CS 432: Databases

FixIIT – Maintenance Portal

Assignment 1

Submitted by: Team CODEBASE

Abhitej Singh Bhullar (23110009)
Shiv Patel (23110302)
Shivansh Shukla (23110303)
Soham Shrivastava (23110315)
Arin Mehta (23110038)

Github Link:
https://github.com/ArinMehta/FixIIT

Contents

1 Project Objective

The primary objective of this project is to design, implement, and evaluate a robust database-driven software system that integrates theoretical database concepts with real-world system development. The system, named **FixIIT – Campus Maintenance Management System**, is developed to manage maintenance requests across a university campus environment. It enables members to raise maintenance tickets, allows supervisors to assign technicians, tracks ticket statuses, records communication logs, and collects service feedback.

The project emphasizes proper relational schema design, enforcement of integrity constraints, referential integrity, domain-specific logical validations, and real-world data modelling. By implementing strict SQL modes and constraint-driven design, the system ensures correctness, scalability, and consistency while demonstrating practical applicability of database management principles.

2 System Overview

The FixIIT system is a structured relational database built using MySQL. The system consists of ten interrelated tables representing core entities such as members, tickets, roles, categories, statuses, locations, assignments, comments, and feedback.

The workflow begins when a member raises a maintenance ticket specifying category, priority, and location. The ticket is assigned a status and can later be allocated to a technician through the assignments module. Communication between stakeholders is maintained through ticket comments (designed as a weak entity to enforce strict dependency on tickets), while service quality is measured using feedback and ratings. Role-based access control is implemented using the `member_roles` table.

Strict SQL modes are enabled to prevent invalid data insertion:

```
SET SESSION sql_mode =
'STRICT_TRANS_TABLES,NO_ZERO_DATE,NO_ZERO_IN_DATE,
ERROR_FOR_DIVISION_BY_ZERO,ONLY_FULL_GROUP_BY';
```

The database ensures enforcement of business logic through CHECK constraints, UNIQUE constraints, and Foreign Key relationships with ON DELETE and ON UPDATE rules.

Module A

3 Constraint Compliance

3.1 Member Table Requirement

The project requires a Member table with specific attributes.

Implementation:

```
CREATE TABLE members (
  member_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(80) NOT NULL,
```

```
  image VARCHAR(255) NOT NULL,
  age INT NOT NULL,
  email VARCHAR(120) NOT NULL,
  contact_number VARCHAR(20) NOT NULL,
  address VARCHAR(200) NOT NULL,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (email),
  CHECK (age > 16)
);
```

This table includes all mandatory attributes and enforces age validation and email uniqueness.

3.2 Minimum Functionalities

The system supports more than five core functionalities:

- Ticket Creation
- Technician Assignment
- Status Tracking
- Comment Logging
- Feedback Submission
- Role Management

Functional integration is verified through JOIN queries across tickets, members, categories, and statuses.

3.3 Minimum Entities

The system includes the following entities:

Members, Roles, Tickets, Categories, Statuses, Locations, Assignments, Ticket Comments, Feedback.

Total Entities = 9 (greater than required 5).

3.4 Minimum Tables

The database contains exactly ten tables:

```
roles, statuses, categories, locations, members,
member_roles, tickets, assignments, ticket_comments,
feedback
```

3.5 Primary and Foreign Keys

All tables contain primary keys, and relationships use foreign keys:

```
CONSTRAINT fk_tickets_member
FOREIGN KEY (member_id)
REFERENCES members(member_id)
ON DELETE RESTRICT ON UPDATE CASCADE;
```

Weak Entity Implementation (Composite Key):

```
PRIMARY KEY (ticket_id, comment_seq)
```

This composite key is used in the `ticket_comments` table to ensure that comments are structurally dependent on their parent ticket (Weak Entity), preventing orphan comments.

3.6 Implemented Schema Diagram

The following figure presents the physical schema diagram generated directly from the implemented MySQL database. It visualizes the ten tables, their respective attributes, data types, and the enforced relationships (foreign key constraints) connecting them. This diagram confirms that the implemented structure aligns precisely with the project design requirements.

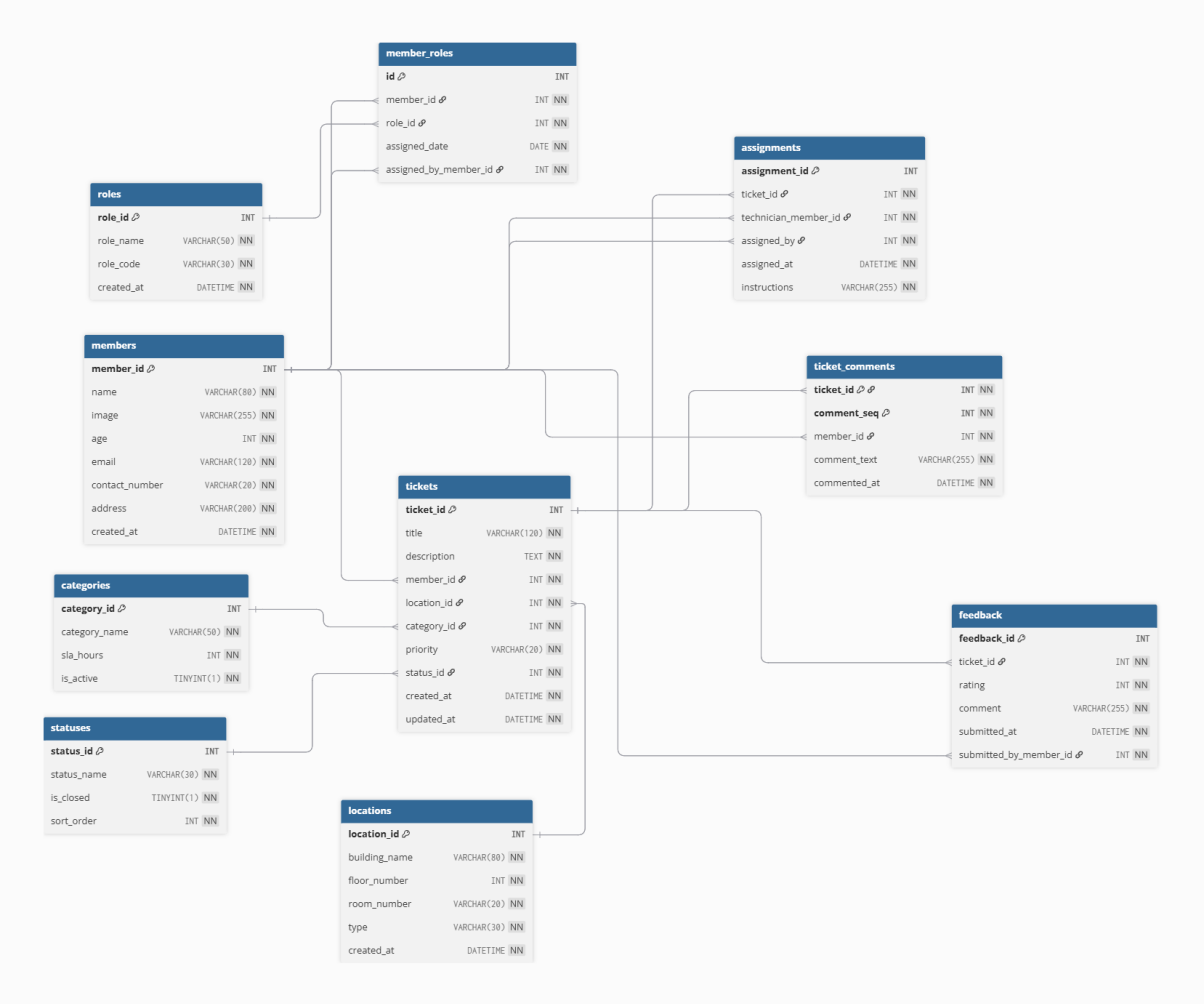

Figure 1: Physical Schema Diagram of the FixIIT Database Implementation.

3.7 Functional Coverage

Example of integrated query:

```
SELECT t.ticket_id, m.name, c.category_name,
s.status_name
FROM tickets t
JOIN members m ON t.member_id = m.member_id
JOIN categories c ON t.category_id = c.category_id
JOIN statuses s ON t.status_id = s.status_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ticket_id	name	category_name	status_name
6	Shivansh	Air Conditioning	Closed
17	Shivansh	Air Conditioning	Closed
5	Dilip Singh	Carpentry	Closed
10	Student A	Carpentry	Assigned
13	Attendant A	Carpentry	In_Progress
3	Attendant B	Civil Maintenance	Closed
18	Student B	Civil Maintenance	Closed
1	Prof. ABC	Electrical	Closed
7	Abhishek	Electrical	Assigned
11	Farhan Obaid	Electrical	In_Progress
15	Jiya	Electrical	On_Hold
14	Student B	Housekeeping	In_Progress
20	Abhinav	Housekeeping	Closed
4	Soham	IT Support	Closed
9	Soham	IT Support	Assigned
16	Dean, SA	IT Support	On_Hold
19	Attendant A	IT Support	Closed
2	Arin Mehta	Plumbing	Closed
8	Abhinav	Plumbing	Assigned
12	Arin Mehta	Plumbing	In_Progress

Figure 2: Result Grid showing integrated query execution.

This confirms that schema design directly supports operational workflow.

3.8 Real-Life Data (10–20 Rows)

Example:

```
INSERT INTO categories (category_name, sla_hours, is_active)
VALUES ('Electrical', 24, 1),
       ('Plumbing', 24, 1),
       ('Air Conditioning', 48, 1);
```

Each major table contains between 10–30 meaningful rows reflecting real campus scenarios.

3.9 Referential Integrity

Implemented using cascading and restrictive rules:

```
FOREIGN KEY (ticket_id)
REFERENCES tickets(ticket_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

This prevents orphan records and maintains consistency.

3.10 NOT NULL Columns

Example from tickets table:

```
title VARCHAR(120) NOT NULL,
description TEXT NOT NULL,
member_id INT NOT NULL,
location_id INT NOT NULL,
```

```
category_id INT NOT NULL,
priority VARCHAR(20) NOT NULL,
status_id INT NOT NULL
```

Every table contains at least three NOT NULL columns.

3.11 Unique Row Identification

Implemented through:

```
member_id INT AUTO_INCREMENT PRIMARY KEY
UNIQUE (email)
UNIQUE (building_name, floor_number, room_number)
```

Each row is uniquely identifiable.

3.12 Logical Constraints

Examples of domain constraints:

```
CHECK (priority IN ('Low','Medium',
'High','Urgent','Emergency'))

CHECK (rating BETWEEN 1 AND 5)

CHECK (created_at <= updated_at)

CHECK (sla_hours > 0)
```

These enforce real-world business rules at the database level.

4 Data Validation & Testing

To ensure the robustness of the database, we performed negative testing to verify that the integrity constraints (CHECK, UNIQUE, FOREIGN KEY) are correctly enforcing business rules.

4.1 Validation of CHECK Constraint (Age)

The following query was executed to test the age restriction logic:

Listing 1: Testing Invalid Age Insertion

```
INSERT INTO members
(name, image, age, email, contact_number, address)
VALUES
('Invalid Age User', 'invalid.jpg', 10, 'invalid@iitgn.ac.in',
 '+91 9000000111', 'Test Address');
```

Observation: MySQL rejected the query with **Error Code: 3819. Check constraint 'members_chk_1' is violated.**

Explanation: The `members` table includes the constraint:

```
CHECK (age > 16)
```

Since the input value (10) violates this condition, the system correctly prevents the insertion of invalid user data.

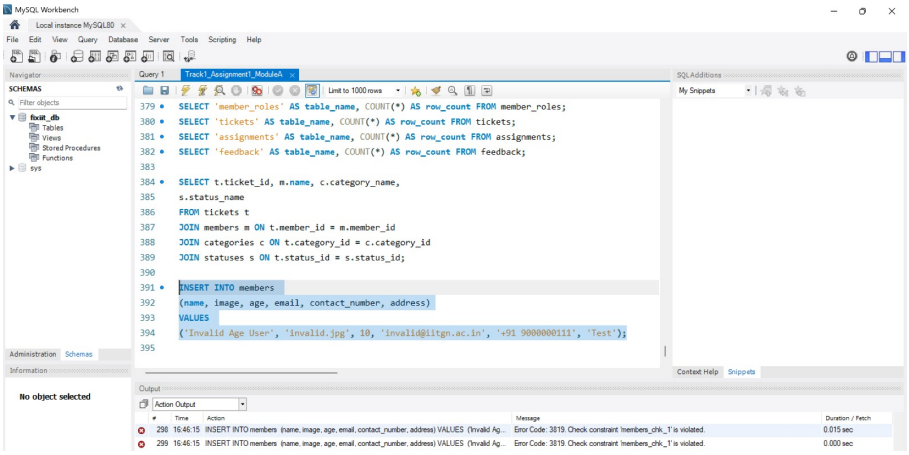

Figure 3: Screenshot of MySQL Workbench rejecting invalid age.

4.2 Validation of UNIQUE Constraint (Duplicate email

The following query was executed:

```
INSERT INTO members
(name, image, age, email, contact_number, address)
VALUES
('Another User', 'user.jpg', 22,
 'shiv.patel@iitgn.ac.in', '+91 9000000222', 'Test Address');
```

Error Explanation:

The query fails because the `email` attribute in the `members` table has a UNIQUE constraint. Since the email `shiv.patel@iitgn.ac.in` already exists in the database, MySQL prevents duplicate entries and raises a duplicate key error.

This confirms that the UNIQUE constraint is properly enforced.

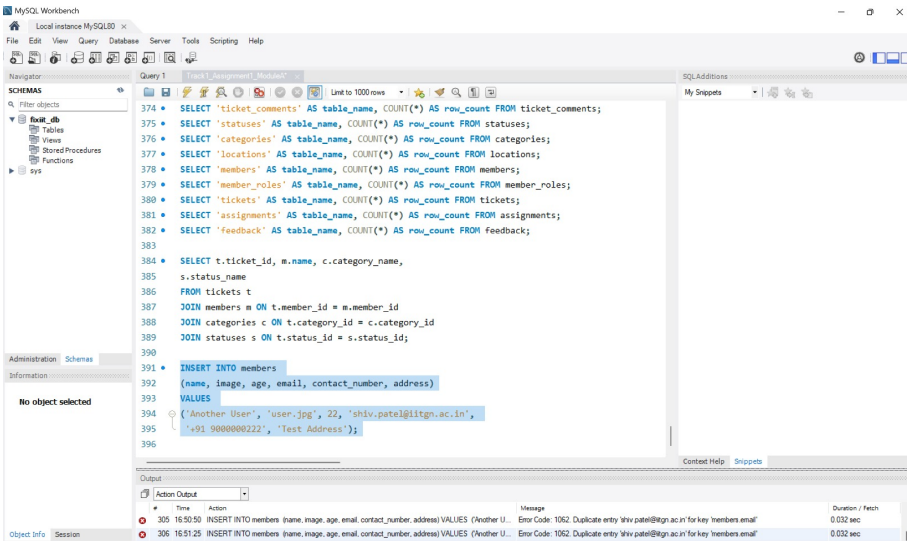

Figure 4: Screenshot of MySQL Workbench rejecting duplicate email

4.3 Validation of CHECK Constraint on Priority

The following query was executed:

```
INSERT INTO tickets
(title, description, member_id, location_id, category_id,
   priority, status_id)
VALUES
('Bad priority', 'Invalid priority value', 3, 1, 1, 'SuperHigh',
   1);
```

Error Explanation:

The query fails because the `priority` column has a CHECK constraint:

```
CHECK (priority IN ('Low','Medium','High','Urgent','Emergency'))
```

Since the value `'SuperHigh'` is not part of the allowed set, the CHECK condition evaluates to FALSE and MySQL rejects the insertion.

This demonstrates that the schema correctly enforces domain integrity.

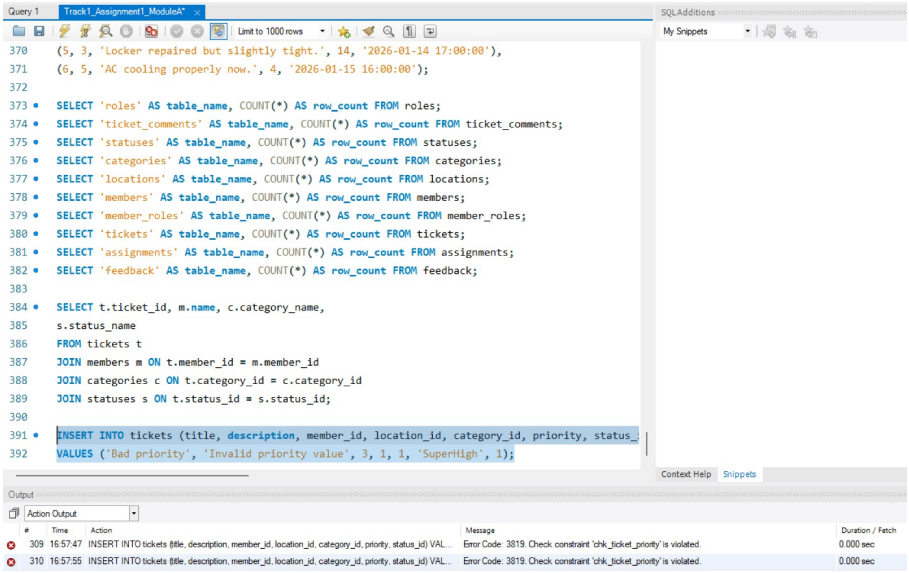

Figure 5: Screenshot of MySQL Workbench rejecting invalid priority

Module B

5 Conceptual Design (ER Diagram)

This section presents the conceptual modeling of the FixIIT system. The design translates the relational schema defined in Module A into a formal Entity-Relationship (ER) diagram, strictly adhering to the Silberschatz notation.

5.1 ER Diagram

The diagram below visualizes the ten entities and their interrelationships. It captures cardinality constraints, participation requirements, and the specific "Weak Entity" structure implemented for comments.

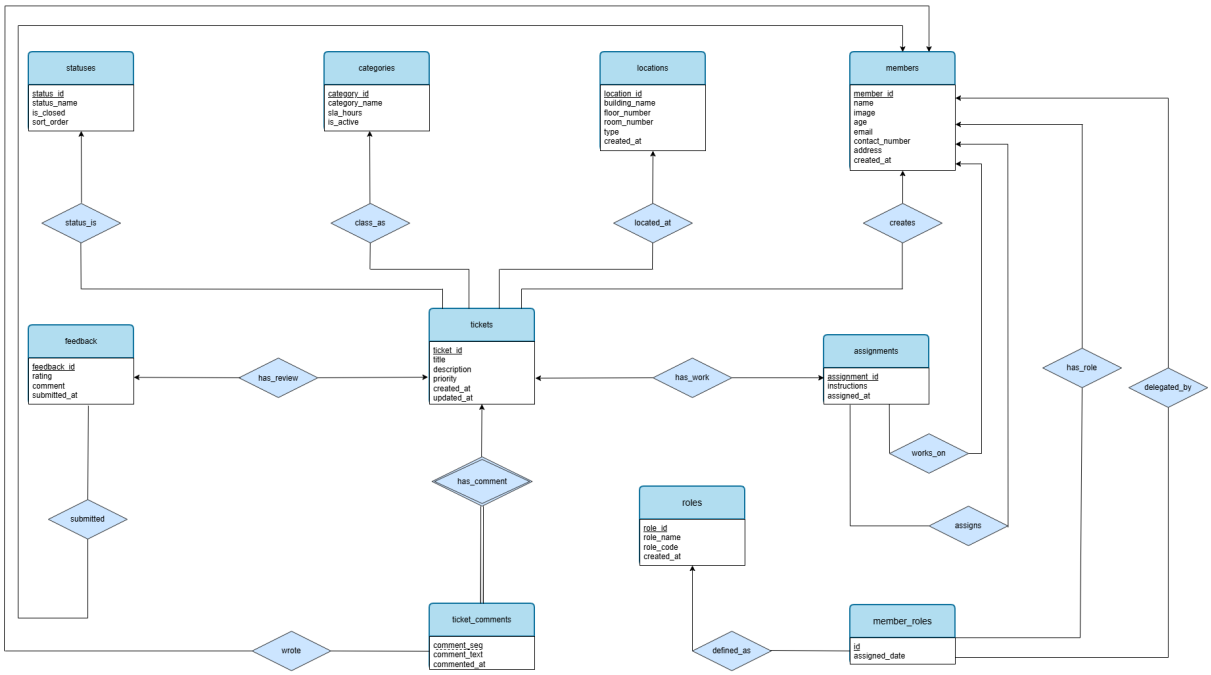

Figure 6: Entity-Relationship Diagram for FixIIT System

5.2 Notation Legend

To ensure clarity, the following notation standards (Silberschatz) are used in the diagram above:

- **Single Rectangle:** Represents a **Strong Entity** (e.g., `Members`, `Tickets`).
- **Single Diamond:** Represents a standard relationship between entities.
- **Double Diamond & Double Line:** Represents an **Identifying Relationship** with **Total Participation** (used for the Weak Entity `Ticket_Comments`).
- **Arrow (→):** Represents the **"One"** side of a relationship (Cardinality 1).
- **Plain Line (—):** Represents the **"Many"** side of a relationship (Cardinality N).

- **Dashed Underline:** Indicates the **Discriminator** (Partial Key) of a Weak Entity.

5.3 Schema-to-ER Transition & Justification

Since the project followed a schema-first approach (Module A), the ER diagram was derived by reverse-engineering the SQL constraints into conceptual models.

5.3.1 Entity Classifications

- **Strong Entities:** Entities like `Members`, `Tickets`, and `Locations` have independent Primary Keys (e.g., `member_id`) and exist regardless of other data.
- **Weak Entity (`Ticket_Comments`):** Modeled as a weak entity because a comment cannot exist without a parent ticket.
  - **Constraint:** `PRIMARY KEY (ticket_id, comment_seq)`
  - **Visual:** Connected to `Tickets` via a **Double Diamond** (`has_comment`) and a **Double Line**, indicating that every comment must belong to a ticket (Total Participation).
- **Associative Entity (`Member_Roles`):** While conceptually a Many-to-Many relationship between Members and Roles, it is modeled as an entity because it contains relationship-specific attributes (`assigned_date`) and a distinct surrogate key (`id`).

5.3.2 Relationship Justifications

The cardinalities and participation constraints in the diagram are justified as follows:

1. **One-to-Many (1:N) [Arrow → Line]:**
   - *Example:* `Member` → `creates` — `Ticket`.
   - *Justification:* A single Member (One) can create multiple Tickets (Many). However, a specific Ticket is created by exactly one Member (Mandatory participation on the Ticket side).
2. **One-to-One (1:1) [Arrow → Arrow]:**
   - *Example 1:* `Ticket` ↔ `Assignment`.
   - *Justification:* This represents a **0..1 to 1** relationship. A ticket may have **zero or one** active assignment (it starts unassigned). However, an assignment MUST belong to exactly one ticket. The unique constraint on `ticket_id` enforces the maximum cardinality of 1.
   - *Example 2:* `Ticket` ↔ `Feedback`.
   - *Justification:* This is also a **0..1 to 1** relationship. A ticket is not required to have feedback (it can be 0), but if feedback exists, it is strictly linked to that single ticket. The double arrows in the diagram denote that the *maximum* cardinality is 1 in both directions.
3. **Recursive/Dual Relationships:** The `Members` entity connects to `Assignments` twice, representing two distinct contexts:

- **Technician Context:** `works_on` (Member who performs the job).
- **Manager Context:** `assigns` (Member who authorizes the job).

5.4 Database Normalization Analysis

The database schema has been designed to strictly adhere to the Third Normal Form (3NF) to reduce redundancy and ensure data integrity.

5.4.1 First Normal Form (1NF)

Requirement: Atomicity of attributes and unique identification of rows.

- All columns in the tables contain atomic values (e.g., `address` is stored as a single string, no comma-separated lists for contacts).
- Every table has a defined Primary Key (e.g., `member_id`, `ticket_id`) ensuring row uniqueness.

5.4.2 Second Normal Form (2NF)

Requirement: No partial dependencies (Non-prime attributes must depend on the *whole* candidate key).

- For tables with single-column Primary Keys (e.g., `Tickets`, `Members`), 2NF is automatically satisfied.
- For the Weak Entity `ticket_comments` (PK: `ticket_id, comment_seq`), the non-prime attribute `comment_text` depends on the specific combination of the ticket and the sequence number, not just one of them. Thus, 2NF is satisfied.

5.4.3 Third Normal Form (3NF)

Requirement: No transitive dependencies (Non-prime attributes must depend *only* on the Primary Key).

- In the `Tickets` table, attributes like `priority` or `description` depend solely on `ticket_id`. They do not depend on `category_id` or `location_id`.
- Information related to locations (e.g., `building_name`) is moved to a separate `Locations` table, referenced only by `location_id`. This prevents transitive dependency anomalies (e.g., updating a building name in one place reflects everywhere).
- **Conclusion:** The schema successfully achieves 3NF compliance.

5.5 Additional Constraints

The ER diagram implicitly supports the logical constraints defined in the SQL schema:

- **Domain Constraints:** Attributes like `priority` (in Tickets) and `type` (in Locations) are restricted to specific domain values (e.g., 'Urgent', 'Classroom').

- **Temporal Constraints:** The `created_at` and `updated_at` timestamps in the `Tickets` entity ensure logical chronology (a ticket cannot be updated before creation).
- **Total Participation:** The double line connecting `Ticket_Comments` ensures that no orphan comments exist in the system.

6 Team Contributions

Member Name	**Contribution**
Abhitej Singh Bhullar	20%
Shiv Patel	20%
Shivansh Shukla	20%
Soham Shrivastava	20%
Arin Mehta	20%

Table 1: Distribution of Work for Module A & B