Assignment 1: Module B Report

Project: CheckInOut - Hostel Management System

Group Member:

1. Mohit Kamlesh Panchal (Roll Number: 23110208)

GitHub Link: https://github.com/rayvego/databases-assignment-1

---

1. Introduction & Requirement Analysis

This project aims to modernize hostel administration by replacing manual registers with a robust, database-driven system. The "CheckInOut" system ensures real-time tracking of student occupancy, visitor movements, and facility maintenance.

1.1 Problem Statement & Scope

Current manual processes lead to errors in room allocation, security lapses during visitor entry, and delayed maintenance responses. **Proposed Solution:** A relational database system that enforces strict integrity constraints and provides real-time data access for administrators.

1.2 Core System Functionalities

The database is designed to support the following 5 critical operations:

1. **Dynamic Room Allocation:** Automatically assigns rooms based on type (AC/Non-AC) and capacity, preventing overbooking.
2. **Gate Pass Management:** A digital log for student entry/exit that calculates duration and flags late returns.
3. **Visitor Tracking:** Records visitor details and links them to specific student visits for security auditing.
4. **Maintenance Workflow:** A ticketing system where students report issues and staff resolve them, tracking status changes.
5. **Fee & Payment Management:** Tracks hostel and mess fee payments to identify defaulters.

---

2. Conceptual Design: UML Class Diagram

The UML Class Diagram serves as the object-oriented blueprint for the system, focusing on entities and their relationships before implementation.

2.1 Key Design Decisions

- **Inheritance (Generalization):** We identified that `Student` and `Staff` share common attributes like `Name`, `Email`, and `Contact`. To avoid redundancy, we created a `Member` superclass. This adheres to the **"Don't Repeat Yourself" (DRY)** principle.
- **Composition:** `HostelBlock` is composed of `Room` objects. This strong relationship implies that a Room cannot exist without a Block.
- **Association:** We used an association class pattern for `Allocation` to handle the Many-to-Many relationship between `Student` and `Room`, allowing us to track *history* (StartDate, EndDate) rather than just current status.

---

3. Logical Design: Entity-Relationship (ER) Diagram

The ER Diagram translates the conceptual model into a relational schema, defining the exact structure of tables, keys, and cardinality.

3.1 Detailed Table Descriptions

The schema consists of 11 normalized tables (3NF):

1. **`Member` (Superclass):**
    - *Purpose:* Stores base identity information for all users.
    - *Key Attributes:* `MemberID` (PK), `Email` (Unique), `UserType` (Discriminator).
2. **`Student` (Subclass):**
    - *Purpose:* Extends Member with academic details.
    - *Key Attributes:* `StudentID` (PK/FK), `EnrollmentNo`, `GuardianContact`.
3. **`Staff` (Subclass):**
    - *Purpose:* Extends Member with employment details.
    - *Key Attributes:* `StaffID` (PK/FK), `Designation`, `ShiftDetails`.
4. **`HostelBlock`:**
    - *Purpose:* Represents physical buildings.
    - *Key Attributes:* `BlockID`, `WardenID` (FK to Staff).
5. **`Room`:**
    - *Purpose:* Individual units within blocks.
    - *Key Attributes:* `RoomID`, `Capacity`, `CurrentOccupancy`.
6. **`Allocation`:**
    - *Purpose:* Links Students to Rooms for specific durations.
    - *Key Attributes:* `AllocationID`, `CheckInDate`, `Status`.
7. **`GatePass`:**
    - *Purpose:* Logs daily movement of students.

    - *Key Attributes:* `PassID`, `OutTime`, `ExpectedInTime`.
8. **`Visitor`**:
    - *Purpose:* Stores unique visitor profiles to avoid re-entry of data.
    - *Key Attributes:* `VisitorID`, `GovtIDProof`.
9. **`VisitLog`**:
    - *Purpose:* Records individual visits by a Visitor to a Student.
    - *Key Attributes:* `VisitID`, `CheckInTime`, `Purpose`.
10. **`MaintenanceRequest`**:
    - *Purpose:* Tracks facility issues.
    - *Key Attributes:* `RequestID`, `Priority`, `Status` (Open/Resolved).
11. **`FeePayment`**:
    - *Purpose:* Financial records for hostel fees.
    - *Key Attributes:* `PaymentID`, `Amount`, `TransactionID`.

---

4. Database Integrity & Constraints

To ensure data quality and prevent logical errors, the following constraints were implemented in SQL:

4.1 Domain Integrity (CHECK Constraints)

- **Age Restriction:** `CHECK (Age >= 16)` ensures valid student/staff entries.
- **Time Logic:** `CHECK (ExpectedInTime > OutTime)` prevents logical errors in Gate Pass logs.
- **Occupancy Limits:** `CHECK (CurrentOccupancy >= 0)` and `CHECK (Capacity > 0)` prevents invalid room states.
- **Payment Validity:** `CHECK (Amount > 0)` ensures positive financial transactions.

4.2 Referential Integrity (Foreign Keys)

- **Cascading Actions:** `ON DELETE CASCADE` is used for `Student` -> `Allocation` relationships. If a student record is removed, their room allocation is automatically cleared.
- **Set Null:** `ON DELETE SET NULL` is used for `HostelBlock.WardenID`. If a warden leaves, the block remains but has no assigned warden until updated.

4.3 Normalization (3NF)

All tables are in **Third Normal Form (3NF)**.

- *1NF:* All attributes are atomic.
- *2NF:* All non-key attributes depend on the entire Primary Key.
- *3NF:* No transitive dependencies exist (e.g., `BlockName` is in `HostelBlock`, not repeated in `Room`).

---

5. UML to ER Transition

This section explains how the UML class diagram was systematically converted into the ER diagram while preserving all semantic relationships.

5.1 Class-to-Entity Mapping

Each UML class directly maps to one ER entity with identical attributes:

UML Class	ER Entity	Mapping Notes
Member	MEMBER	Superclass - all attributes preserved
Student	STUDENT	Subclass - inherits MemberID as FK/PK
Staff	STAFF	Subclass - inherits MemberID as FK/PK
HostelBlock	HOSTEL_BLOCK	Name changed to snake_case for SQL
Room	ROOM	All attributes preserved
Allocation	ALLOCATION	Association class becomes entity
GatePass	GATE_PASS	All attributes preserved
Visitor	VISITOR	All attributes preserved
VisitLog	VISIT_LOG	All attributes preserved
MaintenanceRequest	MAINTENANCE_REQUEST	All attributes preserved

UML Class	ER Entity	Mapping Notes
FeePayment	FEE_PAYMENT	All attributes preserved

5.2 Relationship-to-ER Mapping

<table>
<tr><th>UML Relationship</th><th>ER Equivalent</th><th>Cardinality</th></tr>
<tr><td>Member <|-- Student (Generalization)</td><td>MEMBER ||--o{ STUDENT</td><td>1:M (total participation)</td></tr>
<tr><td>Member <|-- Staff (Generalization)</td><td>MEMBER ||--o{ STAFF</td><td>1:M (total participation)</td></tr>
<tr><td>HostelBlock *-- Room (Composition)</td><td>HOSTEL_BLOCK ||--o{ ROOM</td><td>1:M (total)</td></tr>
<tr><td>HostelBlock -- Staff : warden</td><td>HOSTEL_BLOCK ||--o| STAFF</td><td>1:1 (optional)</td></tr>
<tr><td>Student -- Allocation (Association)</td><td>STUDENT ||--o{ ALLOCATION</td><td>1:M</td></tr>
<tr><td>Room -- Allocation (Association)</td><td>ROOM ||--o{ ALLOCATION</td><td>1:M</td></tr>
</table>

5.3 Key Design Decisions in Transition

- **Primary Keys:** UML `+int MemberID PK` becomes ER `int MemberID PK`
- **Foreign Keys:** UML `+int StudentID PK, FK` becomes ER `int StudentID PK, FK`
- **Multiplicity:**
  - UML `1` → ER `||` (exactly one)
  - UML `M` → ER `o{` (zero or more)
  - UML `0..1` → ER `o|` (zero or one)

---

6. Relationship Justification with Examples

Each relationship type is justified with real-world examples from the hostel management domain.

6.1 Generalization (Member → Student, Staff)

- **Type:** Total specialization, partial parent
- **Justification:** Both students and staff share common attributes (Name, Email, Contact, Age, Gender). Creating a Member superclass eliminates data redundancy and ensures consistent identity management.
- **Example:** When a student becomes a staff member, their basic details already exist in Member table; only the role-specific attributes need to be added.

6.2 Composition (HostelBlock → Room)

- **Type:** 1:M with total participation
- **Justification:** A room cannot exist independently without a hostel block. When a block is demolished, all its rooms are automatically removed.
- **Example:** Block "Himalaya" contains rooms 101, 102, 201, 202. Deleting BlockID=1 removes all these rooms.

6.3 Association Class (Allocation)

- **Type:** M:M resolved through association entity
- **Justification:** A student can occupy multiple rooms over time (transfers), and a room can have multiple students (bed changes). Tracking historical allocations requires dates.
- **Example:** StudentID=3 was in RoomID=1 from Aug 2023 to Dec 2023, then moved to RoomID=2 from Jan 2024 onwards.

6.4 One-to-Many (Student → GatePass)

- **Type:** 1:M
- **Justification:** A student can request multiple gate passes over time for different purposes (shopping, medical, family events).
- **Example:** StudentID=5 has 3 gate passes: one for library (Feb 3), one for family function (Feb 2), one for night stay (Feb 4).

6.5 Optional Association (HostelBlock → Staff as Warden)

- **Type:** 1:0..1
- **Justification:** A hostel block may or may not have an assigned warden at any given time. The relationship allows null warden assignment.
- **Example:** Block "Yamuna" has WardenID=2 (Sita Devi), but Block "Vindhya" may have NULL warden during transition period.

6.6 Weak Entity (VisitLog)

- **Type:** Identifying relationship with Visitor and Student
- **Justification:** VisitLog cannot exist without both Visitor and Student. A visit is identified by the combination of who visited whom and when.
- **Example:** VisitorID=1 visiting StudentID=3 creates VisitID=5. Without both foreign keys, the visit record is meaningless.