University of Liverpool

CSCK542 Databases and Information Systems August 2025 B

End of Module Assignment – 2

Submitted by

Group B

University Record Management System (URMS)

Full Name	Role/Tasks	Teams
Nduta, Tabitha Elsie	(Back End) Database Engineer;	Team A
Issa, Shafi Suleiman	(Back End) Database Engineer;	
Chan, Man Man	(Front End) Software Engineer;	Team B
Kerai, Nikul	(Front End) Software Engineer; Tester	
Adepoju, Samuel Temitayo	Project Manager;	Team C

Word Count: 1498

13/10/2025

Table of Contents

Table of Figures

1 Introduction

1.1 Problem statement and context

This project presents the design and implementation of a University Record Management System (URMS) developed to streamline the administration of academic and personnel data within a higher education environment. This initiative applies relational design, normalization, and solid software engineering to deliver an optimized, query-driven system that aids universities manage sprawling academic, administrative, and research data while eliminating redundancy, inconsistency, and poor accessibility.

1.2 Purpose and scope

Thus, the project's primary objective is to design and implement a database-centric record management solution capable of managing heterogeneous university datasets, including students, lecturers, non-academic staff, and course modules. The system's scope encompasses schema design, data population, SQL query execution, and Python-based interface development. This report outlines the analytical, technical, and procedural steps undertaken—from conceptual design to testing—demonstrating the system's functionality, reliability, and alignment with academic data management standards.

2 Project Management Process

The project followed a structured, time-bound development cycle spanning 24 September to 4 October 2025, adopting a phased approach to ensure effective coordination between database, software, and documentation teams (Figure2.1) See full structure in Appendix A.

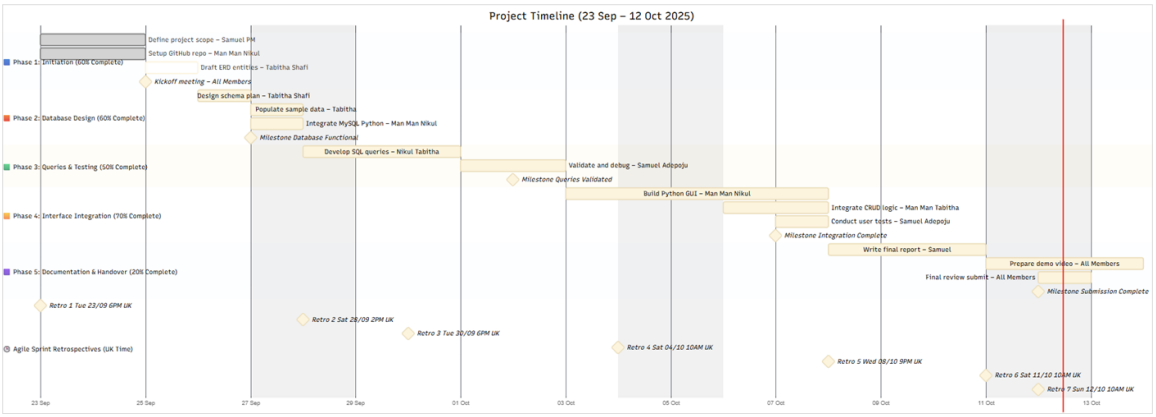


Figure 2.1: Project workflow

The workflow combined parallel tasking with sequential dependencies, allowing concurrent development while maintaining integration milestones. A hybrid Agile–Waterfall model was applied.

2.1 Project Organization, roles and responsibilities and Collaboration tools

To ensure specialization and accountability, the project was divided into three primary teams:

Each team coordinated through a shared GitHub repository, with sync-ups on Microsoft Teams to update progress, resolve dependencies, and document version changes. (see roles-and-responsibilities for more detail.

2.2 Work breakdown structure and scheduling

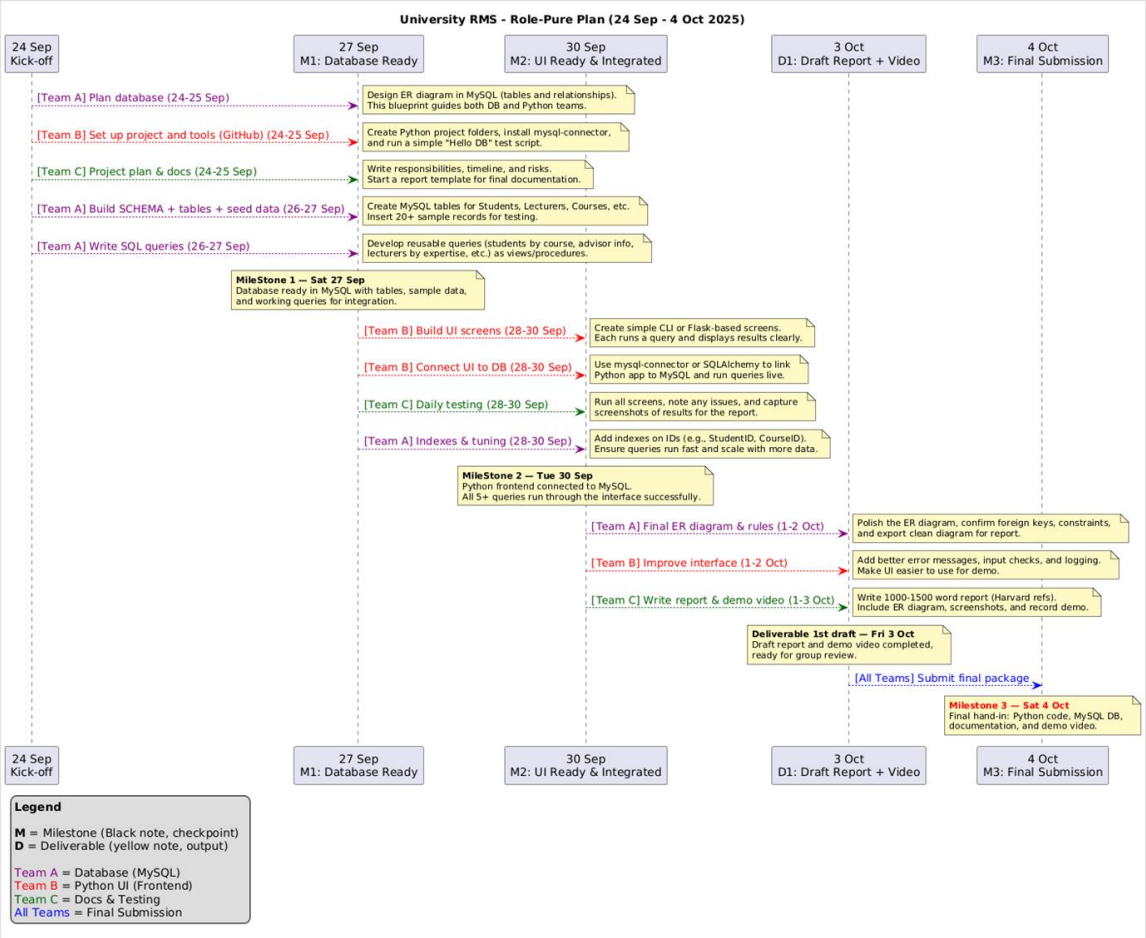


Figure 2.2: Workflow and WBS with timelines

2.3 Risks and mitigation employed

Risks encountered during the project involved scope and planning risks involving ambiguous requirements and limited stakeholder availability influencing rework, delays, and pressure on milestones, were mitigated through regular scope clarifications with the lecturer and structured, time-boxed adaptive sprint meetings and decisions. Further technical risks encountered were intermittent Python–MySQL/GUI integration failures, dependency/version drift, related to OS dependencies, resolved through OS platform swapping.

3 Backend Design, Development and Testing

3.1 Database schema and ERD Design

Back-end development began by designing a normalized relational schema for university operations.

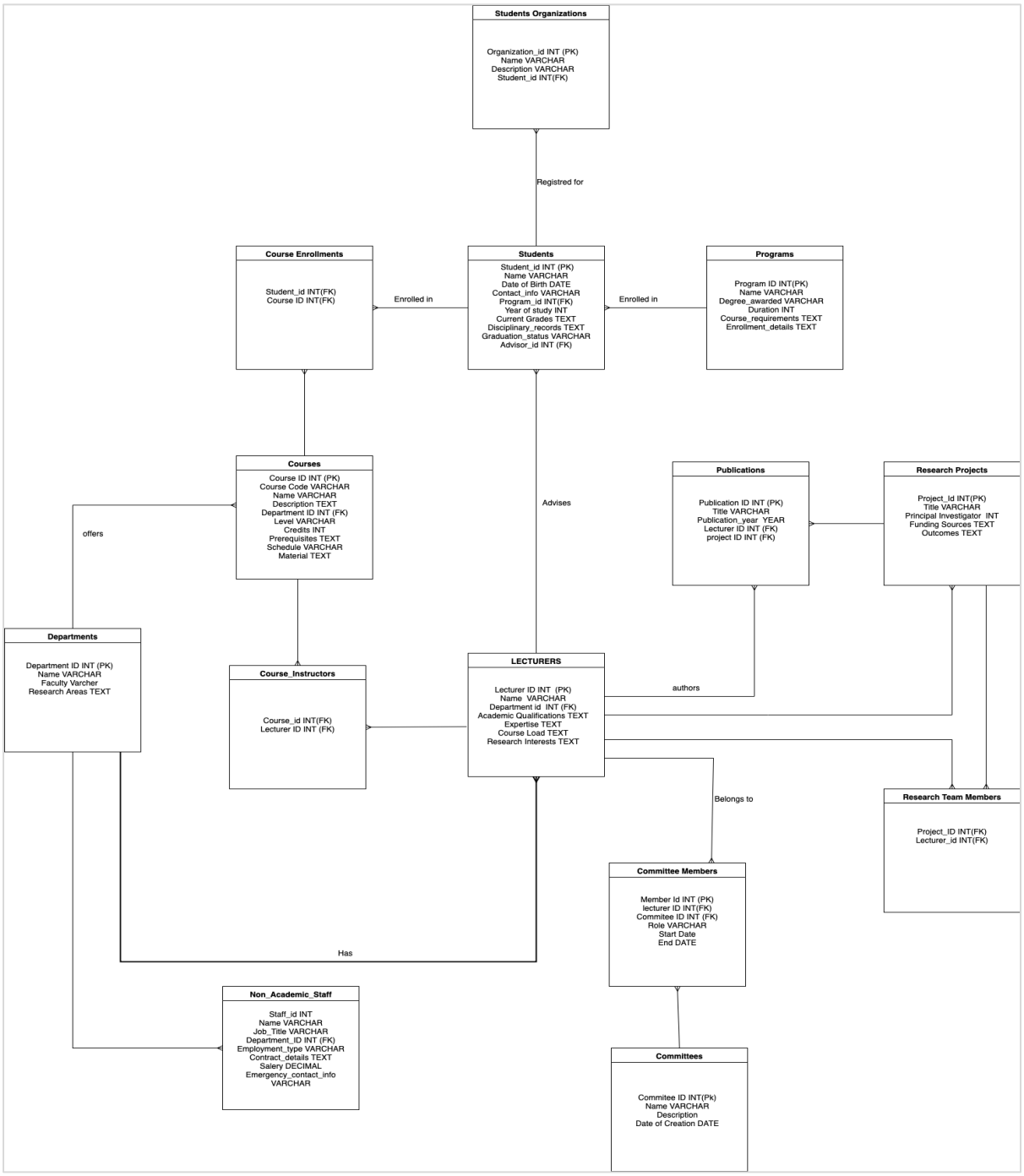


Figure 3.1: ERD Mapped to Queries and Keys

The ERD in Figure 3.1 models core entities—Students, Lecturers, Departments, Programs, Courses, Research Projects, Publications, Committees, and Non-Academic Staff—with associative tables

preserving many-to-many links (Grambow & Ruttmann, 2023) see detailed notation in Crowfoot notation Table in Appendix A. These relationships were built on six core queries:

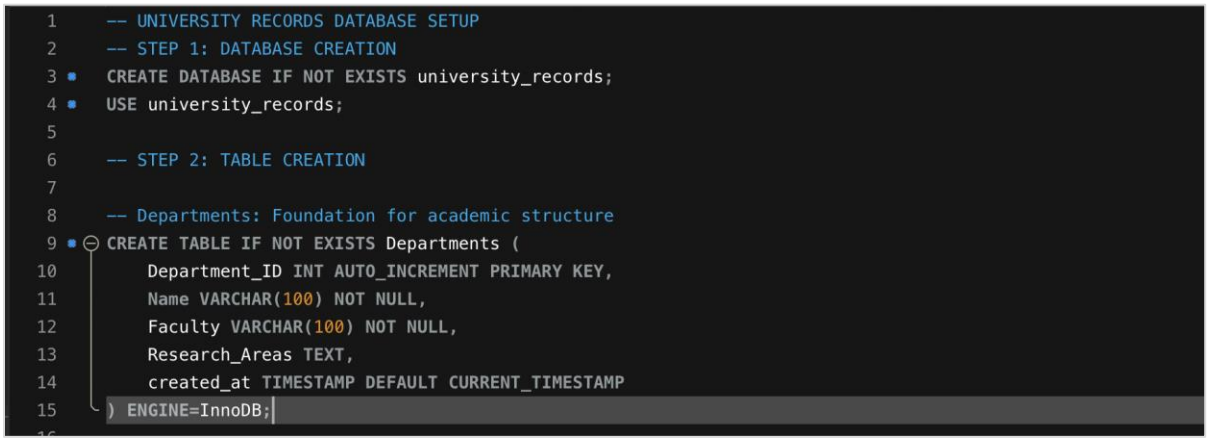
- (Query1) Find all students enrolled in a specific course taught by a particular lecturer.
- (Query2) List all students with an average grade above 70% who are in their final year of studies.
- (Query3) Identify students who haven't registered for any courses in the current semester.
- (Query6) List all courses taught by lecturers in a specific department.
- (Query7) Identify lecturers who have supervised the most student research projects.
- (Query10) Find all staff members employed in a specific department.

3.2 Backend Technologies

The backend was implemented on MySQL, designed in MySQL Workbench and deployed from the approved schema. Database access, CRUD operations, and query execution were performed in Python via MySQL Connector, while Visual Studio facilitated front-end development, integrated testing, and validation.

3.3 Query Implementation workflow

The team authored the MySQL 8.0 schema in Workbench using SQL DDL (Figure 3.2) and normalized it to 1NF/2NF to reduce redundancy and improve performance (McLaughlin, 2013).



```
-- UNIVERSITY RECORDS DATABASE SETUP
-- STEP 1: DATABASE CREATION
CREATE DATABASE IF NOT EXISTS university_records;
USE university_records;

-- STEP 2: TABLE CREATION

-- Departments: Foundation for academic structure
CREATE TABLE IF NOT EXISTS Departments (
    Department_ID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Faculty VARCHAR(100) NOT NULL,
    Research_Areas TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

Figure 3.2: Schema Creation Script (Database and Departments Table)

Surrogate primary keys, explicit foreign keys with ON UPDATE/DELETE rules, and uniqueness/indexes on natural identifiers were defined to enforce integrity and speed joins. All base tables and junction tables (enrolments, instructors, research teams) were created, followed by six representative queries exercising joins, constraints, and indexed predicates. Dummy data was seeded under transaction control, then validated via INFORMATION_SCHEMA checks, cascade tests, and sample execution plans (Figures 3.3–3.5). Functional checks confirmed students with programme advisors, courses with instructors, and research projects by principal investigator with team sizes (Figures 3.6–3.8). Final Workbench logs and Python connector smoke tests verified seed

load and query execution (Figure 3.9), yielding an integrity-checked, performant, query-ready database

3.3.1 Constructing a database in URMS (DBMS)

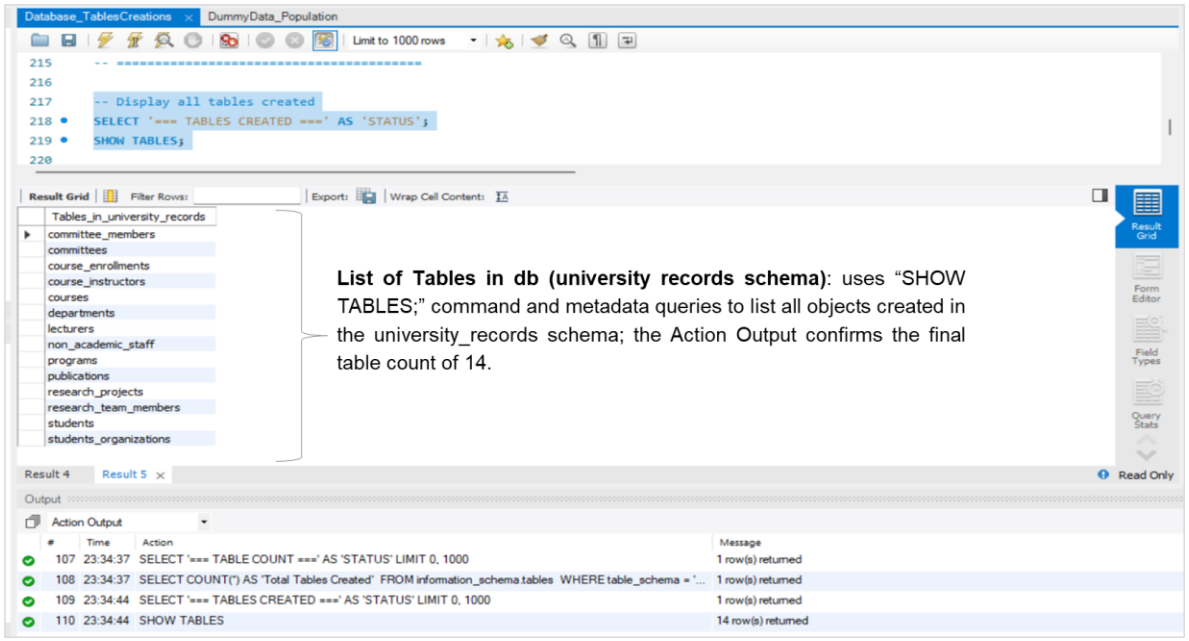


Figure 3.3: List of Tables in university records schema

3.3.2 Populating dummy data

With the structure verified, the schema was populated with dummy data

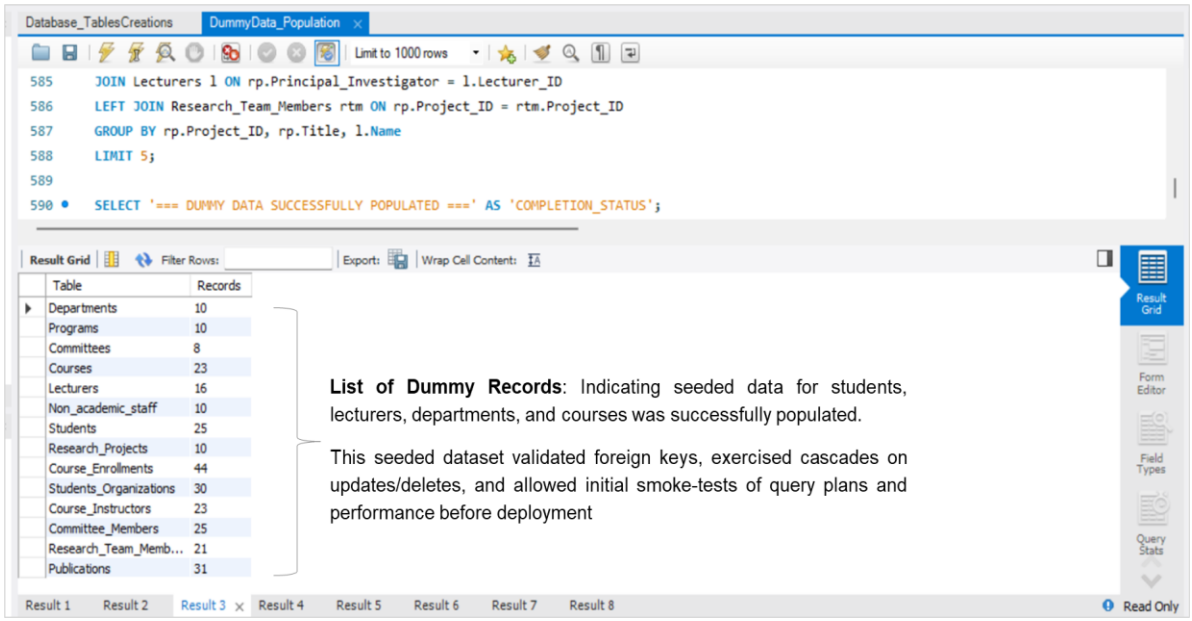


Figure 3.4: List of Dummy Records

3.3.3 Table Structure Keys

```sql
-- Optional: Show the table structure with foreign keys
SELECT
    COLUMN_NAME,
    DATA_TYPE,
    IS_NULLABLE,
    COLUMN_KEY,
    COLUMN_DEFAULT,
    EXTRA
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'university_records'
    AND TABLE_NAME = 'Students'
ORDER BY ORDINAL_POSITION;
```

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

COLUMN_NAME	DATA_TYPE	IS_NULLABLE	COLUMN_KEY	COLUMN_DEFA
Student_id	int	NO	PRI	NULL
Name	varchar	NO		NULL
Date_of_birth	date	NO		NULL
Contact_info	varchar	YES		NULL
Program_id	int	YES	MUL	NULL
Year_of_study	int	YES		NULL
Current_Grades	text	YES		NULL
Disciplinary_records	text	YES		NULL
Graduation_status	varchar	YES		Active
Advisor_id	int	YES	MUL	NULL
created_at	timestamp	YES		CURRENT_TIMEST... DEFAULT_GEN...

The INFORMATION_SCHEMA.COLUMNS query inspects the Students table's metadata to verify its structural and integrity constraints. It lists each column's name, data type, nullability, key role, default, and "extra" attributes—confirming Student_id is the primary key with AUTO_INCREMENT for unique, gap-tolerant identifiers. It also shows Program_id and Advisor_id are foreign-key columns (indexed) that reference Programs and Lecturers, respectively, enforcing referential integrity and preventing orphaned records.

Figure 3.5: Table Structure with Foreign Keys

```sql
-- ========================================
-- QUERY 1: Find all students enrolled in a specific course taught by a particular lecturer
-- ========================================
SELECT DISTINCT
    s.Student_ID,
    s.Name AS Student_Name,
    s.Contact_info,
    c.Course_Code,
    c.Name AS Course_Name,
    l.Name AS Lecturer_Name,
    ce.Semester,
    ce.Academic_year,
    ce.Status AS Enrollment_Status
FROM Students s
JOIN Course_Enrollments ce ON s.Student_ID = ce.Student_id
```

Query (1): Find all students enrolled in a specific course taught by a particular lecturer.
This resolves the many-to-many enrollment link by joining Students → Course_Enrollments (junction) → Courses, constrained by the Lecturers teaching that course. It returns Student_ID/Name/Contact, Course_Code/Name, Lecturer_Name, Semester, Academic_year, and Enrollment_Status—one row per student–course match. Example result: Alice Johnson enrolled in CS201 – Data Structures and Algorithms taught by Prof. James Chen, Fall 2024–2025, status Enrolled.

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

Student_ID	Student_Name	Contact_info	Course_Code	Course_Name	Lecturer_Name	Semester	Academic_year	Enrollment_Status
1	Alice Johnson	alice.j@student.edu, 555-1001	CS201	Data Structures and Algorithms	Prof. James Chen	Fall	2024-2025	Enrolled

Figure 3.6:Query(1)—Result

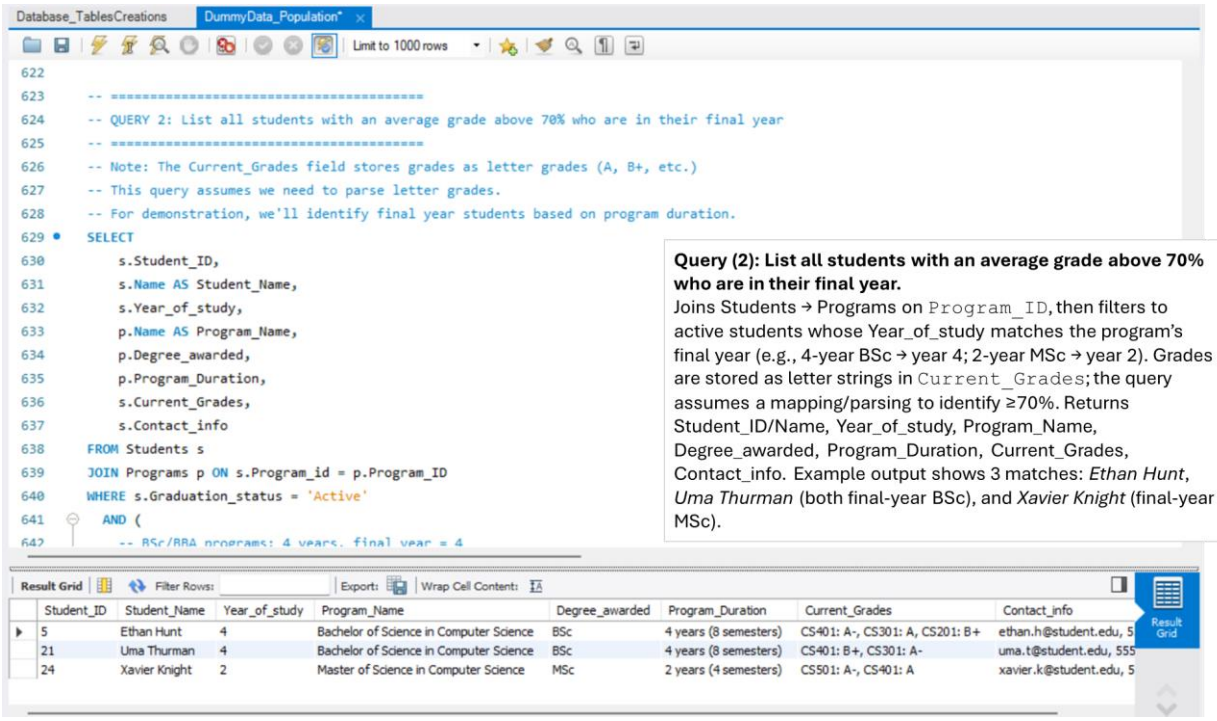


Figure 3.7:Query(2)—Result

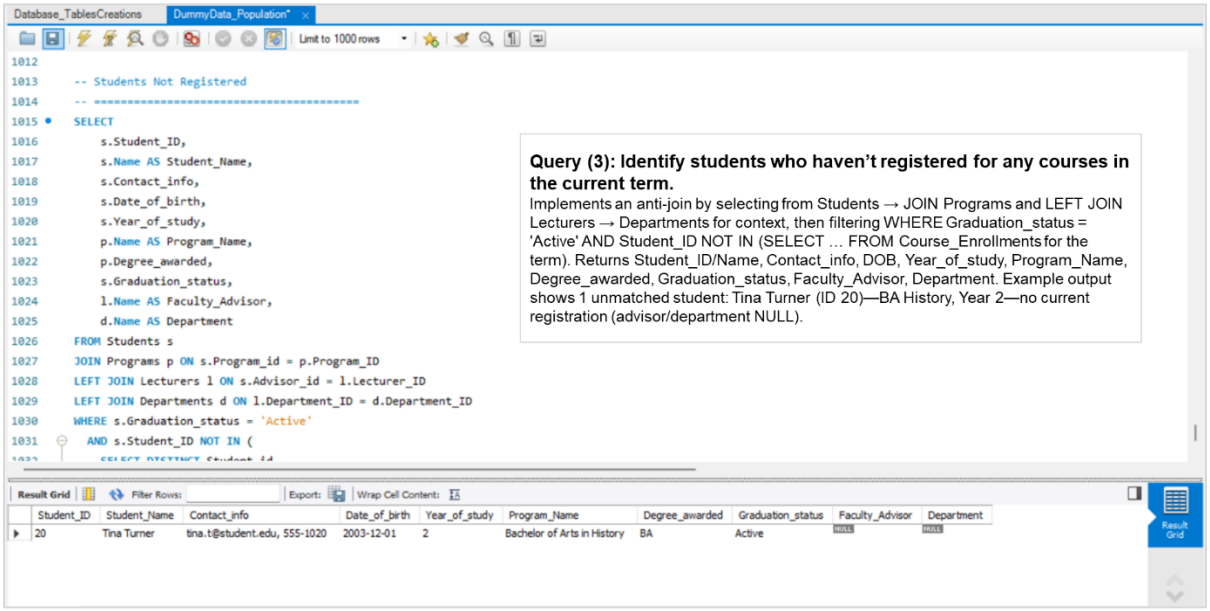


Figure 3.8:Query(3)—Result

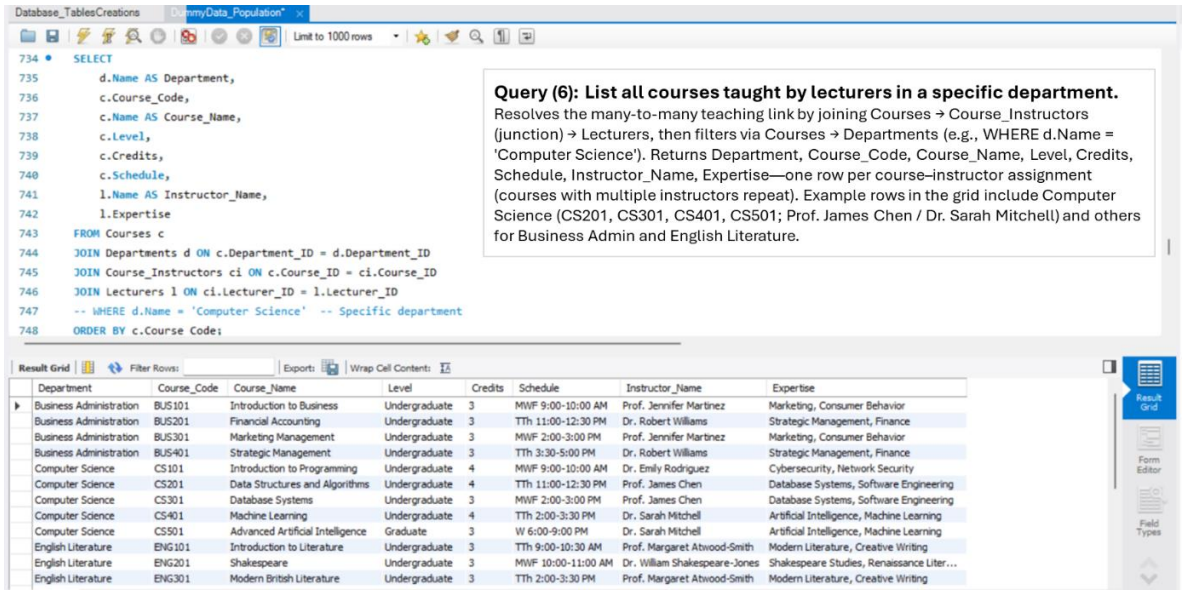


Figure 3.9:Query(6)—Result

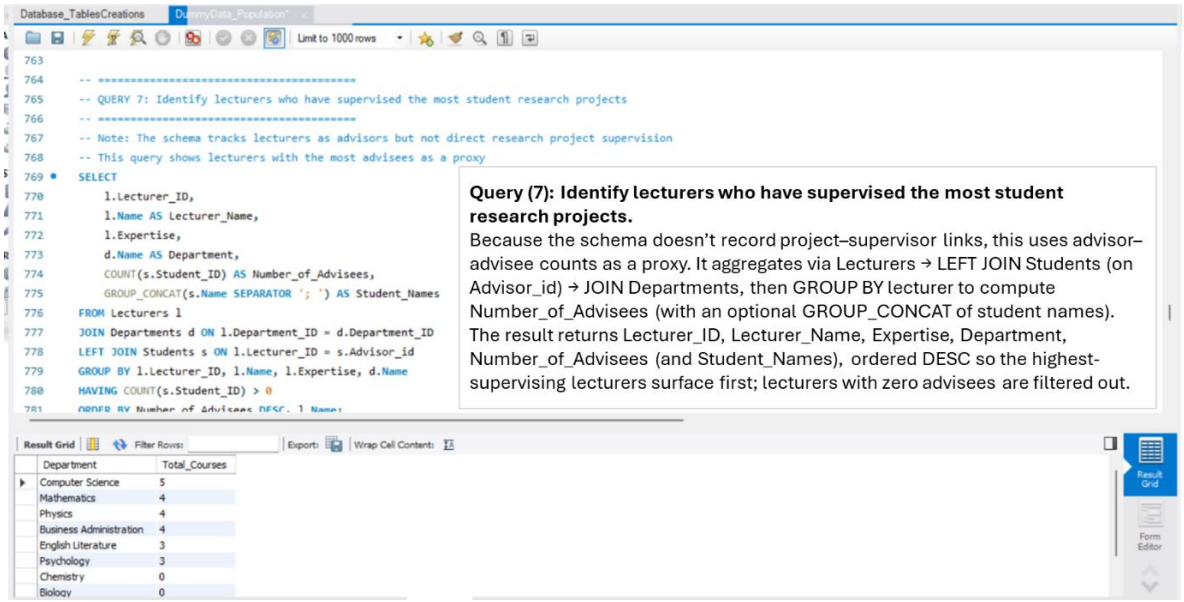


Figure 3.10:Query(7)—Result

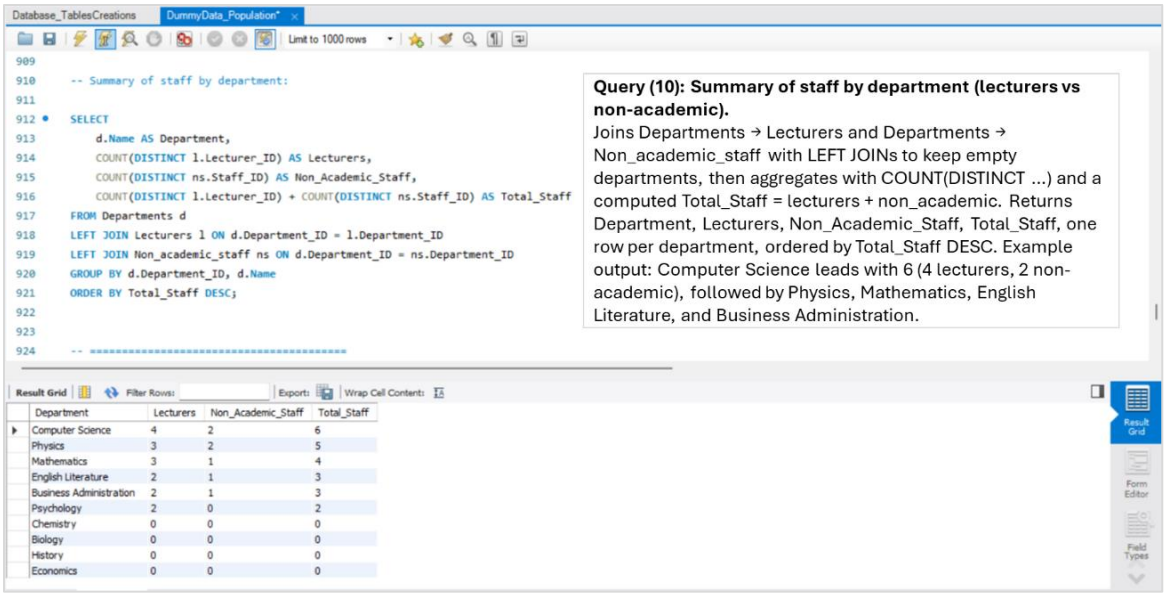


Figure 3.11:Query(10)—Result

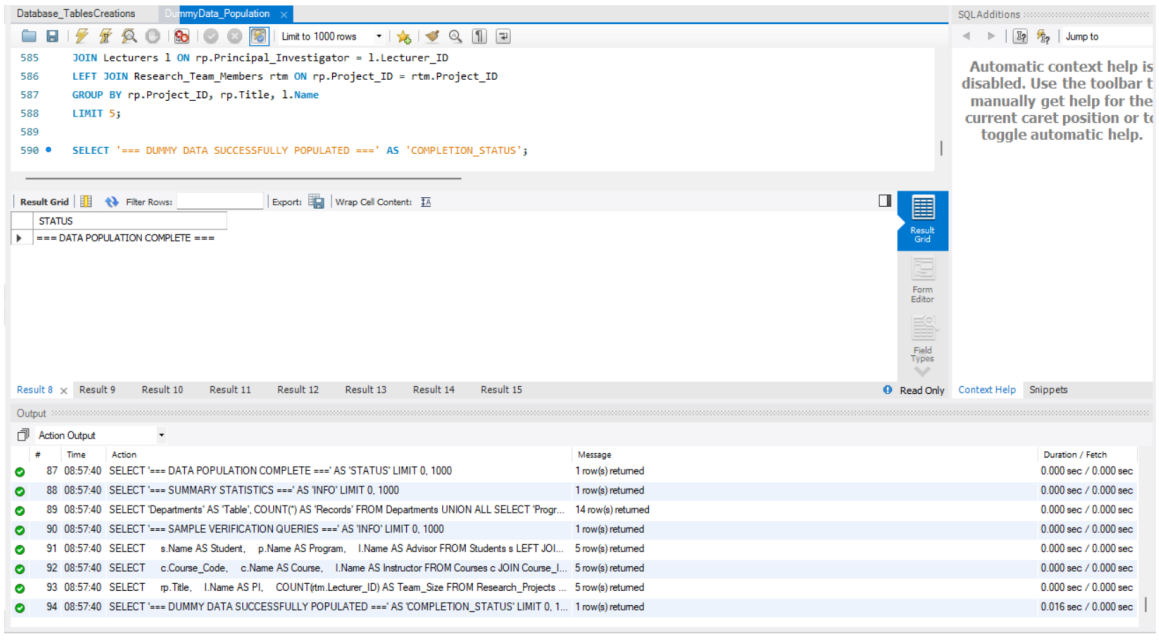


Figure 3.12:Dummy Data Population and Verification

Figure 3.13 verifies the seed load—MySQL Workbench reports “DATA POPULATION COMPLETE”, and the Action Output log shows the validation queries executing successfully.

4 Frontend Design and Development

The URMS frontend is built with Tkinter, Python's standard GUI library. With each widget linked to a Python function that interacts with the MySQL backend. The interface emphasizes consistency, accessibility, clarity, and efficiency, following usability principles that reduce cognitive load and improve accuracy (Darejeh & Singh, 2013). Readable typography, adequate color contrast, and keyboard navigation support inclusive use. Input validation and exception handling improve reliability by preventing malformed inputs and managing run-time errors (Altulaihan et al., 2023).

When users initiate actions (e.g., retrieving enrolled students or lecturer data), Tkinter calls functions that execute parameterized SQL via mysql.connector. Parameterization improves performance and mitigates SQL-injection risks (Sidik et al., 2023). Tight integration between the frontend and backend enables real-time results and reporting, making the system robust and easy to use.

4.1 Interface Requirements and Usability Principles

The interface should be comprehensible, transparent, and uniform across screens. Labels, menus, and action buttons follow familiar patterns to ease navigation and reduce training. Accessibility and responsive feedback foster inclusivity and user confidence. Adhering to established heuristics focuses the design on minimizing errors, simplifying cognition, and optimizing overall user experience (Darejeh & Singh, 2013).

4.2 Python and MySQL Integration

Tkinter widgets invoke Python functions that prepare and execute parameterized SQL queries through mysql.connector, then present the results in the GUI. Input validation and exception handling protect data integrity and system stability. Using parameterized queries follows best practices for secure, real-time application–database integration by preventing SQL injection.

4.3 User Interaction design and workflows

The system implements a dual-panel graphical interface where users interact through a straightforward point-and-click workflow (see Figure 4.1). The left panel contains six numbered query buttons corresponding to different database operations (student enrollment tracking, lecturer research metrics, etc.).

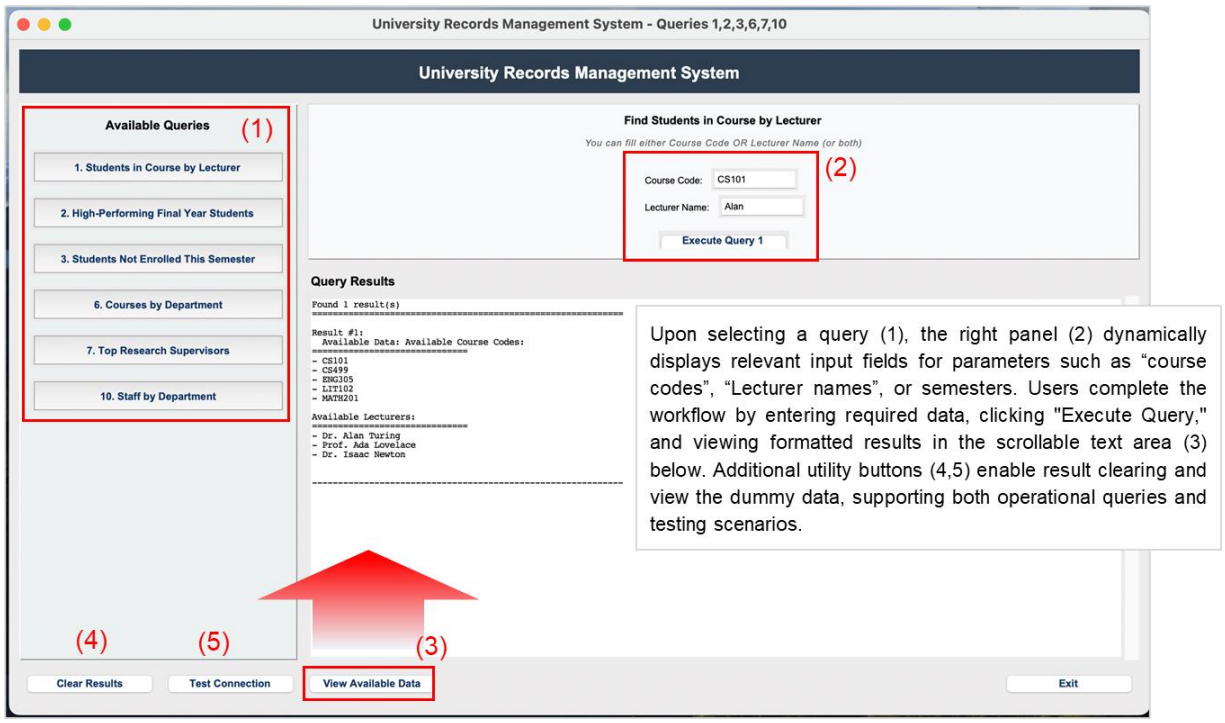


Figure 4.1: The View Available Data Button and Result

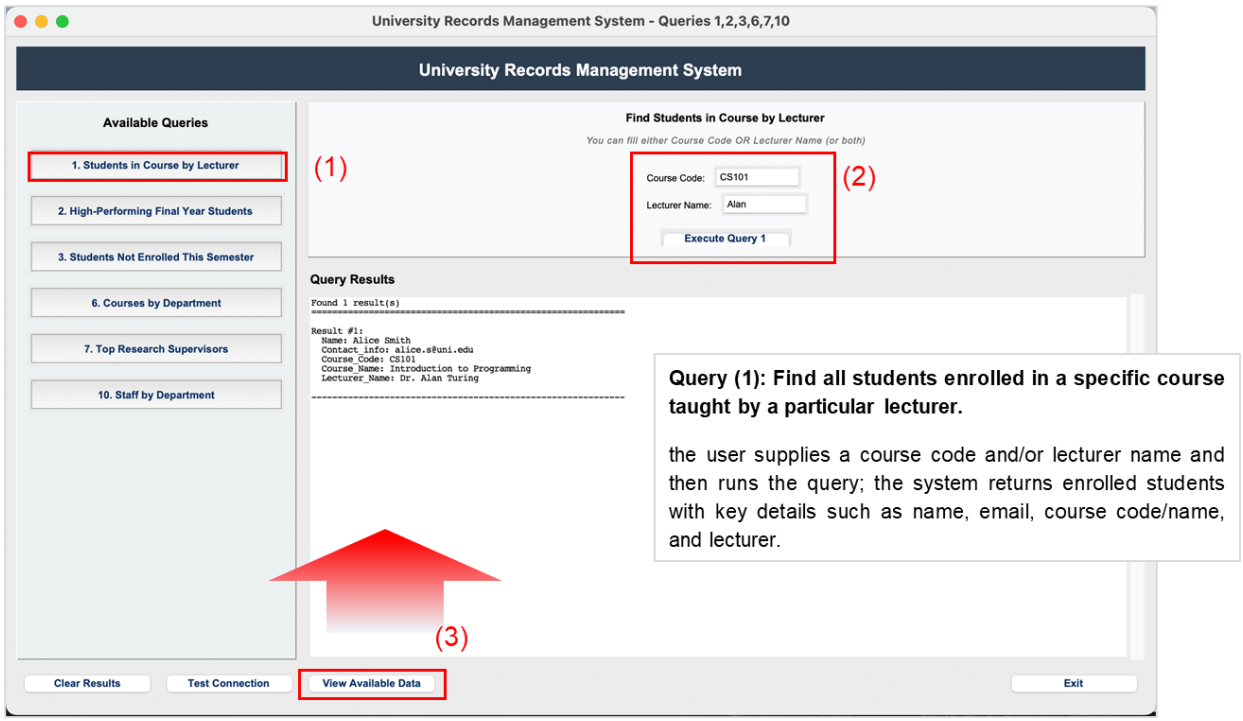


Figure 4-2:Query(1)-Result

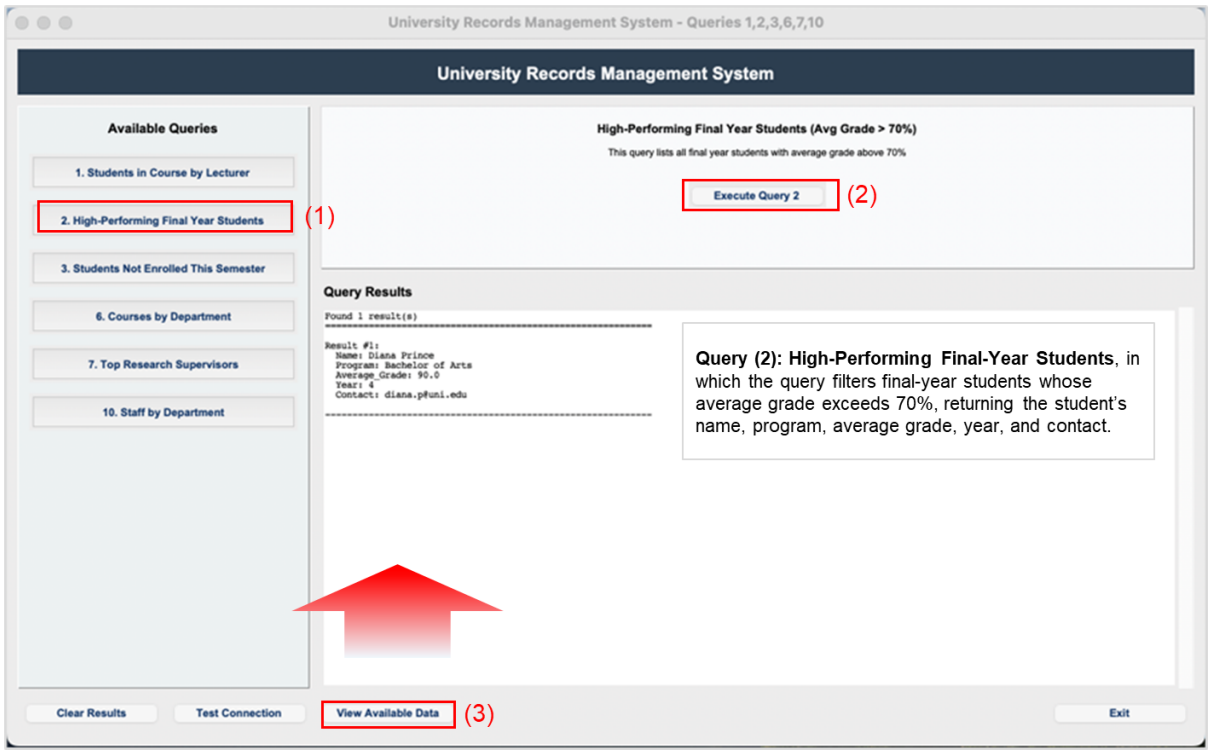


Figure 4-3:Query(2)-Result

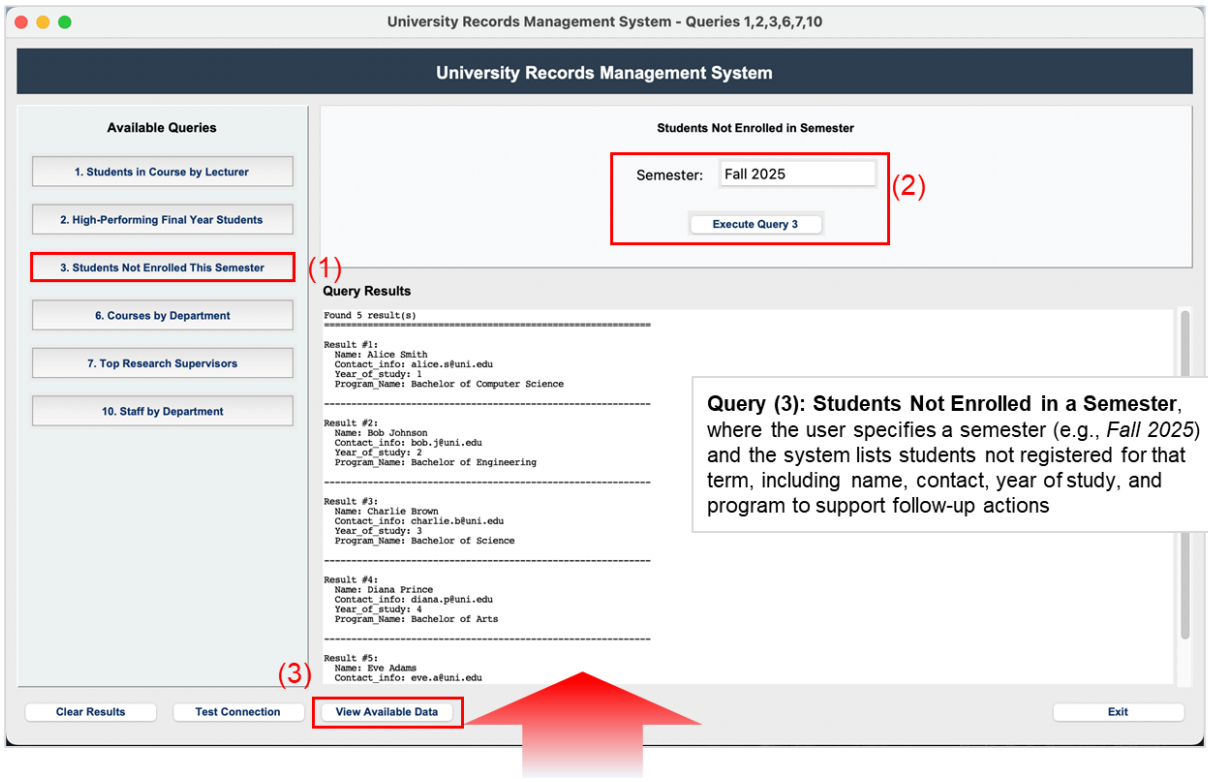


Figure 4-4:Query(3)-Result

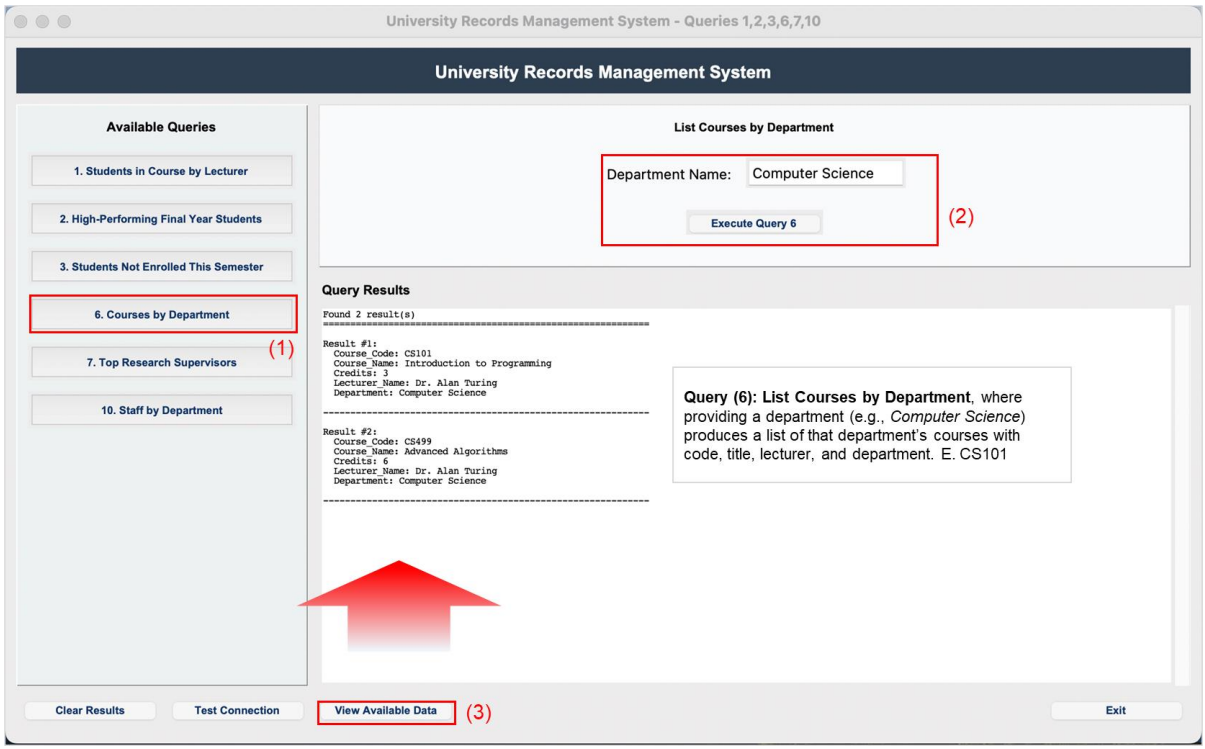


Figure 4-5:Query(6)-Result

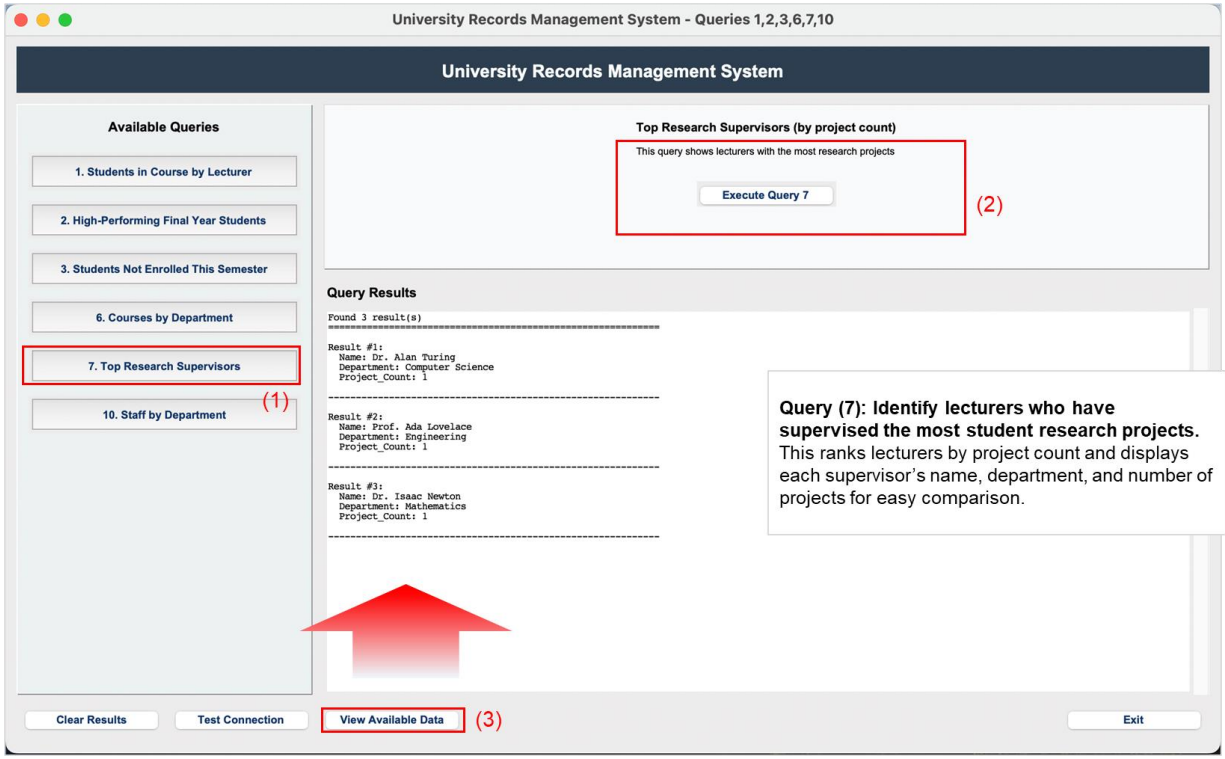


Figure 4-6:Query(7)-Result

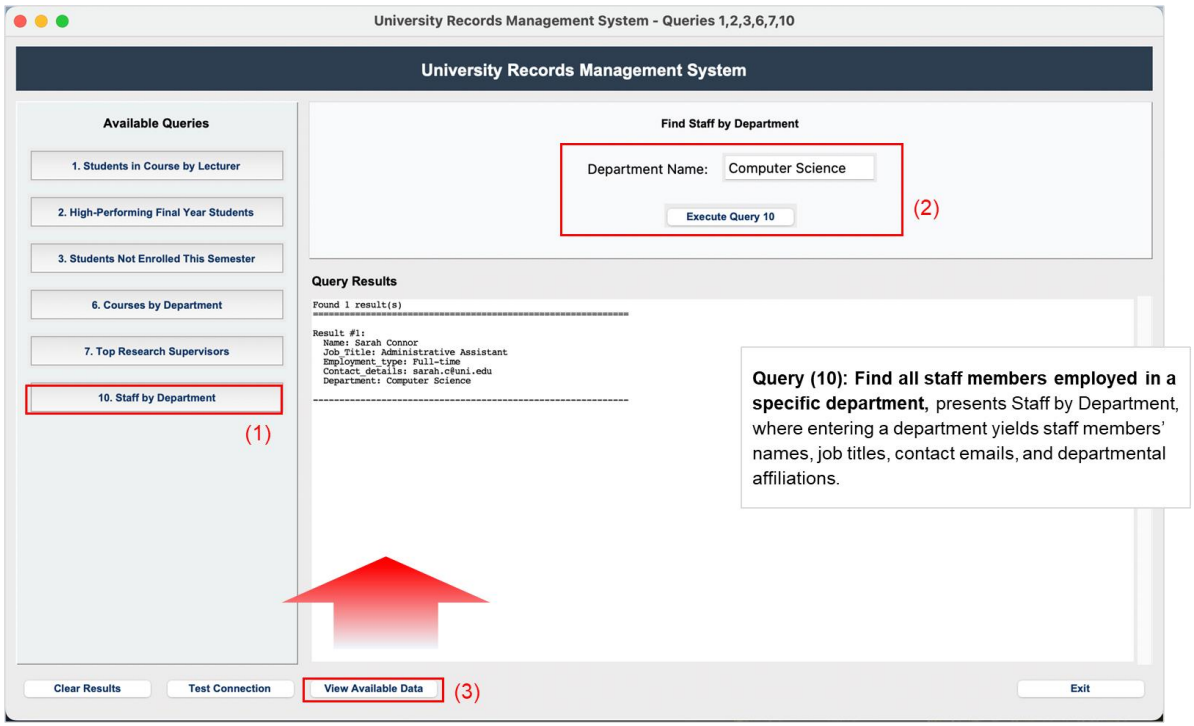


Figure 4-7:Query(10)-Result

4.4 Security and Accessibility Considerations

The system employs SQLAlchemy's parameterized queries to prevent SQL injection attacks. The modular architecture facilitates integration of authentication frameworks and role-based access control. The tkinter interface provides clear visual hierarchy with distinct navigation and content panels, supporting intuitive operation. The scrollable results area accommodates varying information volumes, while button-based navigation enables keyboard interaction. The straightforward design with explicit labels and structured output supports users with diverse technical backgrounds (Chisholm et al., 2001).

5 Implementation Challenges

5.1 Challenges — Cross-platform Front-end/DB Compatibility

In Windows, a macOS-built frontend failed due to cross-platform drift—mismatched SQLAlchemy/ connector versions, unavailable crypto backends, and mixed package managers—breaking builds and blocking tests. A MySQL handshake error requiring the cryptography backend prevented secure authentication (Figure 5.1). Another blocker was that PyCharm used the wrong interpreter/venv, causing ModuleNotFoundError: No module named 'sqlalchemy' (Figure 5.2). We stabilized by pinning versions, standardizing on PyMySQL, documenting DSN/SSL, using reproducible virtualenvs, and adding checks and CI tests. Further details in Section Front-end/DB-Compatibility

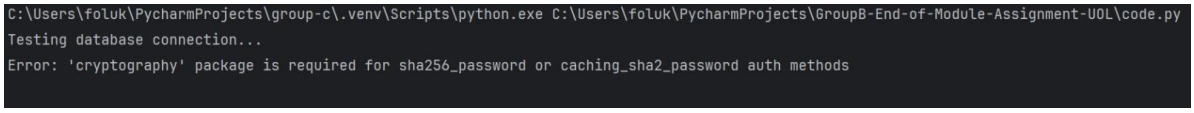


Figure 5.1:Windows Failure — missing MySQL crypto backend

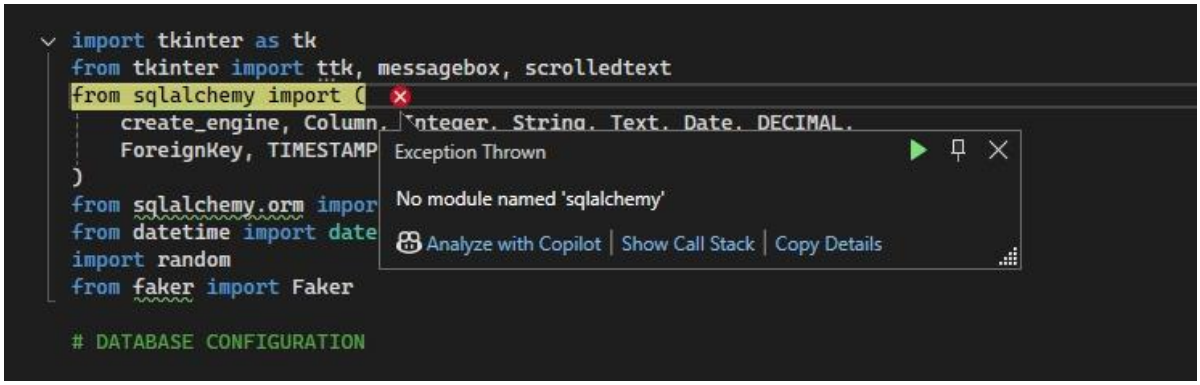


Figure 5.2:IDE Import Error

5.2 Challenges & Resolution — Remote DB Access

MySQL Workbench raised Error 1141 on revoke; remote logins failed. Causes included mismatched user@host, localhost binding, closed port 3306, and mixed auth plugins (Figure 5.3). We fixed accounts, enabled remote bind, opened 3306, standardized authentication afterward. Further details in Remote DB Access

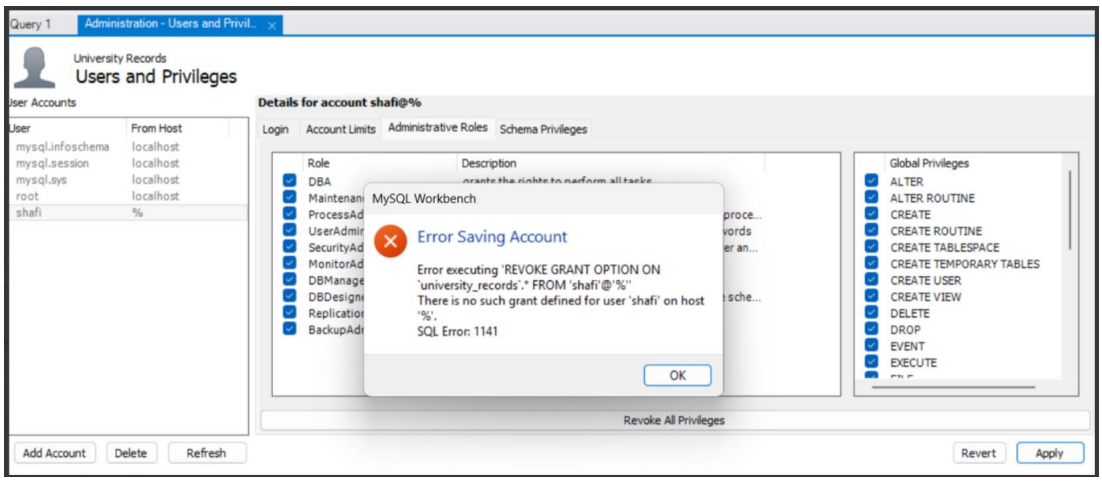


Figure 5.3:Remote DB Access

6 Conclusion and Future Study

6.1 Conclusion

The project advanced from scoping to a working University RMS with a defensible design and documented evidence. The schema was normalized to 1NF and 2NF, with many-to-many relationships resolved via junction tables and constraints that are explicitly shown in the ERD and cross-referenced in the report's data dictionary. Implementation choices (MySQL+Python) are justified against query workloads, with seeded datasets and indexed foreign keys demonstrated through six representative queries whose execution plans and outputs are included as figures. The front end maps cleanly to the schema, translating user actions into parameterized queries; usability, accessibility, and security choices are stated and tied to sources with Harvard referencing. Reproducibility is ensured via a pinned environment, README quick-start, and CI smoke tests; governance covers version control, coding standards (PEP-8), and review gates. Challenges were minor and contained.

In summary, the team delivered what it set out to build—and more: a rigorously normalized data model, a performant MySQL backend, and a clean Python-based UI turning real workflows into reliable results.

References:

Altulaihan, E.A., Alenezi, M., Alomar, N. and Alsubaie, N. (2023). A Survey on Web Application Penetration Testing, *Electronics*, 12(6), 1305. doi: https://doi.org/10.3390/electronics12051229 (Accessed: 1 October 2025).

Chisholm, W., Vanderheiden, G. and Jacobs, I. (2001): Web Content Accessibility Guidelines 1.0. Interactions, 8(4), pp. 35-54. doi: https://doi.org/10.1145/379537.379550.

Darejeh, A. and Singh, D. (2013). A Review on User Interface Design Principles to Increase Software Usability for Users with Less Computer Literacy. *Journal of Computer Science*, 9(11), pp. 1443–1450. doi: https://doi.org/10.3844/jcssp.2013.1443.1450 (Accessed: 3 October 2025).

Grambow, G. & Ruttmann, S. (2023). A practical automated transformation of entity relationship models to relational models. In: DBKDA 2023: The Fifteenth International Conference on Advances in Databases, Knowledge, and Data Applications, Barcelona, Spain, 13–17 March 2023. IARIA/ThinkMind, pp. 5–12. Available at: ThinkMind Digital Library. (Accessed 12 Oct 2025).

Mclaughlin, M. (2013). *MySQL Workbench : Data Modeling & Development*. New York Mcgraw Hill Professional.

Nielsen, J. (1994). Enhancing the Explanatory Power of Usability Heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 152–158. doi: https://doi.org/10.1145/191666.191729 (Accessed: 7 October 2025).

Sidik, R.F., Yutia, S.N. and Fathiyana, R.Z. (2023) 'The effectiveness of parameterized queries in preventing SQL injection attacks at Go', in *Proceedings of the International Conference on Enterprise and Industrial Systems (ICOEINS 2023)*, Atlantis Press, 270, pp. 205–210. doi: https://doi.org/10.2991/978-94-6463-340-5_18 (Accessed: 8 October 2025).

Weimer, W. and Necula, G.C. (2008). Exceptional Situations and Program Reliability. ACM Transactions on Programming Languages and Systems (TOPLAS), 30(2), pp.1–51. doi: https://doi.org/10.1145/1330017.1330019 (Accessed: 6 October 2025).

Appendix A

Project Timeline

Table 0.1: Project Timeline: Tasks, Activities, and Milestones

Main Task	Key Activities	Milestones / Outputs
1. Foundation Setup (24–25 Sep)	- Database planning and ERD design (Team A) - Python environment setup with MySQL connector (Team B) - Documentation structure and test plan draft (Team C)	Project initialization complete
2. Core Development (26–27 Sep)	- MySQL schema creation for students, lecturers, courses,etc. - Insert 20+ sample records - Develop reusable SQL queries and procedures	*Milestone 1:* Database functional with sample data
3. Integration & Interface Development (28–30 Sep)	- Python–MySQL connection setup - Build CLI or Flask interface - Run live queries and validate data retrieval - Add indexes and optimize database	*Milestone 2:* Integrated and functional system
4. Finalization & Documentation (1–3 Oct)	- Complete ERD and schema documentation - Conduct final testing (functional + non-functional) - Write report with screenshots - Record and edit demo video	*Deliverable 1:* Full report and demo video ready
5. Submission and Review (4 Oct)	- Consolidate project files (source, database, docs, video) - Upload final package to GitHub and submit	*Milestone 3:* Final submission completed

URMS Project Gantt Chart:

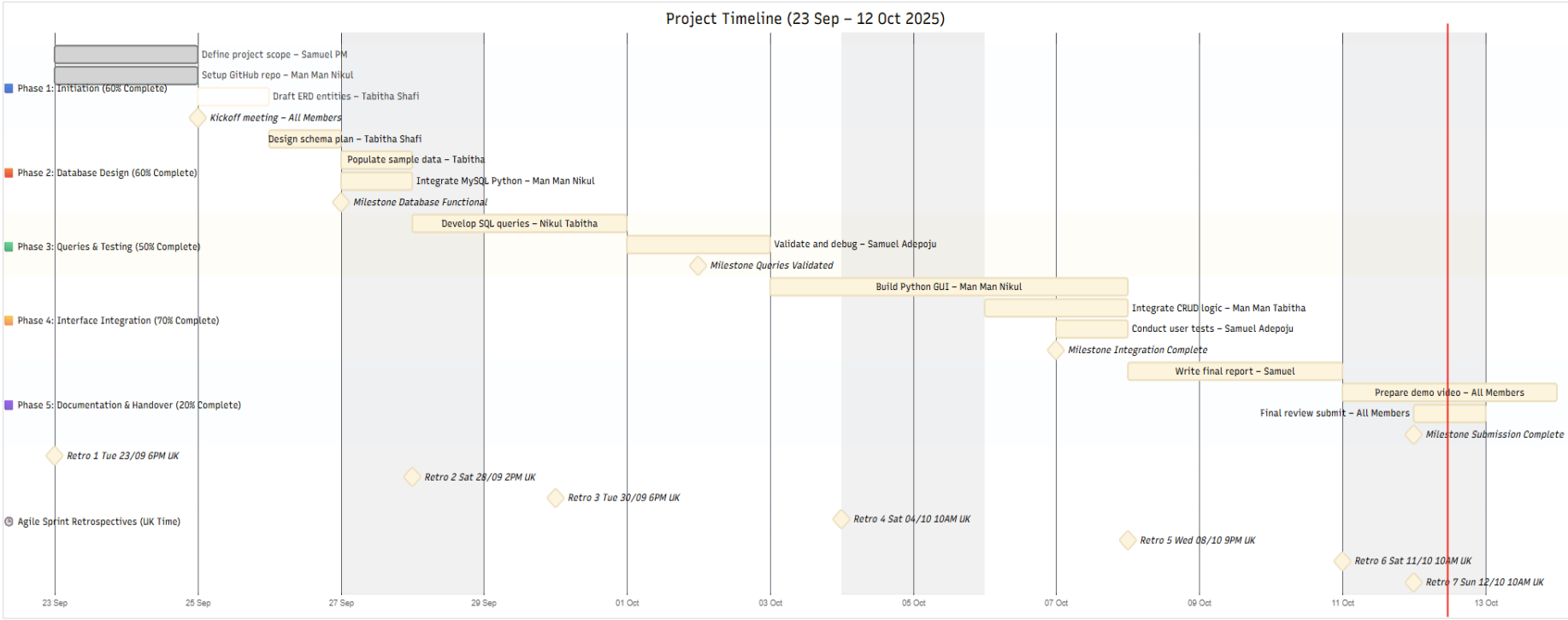


Figure 0-1: Project execution Gantt

Roles and Responsibilities

Table 3.2: Roles Responsibilities and Teams

Full Name	Role/Tasks	Teams
Nduta, Tabitha Elsie	(Back End) Database Engineer;	Team A
Issa, Shafi Suleiman	(Back End) Database Engineer;	
Chan, Man Man	(Front End) Software Engineer;	Team B
Kerai, Nikul	(Front End) Software Engineer; Tester	
Adepoju, Samuel Temitayo	Project Manager; Tester	Team C

- Team A – Database Development (MySQL): Responsible for database architecture, schema design, sample data insertion, query optimization, and export of the final ER diagram.
- Team B – Software Development (Python): Focused on frontend creation, query execution logic, database integration using MySQL-connector, and interface usability.
- Team C – Testing & Documentation: Tasked with test case design, system validation, report writing, and preparation of the demonstration video.

ERD Relationships

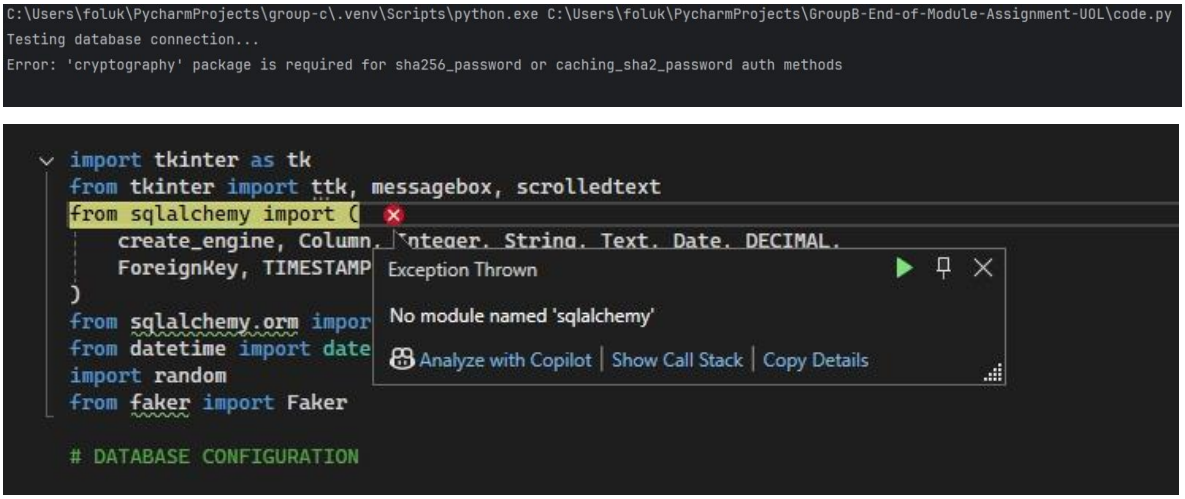
Table 1.2: ERD Relationship Table (mapped to Figure 3.1)

Entity Relationships	Cardinality (L→R)	Optionality (L / R)	Keys (PK→FK)	Validation / Notes
Students belongs to Programs	M:1	Must / May	Programs.Program_id → Students.Program_id	Each student has exactly one Program; Program can exist without students.
Lecturers advises Students	1:M	May / May	Lecturers.Lecturer_id → Students.Advisor_id	Student may be unassigned; optional rule: advisor and student in same department.
Students enrolls in Courses	M:N via Course_Enrollments	Must / Must per row	Students.Student_id → Enrollments.Student_id; Courses.Course_id → Enrollments.Course_id	status IN ('enrolled','dropped','withdrawn','completed'); grade BETWEEN 0 AND 100 (nullable until posted).
Lecturers teaches Courses	M:N via Course_Instructors	May / May	Lecturers.Lecturer_id → Course_Instructors.Lecturer_id; Courses.Course_id → Course_Instructors.Course_id	Total workload per (Course_id,term) ≤ 100; role ENUM('Instructor','TA','Guest').
Departments offers Courses	1:M	May / Must	Departments.Department_id → Courses.Department_id	Each Course belongs to one owning Department.
Departments employs Lecturers	1:M	May / Must	Departments.Department_id → Lecturers.Department_id	Department may have zero lecturers; lecturer must have exactly one department.
Departments employs Non_Academic_Staff	1:M	May / Must	Departments.Department_id → Non_Academic_Staff.Department_id	employment_type IN ('permanent','contract','temp').
Lecturers is PI of Research_Projects	1:M	May / Must	Lecturers.Lecturer_id → Research_Projects.Principal_investigator	Project must have a PI; optional rule: PI also listed as team member.

Lecturers team member of Research_Projects	M:N via Research_Team_Members	May / May	Lecturers.Lecturer_id → Research_Team_Members.Lecturer_id; Research_Projects.Project_id → Research_Team_Members.Project_id	left_on IS NULL OR left_on >= joined_on.
Lecturers serves on Committees	M:N via Committee_Members	May / May	Lecturers.Lecturer_id → Committee_Members.Lecturer_id; Committees.Committee_id → Committee_Members.Committee_id	end_date IS NULL OR end_date >= start_date.
Lecturers authors Publications	1:M (or M:N via Publication_Authors)	May / May	Option A: Lecturers.Lecturer_id → Publications.Lecturer_id; Option B (preferred): via Publication_Authors	author_order > 0; supports co-authorship.
Students joins Student_Organizations	M:N via Students_Organizations	May / May	Students.Student_id → Students_Organizations.Student_id; Student_Organizations.Org_id → Students_Organizations.Org_id	end_date IS NULL OR end_date >= start_date.
Courses requires Courses (prerequisite)	M:N via Course_Prerequisites	May / May	Courses.Course_id → Course_Prerequisites.Course_id; Courses.Course_id → Course_Prerequisites.Prereq_course_id	Course_id <> Prereq_course_id; optional cycle prevention via trigger.
Programs requires Courses	M:N via Program_Requirements	Must / May	Programs.Program_id → Program_Requirements.Program_id; Courses.Course_id → Program_Requirements.Course_id	Use requirement_type + min_credits to model elective buckets.

Challenges — Cross-platform Front-end/DB Compatibility

Issue: The macOS-built front end failed on Windows due to dependency and driver divergence (SQLAlchemy/connector versions and DB-crypto backends).



Impact: Build breaks, inconsistent local setups, and blocked cross-OS testing.

Root causes: SQLAlchemy + connector parity differed by OS (e.g., `mysqlclient` vs `PyMySQL`). OS-specific crypto/backends for DB encryption missing or misconfigured on Windows. Environment drift (pip/venv vs conda; Homebrew vs Chocolatey) created divergent dependency trees.

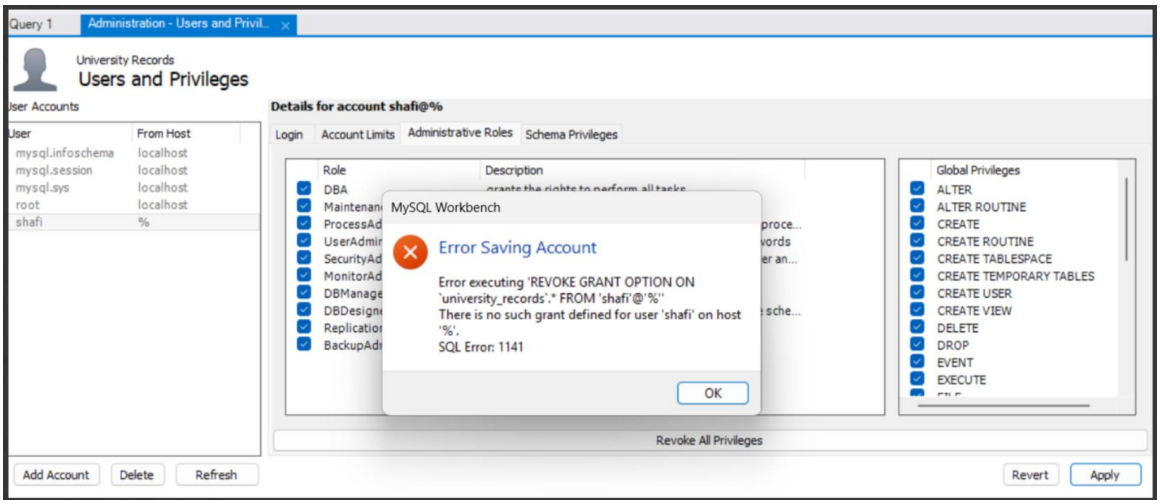
Mitigations:

- Pinned exact versions (hash-locked `requirements.txt`) with OS markers.
- Standardized on `PyMySQL` across macOS/Windows; documented DSN/SSL parameters.
- Adopted reproducible envs (virtualenv + pip-tools) and startup platform checks.
- Added CI smoke tests on both OS targets to detect regressions early.

Result. Builds now reproduce on both platforms; UI-to-DB flows pass CI and manual tests.

Challenges & Resolution — Remote DB Access

Symptom. MySQL Workbench failed to persist user changes: **Error 1141** (“No such grant defined for user ‘shafi’@‘%’”) when the GUI attempted to revoke a non-existent privilege; intermittent remote logins also occurred.



Root causes.

- Mismatched user/host pairs (`'shafi'@'%'` vs `'shafi'@'localhost'`) and GUI revoking a grant that didn't exist.
- Server bound to localhost only (`bind-address`), port 3306 not allow-listed in firewalls.
- Inconsistent authentication plugin/driver versions across clients.

Resolutions.

- Recreated correct `USER`/`HOST` entries; applied explicit `GRANT`/`REVOKE` state and avoided GUI-driven revoke of absent grants.
- Enabled remote bind, opened/allow-listed 3306, and tightened source IP ranges.
- Standardized auth plugin and aligned client drivers.

Result. Remote administration and application connections are stable; least-privilege grants are documented and validated in deployment notes.