HOSTEL MANAGEMENT SYSTEM

DBMS PROJECT

HOSTEL MANAGEMENT SYSTEM

submited by

JAIMIN CHAUDHARI	486106	8022054347	SUJAL PATEL	486152	8022054066
DHAVAL RATHOD	486161	8022054255	DARSH PATEL	486143	8022053574

INDEX

❖ VISION

➢ Our vision is to develop a user-friendly Hostel Management System that automates administrative tasks, enhances communication, and improves overall efficiency for hostel staff and residents.

❖ Key Features

Room Allocator

- Automatically assigns rooms based on student preferences and availability.
- the process of room allocation and vacancy management.

Entry-Exit Time Noter

Records entry and exit times of hostel residents for security monitoring.

Feedback Date Noter

- Manages deadlines for submitting feedback from students.

Date Noter

- Tracks complaint submission deadlines and resolution timelines.

❖ User Packages and Roles

➢ Student Package
- Functions
- Secure student login to access personalized features.
- View and update student details, room assignments, and leave requests.
- Submit feedback and lodge complaints.

➢ Procedures:
- Streamlined new student registration process.
- Option to cancel admission, apply for leave, and submit feedback.

- Triggers:
    - room allocation triggered upon registration.

- Employee Package
    - Functions
    - Employee login for administrative tasks and complaint management.
    - Access employee details, job information, and assigned complaints.

❖ Roles:

- Producer Role: Responsible for handling complaints and feedback.
- Trigger Role: Manages room allocation triggers based on student data.

❖ Benefits

- Efficiency : Simplifies hostel management tasks and reduces manual efforts.
- Transparency : Enhances communication and visibility of key hostel operations.
- User-Friendly Interface : Provides an intuitive platform for students and staff.

❖ Tables

- **Student**
  - ✓ Stud_Info(rollno, sname, sphoneno, saddress, sdob, course, institute)
  - ✓ Stud_Email(rollno, semail) {semail is the foreign key to Stud_Password}
  - ✓ Stud_Password(semail, spass)
  - ✓ Stud_Rooms(rollno rm_no) {rm_no is the foreign key to Rooms}

- **Employee**
  - ✓ Emp_Info(empno, ename, ephoneno, eaddress, gender, marital_st, edob)
  - ✓ Emp_Email(empno, e_email)
  - ✓ Emp_Password(e_email, epass)
  - ✓ Emp_Job_Info(empno, ejob) {ejob is the foreign key to Job}
  - ✓ Job(ejob, sal)
  - ✓ Emp_Comp_Junc(rollno, com_dt, empno) {(rollno, com_dt) is the foreign key to Complain}, {empno is the foreign key to Emp_Info}

- **Vehicle**
  - ✓ Vehicle(vl_id, rollno, reg_no, vl_type) {rollno is the foreign key to Stud_Info}

- **Leave**
  - ✓ Leave(rollno, leave dt, address, reason, no_of_day) {rollno is the foreign key to Stud_Info}

- **Mess-Menu**
  - ✓ Mess(mess id, monday, tuesday, wednesday, thursday, friday, saturday, sunday)
  - ✓ Mess_Fb_Junc(rollno, fb dt, mess id) {(rollno, fb_dt) is the foreign key to Feedback}

- **Feedback**

  Feedback(rollno, fb dt, day, feedback) {rollno is the foreign key to Stud_Info}

- **Rooms**
  - ✓ Rooms(rm_no, capacity, occupancy)

- **Complain**
  - ✓ Complain(rollno, comdt, com_type, is_done) {rollno is the foreign key to Stud_Info}

- Entry-Exit
  - ✓ Entry_Exit(rollno, ee_time, ee_date, place,ee_type) {rollno is the foreign key to Stud_Info}

❖ Cardinality

- Vehicle – Student : n-1
- Entry_Exit – Student : n-1 (weak)
- Rooms – Student : 1-n
- Complain – Student : n-1
- Complain – Employee : n -m
- Student – Feedback : 1-n (weak)
- Feedback – Mess : n-m
- Leave – Student : n-1 (Weak)

❖ FUNCTION

```
create or replace function food_on_day(mess_id_in IN NUMBER, day_in IN VARCHAR2)
 return  VARCHAR2
IS
 food_menu VARCHAR2(100);

begin
 -- Retrieve food from mess_id and day
 select
  case upper(day_in)
   WHEN 'MONDAY' THEN monday
   WHEN 'TUESDAY' THEN tuesday
   WHEN 'WEDNESDAY' THEN wednesday
   WHEN 'THURSDAY' THEN thursday
   WHEN 'FRIDAY' THEN friday
   WHEN 'SATURDAY' THEN saturday
   WHEN 'SUNDAY' THEN sunday

  END
 INTO food_menu
 FROM Mess
 WHERE mess_id = mess_id_in;

 -- Check if a valid menu was found
 IF food_menu IS NOT NULL THEN
  RETURN food_menu;
 ELSE
  RETURN 'No menu available for this day or mess';
 END IF;
EXCEPTION
```

```
    -- if no data found
  WHEN NO_DATA_FOUND THEN
    RETURN 'Mess or day not found';

END;
/
```

-
  ```
  CREATE OR REPLACE FUNCTION record_entry_exit(
    p_rollno IN NUMBER,
    p_place IN VARCHAR2,
    p_ee_type IN VARCHAR2
  ) RETURN VARCHAR2
  IS
  BEGIN

    -- Insert the entry or exit record into Entry_Exit table
    INSERT INTO Entry_Exit (rollno, place, ee_type)
    VALUES (p_rollno, p_place, p_ee_type);

    -- Commit the transaction
    COMMIT;

    -- Return success message
    RETURN 'Entry/Exit record recorded successfully';
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'Error: Roll number not found in Stud_Info';

  END;
  /
  ```

❖ PRODUCER

```
CREATE OR REPLACE PROCEDURE AnalyzeRating IS
BEGIN
  FOR rec IN (SELECT mess_id, day,
                COUNT(*) AS total_feedback,
                AVG(rating) AS avg_rating,
                MAX(rating) AS max_rating,
                MIN(rating) AS min_rating
         FROM Feedback
         GROUP BY mess_id, day)
  LOOP
    -- Display results for each mess ID and day combination
    DBMS_OUTPUT.PUT_LINE('Mess ID: ' || rec.mess_id || ', Day: ' || rec.day);
    DBMS_OUTPUT.PUT_LINE('Total Feedback: ' || rec.total_feedback);
    DBMS_OUTPUT.PUT_LINE('Average Rating: ' || rec.avg_rating);
    DBMS_OUTPUT.PUT_LINE('Maximum Rating: ' || rec.max_rating);
    DBMS_OUTPUT.PUT_LINE('Minimum Rating: ' || rec.min_rating);
    DBMS_OUTPUT.PUT_LINE('------------------------');
  END LOOP;
END;/

CREATE OR REPLACE PROCEDURE Check_Student_Vehicle (
  student_rollno IN Stud_Info.rollno%TYPE
)
IS
  v_vehicle_details Vehicle%ROWTYPE;
BEGIN
```

```
  SELECT *
  INTO v_vehicle_details
  FROM Vehicle
  WHERE rollno = student_rollno;

  -- If a vehicle is found for the student, print the details
  IF v_vehicle_details.vl_id IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('Student has a vehicle with the following details:');
    DBMS_OUTPUT.PUT_LINE('Vehicle ID: ' || v_vehicle_details.vl_id);
    DBMS_OUTPUT.PUT_LINE('Roll Number: ' || v_vehicle_details.rollno);
    DBMS_OUTPUT.PUT_LINE('Registration Number: ' || v_vehicle_details.reg_no);
    DBMS_OUTPUT.PUT_LINE('Vehicle Type: ' || v_vehicle_details.vl_type);
  ELSE
    -- If no vehicle is found for the student, print a message
    DBMS_OUTPUT.PUT_LINE('Student does not have a vehicle.');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Student does not have a vehicle.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;/
```

❖ PACKAGE

```
-- Specification
CREATE OR REPLACE PACKAGE student_pkg AS
  FUNCTION student_login (
    username IN Stud_Password.semail%TYPE,
    password IN Stud_Password.spass%TYPE
  ) RETURN BOOLEAN;

  FUNCTION get_student_details (
    v_rollno IN Stud_Info.rollno%TYPE
  ) RETURN CURSOR;

  FUNCTION get_student_leave_details (
    v_rollno IN Stud_Info.rollno%TYPE
  ) RETURN CURSOR;

  FUNCTION get_room_details (
    v_rollno IN Stud_Info.rollno%TYPE
  ) RETURN CURSOR;

  FUNCTION get_feedback_details (
    v_rollno IN Stud_Info.rollno%TYPE
  ) RETURN CURSOR;

  FUNCTION get_complain_details (
    v_rollno IN Stud_Info.rollno%TYPE
  ) RETURN CURSOR;
```

```
-- Procedures
PROCEDURE new_student_registration(
  p_name      IN Stud_Info.sname%TYPE,
  p_email     IN Stud_Password.semail%TYPE,
  p_password  IN Stud_Password.spass%TYPE,
  p_dob       IN Stud_Info.sdob%TYPE,
  p_address   IN Stud_Info.saddress%TYPE,
  p_phone     IN Stud_Info.sphoneno%TYPE,
  p_course    IN Stud_Info.course%TYPE,
  p_insti     IN Stud_Info.institute%TYPE
);

PROCEDURE cancel_admission (
  p_roll_number IN Stud_Info.rollno%TYPE
);

PROCEDURE lodge_complain (
  p_rollno         IN Stud_Info.rollno%TYPE,
  p_complaint_type IN Complain.com_type%TYPE
);

PROCEDURE submit_feedback (
  p_rollno   IN Stud_Info.rollno%TYPE,
  p_feedback IN Mess_Fb_Junc.feedback%TYPE,
  p_rating   IN Mess_Fb_Junc.rating%TYPE
);

PROCEDURE apply_for_leave (
  p_rollno    IN Stud_Info.rollno%TYPE,
  p_leave_dt   IN Leave.leave_dt%TYPE,
  p_address    IN Leave.address%TYPE,
    p_reason      IN Leave.reason%TYPE,
    p_no_of_day   IN Leave.no_of_day%TYPE
  );
END student_pkg;
/

-- Body
CREATE OR REPLACE PACKAGE BODY student_pkg AS
  -- Functions
  FUNCTION student_login  (
    username IN Stud_Password.semail%TYPE,
    password IN Stud_Password.spass%TYPE
  ) RETURN BOOLEAN IS
  DECLARE
    count NUMBER;
  BEGIN
    -- Check if the provided username and password match any student record
    SELECT COUNT(*) INTO count
    FROM Student_Info
    WHERE semail = username
    AND spass = password;

    -- Return TRUE if a matching record is found, FALSE otherwise
    RETURN count > 0;

  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
      RETURN FALSE;

  END student_login;
```

```
FUNCTION get_student_details (
  v_rollno IN Stud_Info.rollno%TYPE
) RETURN CURSOR IS
DECLARE
  v_cur CURSOR;
BEGIN
  -- Open the cursor for the specified roll number
  OPEN v_cur FOR
    SELECT si.*, se.semail, sr.rm_no
    FROM Stud_Info si
    LEFT JOIN Stud_Email se ON si.rollno = se.rollno
    LEFT JOIN Stud_Rooms sr ON si.rollno = sr.rollno
    WHERE si.rollno = v_rollno;

  RETURN v_cur;
EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
     RETURN NULL;
END get_student_details;

FUNCTION get_student_leave_details (
  v_rollno IN Stud_Info.rollno%TYPE
) RETURN CURSOR IS
DECLARE
  cur CURSOR;
BEGIN

  OPEN cur FOR
  SELECT * FROM Leave
  WHERE rollno = v_rollno;

  RETURN cur;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
    RETURN NULL;
END get_student_leave_details;

FUNCTION get_room_details (
  v_rollno IN Stud_Info.rollno%TYPE
) RETURN CURSOR IS
DECLARE
  cur CURSOR;
BEGIN
  SELECT rm_no INTO v_rm_no
  FROM Stud_Rooms
  WHERE rollno = v_rollno;

  OPEN cur FOR
  SELECT * FROM Rooms
  WHERE rm_no = v_rm_no;
  RETURN cur;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
    RETURN NULL;
END get_room_details;
```

```
FUNCTION get_feedback_details (
  v_rollno IN Stud_Info.rollno%TYPE
) RETURN CURSOR IS
DECLARE
  cur CURSOR;
BEGIN
  OPEN cur FOR
  SELECT * FROM Feedback
  WHERE rollno = v_rollno;

  RETURN cur;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
    RETURN NULL;
END get_feedback_details;

FUNCTION get_complain_details (
  v_rollno IN Stud_Info.rollno%TYPE
) RETURN CURSOR IS
DECLARE
  cur CURSOR;
BEGIN
  OPEN cur FOR
  SELECT * FROM Complain
  WHERE rollno = v_rollno;

  RETURN cur;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred : ' || SQLERRM);
  RETURN NULL;
END get_complain_details;


-- Procedures


PROCEDURE new_student_registration (
  p_name      IN Stud_Info.sname%TYPE,
  p_email     IN Stud_Password.semail%TYPE,
  p_password  IN Stud_Password.spass%TYPE,
  p_dob       IN Stud_Info.sdob%TYPE,
  p_address   IN Stud_Info.saddress%TYPE,
  p_phone     IN Stud_Info.sphoneno%TYPE,
  p_course    IN Stud_Info.course%TYPE,
  p_insti     IN Stud_Info.institute%TYPE
) IS
DECLARE
  p_roll Stud_Info.rollno%TYPE;
BEGIN
  INSERT INTO Student_Info (sname, sphoneno, saddress, sdob, course, institute)
  VALUES (p_name, p_phone, p_address, p_dob, p_course, p_insti);


  SELECT rollno into p_roll FROM Stud_Info
  WHERE sphoneno = p_phone;


  INSERT INTO Stud_Email(rollno, semail)
  VALUES (p_roll, p_email);


  INSERT INTO Stud_Password(semail, spass)
  VALUES (p_email, p_password);


  COMMIT; -- Commit the transaction
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
    ROLLBACK; -- Rollback the transaction to maintain data consistency
END new_student_registration;

PROCEDURE cancel_admission (
  p_roll_number IN Stud_Info.rollno%TYPE
) IS
BEGIN
  -- Attempt to delete the student from the student table
  DELETE FROM Stud_Info WHERE rollno = p_roll_number;

  -- Commit the transaction
  COMMIT;

  EXCEPTION
  -- Handle exceptions
  WHEN NO_DATA_FOUND THEN
    -- Handle the case where no student with the provided roll number is found
    DBMS_OUTPUT.PUT_LINE('No student found with roll number ' || p_roll_number);
  WHEN OTHERS THEN
     -- Handle any other exceptions
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
    ROLLBACK; -- Rollback the transaction to maintain data consistency
END cancel_admission;

PROCEDURE lodge_complain (
  p_rollno         IN Stud_Info.rollno%TYPE,
  p_complaint_type IN Complain.com_type%TYPE
) IS
  BEGIN
  -- Step 1: Input Validation (if necessary)
  -- This could include checking if the provided roll number exists in the Stud_Info table

  -- Step 2: Insert into Database
    INSERT INTO Complain (rollno, com_type, is_done)
    VALUES (p_rollno, p_complaint_type, NULL);

  -- Step 3: Logging (Optional)
  -- Add logging logic here if needed

  COMMIT; -- Commit the transaction

  EXCEPTION
  WHEN OTHERS THEN
      -- Handle any exceptions that might occur
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
      ROLLBACK; -- Rollback the transaction to maintain data consistency
END lodge_complain;

PROCEDURE submit_feedback (
  p_rollno   IN Stud_Info.rollno%TYPE,
  p_feedback IN Mess_Fb_Junc.feedback%TYPE,
  p_rating   IN Mess_Fb_Junc.rating%TYPE
) IS
  BEGIN
    -- Step 1: Input Validation (if necessary)
    -- This could include checking if the provided roll number exists in the Stud_Info table

    -- Step 2: Insert into Database
    INSERT INTO HR.Feedback (rollno, feedback, rating)
    VALUES (p_rollno, p_feedback, p_rating);


  -- Step 3: Logging (Optional)
  -- Add logging logic here if needed

    COMMIT; -- Commit the transaction

  EXCEPTION
    WHEN OTHERS THEN
      -- Handle any exceptions that might occur
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK; -- Rollback the transaction to maintain data consistency
END submit_feedback;

PROCEDURE apply_for_leave (
  p_rollno      IN Stud_Info.rollno%TYPE,
  p_leave_dt    IN Leave.leave_dt%TYPE,
  p_address     IN Leave.address%TYPE,
  p_reason      IN Leave.reason%TYPE,
  p_no_of_day   IN Leave.no_of_day%TYPE
) IS
  BEGIN
  -- Step 1: Input Validation (if necessary)
  -- This could include checking if the provided roll number exists in the Stud_Info table
  -- Also, validate other parameters as needed

  -- Step 2: Insert into Database
    INSERT INTO Leave (rollno, leave_dt, address, reason, no_of_day)
    VALUES (p_rollno, p_leave_dt, p_address, p_reason, p_no_of_day);
```

```
    -- Step 3: Notification (if necessary)
    -- Add notification logic here if needed

    -- Step 4: Logging
    -- Add logging logic here if needed

     COMMIT; -- Commit the transaction
    EXCEPTION
     WHEN OTHERS THEN
      -- Handle any exceptions that might occur
       DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
       ROLLBACK; -- Rollback the transaction to maintain data consistency
  END apply_for_leave;

END student_pkg;
/
```

❖ Trigger

```
-- 1. Room Allocator Trigger
CREATE OR REPLACE TRIGGER room_allocator_trigger
BEFORE INSERT ON Stud_Info
FOR EACH ROW
DECLARE
  v_room_no Rooms.rm_no%TYPE;
BEGIN
  -- Find the room with the smallest difference between capacity and occupancy
  SELECT rm_no INTO v_room_no
  FROM (
    SELECT rm_no, capacity - NVL(occupancy, 0) AS diff
    FROM Rooms
    ORDER BY diff
  )
  WHERE ROWNUM = 1;

  -- Allocate the room to the student in Stud_Rooms table
  INSERT INTO Stud_Rooms (rollno, rm_no)
  VALUES (:NEW.rollno, v_room_no);

  -- Increase occupancy of the allocated room
  UPDATE Rooms
  SET occupancy = NVL(occupancy, 0) + 1
  WHERE rm_no = v_room_no;
END;
/

-- 2. Entry-Exit Time and Date Noter
CREATE OR REPLACE TRIGGER entry_exit_time_noter_trigger
```

```
BEFORE INSERT ON Entry_Exit
FOR EACH ROW
BEGIN
    -- Set the entry/exit date to the current system date
    IF :NEW.ee_date IS NULL THEN
        :NEW.ee_date := TRUNC(SYSDATE);
    END IF;

    -- Set the entry/exit time to the current system timestamp
    IF :NEW.ee_time IS NULL THEN
        :NEW.ee_time := SYSTIMESTAMP;
    END IF;
END;
/

-- 3. Feedback Date Noter
CREATE OR REPLACE TRIGGER feedback_date_noter_trigger
BEFORE INSERT ON Feedback
FOR EACH ROW
BEGIN
    -- Set the feedback date to the current system date
    IF :NEW.fb_dt IS NULL THEN
        :NEW.fb_dt := SYSDATE;
    END IF;

    -- Set the day of the week
    :NEW.day := TO_CHAR(SYSDATE, 'DAY');
END;
/

-- 4. Complain Date Noter
```

```
CREATE OR REPLACE TRIGGER complain_date_noter_trigger
BEFORE INSERT ON Complain
FOR EACH ROW
BEGIN
   -- Set the complain date to the current system date
   IF :NEW.com_dt IS NULL THEN
      :NEW.com_dt := SYSDATE;
   END IF;
END;
/

-- 5. Roll No Allocator
                                          CREATE OR REPLACE TRIGGER student_before_insert
BEFORE INSERT ON Stud_Info
FOR EACH ROW
DECLARE
   max_rollno Stud_Info.rollno%TYPE;
BEGIN
   SELECT COUNT(*) INTO max_rollno FROM Stud_Info;
   max_rollno := max_rollno + 1;
   :NEW.rollno := 'STUD' || LPAD(TO_CHAR(max_rollno, 5, 0));
END;
/

-- 6. Emp No Allocator
CREATE OR REPLACE TRIGGER employee_before_insert
BEFORE INSERT ON Emp_Info
FOR EACH ROW
DECLARE
   max_empno Emp_Info.empno%TYPE;
BEGIN
```

```
    SELECT COUNT(*) INTO max_empno FROM Emp_Info;
    max_empno := max_empno + 1;
    :NEW.empno := 'EMP' || LPAD(TO_CHAR(empno, 5, 0));
END;
/

-- 7. Complain Assigner
CREATE OR REPLACE TRIGGER assign_complaint_to_employee
AFTER INSERT ON Complain
FOR EACH ROW
DECLARE
    v_empno Emp_Info.empno%TYPE;
BEGIN
    -- Retrieve the employee number based on the complaint type
    SELECT empno INTO v_empno
    FROM (
        SELECT empno
        FROM Emp_Job_Info
        WHERE ejob = (
            CASE :NEW.com_type
                WHEN 'MESS' THEN 'MESS STAFF'
                WHEN 'CLEANING' THEN 'SWEEPER'
                WHEN 'GARDENING' THEN 'GARDENER'
                ELSE 'OFFICE STAFF'
            END
        )
        ORDER BY DBMS_RANDOM.VALUE
    )
    WHERE ROWNUM = 1;

    -- Insert the complaint into the Emp_Comp_Junc table
    INSERT INTO Emp_Comp_Junc (rollno, com_dt, empno)
    VALUES (:NEW.rollno, SYSDATE, v_empno);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Handle the case where no employees are found for the given job role
        INSERT INTO Complain (rollno, com_dt, com_type, is_done)
        VALUES (:NEW.rollno, SYSDATE, :NEW.com_type, 'N');

        -- You can log the error or take other appropriate actions
        DBMS_OUTPUT.PUT_LINE('No employees found for the specified job role.');
    WHEN OTHERS THEN
        -- Handle other exceptions if needed
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;/
```

❖ Table query's

```
-- Student tables

-- 1

CREATE TABLE Stud_Info (

    rollno VARCHAR2(20) PRIMARY KEY,

    sname VARCHAR2(100),

    sphoneno VARCHAR2(20),

    saddress VARCHAR2(200),

    sdob DATE,

    course VARCHAR2(100),

    institute VARCHAR2(100),

    CONSTRAINT chk_sphone CHECK(LENGTH(sphoneno) = 10)

);


-- 2

CREATE TABLE Stud_Email (

    rollno VARCHAR2(20),

    semail VARCHAR2(100),

    CONSTRAINT fk_stud_email FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE CASCADE,

    CONSTRAINT fk_stud_password FOREIGN KEY (semail) REFERENCES Stud_Password(semail) ON DELETE
CASCADE,

    CONSTRAINT pk_stud_email PRIMARY KEY (rollno, semail)

);


-- 3

CREATE TABLE Stud_Password (

    semail VARCHAR2(100),

    spass VARCHAR2(100),

    CONSTRAINT pk_stud_password PRIMARY KEY (semail)

);
```

```
-- 4
CREATE TABLE Stud_Rooms (
    rollno VARCHAR2(20),
    rm_no VARCHAR2(20),
    CONSTRAINT fk_stud_rooms_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,
    CONSTRAINT fk_stud_rooms FOREIGN KEY (rm_no) REFERENCES Rooms(rm_no) ON DELETE CASCADE,
    CONSTRAINT pk_stud_rooms PRIMARY KEY (rollno, rm_no)
);

-- Employee tables

-- 5
CREATE TABLE Emp_Info (
    empno VARCHAR2(20) PRIMARY KEY,
    ename VARCHAR2(100),
    ephoneno VARCHAR2(20),
    eaddress VARCHAR2(200),
    gender VARCHAR2(10),
    marital_st CHAR(1),
    edob DATE,
    CONSTRAINT chk_ephoneno CHECK(LENGTH(ephoneno) = 10),
    CONSTRAINT chk_gender CHECK(gender IN ('MALE', 'FEMALE', 'OTHERS')),
    CONSTRAINT chk_marital_st CHECK(marital_st IN ('Y', 'N'))
);

-- 6
CREATE TABLE Emp_Email (
    empno VARCHAR2(20),
    e_email VARCHAR2(100),
    CONSTRAINT fk_emp_password FOREIGN KEY (e_email) REFERENCES Emp_Email(e_email) ON DELETE
CASCADE,

    CONSTRAINT fk_emp_email FOREIGN KEY (empno) REFERENCES Emp_Info(empno) ON DELETE CASCADE,

    CONSTRAINT pk_emp_email PRIMARY KEY (empno, e_email)

);


-- 7

CREATE TABLE Emp_Password (

    e_email VARCHAR2(100),

    epass VARCHAR2(100),

    CONSTRAINT pk_emp_password PRIMARY KEY (e_email)

);


-- 8

CREATE TABLE Emp_Job_Info (

    empno VARCHAR2(20),

    ejob VARCHAR2(100),

    CONSTRAINT fk_emp_job_info FOREIGN KEY (empno) REFERENCES Emp_Info(empno) ON DELETE
CASCADE,

    CONSTRAINT fk_emp_job FOREIGN KEY (ejob) REFERENCES Job(ejob) ON DELETE CASCADE,

    CONSTRAINT pk_emp_job_info PRIMARY KEY (empno)

);


-- 9

CREATE TABLE Job (

    ejob VARCHAR2(20) PRIMARY KEY,

    sal NUMBER,

    CONSTRAINT chk_job CHECK(ejob IN ('MESS STAFF', 'SWEEPER', 'GARDENER', 'OFFICE STAFF'))

);


-- 10

CREATE TABLE Emp_Comp_Junc (
    rollno VARCHAR2(20),

    com_dt DATE,

    empno VARCHAR2(20),

    CONSTRAINT fk_emp_comp_junc_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,

    CONSTRAINT fk_emp_comp_junc_empno FOREIGN KEY (empno) REFERENCES Emp_Info(empno) ON
DELETE CASCADE,

    CONSTRAINT pk_emp_comp_junc PRIMARY KEY (rollno, com_dt, empno)

);


-- Vehicle table


-- 11

CREATE TABLE Vehicle (

    vl_id VARCHAR2(20) PRIMARY KEY,

    rollno VARCHAR2(20),

    reg_no VARCHAR2(50),

    vl_type VARCHAR2(50),

    CONSTRAINT fk_vehicle_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,

    CONSTRAINT chk_vltype CHECK(vl_type in ('BIKE', 'MOPET'))

);


-- Leave table


-- 12

CREATE TABLE Leave (

    rollno VARCHAR2(20),

    leave_dt DATE,

    address VARCHAR2(200),

    reason VARCHAR2(200),

    no_of_day NUMBER,
    CONSTRAINT fk_leave_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE CASCADE,
    CONSTRAINT pk_leave PRIMARY KEY (rollno, leave_dt)
);

-- Mess-Menu table

-- 13
CREATE TABLE Mess (
    mess_id NUMBER PRIMARY KEY,
    monday VARCHAR2(100),
    tuesday VARCHAR2(100),
    wednesday VARCHAR2(100),
    thursday VARCHAR2(100),
    friday VARCHAR2(100),
    saturday VARCHAR2(100),
    sunday VARCHAR2(100)
);

-- 14
CREATE TABLE Mess_Fb_Junc (
    rollno VARCHAR2(20),
    fb_dt DATE,
    mess_id NUMBER,
    CONSTRAINT fk_mess_fb_junc_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,
    CONSTRAINT fk_mess_fb_junc_mess_id FOREIGN KEY (mess_id) REFERENCES Mess(mess_id) ON DELETE
CASCADE,
    CONSTRAINT pk_mess_fb_junc PRIMARY KEY (rollno, fb_dt)
);

-- Feedback table
-- 15
CREATE TABLE Feedback (
    rollno VARCHAR2(20),
    fb_dt DATE,
    day VARCHAR2(20),
    feedback VARCHAR2(20),
    rating NUMBER,
    CONSTRAINT fk_feedback_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,
    CONSTRAINT pk_feedback PRIMARY KEY (rollno, fb_dt, day),
    CONSTRAINT chk_rating CHECK(rating < 6)
);

-- Rooms table

-- 16
CREATE TABLE Rooms (
    rm_no VARCHAR2(20) PRIMARY KEY,
    capacity NUMBER,
    occupancy NUMBER
);

-- Complain table

-- 17
CREATE TABLE Complain (
    rollno VARCHAR2(20),
    com_dt DATE,
    com_type VARCHAR2(50),
    is_done CHAR(1),
    CONSTRAINT fk_complain_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,
    CONSTRAINT chk_isdone CHECK(is_done IN ('Y', 'N')),

    CONSTRAINT chk_comtype CHECK(com_type IN ('MESS', 'CLEANING', 'GARDENING', 'OTHERS')),

    CONSTRAINT pk_complain PRIMARY KEY (rollno, com_dt)

);


-- Entry-Exit table


-- 18

CREATE TABLE Entry_Exit (

    rollno VARCHAR2(20),

    ee_time TIMESTAMP,

    ee_date DATE,

    place VARCHAR2(100),

    ee_type VARCHAR2(20),

    CONSTRAINT fk_entry_exit_rollno FOREIGN KEY (rollno) REFERENCES Stud_Info(rollno) ON DELETE
CASCADE,

    CONSTRAINT pk_entry_exit PRIMARY KEY (rollno, ee_date, ee_time)

);
```