**Week 1: Student Management System Database Project**

**Team Documentation**

**Team Members**

1. Mohammed Ragab - Database Architect
2. Youssef Mohammed - SQL Developer
3. Ahmed Ayman - SQL Developer
4. Ahmed Hesham - SQL Developer
5. Ahmed Harmas - Database Architect
6. Hady Sameh - Database Architect

**1. Database Design**

**1.1 Overview**

Our team designed a comprehensive Student Management System database to handle various aspects of academic administration. The schema includes functionality for:

* Student and faculty management
* Course registration and prerequisites
* Grade tracking and GPA calculation
* Department and classroom organization
* Course scheduling

**1.2 Entity-Relationship Diagram**

A computer screen shot of a diagram

Description automatically generated

**1.3 Table Structure**

The database consists of 10 primary tables:

1. **Department** - Stores academic departments
2. **Users** - Manages all system users (students, tutors, admins)
3. **Grade** - Tracks student GPAs and credit hours
4. **Courses** - Contains course information
5. **Course\_prerequisite** - Manages course prerequisites
6. **Place** - Handles classroom information
7. **ValidDays** - Contains valid days for scheduling
8. **Section** - Manages course sections and scheduling
9. **Course\_registered** - Tracks student course registration
10. **Course\_grade** - Records individual course grades

**2. Implementation**

**2.1 Database Creation**

CREATE DATABASE StudentManagementSystem;

USE StudentManagementSystem;

**2.2 Key Table Implementations**

**Users Table**

CREATE TABLE Users (

id INTEGER PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

email\_address VARCHAR(50) NOT NULL UNIQUE,

ssn VARCHAR(20) NOT NULL UNIQUE,

password\_hash VARCHAR(60) NOT NULL,

major INTEGER,

role VARCHAR(10) NOT NULL DEFAULT 'Unassigned',

FOREIGN KEY (major) REFERENCES Department(id),

CHECK (role IN ('Student', 'Tutor', 'Admin', 'Unassigned'))

);

**Grade Table**

CREATE TABLE Grade (

student\_id INTEGER PRIMARY KEY,

passed\_credit\_hours INTEGER NOT NULL DEFAULT 0,

failed\_credit\_hours INTEGER DEFAULT 0,

gpa FLOAT NOT NULL DEFAULT 0.0 CHECK (gpa >= 0 AND gpa <= 5.0),

FOREIGN KEY (student\_id) REFERENCES Users(id) ON DELETE CASCADE

);

**2.3 Advanced Features**

**Automatic GPA Calculation**

Implemented a trigger TR\_UpdateGradeTable that automatically:

* Updates student GPAs when grades are entered
* Calculates passed and failed credit hours
* Removes course registrations upon grade entry

**3. SQL Queries**

**3.1 Data Analysis Queries**

**Student Course Load**

SELECT u.first\_name, u.last\_name, c.id as course\_id, c.name as course\_name,

s.day, s.start\_time, s.end\_time

FROM Users u

JOIN Course\_registered cr ON u.id = cr.student\_id

JOIN Section s ON cr.section\_id = s.id

JOIN Courses c ON s.course\_id = c.id

WHERE u.id = 1;

**Department Enrollment Statistics**

SELECT d.name as department\_name, COUNT(u.id) as student\_count

FROM Department d

LEFT JOIN Users u ON d.id = u.major

WHERE u.role = 'Student'

GROUP BY d.id, d.name;

**3.2 Performance Analysis**

**GPA Distribution Query**

SELECT

CASE

WHEN gpa >= 4.5 THEN '4.5-5.0'

WHEN gpa >= 4.0 THEN '4.0-4.49'

WHEN gpa >= 3.5 THEN '3.5-3.99'

WHEN gpa >= 3.0 THEN '3.0-3.49'

WHEN gpa >= 2.5 THEN '2.5-2.99'

WHEN gpa >= 2.0 THEN '2.0-2.49'

ELSE 'Below 2.0'

END as gpa\_range,

COUNT(\*) as student\_count

FROM Grade

GROUP BY CASE

WHEN gpa >= 4.5 THEN '4.5-5.0'

*-- ... [rest of the CASE statement]*

END

ORDER BY gpa\_range DESC;

**4. Challenges and Solutions**

1. **Complex GPA Calculation**
   * Challenge: Automating GPA updates when new grades are entered
   * Solution: Created a sophisticated trigger that recalculates GPA, passed, and failed hours
2. **Course Prerequisites**
   * Challenge: Ensuring proper course sequencing
   * Solution: Implemented a separate Course\_prerequisite table with appropriate constraints
3. **Scheduling Conflicts**
   * Challenge: Preventing double-booking of rooms and time slots
   * Solution: Created ValidDays table and implemented proper foreign key constraints

**5. Future Improvements**

1. Add indexing for performance optimization
2. Implement stored procedures for common operations
3. Add data archiving mechanism for graduated students
4. Create views for commonly used query combinations

**6. Individual Contributions**

* Mohammed Ragab: Led database schema design
* Youssef Mohammed: Implemented core tables and triggers
* Ahmed Ayman: Implemented core tables and triggers
* Ahmed Hesham: Implemented core tables and triggers
* Ahmed Harmas: Schema design and ER diagrams
* Hady Sameh: Schema design and ER diagrams