DATABASE SYSTEMS LAB

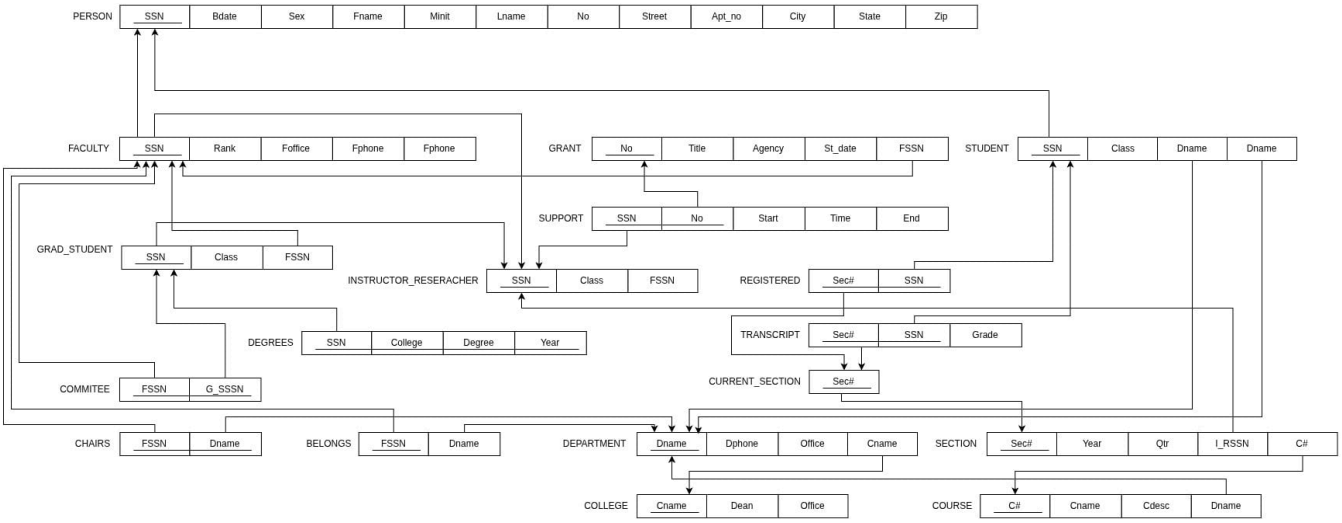
REPORT

LAB 5-6

Trần Đình Đăng Khoa - ID: 2211649
Lab Instructor: Võ Thị Kim Anh

Relational Schema for UNIVERSITY Database

Based on the ERD in Figure 4.9, the relational schema is as follow:



SQL Database Creation

```sql
-- Create the UNIVERSITY database
CREATE DATABASE UNIVERSITY;
USE UNIVERSITY;

-- Create the DEPARTMENT table
CREATE TABLE DEPARTMENT (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL,
    Location VARCHAR(100)
```

```
);

-- Create the STUDENT table
CREATE TABLE STUDENT (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DateOfBirth DATE,
    DepartmentID INT,
    Gender CHAR(1) CHECK (Gender IN ('M', 'F')),
    GraduationYear INT,
    Major VARCHAR(100),
    FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID)
);

-- Create the INSTRUCTOR table
CREATE TABLE INSTRUCTOR (
    InstructorID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DepartmentID INT,
    Salary DECIMAL(10,2),
    FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID)
);

-- Create the SEMESTER table
CREATE TABLE SEMESTER (
    SemesterID INT PRIMARY KEY,
    SemesterName VARCHAR(50) NOT NULL,
    Year INT NOT NULL,
    StartDate DATE,
    EndDate DATE
);

-- Create the COURSE table
CREATE TABLE COURSE (
    CourseID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Credits INT,
    DepartmentID INT,
    SemesterID INT,
    FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID),
    FOREIGN KEY (SemesterID) REFERENCES SEMESTER(SemesterID)
);

-- Create the ENROLLMENT table
```

```sql
CREATE TABLE ENROLLMENT (
    StudentID INT,
    CourseID INT,
    SemesterID INT,
    Grade DECIMAL(3,1) CHECK (Grade >= 0 AND Grade <= 10),
    PRIMARY KEY (StudentID, CourseID, SemesterID),
    FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
    FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID),
    FOREIGN KEY (SemesterID) REFERENCES SEMESTER(SemesterID)
);

-- Create the CLASS table
CREATE TABLE CLASS (
    ClassID INT PRIMARY KEY,
    ClassName VARCHAR(100) NOT NULL,
    Capacity INT,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID)
);

-- Create the STUDENT_CLASS junction table
CREATE TABLE STUDENT_CLASS (
    StudentID INT,
    ClassID INT,
    PRIMARY KEY (StudentID, ClassID),
    FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
    FOREIGN KEY (ClassID) REFERENCES CLASS(ClassID)
);

-- Create the RESEARCH_PROJECT table
CREATE TABLE RESEARCH_PROJECT (
    ProjectID INT PRIMARY KEY,
    Title VARCHAR(200) NOT NULL,
    Budget DECIMAL(12,2),
    StartDate DATE,
    EndDate DATE,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID)
);

-- Create the PROJECT_ADVISOR junction table
CREATE TABLE PROJECT_ADVISOR (
    ProjectID INT,
    InstructorID INT,
    Role VARCHAR(50),
    PRIMARY KEY (ProjectID, InstructorID),
```

```
    FOREIGN KEY (ProjectID) REFERENCES RESEARCH_PROJECT(ProjectID),
    FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID)
);

-- Create the STUDENT_PROJECT junction table
CREATE TABLE STUDENT_PROJECT (
    ProjectID INT,
    StudentID INT,
    PRIMARY KEY (ProjectID, StudentID),
    FOREIGN KEY (ProjectID) REFERENCES RESEARCH_PROJECT(ProjectID),
    FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID)
);
```

Data Querying Implementation

Basic Queries

1. List all students with ID, name, and birth date

```
SELECT StudentID, CONCAT(FirstName, ' ', LastName) AS FullName, DateOfBirth
FROM STUDENT;
```

2. List instructors from Computer Science department

```
SELECT i.InstructorID, i.FirstName, i.LastName
FROM INSTRUCTOR i
JOIN DEPARTMENT d ON i.DepartmentID = d.DepartmentID
WHERE d.DepartmentName = 'Computer Science';
```

3. List courses taught in the current semester

```
SELECT c.CourseID, c.Title
FROM COURSE c
JOIN SEMESTER s ON c.SemesterID = s.SemesterID
WHERE CURRENT_DATE BETWEEN s.StartDate AND s.EndDate;
```

4. Find students with surname "Nguyen"

```
SELECT StudentID, FirstName, LastName
FROM STUDENT
WHERE LastName = 'Nguyen';
```

5. Find students with GPA above 8.0

```sql
SELECT s.StudentID, s.FirstName, s.LastName, AVG(e.Grade) AS GPA
FROM STUDENT s
JOIN ENROLLMENT e ON s.StudentID = e.StudentID
GROUP BY s.StudentID, s.FirstName, s.LastName
HAVING AVG(e.Grade) > 8.0;
```

Intermediate Queries

6. List students by department, ordered by name

```sql
SELECT d.DepartmentName, s.FirstName, s.LastName
FROM STUDENT s
JOIN DEPARTMENT d ON s.DepartmentID = d.DepartmentID
ORDER BY d.DepartmentName, s.LastName, s.FirstName;
```

7. Find students with highest GPA in each class

```sql
WITH StudentGPA AS (
    SELECT s.StudentID, s.FirstName, s.LastName, sc.ClassID,
           AVG(e.Grade) AS GPA,
           RANK() OVER (PARTITION BY sc.ClassID ORDER BY AVG(e.Grade) DESC)
as RankInClass
    FROM STUDENT s
    JOIN STUDENT_CLASS sc ON s.StudentID = sc.StudentID
    JOIN ENROLLMENT e ON s.StudentID = e.StudentID
    GROUP BY s.StudentID, sc.ClassID
)
SELECT sg.ClassID, c.ClassName, sg.StudentID, sg.FirstName, sg.LastName,
sg.GPA
FROM StudentGPA sg
JOIN CLASS c ON sg.ClassID = c.ClassID
WHERE sg.RankInClass = 1;
```

8. Count enrolled students per course in current semester

```sql
SELECT c.CourseID, c.Title, COUNT(e.StudentID) AS EnrolledStudents
FROM COURSE c
LEFT JOIN ENROLLMENT e ON c.CourseID = e.CourseID
JOIN SEMESTER s ON c.SemesterID = s.SemesterID
WHERE CURRENT_DATE BETWEEN s.StartDate AND s.EndDate
GROUP BY c.CourseID, c.Title;
```

9. Find instructors who haven't advised any research projects

```sql
SELECT i.InstructorID, i.FirstName, i.LastName
FROM INSTRUCTOR i
WHERE i.InstructorID NOT IN (SELECT DISTINCT InstructorID FROM
PROJECT_ADVISOR);
```

10. Find students enrolled in at least 3 courses in current semester

```sql
SELECT s.StudentID, s.FirstName, s.LastName, COUNT(e.CourseID) AS
CourseCount
FROM STUDENT s
JOIN ENROLLMENT e ON s.StudentID = e.StudentID
JOIN SEMESTER sem ON e.SemesterID = sem.SemesterID
WHERE CURRENT_DATE BETWEEN sem.StartDate AND sem.EndDate
GROUP BY s.StudentID, s.FirstName, s.LastName
HAVING COUNT(e.CourseID) >= 3;
```

Advanced Queries

11. Find courses with at least 10 enrolled students

```sql
SELECT c.CourseID, c.Title, COUNT(e.StudentID) AS EnrolledStudents
FROM COURSE c
JOIN ENROLLMENT e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.Title
HAVING COUNT(e.StudentID) >= 10;
```

12. Find students with all grades above 7.0

```sql
SELECT s.StudentID, s.FirstName, s.LastName
FROM STUDENT s
WHERE NOT EXISTS (
    SELECT 1
    FROM ENROLLMENT e
    WHERE e.StudentID = s.StudentID AND e.Grade <= 7.0
) AND EXISTS (
    SELECT 1 FROM ENROLLMENT e WHERE e.StudentID = s.StudentID
);
```

13. List students born in October

```sql
SELECT StudentID, FirstName, LastName, DateOfBirth
FROM STUDENT
WHERE MONTH(DateOfBirth) = 10;
```

14. Find highest-paid instructors in each department

```sql
WITH DeptSalaries AS (
    SELECT i.InstructorID, i.FirstName, i.LastName, i.Salary,
i.DepartmentID,
           RANK() OVER (PARTITION BY i.DepartmentID ORDER BY i.Salary DESC)
as SalaryRank
    FROM INSTRUCTOR i
)
SELECT ds.InstructorID, ds.FirstName, ds.LastName, ds.Salary,
d.DepartmentName
FROM DeptSalaries ds
JOIN DEPARTMENT d ON ds.DepartmentID = d.DepartmentID
WHERE ds.SalaryRank = 1;
```

15. Calculate total research funding by department

```sql
SELECT d.DepartmentID, d.DepartmentName, SUM(rp.Budget) AS TotalFunding
FROM DEPARTMENT d
LEFT JOIN RESEARCH_PROJECT rp ON d.DepartmentID = rp.DepartmentID
GROUP BY d.DepartmentID, d.DepartmentName
ORDER BY TotalFunding DESC;
```

Implementation of Views, Stored Procedures, and Triggers

Views

1. Create StudentList View

```sql
CREATE VIEW DanhSachSinhVien AS
SELECT s.StudentID, s.FirstName, s.LastName, s.DateOfBirth, d.DepartmentName
FROM STUDENT s
JOIN DEPARTMENT d ON s.DepartmentID = d.DepartmentID;
```

2. Create ClassAverageGrade View

```sql
CREATE VIEW DiemTrungBinhLop AS
SELECT c.ClassID, c.ClassName, AVG(e.Grade) AS AverageGrade
FROM CLASS c
JOIN STUDENT_CLASS sc ON c.ClassID = sc.ClassID
JOIN STUDENT s ON sc.StudentID = s.StudentID
JOIN ENROLLMENT e ON s.StudentID = e.StudentID
GROUP BY c.ClassID, c.ClassName;
```

Stored Procedures

3. Create AddStudent Procedure

```sql
DELIMITER //
CREATE PROCEDURE ThemSinhVien(
    IN p_FirstName VARCHAR(50),
    IN p_LastName VARCHAR(50),
    IN p_DateOfBirth DATE,
    IN p_DepartmentID INT,
    IN p_Gender CHAR(1),
    IN p_Major VARCHAR(100)
)
BEGIN
    DECLARE new_id INT;
    SELECT COALESCE(MAX(StudentID), 0) + 1 INTO new_id FROM STUDENT;

    INSERT INTO STUDENT (StudentID, FirstName, LastName, DateOfBirth,
DepartmentID, Gender, Major)
    VALUES (new_id, p_FirstName, p_LastName, p_DateOfBirth, p_DepartmentID,
p_Gender, p_Major);

    SELECT new_id AS NewStudentID;
END //
DELIMITER ;
```

4. Create UpdateGrade Procedure

```sql
DELIMITER //
CREATE PROCEDURE CapNhatDiem(
    IN p_StudentID INT,
    IN p_CourseID INT,
    IN p_SemesterID INT,
    IN p_Grade DECIMAL(3,1)
)
BEGIN
```

```
    UPDATE ENROLLMENT
    SET Grade = p_Grade
    WHERE StudentID = p_StudentID
    AND CourseID = p_CourseID
    AND SemesterID = p_SemesterID;

    IF ROW_COUNT() = 0 THEN
        INSERT INTO ENROLLMENT (StudentID, CourseID, SemesterID, Grade)
        VALUES (p_StudentID, p_CourseID, p_SemesterID, p_Grade);
    END IF;
END //
DELIMITER ;
```

Triggers

5. Create GPA Update Trigger

```
DELIMITER //
CREATE TRIGGER CapNhatDiemTrungBinh
AFTER INSERT ON ENROLLMENT
FOR EACH ROW
BEGIN
    -- Assuming we have a StudentGPA table to store calculated GPAs
    -- If not, this would be handled by application logic or views
    UPDATE StudentGPA
    SET GPA = (
        SELECT AVG(Grade)
        FROM ENROLLMENT
        WHERE StudentID = NEW.StudentID
    )
    WHERE StudentID = NEW.StudentID;
END //
DELIMITER ;
```

6. Create Class Capacity Check Trigger

```
DELIMITER //
CREATE TRIGGER KiemTraSoLuongSinhVien
BEFORE INSERT ON STUDENT_CLASS
FOR EACH ROW
BEGIN
    DECLARE student_count INT;
    DECLARE max_capacity INT;
```

```
    SELECT COUNT(*)
    INTO student_count
    FROM STUDENT_CLASS
    WHERE ClassID = NEW.ClassID;

    SELECT Capacity
    INTO max_capacity
    FROM CLASS
    WHERE ClassID = NEW.ClassID;

    IF student_count >= max_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Class has reached maximum capacity';
    END IF;
END //
DELIMITER ;
```

Performance Analysis and Optimization

For complex queries such as finding students with the highest GPA in each class, the performance can be improved by:

1. **Indexing Strategy**: Creating indexes on frequently joined or filtered columns:

```
CREATE INDEX idx_student_department ON STUDENT(DepartmentID);
CREATE INDEX idx_enrollment_student ON ENROLLMENT(StudentID);
CREATE INDEX idx_enrollment_course ON ENROLLMENT(CourseID);
CREATE INDEX idx_enrollment_semester ON ENROLLMENT(SemesterID);
```

2. **Query Optimization**: Using window functions where appropriate instead of complex self-joins or subqueries.
3. **Database Normalization**: The current schema follows normalized design principles, reducing redundancy and maintaining data integrity.
4. **Execution Plan Analysis**: For key queries, we can use EXPLAIN to analyze the execution path and identify bottlenecks:

```
EXPLAIN SELECT s.StudentID, s.FirstName, s.LastName, AVG(e.Grade) AS GPA
FROM STUDENT s
JOIN ENROLLMENT e ON s.StudentID = e.StudentID
```

```sql
GROUP BY s.StudentID, s.FirstName, s.LastName
HAVING AVG(e.Grade) > 8.0;
```