | device_id | add_id | device_model | device_type | total_energy_consumption
-----+-----+-----+-----+-----+-----
      1 |         2 |    1001 | LED LIGHTS   | LED LIGHT   |              0.13
      1 |         3 |    1001 | GE CAFE 400 REFRIGERATOR | REFRIGERATOR |             18.00
      1 |         4 |    1002 | LG SMART AC UNIT | AC SYSTEM   |              5.27
      1 |         5 |    1002 | LED LIGHTS   | LED LIGHT   |              0.12
      1 |         6 |    1002 | DYSON V10 CORDLESS FAN | FAN         |              1.03
      1 |         7 |    1002 | WHIRLPOOL XYZ WASHER | DISHWASHER  |              3.70
      1 |        19 |    1001 | WHIRLPOOL XYZ WASHER | DISHWASHER  |              1.90
(7 rows)

project=#
```

2. Query Number: 2

--QUERY 2:

```
SELECT ED.TYPE,AVG(EDATA.VALUE)
FROM ENROLLDEVICE AS ED
JOIN ENERGYDATA AS EDATA ON ED.DEVICE_ID=EDATA.DEVICE_ID
WHERE DATE_PART('MONTH',EDATA.TIME_INTERVAL)=8 AND
DATE_PART('YEAR',EDATA.TIME_INTERVAL)=2022
GROUP BY ED.TYPE;
```

```
project=# SELECT ED.TYPE,AVG(EDATA.VALUE)
project=# FROM ENROLLDEVICE AS ED
project=# JOIN ENERGYDATA AS EDATA ON ED.DEVICE_ID=EDATA.DEVICE_ID
project=# WHERE DATE_PART('MONTH',EDATA.TIME_INTERVAL)=8 AND
project=# DATE_PART('YEAR',EDATA.TIME_INTERVAL)=2022
project=# GROUP BY ED.TYPE;
```

type	avg
AC SYSTEM	11.9550000000000000
DISHWASHER	1.8666666666666667
FAN	0.428000000000000000
LED LIGHT	0.106666666666666667
OVEN	0.460000000000000000
REFRIGERATOR	9.9500000000000000

(6 rows)

project=# █

3. Query Number: 3

```
--QUERY 3:
--ASSUMPTION THAT EACH DOOR IS CLOSED WITHIN 1 HOUR
WITH DOO(DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL) AS
(SELECT DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL FROM ENERGYDATA WHERE EVENT_LABEL='DOOR OPENED'),
DC(DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL) AS
(SELECT DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL FROM ENERGYDATA WHERE EVENT_LABEL='DOOR CLOSED')
SELECT ADD_ID,M_NUMBER,DOO.DEVICE_ID,DOO.EVENT_LABEL,DOO.TIME_INTERVAL AS OPENED_TIME,DC.EVENT_LABEL,
DC.TIME_INTERVAL AS CLOSED_TIME
FROM DOO JOIN DC ON DOO.DEVICE_ID=DC.DEVICE_ID
JOIN ENROLLDEVICE ED ON ED.DEVICE_ID=DOO.DEVICE_ID
JOIN DATABASEDEVICE DD ON DD.MODEL_ID=ED.MODEL_ID
WHERE DC.TIME_INTERVAL BETWEEN DOO.TIME_INTERVAL
AND (DOO.TIME_INTERVAL + INTERVAL '1' HOUR) AND DC.TIME_INTERVAL-DOO.TIME_INTERVAL > INTERVAL '30' MINUTE;
```

```
project=# --QUERY 3:
project=# --ASSUMPTION THAT EACH DOOR IS CLOSED WITHIN 1 HOUR
project=# WITH DOO(DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL) AS
project=# (SELECT DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL FROM ENERGYDATA WHERE EVENT_LABEL='DOOR OPENED'),
project=# DC(DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL) AS
project=# (SELECT DATA_ID,DEVICE_ID,EVENT_LABEL,TIME_INTERVAL FROM ENERGYDATA WHERE EVENT_LABEL='DOOR CLOSED')
project=# SELECT ADD_ID,M_NUMBER,DOO.DEVICE_ID,DOO.EVENT_LABEL,DOO.TIME_INTERVAL AS OPENED_TIME,DC.EVENT_LABEL,
project=# DC.TIME_INTERVAL AS CLOSED_TIME
project=# FROM DOO JOIN DC ON DOO.DEVICE_ID=DC.DEVICE_ID
project=# JOIN ENROLLDEVICE ED ON ED.DEVICE_ID=DOO.DEVICE_ID
project=# JOIN DATABASEDEVICE DD ON DD.MODEL_ID=ED.MODEL_ID
project=# WHERE DC.TIME_INTERVAL BETWEEN DOO.TIME_INTERVAL
project=# AND (DOO.TIME_INTERVAL + INTERVAL '1' HOUR) AND DC.TIME_INTERVAL-DOO.TIME_INTERVAL > INTERVAL '30' MINUTE;
add_id |          m_number          | device_id | event_label |      opened_time      | event_label |      closed_time
-----+-----+-----+-----+-----+-----+-----
  1001 | GE CAFE 400 REFRIGERATOR |         3 | DOOR OPENED | 2022-08-28 10:10:00 | DOOR CLOSED | 2022-08-28 10:50:20
  1003 | GE CAFE 400 REFRIGERATOR |         9 | DOOR OPENED | 2022-08-29 16:14:52 | DOOR CLOSED | 2022-08-29 16:49:36
  1003 | GE CAFE 400 REFRIGERATOR |         9 | DOOR OPENED | 2022-09-09 14:20:21 | DOOR CLOSED | 2022-09-09 14:59:00
  1004 | GE CAFE 400 REFRIGERATOR |        12 | DOOR OPENED | 2022-09-19 20:40:00 | DOOR CLOSED | 2022-09-19 21:25:25
  1001 | GE CAFE 400 REFRIGERATOR |         3 | DOOR OPENED | 2022-09-29 12:00:00 | DOOR CLOSED | 2022-09-29 12:50:05
(5 rows)

project=#
```

4. Query Number: 4

```
--QUERY 4:

CREATE VIEW PRICEINTERVALS AS
SELECT
  ZCODE,
  PRICE,
  MIN(TIMENOTED) AS STARTTIME,
  LEAD(TIMENOTED) OVER (PARTITION BY ZCODE ORDER BY TIMENOTED) AS ENDTIME
FROM
  CONSUMPTIONPRICES
GROUP BY
  ZCODE, PRICE, TIMENOTED;

SELECT
  A.ADD_ID,
  A.CITY,
  A.STATE,
  A.ZCODE,
  SUM(E.VALUE * PI.PRICE) AS "TOTALENERGYCOST (IN ¢)"
FROM
  SERVICELOCATION A
JOIN
  ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
JOIN
  ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
JOIN
  PRICEINTERVALS PI ON A.ZCODE = PI.ZCODE AND E.TIME_INTERVAL >= PI.STARTTIME AND E.TIME_INTERVAL < PI.ENDTIME
WHERE
  E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
GROUP BY
  A.ADD_ID, A.CITY, A.STATE, A.ZCODE;
```

```
project=# CREATE VIEW PRICEINTERVALS AS
project=# SELECT
project=#   ZCODE,
project=#   PRICE,
project=#   MIN(TIMENOTED) AS STARTTIME,
project=#   LEAD(TIMENOTED) OVER (PARTITION BY ZCODE ORDER BY TIMENOTED) AS ENDTIME
project=# FROM
project=#   CONSUMPTIONPRICES
project=# GROUP BY
project=#   ZCODE, PRICE, TIMENOTED;
CREATE VIEW
project=# SELECT
project=#   A.ADD_ID,
project=#   A.CITY,
project=#   A.STATE,
project=#   A.ZCODE,
project=#   SUM(E.VALUE * PI.PRICE) AS "TOTALENERGYCOST (IN ¢)"
project=# FROM
project=#   SERVICELOCATION A
project=# JOIN
project=#   ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
project=# JOIN
project=#   ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
project=# JOIN
project=#   PRICEINTERVALS PI ON A.ZCODE = PI.ZCODE AND E.TIME_INTERVAL >= PI.STARTTIME AND E.TIME_INTERVAL < PI.ENDTIME
project=# WHERE
project=#   E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
project=# GROUP BY
project=#   A.ADD_ID, A.CITY, A.STATE, A.ZCODE;
add_id | city | state | zcode | TOTALENERGYCOST (IN ¢)
-----+-----+-----+-----+-----
1001 | BOSTON | MASSACHUSETTS | 2108 | 222.1600
1002 | ORLANDO | FLORIDA | 32789 | 2065.3200
1003 | DENVER | COLARADO | 80014 | 175.3700
1004 | NEW YORK | NEW YORK | 11221 | 545.5200
1005 | SAN DIEGO | CALIFORNIA | 91911 | 1355.2900
(5 rows)

project=#
```

5. Query Number: 5

```
--QUERY 5:

--FUNCTION THAT RETURNS THE AVG ENERGY CONSUMPTION OF OTHER SERVICE_LOCATIONS
--THAT HAVE 5% HIGHER OR LOWER SQUARE FOOTAGE
CREATE OR REPLACE FUNCTION TOTALSIMENERGY(ADD_ID NUMERIC, AREA NUMERIC)
    RETURNS NUMERIC(5,2)
    LANGUAGE PLPGSQL
    AS
    $$
    DECLARE AVG_VALUE NUMERIC(5,2);
    BEGIN
        WITH TEMP AS
        (SELECT * FROM SQUAREFOOTAGETOTALENERGY
        WHERE SQUAREFOOTAGETOTALENERGY.ADD_ID!=TOTALSIMENERGY.ADD_ID
        AND TOTALSIMENERGY.AREA BETWEEN SQUAREFOOTAGETOTALENERGY.AREA*0.95 AND SQUAREFOOTAGETOTALENERGY.AREA*1.05)
        SELECT AVG(TEMP.TOTALENERGYCONSUMPTION) INTO AVG_VALUE
        FROM TEMP;
        RETURN AVG_VALUE;
    END;
    $$;

--VIEW THAT CALCULATES THE TOTAL ENERGY CONSUMPTION OF
--ALL SERVICE LOCATIONS FOR THE MONTHS OF AUGUST
CREATE VIEW SQUAREFOOTAGETOTALENERGY AS (
    SELECT
        A.ADD_ID,
        A.AREA,
        SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
    FROM
        SERVICELOCATION A
    JOIN
        ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
    JOIN
        ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
    WHERE
        E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
    GROUP BY
        A.ADD_ID
);
```

```
--QUERY TO PRINT THE TOTAL ENERGY CONSUMED IN THE MONTH OF AUGUST
--AS PERCENTAGE OF AVERAGE ENERGY CONSUMED BY OTHER SERVICE LOCATIONS
--WITH 5% HIGHER LOWER SQUARE FOOTAGE
--COLUMN WITH PERCENTAGEOFAVG VALUE AS NULL DO NOT HAVE ANY
--OTHER SERVICE LOCATION WHO HAS SQUARE FOOTAGE 5% LOWER OR HIGHER
```

```
SELECT
```

```
    A.ADD_ID,
    A.CITY,
    A.STATE,
    A.ZCODE,
    SUM(E.VALUE) AS TOTALENERGYCONSUMED,
    (SUM(E.VALUE)*100)/TOTALSIMENERGY(A.ADD_ID,A.AREA) AS PERCENTAGEOFAVG
```

```
FROM
```

```
    SERVICELOCATION A
```

```
JOIN
```

```
    ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
```

```
JOIN
```

```
    ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
```

```
JOIN
```

```
    SQUAREFOOTAGETOTALENERGY SFTE ON A.ADD_ID = SFTE.ADD_ID
```

```
WHERE
```

```
    E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
```

```
GROUP BY
```

```
    A.ADD_ID;
```

```

project=# CREATE OR REPLACE FUNCTION TOTALSIMENERGY(ADD_ID NUMERIC,AREA NUMERIC)
project=# RETURNS NUMERIC(5,2)
project=# LANGUAGE PLPGSQL
project=# AS
project=# $$
project=# DECLARE AVG_VALUE NUMERIC(5,2);
project=# BEGIN
project=#     WITH TEMP AS
project=#     (SELECT * FROM SQUAREFOOTAGETOTALENERGY
project=#     WHERE SQUAREFOOTAGETOTALENERGY.ADD_ID!=TOTALSIMENERGY.ADD_ID
project=#     AND TOTALSIMENERGY.AREA BETWEEN SQUAREFOOTAGETOTALENERGY.AREA*0.95 AND SQUAREFOOTAGETOTALENERGY.AREA*1.05)
project=#     SELECT AVG(TEMP.TOTALENERGYCONSUMPTION) INTO AVG_VALUE
project=#     FROM TEMP;
project=#     RETURN AVG_VALUE;
project=# END;
project=# $$;
project=# CREATE FUNCTION
project=# --VIEW THAT CALCULATES THE TOTAL ENERGY CONSUMPTION OF
project=# --ALL SERVICE LOCATIONS FOR THE MONTHS OF AUGUST
project=# CREATE VIEW SQUAREFOOTAGETOTALENERGY AS (
project=#     SELECT
project=#     A.ADD_ID,
project=#     A.AREA,
project=#     SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
project=#     FROM
project=#     SERVICELOCATION A
project=#     JOIN
project=#     ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
project=#     JOIN
project=#     ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
project=#     WHERE
project=#     E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
project=#     GROUP BY
project=#     A.ADD_ID
project=# );
project=# CREATE VIEW
project=#

```

```

project=# SELECT
project=#     A.ADD_ID,
project=#     A.CITY,
project=#     A.STATE,
project=#     A.ZCODE,
project=#     SUM(E.VALUE) AS TOTALENERGYCONSUMED,
project=#     (SUM(E.VALUE)*100)/TOTALSIMENERGY(A.ADD_ID,A.AREA) AS PERCENTAGEOFAVG
project=# FROM
project=#     SERVICELOCATION A
project=# JOIN
project=#     ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
project=# JOIN
project=#     ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
project=# JOIN
project=#     SQUAREFOOTAGETOTALENERGY SFTE ON A.ADD_ID = SFTE.ADD_ID
project=# WHERE
project=#     E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01'
project=# GROUP BY
project=#     A.ADD_ID;

```

add_id	city	state	zcode	totalenergyconsumed	percentageofavg
1001	BOSTON	MASSACHUSETTS	2108	11.79	10.8155215117879094
1002	ORLANDO	FLORIDA	32789	109.01	924.5971162001696353
1003	DENVER	COLARADO	80014	10.37	
1004	NEW YORK	NEW YORK	11221	22.72	
1005	SAN DIEGO	CALIFORNIA	91911	44.27	

(5 rows)

6. Query Number: 6

```
--QUERY 6:
WITH MONTHLYENERGYAUGUST AS (
    SELECT
        A.ADD_ID,
        A.CITY,
        A.STATE,
        A.ZCODE,
        SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
    FROM
        SERVICELOCATION A
    JOIN
        ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
    JOIN
        ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
    WHERE
        (E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01')
    GROUP BY
        A.ADD_ID, A.CITY, A.STATE, A.ZCODE
),
MONTHLYENERGYSEPTEMBER AS (
    SELECT
        A.ADD_ID,
        A.CITY,
        A.STATE,
        A.ZCODE,
        SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
    FROM
        SERVICELOCATION A
    JOIN
        ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
    JOIN
        ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
    WHERE
        (E.TIME_INTERVAL >= '2022-09-01' AND E.TIME_INTERVAL < '2022-10-01')
    GROUP BY
        A.ADD_ID, A.CITY, A.STATE, A.ZCODE
)
```



```

SELECT
  M1.ADD_ID,
  M1.CITY,
  M1.STATE,
  M1.ZCODE,
  (M2.TOTALENERGYCONSUMPTION - M1.TOTALENERGYCONSUMPTION) / M1.TOTALENERGYCONSUMPTION * 100 AS PERCENTAGEINCREASE
FROM
  MONTHLYENERGYAUGUST M1
JOIN
  MONTHLYENERGYSEPTEMBER M2 ON M1.ADD_ID=M2.ADD_ID
ORDER BY
  PERCENTAGEINCREASE DESC
LIMIT 1;

```

```

project=# WITH MONTHLYENERGYAUGUST AS (
project=#     SELECT
project=#         A.ADD_ID,
project=#         A.CITY,
project=#         A.STATE,
project=#         A.ZCODE,
project=#         SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
project=#     FROM
project=#         SERVICELOCATION A
project=#     JOIN
project=#         ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
project=#     JOIN
project=#         ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
project=#     WHERE
project=#         (E.TIME_INTERVAL >= '2022-08-01' AND E.TIME_INTERVAL < '2022-09-01')
project=#     GROUP BY
project=#         A.ADD_ID, A.CITY, A.STATE, A.ZCODE
project=# ),
project=# MONTHLYENERGYSEPTEMBER AS (
project=#     SELECT
project=#         A.ADD_ID,
project=#         A.CITY,
project=#         A.STATE,
project=#         A.ZCODE,
project=#         SUM(E.VALUE) AS TOTALENERGYCONSUMPTION
project=#     FROM
project=#         SERVICELOCATION A
project=#     JOIN
project=#         ENROLLDEVICE ED ON A.ADD_ID = ED.ADD_ID
project=#     JOIN
project=#         ENERGYDATA E ON ED.DEVICE_ID = E.DEVICE_ID
project=#     WHERE
project=#         (E.TIME_INTERVAL >= '2022-09-01' AND E.TIME_INTERVAL < '2022-10-01')
project=#     GROUP BY
project=#         A.ADD_ID, A.CITY, A.STATE, A.ZCODE
project=# )
project=# SELECT
project=#     M1.ADD_ID,
project=#     M1.CITY,
project=#     M1.STATE,
project=#     M1.ZCODE,
project=#     (M2.TOTALENERGYCONSUMPTION - M1.TOTALENERGYCONSUMPTION) / M1.TOTALENERGYCONSUMPTION * 100 AS PERCENTAGEINCREASE
project=# FROM
project=#     MONTHLYENERGYAUGUST M1
project=# JOIN
project=#     MONTHLYENERGYSEPTEMBER M2 ON M1.ADD_ID=M2.ADD_ID
project=# ORDER BY
project=#     PERCENTAGEINCREASE DESC
project=# LIMIT 1;
  add_id | city   | state | zcode | percentageincrease
-----+-----+-----+-----+-----
    1004 | NEW YORK | NEW YORK | 11221 | 252.2447183098591500
(1 row)

project=#

```

5. Conclusion and Future Scope

All of the queries given in the question are successfully implemented which proves that the relational schema developed for the SHEMS database is correct. All the queries give meaningful results with validated results, thus proving this study to be a success. This study's future scope involves developing a comprehensive web application seamlessly integrated with the database.

References

1. Ceri, S., Pernici, B. and Wiederhold, G., 1987. Distributed database design methodologies. *Proceedings of the IEEE*, 75(5), pp.533-546.
2. Storey, V.C. and Goldstein, R.C., 1988. A methodology for creating user views in database design. *ACM Transactions on Database Systems (TODS)*, 13(3), pp.305-338.
3. Kanellakis, P.C., 1990. Elements of relational database theory. In *Formal models and semantics* (pp. 1073-1156). Elsevier.