Khulna University of Engineering & Technology

Course title: Database System Laboratory

Course No: CSE 3210

Project Report on: Hall Management Database System Design

Submitted To:	Submitted By:
Ms. Dola Das Assistant Professor, Department of CSE, Khulna University of Engineering & Technology. Mr. Md. Abdus Salim Mollah Assistant Professor, Department of CSE, Khulna University of Engineering & Technology.	Md. Abu Hanif Khan Roll no: 2009054 Year:3rd Semester:2nd Batch: 2020 Department: Electronics and Communication Engineering, Khulna University of Engineering & Technology

Objectives:

- To design and implement a relational database for managing Hall information.
- To integrate various entities, including staff, students, payments, and meal items.
- To perform advanced data retrieval and manipulation using SQL queries.

Introduction:

This project aims to design and manage a comprehensive database system for a University Hall Management System. The key objective is to automate and streamline the storage, retrieval, and management of data related to student accommodation, room allocation, maintenance requests, and fee management. By implementing this system, universities can efficiently oversee hall-related operations, ensuring transparency, reducing manual workloads, and improving accessibility for both administrators and students. This approach will significantly enhance operational efficiency and support better decision-making through organized data management.

Project Feature:

The ER diagram provides a concise representation of the Hall Management System project, featuring six core entities: Hall, Room, Student, Staff, Meal Schedule, and Payment Details.

Core Entities and Attributes:

1. Hall Entity

- **Attributes**:
  - hall_id (Primary Key)
  - hall_name
  - capacity (Total capacity of the hall)
  - location

2. Room Entity

- **Attributes**:

    - room_id (Primary Key)
    - room_number
    - capacity (Maximum capacity of the room)
    - hall_id (Foreign Key referencing hall_id in the Hall table)
    - occupant_id (Foreign Key referencing student_id in the Student table, if applicable)

3. Student Entity

- **Attributes**:
    - student_id (Primary Key)
    - full_name
    - email
    - dob (Date of Birth)
    - department (Department of study)
    - enrollment_year
    - hall_id (Foreign Key referencing hall_id in the Hall table)

4. Staff Entity

- **Attributes**:
    - staff_id (Primary Key)
    - full_name
    - email
    - position (e.g., Hall Manager, Cleaning Staff)
    - hall_id (Foreign Key referencing hall_id in the Hall table)

5. Meal Schedule Entity

- **Attributes**:
    - schedule_id (Primary Key)
    - hall_id (Foreign Key referencing hall_id in the Hall table)
    - meal_date
    - meal_time
    - menu_item

6. Payment Details Entity

- **Attributes**:
    - payment_id (Primary Key)
    - student_id (Foreign Key referencing student_id in the Student table)

- total_amount (Total amount payable by the student)
- refund (Any refundable amount)
- net_payment (Net payment after refund adjustment)
- receipt_sl (Receipt serial number, unique)
- paid_date (Payment date)

Relationships:

1. **Hall and Room**:
   - The relationship between Hall and Room is **one-to-many**, where a single hall contains multiple rooms, but each room belongs to only one hall.
2. **Hall and Student**:
   - The relationship between Hall and Student is **one-to-many**, where a single hall accommodates multiple students, but each student is assigned to only one hall.
3. **Hall and Staff**:
   - The relationship between Hall and Staff is **one-to-many**, where a single hall can have multiple staff members, but each staff member is assigned to only one hall.
4. **Room and Student**:
   - The relationship between Room and Student is **one-to-one** or **one-to-many**, depending on room capacity, where a room may accommodate one or more students.
5. **Hall and Meal Schedule**:
   - The relationship between Hall and Meal Schedule is **one-to-many**, where a single hall has multiple meal schedules, but each schedule belongs to only one hall.
6. **Student and Payment Details**:
   - The relationship between Student and Payment Details is **one-to-one**, where each student can have only one payment record per transaction, but a student may have multiple transactions over time.

Summary:

The schema design supports efficient management of hall operations, including room assignments, staff coordination, student accommodation, meal scheduling, and financial transactions. Key features include:

- Tracking students assigned to specific halls and rooms.
- Viewing staff members responsible for individual halls.
- Generating meal schedules for a hall.
- Managing and querying payment records for students, including refunds and receipts.

ER Diagram:

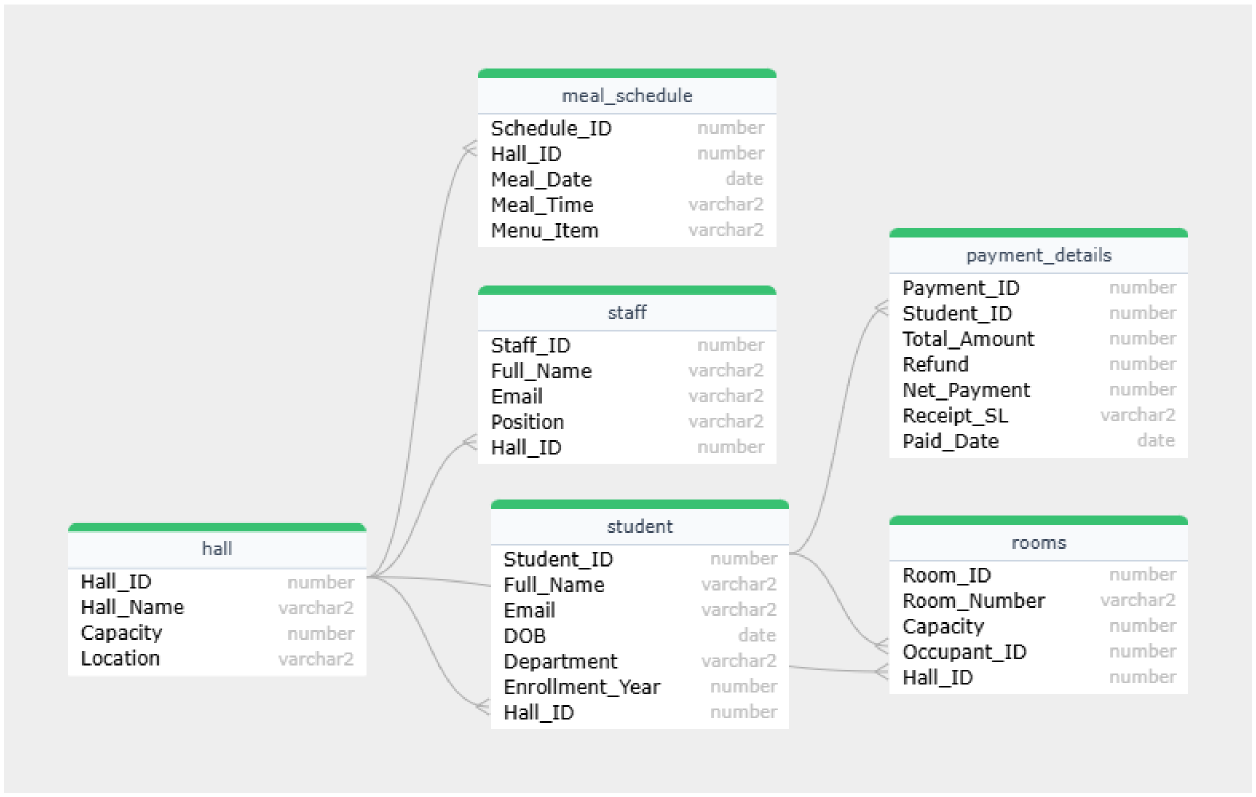

Figure: ER Diagram of University Hall Management System

OPERATIONS:

1. Describe Table Structure:

- DESC hall;

2. Retrieve All Records from Tables:

- SELECT * FROM hall;

3. Add a New Column:

- Add a new column phone_number to the student table:
    - ALTER TABLE student ADD phone_number VARCHAR2(20);

4. Rename a Column:

- Rename column email to student_email in the student table:
    - ALTER TABLE student RENAME COLUMN email TO student_email;

5. Modify Column Data Type:

- Update the data type of the capacity column in the hall table:
    - Create a backup of hall:
        - CREATE TABLE hall_backup AS SELECT * FROM hall;
    - Update data to avoid conflicts:
        - UPDATE hall SET capacity = NULL;
    - Modify column capacity data type:
        - ALTER TABLE hall MODIFY capacity NUMBER(5);
    - Restore original values to capacity:
        - UPDATE hall h SET h.capacity = (SELECT b.capacity FROM hall_backup b WHERE h.Hall_ID = b.Hall_ID);

6. Drop a Column:

- Drop column occupant_id from the rooms table:
    - ALTER TABLE rooms DROP COLUMN occupant_id;

7. Update Records:

- Update the department for a student with ID 2009054:
    - UPDATE student SET department = 'Electronics and Communication Engineering' WHERE student_id = 2009054;

8. Delete a Record:

- Delete a student record with ID 2009001:
    - DELETE FROM student WHERE student_id = 2009001;

9. Set Operations and Subqueries:

- Retrieve student information assigned to Hall H01:
    - SELECT Full_Name, Student_ID FROM student WHERE Student_ID IN (SELECT occupant_id FROM rooms WHERE hall_id = '1');
- Retrieve students and staff in Hall 1, including duplicates:

    - SELECT Full_Name, Student_ID FROM student WHERE hall_id = '1' UNION ALL SELECT Full_Name, staff_id FROM staff WHERE hall_id = '1';
- Find common students and staff in Hall 3:
    - SELECT Full_Name, Student_ID FROM student WHERE hall_id = '3' INTERSECT SELECT Full_Name, staff_id FROM staff WHERE hall_id = '3';
- Find students not staff in Hall 4:
    - SELECT Full_Name, Student_ID FROM student WHERE hall_id = '4' MINUS SELECT Full_Name, staff_id FROM staff WHERE hall_id = '4';
- Combine students and staff with meal plans in Hall 5:
    - SELECT Full_Name, Student_ID FROM student WHERE hall_id = '5' AND Student_ID IN (SELECT Student_ID FROM meal_schedule WHERE hall_id = '5') UNION SELECT Full_Name, staff_id FROM staff WHERE hall_id = '5' AND staff_id IN (SELECT staff_id FROM meal_schedule WHERE hall_id = '5');

10. **Aggregate Functions:**
- Calculate net payment divided by 10:
    - SELECT payment_id, student_id, net_payment, (net_payment / 10) AS net_payment_divided FROM payment_details;
- Fetch payments for specific students (IDs 2009001 and 2009054):
    - SELECT * FROM payment_details WHERE student_id = 2009001 OR student_id = 2009054;
- Fetch records where net payment is between 5000 and 5500:
    - SELECT * FROM payment_details WHERE net_payment BETWEEN 5000 AND 5500;
- Fetch records where the student has a specific net payment (e.g., 5000 or 5100):
    - SELECT * FROM payment_details WHERE net_payment IN (5000, 5100);
- Fetch payments sorted by net payment (ascending and descending):
    - SELECT * FROM payment_details ORDER BY net_payment ASC;
    - SELECT * FROM payment_details ORDER BY net_payment DESC;
- Count the total number of payments:
    - SELECT COUNT(*) AS total_payments FROM payment_details;
- Fetch the maximum and minimum net payment amounts:

    - SELECT MAX(net_payment) AS max_payment, MIN(net_payment) AS min_payment FROM payment_details;
- Get the total payments per student:
    - SELECT student_id, SUM(net_payment) AS total_payment FROM payment_details GROUP BY student_id;
- Fetch students who have paid more than 5000 in total:
    - SELECT student_id, SUM(net_payment) AS total_payment FROM payment_details GROUP BY student_id HAVING SUM(net_payment) > 5000;
- Get average net payment:
    - SELECT AVG(net_payment) AS average_net_payment FROM payment_details;

11. Joins:

- **Inner Join**:
    - SELECT student.student_id, student.full_name, hall.hall_name FROM student JOIN hall ON student.hall_id = hall.hall_id;
- **Natural Join**:
    - SELECT student.student_id, student.full_name, hall.hall_name FROM student NATURAL JOIN hall;
- **Equi Join**:
    - SELECT rooms.room_number, student.student_id, student.full_name FROM rooms JOIN student ON rooms.occupant_id = student.student_id;
- **Non-Equi Join**:
    - SELECT student.student_id, student.full_name, meal_schedule.meal_date FROM student JOIN meal_schedule ON student.enrollment_year < EXTRACT(YEAR FROM meal_schedule.meal_date);
- **Self Join**:
    - SELECT s1.Staff_ID AS Staff_ID, s1.Full_Name AS Staff_Name, s2.Full_Name AS Colleague_Name FROM staff s1 LEFT JOIN staff s2 ON s1.Hall_ID = s2.Hall_ID AND s1.Staff_ID != s2.Staff_ID;
- **Left Outer Join**:
    - SELECT student.Student_ID, student.Full_Name AS Student_Name, hall.Hall_Name AS Assigned_Hall, hall.Location AS Hall_Location FROM student LEFT OUTER JOIN hall ON student.Hall_ID = hall.Hall_ID;

- **Right Outer Join**:
  - SELECT hall.Hall_ID, hall.Hall_Name, student.Student_ID AS Resident_Student_ID, student.Full_Name AS Resident_Student_Name FROM hall RIGHT OUTER JOIN student ON student.Hall_ID = hall.Hall_ID;

12. PL/SQL Operations:

- **Max Capacity of a Hall**:

```
DECLARE max_capacity NUMBER;
hall_name VARCHAR2(100);
BEGIN
        SELECT Capacity, Hall_Name INTO max_capacity, hall_name FROM
        hall WHERE Capacity = (SELECT MAX(Capacity) FROM hall WHERE
        Capacity IS NOT NULL);
        DBMS_OUTPUT.PUT_LINE('Hall with Maximum Capacity: '
        ||hall_name || ' (' || max_capacity || ')');
END;
```

- **Loop and Cursor to Fetch Room Details**:

```
DECLARE
CURSOR room_cursor IS
        SELECT Room_ID, Room_Number, Capacity FROM rooms WHERE
Hall_ID = 1;
room_row room_cursor%ROWTYPE;
BEGIN
        OPEN room_cursor;
        LOOP FETCH room_cursor INTO room_row;
        EXIT WHEN room_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Room ID: ' || room_row.Room_ID || ',
        Number: ' || room_row.Room_Number || ', Capacity: ' ||
        room_row.Capacity); END LOOP; CLOSE room_cursor;
END;
```

- **Update Staff Hall Assignment**:

```
DECLARE
staff_id NUMBER := 101;
new_hall_id NUMBER := 2;
BEGIN
UPDATE staff SET Hall_ID = new_hall_id WHERE Staff_ID = staff_id;
IF SQL%ROWCOUNT = 0 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('No staff member found with the given
        ID.');
ELSE
        DBMS_OUTPUT.PUT_LINE('Staff hall assignment updated
successfully.');
END IF;
EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('An error
occurred: ' || SQLERRM); END;
```

- **Delete a Student and Process Refund**:

```
DECLARE
student_id NUMBER := 1001;
refund_amount NUMBER := 100;
BEGIN UPDATE payment_details SET Refund = Refund + refund_amount,
Net_Payment = Net_Payment - refund_amount WHERE Student_ID =
student_id;
DELETE FROM student WHERE Student_ID = student_id;
IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No student found with the given ID.');
ELSE
        DBMS_OUTPUT.PUT_LINE('Student deleted successfully, and refund
        processed.');
END IF;
EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('An error
occurred: ' || SQLERRM);
 END;
```

- **Insert Meal Schedules for a Day**:

```
DECLARE meal_date DATE := TO_DATE('19/12/2024', 'DD/MM/YYYY');
meal_time VARCHAR2(20);
menu_item VARCHAR2(200);
BEGIN
FOR i IN 1..3 LOOP
        IF i = 1 THEN
                meal_time := 'Breakfast'; menu_item := 'Pancakes';
        ELSIF i = 2 THEN
                meal_time := 'Lunch'; menu_item := 'Grilled Chicken';
        ELSE
                meal_time := 'Dinner'; menu_item := 'Pasta';
        END IF;
        INSERT INTO meal_schedule (Schedule_ID, Hall_ID, Meal_Date,
        Meal_Time, Menu_Item) VALUES (i, 1, meal_date + i, meal_time,
        menu_item);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Meal schedule inserted successfully.'); END;
```

- **Create Procedure to Add Hall**:

```
CREATE OR REPLACE PROCEDURE add_hall (p_hall_id IN NUMBER,
p_hall_name IN VARCHAR2, p_capacity IN NUMBER, p_location IN
VARCHAR2)
AS
BEGIN
INSERT INTO hall (hall_id, hall_name, capacity, location) VALUES (p_hall_id,
p_hall_name, p_capacity, p_location);
COMMIT;
DBMS_OUTPUT.PUT_LINE('Hall added successfully!');
EXCEPTION WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('Error: Duplicate hall ID.');
WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END
add_hall;
```

- **Execute Procedure to Add Hall**:

```
BEGIN
add_hall(7, 'Main Auditorium', 500, 'Building A');
END;
```

Discussion:

The Hall Management System database schema is designed to be both efficient and scalable. It is structured to avoid redundancy by ensuring that each piece of data is stored in its respective table, with foreign keys linking related information. This approach upholds data integrity and allows for efficient handling of large datasets. The design also supports scalability, making it easy to extend the system in the future. For example, new features like additional hall facilities, payment methods, or student services can be seamlessly integrated by adding new tables and linking them using foreign keys, ensuring the system can evolve as needed.

Conclusion:

The Hall Management System effectively streamlines the management of room assignments, student accommodation, staff coordination, meal scheduling, and payment processing. With a well-structured database and optimized SQL queries, the system ensures precise data retrieval and smooth operation. The ER diagram and schema clearly illustrate the relationships between students, rooms, staff, and meals, promoting better organization and clarity. This project demonstrates the power of a database system in improving hall management efficiency, providing a strong foundation for future improvements such as automated notifications, enhanced reporting, and advanced analytics for further optimization of operations.