National Economics University
Faculty of Data Science and Artificial Intelligence

PROJECT 2: EMPLOYEE INFORMATION MANAGER

FINAL REPORT

Team Members

Full Name	Student's ID
Chử Vũ Thảo Hiền	11247287
Phan Thị Anh Quỳnh	11247347
Dương Đức Anh	11247257

Supervisor: Tran Hung

December 2025

Abstract

This project presents the design and implementation of an **Employee Information Manager**, a complete relational database system built from an initially unnormalized data set. Based on the UNF table provided and functional dependencies, the database was systematically normalized through **1NF → 2NF → 3NF**, resulting in a stable and anomaly-free schema. The final model consists of four main entities—Employees, Departments, Projects, and Assignments—defined with proper primary keys, foreign keys, and integrity constraints.

Based on the designed 3NF schema, the team implemented the database on **MySQL**, populated it with realistic sample data, and developed a full **Python GUI application** to support CRUD operations, multi-table analytical queries, data visualization, global search, and interactive filtering. The project also follows professional software engineering practices including GitHub version control, documentation, environment configuration, and a demonstration video. This report summarizes the normalization process, implementation decisions, and system functionality, as well as key insights gained throughout the development cycle.

Contents

1 Introduction

1.1 Background and Motivation

In many organizations, employee-related information is stored in large spreadsheets that mix personal data, department names, project assignments, salaries, and managerial relationships. Although this approach seems convenient, it often leads to redundancy, update anomalies, inconsistent reporting, and difficulties in maintaining data accuracy. Without proper normalization, the system becomes increasingly unreliable as the company grows. This project addresses these real-world issues by transforming the initial unnormalized form (UNF) into a clean, scalable, and fully normalized **Third Normal Form (3NF)** relational schema. The resulting database supports reliable CRUD operations, analytical queries, and visualization features designed for HR personnel and project managers.

1.2 Project Objectives

The goals of this project are as follows:

- **Data Modeling:** Convert the provided UNF structure into a normalized 3NF schema using functional dependencies.
- **Database Implementation:** Build a complete MySQL schema with primary keys, foreign keys, constraints, and realistic sample data.
- **Application Development:** Develop a Python GUI application providing CRUD functionality, multi-table queries, dashboard statistics, search, filtering, and CSV export.
- **Engineering Workflow:** Use GitHub to manage the project with proper repository structure, commits, pull requests, and documentation.
- **Communication:** Prepare a LaTeX report, slide deck, and YouTube video demonstrating the system.

1.3 Scope of Work

The scope of this project covers:

- Designing and normalizing the relational schema from UNF to 3NF.
- Implementing all tables using MySQL and generating sample data through `seed.sql`.
- Building a complete, user-friendly Python GUI for database operations and visualization.
- Running analytical queries such as INNER JOIN, LEFT JOIN, multi-table joins, and salary comparisons.
- Producing comprehensive documentation and media for submission.

Out-of-scope features include authentication, high-performance database tuning, cloud deployment, and non-relational database systems, as stated in the project specifications.

2 UNF to 3NF Normalization Process

2.1 Initial UNF Structure

In the initial dataset, all employee-related information was stored in a single unnormalized (UNF) table. This structure contained mixed attributes such as personal details, department information, project assignments, and salary data within the same record. Such design leads to redundancy, inconsistent updates, and anomalies.

The original UNF table consisted of the following attributes:

- EmployeeID
- Name
- DateOfBirth
- Department
- Project
- Manager
- Role
- Salary

Functional Dependencies (FDs) identified from the dataset:

- EmployeeID $\rightarrow$ {Name, DateOfBirth, Department}
- {EmployeeID, Project} $\rightarrow$ {Role, Salary}
- Project $\rightarrow$ Manager

2.2 Conversion to 1NF

To achieve First Normal Form (1NF), all attributes must be atomic and free of repeating groups. The UNF table was restructured so that each tuple represented a single Employee–Project assignment. The resulting 1NF table uses the composite key {EmployeeID, Project}.

2.3 Conversion to 2NF

A relation is in 2NF when:

1. It is already in 1NF
2. All non-key attributes fully depend on the whole primary key

Partial dependencies identified:

- EmployeeID → {Name, DateOfBirth, Department}
- Project → Manager

These were removed by decomposing the table into:

- EMPLOYEES_INFO (dependent solely on EmployeeID)
- PROJECTS_INFO (dependent solely on Project)
- ASSIGNMENTS (dependent on the full key {EmployeeID, Project})

2.4 Conversion to 3NF

To satisfy Third Normal Form (3NF), the schema must not contain transitive dependencies. Two transitive dependencies were observed:

- DepartmentName depends on an implicit DepartmentID
- Manager depends on Project but should reference an EmployeeID in the EMPLOYEES table

These were resolved by introducing:

- DEPARTMENTS table
- Foreign key ManagerEmployeeID referencing EMPLOYEES(EmployeeID)

2.5 Final ERD

The final 3NF schema includes four core entities:

- **DEPARTMENTS**: DepartmentID (PK), DepartmentName (UNIQUE)
- **EMPLOYEES**: EmployeeID (PK), DepartmentID (FK), Name, DateOfBirth
- **PROJECTS**: ProjectID (PK), ManagerEmployeeID (FK), ProjectName (UNIQUE)
- **ASSIGNMENTS**: AssignmentID (PK), EmployeeID (FK), ProjectID (FK), Role, Salary

The ERD visually represents all relations and ensures referential integrity using primary and foreign keys.

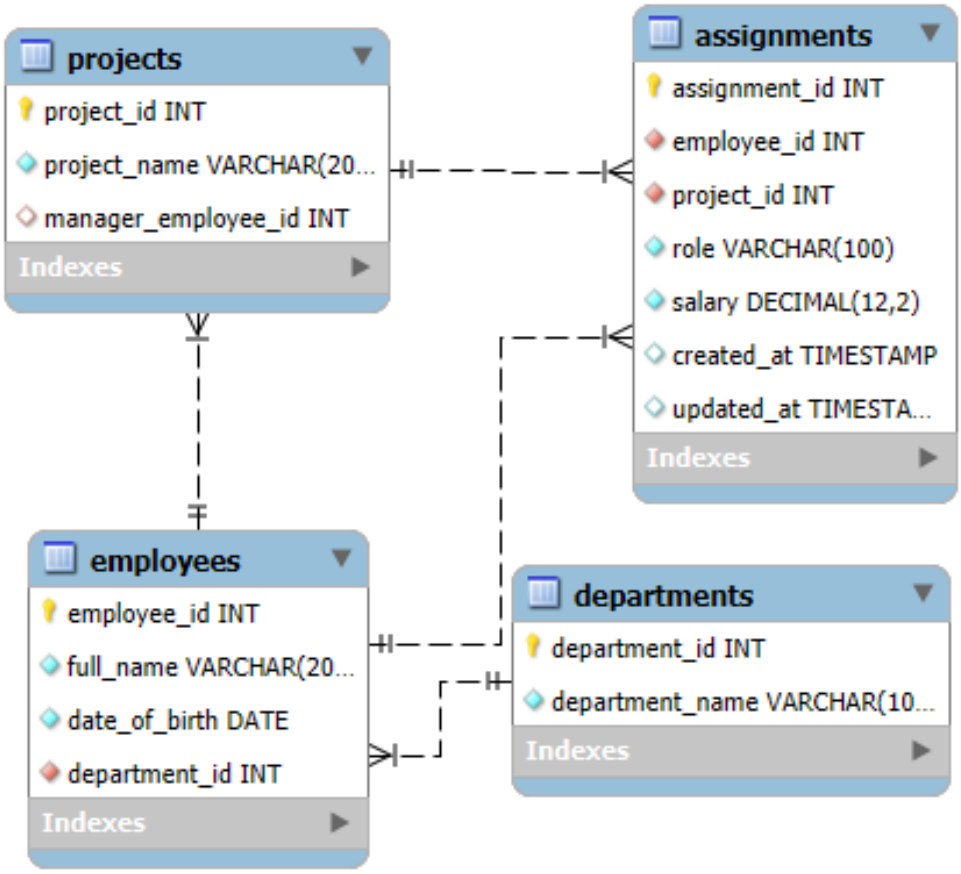

Figure 1: Entity–Relationship Diagram (ERD)

3 ERD Relationships

The ERD diagram uses Crow's Foot notation to model the relationships derived from the Functional Dependencies (FDs):

3.1 Departments Employees (1:N)

This is the *Works In* relationship. A department can have multiple employees (Crow's Foot notation |<), but each employee belongs to exactly one department (||).

3.2 Employees Projects (1:N)

This is the *Manages* relationship. An employee (in the manager role) can manage multiple projects (0<), but each project is managed by exactly one employee (||).

3.3 Employees Projects (N:M) via Assignments

This is the *Assigned To* relationship. The many-to-many relationship between Employees and Projects is normalized using the **Assignments** table. Instead of a direct N:M connection, the model is implemented as two 1:N relationships:

- Employees → Assignments
- Projects → Assignments

The Assignments table stores detailed information such as the employee's role and salary for each project participation.

4 Database Implementation (schema.sql & seed.sql)

4.1 Table Structures

The database was implemented in MySQL following the finalized 3NF schema. The four main tables created in `schema.sql` are:

- **DEPARTMENTS(DepartmentID, DepartmentName)**
- **EMPLOYEES(EmployeeID, Name, DateOfBirth, DepartmentID)**
- **PROJECTS(ProjectID, ProjectName, ManagerEmployeeID)**
- **ASSIGNMENTS(AssignmentID, EmployeeID, ProjectID, Role, Salary)**

Each table was designed with appropriate attribute types (e.g., `INT`, `VARCHAR`, `DATE`, `DECIMAL`), ensuring consistency and accurate data representation.

4.2 Integrity Constraints (PK, FK, UNIQUE, CHECK)

To ensure data integrity, a system of constraints was applied:

- **Primary Keys (PK)** are used to uniquely identify each record.
- **Foreign Keys (FK)** enforce referential integrity across tables.
- **UNIQUE** constraints prevent duplicated names where required (e.g., DepartmentName, ProjectName).
- **CHECK** constraints ensure valid business rules, such as:
  - Salary > 0
  - DateOfBirth $<$ CURRENT_DATE

These constraints prevent update, insertion, and deletion anomalies while ensuring consistency across the database.

4.3 Sample Data Generation

The `seed.sql` file populates the database with realistic synthetic data for testing and demonstration purposes.

Sample data includes:

- 5–8 departments
- 150+ employees with diverse demographics
- 6–10 projects with assigned managers
- 400–800 assignment rows linking employees to projects

Randomized yet controlled data generation was applied to ensure:

- Balanced distribution of employees across departments
- Valid manager references
- Salary ranges consistent with industry standards
- Multiple project assignments per employee

5 Application Design & Architecture

5.1 Technology Stack

The application is built using the following technologies:

- Python — core programming language of the project
- Tkinter — GUI framework for developing desktop interfaces
- MySQL Connector/Python — database communication layer
- Matplotlib — visualization library for dashboard charts
- Pandas — used for data processing and CSV export

5.2 Folder Structure

The project directory is organized into clearly separated modules:

- **database/**
  - contains the schema definition file (`schema.sql`)
  - contains the sample data generation file (`seed.sql`)
- **src/**
  - **db_connection.py**: centralized database connection manager
  - **models/**: contains logic for each database entity
    - employees model
    - departments model
    - projects model
    - assignments model
  - **ui/**: graphical interface components
    - main application window
    - CRUD forms for all entities
    - dashboard and visualization screens
- **assets/**
  - icons, images, and static resources used in the GUI
- **README.md**: documentation for installation and usage

5.3 Database Connection Layer

The database connection layer abstracts all communication with MySQL. It provides:

- a centralized connection interface
- reusable query execution functions
- built-in error handling
- transaction-safe operations

5.4 GUI Design (Forms, Tables, Validation)

The graphical interface is designed with usability and clarity:

- intuitive CRUD forms with input validation
- scrollable tables for viewing and editing records
- dashboard visualizations showing project and employee statistics
- search and filter tools applied to all major entities

6 Functional Features

6.1 CRUD Interfaces

The application provides complete CRUD operations for the required entities:

- **Employees**: Add, edit, delete, and list employee profiles.
- **Departments**: Manage department names and IDs.
- **Projects**: Create and update projects, including manager assignment.
- **Assignments**: Manage employee–project relationships, roles, and salaries.

All CRUD screens follow a consistent design with responsive forms and clean navigation.

6.2 Required Analytical Queries

A dedicated section in the GUI presents the four required analytical queries:

1. **INNER JOIN**: Displays employee name, role, and salary for each project.
2. **LEFT JOIN**: Lists all employees, including those without assigned projects.
3. **Multi-Table JOIN (3+ tables)**: Combines employees, projects, assignments, and manager information.
4. **Above Global Average Salary**: Retrieves employees whose average assignment salary is higher than the overall average across all assignments.

Each query screen includes:

- A scrollable results table
- Sorting capabilities
- A button to export the results as CSV

These analytical features enable HR and project managers to gain actionable insights.

6.3 Dashboard Visualizations

The dashboard aggregates key performance indicators (KPIs) and charts:

- Total number of employees, departments, projects, and active assignments
- Average salary across all assignments
- A histogram of salary distribution
- A Top-N ranking of employees by average salary
- Role distribution chart

Charts are generated dynamically using Matplotlib and updated automatically based on current database contents.

6.4 Search & Filter Mechanisms

A global search bar and advanced filtering options allow users to:

- Search employee names
- Filter by department, project, role, or manager
- Restrict results by salary range
- Combine multiple filters at once

These features improve data navigation and support efficient information retrieval.

7 Testing and Reliability

7.1 Input Validation

Ensuring data correctness is a core non-functional requirement of the application. Comprehensive validation mechanisms were implemented across all forms.

Field-Level Validation

- **Required fields**: The application rejects submissions with empty fields for critical attributes such as Employee Name, Project Name, Role, or Department.
- **Date validation**: The Date of Birth field must follow a valid date format. Incorrect or future dates trigger descriptive warning messages.
- **Numerical constraints**: Salary values must be numeric and fall within an acceptable range (e.g. Salary ≥ 0). Non-numeric input is blocked.
- **Dropdown completeness**: Department and Project fields require selecting from predefined lists generated from the database.

Relational Validation

- Duplicate logical keys are prevented, such as:
  - Duplicate EmployeeID
  - Duplicate ProjectName or DepartmentName
  - Duplicate Employee–Project pairs in the Assignments table
- Foreign keys are validated to ensure:
  - Assigned managers exist in the Employees table
  - Employees belong to valid departments
  - Assignments reference existing employees and projects

These validation mechanisms ensure consistency and prevent database anomalies.

7.2 Error Handling

The application implements structured, user-friendly error handling mechanisms:

- **Database errors**: Issues such as connection failures, foreign key violations, or incorrect SQL syntax are displayed using descriptive dialogue boxes instead of crashing the program.
- **Input errors**: Invalid data triggers inline GUI warnings and prevents form submission.
- **Exception logging**: Unexpected exceptions are recorded in a local log file to support debugging.
- **Graceful failures**: When the database is unavailable, the system notifies the user and remains responsive using retry logic.

This design enhances reliability and ensures stable operation under unexpected conditions.

7.3 Sample Test Scenarios

Representative test cases used to validate the system include:

1. **Employee creation with invalid date** Input: "32/15/2020" Expected: Warning message; record not saved.
2. **Duplicate department creation** Action: Create "Finance" twice Expected: UNIQUE constraint warning.
3. **Assignment creation with missing salary** Expected: Validation error preventing submission.
4. **LEFT JOIN query with no assignments** Setup: New employee with no projects Expected: Listed with NULL role and salary.
5. **Changing a project's manager to a non-existing ID** Expected: Foreign key violation error.
6. **Dashboard update after adding assignments** Expected: KPIs and charts refresh automatically.

These tests confirm coordination between GUI logic and database constraints.

8 GitHub Workflow & Collaboration

8.1 Repository Structure

The GitHub repository follows a clean and professional structure aligned with modern software engineering practices.

- **Main Branch (main)**: Contains stable, production-ready code.

- **Development Branches**: Each feature was developed in separate branches following the feature-branch workflow.

The repository includes:

- `.env.example` template
- `requirements.txt` for reproducible environments
- `docs/` folder for slides and report materials
- `tests/` folder containing test scripts and validation tools

This structure enhances clarity, modularity, and maintainability.

8.2 Issues, PRs, and Versioning

GitHub issue tracking and pull requests were used extensively for collaboration.

Issues

- Each task (GUI creation, SQL queries, dashboard charts, bug fixes) was tracked as an issue.
- Issues were labeled (bug, enhancement, documentation, testing) and assigned to team members.

Pull Requests

- All major features were submitted via PRs.
- Each PR was reviewed before merging to ensure quality.
- PR descriptions included summaries, linked issues, screenshots, and testing notes.

Version Tags

- Version **v1.0.0** marks the stable release submitted for evaluation.

This workflow improved code quality, traceability, and reproducibility.

9 Results & Discussion

The completed Employee Information Manager demonstrates the full lifecycle of data modeling, implementation, and application development.

The transformation from an unnormalized dataset into a fully normalized 3NF schema eliminated redundancy, update anomalies, and inconsistent reporting.

The Python GUI provides an intuitive environment for managing employees, projects, and assignments. CRUD operations behave reliably with strong validation and foreign key enforcement. Analytical queries offer meaningful insights into organizational roles and compensation.

Dashboard visualizations deliver real-time metrics such as employee counts, average salaries, and role distributions, enabling HR and project managers to assess workforce conditions effectively.

Overall, the project demonstrates the successful integration of relational design, SQL implementation, GUI programming, and analytic visualization to address practical data management challenges.

10 Limitations & Future Work

10.1 Current Limitations

Despite meeting all core requirements, several limitations remain:

1. **No Authentication or Role-Based Access Control**: The application operates in open-access mode, suitable for coursework but not secure enough for production.

2. **Basic Visualizations**: Dashboard charts are generated using simple Matplotlib components. More advanced or interactive tools would enhance analytics.

3. **Limited Salary Modeling**: Salaries are stored directly in the Assignments table. Real-world scenarios may require historical tracking, currency conversion, or layered compensation structures.

4. **Static Forms and Tables**: Tkinter provides stable but limited GUI capabilities compared to modern frameworks like PyQt or web-based interfaces.

10.2 Future Enhancements

Potential improvements for future iterations include:

1. **User Authentication**: Implementing login screens and role-based permissions (HR, Manager, Admin).

2. **Historical Data Tracking**: Incorporating temporal tables or audit logs to track changes over time.

3. **More Complex Analytics**: Future analytic modules may include:

   - Salary forecasting
   - Project workload prediction
   - Department-level performance metrics

4. **Web-Based Deployment**: Migrating to Flask or FastAPI would improve scalability, accessibility, and interface flexibility.

5. **Unit and Integration Testing**: Automated testing pipelines (via GitHub Actions) would ensure long-term maintainability and code reliability.

11 Conclusion

This project demonstrates the complete development of a data-driven application—from data normalization and relational schema design to GUI implementation, analytical queries, and dashboard visualization.

By converting an unstructured UNF dataset into a clean 3NF schema, the system eliminates redundancy and ensures data consistency. The MySQL backend, integrated with a Python/Tkinter GUI, provides robust CRUD functionality, analytical insights, and dynamic filtering capabilities.

The inclusion of GitHub-based workflow practices—feature branches, pull requests, issue tracking, and version tags—aligns the project with modern software engineering standards. The final system meets all acceptance criteria and provides a strong foundation for future expansion, including authentication, web deployment, and advanced analytics.

Overall, the Employee Information Manager showcases technical proficiency across database theory, implementation, and practical software development.

12 References

1. Silberschatz, A., Korth, H. F., & Sudarshan, S. *Database System Concepts*. McGraw-Hill.
2. Connolly, T., & Begg, C. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson.
3. MySQL Documentation — https://dev.mysql.com/doc/
4. Python Official Documentation — https://docs.python.org/
5. Tkinter Programming Guide — https://tkdocs.com/
6. Matplotlib Documentation — https://matplotlib.org/
7. GitHub Documentation — https://docs.github.com/

Appendix

Additional materials such as system screenshots, SQL schema diagrams, and test outputs may be referenced in the appendix section of the full report.