SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering, Mumbai
A.Y. 2023 - 24
Course: Database Management Systems

Project Report

Program	MBA Tech Data Science	
Semester	4	
Name of the Project:	Hostel Room Allocation System	
Details of Project Members		
Batch	Roll No.	Name
J3	S061	Sumit Patil
J3	S062	Swayam V.
Date of Submission:		

Contribution of each project Members:

Roll No.	Name:	Contribution
S061	Sumit Patil	Equal
S062	Swayam V.	Equal

Github link of your project: sumitpatil95/DBMS-Major-Project: DBMS Major Project on Hostel Room Allocation System (github.com)

Note:

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004_DBMSProject)
3. Submit all relevant files of your work (Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

Rubrics for the Project evaluation:

First phase of evaluation: Innovative Ideas (5 Marks) Design and Partial implementation (5 Marks)	10 marks

Final phase of evaluation Implementation, presentation and viva, Self-Learning and Learning Beyond classroom	10 marks

Project Report

Hostel Room Allocation System

by

Sumit Patil, Roll number: S061

Swayam V., Roll number: S062

Course: DBMS

AY: 2023-24

Table of Contents

I. Storyline

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document is narrative and graphical documentation of the software design for the project including ER Diagrams, Conceptual Schema, and other supporting requirement information. The purpose of this is to deal with Hostel Management System in an easy and an efficient manner. And create strong and secrete database that allows for any connection in a secret way to prevent any outside or inside attacks. Hostel Management System is designed for Hostel (like schools, Universities).

II. Components of Database Design

So,

This section of the document explains the entities used in the project, their attributes and how they will work together. Basically, this is intended to make the design more easy and understandable for everyone.
Entities

1. Hostel

2. Administrator

3. Student

4. Room

5. Visitor

6. Fees

1. Hostel

An Institution has many hostels and each hostel is represented using this 'Hostel' entity. Hostel model takes part in the following relationships.
1. Administrator manages Hostel.
2. Hostel has Students.
3. Hostel has Rooms.
Attributes

Name	Data Type	Type
Hostel_ID	integer	Primary Key attribute
Hostel_name	string	Non_key attribute
No_of_rooms	integer	Non_key attribute
no_of_students	integer	Non_key attribute

2. Administrator

Every hostel has an administrator and is represented using the ‘administrator’ entity. Administrator entity takes part in following relationships.

1. Administrator manages Hostel.

Attributes

Name	DataType	Type
ID	integer	Primary Key attribute
Fname	string	Non_key attribute
Lname	string	Non_key attribute
Mob_No	string	Non_key attribute
Hostel_id	integer	Foreign Key attribute

3. Student

Every hostel has students and they are represented by the ‘student’ entity. Student entity participates in the following relationships.

1. Hostel has Students.
2. Student has visitor.
3. Students stay at room

Attributes

Name	Data Type	Type
Student_ID	integer	Primary Key attribute
Fname	string	Non_key attribute
Lname	string	Non_key attribute
Mob_No	string	Non_key attribute
Dept	string	Non_key attribute
Year_of_study	integer	Non_key attribute
Hostel_id	integer	Foreign Key attribute
Room_id	integer	Foreign Key attribute

4. Room

Every Hostel has rooms and they are represented using ‘room’ entity. Room entity participates in the following relationships.

1. Hostel has Rooms.
2. Student stays at room.
3. Room has Furniture.

Attributes

Name	Data Type	Type
Hostel_ID	integer	Foreign Key attribute
Room_ID	integer	Partial Key attribute

5. Visitors

Every student has visitors and they are represented using ‘Visitor’ entity. Visitor entity participates in the following relationships.

1. Student has visitors.

Attributes

Name	Data Type	Type
Visitor_ID	integer	Primary Key attribute
In time	Date-time field	Non_key attribute
Out time	Date-time field	Non_key attribute
Date	Date-time field	Non_key attribute
Name	string	Non_key attribute
Student_id	integer	Foreign Key attribute

6. Fees

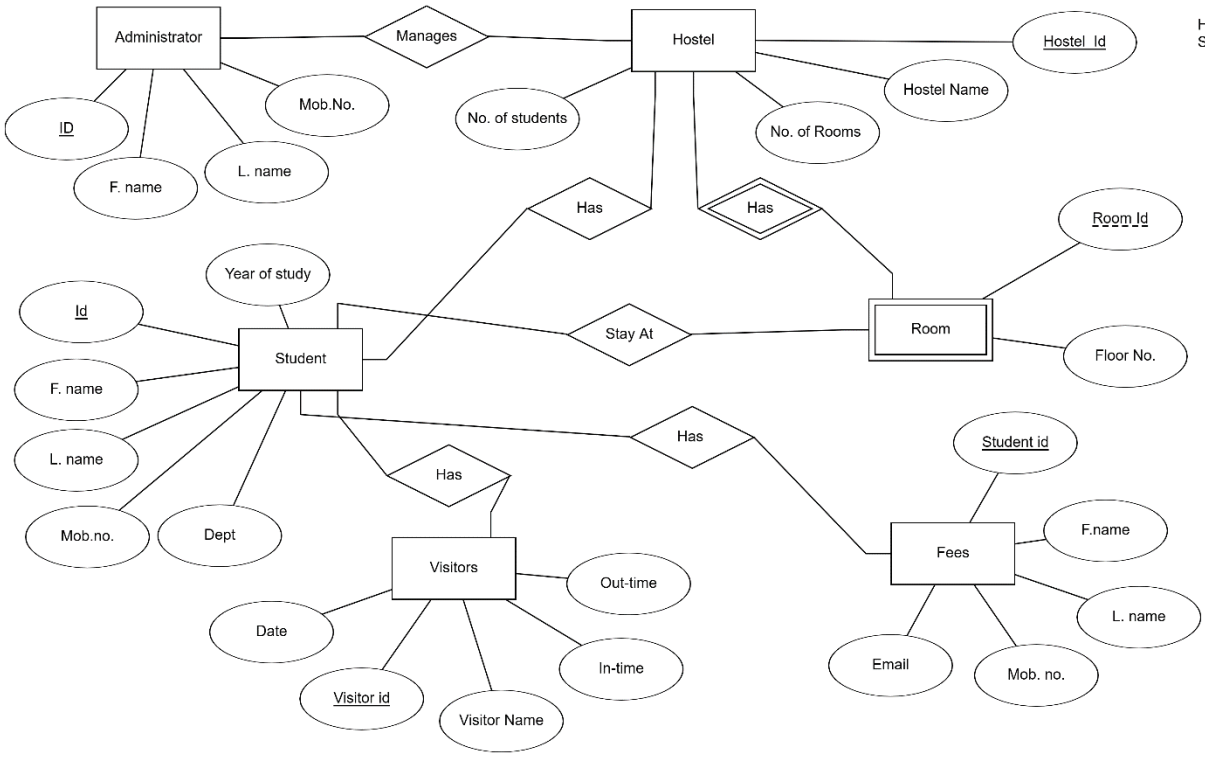
Every Student need to pay particular amount of fees accordingly.

1. Student has Fees

Attributes

Name	Data Type	Type
Student Id	integer	Primary Key attribute
First name	string	Non_key attribute
Last name	string	Non_key attribute
Mobile no.	string	Non_key attribute

III. Entity Relationship Diagram



IV. Relational Model

admin_id	first_name	last_name	mobile_number
1	John	Doe	1234567890
2	Jane	Smith	9876543210
3	Michael	Johnson	5551234567
4	Emily	Davis	9998887776
5	David	Brown	4445556667
6	Sarah	Wilson	7778889990
7	Robert	Martinez	1112223334
8	Jennifer	Anderson	6667778885
9	Christopher	Taylor	2223334445
10	Amanda	Thomas	8889990001

hostel_id	hostel_name	num_students	num_rooms
1	Alpha Hostel	100	50
2	Beta Hostel	120	60
3	Gamma Hostel	80	40
4	Delta Hostel	150	75
5	Epsilon Hostel	90	45
6	Zeta Hostel	110	55
7	Eta Hostel	130	65
8	Theta Hostel	95	48
9	Iota Hostel	105	52
10	Kappa Hostel	140	70

student_id	first_name	last_name	mobile_number	dept	year_of_study
1	Alice	Johnson	1112223333	Computer Science	2
2	Bob	Smith	4445556666	Electrical Engineering	3
3	Charlie	Brown	7778889999	Mechanical Engineering	1
4	Diana	Miller	2223334444	Chemical Engineering	4
5	Eva	Wilson	5556667777	Civil Engineering	2
6	Frank	Taylor	8889990000	Computer Science	3
7	Grace	Anderson	1234567890	Electrical Engineering	1
8	Harry	Davis	9876543210	Mechanical Engineering	4
9	Ivy	Martinez	6667778888	Chemical Engineering	2
10	Jack	White	9998887777	Civil Engineering	3

room_id	floor_number	hostel_id
1	1	1
2	2	1
3	1	2
4	2	2
5	1	3
6	2	3
7	1	4
8	2	4
9	1	5
10	2	5

visitor_id	date	visitor_name	in_time	out_time
1	2024-03-31	John Doe	08:00:00	10:00:00
2	2024-03-31	Jane Smith	09:30:00	11:30:00
3	2024-03-31	Michael Johnson	10:00:00	12:00:00
4	2024-03-31	Emily Davis	11:30:00	13:30:00
5	2024-03-31	David Brown	12:00:00	14:00:00
6	2024-03-31	Sarah Wilson	13:30:00	15:30:00
7	2024-03-31	Robert Martinez	14:00:00	16:00:00
8	2024-03-31	Jennifer Anderson	15:30:00	17:30:00
9	2024-03-31	Christopher Taylor	16:00:00	18:00:00
10	2024-03-31	Amanda Thomas	17:30:00	19:30:00

fee_id	student_id	first_name	last_name	mobile_number	email
1	1	Alice	Johnson	1112223333	alice@example.com
2	2	Bob	Smith	4445556666	bob@example.com
3	3	Charlie	Brown	7778889999	charlie@example.com
4	4	Diana	Miller	2223334444	diana@example.com
5	5	Eva	Wilson	5556667777	eva@example.com
6	6	Frank	Taylor	8889990000	frank@example.com
7	7	Grace	Anderson	1234567890	grace@example.com
8	8	Harry	Davis	9876543210	harry@example.com
9	9	Ivy	Martinez	6667778888	ivy@example.com
10	10	Jack	White	9998887777	jack@example.com

V. Normalization

Till 3NFof the project

Step 1: Analyze Dependencies
Before normalization, let's analyze the functional dependencies in the provided tables:

Administrator:
admin_id -> first_name, last_name, mobile_number
Hostel:
hostel_id -> hostel_name, num_students, num_rooms
Student:

student_id -> first_name, last_name, mobile_number, dept, year_of_study
Room:
room_id -> floor_number, hostel_id
Visitors:
visitor_id -> date, visitor_name, in_time, out_time
Fees:
fee_id -> student_id, first_name, last_name, mobile_number, email
Step 2: Normalize Tables
We will normalize the tables to third normal form (3NF), which means:

Eliminate repeating groups by putting each in a separate table.
Create separate tables for sets of values that apply to multiple records.
Remove dependencies on non-key attributes.
Based on the dependencies, here's how we can normalize the tables:

Table 1: Administrator
Attributes: admin_id (PK), first_name, last_name, mobile_number
This table appears to be in 3NF, as there are no repeating groups, and each attribute is dependent on the primary key.

Table 2: Hostel
Attributes: hostel_id (PK), hostel_name, num_students, num_rooms
This table is also in 3NF, as each attribute is dependent on the primary key.

Table 3: Student
Attributes: student_id (PK), first_name, last_name, mobile_number, dept, year_of_study
This table is in 3NF as well.

Table 4: Room
Attributes: room_id (PK), floor_number, hostel_id (FK)
This table is in 3NF.

Table 5: Visitors
Attributes: visitor_id (PK), date, visitor_name, in_time, out_time
This table is in 3NF.

Table 6: Fees
Attributes: fee_id (PK), student_id (FK), email
A separate table should be created for student details to avoid redundancy:

Table 7: Student_Details
Attributes: student_id (PK), first_name, last_name, mobile_number, dept, year_of_study
Updated Tables:
Table 1: Administrator
Attributes: admin_id (PK), first_name, last_name, mobile_number
Table 2: Hostel

Attributes: hostel_id (PK), hostel_name, num_students, num_rooms
Table 3: Student
Attributes: student_id (PK), first_name, last_name, mobile_number, dept, year_of_study
Table 4: Room
Attributes: room_id (PK), floor_number, hostel_id (FK)
Table 5: Visitors
Attributes: visitor_id (PK), date, visitor_name, in_time, out_time
Table 6: Fees
Attributes: fee_id (PK), student_id (FK), email
Table 7: Student_Details
Attributes: student_id (PK), first_name, last_name, mobile_number, dept, year_of_study
These tables are now normalized up to 3NF.

BCNF of the project

To perform Boyce-Codd Normal Form (BCNF) normalization on the tables, we need to ensure that for every functional dependency

→

X→Y, where

X is a superkey, the determinant

X is a candidate key. If any non-trivial functional dependencies violate this condition, we decompose the table accordingly.

Let's analyze each table and normalize it to BCNF:

Administrator Table
There are no non-trivial functional dependencies in this table, and the primary key
admin_id
admin_id uniquely identifies each tuple. Therefore, the table is already in BCNF.

Hostel Table
There are no non-trivial functional dependencies in this table, and the primary key
hostel_id
hostel_id uniquely identifies each tuple. Therefore, the table is already in BCNF.

Student Table
No non-trivial functional dependencies are present in this table. The primary key
student_id
student_id uniquely identifies each tuple. Hence, the table is in BCNF.

Room Table
The only functional dependency present is

hostel_id
→
floor_number
hostel_id→floor_number, where
hostel_id
hostel_id is not a superkey.
To normalize to BCNF, we decompose the table:
As room_id and floor_number (Room_details) & room_id and hostel_id (Room1)

Visitors Table
There are no non-trivial functional dependencies in this table, and the primary key
visitor_id
visitor_id uniquely identifies each tuple. Therefore, the table is already in BCNF.

Fees Table
The only functional dependency present is
student_id
→
first_name
,
last_name
,
mobile_number
,
email
student_id→first_name,last_name,mobile_number,email, where
student_id
student_id is a superkey. Therefore, the table is already in BCNF.

After normalization, we have the following tables:

Administrator (BCNF)
Hostel (BCNF)
Student (BCNF)
Room (Decomposed into Room and Room_Details)
Visitors (BCNF)
Fees (BCNF)
Each table now satisfies the requirements of Boyce-Codd Normal Form.

VI. SQL Queries

1. Creating tables and inserting values and output

```sql
CREATE TABLE Administrator (
    admin_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    mobile_number VARCHAR(15)
);

CREATE TABLE Hostel (
    hostel_id INT PRIMARY KEY,
    hostel_name VARCHAR(100),
    num_students INT,
    num_rooms INT
);
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    mobile_number VARCHAR(15),
    dept VARCHAR(100),
    year_of_study INT
);

CREATE TABLE Room (
    room_id INT PRIMARY KEY,
    floor_number INT,
    hostel_id INT,
    FOREIGN KEY (hostel_id) REFERENCES Hostel(hostel_id)
);
CREATE TABLE Visitors (
    visitor_id INT PRIMARY KEY,
    date DATE,
    visitor_name VARCHAR(100),
    in_time TIME,
    out_time TIME
);
```

```sql
CREATE TABLE Fees (
    fee_id INT PRIMARY KEY,
    student_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    mobile_number VARCHAR(15),
    email VARCHAR(100),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);

-- Insert values into Administrator table
INSERT INTO Administrator (admin_id, first_name, last_name, mobile_number) VALUES
(1, 'John', 'Doe', '1234567890'),
(2, 'Jane', 'Smith', '9876543210'),
(3, 'Michael', 'Johnson', '5551234567'),
(4, 'Emily', 'Davis', '9998887776'),
(5, 'David', 'Brown', '4445556667'),
(6, 'Sarah', 'Wilson', '7778889990'),
(7, 'Robert', 'Martinez', '1112223334'),
(8, 'Jennifer', 'Anderson', '6667778885'),
(9, 'Christopher', 'Taylor', '2223334445'),
(10, 'Amanda', 'Thomas', '8889990001');

-- Insert values into Hostel table
INSERT INTO Hostel (hostel_id, hostel_name, num_students, num_rooms) VALUES
(1, 'Alpha Hostel', 100, 50),
(2, 'Beta Hostel', 120, 60),
(3, 'Gamma Hostel', 80, 40),
(4, 'Delta Hostel', 150, 75),
(5, 'Epsilon Hostel', 90, 45),
(6, 'Zeta Hostel', 110, 55),
(7, 'Eta Hostel', 130, 65),
(8, 'Theta Hostel', 95, 48),
(9, 'Iota Hostel', 105, 52),
(10, 'Kappa Hostel', 140, 70);
```

```sql
-- Insert values into Student table
INSERT INTO Student (student_id, first_name, last_name, mobile_number, dept,
year_of_study) VALUES
(1, 'Alice', 'Johnson', '1112223333', 'Computer Science', 2),
(2, 'Bob', 'Smith', '4445556666', 'Electrical Engineering', 3),
(3, 'Charlie', 'Brown', '7778889999', 'Mechanical Engineering', 1),
(4, 'Diana', 'Miller', '2223334444', 'Chemical Engineering', 4),
(5, 'Eva', 'Wilson', '5556667777', 'Civil Engineering', 2),
(6, 'Frank', 'Taylor', '8889990000', 'Computer Science', 3),
(7, 'Grace', 'Anderson', '1234567890', 'Electrical Engineering', 1),
(8, 'Harry', 'Davis', '9876543210', 'Mechanical Engineering', 4),
(9, 'Ivy', 'Martinez', '6667778888', 'Chemical Engineering', 2),
(10, 'Jack', 'White', '9998887777', 'Civil Engineering', 3);

-- Insert values into Room table
INSERT INTO Room (room_id, floor_number, hostel_id) VALUES
(1, 1, 1),
(2, 2, 1),
(3, 1, 2),
(4, 2, 2),
(5, 1, 3),
(6, 2, 3),
(7, 1, 4),
(8, 2, 4),
(9, 1, 5),
(10, 2, 5);
```

```sql
-- Insert values into Visitors table
INSERT INTO Visitors (visitor_id, date, visitor_name, in_time, out_time) VALUES
(1, '2024-03-31', 'John Doe', '08:00:00', '10:00:00'),
(2, '2024-03-31', 'Jane Smith', '09:30:00', '11:30:00'),
(3, '2024-03-31', 'Michael Johnson', '10:00:00', '12:00:00'),
(4, '2024-03-31', 'Emily Davis', '11:30:00', '13:30:00'),
(5, '2024-03-31', 'David Brown', '12:00:00', '14:00:00'),
(6, '2024-03-31', 'Sarah Wilson', '13:30:00', '15:30:00'),
(7, '2024-03-31', 'Robert Martinez', '14:00:00', '16:00:00'),
(8, '2024-03-31', 'Jennifer Anderson', '15:30:00', '17:30:00'),
(9, '2024-03-31', 'Christopher Taylor', '16:00:00', '18:00:00'),
(10, '2024-03-31', 'Amanda Thomas', '17:30:00', '19:30:00');
```

```sql
-- Insert values into Fees table
INSERT INTO Fees (fee_id, student_id, first_name, last_name, mobile_number, email) VALUES
(1, 1, 'Alice', 'Johnson', '1112223333', 'alice@example.com'),
(2, 2, 'Bob', 'Smith', '4445556666', 'bob@example.com'),
(3, 3, 'Charlie', 'Brown', '7778889999', 'charlie@example.com'),
(4, 4, 'Diana', 'Miller', '2223334444', 'diana@example.com'),
(5, 5, 'Eva', 'Wilson', '5556667777', 'eva@example.com'),
(6, 6, 'Frank', 'Taylor', '8889990000', 'frank@example.com'),
(7, 7, 'Grace', 'Anderson', '1234567890', 'grace@example.com'),
(8, 8, 'Harry', 'Davis', '9876543210', 'harry@example.com'),
(9, 9, 'Ivy', 'Martinez', '6667778888', 'ivy@example.com'),
(10, 10, 'Jack', 'White', '9998887777', 'jack@example.com');

SELECT * FROM Administrator;
SELECT * FROM Hostel;
SELECT * FROM Student;
SELECT * FROM Room;
SELECT * FROM Visitors;
SELECT * FROM Fees;
```

OUTPUT:

admin_id	first_name	last_name	mobile_number
1	John	Doe	1234567890
2	Jane	Smith	9876543210
3	Michael	Johnson	5551234567
4	Emily	Davis	9998887776
5	David	Brown	4445556667
6	Sarah	Wilson	7778889990
7	Robert	Martinez	1112223334
8	Jennifer	Anderson	6667778885
9	Christopher	Taylor	2223334445
10	Amanda	Thomas	8889990001

hostel_id	hostel_name	num_students	num_rooms
1	Alpha Hostel	100	50
2	Beta Hostel	120	60
3	Gamma Hostel	80	40
4	Delta Hostel	150	75
5	Epsilon Hostel	90	45
6	Zeta Hostel	110	55
7	Eta Hostel	130	65
8	Theta Hostel	95	48
9	Iota Hostel	105	52
10	Kappa Hostel	140	70

student_id	first_name	last_name	mobile_number	dept	year_of_study
1	Alice	Johnson	1112223333	Computer Science	2
2	Bob	Smith	4445556666	Electrical Engineering	3
3	Charlie	Brown	7778889999	Mechanical Engineering	1
4	Diana	Miller	2223334444	Chemical Engineering	4
5	Eva	Wilson	5556667777	Civil Engineering	2
6	Frank	Taylor	8889990000	Computer Science	3
7	Grace	Anderson	1234567890	Electrical Engineering	1
8	Harry	Davis	9876543210	Mechanical Engineering	4
9	Ivy	Martinez	6667778888	Chemical Engineering	2
10	Jack	White	9998887777	Civil Engineering	3

room_id	floor_number	hostel_id
1	1	1
2	2	1
3	1	2
4	2	2
5	1	3
6	2	3
7	1	4
8	2	4
9	1	5
10	2	5

visitor_id	date	visitor_name	in_time	out_time
1	2024-03-31	John Doe	08:00:00	10:00:00
2	2024-03-31	Jane Smith	09:30:00	11:30:00
3	2024-03-31	Michael Johnson	10:00:00	12:00:00
4	2024-03-31	Emily Davis	11:30:00	13:30:00
5	2024-03-31	David Brown	12:00:00	14:00:00
6	2024-03-31	Sarah Wilson	13:30:00	15:30:00
7	2024-03-31	Robert Martinez	14:00:00	16:00:00
8	2024-03-31	Jennifer Anderson	15:30:00	17:30:00
9	2024-03-31	Christopher Taylor	16:00:00	18:00:00
10	2024-03-31	Amanda Thomas	17:30:00	19:30:00

1.

```
SELECT avg(num_students) from Hostel;
```

OUTPUT:

avg(num_students)
112

2.

```
Select * FROM Student WHERE student_id = 6;
```

OUTPUT:

student_id	first_name	last_name	mobile_number	dept	year_of_study
6	Frank	Taylor	8889990000	Computer Science	3

3.

```
UPDATE Visitors
SET date = '2024-04-1', in_time= '08:00:00'
WHERE visitor_name = 'John Doe';
 Select * from Visitors;
```

OUTPUT:

visitor_id	date	visitor_name	in_time	out_time
1	2024-04-1	John Doe	08:00:00	10:00:00
2	2024-03-31	Jane Smith	09:30:00	11:30:00
3	2024-03-31	Michael Johnson	10:00:00	12:00:00
4	2024-03-31	Emily Davis	11:30:00	13:30:00
5	2024-03-31	David Brown	12:00:00	14:00:00
6	2024-03-31	Sarah Wilson	13:30:00	15:30:00
7	2024-03-31	Robert Martinez	14:00:00	16:00:00
8	2024-03-31	Jennifer Anderson	15:30:00	17:30:00
9	2024-03-31	Christopher Taylor	16:00:00	18:00:00
10	2024-03-31	Amanda Thomas	17:30:00	19:30:00

4. count no. of students in each dept

```
SELECT dept, COUNT(*) AS num_students
FROM Student
GROUP BY dept;
```

OUTPUT:

dept	num_students
Chemical Engineering	2
Civil Engineering	2
Computer Science	2
Electrical Engineering	2
Mechanical Engineering	2

5. left join applied on hostel and room tables to see which hostels have room

```
SELECT h.hostel_name, COUNT(r.room_id) AS num_rooms
FROM Hostel h
LEFT JOIN Room r ON h.hostel_id = r.hostel_id
GROUP BY h.hostel_name;
```

OUTPUT:

hostel_name	num_rooms
Alpha Hostel	2
Beta Hostel	2
Delta Hostel	2
Epsilon Hostel	2
Eta Hostel	0
Gamma Hostel	2
Iota Hostel	0
Kappa Hostel	0
Theta Hostel	0
Zeta Hostel	0

6. inner join between student and fees to get avg fee paid by student

```
SELECT AVG(f.fee_id)
FROM Student s
INNER JOIN Fees f ON s.student_id = f.student_id;
```

OUTPUT:

AVG(f.fee_id)
5.5

7. total no. of visitors for each date

```
SELECT date, COUNT(visitor_id) AS total_visitors
FROM Visitors
GROUP BY date;
```

OUTPUT:

date	total_visitors
2024-03-31	9
2024-04-1	1

8.

```
Delete from Administrator where admin_id = 7;
```

OUTPUT:

admin_id	first_name	last_name	mobile_number
1	John	Doe	1234567890
2	Jane	Smith	9876543210
3	Michael	Johnson	5551234567
4	Emily	Davis	9998887776
5	David	Brown	4445556667
6	Sarah	Wilson	7778889990
8	Jennifer	Anderson	6667778885
9	Christopher	Taylor	2223334445
10	Amanda	Thomas	8889990001

9. Getting administrator whose first name starts with “J”

```
SELECT * FROM Administrator WHERE first_name LIKE 'J%';
```

OUTPUT:

admin_id	first_name	last_name	mobile_number
1	John	Doe	1234567890
2	Jane	Smith	9876543210
8	Jennifer	Anderson	6667778885

10.

```
-- Calculate the average number of students per year of study using student_id
SELECT year_of_study, AVG(student_id) AS avg_students
FROM Student
GROUP BY year_of_study;
```

10.

OUTPUT:

year_of_study	avg_students
1	5
2	5
3	6
4	6

11.

```
-- Calculate the average number of rooms per hostel
SELECT hostel_id, hostel_name, AVG(num_rooms) AS avg_rooms
FROM Hostel
GROUP BY hostel_id, hostel_name;
```

OUTPUT:

hostel_id	hostel_name	avg_rooms
1	Alpha Hostel	50
2	Beta Hostel	60
3	Gamma Hostel	40
4	Delta Hostel	75
5	Epsilon Hostel	45
6	Zeta Hostel	55
7	Eta Hostel	65
8	Theta Hostel	48
9	Iota Hostel	52
10	Kappa Hostel	70

12. query returning administrators as students by finding common records between administrator and student tables

```
SELECT first_name, last_name, mobile_number
FROM Administrator
INTERSECT
SELECT first_name, last_name, mobile_number
FROM Student;
```

OUTPUT:

```
SQL query successfully executed. However, the result set is empty.
```

13.

```
alter table Student drop column dept;
select * from Student;
```

OUTPUT:

student_id	first_name	last_name	mobile_number	year_of_study
1	Alice	Johnson	1112223333	2
2	Bob	Smith	4445556666	3
3	Charlie	Brown	7778889999	1
4	Diana	Miller	2223334444	4
5	Eva	Wilson	5556667777	2
6	Frank	Taylor	8889990000	3
7	Grace	Anderson	1234567890	1
8	Harry	Davis	9876543210	4
9	Ivy	Martinez	6667778888	2
10	Jack	White	9998887777	3

14. Decompose the Table Room as:

```sql
CREATE TABLE Room_Details (
    room_id INT PRIMARY KEY,
    floor_number INT,
    FOREIGN KEY (room_id) REFERENCES Room(room_id)
);

SELECT FROM Room_Details;
```

```sql
CREATE TABLE Room1 (
    room_id INT PRIMARY KEY,
    hostel_id INT,
    FOREIGN KEY (hostel_id) REFERENCES Hostel(hostel_id)
);

SELECT * FROM Room;
```

OUTPUT:

Room_Details

room_id	floor_number
empty	

Room1

room_id	hostel_id
empty	

15. if fees is increased by 10% then:

```sql
ALTER TABLE Fees
ADD Fee_payment DECIMAL(10, 2);

UPDATE Fees
SET Fee_payment = 100.00;

SELECT * FROM Fees;
```

OUTPUT:

fee_id	student_id	first_name	last_name	mobile_number	email	Fee_payment
1	1	Alice	Johnson	1112223333	alice@example.com	100
2	2	Bob	Smith	4445556666	bob@example.com	100
3	3	Charlie	Brown	7778889999	charlie@example.com	100
4	4	Diana	Miller	2223334444	diana@example.com	100
5	5	Eva	Wilson	5556667777	eva@example.com	100
6	6	Frank	Taylor	8889990000	frank@example.com	100
7	7	Grace	Anderson	1234567890	grace@example.com	100
8	8	Harry	Davis	9876543210	harry@example.com	100
9	9	Ivy	Martinez	6667778888	ivy@example.com	100
10	10	Jack	White	9998887777	jack@example.com	100

INPUT FOR MORE 10%:

```
UPDATE Fees
SET Fee_payment = Fee_payment * 1.1; -- Increasing fees by 10%

SELECT * FROM Fees;
```

FINAL OUTPUT:

fee_id	student_id	first_name	last_name	mobile_number	email	Fee_payment
1	1	Alice	Johnson	1112223333	alice@example.com	110.00000000000001
2	2	Bob	Smith	4445556666	bob@example.com	110.00000000000001
3	3	Charlie	Brown	7778889999	charlie@example.com	110.00000000000001
4	4	Diana	Miller	2223334444	diana@example.com	110.00000000000001
5	5	Eva	Wilson	5556667777	eva@example.com	110.00000000000001
6	6	Frank	Taylor	8889990000	frank@example.com	110.00000000000001
7	7	Grace	Anderson	1234567890	grace@example.com	110.00000000000001
8	8	Harry	Davis	9876543210	harry@example.com	110.00000000000001
9	9	Ivy	Martinez	6667778888	ivy@example.com	110.00000000000001
10	10	Jack	White	9998887777	jack@example.com	110.00000000000001

16. student on fee_id = 6 has not paid 40 rupees, update the query on fees table:

```
UPDATE Fees
SET Fee_payment = 150
WHERE student_id = 6;

SELECT * FROM Fees;
```

OUTPUT:

fee_id	student_id	first_name	last_name	mobile_number	email	Fee_payment
1	1	Alice	Johnson	1112223333	alice@example.com	110.00000000000001
2	2	Bob	Smith	4445556666	bob@example.com	110.00000000000001
3	3	Charlie	Brown	7778889999	charlie@example.com	110.00000000000001
4	4	Diana	Miller	2223334444	diana@example.com	110.00000000000001
5	5	Eva	Wilson	5556667777	eva@example.com	110.00000000000001
6	6	Frank	Taylor	8889990000	frank@example.com	150
7	7	Grace	Anderson	1234567890	grace@example.com	110.00000000000001
8	8	Harry	Davis	9876543210	harry@example.com	110.00000000000001
9	9	Ivy	Martinez	6667778888	ivy@example.com	110.00000000000001
10	10	Jack	White	9998887777	jack@example.com	110.00000000000001

17. Find minimum fees:

```
SELECT MIN(Fee_payment) AS min_fee_payment
FROM Fees;
```

OUTPUT:

min_fee_payment
110.00000000000001

18. Find maximum fees:

```
SELECT MAX(Fee_payment) AS max_fee_payment
FROM Fees;
```

OUTPUT:

max_fee_payment
150

19. Find maximum number and minimum number of students in hostels:

```
SELECT MIN(num_students) AS min_students, MAX(num_students) AS max_students
FROM Hostel;
```

OUTPUT:

min_students	max_students
80	150

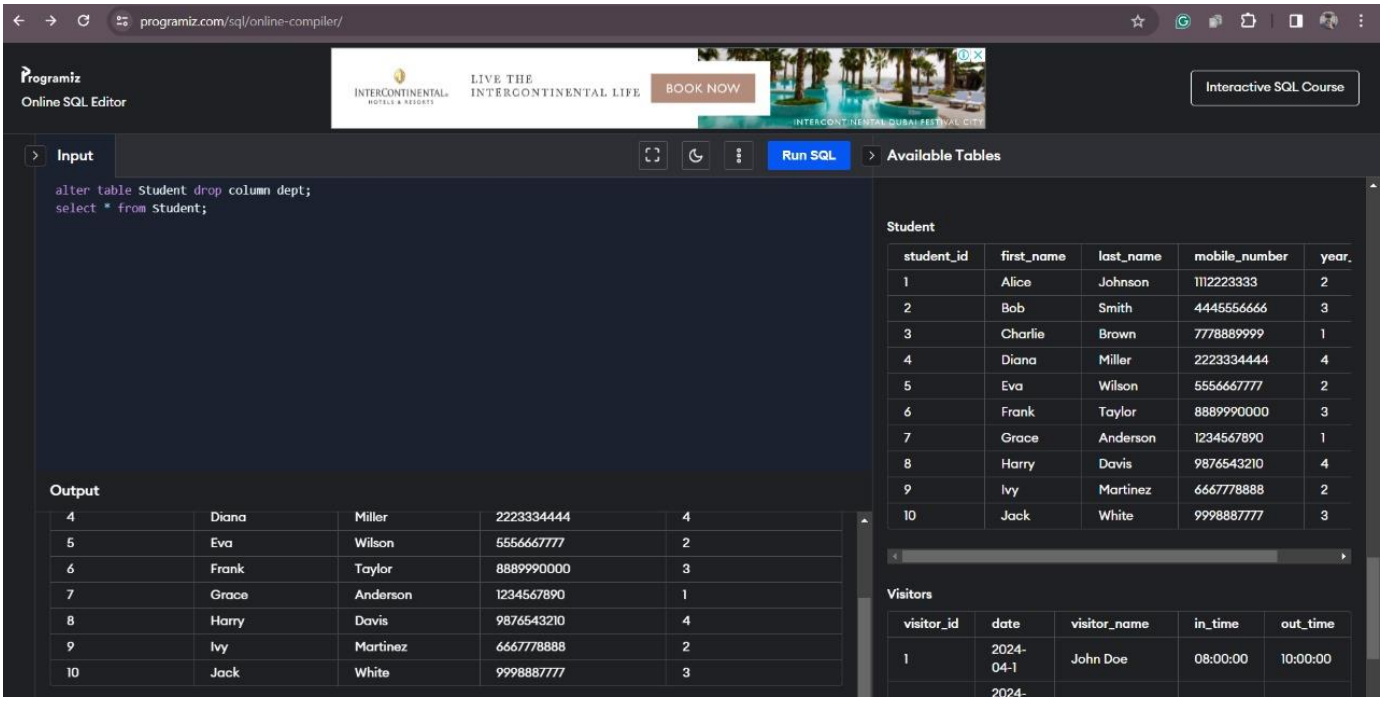
20. which hostel_id contains floor number 1:

```
SELECT * FROM Room
WHERE floor_number LIKE '1%';
```

OUTPUT:

room_id	floor_number	hostel_id
1	1	1
3	1	2
5	1	3
7	1	4
9	1	5

VI. Project demonstration



- **Software used is Programiz editor for sql**
- **It is an open source relational database management system(RDBMS).**
- **It helps to design, develop and manage MySQL databases.**
- **It uses structured query language to manage and manipulate data stored in tables.**

VII. Self -Learning beyond classroom

So, in this beyond the class I learned about Advanced Query Optimization Techniques which teaches us about learning that covers basic query optimization strategies, individuals can delve deeper into advanced

optimization techniques such as indexing strategies, query execution plans, and performance tuning for specific DBMS engines like MySQL, PostgreSQL, Oracle, or SQL Server

And about the Cloud Database Services which talks about the cloud-based DBMS platforms (e.g., Amazon RDS, Google Cloud SQL, Azure SQL Database) can be valuable, including understanding cloud-specific features, scalability options, data backup and recovery strategies, and cost optimization techniques.
And also, about to show a ER Diagram in a proper manner.

VIII. Learning from the Project

When working on a project like a hostel room allocation system in the field of database management systems (DBMS), there are several key learnings which includes.

About data Modeling in this I have learned how to designing the database schema for the hostel room allocation system involves learning about entity-relationship modeling (ER modeling), identifying entities such as students, rooms, allocations, and relationships between them (e.g., a student can be allocated to a room). Understanding concepts like cardinality, normalization, and denormalization is crucial for creating an efficient and scalable database structure.

About what are the database design principles I this when I was applying database design principles such as choosing appropriate data types, defining primary keys, foreign keys, and constraints, and optimizing the database schema for performance and data integrity are essential skills. I Learned about indexing strategies, partitioning, and data organization techniques can also improve the efficiency of queries and data retrieval operations.

While doing the code part I learned about performance optimization which monitor’s the database performance metrics, identifying bottlenecks, optimizing SQL queries and database configurations, caching strategies, and scalability planning are important aspects of ensuring the hostel room allocation system operates efficiently and can handle increasing data loads. Which I learned about this while doing this project.

IX. Challenges Faced

While doing this project I have faced some challenges which are complex data relationships while managing such complex relationships between entities such as students, rooms, allocations, and hostel facilities can be challenging. Designing an efficient database schema that properly represents these relationships while ensuring data integrity and normalization requires careful planning.

In terms of error handling I had to make sure that ensuring the data quality and reliability by implementing robust data validation rules, error handling mechanisms, exception logging, and rollback procedures in case of data entry errors or system failures is critical for a reliable hostel room allocation system.
While doing the coding part some codes were not running properly. And the use of 10 tuples made the code's execution more complex.

X. Conclusion

In conclusion, throughout the project, the project allowed me for deep exploration and application of database design principles, query optimization techniques, transaction management strategies, and security measures. It also provided valuable insights into user interface integration, data validation, error handling, and scalability considerations.
From this project I learned about how transaction management skills with dealing with concurrency issues, implementing transaction control mechanisms, and ensuring ACID properties highlighted the importance of effective transaction management for maintaining data consistency and reliability.

Overall, the project provided valuable hands-on experience and practical skills in database management, software development, and system integration, highlighting the importance of technical expertise, problem-solving abilities, and attention to detail in developing robust and reliable database-driven applications.