

New Jersey Institute of Technology
Term Project Report
CS 631 Summer 2021 - Dr. Vincent Oria
Group Members:
Roland Junior Toussaint, rt28@njit.edu
Muhktar Abare, ma774@njit.edu

Project Summary

This project is to create an application with a user interface in the front end and a back end that connects to a database management system to manage New Jersey Tech entities (students, staff, departments, courses, etc). The application will allow for students to be able to register for classes, taking into account all the requirements and constraints around the student registration process. In addition, the application will be able to list all sections in which students have registered with a lot of other detailed information related to the registration (class list).

Project Requirements

Such an application will allow a new student to be added to the database and also register for courses by providing course code and time or a section number. Furthermore, the application will display a list of such registrations (class list).

1. Student Registration

This program will be used by students to register for courses. The student should provide the code of the course and the time or the section number. The program should then determine whether that particular section has any space available. If there is, then the necessary updates should be done to reflect that the student has been registered for that course. If not, a message should be displayed. This program should allow students to register for a number of courses, one-by-one but to only once for the same student, course, year and semester. In addition, there is a limit to the number of courses a student can take in a semester.

2. Class List Generation

Write a program that generates a class list for each section. The information should include the course code, section code, course name, time, place, weekday(s), instructor and the list of students in the alphabetical order of their last names with their ids, first names, major and year.

3. Student List

Additional requirements consist of the ability to see a list of all the students in the database in the user interface in an HTML table with all student information.

4. Class and Section Information

One more requirement added is for convenience when registering a student to a class. There is a menu item named “classes and Sections” that displays a list of all the classes along with their sections and the week day and time of that particular class for its section.

Application Program Design

There are two main parts, the back-end and the front-end. The back-end itself has two layers within it, a service layer that contains all the logic, and a web layer that exposes the application over HTTP.

The web layer processes HTTP requests coming to the application, and returns HTTP responses. For every HTTP request, the web layer invokes a particular function in the service layer. That function in the service layer will then process the request, access the database, make all the appropriate changes, and then return a response to the web layer. The web layer then sends a response to the caller (client) application.

The service layer contains the logic of the application to meet all the business requirements. It is also responsible for running the queries and accessing the database management system to make the appropriate updates to the tables (the Data Access Object). The service layer makes use of a template to run queries called JdbcTemplate. This is the central class that simplifies the use of JDBC and helps to avoid common errors. It executes core JDBC workflow, leaving application code to provide SQL and extract results. This class executes SQL queries or updates, initiating iteration over ResultSets and catching JDBC exceptions and translating them to the generic, more informative exception hierarchy defined in that class - [Source](#). This template also allows to gracefully bypass exceptions and permit the application to continue to the next line of the code logic, which was extremely important for this project to meet all the requirements.

One major design decision was to determine if the application should try to register a student with the student information and the section number, and if the section is not present, to use the course code and the time of the section for that course. This responsibility was placed on the user, by asking them to either enter the course code and the time along with the student information, OR provide the student information and the section of the course. The logic then turned out to check for the present of the

course code and the time, if this information is present, the application will proceed to register the student with this information, and if the registration fails, the application will not attempt to register the student with the section number, since it is an either-or situation. If the course number and the time were not provided, then the application logic will proceed with the student information and the section number to register the student. And if this fails, the application will not attempt to register the student using course and time information.

The front-end of the application uses client-side, web technologies, mainly HTML, CSS, JavaScript, and the JavaScript HTTP client library. The user interface in the front-end contains a menu where the user can click to fetch data, register a new student, and an additional functionality to display all the student information.

Upon opening the application, you are presented with a message that lets you know all the students have been loaded and you see a listing of all the students. There is another button on the top menu that will take you to the form to register a student. That form takes in the student information, the course code and time, OR the student information and the section number. If during the registration, a validation error occurs, the application will display a message to the user to let them know what the issue was. If the registration was successful, an appropriate message is also displayed to let the user know the operation was successful. The last button on the menu will display the class list. It shows a list of all current student registration, with student, instructor, time, building, location, etc information for that particular student registration.

Analysis

Our schema consists of 11 relational tables in total. These relations include the Students, Departments, Courses, Staff, Sections, Registrations, Buildings, Rooms, SectionInRooms, Assignments and FacultyDepartment tables respectively.

While assessing our Students relation we decided to introduce the student_id attribute as the primary key in order to distinguish students who may share the same name or other pieces of information. We also utilized this attribute as a foreign key for the Registration relation in order to grant us the ability to access which particular student registered for what course. Being that we used the student_id attribute as the foreign key here, we can now gain the registration information of students that may share the same characteristics. This will successfully prevent our queries from treating said students as duplicates.

Within the Registration relation, we also utilized the course_number and section numbers attributes as foreign keys from the Courses and Sections relations respectively. This allows us the ability to pull the course and sections that align with what each student registered for. In addition, these two attributes also serve as the composite key of the Section relation because they are codependent. Each course must have a section and each section a course.

Also within that same Section relation, we set the staff_ssn as a foreign key from our Staff relation to assess the sections each staff member belongs to. Similar to the students table we set the staff_ssn rather than the name of the staff members as our primary key for the Staff relation to avoid our queries from treating each staff member as a duplicate in the case that they may share characteristics. All three attributes(staff_ssn, course_number and section_number) were also used as foreign keys to construct the Assignments relation in order to display all the staff members that teach a particular course.

Moreover, we created a Departments relation to display all the departments that exist within the school. We set the department_code attribute as the primary key and utilized that as well as the staff_ssn attribute from the Staff relation as foreign keys to construct the FacultyDepartments relation. This relation essentially shows us all the staff members and the departments each one belongs to. The departments_code attribute also serves as a foreign key in the Courses relation to illustrate what department each course belongs to.

As for location, we constructed the Buildings, Rooms and SectionInRoom relations to determine the locations of each course which inadvertently tells us where each student and staff member would be depending on the class they are taking or teaching. We set the building_number as the primary key for the Buildings relation and matched it with the room_number attribute to form the composite key for the Rooms relation. The reason being that they are codependent, each building has to have a room and each room a building. We then utilized both of these attributes as well as the course_number and section_number attributes to form foreign keys for the SectionInRooms relation. Furthermore, we included the weekday and time attributes to the relation so we could perceive where, when and what particular course would be in session at their respective times and locations.

Entity-Relationship Design

Below is a systematic breakdown of our database schema:

Within the “Students” relation there exist eight attributes starting with student_id, which serves as the primary key of the Students table. This is then followed by the student_ssn attribute which is in turn followed by the student_first_name, student_last_name, student_address, student_high_school, student_year and student_major attributes. All of these attributes possess the Varchar data type with the exception of our primary key student_id, which possesses a Bigint unsigned data type that auto increments. Moreover all of the attributes are null by default with the exception of the student_id and the student_ssn attributes.

Our next relation “Departments”, consist of four attributes. The first attribute is called department_code and it possesses a Bigint unsigned data type that is not null and auto increments. It also serves as the primary key of this relation. This is then followed by the department_name attribute which has a Varchar data type and is null by default. After this attribute is the department_budget attribute which contains a decimal data type and is null by default. Lastly we have the department_building_id which possesses a Bigint data type and is not null.

Furthermore, our “Courses” relation contains five attributes. The course_number attribute serves as the primary key and has a Bigint unsigned data type that auto increments and is not null. The second attribute course_name has a Varchar data type and is null by default. The third attribute course_credit and the fourth attribute student_ta_hr_required both possess a Smallint data type which has a default value of zero. Lastly, the department_code attribute also has a Bigint unsigned data type that is not null. This attribute also serves as a foreign key that is reference to the department_code attribute in the Departments relation.

The Fourth relation “Staff”, contains five attributes. The first in line is the staff_ssn attribute which serves as the primary key and possesses a Bigint unsigned data type which is not null. We also have two attributes within this relation that possess the Varchar data type and are not null. These attributes are staff_name and staff_rank. The staff_salary attribute within the relation holds a decimal datatype and has a default value of 0.00. Our last attribute within the relation, which is staff_course_load holds a Tinyint data type and has a default value of 0.

In addition, the “Sections” relation contains six attributes in total. Three of which possess the Bigint unsigned data type and are not null. These three attributes are section_number which auto increments, course_number and staff_ssn. Out of these three, section_number and course_number serve as the composite key while staff_ssn serves as the foreign key referencing the staff_ssn in the Staff relation. Moving on, this relation also includes the section_year attribute which holds a Smallint data type,

section_max_enroll attribute which holds a Tiny int data type and a default value of 0, and lastly the section_semester attribute which holds a Varchar data type which is null by default.

The following relation “Registrations” contains three attributes. All carry the Bigint unsigned data type, are not null and serve as foreign keys. The first attribute student_id references the student_id attribute in the Students relation. The second attribute section_number references the section_number attribute within the Sections relation. As for the third attribute course_number, it references the course_number attribute within the Courses relation.

The seventh relation we have within our database schema is the “Buildings” relation. This relation contains three attributes starting with the primary key building_id, which has a Bigint unsigned data type that is not null and auto increments. The remaining two attributes are building_name and building_location. Both these attributes hold a Varchar data type and are null by default.

Moreover, our eighth relation is the “Room” relation. This relation carries four attributes. The building_id and room_number attributes which auto increments both serve as the composite key, are not null and hold the Bigint unsigned data type. The building_id attribute also serves as a foreign key referencing the building_id attribute within the Buildings relation. The room_capacity attribute holds the Smallint data type and holds a default value of one. The last attribute, room_audio_visual holds a Boolean data type which is false by default.

In addition the ninth relation is the “SectionInRooms” relation. It contains 5 attributes, three of which are foreign keys that hold Bigint unsigned data types and are not null. The building_id attribute references the building_id attribute within the Building relation, the room_number attribute references the room_number attribute in the Rooms relation, and the course_number attribute references the course_number attribute in the Courses relation. As for the section_in_room_weekday and section_in_room_time attributes, they hold a datatype value of Varchar and are null by default.

The tenth relation within our database schema is the “Assignments” relation. This consists of the staff_ssn attribute, the course_number attribute and the section number attribute. All hold the Bigint unsigned data type and are not null. They also serve as foreign keys referencing the attribute staff_ssn in the Staff relation, the attribute course_number in the Course relation and the attribute section_number in the Sections relation.

The last relation within our database schema is the “FacultyDepartment” relation. This particular relation holds two attributes which both serve as foreign keys. The first attribute staff_ssn holds a Bigint unsigned data type, is not null and references the staff_ssn attribute in the Staff relation. The second attribute department_code also holds a Bigint unsigned data type, is not null but references the department_code attribute in the Departments relation.

We then constructed a query by making a series of selections and joining all the relations above to generate the class list for each section that would yield the following results: course code, section code, course name, time, place, weekdays, instructor and list of students with their ids, majors, years, first names and last names in alphabetical order.

Accomplishments

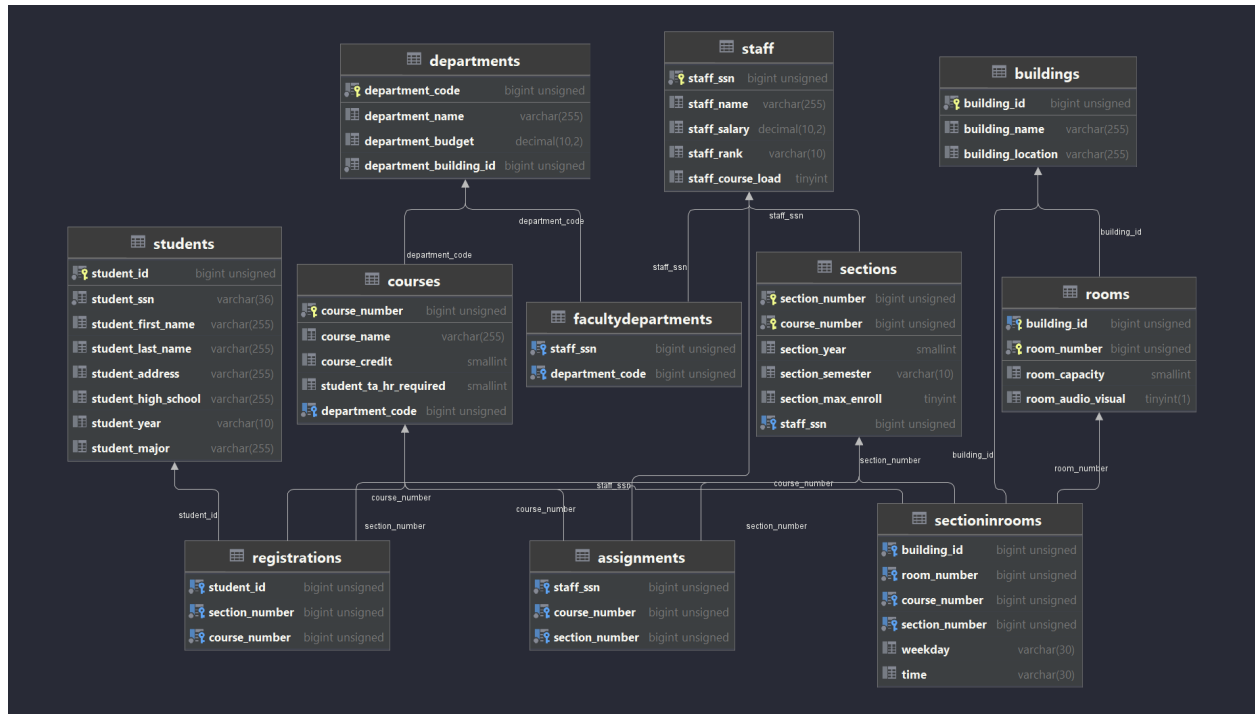
The application is composed of two main parts; the front end which is built using HTML, CSS, and JavaScript, and the back end running on Java. Connectivity to the MySQL database is made through the Java code. Endpoints exposed over HTTP allow the application to be accessible via a web browser or any HTTP client. HTTP requests are made from the front end to the back end to communicate with the application.

The schemas are all created along with all the relations between tables, all stored in a file named schema.sql. The data to populate the tables with tuples is also created and stored in a file called data.sql. When the application runs, it loads up both schema and data files and executes all SQL queries which create the tables and populate them with some initial data.

All the application logic (application requirements) are coded in the back end using Java. For every logical constraint, the application will throw an exception with a message detailing what caused the exception to occur and what possible actions the user can take to make the operation successful on subsequent tries.

The application is accessible via web, with a menu at the top that indicates the different operations and functionalities of the application. It displays a list of all the students, a form to register a student to a class (requirement 1), and a list of classes with student, section, time, location, and instructor information (requirement 2).

Schema Diagram



Student List

NJ Tech

localhost:4200

Guest













NJ Tech ManagementTerm Project Application - CS 631 Summer 2021 - Dr. Vincent Oria

Students Information

Register Student

Class List

STUDENTS INFORMATION

Avatar	ID	SNN	Name	High School	Major
	18	545-35-1491	Andrew Truong	City Charter High School	Biology
	19	620-22-9811	Ascanio Ramos Reyes	Avonworth High School	Computer Science
	5	529-22-8288	Christian Carpena	Downingtown West High School	Child Care Services Management
	17	315-04-4775	Claire Wang	Deer Lakes High School	Chemistry
	1	230-29-1819	David Amajor	West Chester East High School	Data Management Technology
	15	529-22-8288	Diana Zawislak	Keystone Oaks High School	Agriculture, General
	4	123-85-0048	Jake Byford	The Phelps School	Information Science
	10	373-60-4918	Kevin La	Rochester Area High School	Accounting
	6	481-93-3849	Naga Akhil Desabattina	Downingtown East High School	Family & Consumer Sciences
	11	230-29-1819	Prem Kumar Maharajan	Monaca High School	Cinema/Film
	13	472-18-5146	Rahul Patel	Serra Catholic High School	Agricultural Economics
	20	685-02-0051	Roland Junior Toussaint	Avonworth High School	Computer Science

Registration Form

NJ Tech

localhost:4200

Guest

NJ Tech ManagementTerm Project Application - CS 631 Summer 2021 - Dr. Vincent Oria

Students Information

Register Student

Class List

REGISTER STUDENT

Student ID(always required)

Student ID, such as 1, 2, 3

Course Code

Course Code, such as 1, 2, 3

Course Time

Course Time, such as 08:00-10:00 am, 10:00-12:00 pm

OR Section Number

Section Number, such as 1, 2, 3

Register Student

Cancel

Classes and Sections

NJ Tech

localhost:4200

Guest

NJ Tech ManagementTerm Project Application - CS 631 Summer 2021 - Dr. Vincent Oria

Students Information

Classes and Sections

Register Student

Class List

CLASS AND SECTION INFORMATION (9 total)

Course Name	Course Code	Section Number	Week Day	Time
Calculus 101	6	5	Thu-Fri	05:00-08:00 am
Data Mining	1	8	Wed-Thu	10:00-12:00 pm
Data Structures and Algorithms	2	6	Mon-Wed	10:00-02:00 pm
Intro to Abstract Algebra	7	4	Wed-Thu	11:00-02:00 pm
Intro to Database Management	3	2	Tue-Fri	12:00-01:30 pm
Intro to French	4	6	Mon-Wed	06:00-05:30 am
Intro to Nursing Profession	8	3	Mon-Wed-Fri	08:00-10:00 am
Mental Health Nursing	9	9	Tue-Thu	08:00-10:00 am
Network Programming	5	1	Mon-Wed	09:00-11:00 am

Class List

NJ Tech

localhost:4200

Guest

NJ Tech ManagementTerm Project Application - CS 631 Summer 2021 - Dr. Vincent Oria

Students Information

Register Student

Class List

CLASS LIST

Course Code	Section Code	Course Name	Week Day	Time	Building	Building Location	Instructor	Student ID	Student Name	Student Major	Student Year
9	9	Mental Health Nursing	Tue-Thu	08:00-10:00 am	Shagbark Hickory	97 West Road, NW campus	Tonnie Matuszewski	5	Christian Carpena	Child Care Services Management	1st Year
2	6	Data Structures and Algorithms	Mon-Wed	10:00-02:00 pm	Polypody Fissidens Moss	209 Welch Parkway, NJ campus	Bord Piesing	20	Roland Junior Toussaint	Computer Science	3rd Year
1	7	Data Mining	Wed-Thu	10:00-12:00 pm	Chaenothecopsis Lichen	1 Kensington Alley	Aridatha Allabush	20	Roland Junior Toussaint	Computer Science	3rd Year
6	4	Calculus 101	Thu-Fri	05:00-08:00 am	Pink Turtlehead	167 Superior Junction, NJ campus	Lawry Emtage	20	Roland Junior Toussaint	Computer Science	3rd Year
6	4	Calculus 101	Thu-Fri	05:00-08:00 am	Pink Turtlehead	167 Superior Junction, NJ campus	Lawry Emtage	17	Claire Wang	Chemistry	1st Year
2	6	Data Structures and Algorithms	Mon-Wed	10:00-02:00 pm	Polypody Fissidens Moss	209 Welch Parkway, NJ campus	Bord Piesing	17	Claire Wang	Chemistry	1st Year

Risks and Issues

Data integrity has been one of the challenges faced when building this application. For instance, if you try to register a student by passing in the student ID and the section code, you may get a foreign key constraint failure. The reason for this is because even

though a section exists with a course code, a year, etc, that section also needs to be in the SectionsInRooms table. Otherwise, when trying to load up all the class list, you may not see a student in that list because the condition on the join for that SectionsInRooms table and the course table will return false, therefore will not include that registration in the class list. Making sure that all available sections were also included in the SectionsInRooms table solved the issue.

Another issue was with SQL queries that returned no result that were causing exceptions in the logic. This was problematic because when an exception occurs, the application stops executing even though the application logic could take another course of action and still run to completion, and throw an exception. A concrete example of this was with the possibility of having the course code and time and the student ID to register the student to a class, or the student ID and the section code. Part of the logic checks to make sure the student is not already registered to a class, a query is run to find that student in the registration table for that class. If the student is not yet registered for the course, the query result then returns an exception. To fix this issue, the code was changed to gracefully handle the exception in order to let the logic continue its course.

Resources

Roland Junior Toussaint - Developer
Muhktar Abare - Developer

Software

MySQL Community Server
MySQL Workbench
IntelliJ IDEA
Visual Studio Code
Java Virtual Machine (Java)
HTML, CSS, JavaScript (Google Angular Framework)
Postman API Client (HTTP Client)
Google Chrome