University Academic Database System (UADS)

Created By: Josh Pondrom, Harinya Potluri, Derek Hymer

Problem Statement:

A database is a collection of information that is organized so that it can be easily accessed, managed, and updated. We are proposing a database to help manage a University system. A University has so many aspects that goes toward its success; and a functional database that helps students manage their student accounts is very important. Having a database to keep track of all the students attending their University and all their individual account information is highly essential and important. The database will help students sign up for classes needed for their degree major. It will help them build a well-managed schedule to keep track of their classes and activities. The database will help students connect with their Professors and readily give the students information they may need for their classes. The database we are proposing is UADS (University Academic Database System). This database is essential for this system because it will help students sign up for classes, look at their class schedule, access their transcripts, and look up their instructors' details information.

Conceptual Design:

Problem Description:

Every student has a GPA, year, major, and a unique ID.Students and instructors each have a name, address, and a unique ID.

Every instructor belongs to a department. The department has different programs and keeps track of the number students in their department. Each department is located in a building with an address. There could be many departments in one building. Buildings have names and abbreviations. Each building keeps track of the number of rooms available. The rooms are defined by room number and the type of room. Many classes can be taught in one room.

Students must take courses required for their major. The university keeps track of all the courses a student takes in the university and all the courses a student transfers into the university. Each course has a unique course ID, the amount of credit hours it is worth, and what major is offering the course. The class keeps track of the grades all students receive. A course can have many classes being offered. A class can have a class ID, a time it is being offered, and a DayCode which represents the days the class meets. Each class is taught by instructors that belong to different departments. One instructor can teach many classes.

Assumptions:

- Daycode is a 5 digit code that represents which days a class meets. Ex. 01010 = Tu,Th
- Building can have multiple Departments
- Students have to take classes and classes need students
- A class's room will never change
- A person has to be a student or Instructor
- A course has to have classes
- Students need to take classes ex. Degree classes
- A person can only have one emergency contact.
- Two people cannot have the same emergency contact
- Classes can only begin and end on the hour

Functional Requirements:

- Class Sign-up
    - Students can sign up for a class by searching through the available classes for a given course. Will check for conflicts in schedule
    - Entities Involved: Student, Course, Class
- Generate Student Schedule
    - This Contains: Instructors, Classes, Class times. For Monday - Friday
    - Entities Involved: Student, Class, Room, Instructor
- Generate Instructor Schedule
    - This Contains:Classes, Class times. For Monday - Friday
    - Entities Involved: Instructor, Class, Room
- Lookup Degree progress
    - Gives a list of taken classes along with required courses that haven't been taken
    - Entities Involved: Student, Class, Course
- Get Person Info
    - By entering an ID the Database show the info on a person and emergency contact
    - Entities Involved: Person/Instructor/Student, Emergency Contact
- Search For a Class
    - This has different search fields for attributes like Class name, Class ID, Class time, Instructor ID. It will show Class ID, Class name, Class time, Students Enrolled, Instructor Name.
    - Entities Involved: Class, Course, Room
- Add Person
    - Adds a person to the database
    - Entities Involved: Person/Student/Instructor
- Remove Person
    - Deletes a person from the database

    - Entities Involved: Person/Student/Instructor
- Add Course
    - Adds a Course to the Database
    - Entities Involved: Course
- Remove Course
    - Removes a Course from the Database
    - Entities Involved: Course
- Add Class
    - Adds a class to the database
    - Entities Involved: Class, Course, Room
- Remove Class
    - Removes a class from the database
    - Entities Involved: Class, Course, Room
- Add Building
    - Adds a Building to the database
    - Entities Involved: Building, Room

RELATION SUMMARIES

DEPARTMENT SUMMARY

Attribute	Type	Description
Program	Variable String	Name of the Department

LOCATED SUMMARY

Attribute	Type	Description
Program	Variable String	Program of a Department
Address	Variable String	Address of Building Department is in

BUILDING SUMMARY

Attribute	Type	Description
Address	Variable String	Address of the building
Name	Variable String	Name of building
Abbr.	Variable String	Abbreviation of Name

ROOM SUMMARY

Attribute	Type	Description
Type	Variable String	Describes the type of room ex. Lab
Room Number	Integer	Uniquely identifies a room in a building
Address	Variable String	Address of building which the room is in

INSTRUCTOR SUMMARY

Attribute	Type	Description
ID	Integer	Unique ID number of an instructor
Address	Variable String	Address of instructor
Name	Variable String	Name of instructor
Dept	Variable String	Name of the department that the instructor belongs

STUDENT SUMMARY

Attribute	Type	Description
ID	Integer	Unique number of a student
Address	Variable String	Address of student
Name	Variable String	Name of student
Year	Integer	ex. 2 : Sophomore, 3: Junior, etc.

CLASS SUMMARY

Attribute	Type	Description
Class ID	5 Char String	Uniquely identifies each class in a course
DayCode	5 Char String	Represents which days a class meets ex. 01010 Tu,Th
Time	Integer	The time that a class will meet
Course ID	Variable String	Unique name identifying a course

COURSE SUMMARY

Attribute	Type	Description
Course ID	Variable String	Code to uniquely identify each course ex CS2300
Major	Variable String	Name of the major that the course belongs to
Credit Hours	Integer	Number of credit hours the course is worth

TAKES SUMMARY

Attribute	Type	Description
ID	Integer	Uniquely identifies a student
Class ID	5 Char String	Uniquely identifies a class within a course
Course ID	Variable String	Uniquely identifies a course
Grade	Float	Describes a student's grade

IN SUMMARY

Attribute	Type	Description
Course ID	Variable String	Name to identify a course
Class ID	5 Char	Code to identify a particular class in a course
Room Number	Integer	Number to identify a room in a building
Address	Variable String	Address of a building

TAUGHT BY SUMMARY

Attribute	Type	Description
ID	Integer	Unique number for each instructor
Class ID	5 Char	Unique code for each class
Course ID	Variable String	Unique name for each course

REQUIRES SUMMARY

Attribute	Type	Description
ID	Integer	Unique number for each student
Course ID	Variable String	Unique code for a course

PSEUDO CODE FOR FUNCTIONAL REQUIREMENTS

//Class Sign-up//

//This function will sign a student up for a class that doesn't conflict with their classes so far

//It accesses "class" table and "takes" table

Input: ID of person to sign up, Course ID

Steps:

(1) Select the tuples in "class" with the given Course ID and rename it to POS
(2) Select the tuples in "takes" with the given ID, and save that table as TAKING
(3) Join the "class" table and the TAKING table on the given ID, and save it as CUR
(4) Remove tuples in POS that have the same daycode and times as tuples in CUR
(5) Display the list to the user and let them pick one of the classes if any are left
(6) If they pick a class, create a new tuple in "takes" table with the Class ID, Course ID and student ID.

//==========//

//Generate Student Schedule//

//This function will display a student's schedule for the week.

//It access "class" table, "in" table, and "takes" table

Input: ID of student

Steps:

(1) Select tuples in "takes" with ID rename to TAKING
(2) Join the TAKING and the "class" table where ClassID is equal in both. Rename to CUR
(3) Join the "in" table with the CUR table where ClassID is equal in both.
(4) Look at the Daycode, Room, and Times of those classes to build a schedule
(5) Display the Schedule

//==========//

//Generate Instructor Schedule//

//This function will display an instructor's schedule of classes for the week

//It accesses "class" table, "in" table, and "taught by" table

Input: ID of Instructor

Steps:

(1) Select on "taught by" table for tuples with ID rename to INSC
(2) Join INSC with "class" table where Course ID, and Class ID match. Rename to INSC
(3) Join the new INSC table with "in" where Course ID and ClassID match. Rename to INSS
(4) Look at the Daycode and Times of those classes to build a schedule.
(5) Display the Schedule

//==========//

//Lookup Degree progress//

//This function will output a document with their required courses and courses they have not taken and their GPA so far.

It accesses the "requires" table, the "class" table. and the "takes" table

Input: ID of the student

Steps:

(1) Select on "takes" for ID, save result as TAKEN
(2) Join TAKEN with "course" and keep the name TAKEN
(3) Select on "requires" for ID where taken is false. Rename TOTAKE
(4) Display the two lists and the student's GPA

//==========//

//Get Person Info//

//This function displays the information on a person

//It accesses the “student” or “instructor” tables

Input: ID of person

Steps:

(1) Select on “student” or “instructor” table for ID
(2) Display student information on the screen along with the emergency contact info

//==========//

//Search For a Class//

//This function will search the “class” table for a user

//It access the “class” table

Input: Course ID

Steps:

(1) Select on the “class” table with the given Course ID
(2) Show the user the resulting list.

//==========//

//Add Person//

//This function will add a person to the database

//It accesses the “Student” or ”Instructor” table and possibly the “emergency contact” table

Input: The person's ID, All the attributes of a person, and Possibly emergency contact information

Steps:

(1) Select on the “person” table with the given ID
(2) If there is a result, prompt for another ID and say that the given person already exists
(3) Else, create a new tuple with the attributes in the input.
(4) Prompt the user if the person has an emergency contact
(5) If so, create a new tuple in the “emergency contact” table and prompt for the attributes of that emergency contact
(6) Else, say person added successfully

//==========//

//Remove Person//

//This function will delete a person from the database and everything that is tied to them.

//It accesses the “student” or “instructor” table, the “takes” table and “requires” table or the “part of”, “class”, “in”, and “taught by” table

Input: The person's ID and if they are a student or instructor

Steps:

(1) If Student:
    (a) Select on “takes” with ID, Delete resulting tuples
    (b) Select on “requires” with ID, Delete resulting tuples
    (c) Select on “emergency contact” with ID, Delete resulting tuples
    (d) Select on “student” with ID, Delete resulting tuples
(2) If Instructor:
    (a) Select on “part of” with ID, Delete resulting tuples
    (b) Select on “class” with ID of instructor, Run remove_class() on result
    (c) Select on “emergency contact” with ID, Delete resulting tuples
    (d) Select on “instructor” with ID, Delete resulting tuples

//==========//

//Add Course//

//This function will add a course to the database that doesn’t exist

//It accesses the “course” table

Input: Course ID and Major and Credit Hours for the course

Steps:

(1) Select on “course” with Course ID
(2) If result, say already exists, and prompt for a new Course ID
(3) If not, create a new tuple in “course” with the user input

//==========//

//Remove Course//

//This function will remove a course and depended classes

//It accesses the “course” table, the “takes” table, the “in” table, and the “class” table

Input: Course ID to delete

Steps:

(1) Select on “takes” table for any tuples with Course ID == to the input.
(2) Set the Course ID to REMVD + Course ID
(3) Select on “in” table for any tuples with Course ID == to the input.
(4) Delete the tuples
(5) Select on “class” table for all tuples with Course ID == to the input.
(6) Delete those tuples
(7) Select on “course” table with given Course ID, set the Course ID to REMVD + Course ID

//==========//

//Add Class//

//This function will add a class to the database

//It accesses the “class” table, “in” table, and “taught by” table

Input: All the attributes apart of a class

Steps:

(1) The database prompts the user for the attributes of a class and which Course ID it belongs to
(2) Add the tuple to the “class” table
(3) Then it will prompt the user to select which room the class is in
(4) Add the tuple to the “in” table
(5) Next it will prompt the user to select an instructor to teach the class
(6) Add a tuple to the “taught by” table with the given information

//==========//

//Remove Class//

//This function will remove a class from the class table and all depending tuples

//IT accesses the “class” table, “in” table, “takes” table, and “taught by”

Input: Course ID and Class ID of the class to remove

Steps:

(1) Select on “taught by” table with the given Course ID and Class ID, Delete the resulting tuples
(2) Select on “in” table with the given Course ID and Class ID, Delete the resulting tuples
(3) Select on “takes” with the given Course ID and Class ID, for the resulting tuples change the class ID to REMVD
(4) Select on the “class” table with the given Course ID and Class ID, for the resulting tuples change the Class ID to REMVD

//==========//

//Add Building//

//This function will add a building to the “building” table

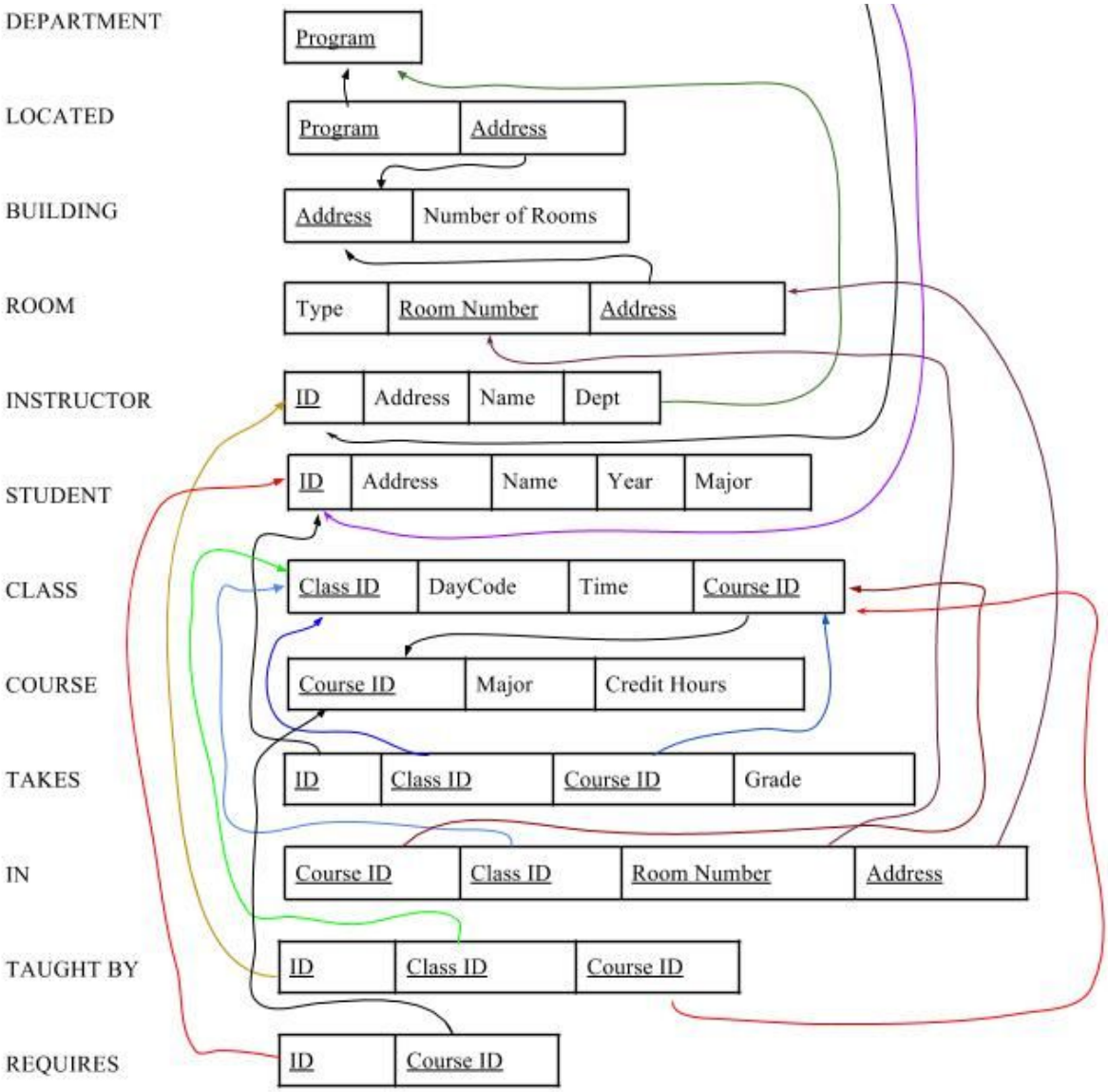
//It accesses the “building” table and “room” table

Input: Address, Name, Abbreviation, and number of rooms

Steps:

(1) Select on “building” with Address
(2) If result, say the building already exist and prompt for new Address
(3) Else, Create a new tuple in “building” with the given information
(4) In the “room” table, create a new tuple for each given number of rooms.
(5) While this is going on, prompt the user for each rooms type.

//==========//


DEPARTMENT
Program
LOCATED
Program
Address
BUILDING
Address
Number of Rooms
ROOM
Type
Room Number
Address
INSTRUCTOR
ID
Address
Name
Dept
STUDENT
ID
Address
Name
Year
Major
CLASS
Class ID
DayCode
Time
Course ID
COURSE
Course ID
Major
Credit Hours
TAKES
ID
Class ID
Course ID
Grade
IN
Course ID
Class ID
Room Number
Address
TAUGHT BY
ID
Class ID
Course ID
REQUIRES
ID
Course ID