**CSCI 340 01W Final Project**

**College Data Management**

**Ashish Acharya**

**Due Date: 5/8/2024 Wed. by 11:59pm**

**CIS**

**Step 1. Create an imaginary scenario. Your scenario should satisfy the following conditions, explain the story behind the scenarios, and all your assumptions, which are required to support relationships given below.**

The college data management system is designed to optimize administrative functions, track students, and maintain academic records efficiently. It caters to a variety of courses spread over different departments, with students taking several courses each term. Professors are responsible for teaching multiple courses, and the system also accommodates support staff such as librarians and administrative workers.

1. **Include at least one one-to-one binary relationships.**

Each student has one unique student ID.

1. **Include at least one one-to-many binary relationships.**

A professor can teach multiple courses, but each course is taught by only one professor.

1. **Include at least one many-to-many binary relationships.**

Students can enroll in multiple courses, and each course can have multiple enrolled students.

1. **Include at least one intersection data (on many-to-many relationships).**

The intersection data occurs between students and courses; each student can enroll in multiple courses, and each course can have multiple enrolled students.

The college data management system is developed to enhance administrative activities, monitor student progress, and manage academic records effectively. It supports a wide range of courses offered across multiple departments, with students enrolled in various courses every semester. Professors handle teaching duties for several courses, and the system includes roles for support staff like librarians and administrative personnel.

**Step 2. Entities with their attributes**

Here are the entities involved in the college data management system along with their attributes:

- **Student**: This entity has attributes such as Student\_ID (**Primary Key**), Name, and Date\_of\_Birth.

- **Course**: This entity is characterized by attributes like Course\_ID (**Primary Key**), Title, and Department.

- **Professor**: Attributes for this entity include Professor\_ID (**Primary Key**), Name, and Department.

- **Enrollment**: This entity includes Enrollment\_ID (**Primary Key**), Student\_ID (**Foreign Key**), and Course\_ID (**Foreign Key**).

**ER Diagram**

**A diagram of a program

Description automatically generated**

**Step 3. Define some referential integrity rules.**

In the college data management system, the following referential integrity rules and constraints are set to maintain data accuracy and consistency:

1. **Foreign Key Constraint**: This ensures that all foreign keys in the database properly reference the corresponding primary keys in their related tables.
2. **Delete Rule - Cascade Delete**: This rule automatically deletes related records in the child tables when a record in the parent table is deleted, preserving data integrity.
3. **Update Rule - Cascade Update**: When a record in the parent table is updated, related records in the child tables are also automatically updated to reflect the changes accurately.
4. **Insert Rule - Restrict Insert**: This rule prevents the insertion of records into child tables unless a corresponding record exists in the parent table.
5. **Check Constraint**: This constraint validates data entered into specific columns against predetermined criteria to ensure data consistency and accuracy.

**Step 4. Convert the ER diagram to tables. Show your tables with primary keys, as well as foreign keys if exists.**

**Student Table**

|  |  |  |
| --- | --- | --- |
| Student Id  **(Primary Key)** | Name | Date\_of\_Birth |

**Course Table**

|  |  |  |
| --- | --- | --- |
| Course\_ID (**Primary Key**) | Title | Department |

**Professor Table**

|  |  |  |
| --- | --- | --- |
| Professor\_ID  (**Primary Key**), | Name | Department |

**Enrollment Table**

|  |  |  |
| --- | --- | --- |
| Enrollment\_ID  (**Primary Key**) | Student \_ID  **(Foreign Key)** | Course\_ID  **(Foreign Key)** |

**Step 5. Discuss database normalization rules on your tables.**

Our college management database adheres to the principles of normalization, ensuring data integrity and minimizing redundancy by eliminating multivalued attributes, partial functional dependencies, and transitive dependencies.

1. **1NF (First Normal Form):** There are no multivalued attributes present in any of the tables. Each attribute within a table contains atomic values without any repeating groups. For example, in the Student table, the Name and Date\_of\_Birth attributes each contain single values for each student, adhering to 1NF.
2. **2NF (Second Normal Form):** There are no partial functional dependencies observed. Every attribute in each table is fully dependent on the primary key of its respective table. For instance, in the Enrollment table, the Student\_ID and Course\_ID attributes together form the composite primary key, and all other attributes such as Enrollment\_ID are fully dependent on this composite key, ensuring compliance with 2NF.
3. **3NF (Third Normal Form):** There are no transitive dependencies present. Each non-key attribute within the tables depends only on the primary key, with no indirect dependencies through other non-key attributes. For instance, in the Professor table, both the Name and Department attributes directly depend on the Professor\_ID primary key, without any transitive dependencies.

**Step 6. Write SQL statements to create your database and tables resulted from step 1 through 5.**

**CREATE DATABASE CollegeDB;**

**USE CollegeDB;**

**CREATE TABLE Student (**

**Student\_ID INT PRIMARY KEY,**

**Name VARCHAR(50),**

**Date\_of\_Birth DATE**

**);**

**CREATE TABLE Course (**

**Course\_ID INT PRIMARY KEY,**

**Title VARCHAR(100),**

**Department VARCHAR(50)**

**);**

**CREATE TABLE Professor (**

**Professor\_ID INT PRIMARY KEY,**

**Name VARCHAR(50),**

**Department VARCHAR(50)**

**);**

**CREATE TABLE Enrollment (**

**Enrollment\_ID INT PRIMARY KEY,**

**Student\_ID INT,**

**Course\_ID INT,**

**FOREIGN KEY (Student\_ID) REFERENCES Student(Student\_ID) ON DELETE CASCADE,**

**FOREIGN KEY (Course\_ID) REFERENCES Course(Course\_ID) ON DELETE CASCADE**

**);**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Step 7. Demonstrate data insertion into at least three tables or more.**

**Inserting data into Student Table**

**USE CollegeDB;**

**INSERT INTO Student (Student\_ID, Name, Date\_of\_Birth)**

**VALUES**

**(2000, 'John Smith', '2000-05-15'),**

**(2001, 'Emily Johnson', '2001-09-20'),**

**(2002, 'Michael Brown', '1999-12-10'),**

**(2003, 'Venessa Diaz', '2020-05-07'),**

**(2004, 'Avery Dawson', '1993-01-25');**

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Inserting data into Course Table**

**USE CollegeDB;**

**INSERT INTO Course (Course\_ID, Title, Department)**

**VALUES**

**(101, 'Introduction to Computer Science', 'Computer Science'),**

**(102, 'Calculus I', 'Mathematics'),**

**(103, 'Quantum Mechanics', 'Physics'),**

**(104, 'Database', 'Computer Science'),**

**(105, 'Calculus II', 'Mathematics'),**

**(106, 'Thermodynamics', 'Physics');**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Inserting data into Professor Table**

**USE CollegeDB;**

**INSERT INTO Professor (Professor\_ID, Name, Department)**

**VALUES**

**(2001, 'Dr. Sarah Lee', 'Computer Science'),**

**(2002, 'Prof. Mark Johnson, 'Mathematics'),**

**(2003, 'Dr. Amanda White', 'Physics'),**

**(2004, 'Prof. Mark Graham, 'Computer Science'),**

**(2005, 'Prof. Tina Dawson', 'Mathematics'),**

**(2006, 'Dr. Marry White', 'Physics');**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Insert into Enrollment Table**

**USE CollegeDB;**

**INSERT INTO Enrollment (Enrollment\_ID, Student\_ID, Course\_ID)**

**VALUES**

**(9000, 2000, 101),**

**(9001, 2000, 102),**

**(9002, 2001, 101),**

**(9003, 2001, 104),**

**(9004, 2002, 103),**

**(9005, 2002, 106),**

**(9006, 2003, 106),**

**(9007, 2004, 105),**

**(9008, 2004, 103);**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Step 8. Data Deletion from tables**

**USE CollegeDB;**

**DELETE FROM Student WHERE Student\_ID = 2003;**

**DELETE FROM Course WHERE Course\_ID = 105;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated**

These SQL statements remove a student with the Student\_ID 2003 from the "Student" table and a course with the Course\_ID 105 from the "Course" table in the "CollegeDB" database. Each statement specifies the condition for deletion based on the unique identifier of the respective table.

**Step 9. Demonstrate data update with at least one table.**

**USE CollegeDB;**

**UPDATE Student**

**SET Name = 'Robert Johnson'**

**WHERE Student\_ID = 2000;**

**UPDATE Course**

**SET Title = 'Advanced Computer Science'**

**WHERE Course\_ID = 101;**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated**

These SQL statements update the name of a student with Student\_ID 2000 to "Robert Johnson" in the "Student" table and change the title of a course with Course\_ID 101 to "Advanced Computer Science" in the "Course" table of the "CollegeDB" database.

**Step 10. Demonstrate a SELECT statement with WHERE statement.**

**USE CollegeDB;**

**SELECT \* FROM Student WHERE Date\_of\_Birth > '1993-09-20';**

**A screenshot of a computer

Description automatically generated**

This SELECT statement with WHERE statement selects all rows from the "Student" table in the "CollegeDB" database where the "Date\_of\_Birth" column is greater than '1993-09-20'.

**Step 11. Demonstrate a SELECT statement with GROUP BY statement.**

**USE CollegeDB;**

**SELECT Department, COUNT(\*) AS Total\_Courses FROM Course GROUP BY Department;**

**A screenshot of a computer

Description automatically generated**

This a SELECT statement with GROUP BY selects the "Department" column and counts the number of occurrences of each department in the "Course" table from the "CollegeDB" database. The ‘GROUP BY’ clause groups the results by department. The result set includes two columns: "Department" and "Total\_Courses", where "Total\_Courses" represents the count of courses offered in each department.

**Step 12. Demonstrate a SELECT statement with ORDER BY statement.**

**USE CollegeDB;**

**SELECT \* FROM Student ORDER BY Name ASC;**

**A screenshot of a computer

Description automatically generated**

This SELECT statement with ORDER BY selects all rows from the "Student" table in the "CollegeDB" database and orders the results by the "Name" column in ascending order. This means that the statement retrieves information about all students from the table, arranging them alphabetically by their names from A to Z.

**Step 13. Demonstrate a SELECT statement with HAVING statement.**

**USE CollegeDB;**

**SELECT Department, COUNT(\*) AS Total\_Courses FROM Course GROUP BY Department HAVING Total\_Courses > 1;**

**A screenshot of a computer

Description automatically generated**This a SELECT statement with HAVING statement selects the "Department" column and counts the number of occurrences of each department in the "Course" table from the "CollegeDB" database. It then groups the results by department using the `GROUP BY` clause. After grouping, the `HAVING` clause filters the results to include only departments where the count of courses (Total\_Courses) is greater than 1. This means that this statement retrieves information about departments that offer more than one course.

**Step 14. Using two related tables (meaning logically connected with primary-key and foreign-key pairs), join these two tables to show matching rows.**

**USE CollegeDB;**

**SELECT \* FROM Student**

**INNER JOIN Enrollment ON Student.Student\_ID = Enrollment.Student\_ID;**

**A screenshot of a computer

Description automatically generated**This SQL query retrieves data from the "Student" table and the "Enrollment" table, joining them based on the "Student\_ID" column. "Student\_ID" serves as both the primary key in the "Student" table and the foreign key in the "Enrollment" table, establishing a connection between the two tables.

**Step 15. Demonstrate a SQL statement in which a DATE data type is subject of where statement.**

**USE CollegeDB;**

**SELECT \* FROM Student WHERE Date\_of\_Birth < '2000-01-01';**

**A screenshot of a computer

Description automatically generated**

This SQL statements selects all columns from the "Student" table in the "CollegeDB" database and retrieves rows where the "Date\_of\_Birth" column is before January 1, 2000.

**Step 16. Demonstrate CREATE VIEW statement.**

**USE CollegeDB;**

**CREATE VIEW Enrollment\_Info AS**

**SELECT**

**s.Student\_ID,**

**s.Name AS Student\_Name,**

**c.Title AS Course\_Title,**

**c.Department AS Course\_Department,**

**p.Professor\_ID,**

**p.Name AS Professor\_Name**

**FROM**

**Student s**

**INNER JOIN Enrollment e ON s.Student\_ID = e.Student\_ID**

**INNER JOIN Course c ON e.Course\_ID = c.Course\_ID**

**INNER JOIN Professor p ON c.Department = p.Department**

**ORDER BY Student\_Name ASC;**

**A screenshot of a computer

Description automatically generated**

This SQL statement creates a view called "Enrollment\_Info" in the "CollegeDB" database. It combines data from the "Student", "Enrollment", "Course", and "Professor" tables. The view includes columns such as student ID and name, course title and department, and professor ID and name. The data is sorted alphabetically by student name.