CS 875 - DATABASE SYSTEMS

UNIVERSITY DATABASE SYSTEM DESIGN

Name	Vineel Gannu
Student No.	200439673
Department	Computer Science
Email	vgc361@uregina.ca

Contents

1. Problem Statement

The University of Canada is an institution that provides different courses to various students. There are huge data associated with the university such as the student details, course details, faculty details, department details, and much more. File systems can be used to store university data. The issue with storing the data using file systems is that the data is inconsistent and there is a possibility of having redundant data. To overcome the problems of file systems, a Database Management System (DBMS) for the university data is designed and developed. Storing the data in a DBMS helps the users to efficiently access the data, enforce the constraints while updating the data and the users can concurrently access the data. A web application is created through which the data is accessed from the databases by different users.

2. Methodology

The following steps are involved in the project while designing the database.

2.1 Requirement Analysis

Firstly, the existing system is identified and analyzed. The University of Canada uses file systems to store and manipulate the data. As this is a tedious task, the University of Canada has decided to store the data in the database. To do this, the database design experts would start their work with the Requirement Analysis step where the data that is to be stored is identified. A concise information is given by the university to the database design experts on what data is to be stored and what sort of data is accessed by different users. On a high-level, there are 3 different users - student, instructor, and admin. The student can access the data such as the student details, the currently enrolled courses, and the previously enrolled courses. The instructor can access the data such as the instructor details, the currently teaching courses, and the previously taught courses. The student and the instructor can access the data through the web application whereas the admin has complete access to the entire DBMS.

2.2 Conceptual Database design

Once the requirements are analyzed, a high-level data description is developed. There are different data models that can describe the data and one such model is the Entity-relationship model. The Entity-relationship model shows different entities that are connected through various relationships. Figure 2.1 shows the Entity-relationship diagram and table 2.1 shows the entity and different entity attributes for the university database design.

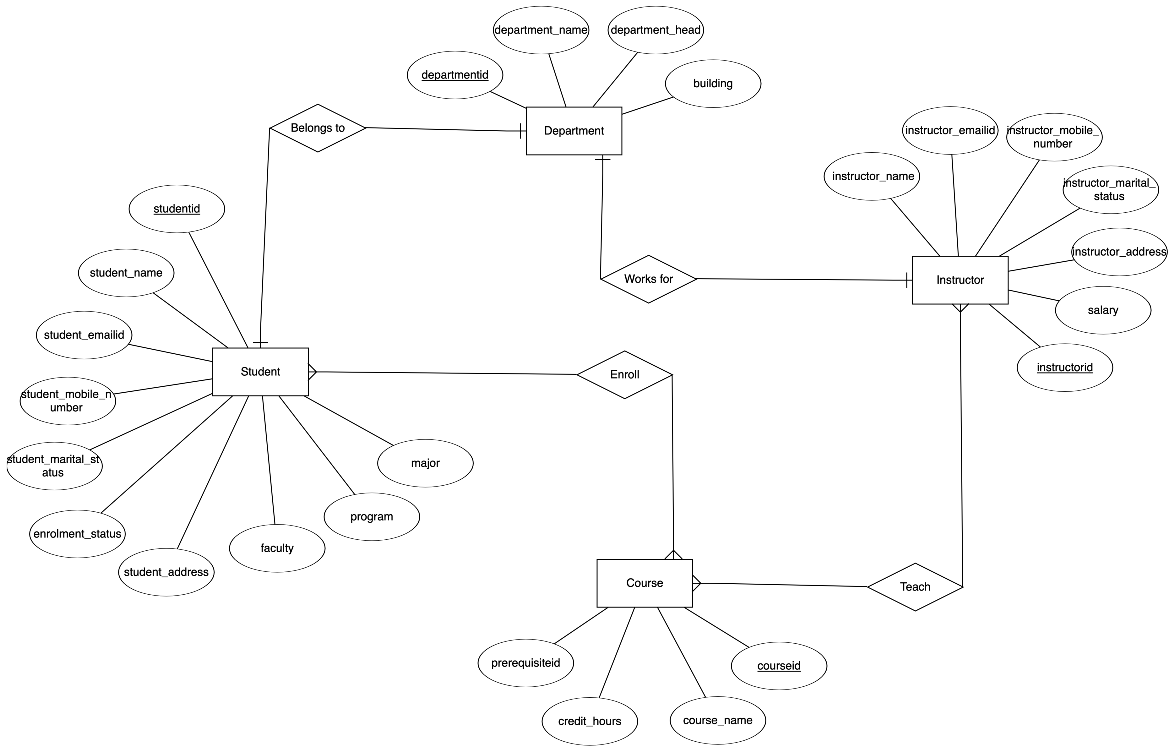


Figure 2.1: Entity-relationship diagram for the university database

For this project, we considered four different entities - Student, Course, Instructor and Department with various attributes. The entities are joined through relationship. Few of the examples are:

- Entity Student belongs to the Entity Department (one-to-one relationship).
- Entity Student opts for more than one course and the same course is opted by more than one student.(many-to-many relationship).

2.3 Logical Database design

To implement the database design, we need to choose a DBMS and then convert the conceptual database design obtained in the above step into a relational database schema. A relational model is used to logically represent the database. It describes how data is stored

Table 2.1: Entity and Entity Attributes

Entity	**Attributes**
Student	studentid, student_name, student_emailid, student_mobile_number, student_marital_status, enrolment_status, student_address, faculty, program, major
Course	courseid, prerequisiteid, course_name, credit_hours
Instructor	instructorid, instructor_name, instructor_emailid, instructor_mobile_number, instructor_marital_status, instructor_address, salary
Department	departmentid, department_name, department_head, building

in the databases. The data in the relational model is stored in the form of relations which are also known as tables. The attributes that are obtained in the conceptual database design are converted to columns in this step. The appropriate columns are identified in each relation and are considered as the primary keys. Figure 2.2 shows the relational model for the university database where there are six different relations. In the entity-relationship model, the entity Student belongs to the entity Department (one-to-one relationship). To represent the one-to-one relationship in the database design, the primary key of one relation is added to another relation. The departmentid which is a primary key in the Department relation is added as a column (foreign key) in the Student table. Similarly, the departmentid is added as a column in the Instructor table. The entity Student opts for more than one Course and the same Course is opted by more than one Student. (many-to-many relationship). To represent the many-to-many relationship in the database design, a new relation is created with the primary keys of both the relations as columns along with some additional columns. Thus, a relation named Student_Enrolled is created. Similarly, a relation named Instructor_Enrolled is created. In Instructor_Enrolled relation, an instructor can teach the same course for a different term and year. Therefore, all the four columns in the Instructor_Enrolled relation acts as primary keys. The relation schema in the relational model are as follows:

1. Student(studentid: string, department_id: string, student_name: string, student_emailid: string, student_mobile_number: integer, student_marital_status: string, enrolment_status: string, student_address: string, faculty: string, program: string, major: string)

2. Instructor(instructorid: string, department_id: string, instructor_name: string, instructor_emailid: string, instructor_mobile_number: integer, instructor_marital_status: string, instructor_address: string, salary: int)

3. Department(departmentid: string, department_name: string, department_head: string, building: string)

4. Course(courseid: string, prerequisiteid: string, course_name: string, credit_hours: int)

5. Student_Enrolled(studentid: string, courseid: string, semester: string, grade: string, term: string, year: int)

6. Instructor_Enrolled(courseid: string, instructorid: string, term: string, year: int)

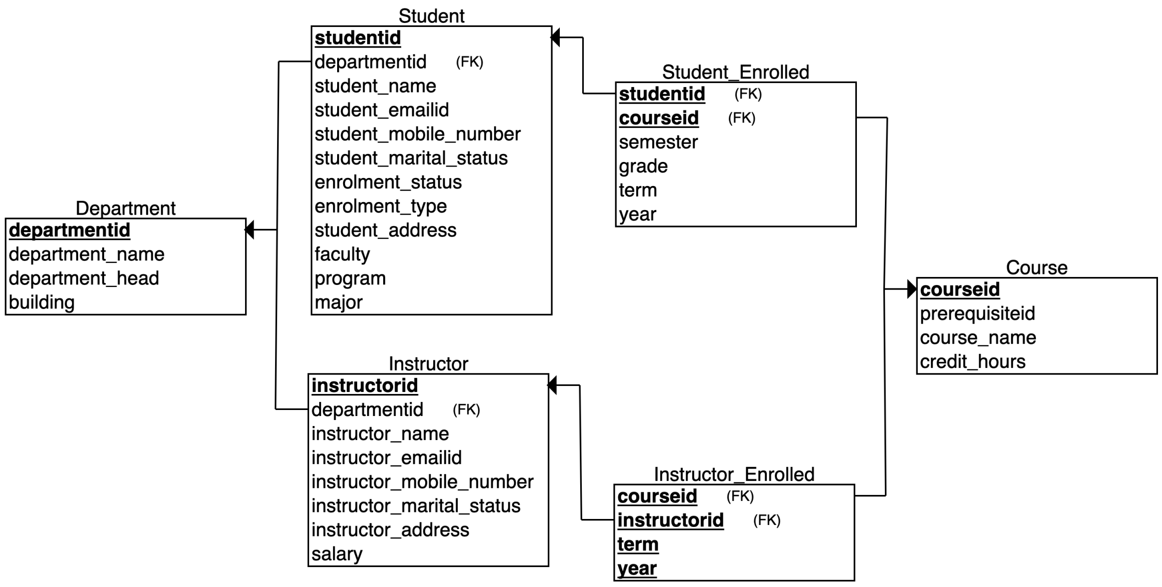


Figure 2.2: Relational model for the university database

2.4 Schema Refinement

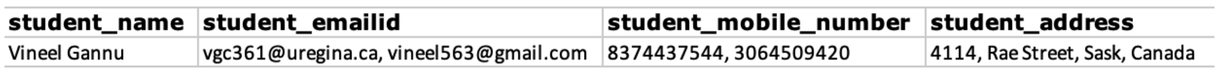
In this step, the relations that are obtained in the relational database schema are analyzed to identify the potential problems. These problems can be solved using Normalization. Database Normalization is the process of decomposing relations to avoid data redundancy. There are different normal forms in the database normalization. Normalization is done so that data redundancy can be reduced and the update anomalies can be avoided. Before normalizing the relation schema, the relations and the columns for the university database are as that of in table 2.2.

Table 2.2: Relations and Columns before Normalization

Relations	**Columns**
Student	studentid, departmentid, student_name, student_emailid, student_mobile_number, student_marital_status, enrolment_status, enrolment_type, student_address, faculty, program, major
Course	courseid, prerequisiteid, course_name, credit_hours
Instructor	instructorid, instructor_name, instructor_emailid, instructor_mobile_number, instructor_marital_status, instructor_address, salary
Department	departmentid, department_name, department_head, building
Student Enrolment	studentid, courseid, semester, grade, term, year
Instructor Enrolment	instructorid, courseid, term, year

2.4.1 First Normal Form

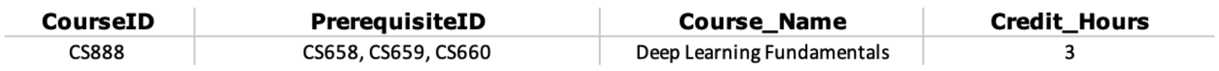
First Normal Form states that the values in the domain(A) are atomic for every attribute A in the relation schema R [2]. Figure 2.3 shows the part of a relation instance of Student relation before the first normal form. The values for the columns student_name, student_emailid, student_mobile_number, student_address are not atomic. For the Student relation to be in the first normal form, these column values are to be atomic that can be obtained by decomposing these columns. Although the student_name is atomic, it can still be decomposed into two different columns. Similarly, the values of the columns instructor_name, instructor_emailid, instructor_mobile_number, instructor_address of the relation Instructor are not atomic and the relation can be in first normal form by decomposing these columns.



student_name	student_emailid	student_mobile_number	student_address
Vineel Gannu	vgc361@uregina.ca, vineel563@gmail.com	8374437544, 3064509420	4114, Rae Street, Sask, Canada

Figure 2.3: Part of a Relation Instance of Student Relation Before First Normal Form

Courseid and Prerequisiteid are few columns in the Course relation. A course can have more than one prerequisite course and it is not a good idea to add additional columns to the Course relation to make the prerequisiteid atomic. This can be solved by adding only one prerequisiteid in each row. Figure 2.4 shows the relation instance of Course relation before the first normal form. Table 2.3 shows the relations and columns after the first normal form along with the primary key(s).



CourseID	PrerequisiteID	Course_Name	Credit_Hours
CS888	CS658, CS659, CS660	Deep Learning Fundamentals	3

Figure 2.4: Relation Instance of Course Relation Before First Normal Form

2.4.2 Second Normal Form

A relation is in second normal form if it is in first normal form and every non-prime attribute is fully dependent on every CK of the relation schema [2]. Let us consider a set of functional dependencies for the Course relation as follows:

$$F_{course} = \{\{courseid\} \rightarrow \{course\_name, credit\_hours\}, \{courseid, prerequisiteid\} \rightarrow \{course\_name, credit\_hours\}\}$$

The non-prime attribute course_name is partially dependent on the candidate key { courseid, prerequisiteid }. Thus, the Course relation is not in second normal form and it can be decomposed into two different relations with one being the existing Course relation from

Table 2.3: Relations and Columns after First Normal Form

Relations	**Columns**	**Primary Key(s)**
Student	studentid, departmentid, student_first_name, student_last_name, student_primary_emailid, student_secondary_emailid, student_primary_mobile_number, student_secondary_mobile_number, student_marital_status, enrolment_status, enrolment_type, student_house_number, student_city, student_province, student_country, student_zip_code, faculty, program, major	studentid
Instructor	instructorid, instructor_first_name, instructor_second_name, instructor_primary_emailid, instructor_secondary_emailid, instructor_primary_mobile_number, instructor_secondary_mobile_number, instructor_marital_status, instructor_house_number, instructor_city, instructor_province, instructor_country, instructor_zip_code, salary	instructorid
Course	courseid, prerequisiteid, course_name, credit_hours	courseid, prerequisiteid
Department	departmentid, department_name, department_head, building	departmentid
Student Enrolment	studentid, courseid, semester, grade, term, year	studentid, courseid
Instructor Enrolment	instructorid, courseid, term, year	instructorid, courseid, term, year

which the column prerequisiteid is removed and the other relation is the Prerequisite relation with the columns courseid and prerequisiteid, of which both are primary keys. Once the relation is decomposed, both the obtained relations are in second normal form. Since all other relations(except Instructor Enrolment) has only one primary key, they are in second normal form by default. Since all the columns in the Instructor Enrolment are primary keys, the relation is in the second normal form. Table 2.4 shows the relations, columns and primary key(s) after the second normal form.

Table 2.4: Relations and Columns after Second Normal Form

Relations	**Columns**	**Primary Key(s)**
Student	studentid, departmentid, student_first_name, student_last_name, student_primary_emailid, student_secondary_emailid, student_primary_mobile_number, student_secondary_mobile_number, student_marital_status, enrolment_status, enrolment_type, student_house_number, student_city, student_province, student_country, student_zip_code, faculty, program, major	studentid
Instructor	instructorid, instructor_first_name, instructor_second_name, instructor_primary_emailid, instructor_secondary_emailid, instructor_primary_mobile_number, instructor_secondary_mobile_number, instructor_marital_status, instructor_house_number, instructor_city, instructor_province, instructor_country, instructor_zip_code, salary	instructorid
Course	courseid, course_name, credit_hours	courseid
Prerequisite	courseid, prerequisiteid	courseid, prerequisiteid
Department	departmentid, department_name, department_head, building	departmentid
Student Enrolment	studentid, courseid, semester, grade, term, year	studentid, courseid
Instructor Enrolment	instructorid, courseid, term, year	instructorid, courseid, term, year

2.4.3 Third Normal Form

A relation schema is in third normal form with respect to a set F of FDs if no non-prime attribute in relation schema is transitively dependent upon a candidate key of the relation schema [2]. Let us consider a set of functional dependencies for the Student relation as follows:

$F_{student} = \{\{studentid\} \rightarrow \{student\_zip\_code\}, \{student\_zip\_code\} \rightarrow \{student\_city, student\_province, student\_country\}\}$

The non-prime attributes {student_city, student_province, student_country} are transitively dependent upon the candidate key studentid and thus the Student relation is not in third normal form. The Student relation is decomposed into two different relations in such a way that the decomposed relations are in third normal form. Similarly, in the Instructor relation, the non-prime attributes {instructor_city, instructor_province, instructor_country} are transitively dependent upon the candidate key instructorid. Table 2.5 shows the relations, columns and primary key(s) after the third normal form.

Table 2.5: Relations and Columns after Third Normal Form

Relations	**Columns**	**Primary Key(s)**
Student	studentid, departmentid, student_first_name, student_last_name, student_primary_emailid, student_secondary_emailid, student_primary_mobile_number, student_secondary_mobile_number, student_marital_status, enrolment_status, enrolment_type, student_house_number, zip_code, faculty, program, major	studentid
Instructor	instructorid, instructor_first_name, instructor_second_name, instructor_primary_emailid, instructor_secondary_emailid, instructor_primary_mobile_number, instructor_secondary_mobile_number, instructor_marital_status, instructor_house_number, zip_code, salary	instructorid
Address	zip_code, city, province, country	zip_code
Course	courseid, course_name, credit_hours	courseid
Prerequisite	courseid, prerequisiteid	courseid, prerequisiteid
Department	departmentid, department_name, department_head, building	departmentid
Student Enrolment	studentid, courseid, semester, grade, term, year	studentid, courseid
Instructor Enrolment	instructorid, courseid, term, year	instructorid, courseid, term, year

2.4.4 BCNF and Higher Normal Forms

A relation schema is in Boyce-Codd Normal form if the relation schema is in the first normal form and no attribute is transitively dependent upon any candidate key in the relation schema [2]. For all the relations that are in third normal form, there are no attributes that are transitively dependent upon any candidate key and thus the relations are in BCNF [2]. A relation schema is in fourth normal form if for every multi-valued dependency (MVD) in the relation schema, either the MVD is trivial or the left-hand side of the MVD (LHS) contains the candidate key for R. For any given MVD in the relations, the MVD will either be trivial or the LHS of the MVD contains the candidate key and therefore the relations are also in fourth normal form.

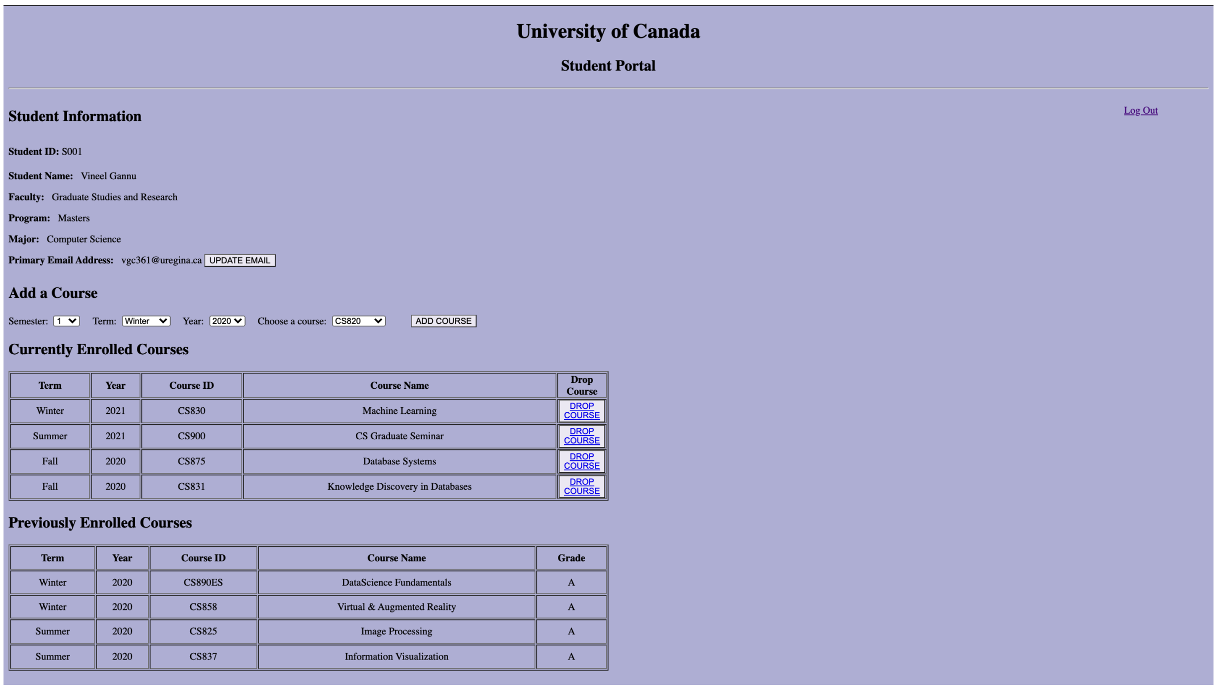
2.5 Physical Database design

PostgreSQL, which is a relational database management system software, is used to create the database and the relations. Indexing can be build on the relations if the number of records in the relation are huge.

2.6 Web Application

A web application is created where users can interact with the data. Django web framework along with HTML is used to create the web application. Two types of users use the web application - Students and Instructors. Students and Instructors have different portals through which they can access the data. The three operations that the users can perform are insert, update, and delete.

2.6.1 Student Portal



University of Canada

Student Portal

Student Information

Log Out

Student ID: S001

Student Name: Vineel Gannu

Faculty: Graduate Studies and Research

Program: Masters

Major: Computer Science

Primary Email Address: vgc361@uregina.ca UPDATE EMAIL

Add a Course

Semester: 1 Term: Winter Year: 2020 Choose a course: CS820 ADD COURSE

Currently Enrolled Courses

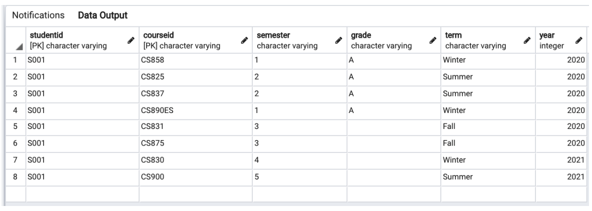
Term	Year	Course ID	Course Name	Drop Course
Winter	2021	CS830	Machine Learning	DROP COURSE
Summer	2021	CS900	CS Graduate Seminar	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE
Fall	2020	CS831	Knowledge Discovery in Databases	DROP COURSE

Previously Enrolled Courses

Term	Year	Course ID	Course Name	Grade
Winter	2020	CS890ES	DataScience Fundamentals	A
Winter	2020	CS858	Virtual & Augmented Reality	A
Summer	2020	CS825	Image Processing	A
Summer	2020	CS837	Information Visualization	A

Figure 2.5: Student Portal

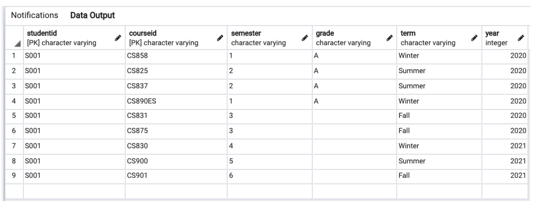
When a student logs into the portal, they can see their basic details and the currently and previously enrolled courses. The student can add/enroll in a course. They can choose the term, year, semester, and courseid, and on click of the Add Course button, the record is added to the Student Enrolment table. Figure 2.5 shows the student portal and figures 2.6 and 2.7 shows the student enrolment table and the student portal (currently enrolled courses section) before and after enrolling in a course.



Notifications **Data Output**

	studentid [PK] character varying	courseid [PK] character varying	semester character varying	grade character varying	term character varying	year integer
1	S001	CS858	1	A	Winter	2020
2	S001	CS825	2	A	Summer	2020
3	S001	CS837	2	A	Summer	2020
4	S001	CS890ES	1	A	Winter	2020
5	S001	CS831	3		Fall	2020
6	S001	CS875	3		Fall	2020
7	S001	CS830	4		Winter	2021
8	S001	CS900	5		Summer	2021

(a) Before Adding a Course

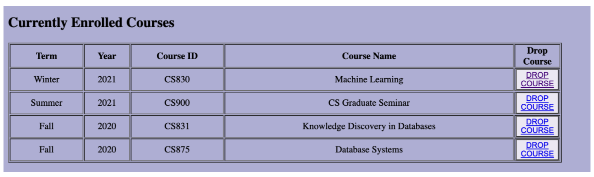


Notifications **Data Output**

	studentid [PK] character varying	courseid [PK] character varying	semester character varying	grade character varying	term character varying	year integer
1	S001	CS858	1	A	Winter	2020
2	S001	CS825	2	A	Summer	2020
3	S001	CS837	2	A	Summer	2020
4	S001	CS890ES	1	A	Winter	2020
5	S001	CS831	3		Fall	2020
6	S001	CS875	3		Fall	2020
7	S001	CS830	4		Winter	2021
8	S001	CS900	5		Summer	2021
9	S001	CS901	6		Fall	2021

(b) After Adding a Course

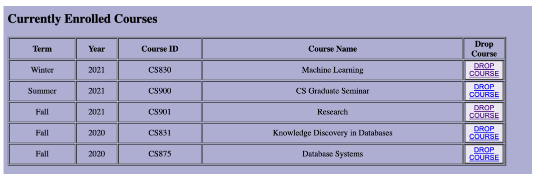
Figure 2.6: Student Enrolment Table Before and After Adding a Course



Currently Enrolled Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2021	CS830	Machine Learning	DROP COURSE
Summer	2021	CS900	CS Graduate Seminar	DROP COURSE
Fall	2020	CS831	Knowledge Discovery in Databases	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(a) Before Adding a Course



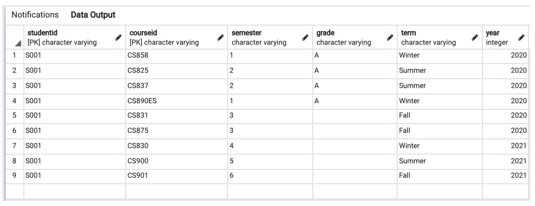
Currently Enrolled Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2021	CS830	Machine Learning	DROP COURSE
Summer	2021	CS900	CS Graduate Seminar	DROP COURSE
Fall	2021	CS901	Research	DROP COURSE
Fall	2020	CS831	Knowledge Discovery in Databases	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(b) After Adding a Course

Figure 2.7: Student Portal Before and After Adding a Course

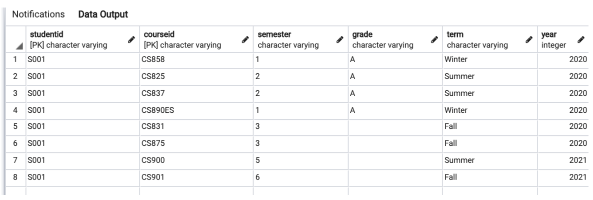
Under the 'currently enrolled courses' section in the student portal, the students can see the courseid, course_name, term, and year of the currently enrolled courses. The student can also drop any of the currently enrolled courses. When the student clicks on the Drop Course button, the corresponding record from the Student Enrolment relation gets deleted. Figures 2.8 and 2.9 shows the student enrolment table and the student portal before and after deleting a course.



Notifications Data Output

	studentid [PK] character varying	courseid [PK] character varying	semester character varying	grade character varying	term character varying	year integer
1	S001	CS858	1	A	Winter	2020
2	S001	CS825	2	A	Summer	2020
3	S001	CS837	2	A	Summer	2020
4	S001	CS890ES	1	A	Winter	2020
5	S001	CS831	3		Fall	2020
6	S001	CS875	3		Fall	2020
7	S001	CS830	4		Winter	2021
8	S001	CS900	5		Summer	2021
9	S001	CS901	6		Fall	2021

(a) Before Deleting a Course

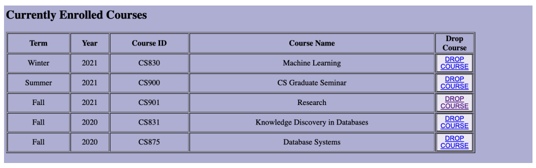


Notifications Data Output

	studentid [PK] character varying	courseid [PK] character varying	semester character varying	grade character varying	term character varying	year integer
1	S001	CS858	1	A	Winter	2020
2	S001	CS825	2	A	Summer	2020
3	S001	CS837	2	A	Summer	2020
4	S001	CS890ES	1	A	Winter	2020
5	S001	CS831	3		Fall	2020
6	S001	CS875	3		Fall	2020
7	S001	CS900	5		Summer	2021
8	S001	CS901	6		Fall	2021

(b) After Deleting a Course

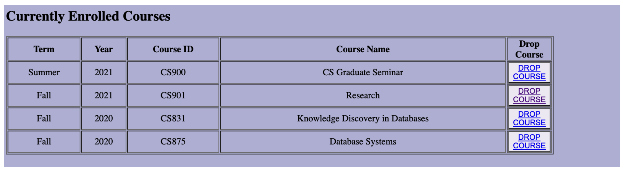
Figure 2.8: Student Enrolment Table Before and After Deleting a Course



Currently Enrolled Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2021	CS830	Machine Learning	DROP COURSE
Summer	2021	CS900	CS Graduate Seminar	DROP COURSE
Fall	2021	CS901	Research	DROP COURSE
Fall	2020	CS831	Knowledge Discovery in Databases	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(a) Before Deleting a Course



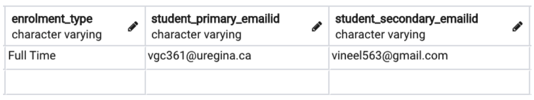
Currently Enrolled Courses

Term	Year	Course ID	Course Name	Drop Course
Summer	2021	CS900	CS Graduate Seminar	DROP COURSE
Fall	2021	CS901	Research	DROP COURSE
Fall	2020	CS831	Knowledge Discovery in Databases	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(b) After Deleting a Course

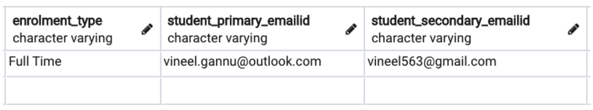
Figure 2.9: Student Portal Before and After Deleting a Course

Under the previously enrolled courses section in the student portal, the students can see the courseid, course_name, term, and year of the previously enrolled courses along with the grade. The student can update the primary email address. Figures 2.10 and 2.11 shows the student enrolment table and the student portal before and after email update.



enrolment_type character varying	student_primary_emailid character varying	student_secondary_emailid character varying
Full Time	vgc361@uregina.ca	vineel563@gmail.com

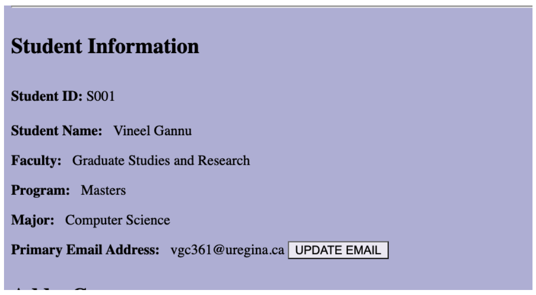
(a) Before Email Update



enrolment_type character varying	student_primary_emailid character varying	student_secondary_emailid character varying
Full Time	vineel.gannu@outlook.com	vineel563@gmail.com

(b) After Email Update

Figure 2.10: Student Table Before and After Email Update



Student Information

Student ID: S001

Student Name: Vineel Gannu

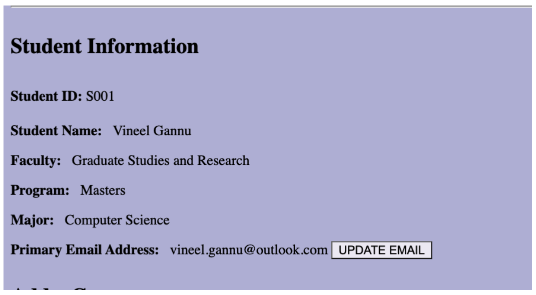
Faculty: Graduate Studies and Research

Program: Masters

Major: Computer Science

Primary Email Address: vgc361@uregina.ca

(a) Before Email Update



Student Information

Student ID: S001

Student Name: Vineel Gannu

Faculty: Graduate Studies and Research

Program: Masters

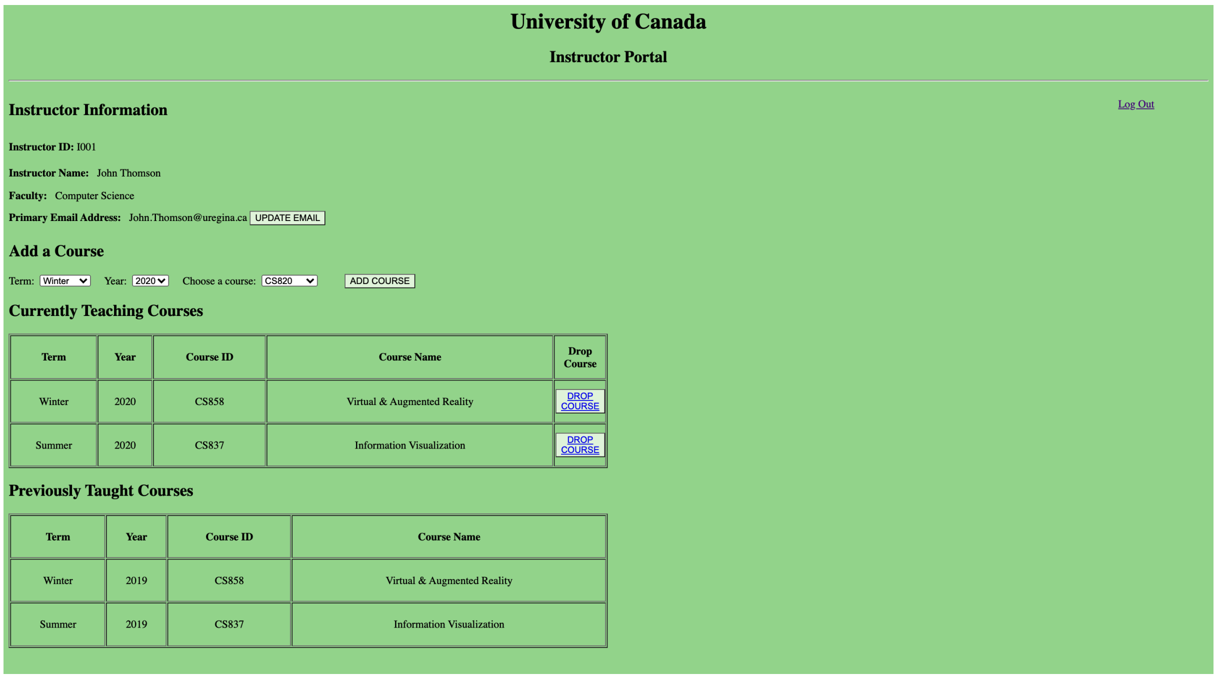
Major: Computer Science

Primary Email Address: vineel.gannu@outlook.com

(b) After Email Update

Figure 2.11: Student Portal Before and After Email Update

2.6.2 Instructor Portal



University of Canada

Instructor Portal

Instructor Information

Instructor ID: I001

Instructor Name: John Thomson

Faculty: Computer Science

Primary Email Address: John.Thomson@uregina.ca

Add a Course

Term: Winter Year: 2020 Choose a course: CS820

Currently Teaching Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2020	CS858	Virtual & Augmented Reality	DROP COURSE
Summer	2020	CS837	Information Visualization	DROP COURSE

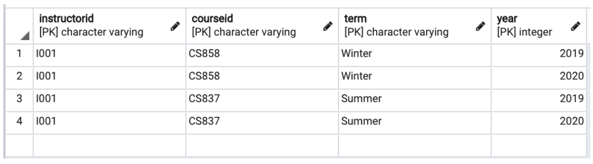
Previously Taught Courses

Term	Year	Course ID	Course Name
Winter	2019	CS858	Virtual & Augmented Reality
Summer	2019	CS837	Information Visualization

Figure 2.12: Instructor Portal

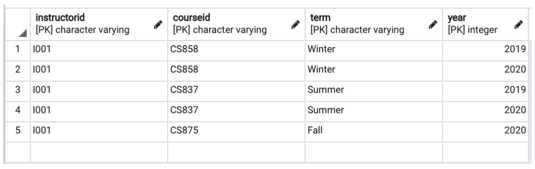
When an instructor logs into the portal, they can see their basic details and the currently and previously taught courses. The instructor can add/enroll in a course. They can choose the term, year, and courseid and on click of the Add Course button, the record is added to the Instructor Enrolment table. Figure 2.12 shows the instructor portal and figures 2.13 and 2.14

shows the instructor enrolment table and the instructor portal before and after enrolling in a course.



	instructorid [PK] character varying	courseid [PK] character varying	term [PK] character varying	year [PK] integer
1	I001	CS858	Winter	2019
2	I001	CS858	Winter	2020
3	I001	CS837	Summer	2019
4	I001	CS837	Summer	2020

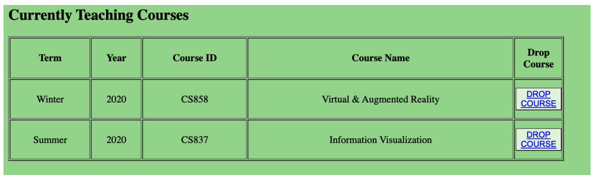
(a) Before Adding a Course



	instructorid [PK] character varying	courseid [PK] character varying	term [PK] character varying	year [PK] integer
1	I001	CS858	Winter	2019
2	I001	CS858	Winter	2020
3	I001	CS837	Summer	2019
4	I001	CS837	Summer	2020
5	I001	CS875	Fall	2020

(b) After Adding a Course

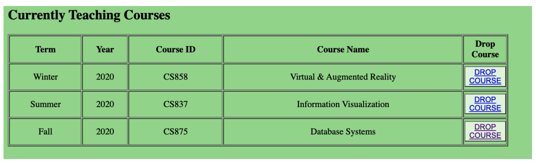
Figure 2.13: Instructor Enrolment Table Before and After Adding a Course



Currently Teaching Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2020	CS858	Virtual & Augmented Reality	DROP COURSE
Summer	2020	CS837	Information Visualization	DROP COURSE

(a) Before Adding a Course



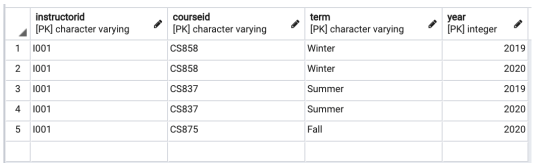
Currently Teaching Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2020	CS858	Virtual & Augmented Reality	DROP COURSE
Summer	2020	CS837	Information Visualization	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(b) After Adding a Course

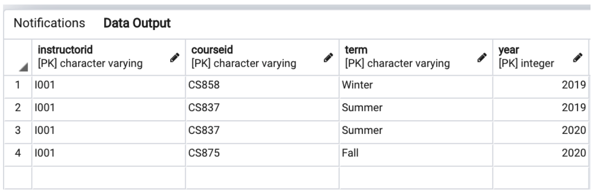
Figure 2.14: Instructor Portal Before and After Adding a Course

Under the 'currently teaching courses' section in the instructor portal, the instructors can see the courseid, course_name, term, and year of the current teaching courses. The instructor can also drop any of the currently teaching courses. When the instructor clicks on the Drop Course button, the corresponding record from the Instructor Enrolment relation gets deleted. Figures 2.15 and 2.16 shows the instructor enrolment table and the instructor portal before and after deleting a course.



	instructorid [PK] character varying	courseid [PK] character varying	term [PK] character varying	year [PK] integer
1	I001	CS858	Winter	2019
2	I001	CS858	Winter	2020
3	I001	CS837	Summer	2019
4	I001	CS837	Summer	2020
5	I001	CS875	Fall	2020

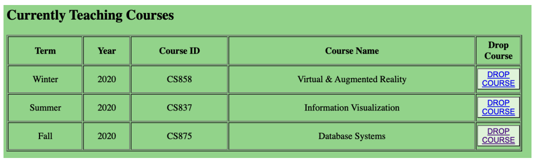
(a) Before Deleting a Course



	instructorid [PK] character varying	courseid [PK] character varying	term [PK] character varying	year [PK] integer
1	I001	CS858	Winter	2019
2	I001	CS837	Summer	2019
3	I001	CS837	Summer	2020
4	I001	CS875	Fall	2020

(b) After Deleting a Course

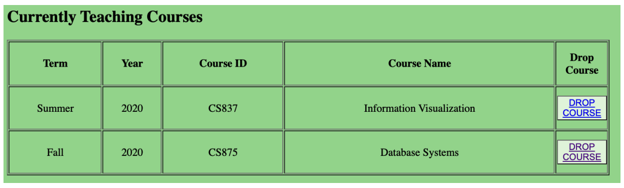
Figure 2.15: Instructor Enrolment Table Before and After Deleting a Course



Currently Teaching Courses

Term	Year	Course ID	Course Name	Drop Course
Winter	2020	CS858	Virtual & Augmented Reality	DROP COURSE
Summer	2020	CS837	Information Visualization	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(a) Before Deleting a Course



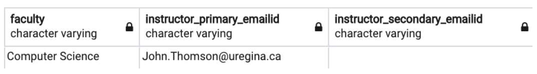
Currently Teaching Courses

Term	Year	Course ID	Course Name	Drop Course
Summer	2020	CS837	Information Visualization	DROP COURSE
Fall	2020	CS875	Database Systems	DROP COURSE

(b) After Deleting a Course

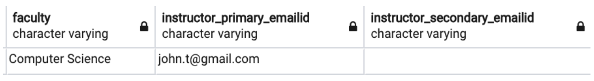
Figure 2.16: Instructor Portal Before and After Deleting a Course

Under the previously taught courses section in the instructor portal, the instructors can see the courseid, course_name, term, and year of the previously taught courses. The instructor can update the primary email address. Figures 2.17 and 2.18 shows the instructor enrolment table and the instructor portal before and after email update.



faculty character varying	instructor_primary_emailid character varying	instructor_secondary_emailid character varying
Computer Science	John.Thomson@uregina.ca	

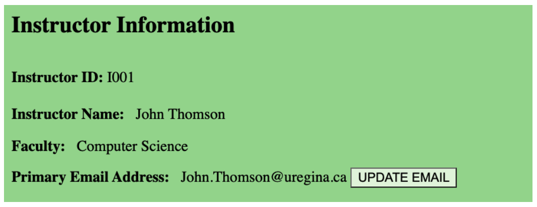
(a) Before Email Update



faculty character varying	instructor_primary_emailid character varying	instructor_secondary_emailid character varying
Computer Science	john.t@gmail.com	

(b) After Email Update

Figure 2.17: Instructor Enrolment Table Before and After Email Update



Instructor Information

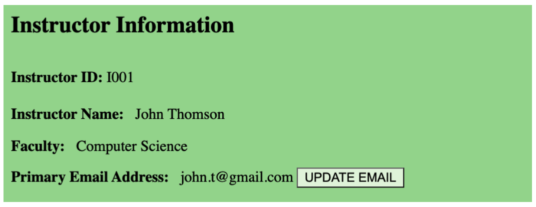
Instructor ID: I001

Instructor Name: John Thomson

Faculty: Computer Science

Primary Email Address: John.Thomson@uregina.ca

(a) Before Email Update



Instructor Information

Instructor ID: I001

Instructor Name: John Thomson

Faculty: Computer Science

Primary Email Address: john.t@gmail.com

(b) After Email Update

Figure 2.18: Instructor Portal Before and After Email Update

3. Conclusion

The data redundancy and update anomaly issues faced by the University of Canada in storing the university-related data is solved by designing a database and storing the data in the database. Additionally, the users - Student and the Instructor can interact with the database through a web application where different users have access to different portals. Admin, on the other hand, has complete access to the database.

References

[1] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, Addison-Wesley, 2004.

[2] C. Date, *An Introduction to Database Systems*, Pearson, 2005. [Cited on pages 6, 7, 9, and 11]

[3] https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5

[4] https://www.djangoproject.com/