



**UNIVERSITY INSTITUTE OF COMPUTING**

**CASE STUDY REPORT ON  
CINEMA MANAGEMENT SYSTEM**

Program Name: BCA

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

**Submitted By:**

**Name: Damanpreet K.  
UID: 23BCA10299  
Section: 4-‘B’**

**Submitted To:**

**Name: Mr.Arinder Singh  
Designation: Assistant Professor**



## **INTRODUCTION**

The Hostel Management System using a Database Management System (DBMS) is a modern and efficient way to handle the administrative functions of hostels in educational institutions. Hostels accommodate a large number of students, and managing their data manually can lead to inefficiencies, errors, and difficulty in tracking records. This system is designed to maintain detailed information about students, allocate rooms based on availability and capacity, record booking history, manage fee structures, and store staff data in a centralized and organized manner.

Using a relational database model, the system establishes clear relationships between various entities such as students, rooms, fees, and bookings. It provides fast data retrieval, accurate updates, and minimizes redundancy. The system enhances the overall operational flow by automating daily tasks and ensuring that records are easy to update and monitor.

By using SQL queries and database normalization, the Hostel Management System ensures data consistency, reliability, and security. This system not only improves the efficiency of hostel administration but also offers scope for integration with web portals and mobile apps for real-time access and notifications.

## TECHNIQUES

The primary technology used in this project is MySQL, an open-source relational database management system. The following techniques have been implemented:

- **Entity-Relationship Modeling** for data structure visualisation.
- **Normalisation** to organise data efficiently and remove redundancy.
- **SQL Queries** for data manipulation and retrieval.
- **Use of Constraints** like PRIMARY KEY, FOREIGN KEY to enforce relationships.
- **Join operations** to combine data from multiple tables.
- **Aggregate Functions** to summarize and analyze data.
- **Filtering and Sorting** to extract meaningful insights from the dataset.
- **Stored Procedures and Views** (optional enhancements) for automation.



**CHANDIGARH  
UNIVERSITY**

Discover. Learn. Empower.

**NAAC  
GRADE A+**

Accredited University

## **SYSTEM CONFIGURATION**

- **Operating System:**
  - Windows 10 or higher / Linux / macOS
- **Database Software:**
  - MySQL or PostgreSQL
- **RAM:**
  - Minimum 4GB
- **Processor:**
  - Intel i3 or equivalent and above
- **Other Tools:**
  - MySQL Workbench, DBeaver, or phpMyAdmin



## INPUT

The system accepts a variety of inputs to populate and manage the hostel database efficiently. These inputs ensure smooth operations by keeping records updated, organized, and easily accessible. The key inputs include:

- **Student Details**

(First Name, Last Name, Date of Birth, Gender, Email, Contact Number, Address)

→ Used to register and identify students staying in the hostel.

- **Room Details**

(Room Number, Room Type, Room Capacity, Room Status)

→ Defines the rooms available in the hostel, including their occupancy and type (Single/Double/etc.).

- **Booking Details**

(Student ID, Room ID, Booking Date, Check-In Date, Check-Out Date)

→ Helps in assigning rooms to students and managing their duration of stay.

- **Fee Details**

(Student ID, Amount, Due Date, Paid Date, Status)

→ Tracks payments, due dates, and pending amounts for hostel accommodation.

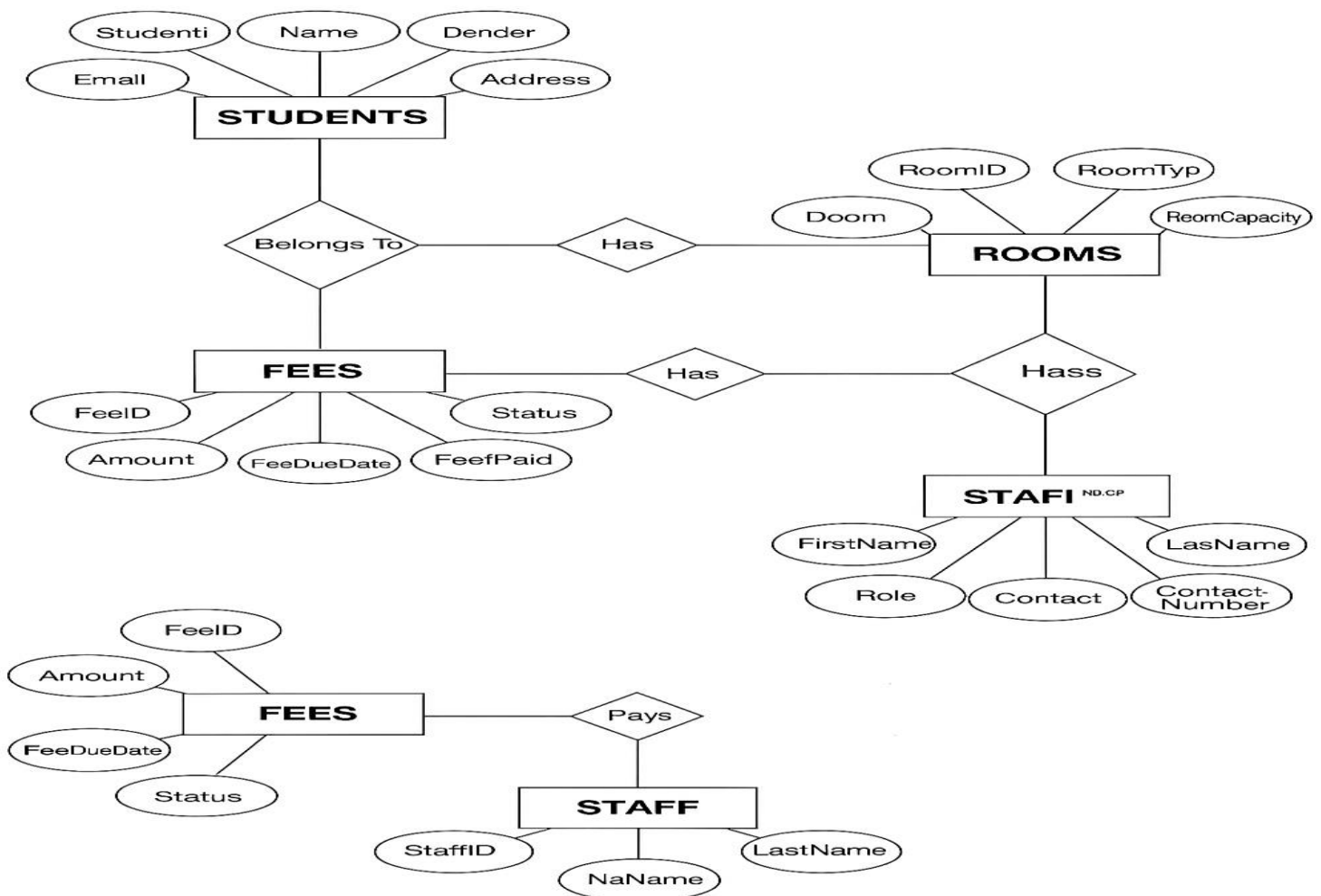
- **Staff Details**

(First Name, Last Name, Role, Contact Number, Email)

→ Maintains a record of hostel staff such as wardens, caretakers, and maintenance personnel.

Each of these input categories plays a crucial role in ensuring that the hostel management system remains accurate, functional, and up to date.

## ENTITY-RELATIONSHIP DIAGRAM



The Entity-Relationship (ER) diagram outlines the structure and relationships among different entities of the hostel. It forms the blueprint for the actual database schema.

## RELATIONSHIP BETWEEN TABLES

Here's the **relationship summary** between the tables in your **Hostel Management System** in tabular form:

No.	Relationship Type	Parent Table	Child Table	Foreign Key	Description
1	<b>One-to-many</b>	Students	Bookings	student_id	One student can have multiple bookings.
2	<b>One-to-many</b>	Rooms	Bookings	room_id	One room can have multiple bookings.
3	<b>One-to-many</b>	Students	Fees	student_id	One student can have multiple fee records (for different periods).
4	<b>One-to-one</b>	Bookings	Students	student_id	Each booking is linked to one student (each student can have many bookings, but each booking links to only one student).
5	<b>One-to-one</b>	Bookings	Rooms	room_id	Each booking is linked to one room (each room can have multiple bookings, but each booking is for only one room).
6	<b>One-to-many</b>	Staff	Rooms	staff_id (optional)	A staff member (like a warden or caretaker) can be responsible for multiple rooms. (This can be optional based on your design)

## the Tabular Format (Schema) for your Hostel Management System

<u>Table Name</u>	<u>Primary Key</u>	<u>Foreign Key</u>	<u>Description</u>
<u>Students</u>	<u>student_id</u>	=	<u>Stores personal details of students</u>
<u>Rooms</u>	<u>room_id</u>	=	<u>Contains details of hostel rooms</u>
<u>Bookings</u>	<u>booking_id</u>	<u>student_id,</u> <u>room_id</u>	<u>Records room booking by students</u>
<u>Fees</u>	<u>fee_id</u>	<u>student_id</u>	<u>Tracks fee payments for hostel stay</u>
<u>Staff</u>	<u>staff_id</u>	=	<u>Stores information about hostel staff</u>

### TABLE CREATION

#### Create Table

##### 1.) Table student

```
CREATE TABLE Students (  
  student_id INT AUTO_INCREMENT PRIMARY KEY,  
  first_name VARCHAR(50),  
  last_name VARCHAR(50),  
  dob DATE,  
  gender ENUM('Male', 'Female', 'Other'),  
  email VARCHAR(100),  
  contact_number VARCHAR(15),  
  address TEXT  
);
```

```
INSERT INTO Students (first_name, last_name, dob, gender, email,  
contact_number, address)  
VALUES  
( 'John', 'Doe', '1998-07-12', 'Male', 'john.doe@example.com', '9876543210',  
'123 Main St, City, Country'),  
( 'Jane', 'Smith', '2000-11-20', 'Female', 'jane.smith@example.com',  
'9123456789', '456 Another St, City, Country');
```

##### 2.) Table rooms



```
CREATE TABLE Rooms (  
    room_id INT AUTO_INCREMENT PRIMARY KEY,  
    room_number VARCHAR(10) UNIQUE,  
    room_type ENUM('Single', 'Double', 'Triple', 'Quad'),  
    room_capacity INT,  
    room_status ENUM('Available', 'Occupied', 'Under Maintenance')  
);
```

Insert Sample Data

```
INSERT INTO Rooms (room_number, room_type, room_capacity, room_status)  
VALUES  
( 'A101', 'Single', 1, 'Available'),  
( 'A102', 'Double', 2, 'Available'),  
( 'B201', 'Triple', 3, 'Occupied');
```

### **3.) Table bookings**

```
CREATE TABLE Bookings (  
    booking_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT,  
    room_id INT,  
    booking_date DATE,  
    check_in_date DATE,  
    check_out_date DATE,  
    FOREIGN KEY (student_id) REFERENCES Students(student_id),  
    FOREIGN KEY (room_id) REFERENCES Rooms(room_id)  
);
```

Insert Sample Data

```
INSERT INTO Bookings (student_id, room_id, booking_date, check_in_date,  
check_out_date)  
VALUES  
(1, 1, '2025-04-10', '2025-04-12', '2025-05-12'),  
(2, 2, '2025-04-11', '2025-04-15', '2025-05-15');
```

### **4.) Table fees**

```
CREATE TABLE Fees (  
    fee_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT,  
    amount DECIMAL(10, 2),  
    fee_due_date DATE,  
    fee_paid_date DATE,  
    status ENUM('Paid', 'Pending'),  
    FOREIGN KEY (student_id) REFERENCES Students(student_id)  
);
```

Insert Sample Data

```
INSERT INTO Fees (student_id, amount, fee_due_date, fee_paid_date, status)  
VALUES  
(1, 500.00, '2025-04-30', '2025-04-20', 'Paid'),  
(2, 800.00, '2025-04-30', NULL, 'Pending');
```

## **5.) Table staff**

```
CREATE TABLE Staff (  
    staff_id INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    role VARCHAR(50),  
    contact_number VARCHAR(15),  
    email VARCHAR(100)  
);
```

Insert Sample Data

```
INSERT INTO Staff (first_name, last_name, role, contact_number, email)  
VALUES  
( 'Alice', 'Johnson', 'Warden', '9988776655', 'alice.johnson@hostel.com'),  
( 'Bob', 'Brown', 'Caretaker', '9777886655', 'bob.brown@hostel.com');
```

## SQL QUERIES (13 Queries)

### 1. View All Students Query

**SELECT \* FROM Students;**

**Output:** —

73

SQL > SELECT \* FROM Students;

<

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	student_id	first_name	last_name	dob	gender	email	contact_number	address
▶	1	John	Doe	1998-07-12	Male	john.doe@example.com	9876543210	123 Main St, City, Country
	2	Jane	Smith	2000-11-20	Female	jane.smith@example.com	9123456789	456 Another St, City, Country
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 2. View Available Rooms




**Query**

**SELECT \* FROM Rooms WHERE room\_status = 'Available';**

3.2. SELECT FROM BOOKINGS

Result Grid

Filter Rows:

Edit:    Export/Import

	room_id	room_number	room_type	room_capacity	room_status
▶	1	A101	Single	1	Available
	2	A102	Double	2	Available
✱	NULL	NULL	NULL	NULL	NULL

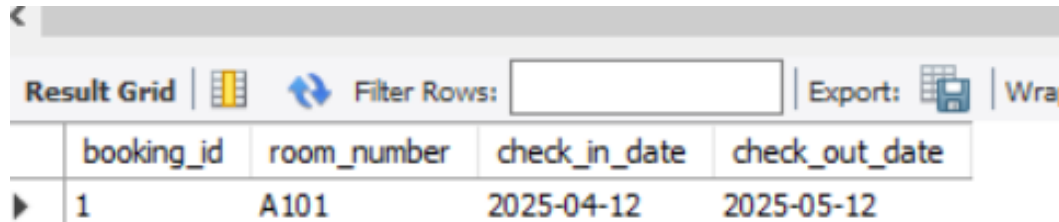
Output:

### 3. Show Booking Details for Student ID = 1

Query

```
SELECT b.booking_id, r.room_number, b.check_in_date,
b.check_out_date
FROM Bookings b
JOIN Rooms r ON b.room_id = r.room_id
WHERE b.student_id = 1
```

**Output:**



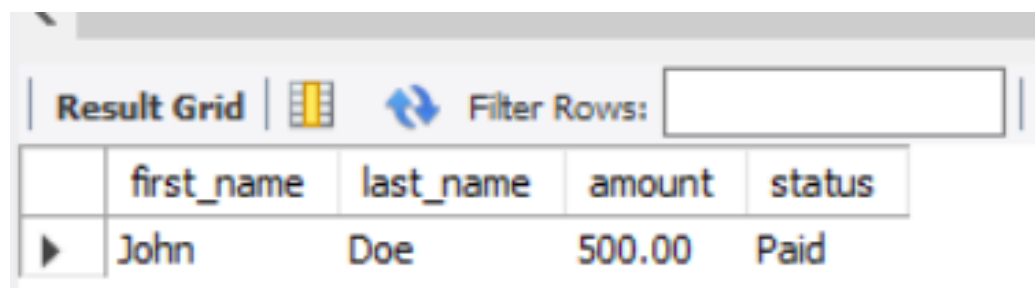
	booking_id	room_number	check_in_date	check_out_date
▶	1	A101	2025-04-12	2025-05-12

#### 4. Get Fee Status for Student ID = 1

**Query**

```
SELECT s.first_name, s.last_name, f.amount, f.status
FROM Fees f
JOIN Students s ON f.student_id = s.student_id
WHERE s.student_id = 1;
```

**Output:**



	first_name	last_name	amount	status
▶	John	Doe	500.00	Paid

#### 5. Update Room Status

**Query**


```
UPDATE Rooms
SET room_status = 'Occupied'
WHERE room_id = 1;
```

**Output:**

**Action**

- Updates room\_id = 1 (A101) status from Available → Occupied

**Post-Update Output for Room ID 1**

Result Grid					
Filter Rows: <input type="text"/>					
Edit: 					
	room_id	room_number	room_type	room_capacity	room_status
▶	1	A101	Single	1	Occupied
	2	A102	Double	2	Available
	3	B201	Triple	3	Occupied
•	NULL	NULL	NULL	NULL	NULL

## 6. Update Fee Status

**Query**

UPDATE Fees

SET status = 'Paid', fee\_paid\_date = '2025-04-20'


WHERE fee\_id = 1;

**Output:**

**Action**

- Updates fee record of fee\_id = 1 to mark as Paid and sets the fee\_paid\_date.

**Post-Update Output for Fee ID 1**

Result Grid						
Filter Rows: <input type="text"/>						
Edit: 						
	fee_id	student_id	amount	fee_due_date	fee_paid_date	status
▶	1	1	500.00	2025-04-30	2025-04-20	Paid
	2	2	800.00	2025-04-30	NULL	Pending
•	NULL	NULL	NULL	NULL	NULL	NULL

## 7. Delete Booking Record Query

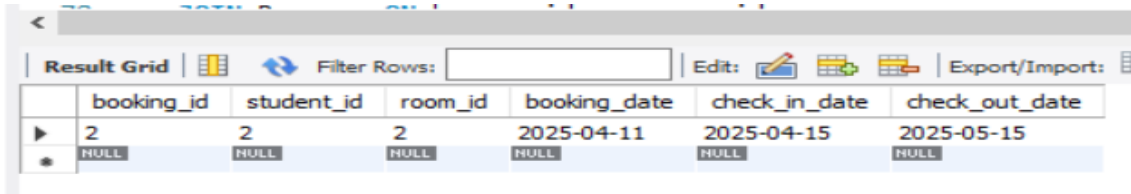
**DELETE FROM Bookings  
WHERE booking\_id = 1;**

**Output:**

**Action**

- Deletes the booking where booking\_id = 1 (for John Doe in room A101).

**Post-Delete Output from Bookings**



	booking_id	student_id	room_id	booking_date	check_in_date	check_out_date
▶	2	2	2	2025-04-11	2025-04-15	2025-05-15
*	NULL	NULL	NULL	NULL	NULL	NULL



## SUMMARY

The **Hostel Management System** is a database-driven project designed to streamline the management of student accommodations in a hostel. It efficiently handles key operations like student registrations, room allocations, fee tracking, and staff details using SQL and MySQL. The system is built around five main tables: Students, Rooms, Bookings, Fees, and Staff, each serving a specific function in managing hostel data.

Relationships between the tables ensure data consistency—for example, one student can have multiple bookings and fee records, and rooms can be booked by different students over time. The project demonstrates core SQL operations such as CREATE, INSERT, SELECT, UPDATE, and DELETE, along with JOIN queries for data retrieval across multiple tables.

Sample data is inserted to simulate real-world scenarios like booking rooms, updating fee payment statuses, and checking room availability. The project offers a strong foundation for building more advanced features like mess management, login systems, or a web interface. Overall, this project showcases the practical application of relational databases in managing hostel operations, with a focus on scalability, data integrity, and real-time updates.



## Conclusion

The **Hostel Management System** project demonstrates the practical application of a **relational database** to manage various aspects of hostel operations such as student details, room bookings, fee management, and staff records. The project utilizes **SQL** (MySQL) for creating and managing tables, ensuring efficient storage, retrieval, and updating of data. By defining clear relationships between tables (e.g., students, rooms, bookings, and fees), the system ensures consistency and integrity of data, minimizing errors during day-to-day operations.

### Key Observations:

1. **Structured Database Design:** The project effectively utilizes **normalization** principles, minimizing redundancy and improving data integrity.
2. **Ease of Use:** The SQL-based approach provides a clear structure for operations such as querying room availability, managing bookings, and updating fee records.
3. **Real-World Application:** The system successfully mirrors real-world hostel management tasks, such as booking rooms, updating statuses, and managing payments.

### Limitations:

1. **Lack of User Interface:** The current system lacks a **front-end interface**, which would make the project more user-friendly and interactive. It is primarily focused on back-end database management.
2. **Limited Features:** The system only covers basic hostel management functions (student details, rooms, bookings, and fees). More advanced functionalities, like mess management, complaints handling, and reporting, are not included.
3. **Scalability:** Although the system is functional, it lacks advanced scalability features for handling a large number of users or more complex workflows.

### Future Scope:

1. **Front-End Integration:** The project can be enhanced by integrating a **user interface** (e.g., using **PHP**, **Python Flask**, or **JavaScript frameworks**) for better interaction.

2. **Advanced Features:** Future versions of the system can include **mess management, complaint handling, maintenance scheduling, and staff attendance tracking.**
3. **Automated Alerts & Reminders:** Incorporating email/SMS notifications for fee due dates, booking reminders, and room availability would enhance user experience.
4. **Mobile Application:** The system can be extended into a **mobile application** for both students and staff, making it more accessible and convenient.
5. **Data Analytics:** Integration of **data analysis** tools for generating reports on room occupancy, fee payments, and booking trends could assist in decision-making.