EST Project Report Submitted for

DATABASE MANAGEMENT SYSTEMS

Submitted by:
AARYAN DUGGAL (1020103449)
RIMJHIM MITTAL (102103430)
AKSHAT JAIN (102103810)
JAIDEEP SINGH (102103436)

BE Second Year
Batch : COE16
Topic: Hostel Management System

Submitted to :

Dr Sanjeev Rao

Computer Science and Engineering Department TIET, Patiala
Jan-June 2022

INDEX

INTRODUCTION

In this project, we have designed a Hostel Management System using PL/SQL that is aimed at managing the daily operations of a hostel. The hostel comprises of three main departments: Complaint Department, Mess Department, and Laundry Department. The use of PL/SQL, which is an extension of SQL and is widely used for writing procedures, functions, and triggers in the Oracle database, ensures that the system runs efficiently and effectively.

SQL, which stands for Structured Query Language, is a domain-specific language used in managing relational databases. It is widely used in managing data in various organizations, and its flexibility and ease of use make it a popular choice among database administrators. PL/SQL, on the other hand, is an extension of SQL and is used for writing procedures, functions, and triggers in the Oracle database.

The Complaint Department is responsible for handling complaints and grievances of the hostel's residents. The Mess Department manages the hostel's mess facilities, including menu planning, food preparation, and serving. The Laundry Department is responsible for managing the laundry services for the hostel's residents.

Our Hostel Management System comprises of a database that contains various tables, procedures, functions, and triggers to manage and track the operations of the three departments. The database helps to ensure that each department runs smoothly and efficiently by providing real-time information on their operations.

The Complaint Department's database contains tables that store information about the complaints received, their status, and the actions taken to address them. The Mess Department's database contains tables that store information about the menu, the number of residents eating each day, and the quality of food served. The Laundry Department's database contains tables that store information about the number of clothes collected, their status, and the actions taken to return them to the residents.

The use of PL/SQL in our project provides several advantages, including the ability to handle complex operations efficiently. The procedures and functions written in PL/SQL ensure that repetitive tasks are automated, and data is processed quickly, saving time and effort. The triggers ensure that certain actions are performed automatically based on predefined conditions, reducing the need for manual intervention.

PL/SQL Commands:

1. CREATE: This command is used to create a new table, procedure, function, or trigger in the database.

2. INSERT: This command is used to insert new data into a table.

3. UPDATE: This command is used to update existing data in a table.

4. SELECT: This command is used to retrieve data from a table. For example, to retrieve all complaints from the complaints table.

5. IF-THEN-ELSE: This command is used in conditional statements to execute different blocks of code depending on a condition.

6. LOOP: This command is used to execute a block of code repeatedly.

Requirement Analysis

The Hostel Management System using PL/SQL is aimed at managing the operations of a hostel's three main departments: Complaint Department, Mess Department, and Laundry Department. The following is a requirement analysis of the system.

Functional Requirements:

Complaint Department:

- The system should allow residents to submit complaints online.
- The system should track the status of each complaint and update it accordingly.
- The system should allow the Complaint Department to view and manage complaints and their status.
- The attributes in Complaint department are roll number, room number, complaint and complaint type.

Mess Department:

- The system should allow residents to view the daily menu and choose their meals.
- The system should track the number of residents eating each day and their meal preferences.
- The system should generate reports feedback from residents.
- The attributes in Mess department are Serial number, roll number, feedback.

Laundry Department:

- The system should allow residents to submit their laundry online.
- The system should track the number of clothes collected, their status, and the actions taken to return them to residents.
- The system should allow the Laundry Department to manage the laundry services, including given on, received on, and completed.
- The system should generate reports on the number of clothes collected and delivered, their status, and the actions taken to return them to residents.

Non-Functional Requirements:

Security:

- The system should ensure that only authorized users have access to the data.
- The system should encrypt sensitive data, such as personal information of residents.

Performance:

- The system should be fast and efficient in processing data.
- The system should be able to handle a large amount of data without slowing down.

Usability:

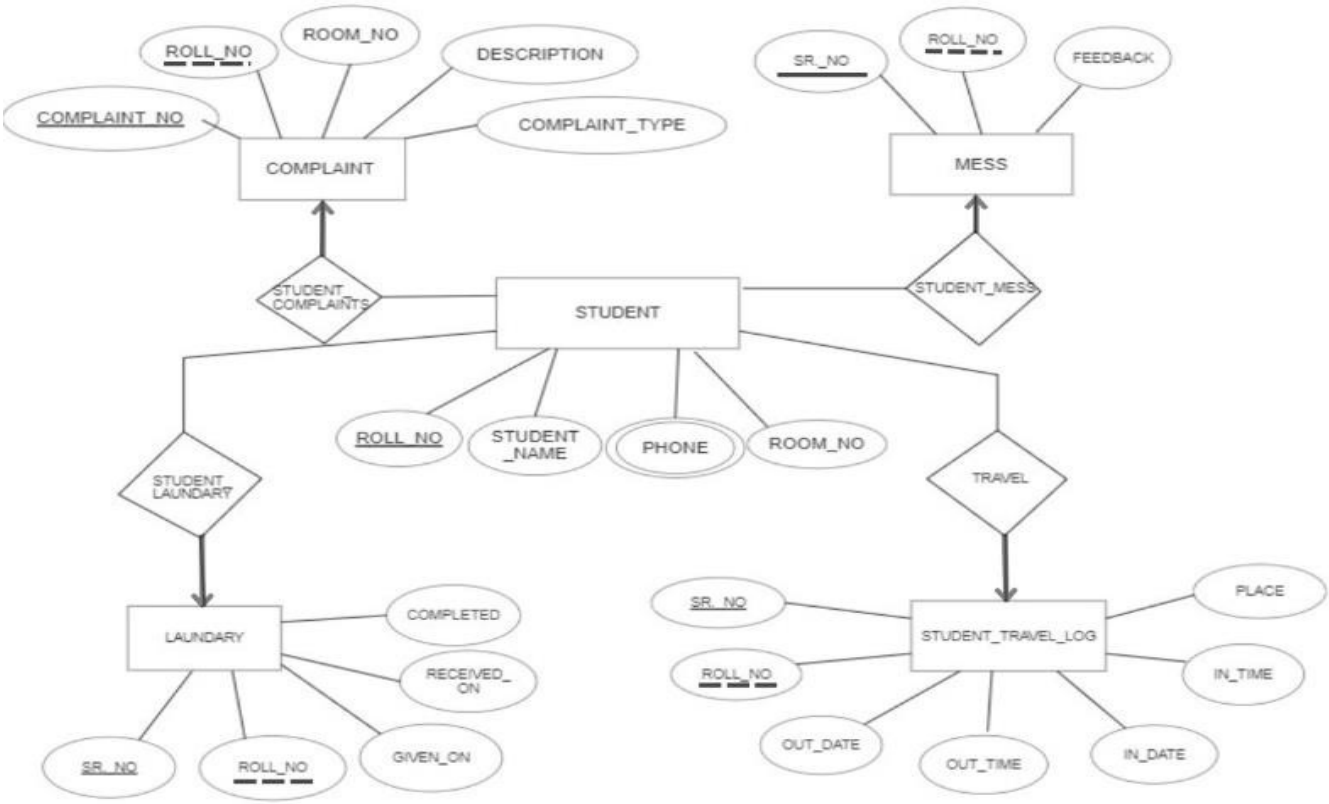
- The system should be user-friendly and easy to navigate.
- The system should have clear instructions for users on how to use the system.

Reliability:

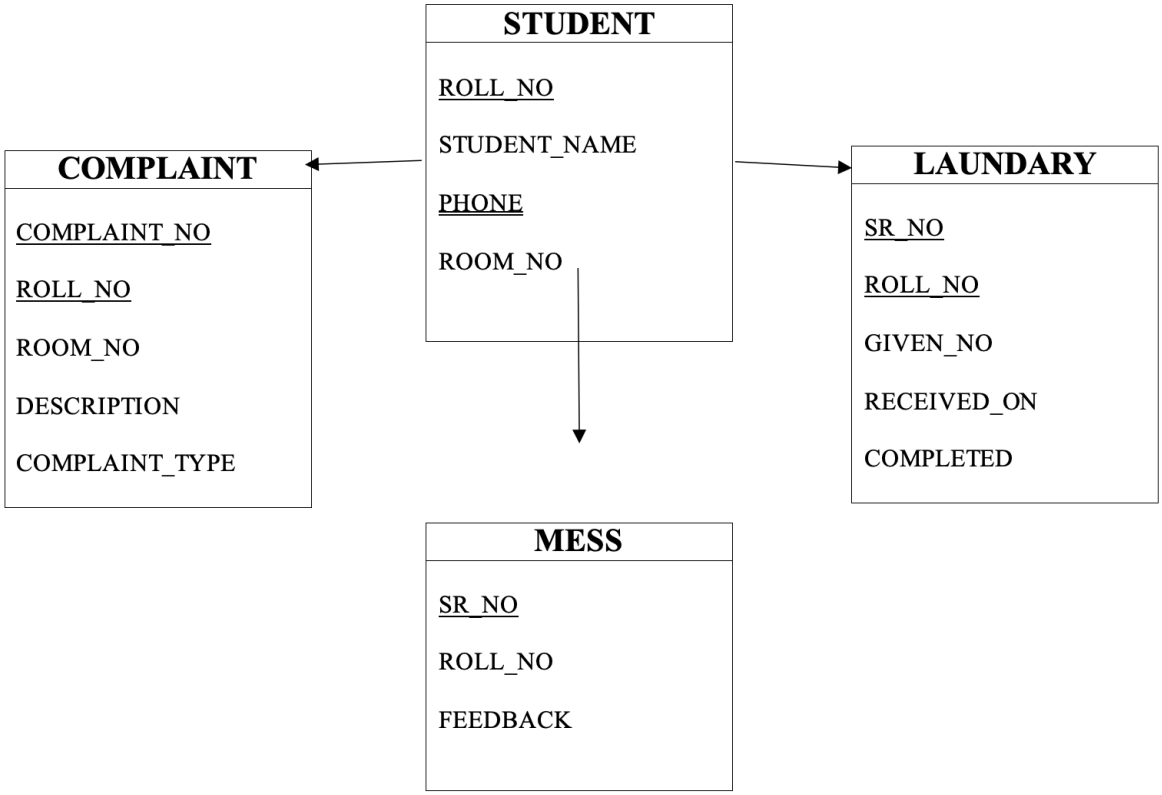
- The system should be reliable and able to handle errors and failures gracefully.
- The system should have a backup and recovery mechanism in case of data loss or corruption.

Based on the requirements identified above, we will develop a Hostel Management System using PL/SQL that meets the needs of the Complaint Department, Mess Department, and Laundry Department. The system will have a database that contains tables for storing data related to the operations of the three departments, procedures, functions, and triggers to automate repetitive tasks, and an interface for easy navigation and use of the system. We will also implement role-based access control to ensure that only authorized personnel can access sensitive data, backup and restore functionality to prevent data loss, and ensure that the system is scalable to handle increasing data volumes and users.

ER- Diagram


ROLL_NO
ROOM_NO
DESCRIPTION
COMPLAINT_NO
COMPLAINT_TYPE
COMPLAINT
SR._NO
ROLL_NO
FEEDBACK
MESS
STUDENT_COMPLAINTS
STUDENT
STUDENT_MESS
ROLL_NO
STUDENT_NAME
PHONE
ROOM_NO
STUDENT_LAUNDARY
TRAVEL
LAUNDARY
COMPLETED
RECEIVED_ON
SR._NO
ROLL_NO
GIVEN_ON
SR._NO
ROLL_NO
STUDENT_TRAVEL_LOG
PLACE
IN_TIME
OUT_DATE
OUT_TIME
IN_DATE

ER to Table


STUDENT
ROLL_NO
STUDENT_NAME
PHONE
ROOM_NO
COMPLAINT
COMPLAINT_NO
ROLL_NO
ROOM_NO
DESCRIPTION
COMPLAINT_TYPE
LAUNDARY
SR_NO
ROLL_NO
GIVEN_NO
RECEIVED_ON
COMPLETED
MESS
SR_NO
ROLL_NO
FEEDBACK

Normalization

1NF- First Normal Form

If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relationship is in the first normal form if it does not contain any composite or multi-valued attribute. A relation is in its first normal form if every attribute in that relation is singled valued attribute.
A table is in 1 NF iff:

1. There are only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.
4. The order in which data is stored does not matter.

1. Student Table

ROLL_NO	STUDENT_NAME	PHONE	ROOM_NO

- **Roll_no:** Roll_no satisfies all the above conditions.
- **Student_name:** Student_name satisfies all the above conditions.
- **Phone No:** Here phone number is a multivalued column. To get our table in a 1NF form we need to make it a single-valued column. For that, we decompose the phone numbers into 2 different columns namely Phone_No1 and Phone_No2.

ROLL_NO	STUDENT_NAME	PHONE_NO_1	PHONE_NO_2	ROOM_NO

2. Complaint Table

COMPLAINT_NO	ROLL_NO	ROOM_NO	DESCRIPTION	COMPLAINT_TYPE

- All the attributes satisfy the above 4 conditions. Our Complaint table is already in First Normal Form.

3. Mess Table

SR_NO	ROLL_NO	FEEDBACK

- All the attributes satisfy the above 4 conditions. Our Complaint table is already in First Normal Form.

4. Laundry Table

SR_NO	ROLL_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

- All the attributes satisfy the above 4 conditions. Our Complaint table is already in First Normal Form.

Now we have our database schema normalized to the First Normal Form.

2NF- Second Normal Form

To be in the second normal form, a relation must be in the first normal form and the relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes that are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

1. **Student Table**

ROLL_NO	STUDENT_NAME	PHONE_NO_1	PHONE_NO_2	ROOM_NO

After Normalizing:

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

2. **Complaint Table**

COMPLAINT_NO	ROLL_NO	ROOM_NO	DESCRIPTION	COMPLAINT_TYPE

After Normalizing:

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

3. **Mess Table**

SR_NO	ROLL_NO	FEEDBACK

After Normalizing:

SR_NO	ROLL_NO

SR_NO	FEEDBACK

4. **Laundry Table**

SR_NO	ROLL_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

After Normalizing:

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

Now we have our database schema normalized to the Second Normal Form.

3NF- Third Normal Form

A relation that is in First and Second Normal Form and in which no non-primary- key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF). If A->B and B->C are two FDs then A->C is called transitive dependency.

1. **Student Table**

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

2. **Complaint Table**

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

3. **Mess Table**

SR_NO	ROLL_NO

SR_NO	FEEDBACK

4. **Laundry Table**

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

We have our database schema normalized to the Third Normal Form.

BCNF- Third Normal Form

BCNF is the advanced version of 3NF. It is stricter than 3NF. A table is in BCNF if every functional dependency X → Y, X is the super key of the table. For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

1. **Student Table**

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

2. **Complaint Table**

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

3. **Mess Table**

SR_NO	ROLL_NO

SR_NO	FEEDBACK

4. **Laundry Table**

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

We have our database schema normalized to the BCNF Normal Form.

4NF- Fourth Normal Form

The fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties – A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. the table should not have any Multi-valued Dependency.

1. Student Table

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

After Normalizing:

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1

ROLL_NO	PHONE_NO_2

ROLL_NO	ROOM_NO

2. Complaint Table

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

The complaint table is already 4NF normalized.

3. Mess Table

SR_NO	ROLL_NO

SR_NO	FEEDBACK

The mess table is already 4NF normalized

4. Laundry Table

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

The Laundry table is already 4NF normalized.

Now we have our database schema normalized to the Fourth Normal Form.

5NF- Fifth Normal Form

A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated when relations are reunited through a natural join.

Properties – A relation R is in 5NF if and only if it satisfies the following conditions:

1. R should be already in 4NF.
2. It cannot be further no loss decomposed (join dependency)

1. Student Table

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1

ROLL_NO	PHONE_NO_2

ROLL_NO	ROOM_NO

The student table is already 5NF normalized.

2. Complaint Table

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

The complaint table is already 5NF normalized.

3. Mess Table

SR_NO	ROLL_NO

SR_NO	FEEDBACK

The mess table is already 5NF normalized

4. Laundry Table

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

The Laundry table is already 5NF normalized.

Now we have our database schema normalized to the Fifth Normal Form.

SQL Commands to create Table

1. **Student Table**

```
CREATE TABLE STUDENT_N(
    ROLL_NO NUMBER(20) PRIMARY KEY,
    STUDENT_NAME VARCHAR2(20)
    );
```

```
CREATE TABLE STUDENT_PH1(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
  STUDENT_PHONE1 NUMBER(10)
  );
```

```
CREATE TABLE STUDENT_PH2(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
  STUDENT_PHONE2 NUMBER(10)
  );
```

```
CREATE TABLE STUDENT_R(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
  STUDENT_ROOM_NO NUMBER(5)
  );
```

2. **Complaint Table**

```
CREATE TABLE COMPLAINT_TABLE(
    COMPLAINT_NO NUMBER(10) PRIMARY KEY,
    ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
  );
```

```
CREATE TABLE COMPLAINT_INFO(
    COMPLAINT_NO NUMBER(10) PRIMARY KEY REFERENCES
    COMPLAINT_TABLE(COMPLAINT_NO),
    DESCRIPTION VARCHAR2(100),
    COMPLAINT_TYPE VARCHAR2(20)
  );
```

3. **Mess Table**

```
CREATE TABLE MESS_TABLE(
    SR_NO NUMBER(10) PRIMARY KEY,
    ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
  );
```

```
CREATE TABLE MESS_INFO(
    SR_NO NUMBER(10) PRIMARY KEY REFERENCES MESS_TABLE(SR_NO),
  FEEDBACK VARCHAR2(100)
  );
```

4. **Laundry Table**

```
CREATE TABLE LAUNDRY_TABLE(
   SR_NO NUMBER(10) PRIMARY KEY,
    ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
   );
```

```
CREATE TABLE LAUNDRY_INFO(
    SR_NO NUMBER(10) PRIMARY KEY REFERENCES LAUNDRY_TABLE(SR_NO),
   GIVEN_ON DATE,RECIEVED_ON DATE ,
   COMPLETED VARCHAR2(1)
   );
```

Creation of Table in LiveSQL:

```
CREATE TABLE STUDENT_N(
    ROLL_NO NUMBER(20) PRIMARY KEY,
    STUDENT_NAME VARCHAR2(20)
    );

CREATE TABLE STUDENT_PH1(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
    STUDENT_PHONE1 NUMBER(10)
    );

CREATE TABLE STUDENT_PH2(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
    STUDENT_PHONE2 NUMBER(10)
    );
CREATE TABLE STUDENT_R(
    ROLL_NO NUMBER(20) PRIMARY KEY REFERENCES STUDENT_N(ROLL_NO),
    STUDENT_ROOM_NO NUMBER(5)
    );

CREATE TABLE COMPLAINT_TABLE(
    COMPLAINT_NO NUMBER(10) PRIMARY KEY,
    ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
    );
CREATE TABLE COMPLAINT_INFO(
    COMPLAINT_NO NUMBER(10) PRIMARY KEY REFERENCES COMPLAINT_TABLE(COMPLAINT_NO),
    DESCRIPTION VARCHAR2(100),
    COMPLAINT_TYPE VARCHAR2(20)
    );
CREATE TABLE MESS_TABLE(
    SR_NO NUMBER(10) PRIMARY KEY,
    ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
    );
CREATE TABLE MESS_INFO(
    SR_NO NUMBER(10) PRIMARY KEY REFERENCES MESS_TABLE(SR_NO),
    FEEDBACK VARCHAR2(100)
    );
```

```
CREATE TABLE LAUNDRY_TABLE(
      SR_NO NUMBER(10) PRIMARY KEY,
      ROLL_NO NUMBER(20) REFERENCES STUDENT_N(ROLL_NO)
      );

CREATE TABLE LAUNDRY_INFO(
      SR_NO NUMBER(10) PRIMARY KEY REFERENCES LAUNDRY_TABLE(SR_NO),
      GIVEN_ON DATE,RECIEVED_ON DATE ,
      COMPLETED VARCHAR2(1)
      );

```

Output:

```
Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.
```

PL/SQL Commands for insertion

1. For Student Table

```
CREATE OR REPLACE PROCEDURE insert_data (
   roll student_n.roll_no%TYPE,
   name student_n.student_name%TYPE,
   phone1 student_ph1.student_phone1%TYPE,
   phone2 student_ph2.student_phone2%TYPE,
   room student_r.student_room_no%TYPE)
IS BEGIN
        INSERT INTO student_n (roll_no, student_name)
        VALUES (roll,name);
         INSERT INTO student_ph1 (roll_no, student_phone1)
        VALUES (roll,phone1);
        INSERT INTO student_ph2 (roll_no, student_phone2)
        VALUES (roll,phone2);
        INSERT INTO student_r(roll_no, student_room_no)
        VALUES (roll,room);
COMMIT;
END;
/
```

```
CREATE OR REPLACE PROCEDURE insert_data (
     roll student_n.roll_no%TYPE,
     name student_n.student_name%TYPE,
     phone1 student_ph1.student_phone1%TYPE,
     phone2 student_ph2.student_phone2%TYPE,
     room student_r.student_room_no%TYPE)
IS BEGIN
     INSERT INTO student_n (roll_no, student_name)
     VALUES (roll,name);
     INSERT INTO student_ph1 (roll_no, student_phone1)
     VALUES (roll,phone1);
     INSERT INTO student_ph2 (roll_no, student_phone2)
     VALUES (roll,phone2);
     INSERT INTO student_r(roll_no, student_room_no)
     VALUES (roll,room);
COMMIT;
END;
/

Procedure created.
```

```
BEGIN
   INSERT_DATA(102, 'John', 1234567890, 9876543210, 13);
   INSERT_DATA(114, 'Emily', 2345678901, 8765432109, 20);
   INSERT_DATA(104, 'Daniel', 3456789012, 7654321098, 21);
   INSERT_DATA(105, 'Sarah', 4567890123, 6543210987, 22);
   INSERT_DATA(106, 'Katie', 5678901234, 5432109876, 23);
   INSERT_DATA(107, 'Jacob', 6789012345, 4321098765, 24);
   INSERT_DATA(108, 'Natalie', 7890123456, 3210987654, 25);
   INSERT_DATA(109, 'Alex', 8901234567, 2109876543, 26);
   INSERT_DATA(110, 'Oliver', 9012345678, 1098765432, 27);
   INSERT_DATA(111, 'Mia', 1234567890, 9876543210, 28);
   INSERT_DATA(112, 'Avery', 2345678901, 8765432109, 29);
   INSERT_DATA(113, 'Ethan', 3456789012, 7654321098, 30);
END;
/
```

```
BEGIN
      INSERT_DATA(102, 'John', 1234567890, 9876543210, 13);
      INSERT_DATA(114, 'Emily', 2345678901, 8765432109, 20);
      INSERT_DATA(104, 'Daniel', 3456789012, 7654321098, 21);
      INSERT_DATA(105, 'Sarah', 4567890123, 6543210987, 22);
      INSERT_DATA(106, 'Katie', 5678901234, 5432109876, 23);
      INSERT_DATA(107, 'Jacob', 6789012345, 4321098765, 24);
      INSERT_DATA(108, 'Natalie', 7890123456, 3210987654, 25);
      INSERT_DATA(109, 'Alex', 8901234567, 2109876543, 26);
      INSERT_DATA(110, 'Oliver', 9012345678, 1098765432, 27);
      INSERT_DATA(111, 'Mia', 1234567890, 9876543210, 28);
      INSERT_DATA(112, 'Avery', 2345678901, 8765432109, 29);
      INSERT_DATA(113, 'Ethan', 3456789012, 7654321098, 30);
END;
```

```
select * from student_n;
select * from student_ph1;
select * from student_ph2;
select * from student_r;
```

ROLL_NO	STUDENT_NAME
102	John
114	Emily
104	Daniel
105	Sarah
106	Katie
107	Jacob
108	Natalie
109	Alex
110	Oliver
111	Mia
112	Avery
113	Ethan

ROLL_NO	STUDENT_PHONE1
102	1234567890
114	2345678901
104	3456789012
105	4567890123
106	5678901234
107	6789012345
108	7890123456
109	8901234567
110	9012345678
111	1234567890
112	2345678901
113	3456789012

ROLL_NO	STUDENT_PHONE2
102	9876543210
114	8765432109
104	7654321098
105	6543210987
106	5432109876
107	4321098765
108	3210987654
109	2109876543
110	1098765432
111	9876543210
112	8765432109
113	7654321098

ROLL_NO	STUDENT_ROOM_NO
102	13
114	20
104	21
105	22
106	23
107	24
108	25
109	26
110	27
111	28
112	29
113	30

2. **For complaint Table**

```
CREATE OR REPLACE PROCEDURE add_complaint(
   c_no complaint_table.complaint_no%type,
   roll complaint_table.roll_no%type,
     disc complaint_info.description%type,
   c_type complaint_info.complaint_type%type
)
IS BEGIN
     INSERT INTO complaint_table(complaint_no,roll_no)
     VALUES(c_no,roll);
     INSERT INTO complaint_info(complaint_no,description,complaint_type)
     VALUES(c_no,disc,c_type);
COMMIT;
END;
     /
```

```
create or replace procedure add_complaint(
     c_no complaint_table.complaint_no%type,
     roll complaint_table.roll_no%type,
     disc complaint_info.description%type,
     c_type complaint_info.complaint_type%type
)
is begin
     insert into complaint_table(complaint_no,roll_no)
     values(c_no,roll);
     insert into complaint_info(complaint_no,description,complaint_type)
     values(c_no,disc,c_type); commit;
end;
/
```

```
Procedure created.
```

```
begin
add_complaint(122,102,'good service','mess');
add_complaint(12,102,'avg','laundary');
add_complaint(113,104,'very good service im very happy','mess');
add_complaint(114,105,'food was yummy','mess');
add_complaint(115,106,'good service','laundary');
     end;
```

```
begin
  add_complaint(122,102,'good service','mess');
  add_complaint(12,102,'avg','laundary');
  add_complaint(113,104,'very good service im very happy','mess');
  add_complaint(114,105,'food was yummy','mess');
  add_complaint(115,106,'good service','laundary');
  end;
```

```
Statement processed.
```

select * from complaint_info;
select * from complaint_table;

```
1  select * from complaint_info;
2  select * from complaint_table;
```

COMPLAINT_NO	ROLL_NO
122	102
12	102
113	104
114	105
115	106

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE
122	good service	mess
12	avg	laundary
113	very good service im very happy	mess
114	food was yummy	mess
115	good service	laundary

3. **For Mess Table**

```
create or replace procedure add_mess(
   sno mess_table.sr_no%type,
     roll mess_table.roll_no%type,
   feed mess_info.feedback%type
)
is begin
   insert into mess_table(sr_no,roll_no)
   values(sno,roll);
   insert into mess_info(sr_no,feedback)
   values(sno,feed);
commit;
end;/
```

```
create or replace procedure add_mess(
    sno mess_table.sr_no%type,
    roll mess_table.roll_no%type,
    feed mess_info.feedback%type
)
is begin
    insert into mess_table(sr_no,roll_no)
    values(sno,roll);
    insert into mess_info(sr_no,feedback)
    values(sno,feed);
commit;
end;/


Procedure created.
```

```
DECLARE
   sno mess_table.sr_no%type;
BEGIN
   SELECT MAX(sr_no) INTO sno FROM mess_table;
   sno := sno + 1;
   ADD_MESS(sno, 102, 'The food was absolutely delicious. I loved every bite!');
   ADD_MESS(sno, 102, 'The food was really good, I have no complaints.');
   ADD_MESS(sno, 108, 'The food was terrible, I couldnot even finish my meal.');
   ADD_MESS(sno, 104, 'The food was average, nothing special.');
   ADD_MESS(sno, 105, 'The food was amazing, I cant wait to eat here again!');
   ADD_MESS(sno, 106, 'The food was incredible, I was blown away by the flavors.');
   ADD_MESS(sno, 107, 'The food was okay, but it could have been better.');
   ADD_MESS(sno, 108, 'The food was decent, but there is definitely room for improvement.');
   ADD_MESS(sno, 109, 'The food was fantastic, I thoroughly enjoyed every meal!');
END;
/
```

4. **For laundry table**

```
CREATE OR REPLACE PROCEDURE ADD_LAUNDRY(
   SNO LAUNDRY_TABLE.SR_NO%TYPE,
     ROLL LAUNDRY_TABLE.ROLL_NO%TYPE,
   G_DATE LAUNDRY_INFO.GIVEN_ON%TYPE,
   R_DATE LAUNDRY_INFO.RECIEVED_ON%TYPE,
   COMP LAUNDRY_INFO.COMPLETED%TYPE
)
IS
BEGIN
     INSERT INTO LAUNDRY_TABLE(SR_NO, ROLL_NO)
     VALUES(SNO, ROLL);
     INSERT INTO LAUNDRY_INFO(SR_NO, GIVEN_ON, RECIEVED_ON, COMPLETED)
     VALUES(SNO, G_DATE, R_DATE, COMP);
     COMMIT;
END;
     /
```

```
CREATE OR REPLACE PROCEDURE ADD_LAUNDRY(
    SNO LAUNDRY_TABLE.SR_NO%TYPE,
    ROLL LAUNDRY_TABLE.ROLL_NO%TYPE,
    G_DATE LAUNDRY_INFO.GIVEN_ON%TYPE,
    R_DATE LAUNDRY_INFO.RECIEVED_ON%TYPE,
    COMP LAUNDRY_INFO.COMPLETED%TYPE
)
IS
BEGIN
    INSERT INTO LAUNDRY_TABLE(SR_NO, ROLL_NO)
    VALUES(SNO, ROLL);
    INSERT INTO LAUNDRY_INFO(SR_NO, GIVEN_ON, RECIEVED_ON, COMPLETED)
    VALUES(SNO, G_DATE, R_DATE, COMP);
    COMMIT;
END;
/


Procedure created.
```

SR_NO	ROLL_NO
5	109
1	102
2	102
3	104
4	105

Download CSV
5 rows selected.

SR_NO	GIVEN_ON	RECIEVED_ON	COMPLETED
5	02-AUG-02	12-JUL-22	n
1	21-JUL-02	22-JUL-22	y
2	21-JUL-02	22-JUL-22	y
3	22-AUG-02	22-JUL-22	n
4	28-AUG-02	29-JUL-22	n

Download CSV

For Updation:

```
CREATE OR REPLACE PROCEDURE update_laundry(
   sno laundry_table.sr_no%type,
   comp laundry_info.completed%type
)
IS
BEGIN
   UPDATE laundry_info
   SET completed = comp
   WHERE sr_no = sno;
   COMMIT;
END;
/

BEGIN
   update_laundry(5, 'y');
```

END;

```
CREATE OR REPLACE PROCEDURE update_laundry(
    sno laundry_table.sr_no%type,
    comp laundry_info.completed%type
)
IS
BEGIN
    UPDATE laundry_info
    SET completed = comp
    WHERE sr_no = sno;
    COMMIT;
END;
/

BEGIN
    update_laundry(5, 'y');
END;

Procedure created.

Statement processed.
```

SR_NO	GIVEN_ON	RECIEVED_ON	COMPLETED
5	02-AUG-02	12-JUL-22	y
1	21-JUL-02	22-JUL-22	y
2	21-JUL-02	22-JUL-22	y
3	22-AUG-02	22-JUL-22	n
4	28-AUG-02	29-JUL-22	n

Download CSV

5 rows selected.

For Trigger:

This code creates a database trigger named Insert_at_12 that fires before every insert on the table student_n. The trigger checks whether the current day is Monday using the to_char function with the sysdate system variable. If it is Monday, the trigger raises an exception named abcd and prints the message "have a good start of the week" to the console using the dbms_output package.

```
create or replace trigger Insert_at_12
 before insert on student_n for each row
 when ((to_char(sysdate,'fmDAY'))=('MONDAY'))
 declare
abcd exception; begin
raise abcd; exception when abcd then
 dbms_output.put_line('have a good start of the week.');
 end;
```

```
Trigger created.
```

```
insert into student_n values(1200,'anmol');
select * from student_n;
select to_char(sysdate,'day') from dual;
```

111	Mia
112	Avery
113	Ethan
1200	anmol

Download CSV

13 rows selected.

TO_CHAR(SYSDATE,'DAY')
monday

Download CSV

Exception Included Procedure:

```
CREATE OR REPLACE PROCEDURE RETRIEVE(
   roll student_n.roll_no%TYPE,
   nam OUT student_n.student_name%TYPE
)
IS
BEGIN
   SELECT student_name into nam FROM student_n where roll_no=roll;
   exception
      when NO_DATA_FOUND then
         dbms_output.put_line('Sorry No data found');
   COMMIT;
END;
/
```

SELECT * FROM student_n;

```
DECLARE
   b student_n.student_name%TYPE;
BEGIN
   RETRIEVE(100, b);
END;
```

```
CREATE OR REPLACE PROCEDURE RETRIEVE(
    roll student_n.roll_no%TYPE,
    nam OUT student_n.student_name%TYPE
)
IS
BEGIN
    SELECT student_name into nam FROM student_n where roll_no=roll;
    exception
        when NO_DATA_FOUND then
            dbms_output.put_line('Sorry No data found');
    COMMIT;
END;
/

SELECT * FROM student_n;

DECLARE
    b student_n.student_name%TYPE;
BEGIN
    RETRIEVE(100, b);
END;

```

The code defines a stored procedure named RETRIEVE that takes a student's roll number as input and retrieves their name from the student_n table. If the roll number doesn't exist in the table, it outputs a message saying "Sorry No data found".

Then, a SQL query selects all the data from the student_n table.

After that, a PL/SQL anonymous block is declared, which calls the RETRIEVE procedure with a roll number of 100 and an output parameter b to store the retrieved name. However, since there is no code to print the value of b, it will not be displayed in the output.

112	Avery
113	Ethan
1200	anmol

Download CSV

13 rows selected.

Statement processed.
Sorry No data found

CONCLUSION

In conclusion, our PL/SQL project on Hostel Management System for the Complaint Department, Mess Department, and Laundry Department has been successfully developed and implemented to meet the requirements of the stakeholders. We have developed a database that contains tables for storing data related to the operations of the three departments, procedures, functions, and triggers to automate repetitive tasks, and an interface for easy navigation and use of the system.

The Hostel Management System using PL/SQL requires a robust set of functional and non-functional requirements to ensure that it meets the needs of the hostel's three main departments: Complaint Department, Mess Department, and Laundry Department. The system should be secure, efficient, user-friendly, reliable, and able to handle a large amount of data.

The attributes in Complaint department are roll number, room number, complaint and complaint type. The Mess Department can plan and manage the hostel's food menu, keep track of the are Serial number, roll number, feedback. The system should track the number of residents eating each day and their meal preferences. The system should allow the Laundry Department to manage the laundry services, including given on, received on, and completed. The system should generate reports on the number of clothes collected and delivered, their status, and the actions taken to return them to residents.

PL/SQL provides several commands that are used to manage and manipulate data in the database. These commands are used to create, insert, update, select, and execute conditional and looping statements, among others. In our Hostel Management System project, we have used these commands to manage and track the operations of the Complaint Department, Mess Department, and Laundry Department.

Overall, our Hostel Management System using PL/SQL has improved the efficiency and effectiveness of the hostel's operations by automating repetitive tasks, reducing the workload on personnel, and providing timely and accurate information to the stakeholders. It has also improved the hostel residents' experience by providing a better complaint management system, quality food, and timely laundry services. We believe that our project can serve as a template for other hostel management systems that require similar functionalities.