CS157A Final Project

Summer 2024

Final Report

NewTown Database Application Program

Developed and Curated by:

Rahal, Jashandeep <016585738, jashandeep.rahal@sjsu.edu>

Mesa, Aleatha <016011255, aleatha.mesa@sjsu.edu>

Khant, Edward <015885714, minthwin.khant@sjsu.edu>

Table of Contents

Project Overview

The goal of this project is to develop a database application program for rapidly expanding NewTown Unified School District that provides the framework of such school data as staff, students and courses for data input, storage, query and maintenance by incorporating data validations and constraints while providing a user-friendly interface utilizing texts, forms and checkboxes.

The NewTown Database Application Program is our team members' collaboration toward developing a web application for a school's database management of ten logically related entities - School, Staff, Department, Class, Course, Student, Club, Report, Enrollment and Technology. The application utilizes Java and Apache Tomcat with a MySQL database for data management - Java handles the logic and operations; Tomcat serves as the web execution environment; MySQL manages the persistent data, creating a comprehensive solution for the web application.

The project repository constituting the application and data is hosted in GitHub, and is accessible here, and includes an ER diagram, XML documents, SQL scripts, Java packages - *servlets* and *utility* - along with a readme file for application setup and execution. The application also showcases basic CRUD - Create, Read, Update, and Delete - operations while ensuring data handling and concurrency control. Dummy data is also provided in the repository to demonstrate the app's satisfaction of the project's functional and nonfunctional requirements.

Functional Requirements & Architecture

The NewTown database application is designed to perform basic CRUD functions with the following Java Classes:

1) EnrollServlet.java [C]
    - takes in the following required inputs: first name, last name, grade level, email, address, date of birth;
    - generates a new student ID
    - registers the new student under the previously generated student ID and input values above

2) ClubsServlet.java [R]
    - prints a table constituting two columns labeled “Club” and “Student” and succeeding data rows of the names of enrolled students who have joined a club and the names of the clubs they have joined

3) CourseList.java [R]
    - takes in the following input: course name;
    - prints a table constituting four columns labeled “Class ID,” “Room Number,” “Schedule,” and “Teacher” and succeeding data rows of class IDs, class locations, class schedules and the names of staff teaching the classes

4) Reports.java [R]
    - prints a table constituting three columns labeled “StudentID,” “Grade,” and “Description” and succeeding data rows of student IDs, their grade in class, and their description of academic performance

5) StaffServlet.java [R]
    - prints a table constituting three columns labeled “Name,” “Email,” and “Position” and succeeding data rows of full names of staff members, their email addresses and their positions

6) StudentServlet.java [R]
    - prints a table constituting one column labeled “Name” and succeeding data rows of full names of students registered in the database

7) StudentClasses.java [R]
    - takes in the following input: first name, last name;
    - prints a table constituting three columns labeled “Course,” “Schedule,” and “Room Number” and succeeding data rows of names of the courses the student has enrolled in, each class’s schedule and the corresponding room number

8) TopPerformerServlet.java [R]
    - prints a table constituting two columns labeled “Name” and “Grade” and succeeding data rows of the full names of students who have A grade in the class and the corresponding grade of the student

9) TotalEnrollment.java [R]

- prints the total number of students enrolled in the school

10) UpdateStudentServlet.java [U]

- takes in the following input as a selection value: student ID;
- takes in user input for the following required field values: first name, last name, grade level, email, address, date of birth)

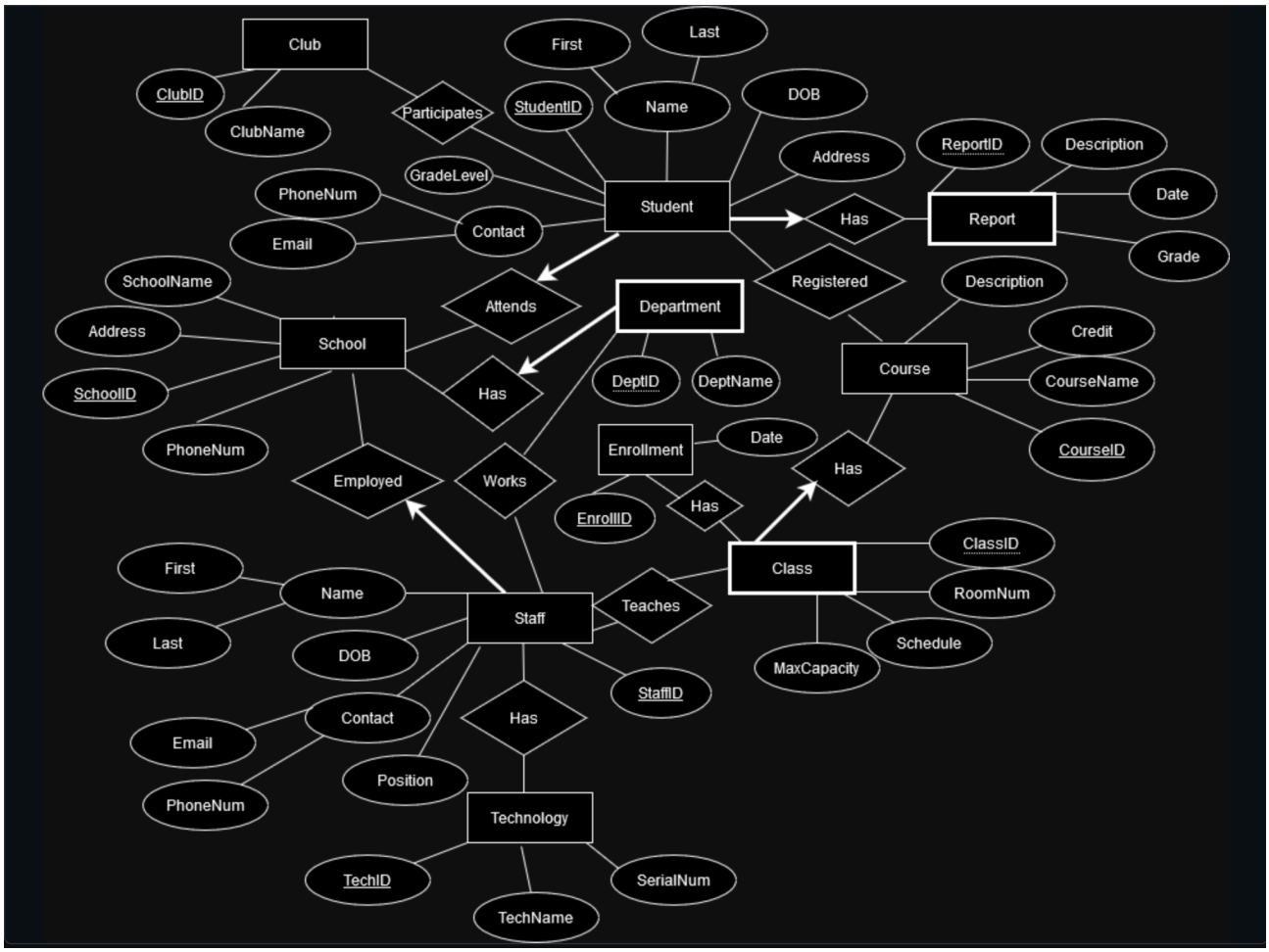
11) DisEnrollServlet.java [D]

- takes in the following inputs: first name, last name;
- delete the record in Students relation that whose values for FirstName and LastName attributes respectively match the input values of first name and last name

In addition, we have also developed the DatabaseUtility class to establish JDBC connection to the MySQL database with set credentials and the DatabaseTest class to test if the database connection is successful. All these Java classes incorporate Java exception handlings and try-catch blocks to avoid system crash as well as provide more specific error messages to the backend user.

In terms of SQL script development, such constraints as NOT NULL constraints, primary key constraints, and foreign key constraints are appropriately applied across all relation definitions and attribute declarations. DROP-CREATE style also ensures that any existing relation with the same name is dropped, providing a clean canvas for a distinct relation to exist. In addition, LOCK-UNLOCK is also incorporated to control concurrent access to tables, ensuring the data integrity during transactions.

Entity-Relationship Data Model



NewTown Database Application - Entity-Relationship Diagram

Entities included in our ER diagram, as alphabetically sorted, are

1) Class
2) Club
3) Course
4) Department
5) Enrollment
6) Report
7) School
8) Staff
9) Student
10) Technology

Database Design

Relations/ Tables

The entities in our ER diagram are essentially the same as relations built in our app development with the latter having a plural sense instead of the former's singular sense. We designed the relations to have basic information in addition to primary keys for each relation, with some relations attributed with one or more foreign keys designated to link in between the relations. The following relations are developed in our team's project:

Classes
- attributes: ClassID, CourseID, StaffID, RoomNumber, Schedule, MaxCapacity;
- primary key: (ClassID);
- foreign key: (CourseID), (StaffID);

Clubs
- attributes: ClubID, ClubName, StudentID;
- primary key: (ClubID, StudentID);

Courses
- attributes: CourseID, CourseName, Description, Credits;
- primary key: (CourseID);

Department
- attributes: DepartmentID, DepartmentName;
- primary key: (DepartmentID);

Enrollment
- attributes: EnrollmentID, StudentID, ClassID, EnrollmentDate
- primary key: (EnrollmentID);
- foreign key: (StudentID); (ClassID);

Reports
- attributes: ReportID, StudentID, Description, GeneratedDate, GradeInClass;
- primary key: (ReportID);
- foreign key: (StudentID);

School
- attributes: SchoolID, SchoolName, Address, Principal, PhoneNumber;
- primary key: (SchoolID);

Staff
- attributes: StaffID, FirstName, LastName, DateOfBirth, Position, DepartmentID, Email, PhoneNumber;

- primary key: (StaffID);
- foreign key: (DepartmentID)

Students
- attributes: StudentID, FirstName, LastName, DateOfBirth, GradeLevel, Address, Email;
- primary key: (StudentID);

Technologies
- attributes: TechnologyName, SerialNumber, AssignedToStaffID, TechID;
- primary key: (TechID);
- foreign key: ('AssignedToStaffID');

RELATIONSHIPS

The following relationships are also developed in order to connect between entities in our design.

1. Has (Staff, Technology)
2. Has (Class, Enrollment)
3. Has (Class, Course)
4. Has (School, Department)
5. Has (Student, Report)
6. Registered (Course, Student)
7. Participates (Student, Club)
8. Attends (Student, School)
9. Employed (School, Staff)
10. Works (Staff, Department)
11. Teaches (Staff, Class)

Major Design Decisions

Frontend: We went with JSP for our frontend because it works perfectly with our Java-based backend. JSP is great for generating content and lets us easily mix Java code with HTML, making it a great fit for our web app.

Backend: We chose Apache Tomcat as our servlet container. It's a web server that's known for its reliability and support for Java Servlet and JSP. Plus, Tomcat is widely used and has tons of documentation, which makes our lives easier.

Database: MySQL was a requirement for our project, and it turned out to be a solid choice. It's reliable, performs well, and handles complex queries. Using JDBC, MySQL integrates very smoothly with our backend.

We designed our servlets to handle data between the frontend and backend using GET and POST requests. This allowed us to link different parts of the app easily and provide easy data transactions. While this works well for our current setup, it can be expanded in the future to be more advanced.

Our application’s architecture supports horizontal scaling, which means we can add more servers to handle increased load as the number of users grows.

We used try-catch blocks to handle exceptions and prevent crashes, ensuring a more user-friendly experience

Implementation Details

Backend

- Programming Language: Java
- Web Server: Apache Tomcat
- DBMS: MYSQL
- Connection: JDBC

Frontend

- User Interface: HTML and JSP

Components of the backend implementation include servlets which are Java classes that handle HTTP requests and responses. Contains sql statements to interact with the database. MYSQL is the DBMS we used which stores the dummy data we generated along with any user input captured from the frontend. We used MYSQL-Connector-J, a JDBC driver, to manage the connection with the database and the application.

For the frontend implementation, we used HTML to design our web pages and JSP to handle the content. Users can navigate through the pages with hyperlinks provided and use the forms to submit information such as student enrollment, updating student's information, etc. There is also a search function where users can search for a specific student to see what class they are enrolled in.

System Demonstration

==========

https://drive.google.com/file/d/1mtlEr8Tidjxp9uuG7_5nnWvo6esmZByH/view?usp=sharing

Conclusion/improvements/lessons learned

Building this web application was a rewarding experience. We managed to create a functional web app that allows users to search for classes, enroll and disenroll from courses, search for clubs, view information about staff and students and more. The project allowed us to apply all the things we learned from class and create a useable application

We gained experience with JDBC, learning how to connect our Java application to a MySQL database, execute queries, and handle data efficiently. Using Apache Tomcat and servlets provided us with insights into server-side programming. We learned how to manage client requests and ensure smooth communication between the frontend and backend. Combining different technologies and concepts into a complete application was an enjoyable process.

Improvements:

We could improve our app by using APIs for data exchange. This would also make it easier to integrate with external services and handle data more efficiently.

Switching to a modern frontend framework like React or Angular could really boost our user interface. These frameworks have great tools for creating dynamic designs, which would enhance the overall user experience. Also spending more time on frontend design and styling would make our app look better and be more user-friendly.

We could make our database interactions faster by refining our SQL queries. This would improve performance and scalability as our user base grows to make sure we aren't calling unnecessary queries.

Adding testing would allow our application to help catch bugs early. This would make the app more reliable and improve user satisfaction.

Overall, this project was a good opportunity to apply everything we've learned in a practical setting. From working with JDBC and MySQL to managing server-side logic with Apache Tomcat and creating dynamic web pages with JSP, we learned a lot. It was nice to see all the pieces come together into a real web app.