SMART HOME AUTOMATION SYSTEM

DBMS PROJECT WORK SUBMITTED TO PSGR KRISHNAMMAL COLLEGE FOR WOMEN IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF COMPUTER APPLICATIONS OF BHARATHIAR UNIVERSITY, COIMBATORE – 641 046.

Submitted by
MAHENDRAN KAVISHKAVENI (22BCA059)
JASVADHANI D (22BCA045)

Guided By,
Dr. R. SURIYAGRACE M.Sc., M.Phil., Ph. D.,
Assistant Professor, Department of BCA,
PSGR Krishnammal College for Women,
Coimbatore – 641 004.

DEPARTMENT OF BCA
PSGR KRISHNAMMAL COLLEGE FOR WOMEN
College of Excellence
An Autonomous Institution, Affiliated to Bharathiar University
Accredited with 'A++' Grade by NAAC, An ISO 9001:2015 Certified Institution
Peelamedu, Coimbatore-641004.
November 2023

CERTIFICATE

This is to certify that it DBMS project entitled **"SMART HOME AUTOMATION SYSTEM"** Submitted to PSGR Krishnammal College for Women, Coimbatore in partial fulfilment of the requirement for the award of the Degree of Bachelor of Computer Applications is a record of original work done by **MAHENDRAN KAVISHKAVENI (22BCA059), JASVADHANI D (22BCA059)** during her period of study in Department of BCA and CS(AI), PSGR Krishnammal College for Women, Coimbatore under my supervision and guidance and her DBMS **Project** has not formed the basis for the award of any Degree/ Diploma/ Associate/ Fellowship or similar title to any candidate of any university.

Forwarded by

Faculty Guide	Head of the Department
Dr. R. SURIYAGRACE	**Mrs. K. GEETHALAKSHMI**
MSc., M.Phil., Ph. D.	**MCA., M Phil., B.Ed., (Ph.D.)**

DECLARATION

I hereby declare that the DBMS project entitled **"SMART HOME AUTOMATION SYSTEM "** submitted to PSGR Krishnammal College for Women, Coimbatore for the award of the **Degree of Bachelor of Computer Application** is a record of original work done by **MAHENDRAN KAVISHKAVENI (22BCA059), JASVADHANI D (22BCA045)** under the guidance of **Dr.R.SURIYAGRACE, MSc., M.Phil., Ph.D., Assistant Professor, Department of BCA and CS(AI),** PSGR Krishnammal College for Women, Coimbatore and this project have not found the basis for the award of any Degree/Diploma or similar title to any candidate of any university.

Place: Coimbatore

Date:

MAHENDRAN KAVISHKAVENI
(22BCA059)
JASVADHANI D
(22BCA045)

Endorsed by

Place: Coimbatore
Date:

Dr. R. SURIYAGRACE M.Sc., M.Phil., Ph. D.
(Faculty guide)

TABLE OF CONTENTS

CHAPTER I

CASE STUDY

1.1 ABSTRACT

The Smart Home Automation System is a database management project aimed at creating an efficient and interconnected system to automate various tasks in a smart home environment. The project focuses on designing a robust database to store information about users, devices, rooms, schedules, and security settings. By implementing the system, users can control and monitor their smart home devices seamlessly, enhancing convenience, energy efficiency, and security.

1.2 PROBLEM DEFINITION

The Smart Home Automation System project aims to address the challenge of managing a complex network of smart devices in a residential setting. The problem involves designing and implementing a well-structured database to store user information, device details, room specifications, scheduled activities, and security settings. Additionally, the system should provide a user-friendly interface to allow users to control and monitor their smart devices efficiently. The main goal is to create a scalable and reliable solution that enhances the overall smart home experience for users by integrating various devices and ensuring smooth automation and secure operations.

1.3 OBJECTIVE

The primary objective of it smart home automation system project is to design and implement a robust and user-friendly database management system. The system will provide a platform for users to efficiently control and monitor smart devices within their homes. The key goals are as follows:

1.Create a centralized database to store information about users, devices, rooms, schedules, and security settings.

2. Allow users to remotely control smart devices, such as lights, thermostats, and appliances, via the database interface.

3. Implement scheduling capabilities for devices to automate routine tasks based on user-defined time and repetition settings.

4. Provide security features to monitor and manage security devices like cameras and motion sensors, and receive alerts for any suspicious activities.

5. Optimize energy usage by analyzing device power consumption and suggesting energy-efficient practices to users.

1.4 TABLE DESCRIPTION

User Table

The User table stores information about users, including their unique User ID, name, email, password, phone number, and address details like street, city, state, and postal code.

User ID	Name	Email	Password	Phone No	Address
101	A	a@gmail.com	123	555,678	R28 Karachi
102	B	b@gmail.cm	456	222	G25 Surrey
103	C	c@gmail.com	789	444,273	Y78 York

Device Table

The Device table contains records of various devices with unique Device IDs, along with their names, types, manufacturer information, and power consumption in watts.

Device ID	Device Name	Device Type	Manufacture	Power Consumption	User ID	Room ID
1	S. Lock	Security	A-Ring	8W	101	201
2	S. Bulb	Lighting	Philips	5W	102	202
3	S. CCTV	Surveillance	Nest	4W	103	203

Room Table

The Room table records details about different rooms, identified by unique Room IDs, along with their names, types, square footage, temperature, and humidity.

Room ID	Name	Type	Square Footage	Temperature
201	Living Room	Living Area	300 sqft	72^0F
202	Bed Room	Bedroom	150 sqft	68^0F
203	Kitchen	CookingArea	100 sqft	75^0F

Schedule Table

The Schedule table holds information about scheduled activities, each with a unique Schedule ID, corresponding User ID (foreign key), and Device ID (foreign key).

Schedule ID	User ID	Device ID	Start Time	End Time	Repeat
Schedule 1	101	201	08:00	12:00	Daily
Schedule 2	101	202	09:00	13:00	Weekly
Schedule 3	103	203	10:00	11:00	Daily

Security Table

The Security table tracks security settings for users and devices, with unique Security IDs, associated User ID (foreign key), Device ID (foreign key), arm status (armed or disarmed), alert status (triggered or normal).

Security ID	User ID	Device ID	Arm status	Alert status
01	101	201	Armed	Normal
02	102	202	Disarmed	Triggered
03	102	203	Disarmed	Normal

These tables will serve as the foundation for storing and managing data related to users, devices, rooms, schedules, and security settings in the smart home automation system.

CHAPTER II

NORMALIZATION

2.1 ENTITIES AND ATTRIBUTES

In the Smart Home Automation System database, there are five primary entities. The User entity includes attributes for user identification, name, email, password, phone number, and address. The Device entity encompasses details about connected devices, such as device name, type, manufacturer, power consumption, and their association with users and rooms. The Room entity tracks room-specific data, including room name, type, square footage, and current temperature. Schedules for device automation are stored in the Schedule entity, featuring schedule ID, start/end times, and repeat frequency, tied to both users and devices. Lastly, the Security entity manages home security settings with attributes for arm status, alert status, and associations with users and devices. These entities form the foundation for a comprehensive Smart Home Automation System, facilitating user management, device control, room monitoring, scheduling, and security configuration.

2.2 CARDINALITY AND RELATIONSHIP

In the Smart Home Automation System database, The defined cardinalities and relationships are: one-to-one between User and Device, allowing each user to have a single device; one-to-many between User and Schedule, enabling users to have multiple schedules; one-to-many between Device and Security, allowing devices to be linked to multiple security records; one-to-many between Room and Device, permitting rooms to host multiple devices; and one-to-many between User and Security, allowing users to have multiple security records. Additionally, there's a one-to-many relationship between Room and Schedule, allowing rooms to have multiple schedules. These relationships are pivotal for efficient data management, ensuring seamless control and interaction within the Smart Home Automation System.

2.3 NORMAL FORMS TYPES

The Smart Home Automation System (SHAS) is an advanced technological solution engineered to enhance residential living by automating various aspects of home management. SHAS harnesses cutting-edge technology to streamline daily tasks, elevate communication between residents and their homes, and boost overall efficiency in household operations. It aims to simplify and optimize domestic living through the seamless integration of smart devices, offering homeowners convenience, control, and energy efficiency.

2.4 NORMALIZED TABLE

User Table

User ID	Name	Email	Password	Phone No	Address
101	Nick	a@gmail.com	123	555	R28 Karachi
101	Nick	a@gmail.com	123	678	R28 Karachi
102	Tony	b@gmail.cm	456	222	G25 Surrey
103	Jack	c@gmail.com	789	444	Y78 York
103	Jack	c@gmail.com	789	273	Y78 York

Device Table

Device ID	Device Name	Power consumption	User ID	Room ID
1	S. Lock	8W	101	201
2	S. Bulb	5W	102	202
3	S. CCTV	4W	103	203

Device Type Table

Device Type	Manufacture
Security	A-Ring
Lighting	Philips
Surveillance	Nest

Room Table

Room ID	Name	Temperature
201	Living Room	72^0F
202	Bed Room	68^0F
203	Kitchen	75^0F

Room Type Table

Square Footge	Type
300 sqft	Living Area
150 sqft	Bedroom
100 sqft	Cooking Area

Schedule Table

Schedule ID	Start Time	End Time	Repeat
Schedule 1	08:00	12:00	Daily
Schedule 2	09:00	13:00	Weekly
Schedule 3	10:00	11:00	Daily

Schedule Extra Table

Room ID	Device ID
1	201
2	202
3	203

Security Table:

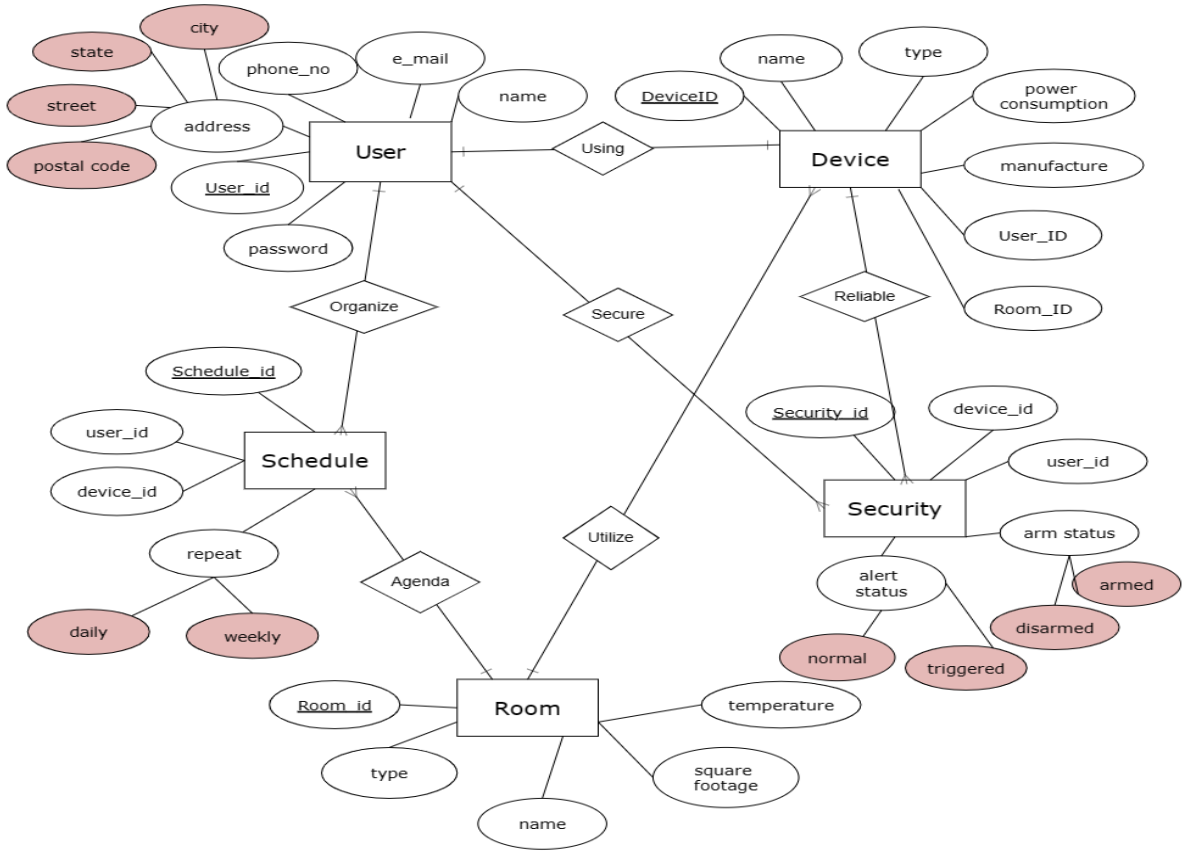
Security ID	Arm Status	Alert Status	User ID	Device ID
01	Armed	Normal		
02	Disarmed	Triggered		
03	Disarmed	Normal		

CHAPTER III

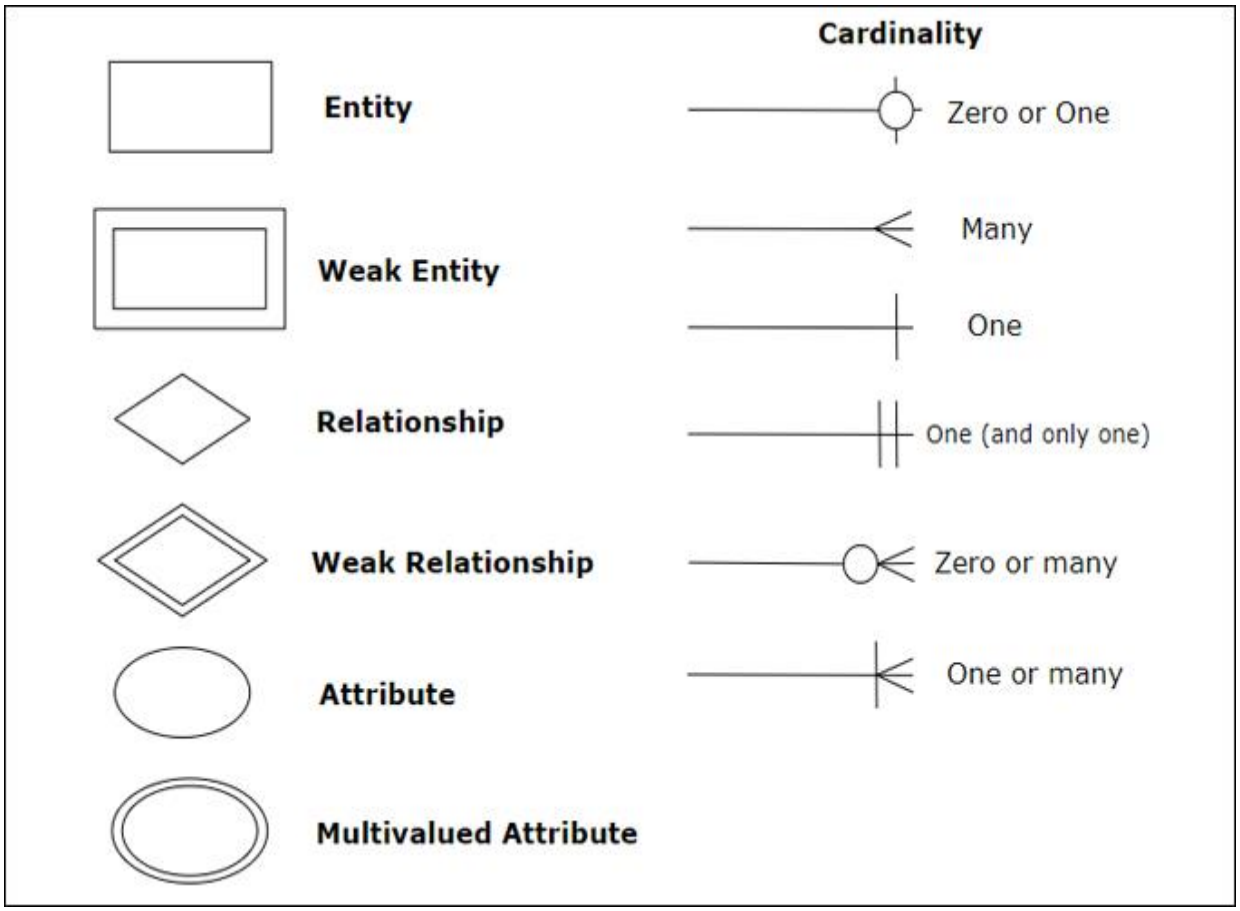
ENTITY RELATIONSHIP (ER DIAGRAM)

3.1 DESCRIPTION OF ER DIAGRAM

The Smart Home Automation System (SHAS) is a cutting-edge technology solution engineered to transform residential living by automating and optimizing various aspects of home management. SHAS utilizes state-of-the-art technology to simplify daily tasks, elevate communication between residents and their homes, and enhance overall operational efficiency in household management. It integrated system aims to deliver convenience, control, and energy efficiency to homeowners, streamlining their daily routines and creating a more comfortable living environment.



3.2 NOTATIONS OF DIAGRAM



DEFINITION

3.1 Primary Key:

The Smart Home Automation System (SHAS) is an advanced technological solution aimed at transforming traditional homes into intelligent living spaces. SHAS utilizes cutting-edge technology to automate and optimize various aspects of home life, enhancing convenience, energy efficiency, and security for homeowners.

User	User_id
Device	Device_id
Room	Room_id
Security	Security_id
Schedule	Schedule_id

3.2 Foreign Key:

A foreign key is a critical element in a database that establishes a link between two tables. In the context of a smart home automation system, it could be used to connect data in different tables, such as associating specific devices with rooms or users.

Device	User_id
Security	Device_id
Schedule	User_id

3.3 Composite Key

A composite key is a combination of multiple attributes that, together, uniquely identify a record in a database table. In a smart home system, it might be used to uniquely identify a device by considering a combination of factors like its serial number and location.

3.4 Composite Attributes:

A composite attribute is an attribute that can be further divided into sub-parts, each with its own meaning. In a smart home system, a composite attribute could represent a complex device attribute, like "sensor data," which might include sub-attributes for temperature, humidity, and light level.

3.5 Single Attributes:

A single attribute in a smart home system could represent a basic piece of information, like the "DeviceName" attribute that stores the name of a smart device.

3.6 Single-Valued Attributes:

A single-valued attribute stores a single value for each entity. In a smart home system, the "DeviceStatus" attribute could be single-valued, indicating whether a device is "On" or "Off."

3.7 Multi-Valued Attributes:

A multi-valued attribute can hold multiple values for a single entity. In a smart home system, the "DeviceFeatures" attribute might be multi-valued, listing various features or capabilities of a device.

3.8 Derived Attributes:

A derived attribute in a smart home system could be "MonthlyEnergyConsumption," which is calculated based on the historical energy usage data of a device.

3.10 Stored Attributes:

Most attributes in a smart home system are stored attributes, as they directly represent information about devices, users, or events within the system

3.11 Complex Attributes:

A complex attribute could represent something like the "DeviceLocation," which includes sub-attributes for latitude and longitude to pinpoint the exact location of a smart device.

3.12 Null Value Attributes:

In a smart home system, a "LastMaintenanceDate" attribute might sometimes be null if maintenance history is unknown or not applicable to a device.

3.13 Key Attributes:

Key attributes in a smart home system would be attributes like "DeviceID" or "UserID," which uniquely identify devices and users, respectively.

3.14 Value Set of Attributes:

The value set of an attribute like "DeviceType" might include values such as "Lighting," "Thermostat," or "Security" to categorize different types of smart devices.

CHAPTER IV

DATA DEFINITION LANGUAGE

4.1 CREATE

In the realm of Smart Home Automation Systems, the "CREATE" operation plays a pivotal role in setting up and expanding the ecosystem of connected devices. When "CREATE" in its context, It essentially adds new devices or components to smart home network. It can include adding smart bulbs, sensors, thermostats, or any other compatible devices. "CREATE" allows to establish a foundation for automation by bringing these devices into the system and configuring them to work together seamlessly

```
SQL> create table Room_details (room_id int, name char(30), type char(20), temperature int);

Table created.
```

4.2 RENAME

"RENAME" allows to personalize and organize Smart Home Automation System by changing the names of devices, rooms, or scenes. Its operation is particularly handy for improving the clarity of voice commands and making it easier to identify and control specific devices. For example, can "RENAME" a smart light in the living room to "Living Room Lamp" for more intuitive control.

```
SQL> rename Room_details to room1;

Table renamed.
```

4.3 ALTER

“ALTER” comes into play when want to make adjustments or modifications to the existing Smart Home Automation setup. Its operation allows to tweak settings, change device configurations, or update automation rules. For instance, if decide to change the temperature thresholds for a smart thermostat’s automated heating schedule, would use the “ALTER” command to modify the existing rule without starting from scratch.

```
SQL> alter table room1 add square_fooatge int;

Table altered.
```

4.4 TRUNCATE

In the context of Smart Home Automation, "TRUNCATE" is akin to starting fresh with a clean slate. When “TRUNCATE” automation system, remove all existing devices, rules, and configurations, effectively resetting r smart home network to its initial state. It can be useful if want to reorganize devices or if encounter issues that require a clean restart.

```
SQL> truncate table room1;

Table truncated.
```

4.5 DROP

The "DROP" operation is used when want to remove devices or components from the Smart Home Automation System. It could be because no longer use a particular device, or reconfiguring the automation setup. Dropping a device ensures that it is no longer part of the network, and any associated automation rules or routines are updated accordingly.

```
SQL> drop table room1;

Table dropped.
```

CHAPTER V

DATA MANIPULATION LANGUAGE

5.1 SELECT

In the world of Smart Home Automation Systems, the "SELECT" operation serves as a fundamental query tool. It allows users to retrieve information about the status, settings, or data from their connected devices. For instance, can "SELECT" the current temperature reading from a smart thermostat or inquire about the status of a smart locks to check if they are locked or unlocked.

```
SQL> select * from userr;

      U_ID NAME                 EMAIL                     PHONE ADDRESS
---------- -------------------- -------------------- ---------- --------------------
       101 nick                 a@gmail.com                 555 R288 Karachi
       102 tony                 b@gmail.com                 666 G25 Surrey
```

5.2 INSERT

"INSERT" is a critical operation for adding new data or records into the Smart Home Automation System. In context, it's about adding new devices, automation rules, or scenes to the network. For instance, when purchase a new smart device and want to integrate it into the smart home, would use the "INSERT" command to add it to the system, configure its settings, and make it part of an automation routines.

```
SQL> create table userr(u_id int, name char(20), email char(20), phone int, address char(20));

Table created.
```

```
SQL> insert into userr values('101','nick','a@gmail.com','555','R288 Karachi');
1 row created.
SQL> insert into userr values('102','tony','b@gmail.com','666','G25 Surrey');
1 row created.
```

5.3 UPDATE

"UPDATE" plays a pivotal role in fine-tuning a smart home setting. With it operation, it can modify the configuration of the devices or automation rules. For example, wish to change the brightness level of smart lights or adjust the timing of when sprinklers water garden, would use the "UPDATE" command to make these adjustments.

```
SQL> update userr set name='jasmine' where user_id=102;
1 row updated.
```

5.4 DELETE

The "DELETE" operation in Smart Home Automation allows users to remove specific data or settings. might use "DELETE" to clear historical data from sensors, remove outdated automation rules, or delete devices that are no longer part of the smart home network. It helps maintain a clutter-free and efficient automation setup

```
SQL> delete from userr where user_id=102;
1 row deleted.
```

CHAPTER VI

TRANSACTION CONTROL LANGUAGE

6.1 GRANT

In a Smart Home Automation System, the "GRANT" command serves as a crucial tool for controlling access and permissions. With "GRANT," system administrators can authorize specific users or devices to access and control certain smart home features. For example, can "GRANT" permission to a family member's smartphone to have full control over the lights and thermostat, allowing them to adjust settings and create automation routines.

```
SQL> create table security(se_id int, arm char(40), alert char(30));

Table created.
```

```
SQL> create user nick identified by pq2;

User created.
```

```
SQL> grant insert, select on security to nick;

Grant succeeded.
```

6.2 REVOKE

Conversely, the "REVOKE" command is used to remove or revoke previously granted access and permissions within the Smart Home Automation System. It can be essential for security and privacy purposes. If no longer want a guest's device to have control over the smart door locks or cameras, would "REVOKE" the permissions granted earlier, ensuring that their access is immediately terminated.

```
SQL> revoke insert, select on security from nick;

Revoke succeeded.
```

CHAPTER VII

DATA CONTROL LANGUAGE

```
SQL> create table schedule(s_id int, room_id int, start_time int, repeat char(20));

Table created.
```

```
SQL> insert into schedule values('1','201','8','daily');

1 row created.

SQL> insert into schedule values('2','202','10','weekly');

1 row created.

SQL> insert into schedule values('3','203','11','daily');

1 row created.
```

7.1 COMMIT

In the context of managing a Smart Home Automation System, the “COMMIT” command is crucial for ensuring data integrity and consistency. When can make changes or updates to the automation settings, such as creating new automation rules or altering device configurations, those changes are initially stored in a temporary state. By using “COMMIT”, confirm that these changes should be permanently saved and applied to r system. It ensures that the smart home operates based on the latest configurations and automation rules that is been set.

```
SQL> commit;

Commit complete.
```

7.2 ROLLBACK

"ROLLBACK" is a powerful DCL command that allows to undo changes made in Smart Home Automation System in case something goes wrong or if it need to revert to a previous state. For example, if it accidentally delete a critical automation rule or make unintended changes to r device settings, it can use "ROLLBACK" to restore r system to its previous state, preventing any disruptions to r smart home's functionality.

```
SQL> delete from schedule where s_id='2';

1 row deleted.
```

```
SQL> rollback;

Rollback complete.
```

```
SQL> select * from schedule;

      S_ID    ROOM_ID START_TIME REPEAT
---------- ---------- ---------- --------------------
         1        201          8 daily
         2        202         10 weekly
         3        203         11 daily
```

7.3 SAVEPOINT

"SAVEPOINT" is a DCL command that lets set a specific point within a transaction where it can later roll back to if necessary. It is valuable when making a series of changes to Smart Home Automation System. For instance, making multiple adjustments in lighting, thermostat, and security settings, can set a "SAVEPOINT" at various stages. If any encounter issues or decide to backtrack on certain changes, can "ROLLBACK" to the designated "SAVEPOINT" without affecting changes made after it.

```
SQL> savepoint insertion;

Savepoint created.
```

```
SQL> update schedule set start_time='8' where s_id='2';

1 row updated.
```

```
SQL> rollback to insertion;

Rollback complete.
```

```
SQL> select * from schedule;

      S_ID    ROOM_ID START_TIME REPEAT
---------- ---------- ---------- --------------------
         1        201          8 daily
         2        202         10 weekly
         3        203         11 daily
```

CHAPTER VIII

DATA INTEGRITY CONSTRAINS

8.1 PRIMARY KEY

In a database for a Smart Home Automation System, the "PRIMARY KEY" constraint is used to uniquely identify each record or row in a table. For example, in a table that stores information about connected devices, the device ID could be designated as the "PRIMARY KEY." It ensures that each device has a unique identifier, preventing duplicate entries and facilitating efficient data retrieval.

```
SQL> create table room1(r_id int PRIMARY KEY, name char(20), type char(20));

Table created.
```

```
SQL> insert into room1 values(201,'kitchen','cooking');

1 row created.
```

8.2 FOREIGN KEY

The "FOREIGN KEY" constraint establishes a link between two tables in a database, typically representing a relationship between them. In the context of a Smart Home Automation System, might have a table for users and another for devices. By using a "FOREIGN KEY" can associate each device with a specific user, ensuring that the database maintains referential integrity. For instance, the user ID in the devices table could be a "FOREIGN KEY" referencing the primary key of the user's table.

```
SQL> create table device(d_id int PRIMARY KEY, name char(30), type char(20), r_id int, FOREIGN KEY(r
_id) references room1(r_id));

Table created.
```

```
SQL> insert into device values(1,'lock','security',201);

1 row created.
```

8.3 NOT NULL

The "NOT NULL" constraint ensures that a specific field or column in a table cannot contain null (empty) values. In a Smart Home Automation System, its constraint is essential for critical data fields. For example, the "email" field in a user table should be marked as "NOT NULL" to guarantee that every user has a valid email address associated with their account.

```
SQL> create table security(s_id int, u_id int, alert char(20) NOT NULL);

Table created.
```

```
SQL> insert into security values(01,101,'normal');

1 row created.
```

8.4 UNIQUE

The "UNIQUE" constraint ensures that the values in a particular column or combination of columns are unique across all records in a table. In the context of a Smart Home Automation System, might use it constraint for fields like device serial numbers or usernames. It prevents duplicate values and ensures data integrity by enforcing uniqueness. For instance, each device's serial number could be marked as "UNIQUE" to prevent multiple devices with the same serial number.

```
SQL>  create table userr(u_id int, name char(30), address char(30) UNIQUE);

Table created.
```

```
SQL> insert into userr values(101, 'nick','karachi');

1 row created.
```

8.5 CHECK

The "CHECK" constraint allows to define specific conditions that data in a column must meet. In a Smart Home Automation System, could use the "CHECK" constraint to ensure that certain values are within a valid range. For example, if have a table for temperature readings, can add a "CHECK" constraint to verify that temperature values fall within a predefined range, ensuring the data's accuracy and integrity.

```
SQL> create table room2(r_id int, name char(20), square int, CHECK(SQUARE>=50));

Table created.
```

```
SQL> insert into room2 values(201,'kitchen',150);

1 row created.
```

CHAPTER IX

AGGREGATE FUNCTION & SORTING

```
SQL> create table room1(r_id int, name char(30), type char(30), temp int, square int);

Table created.
```

```
SQL>  insert into room1 values(201,'kitchen','cooking',75,150);
1 row created.
SQL> insert into room1 values(202,'dining','eating',65,200);
1 row created.
SQL> insert into room1 values(203,'bedroom','sleeping',70,250);
1 row created.
SQL>  insert into room1 values(204,'room','playing',50,300);
1 row created.
```

9.1 AVG()

"AVG" calculates the average (mean) value of a numeric column. It can be handy for analysing data like the average power usage of any devices over time or the average temperature in home throughout the day.

```
SQL> select avg(temp) from room1;

 AVG(TEMP)
----------
        65
```

9.2 MAX()

"MAX" retrieves the maximum value from a column, allowing to identify the highest recorded value among the data. In a smart home context, its function could help to find the maximum temperature reached by thermostat or the highest luminance level of smart lights.

```
SQL> select max(square)from room1;

MAX(SQUARE)
-----------
        300
```

9.3 COUNT()

"COUNT" provides a count of the number of rows in a dataset, which can be valuable for various purposes. For instance, It can use "COUNT" to determine how many devices are connected to smart home system or how many automation rules have set up.

```
SQL> select count(r_id)from room1;

COUNT(R_ID)
-----------
          4
```

9.4 MIN()

The "MIN" aggregate function, on the other hand, returns the minimum value from a column. Might use to find the lowest temperature recorded by sensors or the dimmest setting for smart lights.

```
SQL> select min(square)from room1;

MIN(SQUARE)
-----------
        150
```

9.5 SUM()

The "SUM" aggregate function calculates the total of a numerical column, making it useful for computing the total energy consumption of all devices in smart home. For instance, can use "SUM" to find the total kilowatt-hours used by all devices over a specific time period.

```
SQL> select sum(square)from room1;

SUM(SQUARE)
-----------
        900
```

SORTING

9.6 ASC

Arranging data in ascending order, which means it starts with the lowest value and goes up. For instance, could use ascending sorting to list devices by their power consumption from least to most efficient.

```
SQL> select name from room1 order by name asc;

NAME
------------------------------
bedroom
dining
kitchen
room
```

9.7 DESC

In contrast, descending sorting orders data from the highest value to the lowest. might use it to organize a device list by the devices with the highest power consumption at the top.

```
SQL> select temp from room1 order by temp desc;

      TEMP
----------
        75
        70
        65
        50
```

CHAPTER X

JOIN OPERATIONS

```
SQL>  create table room1(r_id int, name char(30), type char(40));

Table created.
```

```
SQL> create table device(d_id int, name char(30), r_id int);

Table created.
```

```
SQL> create table device(d_id int, name char(30), r_id int);
Table created.
SQL> insert into room1 values(201,'kitchen','cooking');
1 row created.
SQL> insert into room1 values(202,'bedroom','sleeping');
1 row created.
SQL> insert into device values(1,'lock',201);
1 row created.
SQL> insert into device values(2,'bulb',202);
1 row created.
```

```
SQL> set lines 350;
SQL> select * from room1;

      R_ID NAME                           TYPE
---------- ------------------------------ ----------------------------------------
       201 kitchen                        cooking
       202 bedroom                        sleeping

SQL> select * from device;

      D_ID NAME                                R_ID
---------- ------------------------------ ----------
         1 lock                                201
         2 bulb                                202
```

10.1 JOIN

A "JOIN" operation combines data from two or more tables based on a related column between them. In a Smart Home Automation System, might use a "JOIN" to connect data from tables like devices, users, and rooms. For example, could use a "JOIN" to associate devices with the rooms they are located in or to link users to the devices they control.

```
SQL> select * from room1 join device on room1.r_id=device.r_id;

    R_ID NAME                           TYPE                                     D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
     201 kitchen                        cooking                                     1 lock                                201
     202 bedroom                        sleeping                                    2 bulb                                202
```

10.2 RIGHT INNER JOIN

Conversely, a "RIGHT INNER JOIN" combines rows based on a matching condition and includes only the rows from the right (second) table where there is a match in the left (first) table. Type of join is less common but can be useful in specific scenarios.

```
SQL>  select * from room1 right join device on room1.r_id=device.r_id;

    R_ID NAME                           TYPE                                     D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
     201 kitchen                        cooking                                     1 lock                                201
     202 bedroom                        sleeping                                    2 bulb                                202
```

10.3 LEFT INNER JOIN

A "LEFT INNER JOIN" combines rows from two tables based on a matching condition and includes only the rows from the left (first) table where there is a match in the right (second)

table. For instance, want to see a list of devices and their associated rooms, a "LEFT INNER JOIN" would return only devices that are connected to rooms.

```
SQL>  select * from room1 left join device on room1.r_id=device.r_id;

     R_ID NAME                           TYPE                                      D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
      201 kitchen                        cooking                                    1 lock                                 201
      202 bedroom                        sleeping                                   2 bulb                                 202
```

10.4 FULL JOIN

A "FULL JOIN" combines rows from both tables, including all rows from both the left and right tables. When there is no match in one table, the result will still include that table's rows with null values for columns from the other table. In the context of a Smart Home Automation System, a "FULL JOIN" might be used to generate a comprehensive list of all devices and all rooms, with or without matches.

```
SQL> select * from room1 full join device on room1.r_id=device.r_id;

     R_ID NAME                           TYPE                                      D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
      201 kitchen                        cooking                                    1 lock                                 201
      202 bedroom                        sleeping                                   2 bulb                                 202
```

10.5 RIGHT OUTER JOIN

Similar to the "LEFT OUTER JOIN," a "RIGHT OUTER JOIN" combines rows from both tables but includes all rows from the right (second) table and only the matching rows from the left (first) table.

```
SQL>  select * from room1 right outer join  device on room1.r_id=device.r_id;

     R_ID NAME                           TYPE                                      D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
      201 kitchen                        cooking                                    1 lock                                 201
      202 bedroom                        sleeping                                   2 bulb                                 202
```

10.6 LEFT OUTER JOIN

A "LEFT OUTER JOIN" combines rows from both tables based on a matching condition and includes all rows from the left (first) table and only the matching rows from the right (second) table. It is type of join is useful for situations where want to see all records from one table along with any matching records from another.

```
SQL> select * from room1 left outer join device on room1.r_id=device.r_id;

     R_ID NAME                           TYPE                                      D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
      201 kitchen                        cooking                                    1 lock                                 201
      202 bedroom                        sleeping                                   2 bulb                                 202
```

10.7 FULL OUTER JOIN

A "FULL OUTER JOIN" combines rows from both tables, including all rows from both the left and right tables. When there is no match in one table, the result will still include that table's rows with null values for columns from the other table. In the context of a Smart Home Automation System, a "FULL OUTER JOIN" might be used to generate a comprehensive list of all devices and all rooms, with or without matches.

```
SQL> select * from room1 full outer join device on room1.r_id=device.r_id;

     R_ID NAME                           TYPE                                      D_ID NAME                                R_ID
--------- ------------------------------ ---------------------------------------- ---------- ------
      201 kitchen                        cooking                                    1 lock                                 201
      202 bedroom                        sleeping                                   2 bulb                                 202
```

CHAPTER XI

SUB QUARIES

11.1 CREATING A TABLE

```
SQL> create table room1(r_id int, name char(30), temp int, square int);

Table created.
```

11.2 INSERTING VALUES IN A TABLE

```
SQL> insert into room1 values(201,'k',45,85);

1 row created.

SQL> insert into room1 values(202,'oo',56,95);

1 row created.
```

11.3 SELECT, DISPLAYING VALUES IN A TABLE

Subqueries can be used to retrieve specific values or rows from one table and display them in the result of another query. For instance, might use a subquery to find and display the names of all users who have specific types of devices in their smart homes.

```
SQL> select * from room1;

      R_ID NAME                                 TEMP     SQUARE
---------- ------------------------------ ---------- ----------
       201 k                                      45         85
       202 oo                                     56         95
```

11.4 AGGREGATE FUNCTIONS WITH SELECT

Subqueries can also be used with aggregate functions in the SELECT statement. For instance, might use a subquery to calculate the average energy consumption of devices in a particular room and display that value alongside other room information.

```
SQL> select * from room1 where temp=(select min(temp)from room1);

      R_ID NAME                                 TEMP     SQUARE
---------- ------------------------------ ---------- ----------
       201 k                                      45         85
```

11.5 USING LIKE OPERATOR

The LIKE operator within a subquery can be helpful for searching for patterns or partial matches. For instance, it might use a subquery with LIKE to find all devices with names containing the word "light."

```
SQL>  select count(*) from (select  * from room1 where name like 'o%');

  COUNT(*)
----------
         1
```

11.6 UPDATING A ROW

Subqueries can be employed to update rows in one table based on conditions or values retrieved from another table. For example, it could use a subquery to update the status of all devices owned by a specific user when that user's status changes to "inactive."

```
SQL> update room1 set square=100 where temp in(select temp from room1 where temp>54);

1 row updated.

SQL> select * from room1;

      R_ID NAME                           TEMP     SQUARE
---------- ---------------------------- ---------- ----------
       201 k                                    45         85
       202 oo                                   56        100
```

11.7 DELETING A ROW

Subqueries can assist in identifying rows that need to be deleted based on specific criteria. For instance, could use a subquery to find and delete all records of devices that haven't been used for a certain period.

```
SQL> delete from room1 where square=(select min(square) from room1);

1 row deleted.

SQL> select * from room1;

      R_ID NAME                           TEMP     SQUARE
---------- ---------------------------- ---------- ----------
       202 oo                                   56        100
```

11.8 USING WHERE CLAUSE

Subqueries within a WHERE clause allow to filter results based on values returned from another query. For example, could use a subquery to find all devices with energy consumption greater than the average energy consumption across all devices.

```
SQL> select * from room1 where square in(select square from room1 where r_id=202);

      R_ID NAME                           TEMP     SQUARE
---------- ---------------------------- ---------- ----------
       202 oo                                   56        100
```

CHAPTER XII

PL/SQL

12.1 FUNCTIONS IN PL/SQL

Functions in PL/SQL are blocks of code that perform a specific task and return a single value. They are similar to functions in other programming languages. In a Smart Home Automation System, it might use PL/SQL functions to calculate the average energy consumption of devices, convert units, or retrieve specific data from the database. Functions are designed to be reusable and can take parameters to customize their behaviour.

```
SQL>  set serveroutput on;
SQL>  declare
  2   a number;
  3   b number;
  4   c number;
  5   function findmax(x in number, y in number)
  6   return number
  7   is
  8   z number;
  9   begin
 10   if x>y then
 11   z:=x;
 12   else
 13   z:=y;
 14   end if;
 15   return z;
 16   end;
 17  begin
 18   a:=23;
 19   b:=45;
 20  c:=findmax(a,b);
 21   dbms_output.put_line('Maximum(23,45):'||c);
 22   end;
 23   /
Maximum(23,45):45

PL/SQL procedure successfully completed.
```

12.2 PROCEDURS IN PL/SQL

Procedures in PL/SQL are similar to functions but differ in that they do not return a value. Instead, they perform a series of actions or tasks. Procedures can be used to update data, execute automation routines, or perform maintenance tasks within a Smart Home Automation System. They are often employed when it need to execute a sequence of SQL statements or operations.

```
SQL> set serveroutput on;
SQL> declare
  2  a number;
  3  b number;
  4  c number;
  5  procedure findmin(x in number, y in number, z out number) is
  6  begin
  7  if x<y then
  8  z:=x;
  9  else
 10  z:=y;
 11  end if;
 12  end;
 13  begin
 14  a:=23;
 15  b:=27;
 16  findmin(a,b,c);
 17  dbms_output.put_line('Minimum value='||c);
 18  end;
 19  /
Minimum value=23

PL/SQL procedure successfully completed.
```

12.3 TRIGGERS IN PL/SQL

Creating of table

A trigger for table creation is not a common use case, as typically, tables are created using Data Definition Language (DDL) statements rather than triggers. However, it can create a "BEFORE CREATE" trigger that logs information about table creation in a separate audit table. The trigger would capture details like the table name, creator, and creation date for auditing purposes.

```
SQL> create table room1(r_id int, name char(30), temp int, square int);
Table created.
SQL> create table room_details(r_id int, name char(30), temp int, square int);
Table created.
```

Insertion of values

To insert values into two tables simultaneously, can create an "AFTER INSERT" trigger on the source table. The trigger will fire after an insert operation and perform the corresponding inserts into the destination tables. For instance, in a Smart Home Automation System, could have a trigger that inserts information about a new user's registration into both the "Users" and "AccessLogs" tables to record user activity.

```
SQL> insert into room1 values('201','kitchen','50','150');
1 row created.
SQL> insert into room1 values('202','dining','65','200');
1 row created.
SQL> insert into room1 values('203','bedroom','50','300');
1 row created.
```

```
SQL> insert into room_details values('201','kitchen','50','150');
1 row created.
SQL> insert into room_details values('202','dining','65','200');
1 row created.
SQL> insert into room_details values('203','bedroom','50','300');
1 row created.
```

PL/SQL Program

To create a PL/SQL program for displaying tables, can develop a stored procedure or anonymous PL/SQL block that retrieves and presents data from the desired tables. In a Smart Home Automation System, The program might be used to display a list of connected devices and their status, or to show user details and their associated devices. The PL/SQL code can use SELECT statements to fetch the required data and format it for presentation.

```
SQL> set serveroutput on;
SQL> create trigger roo_detail_trigger
  2  before update of temp
  3   on room1
  4   for each row
  5   begin
  6   insert into room_details
  7   values
  8   (:old.r_id,
  9   :old.name,
 10   :old.temp,
 11   :old.square);
 12   end;
 13   /

Trigger created.
```

Displaying the tables

Triggers are typically used for automating actions in response to database events, such as INSERT, UPDATE, or DELETE operations. Displaying tables is typically done using SELECT statements and can be encapsulated in stored procedures or anonymous PL/SQL blocks for ease of use.

```
SQL>  update room1 set temp=90 where r_id=201;

1 row updated.
```

```
SQL> select * from room1;

     R_ID NAME                           TEMP     SQUARE
--------- ---------------------------- ---------- ----------
      201 kitchen                            90        150
      202 dining                             90        200
      202 dining                             90        200
```

12.4 CURSORS IN PL/SQL

Creating of table

Creating a cursor for table creation is an unusual scenario because tables are typically created using Data Definition Language (DDL) statements like CREATE TABLE. Cursors are more commonly used for querying and manipulating data in existing tables. However, if it need to dynamically generate DDL statements based on certain conditions, could use a cursor that selects DDL statements and executes them using dynamic SQL.

```
SQL> create table device(d_id int, name char(40), power int);

Table created.
```

Insertion of values

A cursor for inserting values into a table can be useful when required to insert data into a table based on the result set of a query. For instance, in a Smart Home Automation System. It might have a cursor that selects user data from one table and inserts it into another table for user access management. The cursor fetches records from the source table and inserts them into the destination table.

```
SQL> insert into device values(201,'lock',85);

1 row created.

SQL> insert into device values(202,'cctv',10);

1 row created.
```

PL/SQL Program

Cursors are often used for fetching data from tables. can create a cursor to query a specific table and retrieve data based on certain criteria. In the context of a Smart Home Automation System, it might have a cursor that fetches all the device records from the "Devices" table, allowing to process, analyse, or display it information in r PL/SQL program.

```
SQL> set serveroutput on;
SQL> declare
  2  powerinfo device%rowtype;
  3  cursor devicecursor is
  4   select *
  5   from device
  6   where power>20;
  7   begin
  8   open devicecursor;
  9   fetch devicecursor into
 10   powerinfo;
 11   dbms_output.put_line('d_id='||powerinfo.d_id);
 12   close devicecursor;
 13   end;
 14   /
d_id=201

PL/SQL procedure successfully completed.
```

CHAPTER XIII

CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT

12.1 CONCLUSION

In conclusion, the utilization of PL/SQL cursors is a powerful and versatile approach in managing and manipulating data within a Smart Home Automation System's database. While some scenarios mentioned, such as creating a table with cursors, are unconventional and typically achieved through Data Definition Language (DDL) statements, cursors play a crucial role in various data manipulation tasks.

PL/SQL cursors offer the capability to fetch and process data from existing tables, enabling custom data retrieval based on specified criteria or complex business logic. Its feature are invaluable for building functionalities that require dynamic and real-time data manipulation.

Moreover, cursors are frequently used to facilitate operations like data migration, transformation, and synchronization, especially when information from multiple sources or tables needs to be combined or analysed. For example, it might have a cursor to aggregate device data from various sensors and present it in a user-friendly format.

Cursors also lend themselves to procedural automation. By using cursors, it can systematically perform actions such as inserting, updating, or deleting records, and subsequently enhance the functionality of a Smart Home Automation System.

In the case of creating a PL/SQL program for displaying tables using cursors, it offers the flexibility to present structured data in an organized and user-friendly manner. Its function is vital for providing users with insights into their connected devices, automation rules, and energy consumption patterns.

In summary, PL/SQL cursors play a pivotal role in enhancing the capabilities of a Smart Home Automation System by providing efficient and dynamic data access, manipulation, and presentation. They contribute to the system's adaptability, intelligence, and user experience, ensuring that it remains a cutting-edge and highly functional solution for modern smart homes.

12.2 SCOPE FOR FUTURE ENHANCEMENT

The scope for future development in the realm of Smart Home Automation Systems is remarkably vast, promising an array of exciting possibilities. Firstly, as technology continues to advance, the integration of artificial intelligence (AI) and machine learning holds great potential. Smart homes will become even smarter as AI algorithms learn from user behaviours and preferences. It can lead to more proactive automation, such as adjusting lighting and temperature based on individual preferences, optimizing energy usage, and enhancing security measures. AI can also play a role in predictive maintenance for smart devices, reducing downtimes and ensuring consistent functionality.

Secondly, interoperability and standardization are key areas for development. As the Internet of Things (IoT) ecosystem expands, the seamless integration of various devices from different manufacturers becomes increasingly important. Future enhancements will involve creating common protocols and communication standards to ensure that devices can work harmoniously within a smart home environment. Its standardization will provide users with the freedom to choose devices from various brands without compatibility concerns, promoting a more open and competitive market.

Another major area of development is security and privacy. As smart homes become more interconnected, they become attractive targets for cyber threats. Future enhancements will involve implementing robust security measures, including advanced encryption, secure authentication methods, and intrusion detection systems. Additionally, privacy concerns related to the collection of user data will be addressed, ensuring that user information remains confidential and under their control.

Furthermore, energy efficiency and sustainability will continue to be at the forefront of smart home development. The integration of renewable energy sources, such as solar panels, and the optimization of energy consumption through smart algorithms will be pivotal. Smart homes will not only provide convenience but also contribute to reducing the environmental footprint by actively monitoring and managing energy usage.

In conclusion, the future of Smart Home Automation Systems holds immense potential for delivering ever-smarter, more secure, and energy-efficient living spaces. The development and integration of AI, interoperability standards, security measures, and sustainability initiatives will shape the smart homes of tomorrow, offering users a higher level of convenience, comfort, and peace of mind while also contributing to a greener and more sustainable future.

CHAPTER XIV

BIBLIOGRAPHY

13.1 BOOK REFERENCES

1. Mischel, W., & Baker, N. (1975). Cognitive transformations of reward objects through instructions. Journal of Personality and Social Psychology, 31, 254-261.
2. Mischel, W., & Baker, N. (1975). Cognitive transformations of reward objects through instructions. Journal of Personality and Social Psychology, 31, 254-261.
3. Mischel, W., & Baker, N. (1975). Cognitive transformations of reward objects through instructions. Journal of Personality and Social Psychology, 31, 254-261.
4. Mischel, W., & Baker, N. (1975). Cognitive transformations of reward objects through instructions. Journal of Personality and Social Psychology, 31, 254-261.
5. Mischel, W., & Baker, N. (1975). Cognitive transformations of reward objects through instructions. Journal of Personality and Social Psychology, 31, 254-261.

13.2 WEBSITE REFERENCES

1. https://www.sourcecodester.com/php/16525/lost-and-found-information-system-using-php-and-mysql-db-source-code-free-download.html
2. https://www.sourcecodester.com/php/16525/lost-and-found-information-system-using-php-and-mysql-db-source-code-free-download.html
3. https://www.sourcecodester.com/php/16525/lost-and-found-information-system-using-php-and-mysql-db-source-code-free-download.html
4. https://www.sourcecodester.com/php/16525/lost-and-found-information-system-using-php-and-mysql-db-source-code-free-download.html
5. https://www.sourcecodester.com/php/16525/lost-and-found-information-system-using-php-and-mysql-db-source-code-free-download.html