

# **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)**

<b>Full Name</b>	<b>Role/Tasks</b>	<b>Teams</b>
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

---

1	Introduction.....	3
1.1	Problem statement and context .....	3
1.2	Purpose and scope .....	3
2	Project Management Process .....	4
2.1	Project Organization, roles and responsibilities and Collaboration tools .....	4
2.2	Work breakdown structure and scheduling .....	5
2.3	Risks and mitigation employed .....	5
3	Backend Design, Development and Testing .....	6
3.1	Database schema and ERD Design .....	6
3.2	Backend Technologies .....	7
3.3	Query Implementation workflow .....	7
4	Frontend Design and Development .....	13
4.1	Interface Requirements and Usability Principles .....	13
4.2	Python and MySQL Integration .....	13
4.3	User Interaction design and workflows .....	13
4.4	Security and Accessibility Considerations .....	17
5	Implementation Challenges .....	18
5.1	Challenges — Cross-platform Front-end/DB Compatibility .....	18
5.2	Challenges & Resolution — Remote DB Access.....	18
6	Conclusion and Future Study (130 words).....	20
6.1	Conclusion.....	20
	References: .....	21
	Appendix A .....	22
	Project Timeline .....	22
	URMS Project Gantt Chart: .....	23
	Roles and Responsibilities .....	24
	ERD Relationships.....	25
	Challenges — Cross-platform Front-end/DB Compatibility .....	27
	Challenges & Resolution — Remote DB Access .....	27

## Table of Figures

---

Figure 2.1: Project workflow .....	4
Figure 2.2: Workflow and WBS with timelines.....	5
Figure 3.1: ERD Mapped to Queries and Keys .....	6
Figure 3.2: Schema Creation Script (Database and Departments Table).....	7
Figure 3.3: List of Tables in university records schema .....	8
Figure 3.4: List of Dummy Records .....	8
Figure 3.5: Table Structure with Foreign Keys .....	9
Figure 3.6:Query(1)—Result .....	9
Figure 3.7:Query(2)—Result .....	10
Figure 3.8:Query(3)—Result .....	10
Figure 3.10:Query(6)—Result .....	11
Figure 3.11:Query(7)—Result .....	11
Figure 3.12:Query(10)—Result .....	12
Figure 3.13:Dummy Data Population and Verification .....	12
Figure 4.1: The View Available Data Button and Result .....	14
Figure 4-2:Query(1)-Result.....	14
Figure 4-3:Query(2)-Result.....	15
Figure 4-4:Query(3)-Result.....	15
Figure 4-5:Query(6)-Result.....	16
Figure 4-6:Query(7)-Result.....	16
Figure 4-7:Query(10)-Result.....	17
Figure 5.1:Windows Failure — missing MySQL crypto backend .....	18
Figure 5.2:IDE Import Error .....	18
Figure 5.3:Remote DB Access .....	19
Figure 7-1: Project execution Gantt.....	23

# 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 23 September to 13 October 2025, adopting a phased approach to ensure effective coordination between database, software, and documentation teams (Figure 2.1) See full structure in [Appendix A](#).

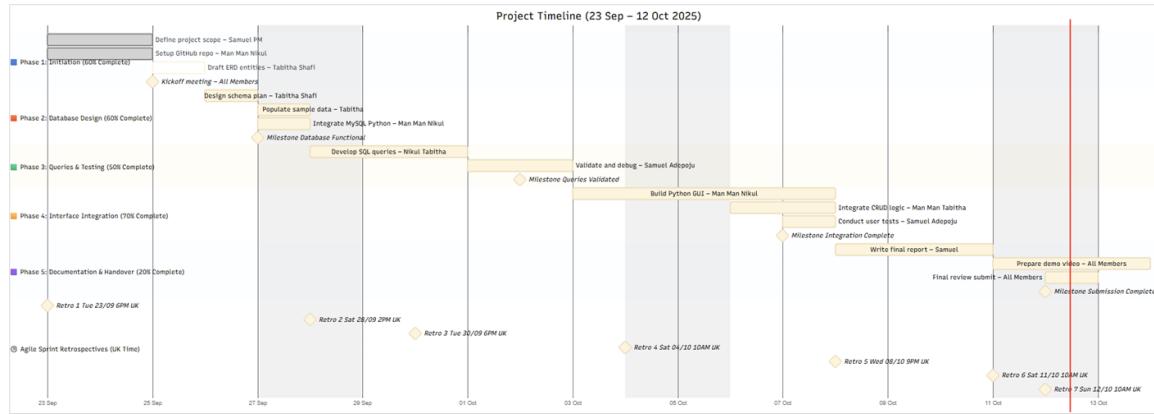


Figure 2.1: Project workflow

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

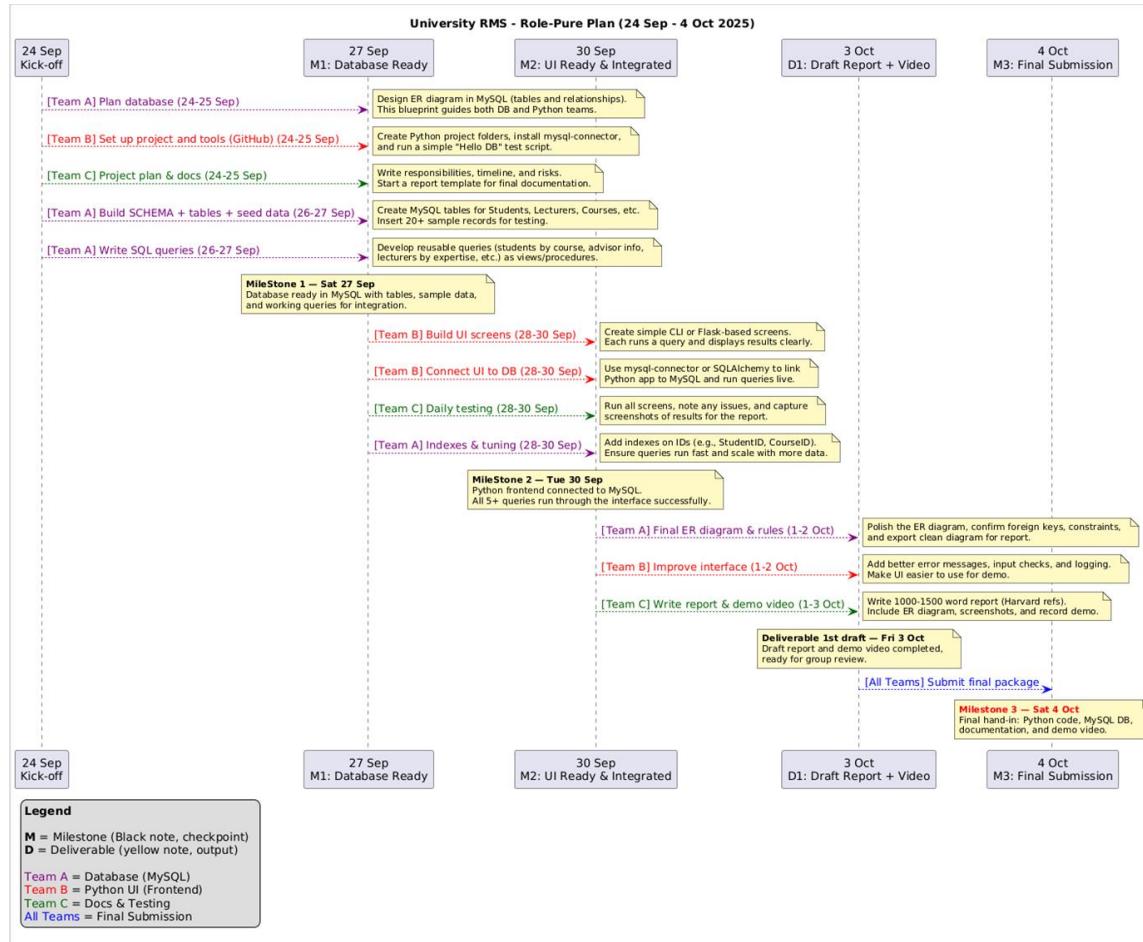


Figure 2.2: Workflow and WBS with timelines

## 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.

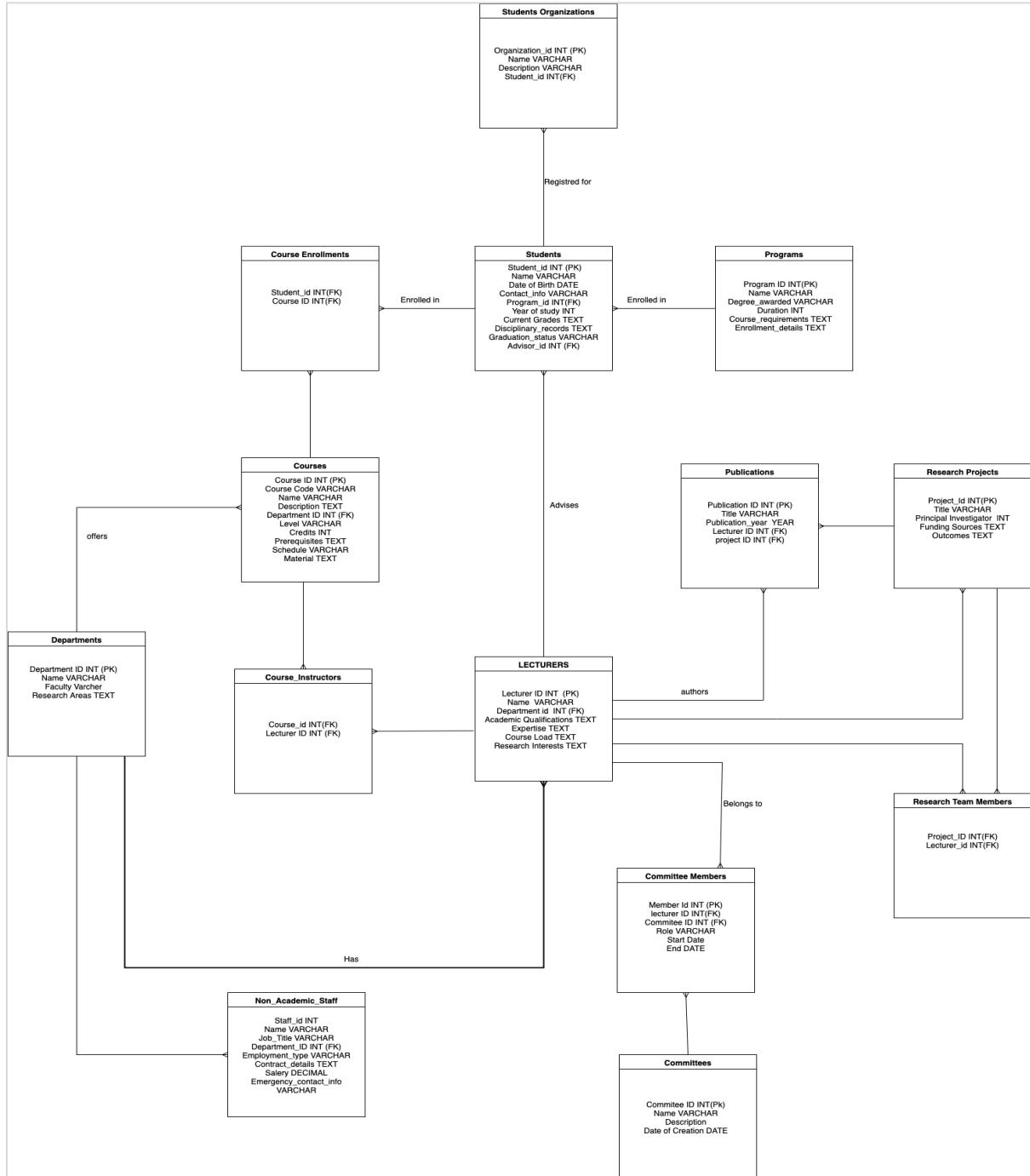


Figure 3.1: ERD Mapped to Queries and Keys

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 & Ruttman, 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).

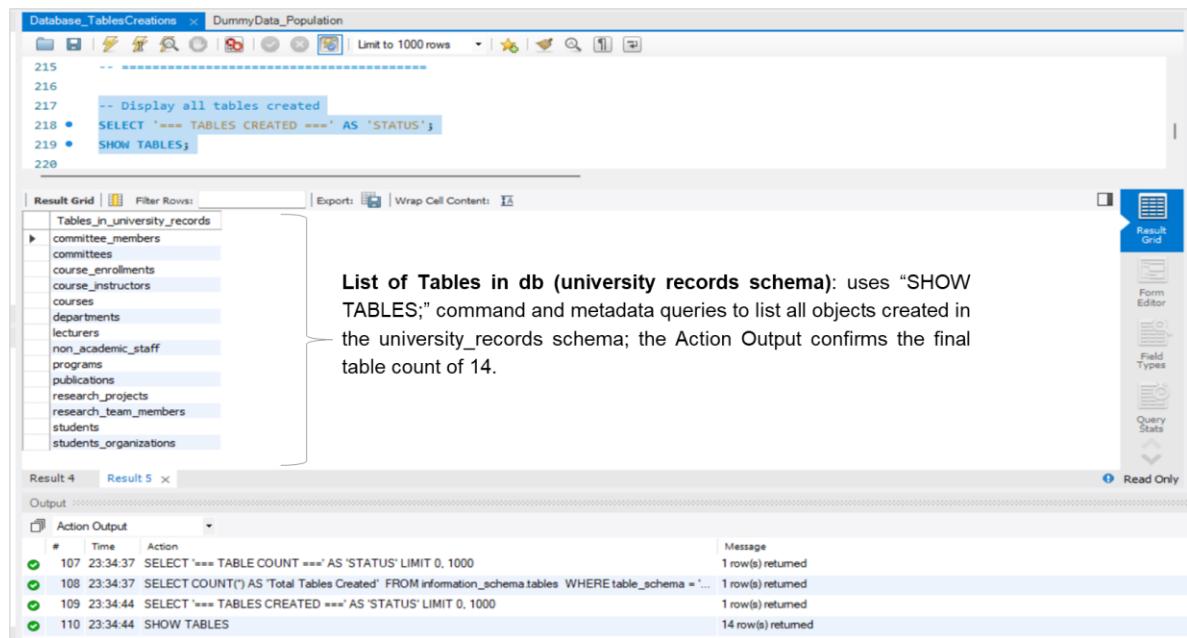
```
1 -- UNIVERSITY RECORDS DATABASE SETUP
2 -- STEP 1: DATABASE CREATION
3 • CREATE DATABASE IF NOT EXISTS university_records;
4 • USE university_records;
5
6 -- STEP 2: TABLE CREATION
7
8 -- Departments: Foundation for academic structure
9 • ⊖ CREATE TABLE IF NOT EXISTS Departments (
10     Department_ID INT AUTO_INCREMENT PRIMARY KEY,
11     Name VARCHAR(100) NOT NULL,
12     Faculty VARCHAR(100) NOT NULL,
13     Research_Areas TEXT,
14     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
15 ) 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)



The screenshot shows the Oracle SQL Developer interface with the 'Database Tables Creations' tab selected. The 'Result Grid' pane displays a list of tables under the schema 'Tables\_in\_university\_records'. The tables listed include: committee\_members, committees, course\_enrollments, course\_instructors, courses, departments, lecturers, non\_academic\_staff, programs, publications, research\_projects, research\_team\_members, students, and students\_organizations. A callout box points to this list with the text: "List of Tables in db (university records schema): uses 'SHOW TABLES;' command and metadata queries to list all objects created in the university\_records schema; the Action Output confirms the final table count of 14."

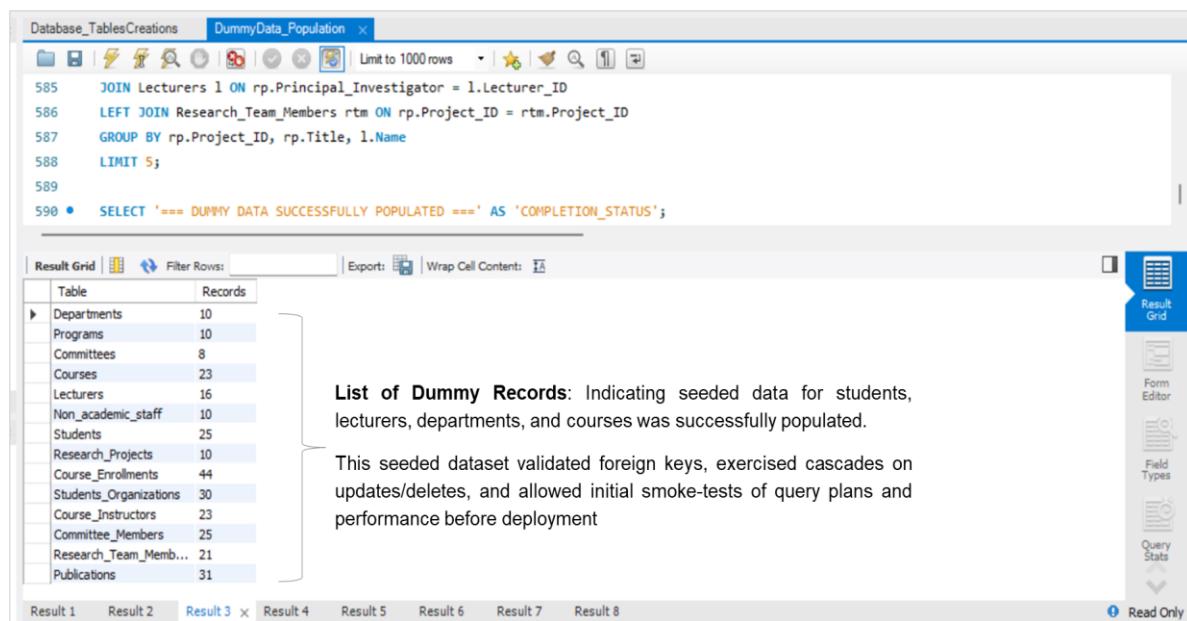
The 'Action Output' pane at the bottom shows the following log entries:

#	Time	Action	Message
107	23:34:37	SELECT *** TABLE COUNT *** AS 'STATUS' LIMIT 0, 1000	1 row(s) returned
108	23:34:37	SELECT COUNT(*) AS 'Total Tables Created' FROM information_schema.tables WHERE table_schema = '...'	1 row(s) returned
109	23:34:44	SELECT *** TABLES CREATED *** AS 'STATUS' LIMIT 0, 1000	1 row(s) returned
110	23:34:44	SHOW TABLES	14 row(s) returned

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



The screenshot shows the Oracle SQL Developer interface with the 'Database Tables Creations' tab selected. The 'Result Grid' pane displays a list of tables and their record counts under the 'Records' column. The tables and their counts are: Departments (10), Programs (10), Committees (8), Courses (23), Lecturers (16), Non\_academic\_staff (10), Students (25), Research\_Projects (10), Course\_Enrollments (44), Students\_Organizations (30), Course\_Instructors (23), Committee\_Members (25), Research\_Team\_Memb... (21), and Publications (31). A callout box points to this list with the text: "List of Dummy Records: Indicating seeded data for students, lecturers, departments, and courses was successfully populated."

The 'Action Output' pane at the bottom shows the following log entry:

#	Time	Action	Message
585	23:34:37	JOIN Lecturers l ON rp.Principal_Investigator = l.Lecturer_ID	
586	23:34:37	LEFT JOIN Research_Team_Members rtm ON rp.Project_ID = rtm.Project_ID	
587	23:34:37	GROUP BY rp.Project_ID, rp.Title, l.Name	
588	23:34:37	LIMIT 5;	
589	23:34:44	SELECT *** DUMMY DATA SUCCESSFULLY POPULATED *** AS 'COMPLETION_STATUS';	

A second callout box points to the log entry with the text: "This seeded dataset validated foreign keys, exercised cascades on updates/deletes, and allowed initial smoke-tests of query plans and performance before deployment"

Figure 3.4: List of Dummy Records

### 3.3.3 Table Structure Keys

```

257    -- Optional: Show the table structure with foreign keys
258 •  SELECT
259      COLUMN_NAME,
260      DATA_TYPE,
261      IS_NULLABLE,
262      COLUMN_KEY,
263      COLUMN_DEFAULT,
264      EXTRA
265  FROM INFORMATION_SCHEMA.COLUMNS
266  WHERE TABLE_SCHEMA = 'university_records'
267    AND TABLE_NAME = 'Students'
268  ORDER BY ORDINAL_POSITION;
269

```

COLUMN_NAME	DATA_TYPE	IS_NULLABLE	COLUMN_KEY	COLUMN_DEF
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_TIMESTAMP... 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

598  -- ----- 599  -- QUERY 1: Find all students enrolled in a specific course taught by a particular lecturer 600  -- ----- 601 •  SELECT DISTINCT 602      s.Student_ID, 603      s.Name AS Student_Name, 604      s.Contact_info, 605      c.Course_Code, 606      c.Name AS Course_Name, 607      l.Name AS Lecturer_Name, 608      ce.Semester, 609      ce.Academic_year, 610      ce.Status AS Enrollment_Status 611  FROM Students s 612  JOIN Course_Enrollments ce ON s.Student_ID = ce.Student_id	<b>Query (1): Find all students enrolled in a specific course taught by a particular lecturer.</b> 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: □ <table border="1"> <thead> <tr> <th>Student_ID</th><th>Student_Name</th><th>Contact_info</th><th>Course_Code</th><th>Course_Name</th><th>Lecturer_Name</th><th>Semester</th><th>Academic_year</th><th>Enrollment_Status</th></tr> </thead> <tbody> <tr> <td>1</td><td>Alice Johnson</td><td>alice.j@student.edu, 555-1001</td><td>CS201</td><td>Data Structures and Algorithms</td><td>Prof. James Chen</td><td>Fall</td><td>2024-2025</td><td>Enrolled</td></tr> </tbody> </table>	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	
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

Database Tables Creations   DummyData\_Population\*   Limit to 1000 rows

```

622
623  -- -----
624  -- QUERY 2: List all students with an average grade above 70% who are in their final year
625  -- -----
626  -- Note: The Current_Grades field stores grades as letter grades (A, B+, etc.)
627  -- This query assumes we need to parse letter grades.
628  -- For demonstration, we'll identify final year students based on program duration.
629 • SELECT
630     s.Student_ID,
631     s.Name AS Student_Name,
632     s.Year_of_study,
633     p.Name AS Program_Name,
634     p.Degree_awarded,
635     p.Program_Duration,
636     s.Current_Grades,
637     s.Contact_info
638   FROM Students s
639   JOIN Programs p ON s.Program_id = p.Program_ID
640   WHERE s.Graduation_status = 'Active'
641   AND (
642       -- BSc/RRA programs: 4 years. final year = 4

```

**Query (2): List all students with an average grade above 70% who are in their final year.**

Joins Students → Programs on Program\_ID, then filters to active students whose Year\_of\_study matches the program's final year (e.g., 4-year BSc → year 4; 2-year MSc → year 2). Grades are stored as letter strings in Current\_Grades; the query assumes a mapping/parsing to identify ≥70%. Returns Student\_ID/Name, Year\_of\_study, Program\_Name, Degree\_awarded, Program\_Duration, Current\_Grades, Contact\_info. Example output shows 3 matches: Ethan Hunt, Uma Thurman (both final-year BSc), and Xavier Knight (final-year MSc).

Student_ID	Student_Name	Year_of_study	Program_Name	Degree_awarded	Program_Duration	Current_Grades	Contact_info
5	Ethan Hunt	4	Bachelor of Science in Computer Science	BSc	4 years (8 semesters)	CS401: A-, CS301: A, CS201: B+	ethan.h@student.edu, 5
21	Uma Thurman	4	Bachelor of Science in Computer Science	BSc	4 years (8 semesters)	CS401: B+, CS301: A-	uma.t@student.edu, 555
24	Xavier Knight	2	Master of Science in Computer Science	MSc	2 years (4 semesters)	CS501: A-, CS401: A	xavier.k@student.edu, 5

Figure 3.7:Query(2)—Result

Database Tables Creations   DummyData\_Population\*   Limit to 1000 rows

```

1012
1013  -- Students Not Registered
1014  -- -----
1015 • SELECT
1016     s.Student_ID,
1017     s.Name AS Student_Name,
1018     s.Contact_info,
1019     s.Date_of_birth,
1020     s.Year_of_study,
1021     p.Name AS Program_Name,
1022     p.Degree_awarded,
1023     s.Graduation_status,
1024     l.Name AS Faculty_Advisor,
1025     d.Name AS Department
1026   FROM Students s
1027   JOIN Programs p ON s.Program_id = p.Program_ID
1028   LEFT JOIN Lecturers l ON s.Advisor_id = l.Lecturer_ID
1029   LEFT JOIN Departments d ON l.Department_ID = d.Department_ID
1030   WHERE s.Graduation_status = 'Active'
1031   AND s.Student_ID NOT IN (
1032       SELECT Student_ID

```

**Query (3): Identify students who haven't registered for any courses in the current term.**

Implements an anti-join by selecting from Students → JOIN Programs and LEFT JOIN Lecturers → Departments for context, then filtering WHERE Graduation\_status = 'Active' AND Student\_ID NOT IN (SELECT ... FROM Course\_Enrollments for the term). Returns Student\_ID/Name, Contact\_info, DOB, Year\_of\_study, Program\_Name, Degree\_awarded, Graduation\_Status, Faculty\_Advisor, Department. Example output shows 1 unmatched student: Tina Turner (ID 20)—BA History, Year 2—no current registration (advisor/department NULL).

Student_ID	Student_Name	Contact_info	Date_of_birth	Year_of_study	Program_Name	Degree_awarded	Graduation_Status	Faculty_Advisor	Department
20	Tina Turner	tina.t@student.edu, 555-1020	2003-12-01	2	Bachelor of Arts in History	BA	Active	NULL	NULL

Figure 3.8:Query(3)—Result

Database Tables Creations    DummyData\_Population\*    X

```

734 •  SELECT
735     d.Name AS Department,
736     c.Course_Code,
737     c.Name AS Course_Name,
738     c.Level,
739     c.Credits,
740     c.Schedule,
741     l.Name AS Instructor_Name,
742     l.Expertise
743   FROM Courses c
744   JOIN Departments d ON c.Department_ID = d.Department_ID
745   JOIN Course_Instructors ci ON c.Course_ID = ci.Course_ID
746   JOIN Lecturers l ON ci.Lecturer_ID = l.Lecturer_ID
747   -- WHERE d.Name = 'Computer Science' -- Specific department
748   ORDER BY c.Course_Code;

```

**Query (6): List all courses taught by lecturers in a specific department.**

Resolves the many-to-many teaching link by joining Courses → Course\_Instructors (junction) → Lecturers, then filters via Courses → Departments (e.g., WHERE d.Name = 'Computer Science'). Returns Department, Course\_Code, Course\_Name, Level, Credits, Schedule, Instructor\_Name, Expertise—one row per course-instructor assignment (courses with multiple instructors repeat). Example rows in the grid include Computer Science (CS201, CS301, CS401, CS501; Prof. James Chen / Dr. Sarah Mitchell) and others for Business Admin and English Literature.

Department	Course_Code	Course_Name	Level	Credits	Schedule	Instructor_Name	Expertise
Business Administration	BUS101	Introduction to Business	Undergraduate	3	MWF 9:00-10:00 AM	Prof. Jennifer Martinez	Marketing, Consumer Behavior
Business Administration	BUS201	Financial Accounting	Undergraduate	3	TTh 11:00-12:30 PM	Dr. Robert Williams	Strategic Management, Finance
Business Administration	BUS301	Marketing Management	Undergraduate	3	MWF 2:00-3:00 PM	Prof. Jennifer Martinez	Marketing, Consumer Behavior
Business Administration	BUS401	Strategic Management	Undergraduate	3	TTh 3:30-5:00 PM	Dr. Robert Williams	Strategic Management, Finance
Computer Science	CS101	Introduction to Programming	Undergraduate	4	MWF 9:00-10:00 AM	Dr. Emily Rodriguez	Cybersecurity, Network Security
Computer Science	CS201	Data Structures and Algorithms	Undergraduate	4	TTh 11:00-12:30 PM	Prof. James Chen	Database Systems, Software Engineering
Computer Science	CS301	Database Systems	Undergraduate	3	MWF 2:00-3:00 PM	Prof. James Chen	Database Systems, Software Engineering
Computer Science	CS401	Machine Learning	Undergraduate	4	TTh 2:00-3:30 PM	Dr. Sarah Mitchell	Artificial Intelligence, Machine Learning
Computer Science	CS501	Advanced Artificial Intelligence	Graduate	3	W 6:00-9:00 PM	Dr. Sarah Mitchell	Artificial Intelligence, Machine Learning
English Literature	ENG101	Introduction to Literature	Undergraduate	3	TTh 9:00-10:30 AM	Prof. Margaret Atwood-Smith	Modern Literature, Creative Writing
English Literature	ENG201	Shakespeare	Undergraduate	3	MWF 10:00-11:00 AM	Dr. William Shakespeare-Jones	Shakespeare Studies, Renaissance Liter...
English Literature	ENG301	Modern British Literature	Undergraduate	3	TTh 2:00-3:30 PM	Prof. Margaret Atwood-Smith	Modern Literature, Creative Writing

Figure 3.9:Query(6)—Result

Database Tables Creations    DummyData\_Population\*    X

```

763
764  -- -----
765  -- QUERY 7: Identify lecturers who have supervised the most student research projects
766  -- -----
767  -- Note: The schema tracks lecturers as advisors but not direct research project supervision
768  -- This query shows lecturers with the most advisees as a proxy
769 •  SELECT
770     l.Lecturer_ID,
771     l.Name AS Lecturer_Name,
772     l.Expertise,
773     d.Name AS Department,
774     COUNT(s.Student_ID) AS Number_of_Advisees,
775     GROUP_CONCAT(s.Name SEPARATOR ', ') AS Student_Names
776   FROM Lecturers l
777   JOIN Departments d ON l.Department_ID = d.Department_ID
778   LEFT JOIN Students s ON l.Lecturer_ID = s.Advisor_id
779   GROUP BY l.Lecturer_ID, l.Name, l.Expertise, d.Name
780   HAVING COUNT(s.Student_ID) > 0
781   ORDER BY Number_of_Advisees DESC, l.Name;

```

**Query (7): Identify lecturers who have supervised the most student research projects.**

Because the schema doesn't record project-supervisor links, this uses advisor-advisee counts as a proxy. It aggregates via Lecturers → LEFT JOIN Students (on Advisor\_id) → JOIN Departments, then GROUP BY lecturer to compute Number\_of\_Advisees (with an optional GROUP\_CONCAT of student names). The result returns Lecturer\_ID, Lecturer\_Name, Expertise, Department, Number\_of\_Advisees (and Student\_Names), ordered DESC so the highest-supervising lecturers surface first; lecturers with zero advisees are filtered out.

Department	Total_Courses
Computer Science	5
Mathematics	4
Physics	4
Business Administration	4
English Literature	3
Psychology	3
Chemistry	0
Biology	0

Figure 3.10:Query(7)—Result

```

909
910 -- Summary of staff by department:
911
912 • SELECT
913     d.Name AS Department,
914     COUNT(DISTINCT l.Lecturer_ID) AS Lecturers,
915     COUNT(DISTINCT ns.Staff_ID) AS Non_Academic_Staff,
916     COUNT(DISTINCT l.Lecturer_ID) + COUNT(DISTINCT ns.Staff_ID) AS Total_Staff
917     FROM Departments d
918     LEFT JOIN Lecturers l ON d.Department_ID = l.Department_ID
919     LEFT JOIN Non_academic_staff ns ON d.Department_ID = ns.Department_ID
920     GROUP BY d.Department_ID, d.Name
921     ORDER BY Total_Staff DESC;
922
923
924 ====

```

**Query (10): Summary of staff by department (lecturers vs non-academic).**  
 Joins Departments → Lecturers and Departments → Non\_academic\_staff with LEFT JOINS to keep empty departments, then aggregates with COUNT(DISTINCT ...) and a computed Total\_Staff = lecturers + non\_academic. Returns Department, Lecturers, Non\_Academic\_Staff, Total\_Staff, one row per department, ordered by Total\_Staff DESC. Example output: Computer Science leads with 6 (4 lecturers, 2 non-academic), followed by Physics, Mathematics, English Literature, and Business Administration.

Figure 3.11:Query(10)—Result

```

585 JOIN Lecturers l ON rp.Principal_Investigator = l.Lecturer_ID
586 LEFT JOIN Research_Team_Members rtm ON rp.Project_ID = rtm.Project_ID
587 GROUP BY rp.Project_ID, rp.Title, l.Name
588 LIMIT 5;
589
590 • SELECT '*** DUMMY DATA SUCCESSFULLY POPULATED ***' AS 'COMPLETION_STATUS';

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

STATUS
*** DATA POPULATION COMPLETE ***

Action	Time	Action	Message	Duration / Fetch
87	08:57:40	SELECT '*** DATA POPULATION COMPLETE ***' AS 'STATUS' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
88	08:57:40	SELECT '*** SUMMARY STATISTICS ***' AS 'INFO' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
89	08:57:40	SELECT 'Departments AS Table', COUNT(*) AS 'Records' FROM Departments UNION ALL SELECT Prog...	14 row(s) returned	0.000 sec / 0.000 sec
90	08:57:40	SELECT '*** SAMPLE VERIFICATION QUERIES ***' AS 'INFO' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
91	08:57:40	SELECT s.Name AS Student, p.Name AS Program, i.Name AS Advisor FROM Students s LEFT JOIN...	5 row(s) returned	0.000 sec / 0.000 sec
92	08:57:40	SELECT c.Course_Code, c.Name AS Course, i.Name AS Instructor FROM Courses c JOIN Course_I...	5 row(s) returned	0.000 sec / 0.000 sec
93	08:57:40	SELECT rp.Title, i.Name AS PI, COUNT(lm.Lecturer_ID) AS Team_Size FROM Research_Projects ...	5 row(s) returned	0.000 sec / 0.000 sec
94	08:57:40	SELECT '*** DUMMY DATA SUCCESSFULLY POPULATED ***' AS 'COMPLETION_STATUS' LIMIT 0, 1...	1 row(s) returned	0.016 sec / 0.000 sec

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.).

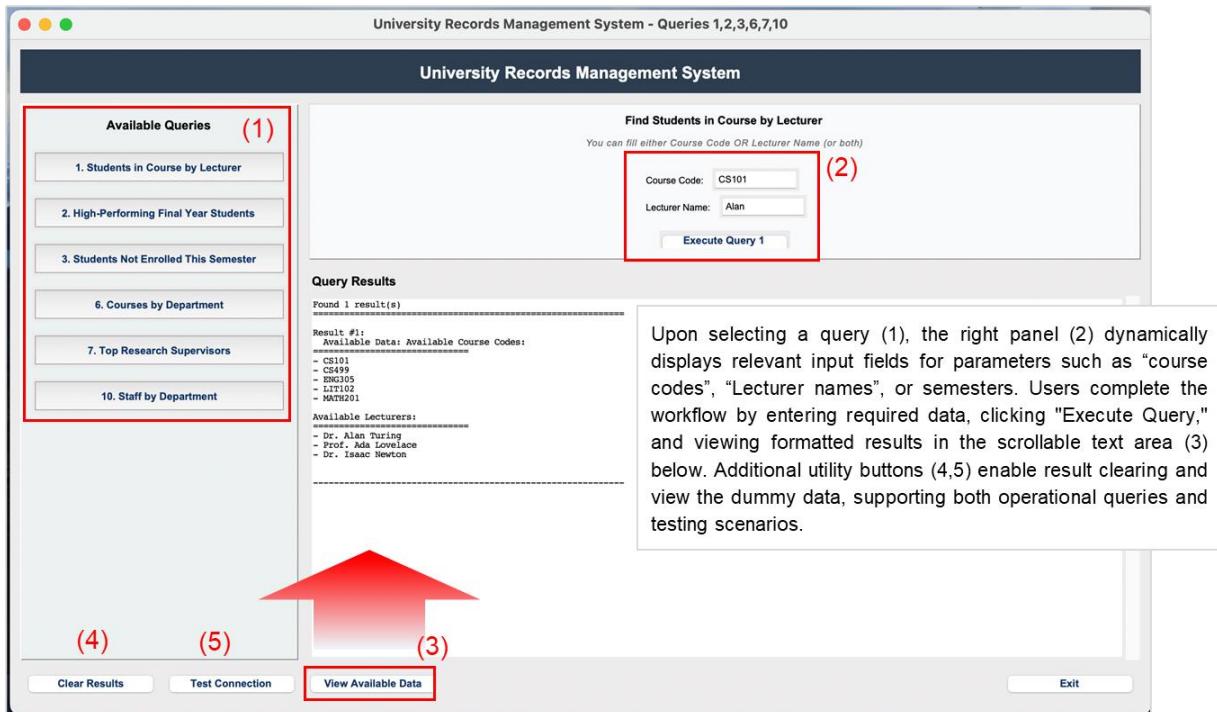


Figure 4.1: The View Available Data Button and Result

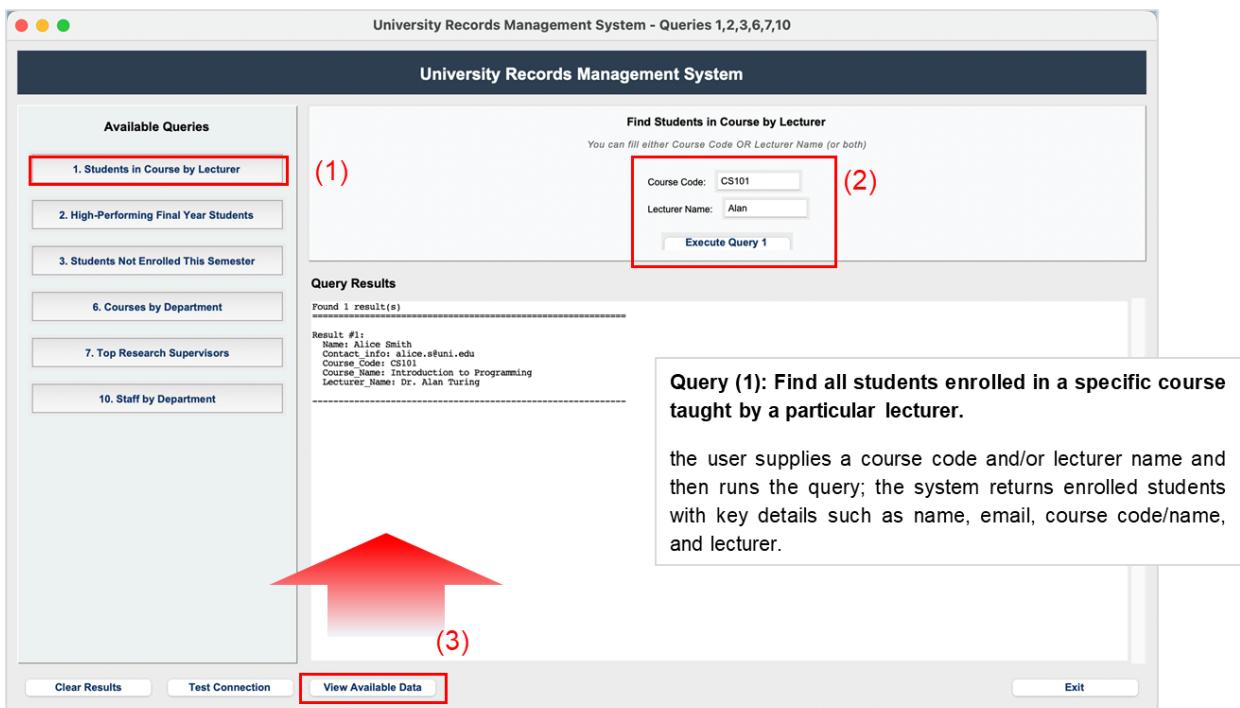


Figure 4-2:Query(1)-Result

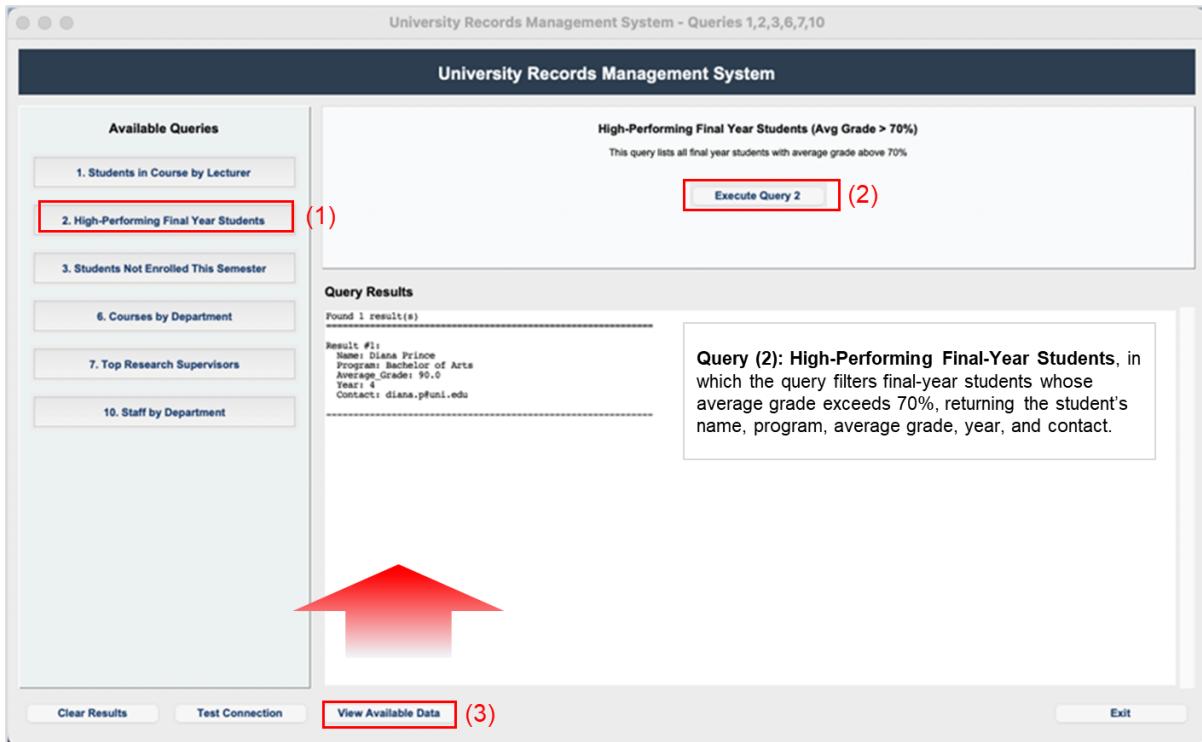


Figure 4-3:Query(2)-Result

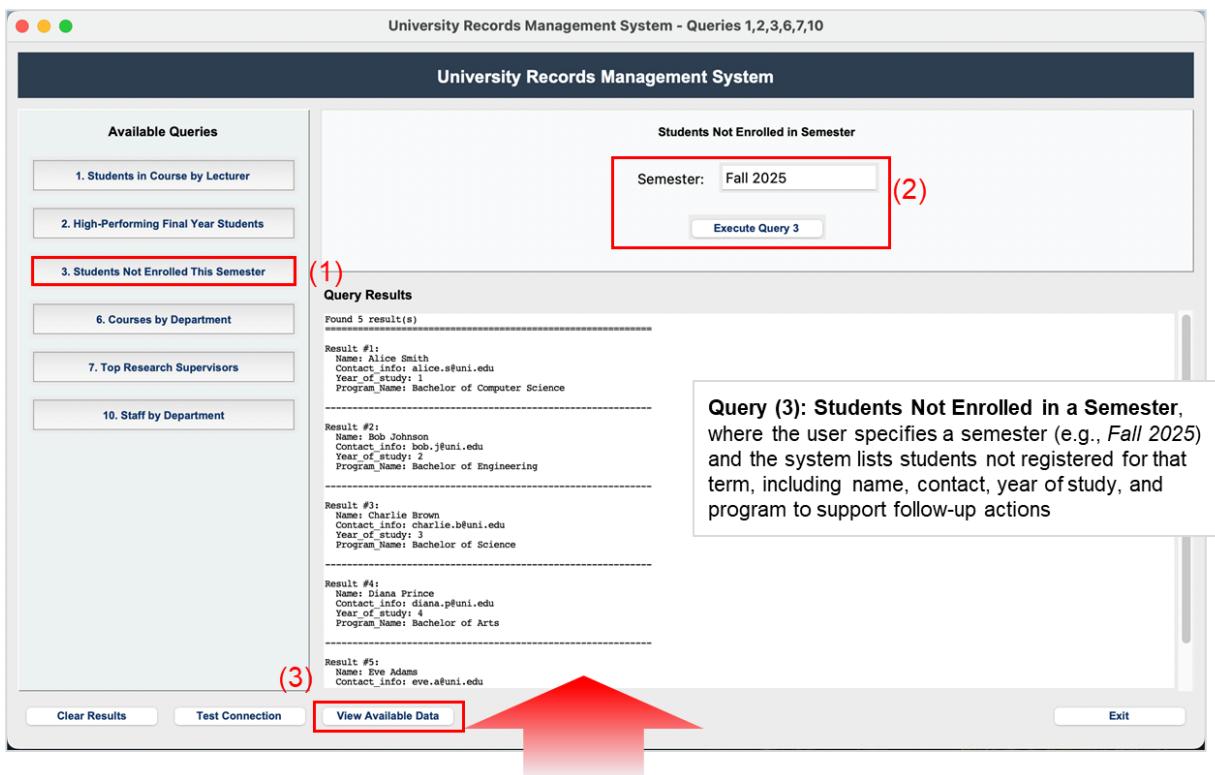


Figure 4-4:Query(3)-Result

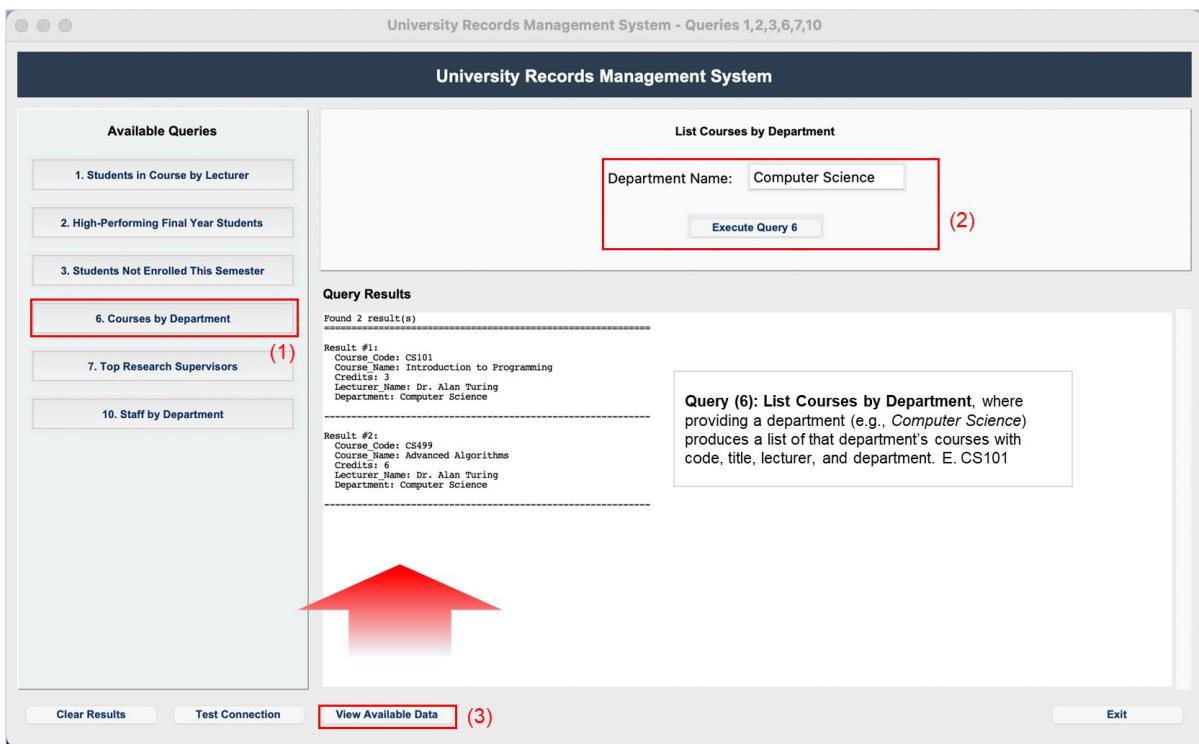


Figure 4-5:Query(6)-Result

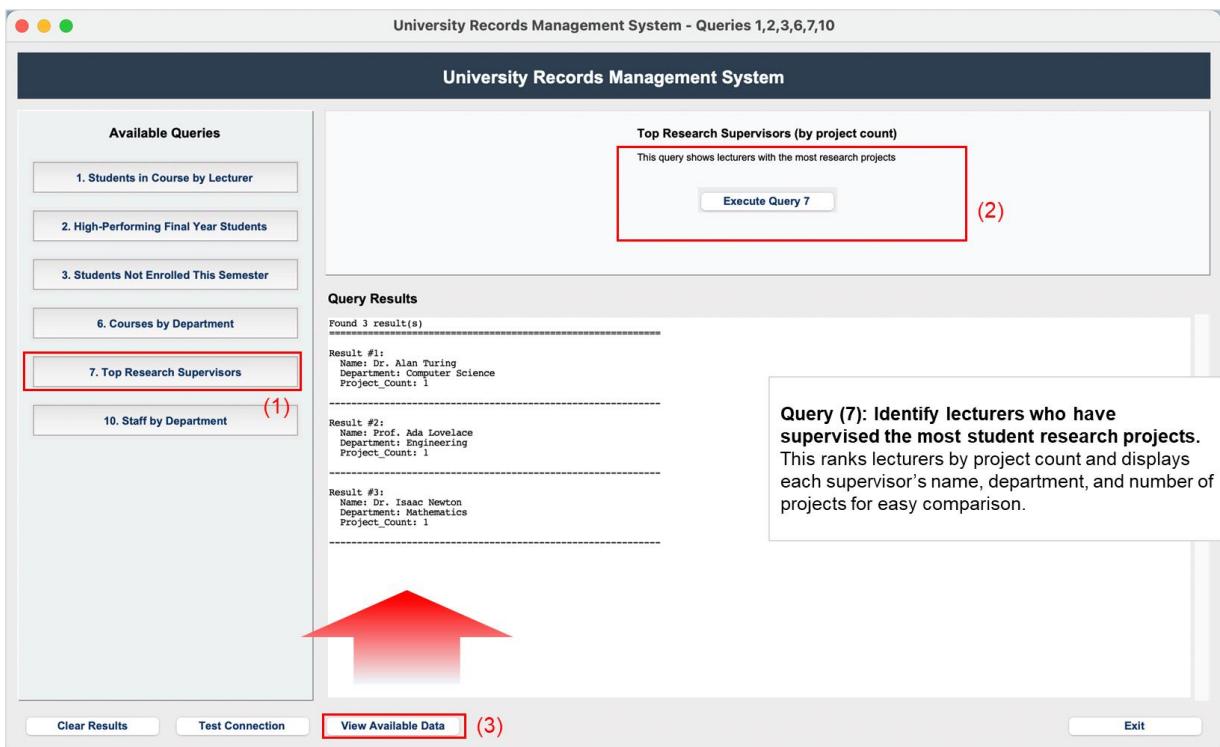


Figure 4-6:Query(7)-Result

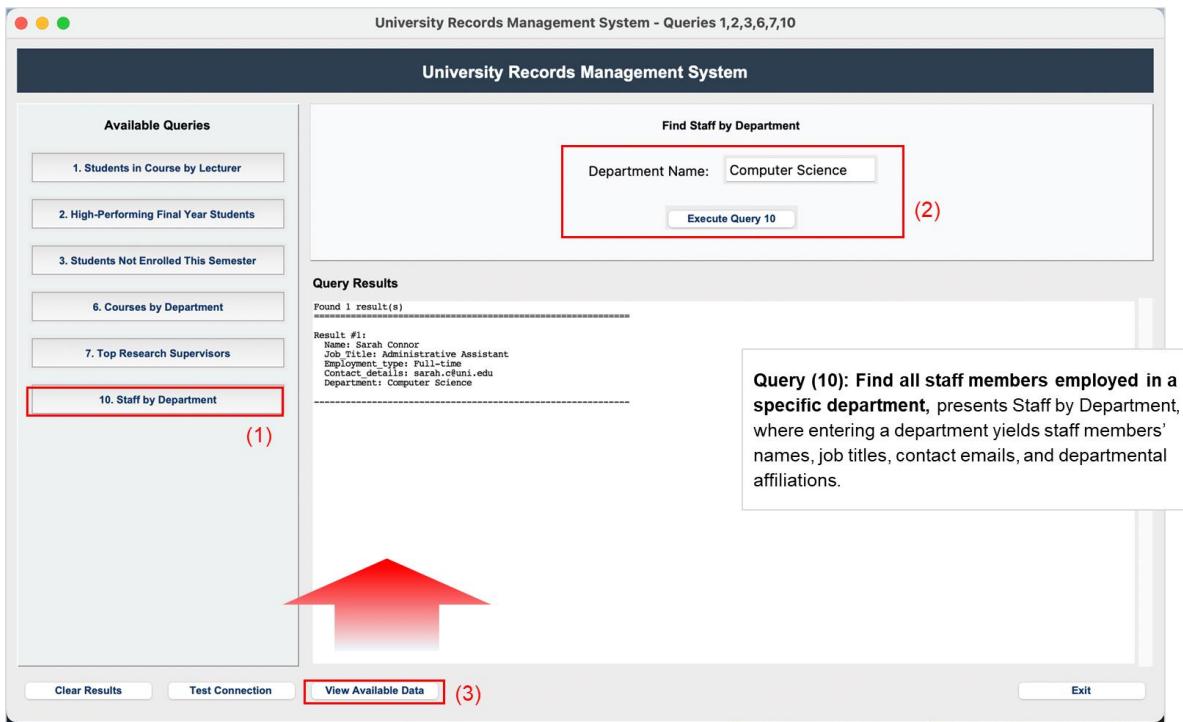


Figure 4-7:Query(10)-Result

#### 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](#)

```
C:\Users\foluk\PycharmProjects\group-c\.venv\Scripts\python.exe C:\Users\foluk\PycharmProjects\GroupB-End-of-Module-Assignment-UOL\code.py
Testing database connection...
Error: 'cryptography' package is required for sha256_password or caching_sha2_password auth methods
```

Figure 5.1:Windows Failure — missing MySQL crypto backend

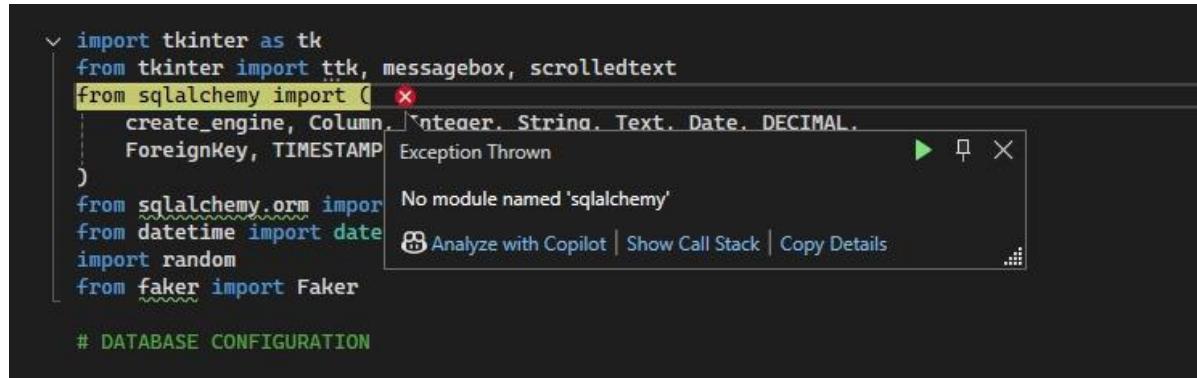


Figure 5.2:IDE Import Error

### 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](#)

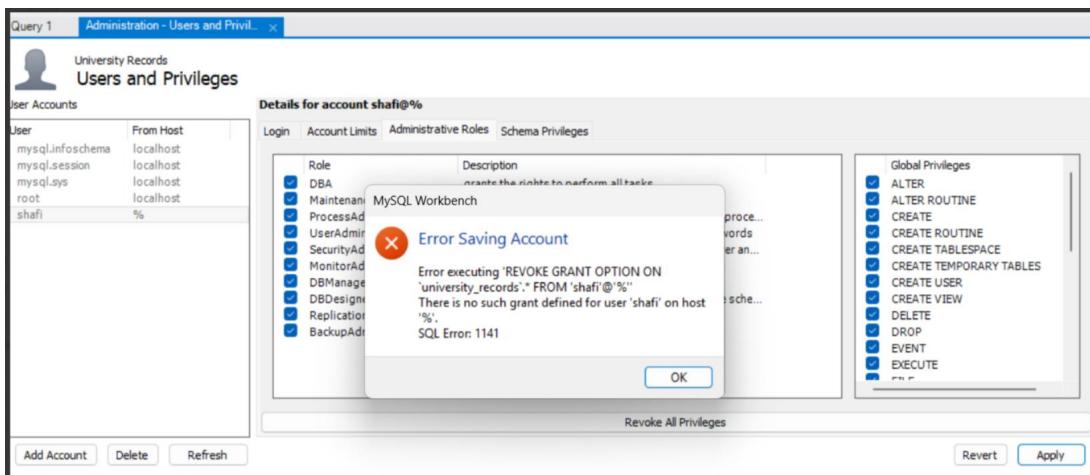


Figure 5.3:Remote DB Access

## 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](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
<b>1. Foundation Setup (24–25 Sep)</b>	- 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
<b>2. Core Development (26–27 Sep)</b>	- MySQL schema creation for students, lecturers, courses, etc. - Insert 20+ sample records - Develop reusable SQL queries and procedures	<i>Milestone 1:</i> Database functional with sample data
<b>3. Integration &amp; Interface Development (28–30 Sep)</b>	- Python–MySQL connection setup - Build CLI or Flask interface - Run live queries and validate data retrieval - Add indexes and optimize database	<i>Milestone 2:</i> Integrated and functional system
<b>4. Finalization &amp; Documentation (1–3 Oct)</b>	- Complete ERD and schema documentation - Conduct final testing (functional + non-functional) - Write report with screenshots - Record and edit demo video	<i>Deliverable 1:</i> Full report and demo video ready
<b>5. Submission and Review (4 Oct)</b>	- Consolidate project files (source, database, docs, video) - Upload final package to GitHub and submit	<i>Milestone 3:</i> Final submission completed

## URMS Project Gantt Chart:

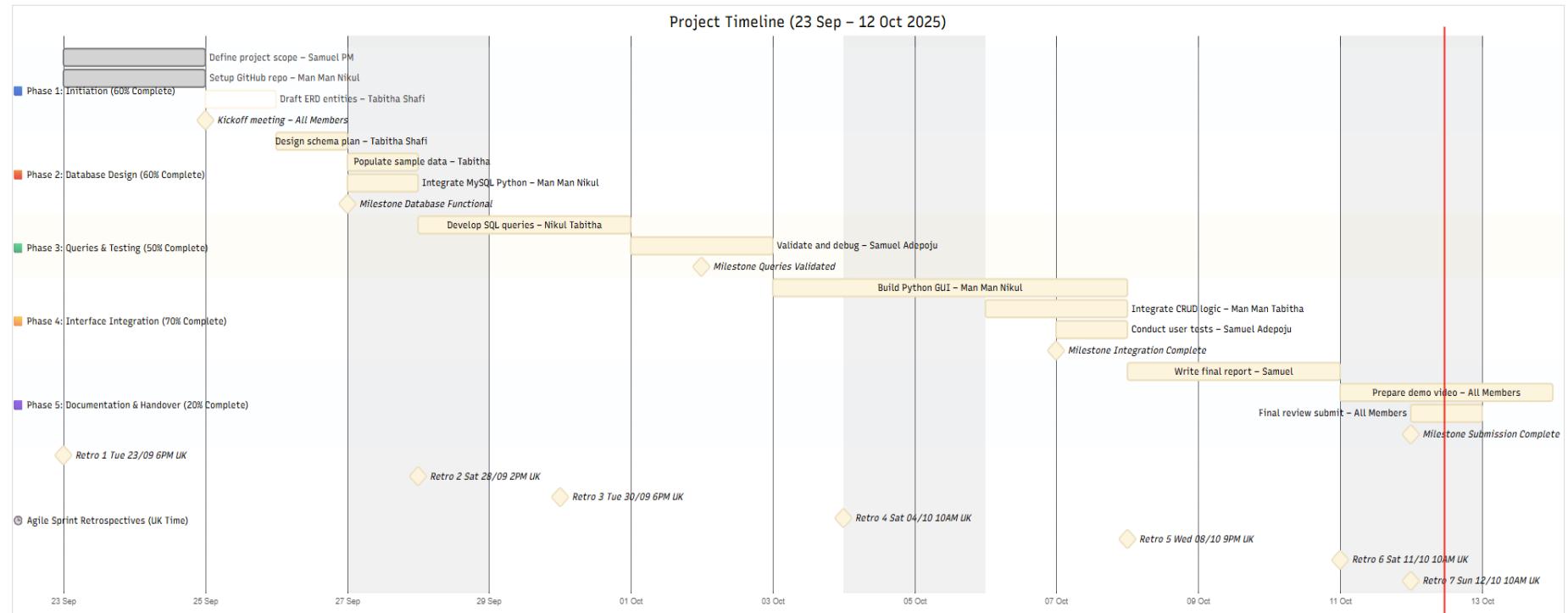


Figure 0-1: Project execution Gantt

## 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)	Optionalit y (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_Enrollment s	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.Preq_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).

```
C:\Users\foluk\PycharmProjects\group-c\.venv\Scripts\python.exe C:\Users\foluk\PycharmProjects\GroupB-End-of-Module-Assigment-UOL\code.py
Testing database connection...
Error: 'cryptography' package is required for sha256_password or caching_sha2_password auth methods
```

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
from sqlalchemy import *
create_engine, Column, Integer, String, Text, Date, DECIMAL,
ForeignKey, TIMESTAMP
)
from sqlalchemy.orm import
from datetime import date
import random
from faker import Faker

# DATABASE CONFIGURATION
```

**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.

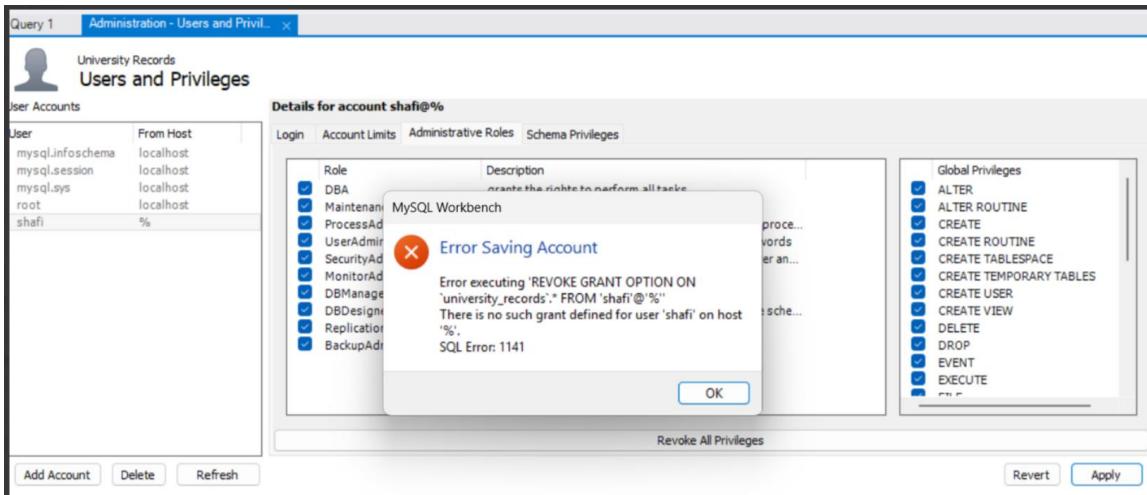
### Mitigations:

- 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 ('shafí'@'%' vs 'shafí'@'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 statements 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.

