CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

# UNIVERSITY INSTITUTE OF COMPUTING

# CASE STUDY REPORT
# ON
# HOSTEL MANAGEMENT SYSTEM

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

| **Submitted by:** | **Submitted to:** |
|---|---|
| Name: Sanjeev Singh | Name: Arvinder Singh |
| UID: 23BCA10354 | Designation: Assist. Prof. |
| Section: 4(B) | |

# Table of Contents

https://github.com/sanjeev711/hostel.git

# ABSTRACT

- **Introduction:**

The Hostel Management System is a database-driven solution aimed at organizing and managing various operations in a hostel setup, such as student data management, room allocation, fee payment tracking, and complaint registration. This project helps simplify administrative tasks by maintaining relational data and allowing structured query execution.

- **Technique:**

The project uses MySQL to implement the relational database model. It involves creating normalized tables, establishing primary and foreign key relationships, enforcing mapping constraints, and using SQL queries for data manipulation and retrieval.

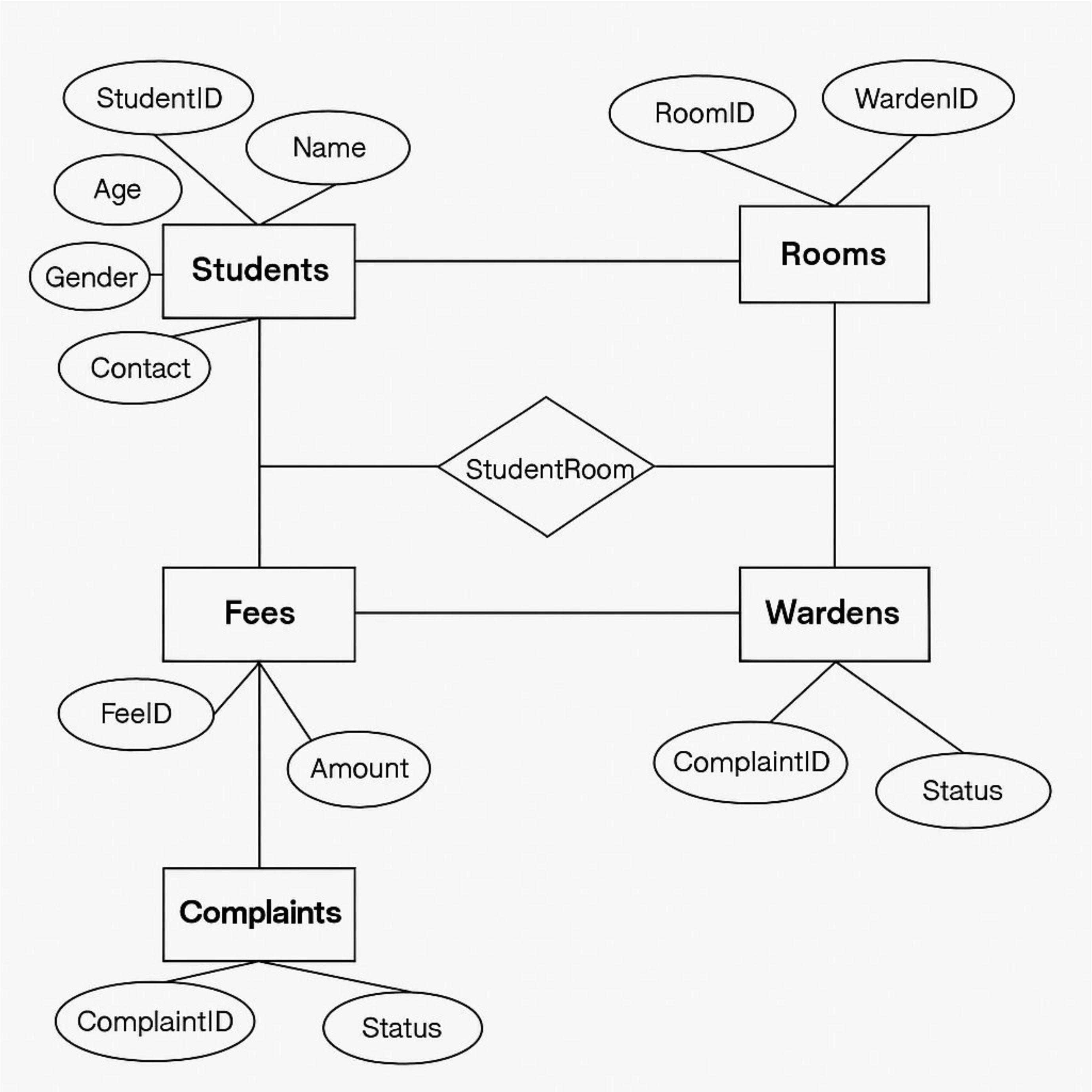
- **System Configuration:**

  - **Database**: MySQL 8.0
  - **Interface**: MySQL Workbench
  - **Operating System**: Windows 10 or higher
  - **RAM**: Minimum 4 GB
  - **Storage**: Minimum 1 GB free space for DBMS and project files
  - **Diagram Tool**: Draw.io (for ER diagrams)

- **INPUT:**

Input includes student personal details, room configurations, warden contact details, room allocations, fee information, and student complaints. Data is inserted manually using SQL INSERT statements.

- **ER DIAGRAM:**


StudentID
Name
Age
Gender
Contact
Students
RoomID
WardenID
Rooms
StudentRoom
Fees
Wardens
FeeID
Amount
ComplaintID
Status
Complaints
ComplaintID
Status

## • ER DIAGRAM DESCRIPTION

The ER Diagram consists of the following entities:

- Student (StudentID, Name, Age, Gender, Contact)

- Room (RoomID, RoomType, Capacity, Occupied)

- Warden (WardenID, WardenName, Contact)

- Fee (FeeID, StudentID, Amount, Status)

- Complaint (ComplaintID, StudentID, Description, Status)

- StudentRoom (Mapping Table)

### Relationships:

- One-to-Many: Student to Fee, Student to Complaint

- Many-to-Many: Student to Room via StudentRoom

## • TABLE RELATIONSHIPS:

The Hostel Management System includes the following table relationships:

1. **One-to-Many:**
   - Student → Fees
   - Student → Complaints
2. **Many-to-Many:**
   - Student ↔ Room (using StudentRoom mapping table)
3. **One-to-One:**
   - Warden → Room (Each room is managed by one warden)

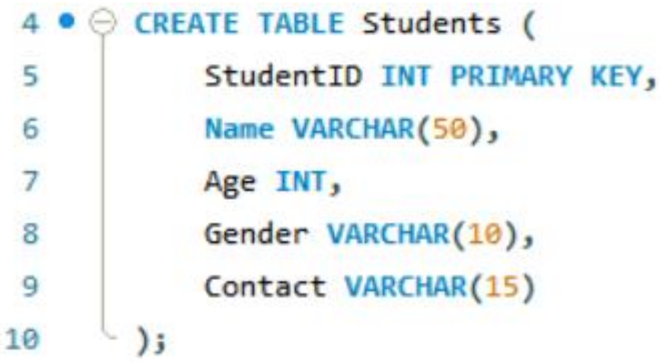
Foreign keys are used to enforce these relationships and maintain referential integrity.

- **TABULAR FORMAT**

| **Table Name** | **Description** |
|---|---|
| **Students** | Stores student personal details |
| **Rooms** | Room type, capacity, occupancy |
| **Wardens** | Details of hostel wardens |
| **StudentRoom** | Mapping table for room allocation |
| **Fees** | Tracks student fee status |
| **Complaints** | Stores complaints lodged by students |

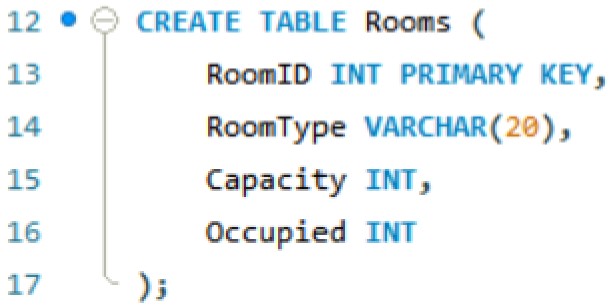
- **TABLE CREATION**

### TABLE STUDENTS



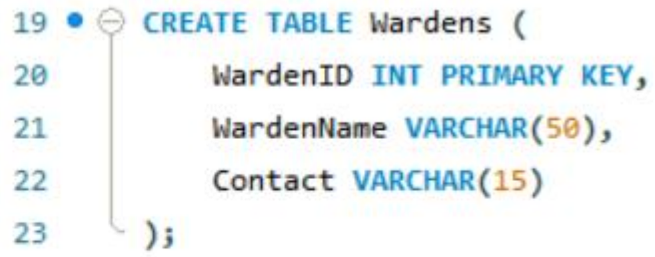
```
CREATE TABLE Students (
      StudentID INT PRIMARY KEY,
      Name VARCHAR(50),
      Age INT,
      Gender VARCHAR(10),
      Contact VARCHAR(15)
);
```

### TABLE ROOMS



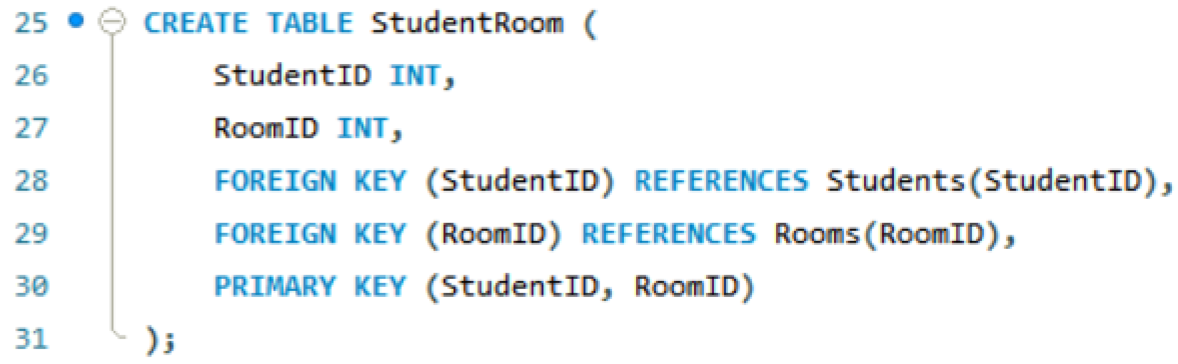
```
CREATE TABLE Rooms (
      RoomID INT PRIMARY KEY,
      RoomType VARCHAR(20),
      Capacity INT,
      Occupied INT
);
```

### TABLE WARDENS



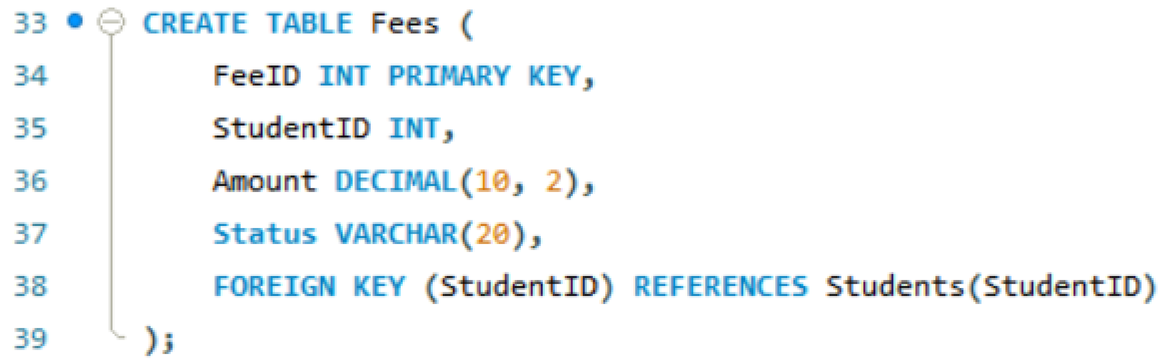
```sql
CREATE TABLE Wardens (
    WardenID INT PRIMARY KEY,
    WardenName VARCHAR(50),
    Contact VARCHAR(15)
);
```

## TABLE STUDENTSROOM



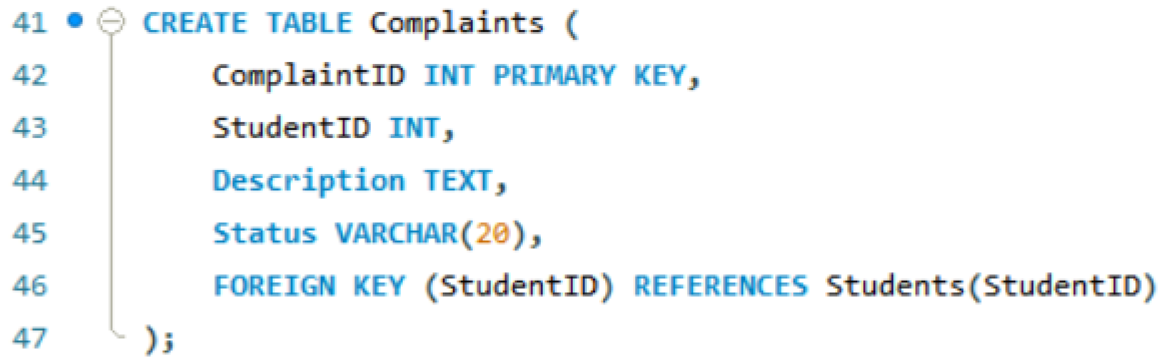
```sql
CREATE TABLE StudentRoom (
    StudentID INT,
    RoomID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID),
    PRIMARY KEY (StudentID, RoomID)
);
```

## TABLE FEES



```sql
CREATE TABLE Fees (
    FeeID INT PRIMARY KEY,
    StudentID INT,
    Amount DECIMAL(10, 2),
    Status VARCHAR(20),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

## TABLE COMPLAINTS



```sql
CREATE TABLE Complaints (
    ComplaintID INT PRIMARY KEY,
    StudentID INT,
    Description TEXT,
    Status VARCHAR(20),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

- ## TABLE REALTION:

In the Hostel Management System, the relationships between tables are critical for maintaining data integrity and ensuring correct mapping of entities. Below are the primary relationships:

1. **Student to Room**:
   - Many-to-Many (via the **StudentRoom** table).
   - A student can be assigned multiple rooms, and a room can accommodate multiple students.
2. **Student to Fee**:
   - One-to-Many (One student can have multiple fee records, but each fee record is associated with only one student).
3. **Student to Complaint**:
   - One-to-Many (A student can file multiple complaints, but each complaint is associated with only one student).
4. **Warden to Room**:
   - One-to-One (Each room is managed by a single warden).

- ## TABULAR FORMAT:

| Table Name | Description | Relationship |
|---|---|---|
| **Students** | Stores student personal details | - |
| **Rooms** | Stores details about rooms (type, capacity) | - |
| **Wardens** | Stores warden details | - |
| **StudentRoom** | Mapping table for students and rooms | Many-to-Many (Student ↔ Room) |
| **Fees** | Stores student fee information | One-to-Many (Student → Fees) |
| **Complaints** | Stores complaints lodged by students | One-to-Many (Student → Complaints) |

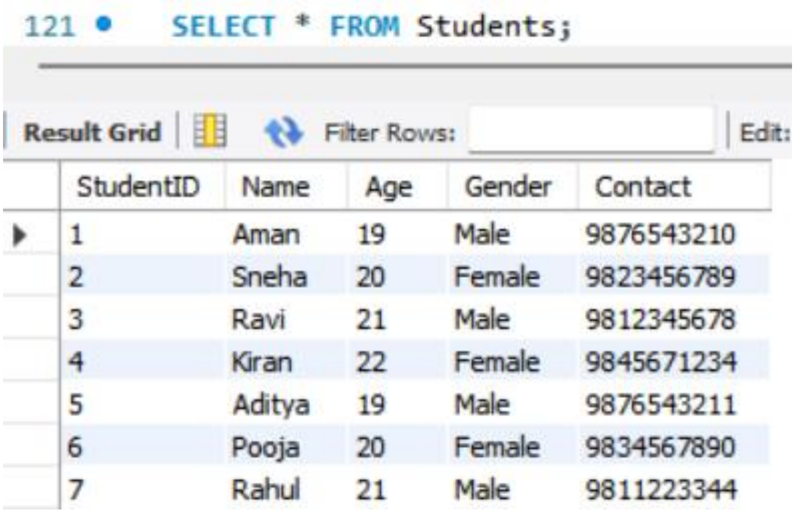
- **SQL IMPLEMENTATION Code:**

```
CREATE DATABASE HostelManagement;
USE HostelManagement;

CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR (50),
   Age INT,
   Gender VARCHAR (10),
   Contact VARCHAR (15)
);

CREATE TABLE Rooms (
   RoomID INT PRIMARY KEY,
   RoomType VARCHAR (20),
   Capacity INT,
   Occupied INT
);

CREATE TABLE Wardens (
   WardenID INT PRIMARY KEY,
   WardenName VARCHAR (50),
   Contact VARCHAR (15)
);

CREATE TABLE StudentRoom (
   StudentID INT,
   RoomID INT,
   FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
   FOREIGN KEY (RoomID) REFERENCES Rooms (RoomID),
   PRIMARY KEY (StudentID, RoomID)
);

CREATE TABLE Fees (
   FeeID INT PRIMARY KEY,
    StudentID INT,
    Amount DECIMAL (10, 2),
    Status VARCHAR (20),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
);

CREATE TABLE Complaints (
    ComplaintID INT PRIMARY KEY,
    StudentID INT,
    Description TEXT,
    Status VARCHAR (20),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
);

INSERT INTO Students VALUES
(1, 'Aman', 19, 'Male', '9876543210'),
(2, 'Sneha', 20, 'Female', '9823456789'),
(3, 'Ravi', 21, 'Male', '9812345678'),
(4, 'Kiran', 22, 'Female', '9845671234'),
(5, 'Aditya', 19, 'Male', '9876543211'),
(6, 'Pooja', 20, 'Female', '9834567890'),
(7, 'Rahul', 21, 'Male', '9811223344'),
(8, 'Simran', 22, 'Female', '9800112233'),
(9, 'Vikram', 20, 'Male', '9876611223'),
(10, 'Neha', 21, 'Female', '9822334455');

INSERT INTO Rooms VALUES
(101, 'Single', 1, 1),
(102, 'Double', 2, 2),
(103, 'Triple', 3, 2),
(104, 'Single', 1, 1),
(105, 'Double', 2, 1),
(106, 'Single', 1, 1),
(107, 'Double', 2, 2),
(108, 'Triple', 3, 3),
(109, 'Single', 1, 0),
(110, 'Double', 2, 0);
```

```
INSERT INTO Wardens VALUES
(1, 'Mr. Sharma', '9990011223'),
(2, 'Mrs. Patel', '9990022334'),
(3, 'Mr. Verma', '9990033445'),
(4, 'Ms. Joshi', '9990044556'),
(5, 'Mr. Khan', '9990055667'),
(6, 'Mrs. Rao', '9990066778'),
(7, 'Mr. Reddy', '9990077889'),
(8, 'Ms. Das', '9990088990'),
(9, 'Mr. Singh', '9990099001'),
(10, 'Mrs. Iyer', '9990101011');

INSERT INTO StudentRoom VALUES
(1, 101),
(2, 102),
(3, 102),
(4, 103),
(5, 103),
(6, 104),
(7, 105),
(8, 106),
(9, 107),
(10, 108);

INSERT INTO Fees VALUES
(1, 1, 10000, 'Paid'),
(2, 2, 9500, 'Unpaid'),
(3, 3, 10500, 'Paid'),
(4, 4, 10000, 'Paid'),
(5, 5, 9800, 'Unpaid'),
(6, 6, 11000, 'Paid'),
(7, 7, 10000, 'Paid'),
(8, 8, 10200, 'Unpaid'),
(9, 9, 9700, 'Paid'),
(10, 10, 10100, 'Paid');
```

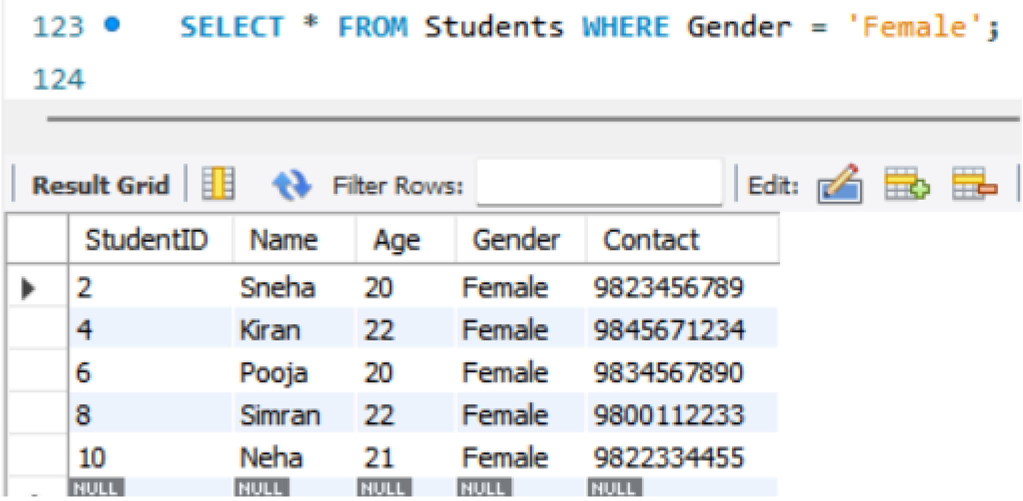
```
INSERT INTO Complaints VALUES
(1, 1, 'Leaky tap in bathroom', 'Resolved'),
(2, 2, 'Broken fan', 'Pending'),
(3, 3, 'No water supply', 'Resolved'),
(4, 4, 'Dirty room', 'Pending'),
(5, 5, 'Wi-Fi not working', 'Pending'),
(6, 6, 'Electric socket issue', 'Resolved'),
(7, 7, 'AC not cooling', 'Pending'),
(8, 8, 'Mosquitoes in room', 'Resolved'),
(9, 9, 'Door lock broken', 'Pending'),
(10, 10, 'No electricity', 'Resolved');

SELECT * FROM Students;

SELECT * FROM Students WHERE Gender = 'Female';

SELECT * FROM Students WHERE Age > 20;

SELECT * FROM Rooms WHERE Capacity > 2;

SELECT * FROM Fees WHERE Status = 'Unpaid';

SELECT s.Name, r.RoomType
FROM Students s
JOIN StudentRoom sr ON s.StudentID = sr.StudentID
JOIN Rooms r ON sr.RoomID = r.RoomID;

SELECT s.Name, f.Amount, f.Status
FROM Students s
JOIN Fees f ON s.StudentID = f.StudentID;

SELECT s.Name, c.Description, c.Status
FROM Students s
JOIN Complaints c ON s.StudentID = c.StudentID;

SELECT COUNT (*) FROM Students;
SELECT AVG(Amount) AS Avg_Fee FROM Fees;

SELECT COUNT (*) FROM Complaints WHERE Status = 'Pending';

SELECT SUM(Occupied) AS Total_Occupied FROM Rooms;

SELECT Status, COUNT (*) FROM Complaints GROUP BY Status;

SELECT Status, COUNT (*) FROM Fees GROUP BY Status;

SELECT Name FROM Students
WHERE StudentID IN (
  SELECT StudentID FROM Fees WHERE Amount > (SELECT
AVG(Amount) FROM Fees)
);

SELECT * FROM Rooms WHERE Capacity > Occupied;

SELECT s.Name, f.Amount
FROM Fees f
JOIN Students s ON s.StudentID = f.StudentID
ORDER BY f.Amount DESC LIMIT 5;

SELECT * FROM Complaints ORDER BY ComplaintID DESC LIMIT 3;

SELECT StudentID, RoomID FROM StudentRoom;

SELECT s.Name, f.Status FROM Students s
LEFT JOIN Fees f ON s.StudentID = f.StudentID;
```

- **SQL QUERIES WITH OUTPUT:**



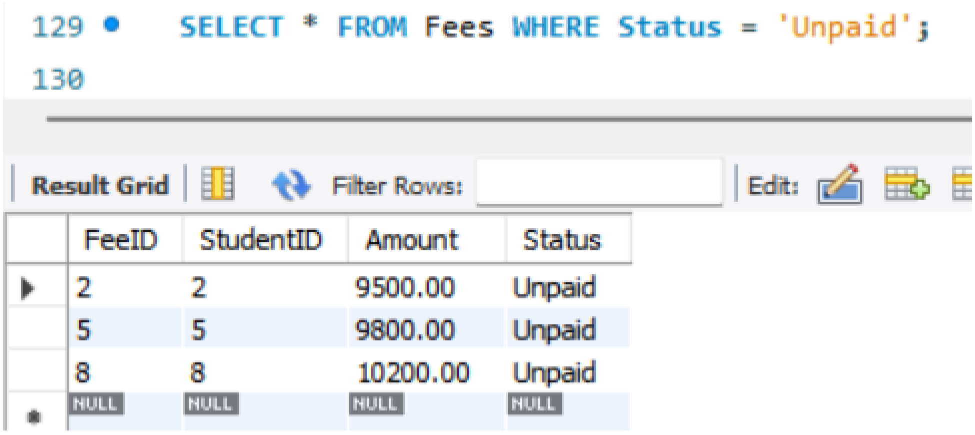
```sql
SELECT * FROM Students;
```

| StudentID | Name | Age | Gender | Contact |
|---|---|---|---|---|
| 1 | Aman | 19 | Male | 9876543210 |
| 2 | Sneha | 20 | Female | 9823456789 |
| 3 | Ravi | 21 | Male | 9812345678 |
| 4 | Kiran | 22 | Female | 9845671234 |
| 5 | Aditya | 19 | Male | 9876543211 |
| 6 | Pooja | 20 | Female | 9834567890 |
| 7 | Rahul | 21 | Male | 9811223344 |



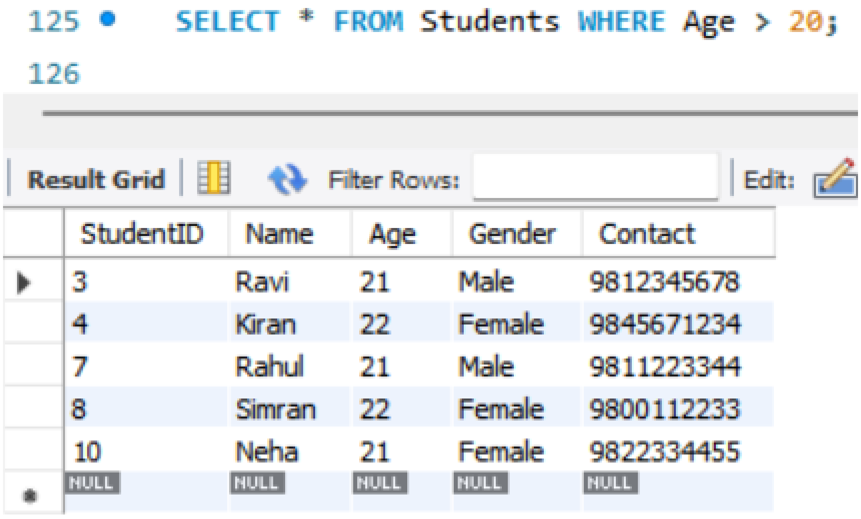
```sql
SELECT * FROM Students WHERE Gender = 'Female';
```

| StudentID | Name | Age | Gender | Contact |
|---|---|---|---|---|
| 2 | Sneha | 20 | Female | 9823456789 |
| 4 | Kiran | 22 | Female | 9845671234 |
| 6 | Pooja | 20 | Female | 9834567890 |
| 8 | Simran | 22 | Female | 9800112233 |
| 10 | Neha | 21 | Female | 9822334455 |
| NULL | NULL | NULL | NULL | NULL |



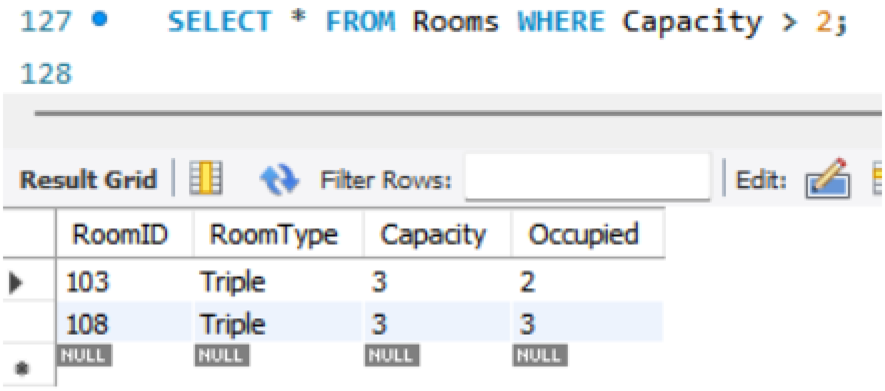
```sql
SELECT * FROM Fees WHERE Status = 'Unpaid';
```

| FeeID | StudentID | Amount | Status |
|---|---|---|---|
| 2 | 2 | 9500.00 | Unpaid |
| 5 | 5 | 9800.00 | Unpaid |
| 8 | 8 | 10200.00 | Unpaid |
| NULL | NULL | NULL | NULL |



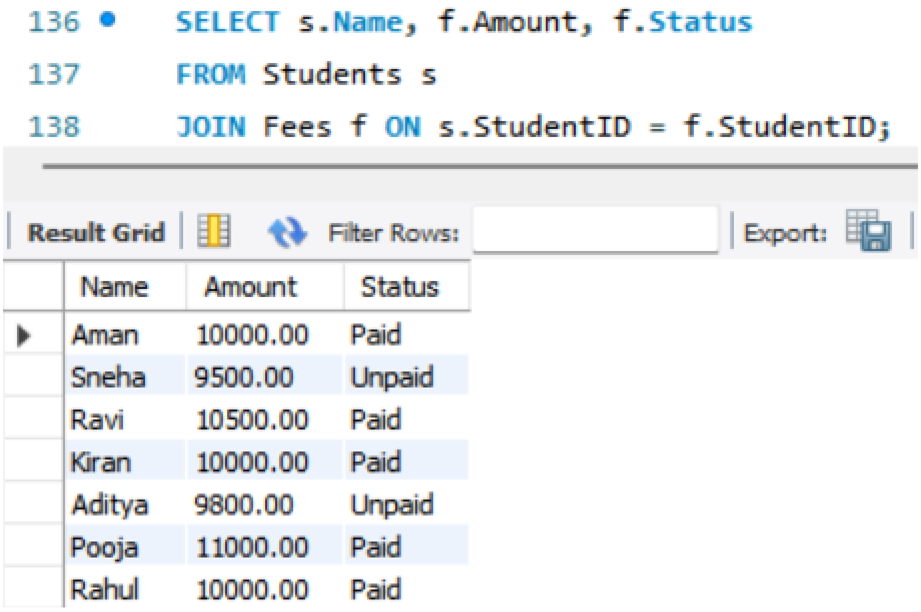
```sql
SELECT * FROM Students WHERE Age > 20;
```

| StudentID | Name | Age | Gender | Contact |
|---|---|---|---|---|
| 3 | Ravi | 21 | Male | 9812345678 |
| 4 | Kiran | 22 | Female | 9845671234 |
| 7 | Rahul | 21 | Male | 9811223344 |
| 8 | Simran | 22 | Female | 9800112233 |
| 10 | Neha | 21 | Female | 9822334455 |
| NULL | NULL | NULL | NULL | NULL |



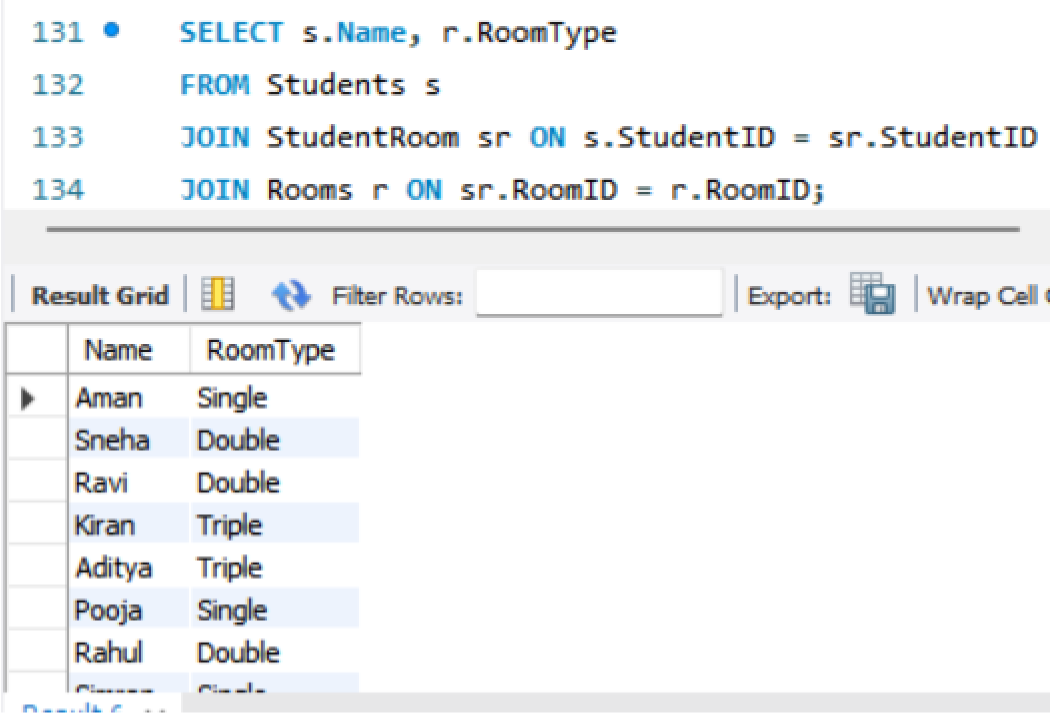
```sql
SELECT * FROM Rooms WHERE Capacity > 2;
```

| RoomID | RoomType | Capacity | Occupied |
|---|---|---|---|
| 103 | Triple | 3 | 2 |
| 108 | Triple | 3 | 3 |
| NULL | NULL | NULL | NULL |



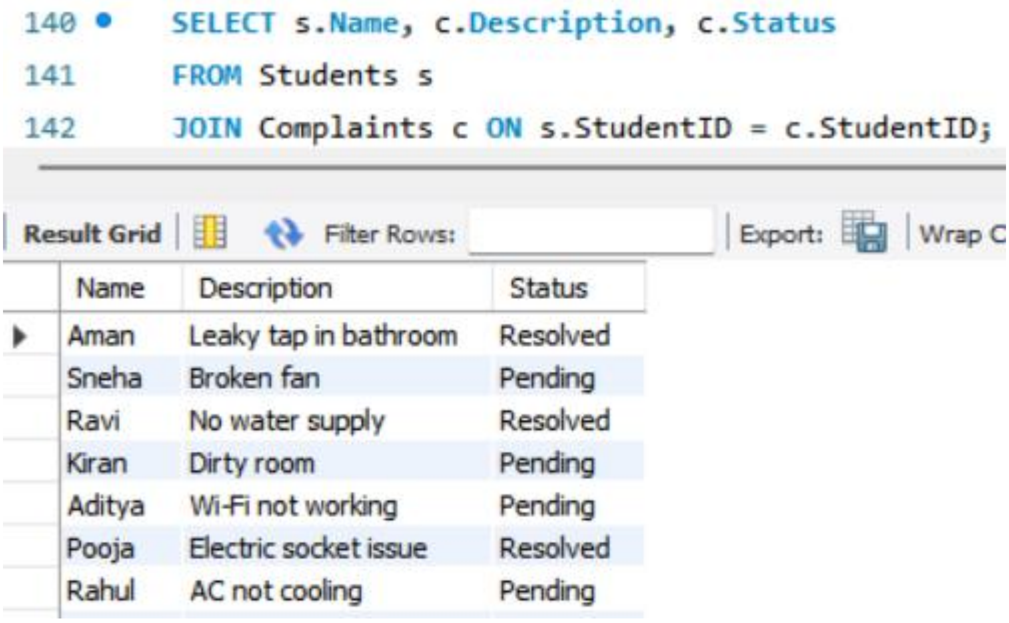
```sql
SELECT s.Name, f.Amount, f.Status
FROM Students s
JOIN Fees f ON s.StudentID = f.StudentID;
```

| Name | Amount | Status |
|---|---|---|
| Aman | 10000.00 | Paid |
| Sneha | 9500.00 | Unpaid |
| Ravi | 10500.00 | Paid |
| Kiran | 10000.00 | Paid |
| Aditya | 9800.00 | Unpaid |
| Pooja | 11000.00 | Paid |
| Rahul | 10000.00 | Paid |



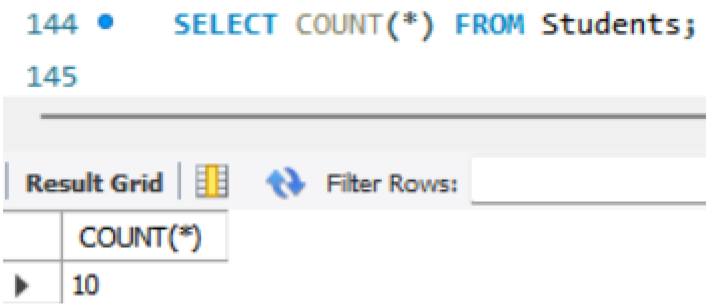
```
SELECT s.Name, r.RoomType
FROM Students s
JOIN StudentRoom sr ON s.StudentID = sr.StudentID
JOIN Rooms r ON sr.RoomID = r.RoomID;
```

| Name | RoomType |
|---|---|
| Aman | Single |
| Sneha | Double |
| Ravi | Double |
| Kiran | Triple |
| Aditya | Triple |
| Pooja | Single |
| Rahul | Double |
| Simran | Single |



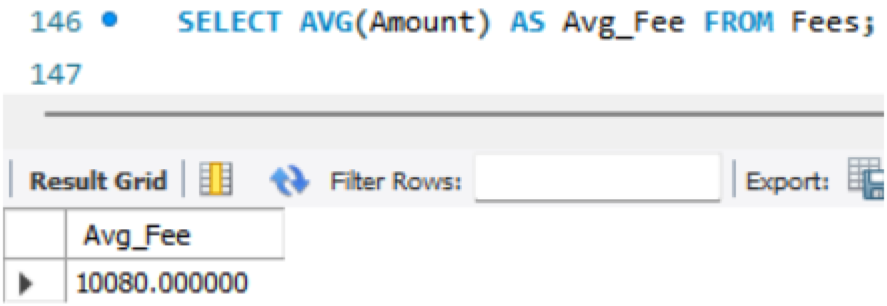
```
SELECT s.Name, c.Description, c.Status
FROM Students s
JOIN Complaints c ON s.StudentID = c.StudentID;
```

| Name | Description | Status |
|---|---|---|
| Aman | Leaky tap in bathroom | Resolved |
| Sneha | Broken fan | Pending |
| Ravi | No water supply | Resolved |
| Kiran | Dirty room | Pending |
| Aditya | Wi-Fi not working | Pending |
| Pooja | Electric socket issue | Resolved |
| Rahul | AC not cooling | Pending |



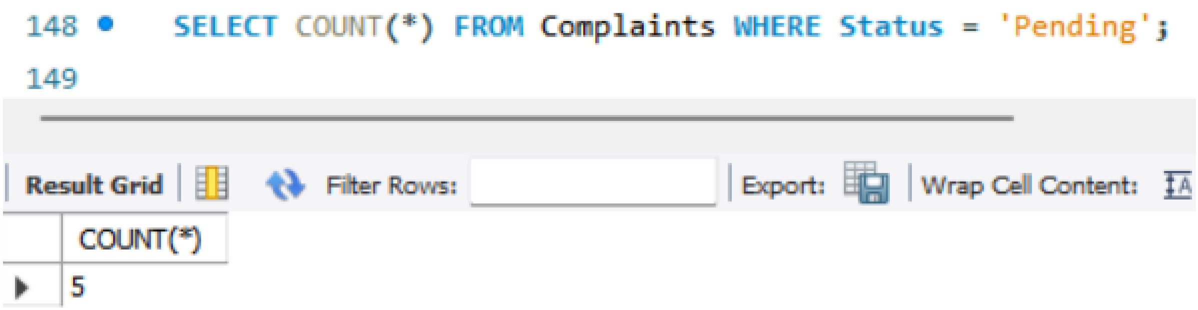
```sql
SELECT COUNT(*) FROM Students;
```

```
COUNT(*)
10
```



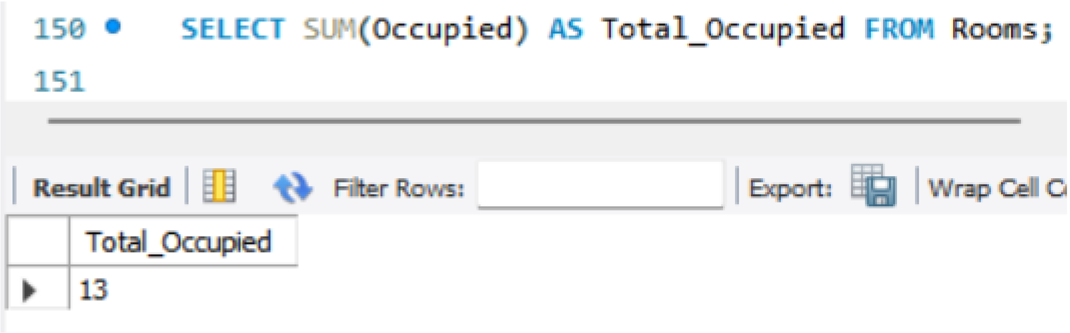
```sql
SELECT AVG(Amount) AS Avg_Fee FROM Fees;
```

```
Avg_Fee
10080.000000
```



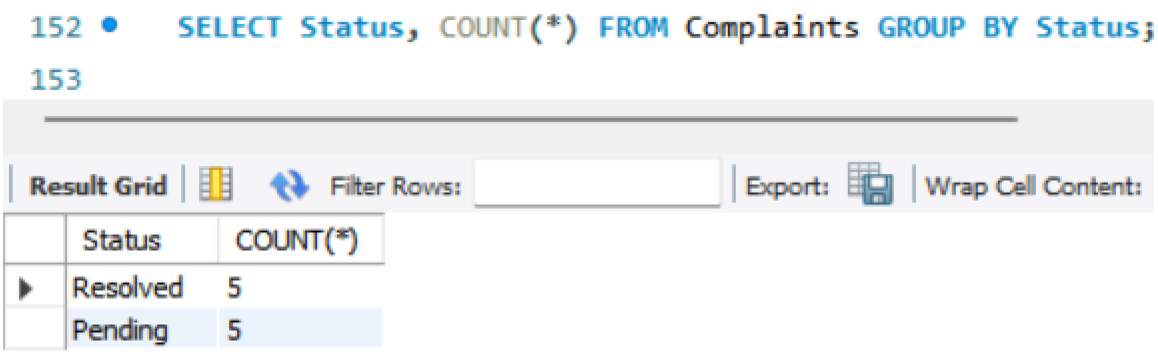
```sql
SELECT COUNT(*) FROM Complaints WHERE Status = 'Pending';
```

```
COUNT(*)
5
```



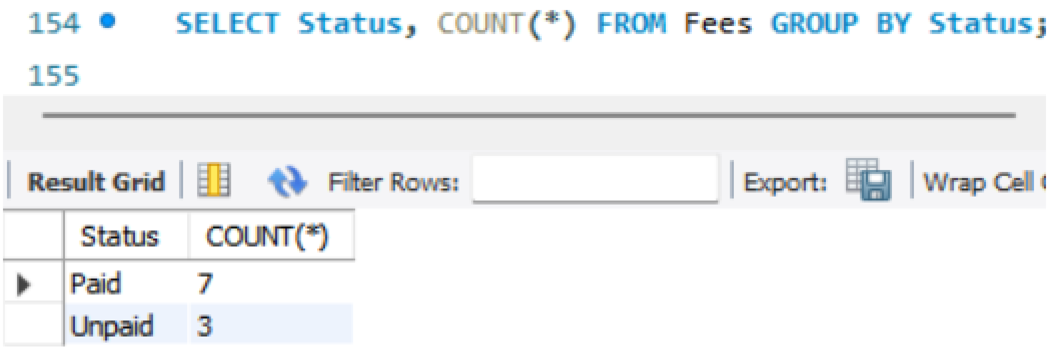
```sql
SELECT SUM(Occupied) AS Total_Occupied FROM Rooms;
```

```
Total_Occupied
13
```



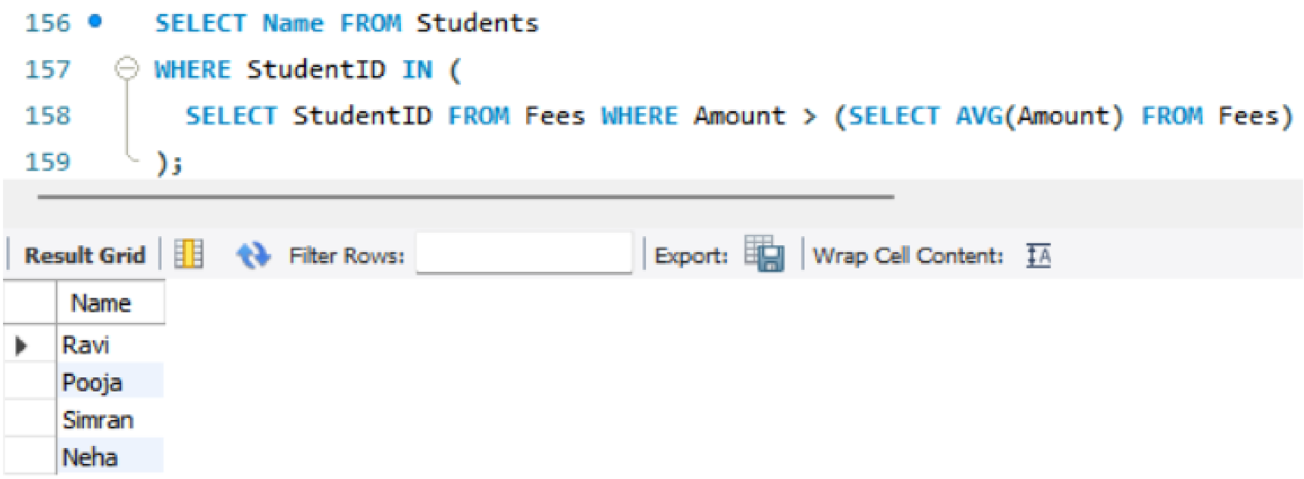
```sql
SELECT Status, COUNT(*) FROM Complaints GROUP BY Status;
```

| Status | COUNT(*) |
|---|---|
| Resolved | 5 |
| Pending | 5 |



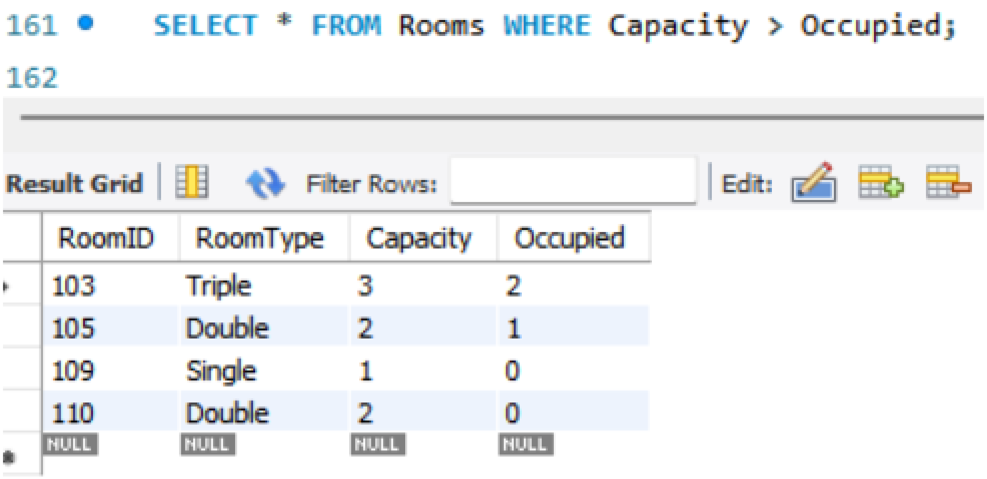
```sql
SELECT Status, COUNT(*) FROM Fees GROUP BY Status;
```

| Status | COUNT(*) |
|---|---|
| Paid | 7 |
| Unpaid | 3 |



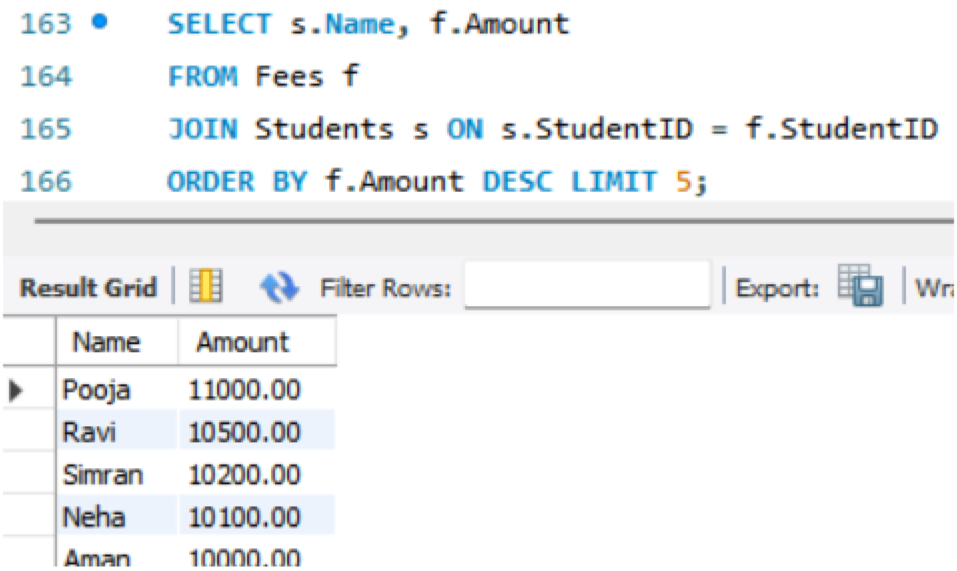
```sql
SELECT Name FROM Students
WHERE StudentID IN (
    SELECT StudentID FROM Fees WHERE Amount > (SELECT AVG(Amount) FROM Fees)
);
```

| Name |
|---|
| Ravi |
| Pooja |
| Simran |
| Neha |



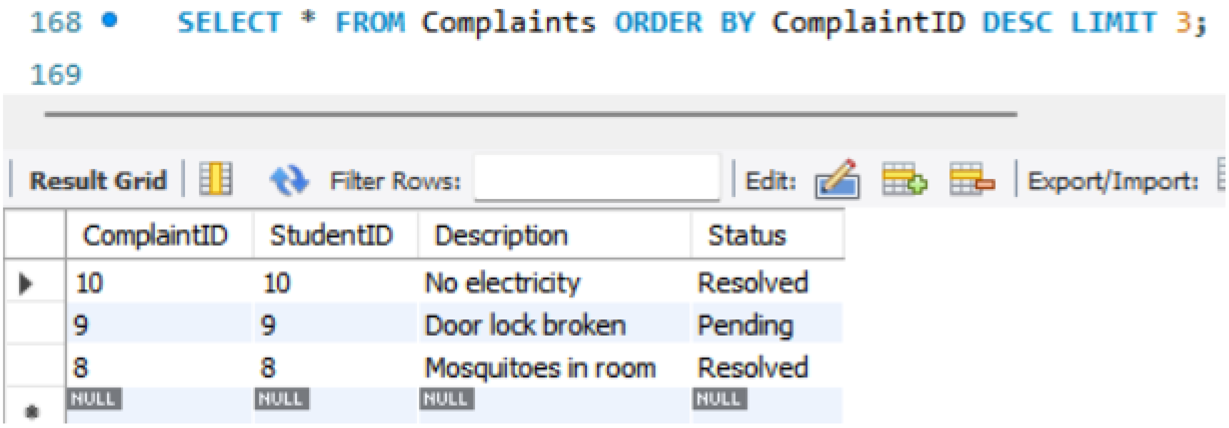
```sql
SELECT * FROM Rooms WHERE Capacity > Occupied;
```

| RoomID | RoomType | Capacity | Occupied |
|---|---|---|---|
| 103 | Triple | 3 | 2 |
| 105 | Double | 2 | 1 |
| 109 | Single | 1 | 0 |
| 110 | Double | 2 | 0 |
| NULL | NULL | NULL | NULL |



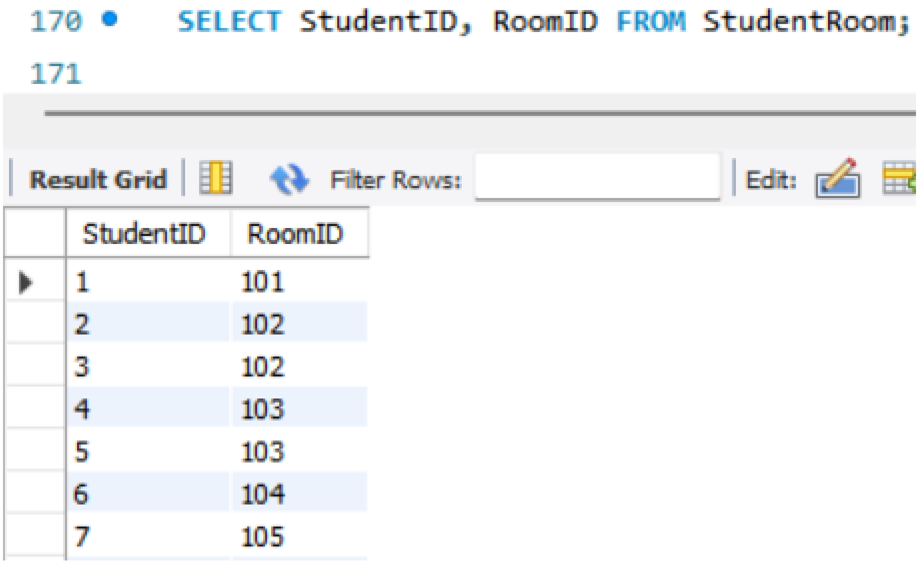
```sql
SELECT s.Name, f.Amount
FROM Fees f
JOIN Students s ON s.StudentID = f.StudentID
ORDER BY f.Amount DESC LIMIT 5;
```

| Name | Amount |
|---|---|
| Pooja | 11000.00 |
| Ravi | 10500.00 |
| Simran | 10200.00 |
| Neha | 10100.00 |
| Aman | 10000.00 |



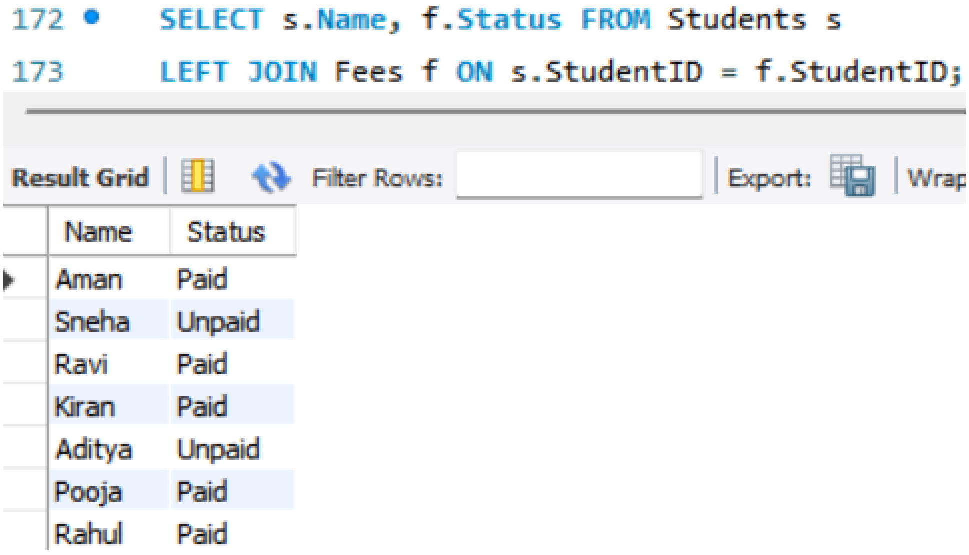
```sql
SELECT * FROM Complaints ORDER BY ComplaintID DESC LIMIT 3;
```

| ComplaintID | StudentID | Description | Status |
|---|---|---|---|
| 10 | 10 | No electricity | Resolved |
| 9 | 9 | Door lock broken | Pending |
| 8 | 8 | Mosquitoes in room | Resolved |
| NULL | NULL | NULL | NULL |



```sql
SELECT StudentID, RoomID FROM StudentRoom;
```

| StudentID | RoomID |
|---|---|
| 1 | 101 |
| 2 | 102 |
| 3 | 102 |
| 4 | 103 |
| 5 | 103 |
| 6 | 104 |
| 7 | 105 |



```sql
SELECT s.Name, f.Status FROM Students s
LEFT JOIN Fees f ON s.StudentID = f.StudentID;
```

| Name | Status |
|---|---|
| Aman | Paid |
| Sneha | Unpaid |
| Ravi | Paid |
| Kiran | Paid |
| Aditya | Unpaid |
| Pooja | Paid |
| Rahul | Paid |

- **SUMMARY:**

## Key Highlights:

- Fully normalized relational schema

- Proper mapping of many-to-many relationships

- Use of constraints for data integrity

  - **Modular Table Setup**:

    - Each entity separated into its own table
    - Easy scalability and maintenance

## Learning Outcomes:

- Practical SQL implementation experience

- Deeper understanding of normalization and constraints

- Complex query construction using joins and aggregates

## Project Application:

- Can be extended to real hostel environments

- Forms the backend for a potential full-stack web app

## Technologies Used:

- **MySQL** for DBMS
- **SQL** for data manipulation and retrieval

## Objectives:

- Design a normalized relational schema

- Implement primary and foreign key relationships

- Apply mapping constraints and many-to-many logic

- Execute meaningful SQL queries on real-world data

### Relationships:

- Student to Room: Many-to-Many (via Student_Room)

- Student to Fees: One-to-Many

- Student to Complaints: One-to-Many

These relationships ensure modularity and real-world mapping.

## • CONCLUSION:

The Hostel Management System provides an efficient and modular approach to managing hostel operations using MySQL. It enhances understanding of database concepts through practical implementation and encourages structured thinking in data modelling.

### Observations:

- The mapping constraints help normalize the data.
- Queries return accurate outputs for all relational operations

### Limitations:

- No front-end interface for users.
- Manual data insertion required.

In conclusion, the Hostel Management System offers a well-structured and normalized database design that simplifies hostel operations. The system effectively handles student data, room allocations, fee management, and complaints using a relational model. Through this project, I gained hands-on experience in SQL, database normalization, and query construction. Although the system does not include a front-end interface, the database structure is robust and can easily be extended to support web applications in the future. The system serves as a valuable tool for hostel administrators, providing accurate and efficient data management while maintaining referential integrity across all entities.

The limitations of the project primarily revolve around the lack of a user interface for interaction and the reliance on manual data entry, but these aspects can be improved in future iterations by integrating a front-end system and automating data entry processes.