HOSTEL MANAGEMENT SYSTEM

DBMS PROJECT

A PROJECT UNDER PROF ARCHANA

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

SUBMITTED BY:

KARTIK GUPTA	102203930
ARUL GUPTA	102203942
MANECK	102213029
ARYAN JOHARY	102203086

TABLE OF CONTENT

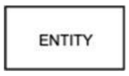
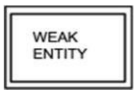
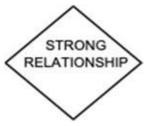
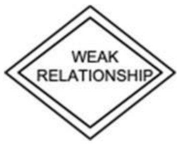
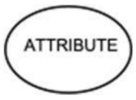
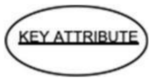
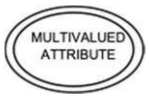
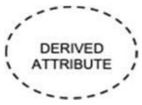
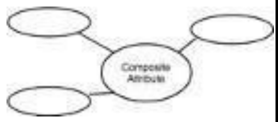
BASIC INTRODUCTION

1. **DATABASE MANAGEMENT SYSTEM:** A Database Management System (DBMS) is a software system that enables users to create, manage, and access databases. A database is a collection of data that is organized in a way that makes it easy to search, retrieve, update, and delete data. A DBMS provides a set of tools and functions for managing databases, including creating and modifying database schemas, storing and retrieving data, and controlling access to the database.

2. **RELATIONAL DATABASE MANAGEMENT SYSTEM:** A relational database management system (RDBMS) is a type of database management system (DBMS) that is based on the relational model of data. The relational model organizes data into one or more tables, where each table consists of rows and columns. Each column in a table represents a specific attribute of the data, and each row represents a specific instance or record.

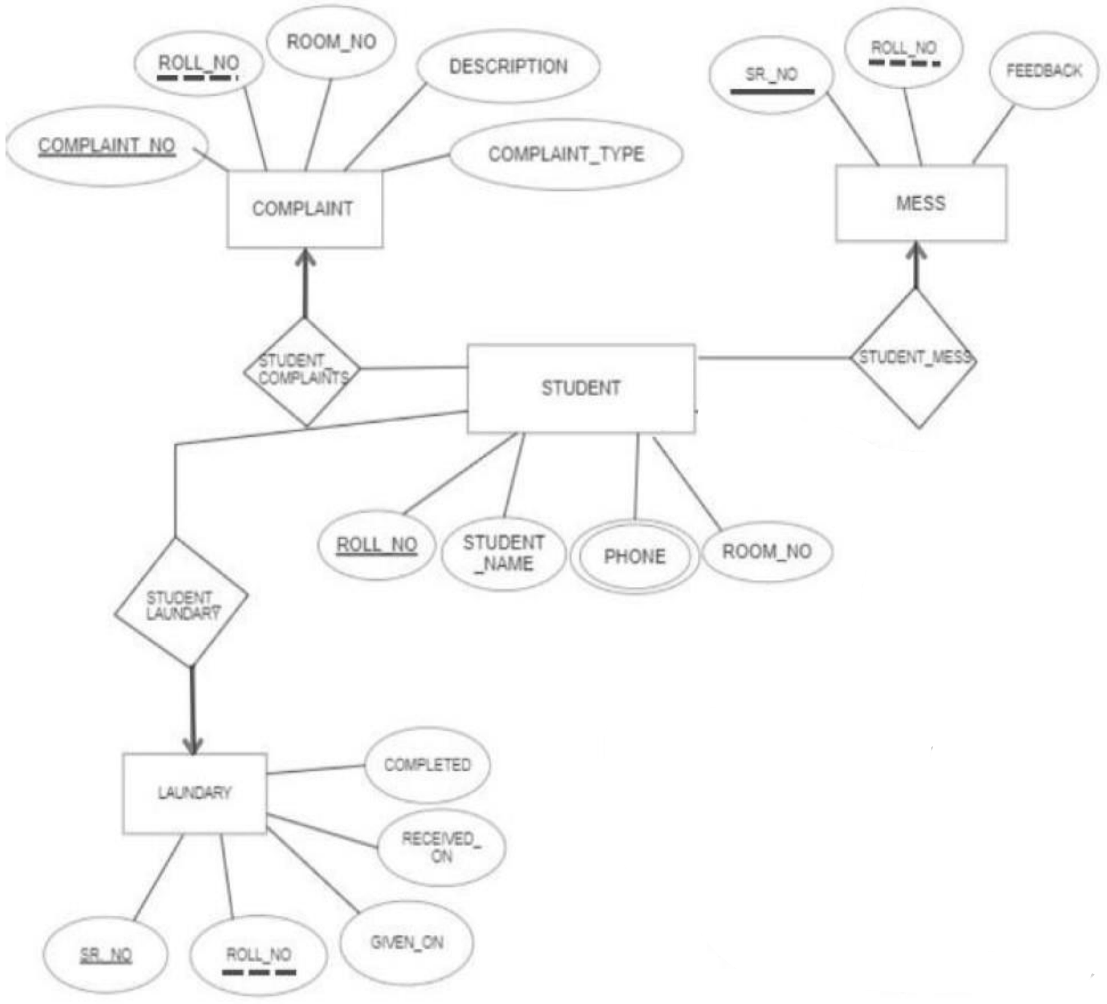
3. **ER DIAGRAM:** An Entity-Relationship (ER) diagram is a type of diagram used to represent the relationships between entities in a database. The ER diagram is a graphical representation of the entities and their relationships to each other. It is a high-level view of the database schema, showing the main entities, attributes, and relationships. In an ER diagram, entities are represented as rectangles, and the relationships between entities are represented as diamonds. Each entity has a set of attributes, which are represented as ovals or circles within the entity rectangle.

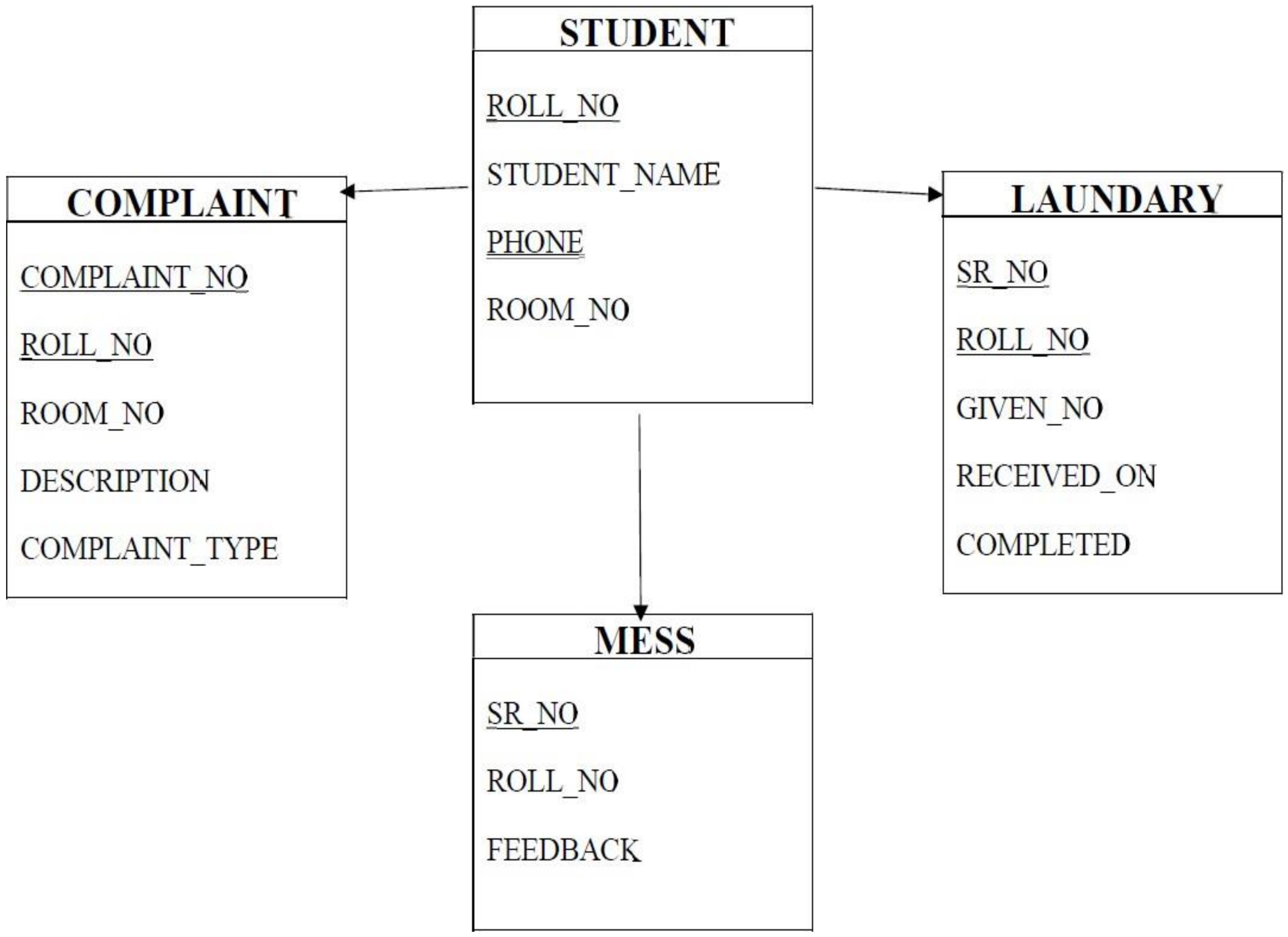
4. TERMINOLOGIES AND SYMBOLS OF ER DIAGRAM:

SYMBOL	DESCRIPTION
	This is a basic entity that is represented by a rectangle with its name inside.
	This is an entity that cannot solely be identified with its attributes (due to the absence of a primary key). It inherits the identifier of its parent entity and often integrates it with a partial key.
	A strong relationship is depicted by a single rhombus with its name inside. In this, an entity is independent, that is, its primary key for any child doesn't contain the primary key of the linked entity.
	A weak relationship is depicted by a double rhombus with the name inside. In this, the child is. Dependent on the parent entity as its primary key would contain a component of the parent's primary key.
	A basic attribute is represented by a single oval with its name written inside.
	This is a special attribute that is used to uniquely identify an entity. It is represented by an oval with its name underlined.
	These are the attributes that can have multiple values (like the Name attribute can have First and Last name) and are represented by a double oval.
	A derived attribute might not be physically present in the database and could be logically derived from any other attribute (Represented by a dotted oval).
	Composite attributes are those attributes which are composed of many other simple attributes.
—	This depicts that not all the entities in the set are a part of the relationship and is depicted by a single line.
═	This means that all the entities in the set are in a relationship and are depicted by a double line.

HOSTEL MANAGEMENT SYSTEM

ER DIAGRAM




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

AIM:-

Our project is based on Hostel Management. In our project, we have tried to modernize the conventional file-based registries still being used.

Description:

In this project, we have focused on 3 main departments namely Complaint Department, Mess Department, and the laundry department. In this project, we have used technologies like SQL and PL/SQL for various operations that can be performed in our database.

Normalization Process:

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 a single valued attribute. A table is in 1 NF if:

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.

Student Table

Roll No -- Roll No column satisfies all the above conditions.

Student_Name – Student_Name column satisfies all the above conditions.

Room_No – Room_no column 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.

STUDENT

ROLL_NO	STUDENT_NAME	PHONE	ROOM_NO

<u>ROLL_NO</u>	STUDENT_NAME	PHONE_NO_1	PHONE_NO_2	ROOM_NO

Complaint Table

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

COMPLAINT

<u>COMPLAINT_NO</u>	<u>ROLL_NO</u>	ROOM_NO	DESCRIPTION	COMPLAINT_TYPE

Mess Table

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

MESS

<u>SR_NO</u>	<u>ROLL_NO</u>	FEEDBACK

Laundry Table

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

LAUNDRY

<u>SR_NO</u>	<u>ROLL_NO</u>	GIVEN_ON	RECEIVED_ON	COMPLETED

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.

Student Table

STUDENT

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

Complaint Table

COMPLAINT

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

Mess Table

MESS

SR_NO	ROLL_NO

SR_NO	FEEDBACK

Laundry Table

LAUNDARY

SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

3NF- Third Normal Form

A relation that is in First and Second Normal Form and in which no non-primarykey 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.

Student Table

ROLL_NO	STUDENT_NAME

ROLL_NO	PHONE_NO_1	PHONE_NO_2

ROLL_NO	ROOM_NO

Complaint Table

COMPLAINT_NO	ROLL_NO

COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE

Mess Table

SR_NO	ROLL_NO

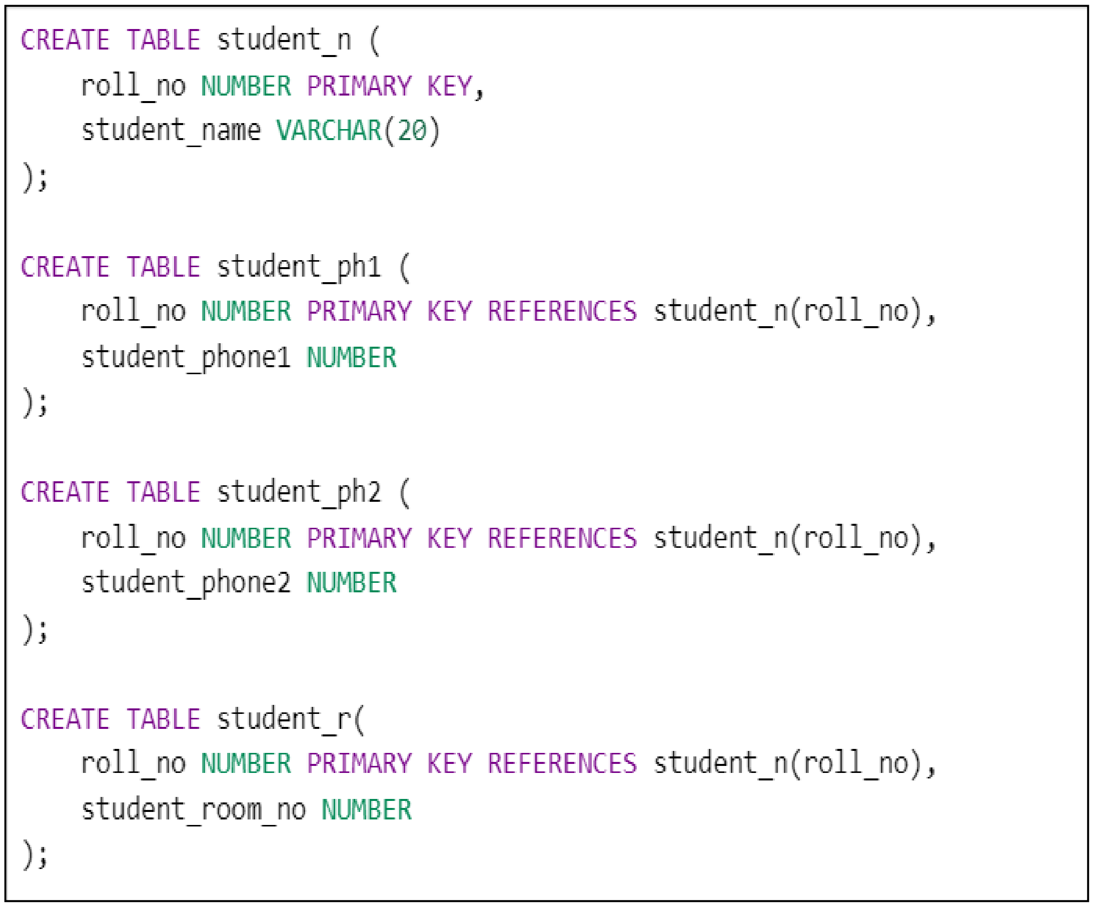
SR_NO	FEEDBACK

Laundry Table

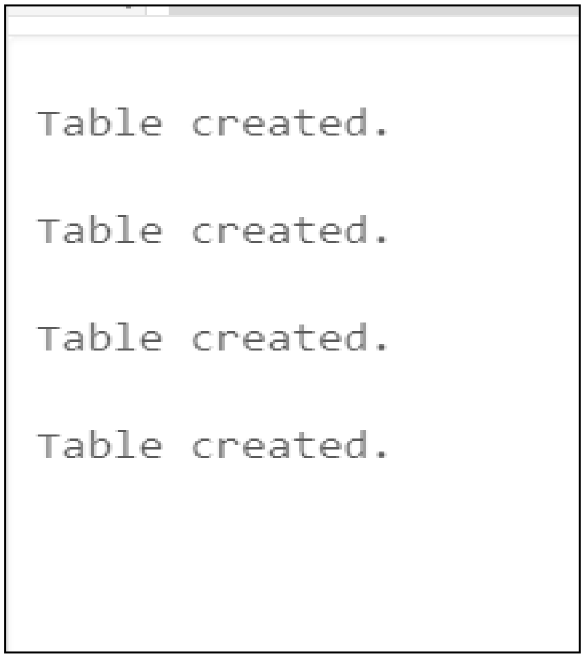
SR_NO	ROLL_NO

SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED

STUDENT TABLE QUERY

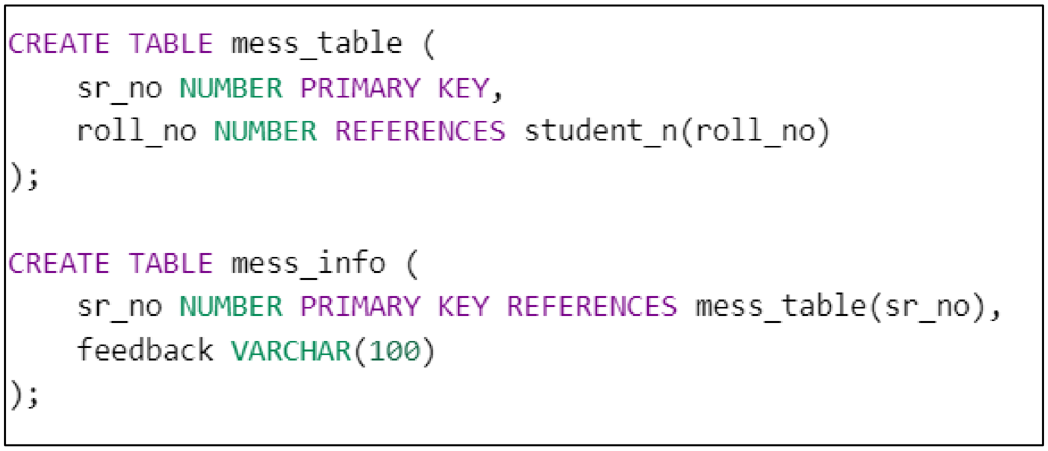


```
CREATE TABLE student_n (
    roll_no NUMBER PRIMARY KEY,
    student_name VARCHAR(20)
);

CREATE TABLE student_ph1 (
    roll_no NUMBER PRIMARY KEY REFERENCES student_n(roll_no),
    student_phone1 NUMBER
);

CREATE TABLE student_ph2 (
    roll_no NUMBER PRIMARY KEY REFERENCES student_n(roll_no),
    student_phone2 NUMBER
);

CREATE TABLE student_r(
    roll_no NUMBER PRIMARY KEY REFERENCES student_n(roll_no),
    student_room_no NUMBER
);
```

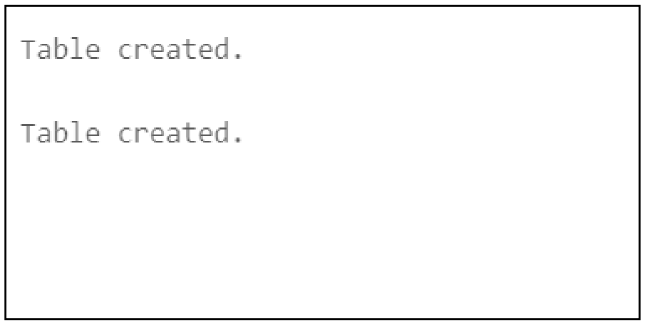


```
Table created.

Table created.

Table created.

Table created.
```

MESS TABLE QUERY

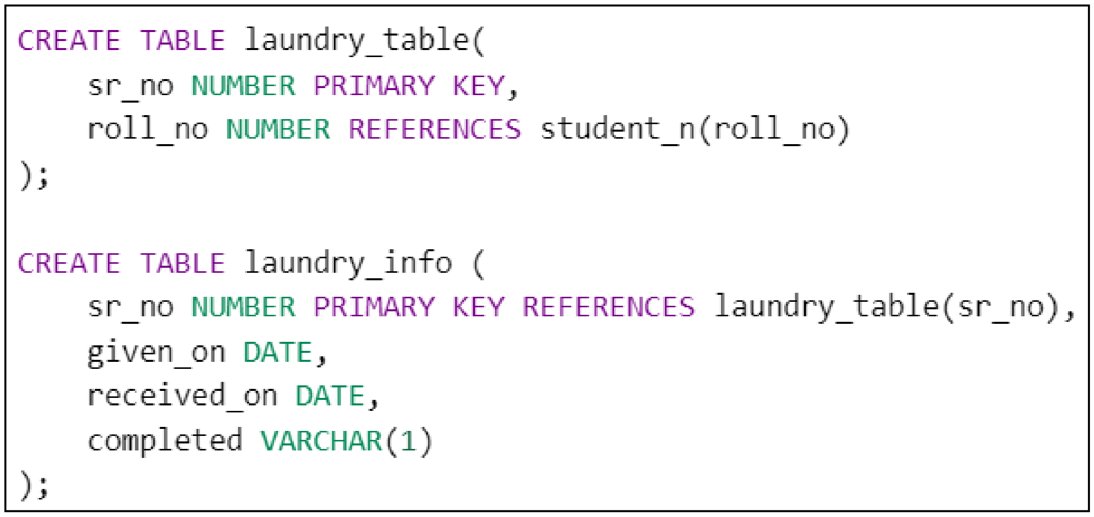


```
CREATE TABLE mess_table (
    sr_no NUMBER PRIMARY KEY,
    roll_no NUMBER REFERENCES student_n(roll_no)
);

CREATE TABLE mess_info (
    sr_no NUMBER PRIMARY KEY REFERENCES mess_table(sr_no),
    feedback VARCHAR(100)
);
```

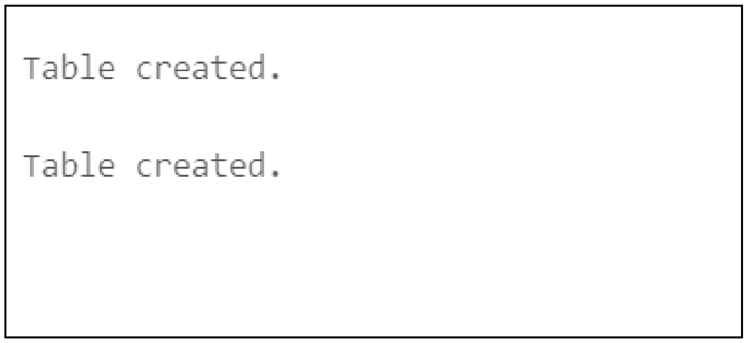


```
Table created.

Table created.
```

LAUNDARY TABLE QUERY

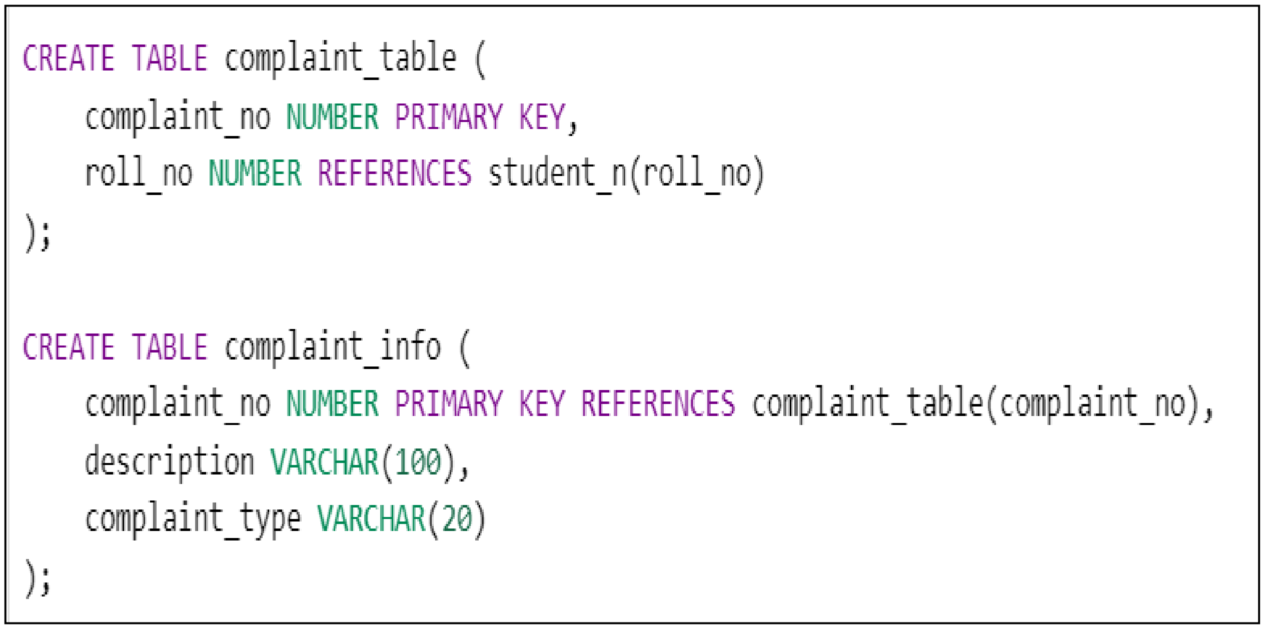


```
CREATE TABLE laundry_table(
    sr_no NUMBER PRIMARY KEY,
    roll_no NUMBER REFERENCES student_n(roll_no)
);

CREATE TABLE laundry_info (
    sr_no NUMBER PRIMARY KEY REFERENCES laundry_table(sr_no),
    given_on DATE,
    received_on DATE,
    completed VARCHAR(1)
);
```

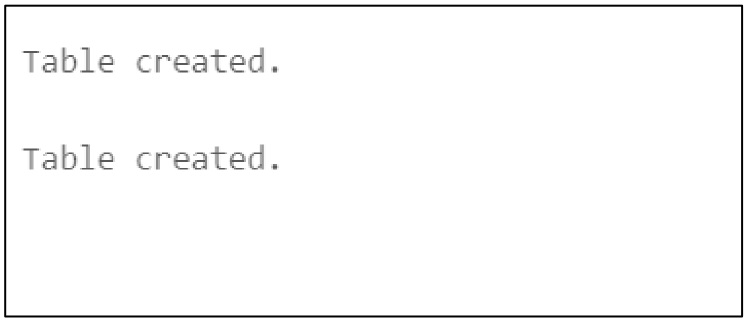


```
Table created.

Table created.
```

COMPLAINT TABLE QUERY



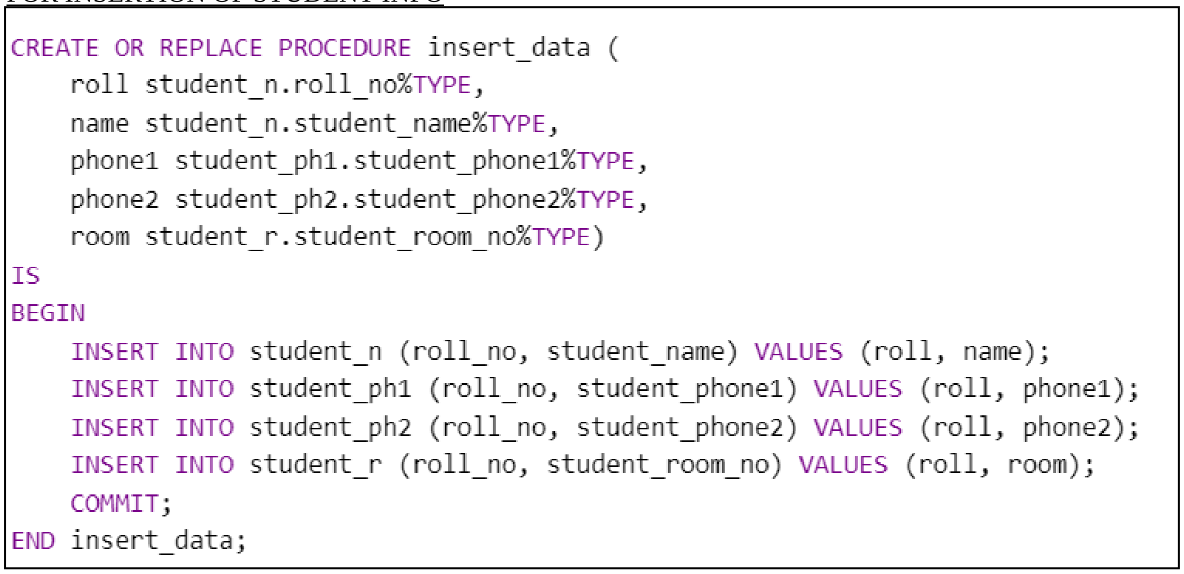
```
CREATE TABLE complaint_table (
    complaint_no NUMBER PRIMARY KEY,
    roll_no NUMBER REFERENCES student_n(roll_no)
);

CREATE TABLE complaint_info (
    complaint_no NUMBER PRIMARY KEY REFERENCES complaint_table(complaint_no),
    description VARCHAR(100),
    complaint_type VARCHAR(20)
);
```



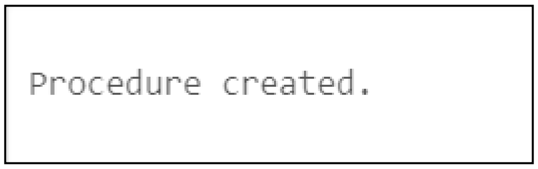
```
Table created.

Table created.
```

PROCEDURES USED

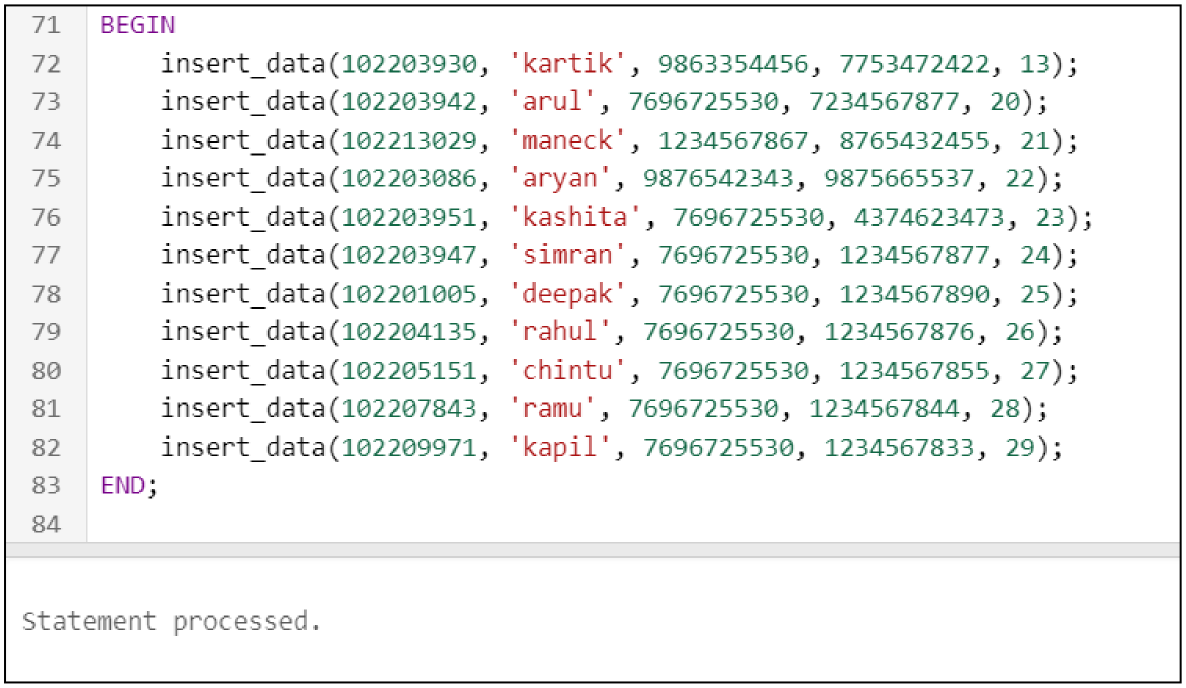
FOR INSERTION OF STUDENT INFO



```
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 insert_data;
```

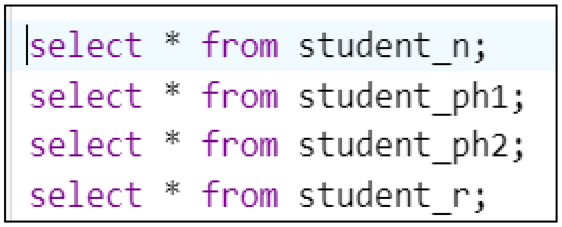


```
Procedure created.
```

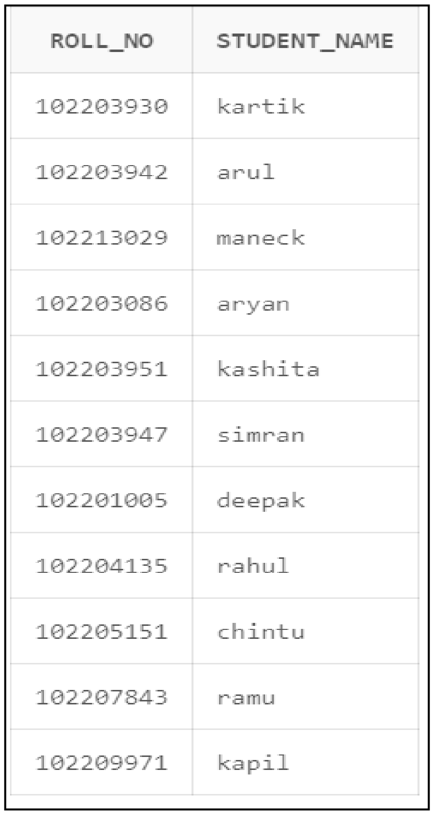


```
BEGIN
    insert_data(102203930, 'kartik', 9863354456, 7753472422, 13);
    insert_data(102203942, 'arul', 7696725530, 7234567877, 20);
    insert_data(102213029, 'maneck', 1234567867, 8765432455, 21);
    insert_data(102203086, 'aryan', 9876542343, 9875665537, 22);
    insert_data(102203951, 'kashita', 7696725530, 4374623473, 23);
    insert_data(102203947, 'simran', 7696725530, 1234567877, 24);
    insert_data(102201005, 'deepak', 7696725530, 1234567890, 25);
    insert_data(102204135, 'rahul', 7696725530, 1234567876, 26);
    insert_data(102205151, 'chintu', 7696725530, 1234567855, 27);
    insert_data(102207843, 'ramu', 7696725530, 1234567844, 28);
    insert_data(102209971, 'kapil', 7696725530, 1234567833, 29);
END;

Statement processed.
```

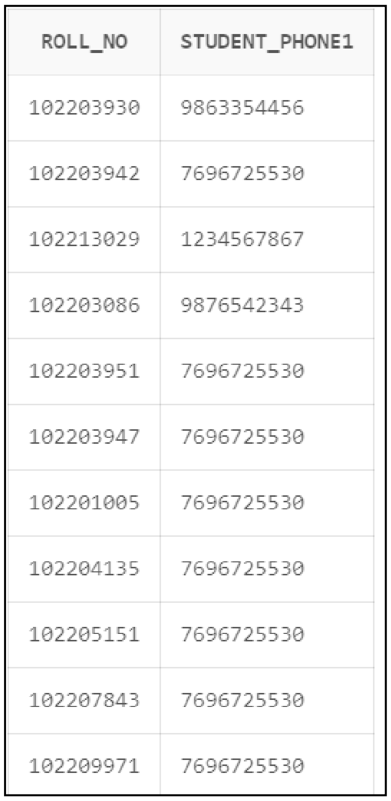
STUDENT TABLE OUTPUT



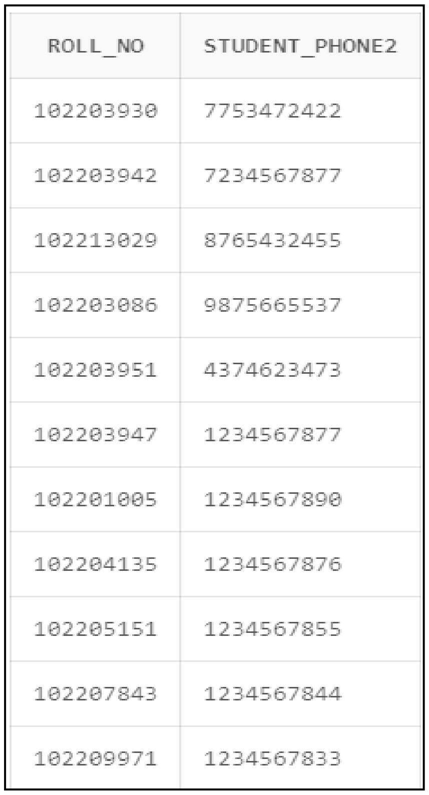
```
select * from student_n;
select * from student_ph1;
select * from student_ph2;
select * from student_r;
```



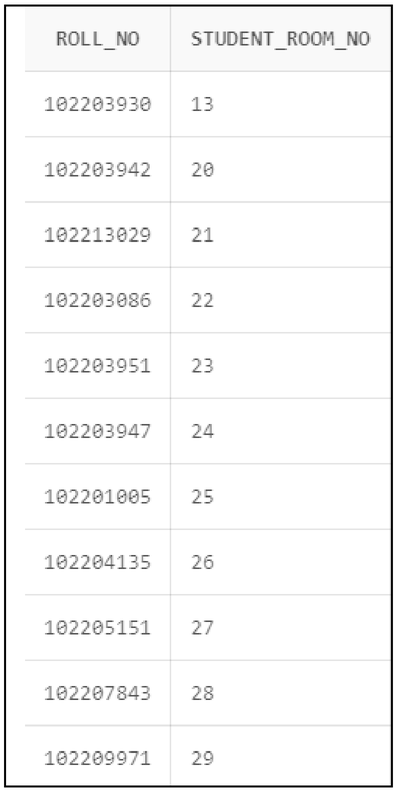
ROLL_NO	STUDENT_NAME
102203930	kartik
102203942	arul
102213029	maneck
102203086	aryan
102203951	kashita
102203947	simran
102201005	deepak
102204135	rahul
102205151	chintu
102207843	ramu
102209971	kapil



ROLL_NO	STUDENT_PHONE1
102203930	9863354456
102203942	7696725530
102213029	1234567867
102203086	9876542343
102203951	7696725530
102203947	7696725530
102201005	7696725530
102204135	7696725530
102205151	7696725530
102207843	7696725530
102209971	7696725530

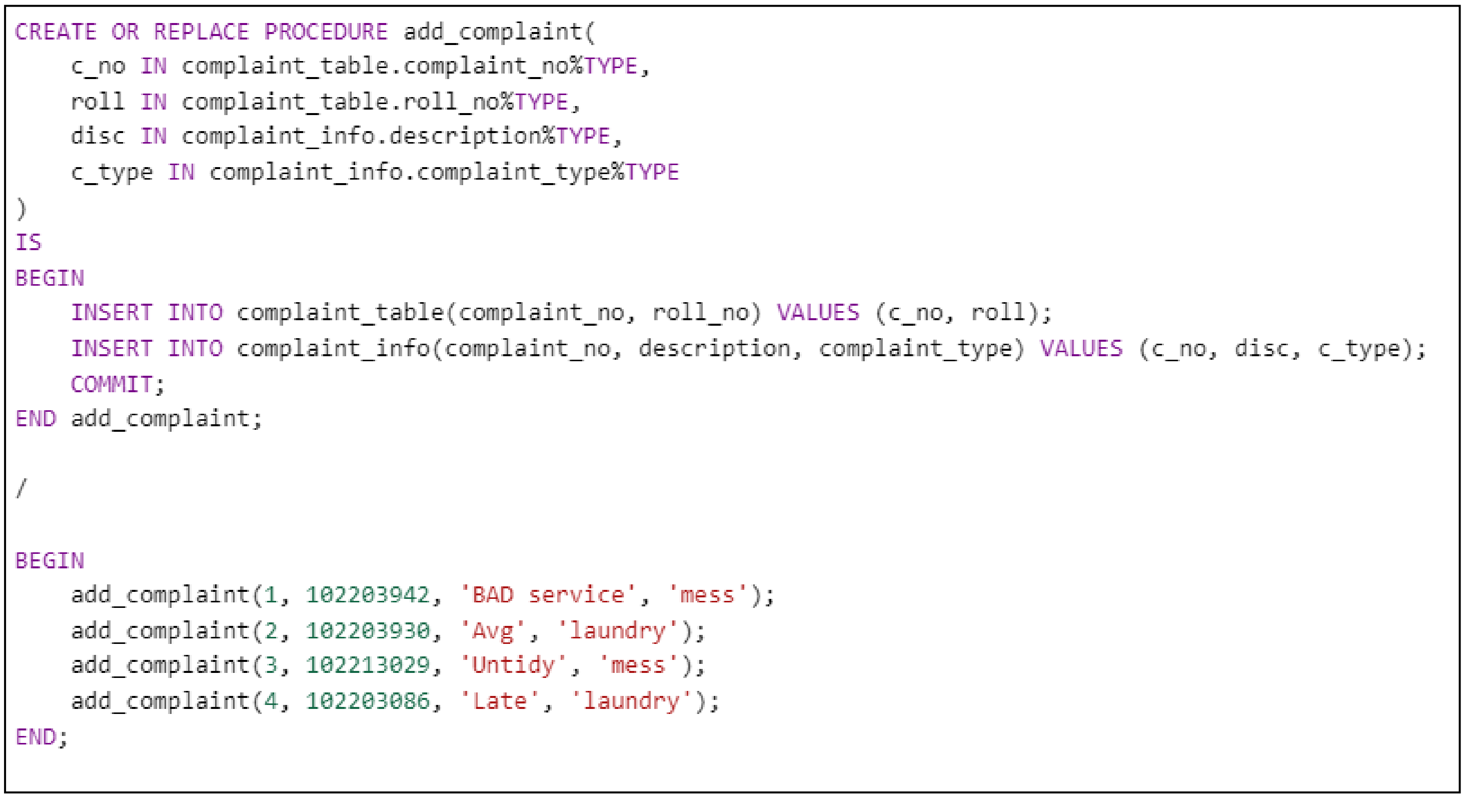


ROLL_NO	STUDENT_PHONE2
102203930	7753472422
102203942	7234567877
102213029	8765432455
102203086	9875665537
102203951	4374623473
102203947	1234567877
102201005	1234567890
102204135	1234567876
102205151	1234567855
102207843	1234567844
102209971	1234567833

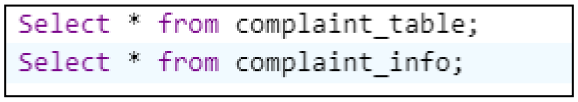


ROLL_NO	STUDENT_ROOM_NO
102203930	13
102203942	20
102213029	21
102203086	22
102203951	23
102203947	24
102201005	25
102204135	26
102205151	27
102207843	28
102209971	29

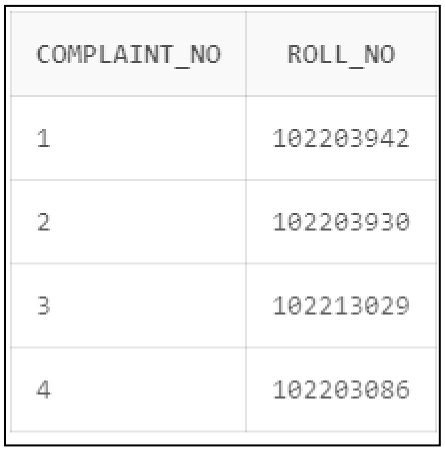
FOR COMPLAINT TABLE



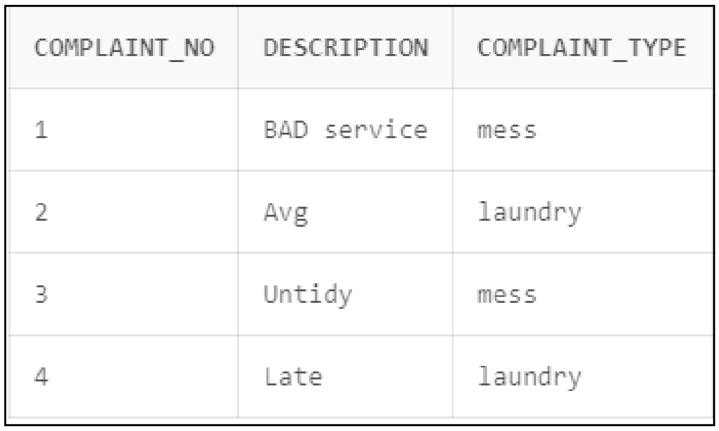
```
CREATE OR REPLACE PROCEDURE add_complaint(
    c_no IN complaint_table.complaint_no%TYPE,
    roll IN complaint_table.roll_no%TYPE,
    disc IN complaint_info.description%TYPE,
    c_type IN complaint_info.complaint_type%TYPE
)
IS
BEGIN
    INSERT INTO complaint_table(complaint_no, roll_no) VALUES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

/

BEGIN
    add_complaint(1, 102203942, 'BAD service', 'mess');
    add_complaint(2, 102203930, 'Avg', 'laundry');
    add_complaint(3, 102213029, 'Untidy', 'mess');
    add_complaint(4, 102203086, 'Late', 'laundry');
END;
```



```
Select * from complaint_table;
Select * from complaint_info;
```

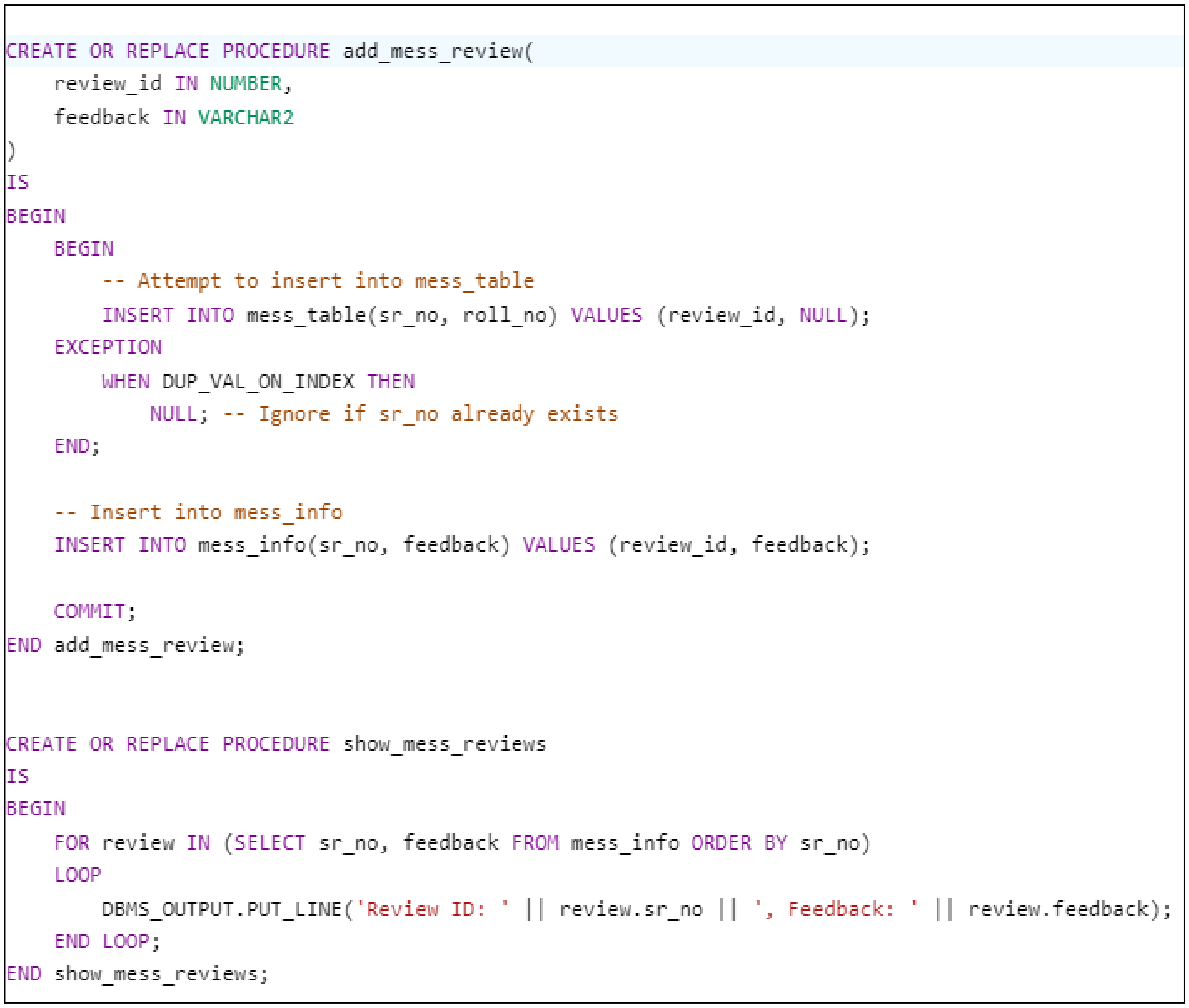


COMPLAINT_NO	ROLL_NO
1	102203942
2	102203930
3	102213029
4	102203086

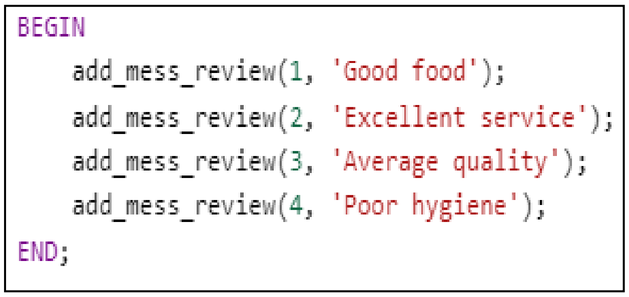


COMPLAINT_NO	DESCRIPTION	COMPLAINT_TYPE
1	BAD service	mess
2	Avg	laundry
3	Untidy	mess
4	Late	laundry

FOR MESS TABLE

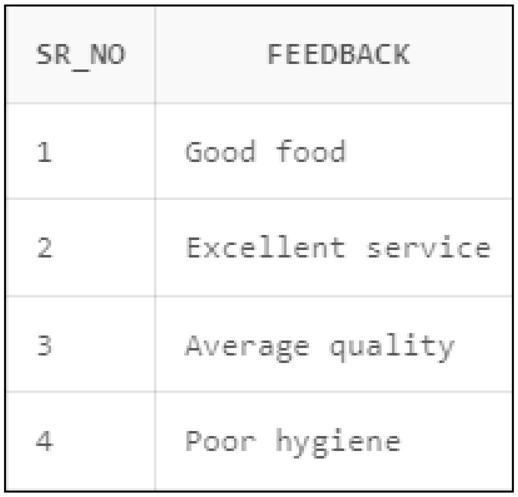


```
CREATE OR REPLACE PROCEDURE add_mess_review(
    review_id IN NUMBER,
    feedback IN VARCHAR2
)
IS
BEGIN
    BEGIN
        -- Attempt to insert into mess_table
        INSERT INTO mess_table(sr_no, roll_no) VALUES (review_id, NULL);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            NULL; -- Ignore if sr_no already exists
    END;

    -- Insert into mess_info
    INSERT INTO mess_info(sr_no, feedback) VALUES (review_id, feedback);

    COMMIT;
END add_mess_review;


CREATE OR REPLACE PROCEDURE show_mess_reviews
IS
BEGIN
    FOR review IN (SELECT sr_no, feedback FROM mess_info ORDER BY sr_no)
    LOOP
        DBMS_OUTPUT.PUT_LINE('Review ID: ' || review.sr_no || ', Feedback: ' || review.feedback);
    END LOOP;
END show_mess_reviews;
```



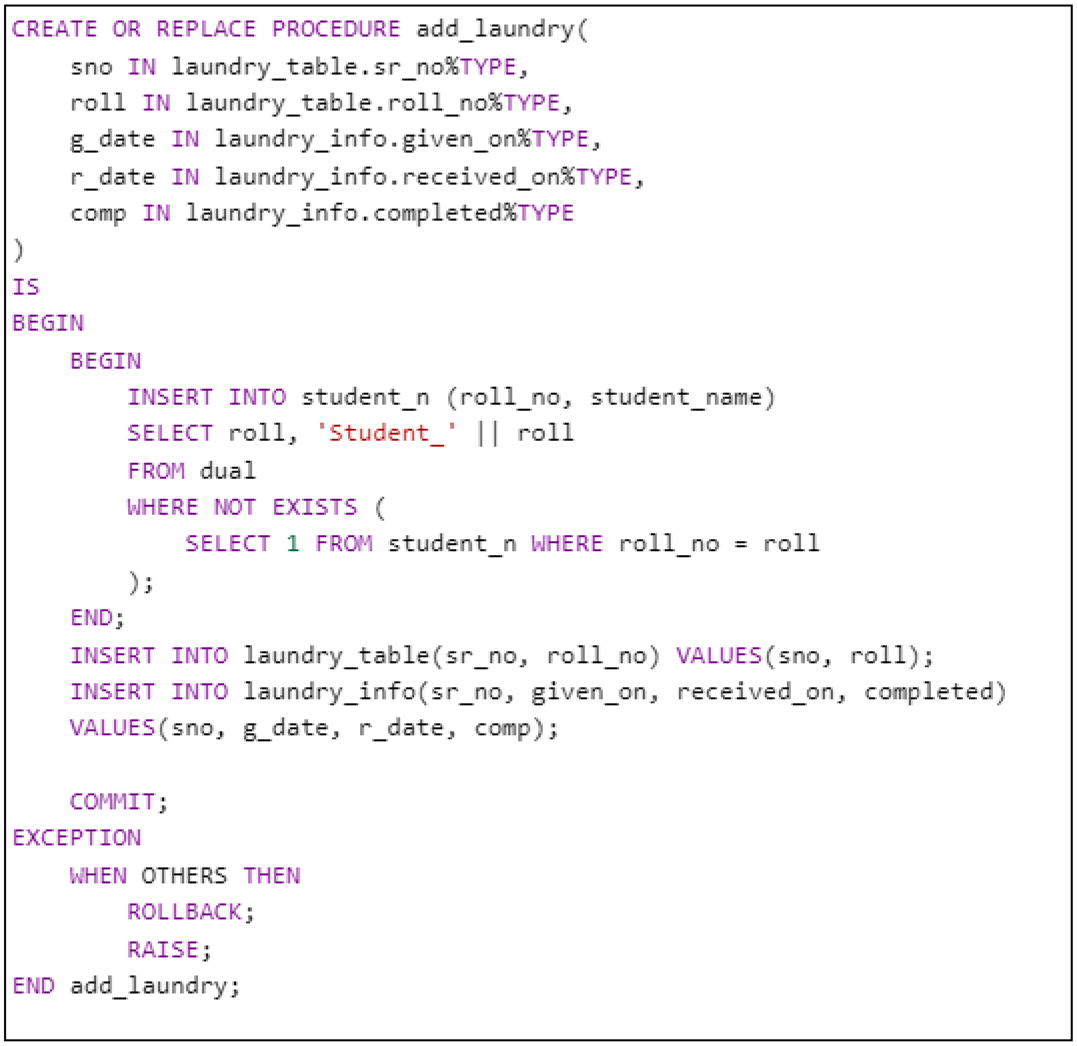
```
BEGIN
    add_mess_review(1, 'Good food');
    add_mess_review(2, 'Excellent service');
    add_mess_review(3, 'Average quality');
    add_mess_review(4, 'Poor hygiene');
END;
```

```sql
Select sr_no,feedback FROM mess_info ORDER BY sr_no;
```

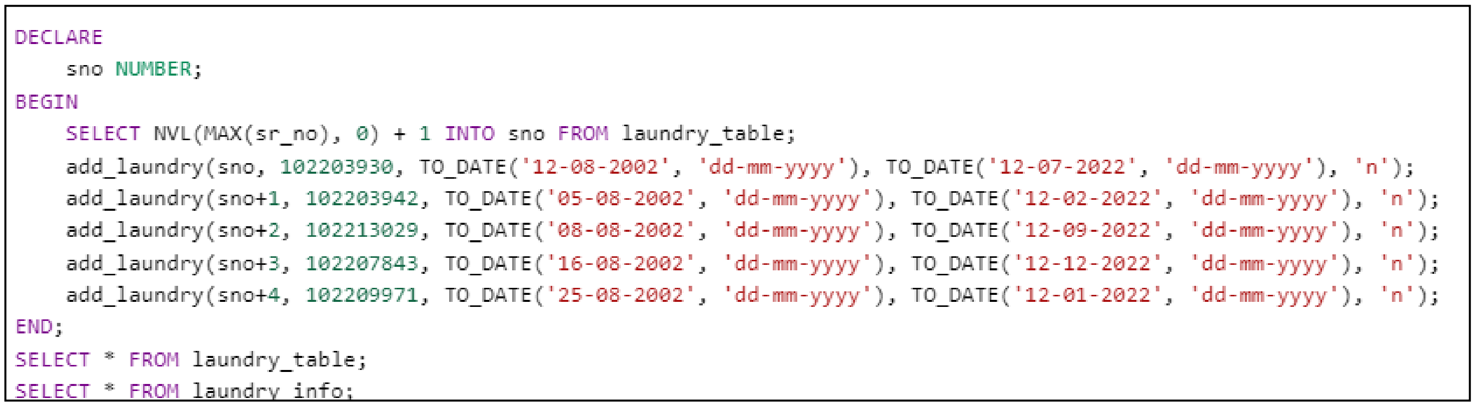


SR_NO	FEEDBACK
1	Good food
2	Excellent service
3	Average quality
4	Poor hygiene

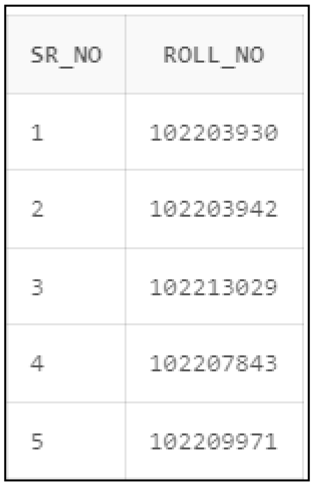
LAUNDRY TABLE



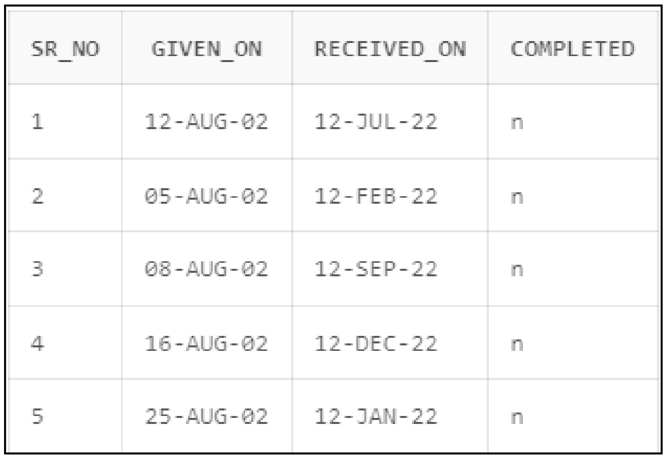
```sql
CREATE OR REPLACE PROCEDURE add_laundry(
    sno IN laundry_table.sr_no%TYPE,
    roll IN laundry_table.roll_no%TYPE,
    g_date IN laundry_info.given_on%TYPE,
    r_date IN laundry_info.received_on%TYPE,
    comp IN laundry_info.completed%TYPE
)
IS
BEGIN
    BEGIN
        INSERT INTO student_n (roll_no, student_name)
        SELECT roll, 'Student_' || roll
        FROM dual
        WHERE NOT EXISTS (
            SELECT 1 FROM student_n WHERE roll_no = roll
        );
    END;
    INSERT INTO laundry_table(sr_no, roll_no) VALUES(sno, roll);
    INSERT INTO laundry_info(sr_no, given_on, received_on, completed)
    VALUES(sno, g_date, r_date, comp);

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END add_laundry;
```



```sql
DECLARE
    sno NUMBER;
BEGIN
    SELECT NVL(MAX(sr_no), 0) + 1 INTO sno FROM laundry_table;
    add_laundry(sno, 102203930, TO_DATE('12-08-2002', 'dd-mm-yyyy'), TO_DATE('12-07-2022', 'dd-mm-yyyy'), 'n');
    add_laundry(sno+1, 102203942, TO_DATE('05-08-2002', 'dd-mm-yyyy'), TO_DATE('12-02-2022', 'dd-mm-yyyy'), 'n');
    add_laundry(sno+2, 102213029, TO_DATE('08-08-2002', 'dd-mm-yyyy'), TO_DATE('12-09-2022', 'dd-mm-yyyy'), 'n');
    add_laundry(sno+3, 102207843, TO_DATE('16-08-2002', 'dd-mm-yyyy'), TO_DATE('12-12-2022', 'dd-mm-yyyy'), 'n');
    add_laundry(sno+4, 102209971, TO_DATE('25-08-2002', 'dd-mm-yyyy'), TO_DATE('12-01-2022', 'dd-mm-yyyy'), 'n');
END;
SELECT * FROM laundry_table;
SELECT * FROM laundry_info;
```

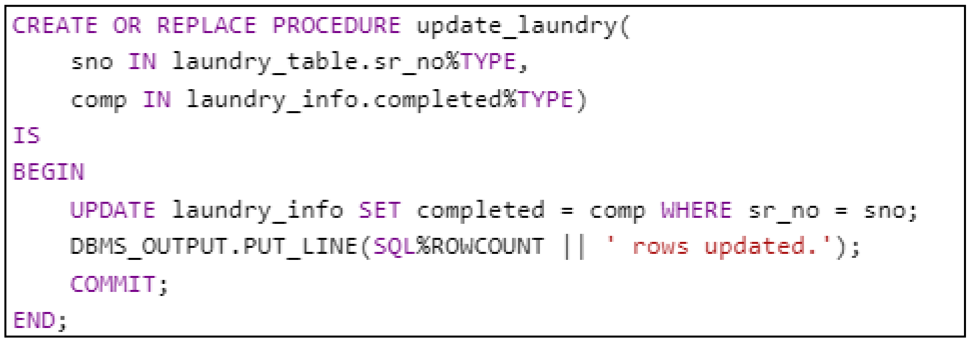


SR_NO	ROLL_NO
1	102203930
2	102203942
3	102213029
4	102207843
5	102209971

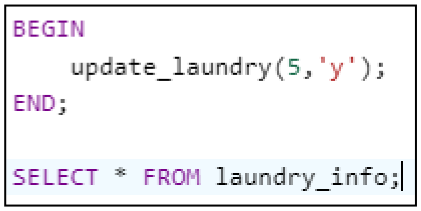


SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED
1	12-AUG-02	12-JUL-22	n
2	05-AUG-02	12-FEB-22	n
3	08-AUG-02	12-SEP-22	n
4	16-AUG-02	12-DEC-22	n
5	25-AUG-02	12-JAN-22	n

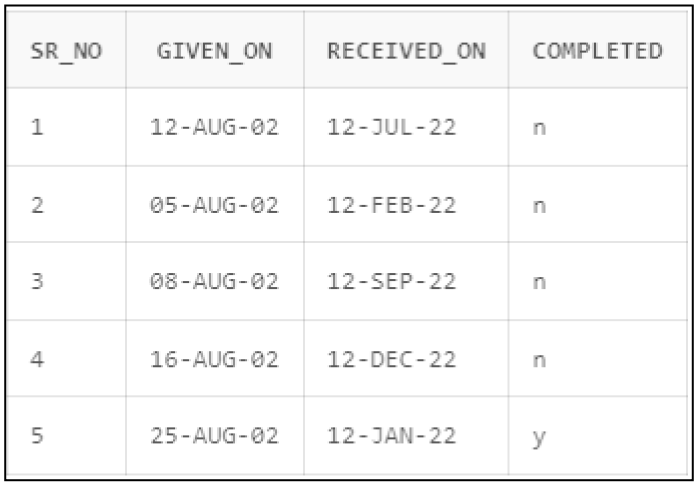
FOR UPDATE



```
CREATE OR REPLACE PROCEDURE update_laundry(
    sno IN laundry_table.sr_no%TYPE,
    comp IN laundry_info.completed%TYPE)
IS
BEGIN
    UPDATE laundry_info SET completed = comp WHERE sr_no = sno;
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
    COMMIT;
END;
```



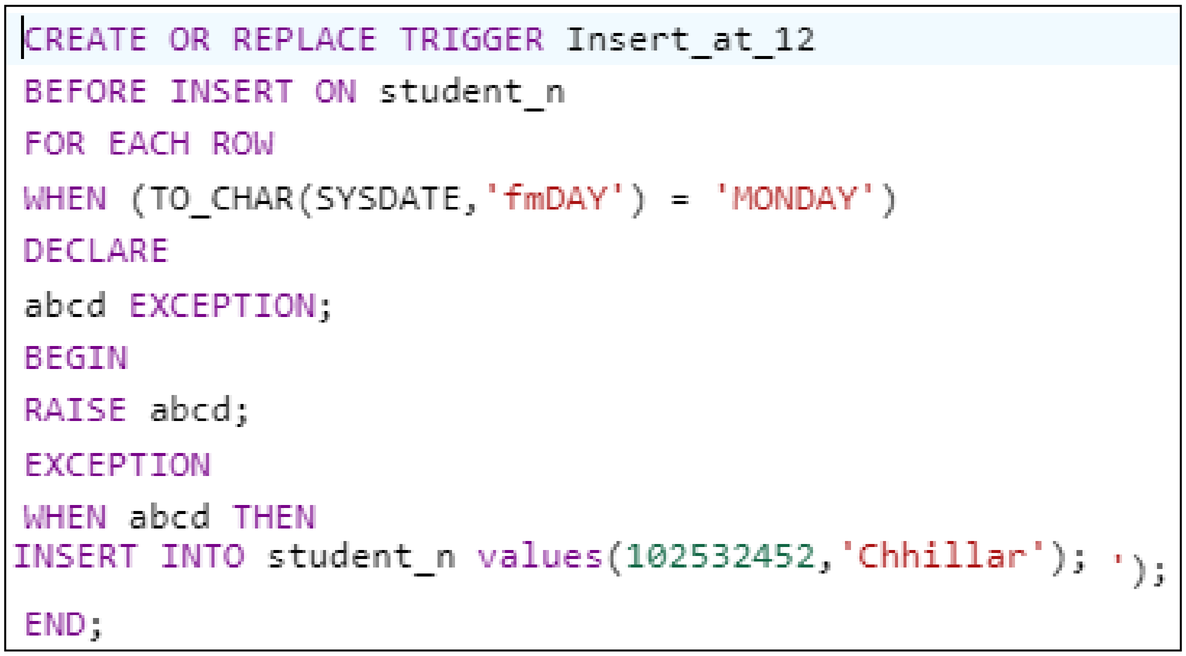
```
BEGIN
    update_laundry(5,'y');
END;

SELECT * FROM laundry_info;
```



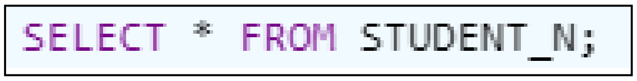
SR_NO	GIVEN_ON	RECEIVED_ON	COMPLETED
1	12-AUG-02	12-JUL-22	n
2	05-AUG-02	12-FEB-22	n
3	08-AUG-02	12-SEP-22	n
4	16-AUG-02	12-DEC-22	n
5	25-AUG-02	12-JAN-22	y

FOR TRIGGERS

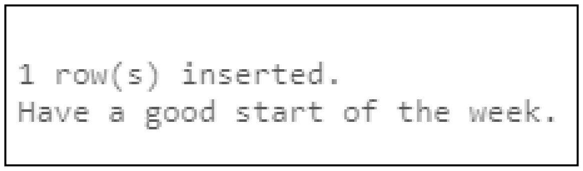
- This trigger executes whenever a new entry is added on the day of Monday. It uses the ‘SYSDATE’ to find out the current day on the machine



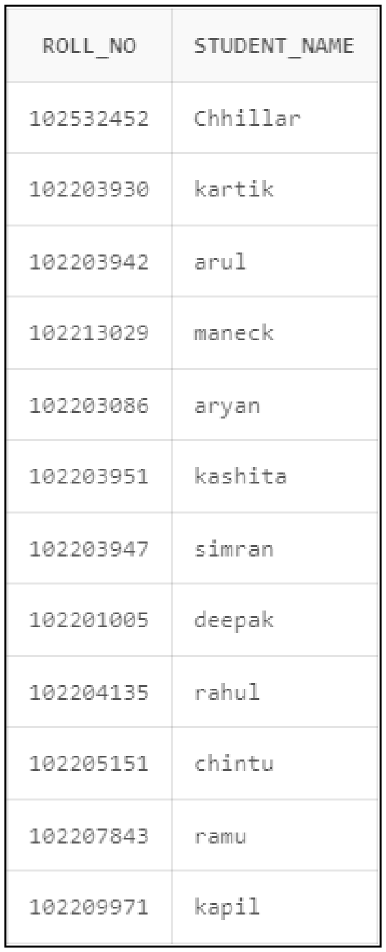
```
CREATE OR REPLACE TRIGGER Insert_at_12
BEFORE INSERT ON student_n
FOR EACH ROW
WHEN (TO_CHAR(SYSDATE,'fmDAY') = 'MONDAY')
DECLARE
abcd EXCEPTION;
BEGIN
RAISE abcd;
EXCEPTION
WHEN abcd THEN
INSERT INTO student_n values(102532452,'Chhillar'); ');

END;
```



```
SELECT * FROM STUDENT_N;
```

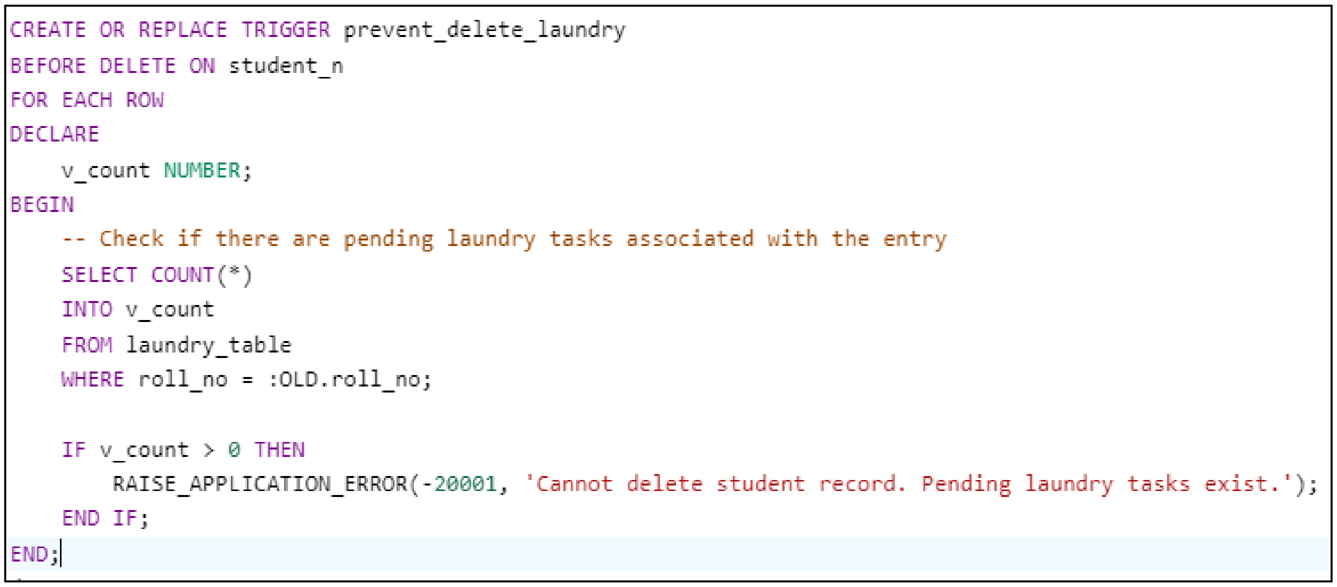


```
1 row(s) inserted.
Have a good start of the week.
```

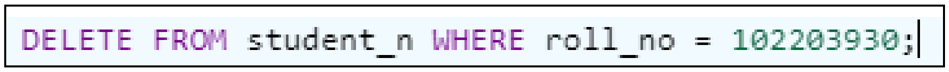


ROLL_NO	STUDENT_NAME
102532452	Chhillar
102203930	kartik
102203942	arul
102213029	maneck
102203086	aryan
102203951	kashita
102203947	simran
102201005	deepak
102204135	rahul
102205151	chintu
102207843	ramu
102209971	kapil

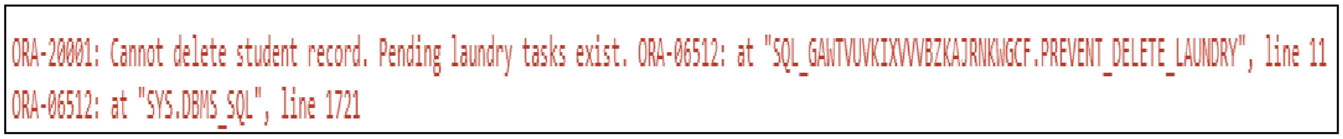
- This trigger executes before a row of student info is deleted. It checks and prevents deletion if there is still laundry left of the respective student and returns the customized error statement.



```
CREATE OR REPLACE TRIGGER prevent_delete_laundry
BEFORE DELETE ON student_n
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    -- Check if there are pending laundry tasks associated with the entry
    SELECT COUNT(*)
    INTO v_count
    FROM laundry_table
    WHERE roll_no = :OLD.roll_no;

    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot delete student record. Pending laundry tasks exist.');
    END IF;
END;
```

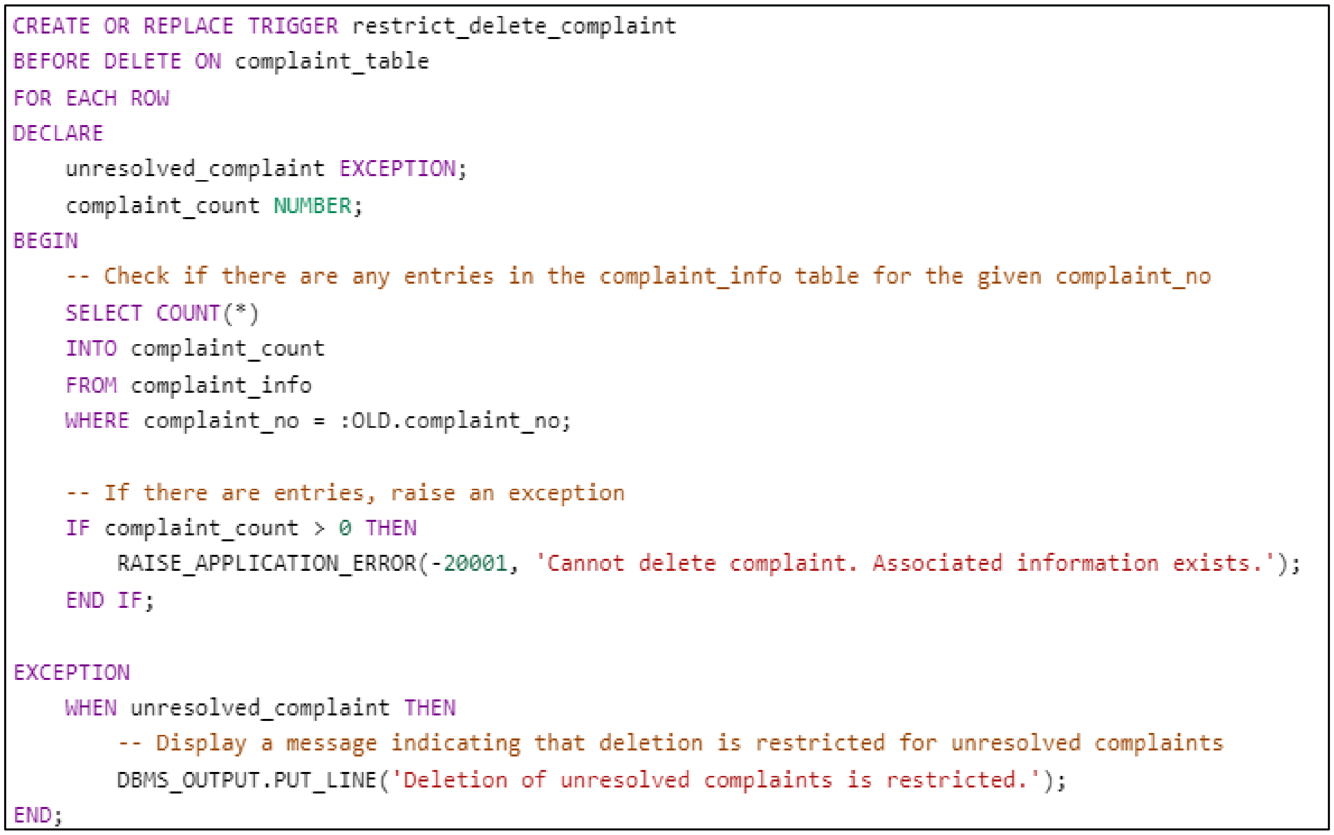


```
DELETE FROM student_n WHERE roll_no = 102203930;
```

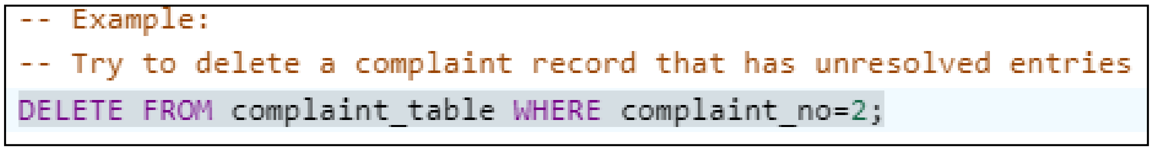


```
ORA-20001: Cannot delete student record. Pending laundry tasks exist. ORA-06512: at "SQL_GAWTVUVKIXVVVBZKAJRNKWGCF.PREVENT_DELETE_LAUNDRY", line 11
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

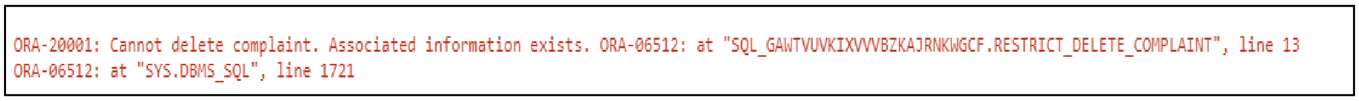
- This trigger is executed before deleting an entry of a student that has unresolved complaint. It will not allow it to be deleted until the complaint is resolved.



```
CREATE OR REPLACE TRIGGER restrict_delete_complaint
BEFORE DELETE ON complaint_table
FOR EACH ROW
DECLARE
    unresolved_complaint EXCEPTION;
    complaint_count NUMBER;
BEGIN
    -- Check if there are any entries in the complaint_info table for the given complaint_no
    SELECT COUNT(*)
    INTO complaint_count
    FROM complaint_info
    WHERE complaint_no = :OLD.complaint_no;

    -- If there are entries, raise an exception
    IF complaint_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot delete complaint. Associated information exists.');
    END IF;

EXCEPTION
    WHEN unresolved_complaint THEN
        -- Display a message indicating that deletion is restricted for unresolved complaints
        DBMS_OUTPUT.PUT_LINE('Deletion of unresolved complaints is restricted.');
END;
```



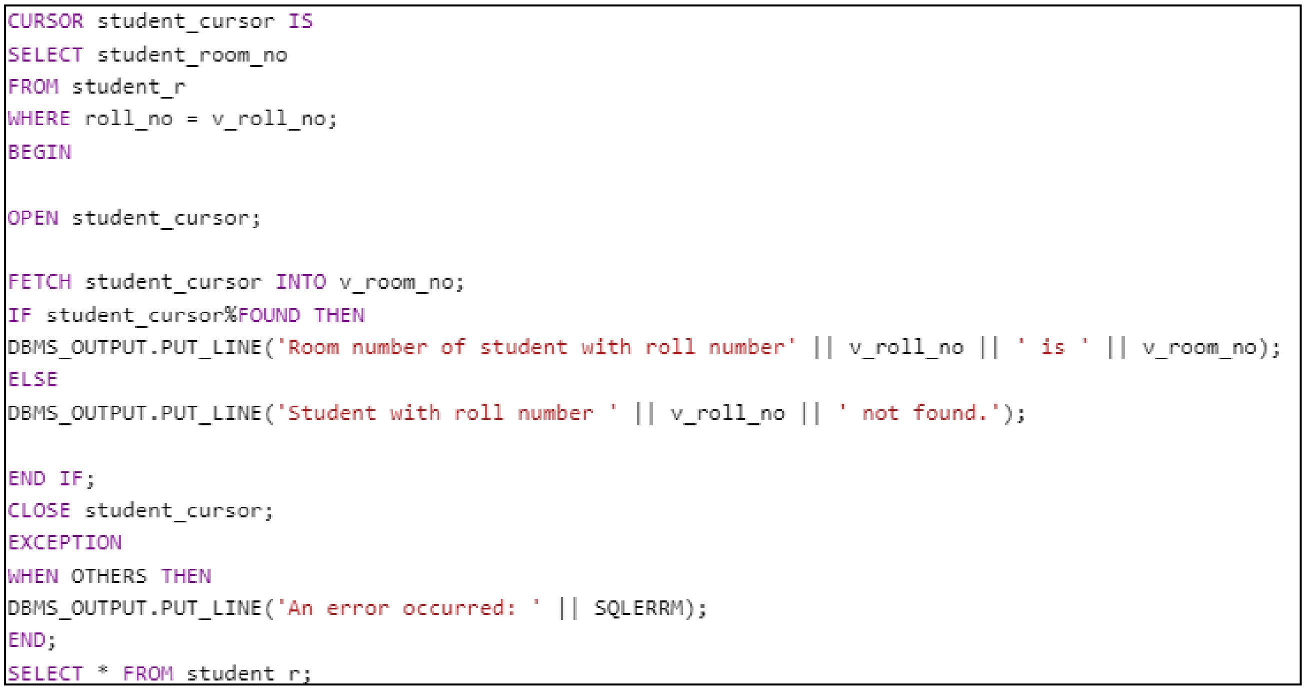
```
-- Example:
-- Try to delete a complaint record that has unresolved entries
DELETE FROM complaint_table WHERE complaint_no=2;
```



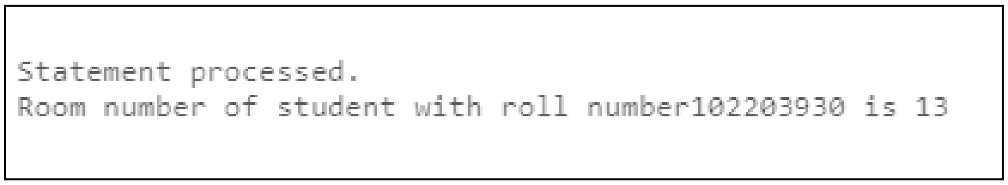
```
ORA-20001: Cannot delete complaint. Associated information exists. ORA-06512: at "SQL_GAWTVUVKIXVVVBZKAJRNKWGCF.RESTRICT_DELETE_COMPLAINT", line 13
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

CURSOR

- This PL/SQL block declares a cursor named **student_cursor** that retrieves the **student_room_no** for a given **roll_no** from the **student_r** table. It then attempts to fetch the result into the variable **v_room_no**. If the cursor finds a record for the given **roll_no**, it prints the room number of the student; otherwise, it prints a message saying the student was not found. Finally, it handles any exceptions that may occur during the execution.
- In summary, this cursor retrieves the room number of a student based on their roll number from the **student_r** table. If the student is found, it prints their room number; otherwise, it indicates that the student was not found. Any errors that occur during the process are also captured and displayed.



```
CURSOR student_cursor IS
SELECT student_room_no
FROM student_r
WHERE roll_no = v_roll_no;
BEGIN

OPEN student_cursor;

FETCH student_cursor INTO v_room_no;
IF student_cursor%FOUND THEN
DBMS_OUTPUT.PUT_LINE('Room number of student with roll number' || v_roll_no || ' is ' || v_room_no);
ELSE
DBMS_OUTPUT.PUT_LINE('Student with roll number ' || v_roll_no || ' not found.');

END IF;
CLOSE student_cursor;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
SELECT * FROM student_r;
```



```
Statement processed.
Room number of student with roll number102203930 is 13
```

CONCLUSION

In conclusion, our hostel management system project has successfully addressed the complexities of managing hostel operations through efficient use of PL/SQL. Our primary objective was to develop a comprehensive system that streamlines student management, laundry services, complaint handling, and mess management. Through diligent planning, collaboration, and implementation, we have achieved significant milestones and delivered a robust solution tailored to the needs of hostel administrators, staff, and students alike.

The system's key features, including student registration, laundry service tracking, complaint resolution, and mess management, empower administrators to efficiently oversee hostel operations while enhancing the overall experience for residents. By leveraging PL/SQL, we ensured data integrity, optimized performance, and facilitated seamless integration across various modules of the system.

Throughout the project lifecycle, we encountered challenges that tested our problem-solving skills and adaptability. However, through teamwork and perseverance, we successfully navigated these hurdles and delivered a solution that meets our stakeholders' expectations.

Looking ahead, there are opportunities for further enhancement and expansion of the hostel management system. Future iterations could explore integrating additional modules for facilities management, event planning, or visitor tracking, thereby providing a more comprehensive solution for hostel administrators.

We extend our sincere gratitude to all team members, stakeholders, and mentors who contributed to the success of this project. Their dedication, expertise, and collaboration were instrumental in bringing our vision to fruition.

........THANK YOU.......