COMP3030 - Final Summary Report

Vu Ai Thanh Truong Gia Bao Nguyen Tuan Anh

23-05-2025

Table of contents

1 Conceptual & Physical Design

The project is aimed to provide a yet minimal but still functional of a management application for a education institution. The main actors that interacts to the app contains: `Administrators`, `Instructors`, and `Students`. There activities will go around the `Courses` at school.

1.1 Functional and Non-functional Designs

1.1.1 Functional

1.1.1.1 User Authentication and Authorization:

- The system must allow users (students, instructors, and admins) to log in using email and password credentials.
- Users must be assigned roles (student, instructor, admin) with role-specific access to features (e.g., students can enroll/unenroll, instructors can view/manage courses, admins can manage users and courses).

1.1.1.2 Course Management

- The system must display a list of courses with filters by course name, department, and schedule, supporting pagination (10 courses per page).
- Users can view detailed course information (e.g., ID, name, department, instructor, location, schedule, semester, availability) on a course detail page.
- Instructors and admins can add, edit, and delete courses, while students can only view course details.
- The system must track and display enrolled students for each course.

1.1.1.3 Enrollment Management:

- Students must be able to enroll in a course if availability is greater than zero and they are not already enrolled, with a confirmation dialog before submission.
- Students must be able to unenroll from a course if they are enrolled, with a confirmation dialog before submission.

- The system must update course availability automatically when students enroll or unenroll.
- The system must prevent enrollment if the course is full (availability ≤ 0) or if the student is already enrolled.

1.1.1.4 User Interface and Navigation:

- The system must provide a responsive web interface using Bootstrap for consistent styling across devices.
- Each role (student, instructor, admin) must have a dedicated dashboard with role-specific options.
- Flash messages must be displayed to provide feedback on actions (e.g., success or error messages for login, enrollment).

1.1.1.5 Analytics

- For management roles like `Administrators` and `Instructors`, they will have access to several dashboards to report about the courses at the institution, or the courses that the instructors are involving in.

1.1.2 Non-functional

- **Performance**:
  - Having high uptime, with all errors are being handled to prevent crashes.
  - Queries should execute within a reasonable time, with minimal complexity. Current target is 0.5 seconds for a dataset of 1000 enrollments.
- **Security**:
  - Passwords are encrypted when being stored in the database, with supports of password hashing function `bcrypt`.

  - Access control systems based on roles are designed properly, prevent unauthorized information access or disclosure.
  - Preventions or mitigations of common web security vulnerabilities as well as applications using SQL database system: SQL Injection, Cross-site Scripting, etc.
- **Scalability**:
  - The application is scalable with increasing numbers of users in an acceptable level, against around 300 users under peak time.
- **Usability**:
  - Provide easy-to-navigate interface, with responsive User Interface with `Bootstrap` library.
- **Reliability**:
  - Ensure data integrity with normalized database design (3NF) and constraints.

1.2 Entity-Relationship Diagram

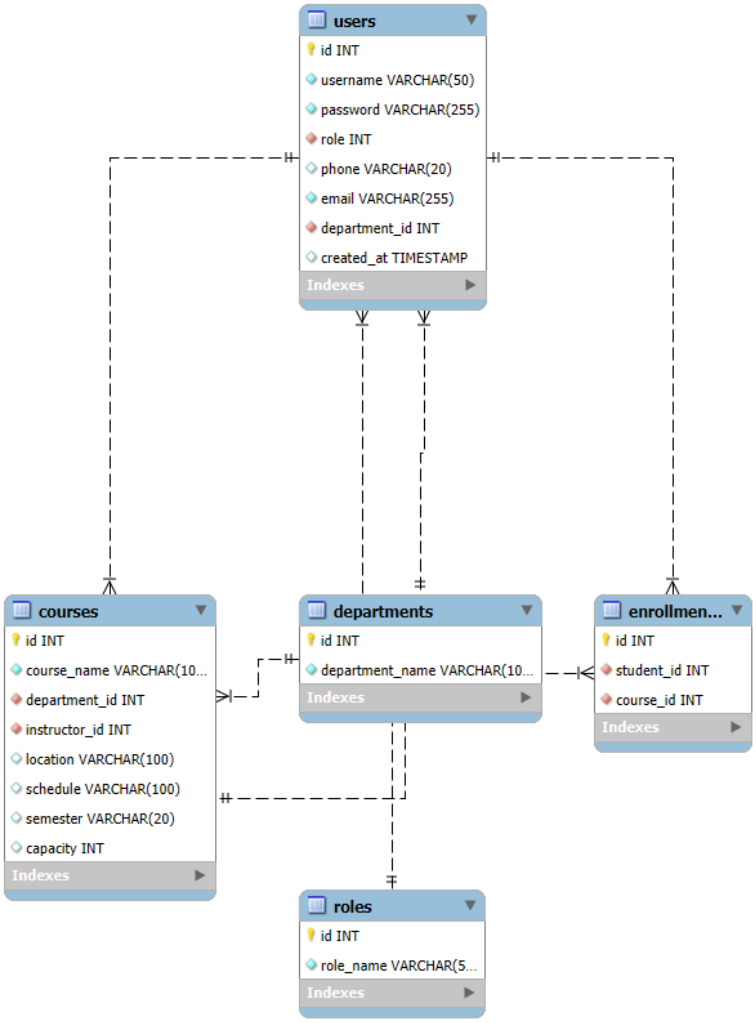


Figure 1: Entity-Relationship Diagram of the project

1.3 Proof of Normalization Form

1.3.1 First Normal Form (1NF):

- All tables have primary keys
- All attributes are atomic
- No repeating groups

1.3.2 Second Normal Form (2NF):

- All non-key attributes are fully functionally dependent on their primary keys
- No partial dependencies exist

1.3.3 Third Normal Form (3NF):

- No transitive dependencies
- Proper use of foreign keys for references
- Related data is properly normalized into separate tables

2 Implement of DB Entities

2.1 Database schema

The full script is in our GitHub repository at this link.

2.2 Views

Currently, there is one view `view_course_details` in the database. It provides a simplified and denormalized view of course information with instructor details. This view joins `courses` and `users` tables to provide course information along with the instructor's name, making it easier to retrieve complete course information without writing complex joins in application queries.

```
CREATE VIEW view_course_details AS
SELECT
    c.id AS course_id,
    c.course_name,
    c.department_id,
    c.instructor_id,
    u.username AS instructor_name,
    c.location,
    c.schedule,
    c.semester,
    c.availability
FROM
    courses c
    LEFT JOIN users u ON c.instructor_id = u.id;
```

2.3 Stored Procedures

There are 03 stored procedures within the database, which are used as quick refrence for most used queries within the lifecycle of the application:

Stored Procedures	Purposes
`get_user_role_counts()`	Collect number of users across roles within the databases, adaptive with future new roles.
`get_course_department_counts()`	Collect number of courses across departments, adaptive with future new departments & courses in the institute.
`get_student_department_counts()`	Collect number of students across departments, adaptive with future new departments, or changing numbers of students
`get_course_count_by_semester()`	Filtering number of courses in each semesters

Table 1: Stored Procedures used in the application

2.4 Triggers

In the functionalities of the application, there are features that automatically update available slots in each course after enrollment. To efficiently operate this feature, we introduced two triggers:

- `before_enrollment_insert()`: This trigger is used to check if there are enough seats left in the course. Whenever a student registered a course, this trigger will reduce the available slots by one. When there is no seat available, it will raise an exception for the business logic to know and handle.
- `after_enrollment_delete()`: In several cases, student may want to drop the course. This trigger simply increase one seat for the course whenever a student unenroll it.

Due to the atomic nature of a trigger in MySQL, there are no race condition error within the the app lifecycle, within the scope of the database itself.

3 Performance Tuning

From our default schema, by design of MySQL, several indexes are automatically created (based on `UNIQUE` keyword, or using `INTEGER`).

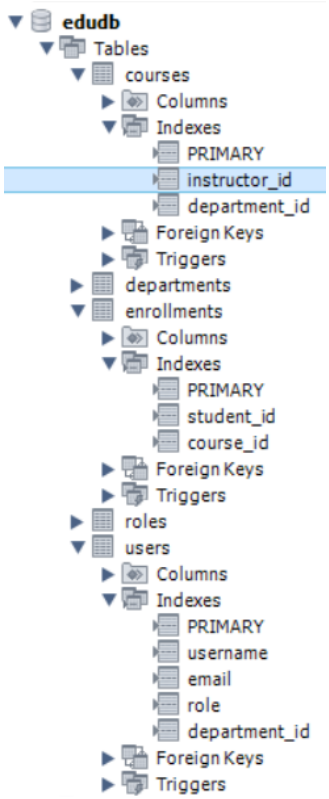


Figure 2: Default indexes in the database

Without any new indexes, normal operations in the database is very fast. For example, the query for a user in the database by email

```
1 •   EXPLAIN SELECT * FROM edudb.users WHERE email = 'instructor_77@gmail.com';
```

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

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	users	NULL	const	email	email	1022	const	1	100.00	NULL

Figure 3: Performance of email matching to user

3.1 Problem with filter

In the application, there is a function to allow users to filter courses based on its schedule. Due to schedule could be any combination of dates within a day, the query could be very slow as there is no current index on `dates`.

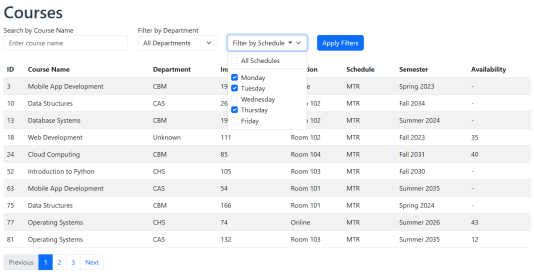


Figure 4: Filtering function of the course search

For example, this query on dates required traversal of all 200 courses in our test database, with no potential keys are used.

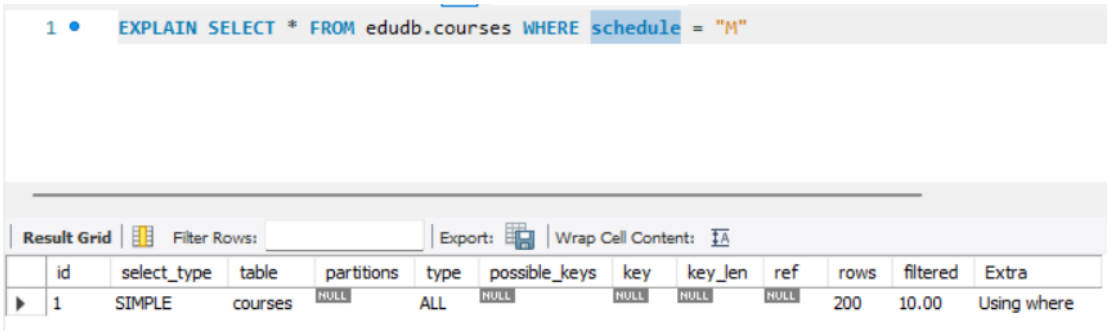


Figure 5: Sample date query

3.2 Selecting over partitions/indexes

To minimize the overhead and complexities of adding more columns, we selected to add an index on table `courses`, over the field `schedule`.

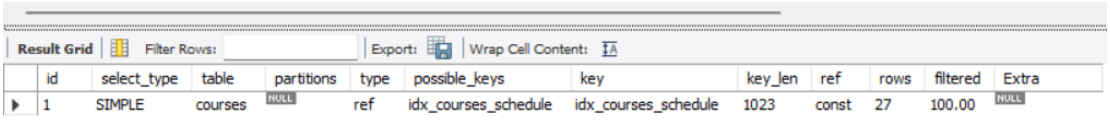
```
CREATE INDEX idx_courses_schedule ON courses(schedule);
```

3.2.1 Failed attempts

1. At first, we tried to used hash partitions, by introducing an user defined function to map the schedule as a bitmask for hashing. However, MySQL do not support custom hash functions.
2. We also tried to use list partitions on the string (as there are only 32 combinations of sorted calendar). However, we found out that the InnoDB engine of MySQL do not support partitions for tables that contains foreign keys.

3.2.2 Performance after tuning

```
1 •   EXPLAIN SELECT * FROM courses WHERE schedule = "M";
```



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

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	courses	NULL	ref	idx_courses_schedule	idx_courses_schedule	1023	const	27	100.00	NULL

Figure 6: Performance after adding the index

With the introduction of this new index, the query performance boost significantly. For example, the database now only needs to filter 19 rows, which are also shared the date is "M". The reason behind this behavior is that, the index introduced created a hash map from newly created `schedule` value. However, compared to represent the schedule as a bitmask, this is slightly slower; but required a much smaller complexity.

4 Security Configuration

4.1 Access Control & Authorization

- The system allow users (students, instructors, and admins) to log in using email and password credentials.
- Users must be assigned roles (student, instructor, admin) with role-specific access to features (e.g., students can enroll/unenroll, instructors can view/manage courses, admins can manage users and courses).

4.2 Password Storage

- For user password, we are not using native functions in SQL. There are two main reasons:

4.2.1 Lack of custom hash functions

MySQL only offers plain cryptographic hash functions, and a lot of them are deprecated like MD5 or SHA1. Also, it only supports AES-256 for symmetric encryption, but it is not are in the scope of our usage for password storage (else, we need to introduce mechanism for user's secret key)

4.2.2 Lack of salt before hashing

For plain usage of cryptographic hash function, user's password is still at risk of rainbow search attack, which attackers attempts to brute-force for the password (for matching hash), or looking up in public database.

- Therefore, passwords in the database are encrypted when being stored in the database, with supports of password hashing function `bcrypt`, which provides the mechanism for a easy to implement password hashing scheme.

4.3 Prevent SQL injection

SQL arguments in the database is implemented using parametrization, which prevents the risk of SQL injection by using `sqlalchemy` to do the input validation instead of the application's business logic. This is also fault-tolerant, to prevent the risk of developer's faults.

```
cursor.execute("SELECT id FROM roles WHERE role_name = %s", (role_name,))
role_id = cursor.fetchone()[0]
cursor.execute(
  "INSERT INTO users (email, username, password, role, phone, department_id)
   ↪  VALUES (%s, %s, %s, %s, %s, %s)",
  (email, username, password, role_id, phone, department_id)
)
```

5 End-to-End Testing & Web Integration

The website provides both the data visualization for analytics, as well as CRUD functions on the objects: users (admin, instructor, student), courses, student enrollment, etc.

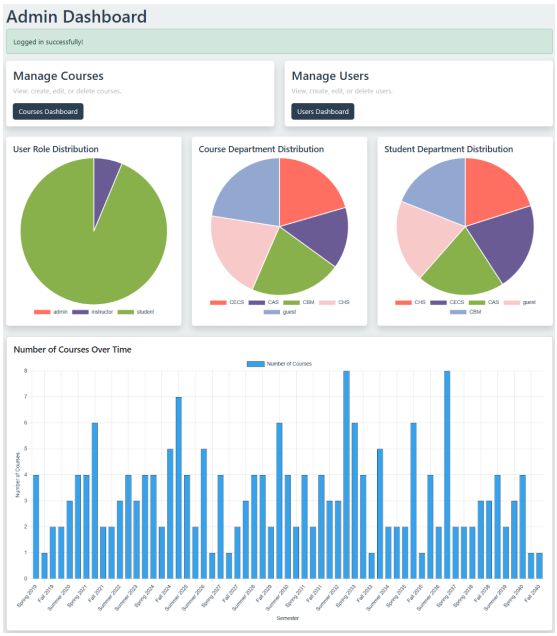


Figure 7: Analytic visualization in the application

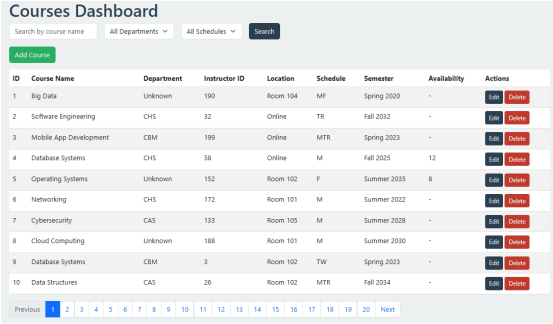


Figure 8: CRUD operations in courses

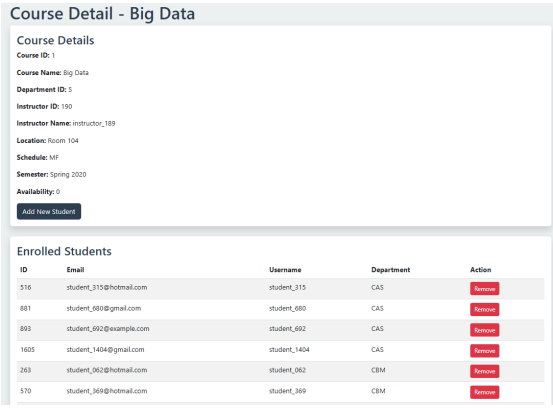


Figure 9: Manage students in courses

More detailed application interaction will be provided in the final presentation.

6 Presentation & Material

- Presentation's slide deck
- Final report
- Source code: https://github.com/Thanh-WuTan/edu-flask-db-project