**Project team #9 - School Course Schedule**

**Team Members: Cameron Behdani**

**Narahari Battala**

**Ryan Scolaro**

## ***PROJECT PROPOSAL***

**Content, Scope and Objectives**

Our project plan is to use a database to recreate a UNCC-esque college course catalogue, where administrators can log in and create or delete courses. Students can then login and register for those courses, as well as viewing and deleting the courses they have registered for on their course schedule. The administrators should then be able to view the students that have registered for each course.

## ***PROJECT ENVIRONMENT***

Our project environment will mimic the environment we use for classroom activities. We will be using a MySQL server on Cloud9 for our database. Our DBMS will be phpMyAdmin. We do not plan on implementing a UI application for our database, but instead intend to rely on phpMyAdmin to display our data and functionality.

## ***HIGH LEVEL REQUIREMENTS***

### **Initial user roles**

|  |  |
| --- | --- |
| **User Role** | **Description** |
| Admin | Responsible for adding, updating, and deleting courses and departments. Can also view and edit teacher and student information. |
| Teacher | Able to see which and how many students have enrolled in their class. |
| Student | Able to register or drop courses. Also able to see which courses they have already enrolled in. |

### **Initial user story descriptions**

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As an admin, I want to add a new department so that new courses can be added in that department. |
| US2 | As an admin, I want to add a new course so that students can enroll into the newly added course. |
| US3 | As a student I want to enroll in classes so that I can attend them. |
| US4 | As a student I want to drop the courses which I enrolled in so that I can enroll for another class. |
| US5 | As a student I want to view my enrolled courses so that I can design my schedule. |
| US6 | As an admin, I want to edit a student’s information so they can be added to/removed from a course. |
| US7 | As an admin, I want to update a department so that a department can properly reflect the courses it contains. |
| US8 | As an admin, I want to delete a department if it no longer exists. |
| US9 | As an admin, I want to update a course so that students can see any changes made to an academic course. |
| US10 | As an admin, I want to delete a course so that students can not enroll or attend a non-existent course. |
| US11 | As teacher, I want to view my courses so that I can see what students I am teaching. |
| US12 | As an admin, I want to edit a teacher’s information to reflect the courses they are teaching. |
| US13 | As a student, I want to search for a course so that I can find the course I wish to take. |

## ***HIGH LEVEL CONCEPTUAL DESIGN***

Entities:

Admin

Teacher

Student

Department

Course

Relationships:

Department contains course

Course has teacher

Student enrolls in course

Student drops course

Admin creates department

Admin creates course

Admin removes department

Admin removes course

Course has student

Admin edits department

Admin edits course

## ***CONCEPTUAL DESIGN***

Entity: **Admin**

Attributes:

username

password

name [composite]

first\_name

last\_name

Entity: **Department**

Attributes:

name

abbreviated\_name

description

Entity: **Course**

Attributes:

name

course\_number

credits

description

Entity: **Prerequisite**

Attributes:

course

prerequisite

Entity: **Section**

Attributes:

section\_num

section\_limit

year

Entity: **Semester**

Attributes:

season

Entity: **Building**

Attributes:

name

Entity: **Room**

Attributes:

number

capacity

Entity: **Lecture\_Type**

Attributes:

type

Entity: **Lecture\_Time**

Attributes:

start\_time

end\_time

Entity: **Lecture\_Day**

Attributes:

day

Entity: **Teacher**

Attributes:

teacher\_id

name [composite]

first\_name

last\_name

Entity: **Student**

Attributes:

student\_id

name [composite]

first\_name

last\_name

Entity: **Transcript**

Attributes:

transcript\_id

Relationship: **Department** has **Course**

Cardinality: Many to Many

Participation:

Department has total participation

Course has total participation

**Assumptions: Each Department will have at least one course and each course should be assigned to at least one department**

Relationship: **Course** has **Prerequisite**

Cardinality: Many to Many

Participation:

Course has partial participation

Prerequisite has total participation

Relationship: **Course** has **Section**

Cardinality: One to Many

Participation:

Course has partial participation

Section has total participation

Relationship: **Section** has **Semester**

Cardinality: Many to One

Participation:

Section has partial participation

Semester has total participation

Relationship: **Section** has **Room**

Cardinality: Many to One

Participation:

Section has partial participation

Room has partial participation

Relationship: **Building** has **Room**

Cardinality: One to Many

Participation:

Building has total participation

Room has total participation

Relationship: **Section** has **Lecture\_Type**

Cardinality: Many to One

Participation:

Section has total participation

Lecture\_Type has total participation

Relationship: **Section** has **Lecture\_Time**

Cardinality: Many to Many

Participation:

Section has total participation

Lecture\_Time has total participation

Relationship: **Section** has **Lecture\_Day**

Cardinality: Many to Many

Participation:

Section has partial participation

Lecture\_Day has partial participation

Relationship: **Section** has **Teacher**

Cardinality: Many to One

Participation:

Section has total participation

Teacher has partial participation

Relationship: **Section** has **Student**

Cardinality: Many to Many

Participation:

Section has partial participation

Student has partial participation

Relationship: **Student** has **Transcript**

Cardinality: One to One

Participation:

Student has total participation

Transcript has total participation

Relationship: **Transcript** has **Course**

Cardinality: Many to Many

Participation:

Transcript has total participation

Course has partial participation

***LOGICAL DESIGN***

Table: **Admin**

Columns:

username

password

first\_name

last\_name

Table: **Department**

Columns:

abbreviated\_name

name

description

Table: **Course**

Columns:

course\_id

name

course\_number

credits

description

Table: **DepartmentCourse**

Columns:

course [foreign key; references **course\_id** of **Course**]

department [foreign key; references **abbreviated\_name** of

**Department**]

Table: **Prerequisite**

Columns:

course [foreign key; references **id** of **Course**]

prerequisite [foreign key; references **id** of **Course**]

Table: **Semester**

Columns:

season

Table: **Teacher**

Columns:

teacher\_id

first\_name

last\_name

Table: **Section**

Columns:

section\_id

section\_num

section\_limit

year

teacher [foreign key; references **teacher\_id** of **Teacher**]

course [foreign key; references **course\_id** of **Course**]

semester [foreign key; references **season** of **Semester**]

room [foreign key; references **room\_id** of **Room**]

lecture\_type [foreign key; references **type** of **Lecture\_Type**]

*Justification: A generated id column is used to avoid having a*

*multi-column foreign key.*

Table: **SectionDayTime**

Columns:

section [foreign key; references **section\_id** of **Section**]

time [foreign key; references **lecture\_time\_id** of **Lecture\_Time**]

day [foreign key; references **day** of **Lecture\_Day**]

Table: **Building**

Columns:

name

Table: **Room**

Columns:

room\_id

number

capacity

building [foreign key; references **name** of **Building**]

*Justification: A generated id column is used to avoid having a multi-column foreign key.*

Table: **Lecture\_Type**

Columns:

type

Table: **Lecture\_Time**

Columns:

lecture\_time\_id

start\_time

End\_time

*Justification: A generated id column is used to avoid having a multi-column foreign key.*

Table: **Lecture\_Day**

Columns:

day

Table: **Student**

Columns:

student\_id

first\_name

last\_name

Table: **StudentSection**

Columns:

student [foreign key; references **student\_id** of **Student**]

section [foreign key; references **section\_id** of **Section**]

Table: **Transcript**

Columns:

transcript\_id

student [foreign key; references **student\_id** of **Student**]

Table: **TranscriptCourse**

Columns:

transcript [foreign key; references **transcript\_id** of **Transcript**]

course [foreign key; references **course\_id** of **Course**]

***Stored Procedures , Views and Triggers***

***Views:***

**View:** CourseInDepartment

Goal: This view contains all of the courses in every department. A user will be able to specify a department in the where clause to see only courses in a specific department. This is usually the first step when a student registers for a class.

**View:** SectionInformation

Goal: This view contains all of the information about every section. A user can use the where clause to specify sections of a specific class. This is usually the second step when a student registers for a class.

**View**: PrerequisiteRequirements

Goal: This view contains all courses that have pre-requisite course requirements and lists them with their respective pre-requisite courses

**Stored Procedures:**

**Stored procedure**: adddepartment

**Parameters**: departmentCode IN , departmentName IN, departmentDescription IN

**Goa**l: This stored procedure is for adding the Department, Admin is the only one who can add the department and details respective to that department

**Stored procedure**: addCourse

**Parameters**: courseNumbercode IN , courseName IN, courseDescription IN,credits IN

**Goa**l: This stored procedure is for adding the Course, Admin is the only one who can add the Course and details respective to that course.

**Stored procedure**: addCoursetoDepartment

**Parameters**: courseId IN , departmentCode IN

**Goa**l: This stored procedure is for adding the Course to a Department, Admin is the only one who can add the Course to a particular department, a Course can be added to any number of departments.

**Stored procedure**: studentRegistration

**Parameters**: firstName IN , lastname IN, student\_id OUT

**Goa**l: This stored procedure is for students who are willing to get their student id . once the registration is done, student will get a unique student id using which student can enroll for classes.

**Stored procedure**: teacherRegistration

**Parameters**: firstName IN , lastname IN,teacher\_id OUT

**Goa**l: This stored procedure is for Teacher Registration, only after registration, the admin can add the teacher to a section using the unique id which the teacher gets after completing the registration process.

**Stored procedure**: addSectiontoCourse

**Parameters**: sectionNumber IN , sectionLimit IN,year IN,teacher\_id IN,courseId IN,semester IN,room\_id IN,lecture\_type IN

**Goa**l: This stored procedure is for adding a section to the any of the available course , only admin can do this as he is the only one authenticated to do so. Students can enroll in these sections.

**Stored procedure**: studentEnrollsIntoSection

**Parameters**: student\_id IN ,section\_id IN

**Goa**l: This stored procedure is for Student enrollment in sections , only after registration student can enroll for classes using his unique id. , here Student can only enroll for section if the section slots are available also if the section the student is trying to enroll conflicts in both day and time with the classes they already enrolled , then they will not be allowed to enroll for the section

**Stored procedure**: timingscheck

**Parameters**: student\_id IN ,time\_id IN

**Goa**l: This stored procedure is for checking the time conflicts if any , when student tries to enroll for a new section , the timings of new sections will be checked with the timings of all the sections he already enrolled , if any conflicts occurs a conflict code will be send which prevents student from enrolling in to that new section

**Stored procedure**: check\_Sectionlimit

**Parameters**: seclimit IN ,room IN, limitresult OUT

**Goa**l: This stored procedure is for checking whether the student limit of a particular section is less than or equal to room capacity , if student limit is more than room capacity then error message will be thrown and section will not get added to the course.

**Stored procedure**: sectionSchedule

**Parameters**: sectionId IN ,daySlot IN, timeSlot OUT

**Goa**l: This stored procedure is for checking the time slot and the day that is being allocated to a section is free or not , if the slots are free then the section will be alloted this slots else error message will be thrown asking the admin to choose different slots for the section.

**Stored procedure**: teacherSection

**Parameters**: teacher\_id IN ,section\_id IN

**Goa**l: This stored procedure is for teacher handling a new sections , only after registration teacher can handle the classes using his unique id. if the section the Teacher is trying to handle conflicts in both day and time with the classes they are already handling , they will not be allowed to handle the section

**Stored procedure**: teachertimingscheck

**Parameters**: teacher\_id IN ,time\_id IN

**Goa**l: This stored procedure is for checking the time conflicts if any , when teacher tries to handle a new section , the timings of new sections will be checked with the timings of all the sections he already enrolled , if any conflicts occurs , then a conflict code will be send which prevents that teacher from handling that new section

***Triggers:***

**Trigger:** AFTER INSERT ON StudentSection

Goal: This Trigger is for Increasing the Student Count of particular section.Right after a student enrolls for a section the trigger gets executed and increases the student count of the enrolled section in the section table by 1. This trigger makes sure that student\_count will not exceed the student\_limit for that section , after the limit is reached if a student tries to enroll for the section , he can’t do so as the slots for the section are already filled up.

***FINAL LOGICAL DESIGN***

Table: **Admin**

Columns:

username

password

first\_name

last\_name

Highest normalization level: 4NF

Index:

Index 1: non-clustered

Columns: last\_name, first\_name

Justification: We typically get the details of the admin using last\_name,but in some cases last\_names can be same so combination of last\_name and first\_name will be a good index.Also if someone forgets their username, the best way to search their username is by using the last name and first name of the person.

Table: **Department**

Columns:

abbreviated\_name

name

description

Highest normalization level: 4NF

Index:

Index 1: non-clustered

Columns: name

Justification: In most cases everyone searches the department by its name as it is the easy and efficient way to get the information, also not everyone knows the abbreviated name of the department.

Table: **Course**

Columns:

course\_id

name

course\_number

credits

description

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: name

Justification: People generally search for a course by its name as very few people are aware of the course numbers , so using it as an index will allow for faster searching.

Table: **DepartmentCourse**

Columns:

course [foreign key; references **course\_id** of **Course**]

department [foreign key; references **department\_id** of

**Department**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: department

Justification: Department will be a good index , generally when someone wants to know all the courses in a department, searching by department gives efficient results.

Table: **Prerequisite**

Columns:

course [foreign key; references **course\_id** of **Course**]

prerequisite [foreign key; references **course\_id** of **Course**]

Highest normalization level: 4NF

Indexes:

Index 1: non clustered

Columns: course

Justification: In most cases prerequisites for a course will be searched based on course , so Course will serve as a good index.

Table: **Semester**

Columns:

season

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: season

Justification: season will be a good index as everyone is aware of only the semester names, and every one uses the semester name to get the result they want.

Table: **Teacher**

Columns:

teacher\_id

first\_name

last\_name

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: first\_name, last\_name

Justification: teacher\_id is the primary key, but most teachers will be searched using first\_name and last\_name so they make up the covering index.

Table: **Section**

Columns:

section\_id

section\_num

section\_limit

year

course [foreign key; references **course\_id** of **Course**]

semester [foreign key; references **semester\_id** of **Semester**]

room [foreign key; references **room\_id** of **Room**]

lecture\_type [foreign key; references **lecture\_type\_id** of **Lecture\_Type**]

*Justification: A generated id column is used to avoid having a*

*multi-column foreign key.*

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: course, section\_num

Justification: Usually a section will be linked to a course , searching by course and section\_num will get the accurate details for a section , so combination of course and section\_num will be a good index.

Table: **SectionDayTime**

Columns:

section [foreign key; references **section\_id** of **Section**]

time [foreign key; references **lecture\_time\_id** of **Lecture\_Time**]

day [foreign key; references **lecture\_day\_id** of **Lecture\_Day**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: section

Justification: details for a particular section will be searched based on section\_id, details will be efficient if searched using section\_id , so section\_id will be a good index.

Table: **Building**

Columns:

name

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: name

Justification: Any building will be remembered by its name , so using its name as an index will be effective and searching will be efficient.

Table: **Room**

Columns:

room\_id

number

capacity

building [foreign key; references **building\_id** of **Building**]

*Justification: A generated id column is used to avoid having a multi-column foreign key.*

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: number

Justification: Room number is the best option to find a room and its details, so using room number as an index will make searching easy.

Table: **Lecture\_Type**

Columns:

type

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: type

Justification: lecture\_type is best way to find all the lectures with lectures of the required type , so it will be good index.

Table: **Lecture\_Time**

Columns:

lecture\_time\_id

start\_time

end\_time

*Justification: A generated id column is used to avoid having a multi-column foreign key.*

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: start\_time, end\_time

Justification: Combination of start\_time and end\_time will be a good index as it is easy way to search.

Table: **Lecture\_Day**

Columns:

day

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: day

Justification: Using day names is easy to remember and search, so the day column will be a good index.

Table: **Student**

Columns:

student\_id

first\_name

last\_name

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: first\_name, last\_name

Justification: Student\_id is the primary key, but most students will be searched using first\_name and last\_name, so both of those attributes make up the index.

Table: **Transcript**

Columns:

transcript\_id

student [foreign key; references **student\_id** of **Student**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: student

Justification: We will be searching through transcripts using student as each student has 1 transcript and we will be searching the table in regards to the particular student we are addressing, so student serves as good index.

Table: **TranscriptCourse**

Columns:

transcript [foreign key; references **transcript\_id** of **Transcript**]

course [foreign key; references **course\_id** of **Course**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: course

Justification: The course attribute in this table is what we will be primarily use to get the transcript for that particular course, so course is the index.

Table: **StudentSection**

Columns:

student [foreign key; references **student\_id** of **Student**]

section [foreign key; references **section\_id** of **Section**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: student

Justification: We will be searching through this table primarily using the student key to get the details of the sections he enrolled so far, so having student as an index will be efficient.

Table: **TeacherSection**

Columns:

teacher [foreign key; references **teacher\_id** of **Teacher**]

section [foreign key; references **section\_id** of **Section**]

Highest normalization level: 4NF

Indexes:

Index 1: non-clustered

Columns: teacher

Justification: We will be searching through this table primarily using the teacher key to get the details of the sections the teacher is handling, so teacher should be the index.