CSC 540 - Project 1 - Spring 2017

Course Registration System

Course: Database Management Concepts and Systems (Dr. Kemafor Ogan)

Team Members

Member Name	Unity ID
Anuja Supekar	*asupeka*
Aditya Shirode	*avshirod*
Gaurav Aradhye	*garadhye*
Kahan Prabhu	*kprabhu*

How to run this project?

1. The **sql** files (table creation, population) for the project are stored in this folder. The procedures are in this folder.
2. To initially create the project (table creation and population), run the **'run_script.sql'** file (which should be outside both of the '/sql' and '/procedures' folders. If it is not, copy the *run_script.sql* file from */sql* folder and put it in the parent directory of both '/sql' and '/procedures' folders.) This script (as you can see in the script) calls the required *sql* files for database schema creation.
3. After you successfully run the **run_script.sql**, the database will be populated with initial values. And is ready to be worked/ tested upon.

This is the recommended method to easily setup the schema.
Although, as requested in the project document, *two* separate *sql* files have been provided in the */sql* folder - **database_creation.sql** for Schema creation (includes table creation, constraints, procedures, triggers) and **database_population.sql** for Table Population.

Project Description

The goal of the project is to design a relational database application for supporting course registration at a university (similar to *MyPack* for *NC State*).
The application manages different kinds of data about students and courses, subject to a variety of application requirements that are given in use cases. More details can be found here Problem Statement.

Related Documents

Problem Statement
Worksheet.MD
ER Diagram
ER Diagram Details
Relational Model
System Constraints
SQL Files

Final Application Executable
README.txt

ER Diagram Explained

CSC 540 - Project 1 - Spring 2017

Entities

1. **UnityUsers**: Stores users of the system. Is classified into Employees (further classified as Faculty and Admins), and Students.
2. **Course**: Stores details about a courses that have been offered. (Not to be confused with a particular CourseOffering).
3. **Department**: Stores the departmentID (Codes such as 'CSC', 'ECE',etc.) and deptName.
4. **Semester**: Stores details for a particular semester (Seasons can be Fall, Spring, Summer), Add-Drop Dates, Semester Start-End Dates.
5. **Credit Requirements**: Stores the Min and Max Credits required corresponding to a combination between Participation Level (Undergraduate/ Graduate) and Residency Level (In-state/ Out-state/ International).
6. **Special Permissions**: Stores details about special permission codes ('PREREQ', 'SPPERM').

We also have two weak entities - **Billing** and **Permission Mapping**. Billing stores the details about students and their bills as they are generated (and updated). Permission mapping stores the special permissions required for a particular course offering (such as prerequisites, GPA requirements).

Relationships

(1) All the users of the system are classified as *UnityUsers*. This has a **'ISA'** relationship with *Employee* and *Students*. Employees have a further *ISA* classification into *Faculty* and *Admin*. A full participation for all of these is necessary, meaning a UnityUser has to be classified into either of the three types - Student, Faculty, or Admin Staff.

(2) *Courses* are **offered under** *Departments*. (One-to-One mapping) *Courses* are **offered during** *Semesters* which results in a *CourseOffering*. (We have represented CourseOffering as a separate Relationship Table, to avoid a complex quaternary relationship between tables.)

(3) *Students* **have** a Bill in *Billing* table, based on *Course Offerings* for that student **in** *Transcript* and *Waitlist*. (One-to-Many mappings)

(4) *Faculty* is a **member of** a particular *Department*. He/ she also **facilitates** a *CourseOffering*. (One-to-one mapping for both)

(5) *CourseOfferings* have *Courses* as **Prerequisites**. (One-to-Many mapping) They also **require** *special permissions* mapped in *Permission Mapping*. (One-to-Many mapping) *Students* can **request** these permissions for a particular Course Offering. (One-to-One mapping)

(6) Based on *Credit Requirement*, there are **Billing Rates** defined per *Semester*. (One-to-Many mapping)

Relational Model Explained

Table: Unity Users

Functional dependencies:

- unityID fname
- unityID lname
- unityID password
- unityID email
- unityID gender
- unityID dateOfBirth
- unityID address
- unityID phone
- unityID userType

Table: Students

Constraints:

-Uniqueness constraints: The studentid should be unique for each entry. - Foreign key constraints: The unityid in table UnityUsers , the deptID in the Departments table and creditReqID in the CreditRequirements are the foreign keys. - Data entry constraints: The residency level for all students must be among 1,2,3 where 1=in state, 2= out-of-state and 3=international and the participation level among 1, 2 where 1=undergraduate and 2=graduate.

Functional dependencies:

- unityID deptID
- unityID overallGPA
- unityID residencyLevel
- unityID participationLevel
- unityID yearEnrolled

Table: Employee

Constraints:

- Uniqueness constraints: The ssn for all employees should be unique.
- Foreign key constraints: The unityID from the UnityUsers table is the foreign key.

Functional dependencies:

- unityID SSN

Table: Faculty

Constraints:

- Uniqueness constraints: The facultyID for each faculty should be unique.
- Foreign key constraints: The unityid from the Employee table and deptID from the Department table are the foreign keys.

Functional dependencies:

- unityID deptID
- unityID Field
- SSN deptID
- SSN Field

Table: Admin

Constraints:

- Uniqueness constraints: The adminID for all the entries should be unique.
- Foreign key constraints: The unityID from the Employee table is the foreign key.

Table: Department

- deptID deptName

Table: Course

Constraints:

- Foreign key contraints: deptID from the Department table is the foreign key.
- Data entry constraints: the minimum credits for all courses should be atleast 1 and the maximum credits for all the courses should be atleast equal to the respective minimum credits for the courses.

Functional Dependencies:

- courseID sectionID
- courseID title
- courseID maxCreditts
- courseID minCredits
- courseID participationLevel

Table: CourseOffering

Constraints:

- Uniqueness constraints: The courseofferingID for all the courses should be unique which is a combination of courseID and sectionID.
- Foreign key constraints: The combination of (courseID, deptID) from the Course table and the semesterID from the Semester are the foreign keys.

- Data entry Constraints: The classDays must be between 1 and 9. Where 1 ->Monday, 2->Tuesday, 3->Wednesday, 4->Thursday, 5->Friday, 6->Mon,wed, 7->Tue,Thur , 8->Mon, Wed, Fri, 9->any other combination. Also, The seats available at any point for a course offering should be at most the total number of seats offered for that course offering.

Functional dependencies:

- courseID, sectionID, semesterID classDays
- courseID, sectionID, semesterID startTime
- courseID, sectionID, semesterID endTime
- courseID, sectionID, semesterID location
- courseID, sectionID, semesterID totalSeatsOffered
- courseID, sectionID, semesterID seatsAvailable
- courseID, sectionID, semesterID waitlistLimit

Table: Semester

Constraints:

- Uniqueness Constraint: The semesterID should be unique for all the semesters.
- Data entry constraints: The drop date for all courses should be on or after the add date for the course and the semester end date should be on or after the semester start date.

Functional Dependencies:

- semesterID season
- semesterID year
- semesterID addDate
- semesterID dropDate
- semesterID semStartDate
- semesterID semEndDate

Table: CreditRequirements

Constraints:

- Uniqueness constraint: The creditReqID for all the entries should be unique.
- Data Entry Constraints: The minimum credit required should be equal to or more than 0 and the maximum credits required should be atleast equal to the minimum credits required. The participation level must be between 1 and 2 where 1=undergraduate and 2=graduate. The residency level must be between 1, 2 and 3 where 1=in state, 2=out of state and 3=international.

Functional dependencies:

We have decided to use participationLevel and residencyLevel as composite primary key as against using the creditReqID

- participationLevel, residencyLevel maxCreditsReq

- participationLevel, residencyLevel → minCreditsReq

Table: Billing

Constraints:

- Uniqueness constraints: The unityID from the UnityUsers table is the foreign key.

Functional Dependencies:

- unityID, billID → billAmount
- unityID, billID → dueDate
- unityID, billID → outstandingAmount

Table: BillingRate

Constraints:

- Foreign key constraints: the semesterID from Semester table and the creditReqID from CreditRequirements table are the foreign keys.

Functional dependencies:

- participationLevel, residencyLevel, semesterID → $/CreditHour

Table: PermissionMapping

Constraints:

- Foreign key constraints: The specialPermissionID from SpecialPermissions table and courseOfferingID from CourseOffering table are the foreign keys.

Table: SpecialPermissions

- specialPermissionID → specialPermissionCode

Table: SpecialPermissionRequests

Constraints:

- Foreign key constraints: The combination of (specialPermissionID, courseOfferingID) from PermissionMapping table , unityID from Students table and unityID from the Admin table are the foreign keys.

Table: Prerequisites

Constraints:

- Foreign key constraints: The courseOfferingID from CourseOffering table and the combination of (courseID, deptID) from the Course table are the foreign keys.
- Data entry constraints: The GPA for all courses for all students must be between 0.00 and 4.334.

Functional dependencies:

- currentCourseID corresponds to courseID in CourseOffering table.
- prereqCourseID corresponds to courseID in Course table.
- currentCourseID, sectionID, semesterID, prereqCourseID grade

Table: Transcript

Constraints:

- Foreign key constraints: The unityID from Students table and courseOfferingID from courseOffering table are the foreign keys.
- Data entry constraints: The course grades should be between A+, A, A-, B+, B. B-, C+, C, C-, F, S, U, AU, INC, PROG where F=Fail, S=Satisfactory, U=Unsatisfactory, AU=Audit, INC=Incomplete, PROG=in Progress. The credit hours taken for all students for all courses should be atleast 1.

Functional Dependencies:

- unityID, courseID, sectionID, semesterID grade
- unityID, courseID, sectionID, semesterID numberOfCredits

Table: Facilitates

Constraints:

- Foreign key constraints: The courseOfferingID from courseOffering table and unityID from the Faculty table are the foreign keys.

Table: Waitlist

Constraints:

- Foreign key constraints: The unityID from Students table and courseOfferingID from CourseOffering table are the foreign keys.

Functional dependencies:

- unityID, courseID, sectionID, semesterID waitlistPosition

Normal Form discussion:

- The application is in the 2 NF normal form. 2NF is easy to implement and less complex as compared to 3NF.
- 3NF increases the number of tables and thus the complexity.
- Initial design was devoid of transitive dependencies. But later there were many modifications as we built the database and there might have been transitive dependencies introduced. Hence it is not completely in 3NF.

<u>Constraints</u> Explained

CSC 540 - Project 1 - Spring 2017

Constraints in ER Model

Key Constraints (Primary/Foreign Key)

1. Hierarchical inheritance between *Faculty*, *Admin*, and *Students* to *UnityUsers*. **UnityID** is the primary key for *UnityUsers*, and it travels down as a foreign key to above three tables.
2. *Students* and *Faculty* are associated with *Department* based on foreign key **DeptID**.
3. Particular *CreditRequirements* are mapped to *Students* by foreign key **CreditReqID**.
4. *Courses* have a foreign key from *Department*, **DeptID**, which in combination with **CourseID**, acts as the primary key for the table.
5. *Semesters* are identified by their primary key **SemesterID**, which is auto-incremented based on combinations of (**Season-Year**).
6. A unique *CourseOffering* is identified based on *CourseOfferingID*, which is auto-incremented for unique combinations of **CourseID**, **DeptID**, **SectionID**, and **SemesterID**.
7. Table of *Billing* has a foreign key **UnityID** from *Students*, and possesses a weak relationship.
8. *Transcript* and *Waitlist* both have foreign keys **-** **UnityID** from *Students* and **CourseOfferingID** from *CourseOffering**. Their combination acts as a primary key for these tables.
9. *PermissionMapping* handles unique combinations of *SpecialPermissions* and *CourseOfferings* based on foreign keys (**SpecialPermissionID**, **CourseOfferingID**).
10. Facilitates has a pairing of **UnityID** from *Faculty* and **CourseOfferingID** from *CourseOffering*.
11. As *BillingRate* is identified per *Credit Requirement* per *Semester*, it offers a foreign key matching of **CreditReqID** and **SemesterID**.
12. *CourseOfferings* have pre-requisites of *Courses* mentioned in *Prerequisites* relationship table by **CourseOfferingID - (CourseID**, **DeptID**) combination.
13. *Students* can submit *SpecialPermission* requests, which are registered by keys **UnityID** (*Students*), **CourseOfferingID** (*CourseOffering*). And are approved by **UnityID** (*Admin*).

Uniqueness Constraints

- *Students*, *Admins*, and *Faculties* can be uniquely identified based on their corresponding member **ID card numbers**, apart from *UnityID*s.
- *Employees* (*Admin*, *Faculty*) can also be uniquely identified by **ssn**.
- *CourseOfferings* are uniquely listed by **CourseOfferingID**, *Semesters* by **SemesterID**, and *CreditRequirements* by **CreditReqID**.

Data Entry Constraints

There are certain fields in database which encode information in numbers. They have been encoded by Data Entry Constraints as follows - * **residencyLevel** for *Students* from *CreditRequirements* = (1-Undergraduate, 2-Graduate). * **participationLevel** for *Students* from *CreditRequirements* = (1-InState, 2-OutOfState, 3-International). * **Grades** have to be between the possible grade range of [A+, C-]. * **ClassDays** are numbers between (0-9). 0:DE, 1-5:M-F, 6:MW, 7:TTH, 8:MWF, 9:Others

Then there are certain inherent and basic data entry constraints, such as - * *GPA* has to be between

(0, 4.333). * *LastDropDay* can't be before *LastEnrollmentDay* for a semester. Same goes for *Start* and *End* days. * Students can't take a course for less than 1 credit hours (transcript). * For any given CourseOffering, number of available seats cannot be greater than totalSeats offered for that offering.

All of the above constraints have been handled in SQL code in database. But the mapping for encoded elements has been handled at the application level. Database just handles the numbers. The string representations for those (except Grades) are done in Java code.

Constraints in Application Code