|  |  |
| --- | --- |
| **DBI202 - Database system of Academic results Fpt University**  Teacher: Đỗ Thị Thu Nga | **Team 7**  DOÃN QUỐC BẢO: Design, Query  NGUYỄN TIẾN ĐẠT: Query, Document  TẠ VIỆT HOÀNG: Query, Document  NGUYỄN DOÃN HIẾU: Query, Data  HỒ PHAN HẢI NAM: Query, Data |

# TABLE OF CONTENTS

## INTRODUCE THE PROBLEM…………………………………………………………………………………………….

### Describe the problem……………………………………………………………………………………………………….

### Management objectives…………………………………………………………………………………………………..

## entity – relationship – er…………………………………………………………………………………………

### difinITION entity – attributE…………………………………………………………………………………………..

### set-up entity – relationship…………………………………………………………………………………………….

## data dictionary………………………………………………………………………………………………………….

### DEFINITION OF TABLES………………………………………………………………………………………………………….

### SET-UP TRIGGER…………………………………………………………………………………………………………………….

## Query………………………………………………………………………………………………………….

# INTRODUCE THE PROBLEM

## 1, Describe the problem

FPT Universities are experiencing rapid growth in the number of students and require an efficient data management system to keep track of individual student information. Currently, student information is stored in various files, books, and unrelated systems, leading to complexity in information retrieval, semester tracking, and report generation.

With the advancement of technology, the school wants to build an automated and integrated student management database to help them monitor the learning progress, manage academic-related data, and provide specific information to stakeholders such as parents and regulatory authorities.

We, group 7, intend to create one of the basic realist databases to ensure perfection in managing students’ info. After a time of consideration, we’ve decided to give out these results:

* Students, recruited by FPT, all have their own information,

which is required as their entrance conditions. Diving deeper into the semester,

Unis will use that info to define and group students by certain groups of info.

* Students: in sum, students’ personal info, both for Uni entrance conditions and for management.
* Major: Each student admitted to the Uni has the right to choose 1 in a whole listed Major, though each still has their own variety, which can be found suitable for any student.
* Course: Belong to the Major, we have a course, that introduces the students to a long session of lectures but separated in general,

courses will differ based on the Major that the students choose.

Courses in a semester will connect to each other as a condition, for example: you need to finish PRO in the previous semester to learn LAB in the succeeding semester.

* Assessment: These are like qualification tests for the students, to show how and in which method the student studied over the past semester effectively, besides, each alternative will have its own minimum required score, which will be based on the importance of the assessment.
* Results: During the whole semester, students will have to face and conquer a diversity of points, included in an academic transcript then to

evaluate the student’s effort and performance during the process. And consider if they either have to retake the course or the final exam.

* Penalty: Talking about the results, for the students who performed effortlessly in the past semester, or in any other situation that they cannot pass the final exam and achieve the desired MINIMUM results, they will then have to face a thing called course retake fee. These will even based on when and how soon you started to retake those courses. Other than that, students who cheat or vandalize, etc... that interfere with the Uni’s rules, will also have to face fees or even disqualification.
* Main Goal:

This database aims to help monitor students' learning progress, manage academic-related data. It intends to streamline the management of students' personal information, majors, courses, assessments, results, and penalties to ensure efficient data management within the educational institution.

# Entity – relationship – er

## difinITION entity – attributE

Given the problem description and management goals, we can outline various entities and their associated attributes as follows:

* Student: Store a student’s information

**StudentID,** First Name, Last Name, MajorID, Sex, DateOfBirth, Address, EnrollYear, Email, Scholarship.

* + EnrollYear: The year which that student enrolls to the school.
  + Schoolarship: The scholarship of a student, from 0 means no scholarship, to 1 means full scholarship.
* Study : Store information about courses the students study

**StudentID**, **CourseId**, **Semester** , AverageScore

* + AverageScore: The average score of a course of a student study in a semester.
* Major : Store information about the majors the school has

**MajorID** , Name.

* Penalty: Store information about penalty of students

**StudentID**, **PenaltyName**, Fee, Date, Details

* + Date: The date which the penalty occurred.
  + Details: The reason for the penalty.
* Course: Store information about the courses the school provides

**CourseID**, CourseName, NumberOfAssessments.

* + NumberOfAssessments: The number of assessments this course has.
* Assessment:Store information about the assessments the courses have

**AssessmentID**, AssessmentName, **CourseID**, PassingScore, Weight

* + PassingScore: The minimum score for passing this particular assessment. Students have to pass all the assessments to pass a course.
  + Weight: The weight of this assessment to the overall score of a course (eg: Weight=0.4 means the assessment take 40% of the overall score)
* Result: Store the results of each assessment

**StudentID**, **CourseID, AssessmentID**, **Semester**, Score

* Prerequisite : Store information about courses that have prerequisite courses that need to be passed before studying

**CourseID**, PrerequisiteID.

## Set-Up ERD

|  |  |
| --- | --- |
| * Key / identifier attribute |  |
| * Attribute description / description |  |
| * Entity | **ENTITY** |
| * Weak entity | **WEAK ENTITY** |
| * Relationship | *Relationship* |
| * Connectivity (force) = 1 |  |
| * Connectivity = N |  |

A diagram of a flowchart

Description automatically generated

A diagram of a student

Description automatically generated

# data dictionary

Just for example on some tables (other table are similar, you have to define all the tables in your database). Note: to run the query you have to define the table 1 first then go to the side tables much

## DEFINITION OF TABLES

### **Table Student**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| StudentID | Varchar(10) |  |  | PK, Not null |
| LastName | nvarchar(50) |  |  | not null |
| FirstName | nvarchar(50) |  |  | not null |
| MajorId | varchar(5) |  |  | FK references Major(MajorId) |
| Sex | char(1) |  | ‘F’ or ‘M’ | not null |
| DateOfBirth | date |  |  |  |
| Address | nvarchar(100) |  |  |  |
| EnrollYear | int |  |  | not null |
| Email | varchar(50) |  |  | not null |
| Schoolarship | float | 0.0 | >=0 and <= 1 | not null |

### **Table major**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| MajorId | varchar(5) |  |  | PK |
| Name | nvarchar(50) |  |  | not null |

### **Table course**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| CourseId | varchar(10) |  |  | PK |
| CourseName | nvarchar(50) |  |  | not null |
| NumberOfAssessments | int |  | >= 0 | not null |

### **Table prerequisite**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| CourseId | varchar(10) |  |  | unique, FK references Course(CourseId) |
| PrerequisiteId | varchar(10) |  |  | unique, FK references Course(CourseId) |

### **table study**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| StudentId | varchar(10) |  |  | PK, FK references Student(StudentId) |
| CourseId | varchar(10) |  |  | PK, FK references Course(CourseId) |
| Semester | varchar(10) |  |  | PK |
| AverageScore | float | 0.0 | >= 0 and <= 10 |  |

### **table assessment**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| AssessmentId | varchar(10) |  |  | PK |
| AssessmentName | nvarchar(50) |  |  | not null |
| CourseId | varchar(10) |  |  | PK, FK references Course(CourseId) |
| PassingScore | float |  | >= 0 and <= 10 | not null |
| Weight | float |  | > 0 and <= 1 | not null |

### **table result**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| AssessmentId | | varchar(10) |  |  | PK |
| StudentId | | varchar(10) |  |  | PK, FK references Student(StudentId) |
| CourseId | | varchar(10) |  |  | PK, FK references Assessment(CourseId) |
| AssessmentId | | varchar(10) |  |  | PK, FK references Assessment(AssessmentId) |
| Semester | | varchar(10) |  |  | PK |
| Score | | float | 0 | >= 0 and <= 10 | not null |

### **Table penalty**

| Column Name | Data Type | Default | Check | Key/ Index/ Constraint |
| --- | --- | --- | --- | --- |
| StudentId | | varchar(10) |  |  | PK, FK references Student(StudentId) |
| PenaltyName | | nvarchar(50) |  |  | PK |
| Fee | | money |  |  |  |
| Date | | date |  |  | PK |
| Details | | nvarchar(200) |  |  |  |
| StudentId | | varchar(10) |  |  | PK, FK references Student(StudentId) |

## Set-up trigger

Update AverageScore of Study when insert, update, delete Result

create trigger tr\_Study\_CalcAverageScore

on Result after insert, update, delete

as

begin

declare @AverageScore float;

declare @StudentId varchar(10);

declare @CourseId varchar(10);

select @StudentId = inserted.StudentId, @CourseId = inserted.CourseId

from inserted;

select @AverageScore = sum(r.Score \* ass.[Weight])

from Result as r

right join Assessment as ass on r.AssessmentId = ass.AssessmentId and r.CourseId = ass.CourseId

where r.StudentId = @StudentId and r.CourseId = @CourseId

and not (ass.AssessmentId = 'FE' and exists (

select 1 from Result where StudentId = @StudentId and CourseId = @CourseId and AssessmentId = 'FE2ND'

)

);

update Study

set AverageScore = @AverageScore

where StudentId = @StudentId and CourseId = @CourseId;

end

# Query

1,Find the highest score for each assessment in a specific course

CREATE PROCEDURE HighestInCourse @IDFind VARCHAR (10)

AS

SELECT A.AssessmentName, MAX(R.Score) AS HighestScore

FROM Assessment AS A

INNER JOIN Result AS R ON A.CourseId = R.CourseId AND A.AssessmentId = R.AssessmentId

WHERE A.CourseId = @IDFind

GROUP BY A.AssessmentName;

2, Calculate the total scholarship amount awarded :

SELECT SUM(Schoolarship) AS TotalScholarshipAmount

FROM Student;

3, List students who passed all assessments in a specific course :

CREATE PROCEDURE PassedCourses @CourseID VARCHAR(10)

AS

SELECT S.StudentId, S.LastName, S.FirstName

FROM Student AS S

WHERE S.StudentId NOT IN (

SELECT R.StudentId

FROM Result AS R

LEFT JOIN Assessment AS A ON R.AssessmentId = A.AssessmentId AND R.CourseId = A.CourseId

WHERE A.CourseId = @CourseID AND (R.Score < A.PassingScore OR R.Score IS NULL)

);

5, List students with penalties incurred before a certain date:

CREATE PROCEDURE PenaltyDetail @date VARCHAR (15)

AS

SELECT S.StudentId, S.LastName, S.FirstName, P.PenaltyName, P.Fee, P.Date, P.Details

FROM Student AS S

INNER JOIN Penalty AS P ON S.StudentId = P.StudentId

WHERE P.Date < @date;

6, List students who failed an assessment with a specific passing score:

CREATE PROCEDURE FailedScore @score FLOAT

AS

SELECT S.StudentId, S.LastName, S.FirstName, A.CourseId, A.AssessmentId, R.Score

FROM Student AS S

INNER JOIN Result AS R ON S.StudentId = R.StudentId

INNER JOIN Assessment AS A ON R.CourseId = A.CourseId AND R.AssessmentId = A.AssessmentId

WHERE R.Score < @score AND R.Score IS NOT NULL;

9, Delete a Course:

CREATE PROCEDURE DeleteCourse @courseID NVARCHAR (15)

AS

DELETE FROM Course

WHERE CourseId = @courseID

10, Select Students in Major:

CREATE PROCEDURE DisplayStudentByMajor @MAJOR NVARCHAR (15)

AS

SELECT StudentId, LastName, FirstName

FROM Student

WHERE MajorId = @MAJOR;

15, Select Students Enrolled in a Specific Semester:

CREATE PROCEDURE DisplayBySemester @semester VARCHAR (5)

AS

SELECT S.StudentId, S.LastName, S.FirstName, ST.Semester, C.CourseName

FROM Student S

JOIN Study ST ON S.StudentId = ST.StudentId

JOIN Course C ON ST.CourseId = C.CourseId

WHERE ST.Semester = @semester;

16, Select Courses Taken by a Student:

CREATE PROCEDURE DisplayCourseByStudent @studentID NVARCHAR(15)

AS

SELECT S.StudentId, S.LastName, S.FirstName, C.CourseName

FROM Student S

JOIN Study ST ON S.StudentId = ST.StudentId

JOIN Course C ON ST.CourseId = C.CourseId

WHERE S.StudentId = @studentID;

17, Select Students Who Passed a Specific Assessment:

CREATE PROCEDURE DisplayStudentByAssessment

@assessment NVARCHAR (10),

@score FLOAT

AS

SELECT S.StudentId, S.LastName, S.FirstName

FROM Student S

JOIN Result R ON S.StudentId = R.StudentId

WHERE R.AssessmentId = @assessment AND R.Score >= @score;

18, Calculate the Total Penalty Fees for Each Student:

SELECT P.StudentId, SUM(Fee) AS TotalPenaltyFees

FROM Penalty P

GROUP BY P.StudentId;

19, List students who received a scholarship above the average scholarship:

SELECT S.StudentId, S.LastName, S.FirstName, S.Schoolarship

FROM Student AS S

WHERE S.Schoolarship > (

SELECT AVG(Schoolarship)

FROM Student

);

20, Find the student with the highest average score:

SELECT TOP 1 S.StudentId, S.LastName, S.FirstName, MAX(St.AverageScore) AS HighestAverageScore

FROM Student AS S

INNER JOIN Study AS St ON S.StudentId = St.StudentId

GROUP BY S.StudentId, S.LastName, S.FirstName

ORDER BY HighestAverageScore DESC;

21, Calculate the overall passing rate for each course:

SELECT C.CourseName,

COUNT(CASE WHEN R.Score >= A.PassingScore THEN 1 ELSE NULL END) AS PassCount,

COUNT(\*) AS TotalCount,

(COUNT(CASE WHEN R.Score >= A.PassingScore THEN 1 ELSE NULL END) \* 100.0) / NULLIF(COUNT(\*), 0) AS PassingRate

FROM Course AS C

LEFT JOIN Assessment AS A ON C.CourseId = A.CourseId

LEFT JOIN Result AS R ON C.CourseId = R.CourseId AND A.AssessmentId = R.AssessmentId

GROUP BY C.CourseName;

22, List students who passed all assessments in all their courses:

SELECT S.StudentId, S.LastName, S.FirstName

FROM Student AS S

WHERE NOT EXISTS (

SELECT C.CourseId

FROM Course AS C

WHERE NOT EXISTS (

SELECT A.AssessmentId

FROM Assessment AS A

WHERE NOT EXISTS (

SELECT R.StudentId

FROM Result AS R

WHERE R.StudentId = S.StudentId

AND R.CourseId = C.CourseId

AND R.AssessmentId = A.AssessmentId

AND R.Score >= A.PassingScore

)

)

);

23, Calculate the average scholarship amount for each major:

SELECT M.[MajorId], M.[Name] AS MajorName,

AVG(S.[Schoolarship]) AS AverageScholarshipAmount

FROM Major AS M

LEFT JOIN Student AS S ON M.[MajorId] = S.[MajorId]

GROUP BY M.[MajorId], M.[Name]

ORDER BY M.[MajorId];

24, List students with the highest and lowest scholarship amounts:

WITH RankedStudents AS (

SELECT StudentId, LastName, FirstName, Schoolarship,

ROW\_NUMBER() OVER (ORDER BY Schoolarship DESC) AS HighRank,

ROW\_NUMBER() OVER (ORDER BY Schoolarship ASC) AS LowRank

FROM Student

)

SELECT \*

FROM RankedStudents

WHERE HighRank = 1 OR LowRank = 1;

25, Calculate the total scholarship amount for each major:

SELECT M.MajorId, M.Name AS MajorName, SUM(S.Schoolarship) AS TotalScholarshipAmount

FROM Major AS M

LEFT JOIN Student AS S ON M.MajorId = S.MajorId

GROUP BY M.MajorId, M.Name

ORDER BY M.MajorId;

26, Find the student with the highest total score across all assessments:

SELECT TOP 1 S.StudentId, S.LastName, S.FirstName, SUM(R.Score) AS TotalScore

FROM Student AS S

INNER JOIN Result AS R ON S.StudentId = R.StudentId

GROUP BY S.StudentId, S.LastName, S.FirstName

ORDER BY TotalScore DESC;

27, List courses with the highest and lowest average scores:

WITH CourseAverageScores AS (

SELECT C.CourseId, C.CourseName, AVG(R.Score) AS AverageScore

FROM Course AS C

LEFT JOIN Result AS R ON C.CourseId = R.CourseId

GROUP BY C.CourseId, C.CourseName

)

SELECT C1.CourseId, C1.CourseName AS CourseNameHighestAverage,

C2.CourseId, C2.CourseName AS CourseNameLowestAverage

FROM (

SELECT CourseId, CourseNameHighestAverage

FROM CourseAverageScores

WHERE AverageScore = (SELECT MAX(AverageScore) FROM CourseAverageScores)

) AS C1

CROSS JOIN (

SELECT CourseId, CourseNameLowestAverage

FROM CourseAverageScores

WHERE AverageScore = (SELECT MIN(AverageScore) FROM CourseAverageScores)

) AS C2;

28, Find students who have consistently improved their scores across semesters:

WITH ImprovedStudents AS (

SELECT S.StudentId, S.LastName, S.FirstName, R.CourseId, R.Semester, MAX(R.Score) AS MaxScore

FROM Student AS S

INNER JOIN Result AS R ON S.StudentId = R.StudentId

GROUP BY S.StudentId, S.LastName, S.FirstName, R.CourseId, R.Semester

)

SELECT I.StudentId, I.LastName, I.FirstName

FROM ImprovedStudents AS I

GROUP BY I.StudentId, I.LastName, I.FirstName

HAVING COUNT(DISTINCT I.MaxScore) = COUNT(I.MaxScore);

29, Calculate the average score for each student across all courses:

SELECT S.StudentId, S.LastName, S.FirstName, AVG(R.Score) AS AverageScore

FROM Student AS S

INNER JOIN Result AS R ON S.StudentId = R.StudentId

GROUP BY S.StudentId, S.LastName, S.FirstName

ORDER BY S.StudentId;

30, List assessments with the highest and lowest average scores:

WITH AssessmentAverageScores AS (

SELECT A.AssessmentId, A.AssessmentName, AVG(R.Score) AS AverageScore

FROM Assessment AS A

LEFT JOIN Result AS R ON A.AssessmentId = R.AssessmentId AND A.CourseId = R.CourseId

GROUP BY A.AssessmentId, A.AssessmentName

)

SELECT A1.AssessmentId, A1.AssessmentName AS AssessmentNameHighestAverage,

A2.AssessmentId, A2.AssessmentName AS AssessmentNameLowestAverage

FROM (

SELECT AssessmentId, AssessmentNameHighestAverage

FROM AssessmentAverageScores

WHERE AverageScore = (SELECT MAX(AverageScore) FROM AssessmentAverageScores)

) AS A1

CROSS JOIN (

SELECT AssessmentId, AssessmentNameLowestAverage

FROM AssessmentAverageScores

WHERE AverageScore = (SELECT MIN(AverageScore) FROM AssessmentAverageScores)

) AS A2;

31, Find students who have failed an assessment more than once:

SELECT S.StudentId, S.LastName, S.FirstName, A.AssessmentId, A.AssessmentName

FROM Student AS S

INNER JOIN Result AS R ON S.StudentId = R.StudentId

INNER JOIN Assessment AS A ON R.AssessmentId = A.AssessmentId

WHERE R.Score < A.PassingScore

GROUP BY S.StudentId, S.LastName, S.FirstName, A.AssessmentId, A.AssessmentName

HAVING COUNT(\*) > 1;

32, Calculate the total number of assessments taken by each student:

SELECT S.StudentId, S.LastName, S.FirstName, COUNT(R.AssessmentId) AS TotalAssessments

FROM Student AS S

LEFT JOIN Result AS R ON S.StudentId = R.StudentId

GROUP BY S.StudentId, S.LastName, S.FirstName

ORDER BY S.StudentId;

33, Find the most common penalty incurred by students:

SELECT PenaltyName, COUNT(PenaltyName) AS PenaltyCount

FROM Penalty

GROUP BY PenaltyName

ORDER BY PenaltyCount DESC

LIMIT 1;

34, Calculate the overall average score for each major:

SELECT M.Name AS MajorName, AVG(R.Score) AS AverageScore

FROM Major AS M

LEFT JOIN Student AS S ON M.MajorId = S.MajorId

LEFT JOIN Result AS R ON S.StudentId = R.StudentId

GROUP BY M.Name

ORDER BY MajorName;

35, Find students who received penalties for the same reason multiple times:

SELECT P.StudentId, S.LastName, S.FirstName, P.PenaltyName, COUNT(P.PenaltyName) AS PenaltyCount

FROM Penalty AS P

INNER JOIN Student AS S ON P.StudentId = S.StudentId

GROUP BY P.StudentId, S.LastName, S.FirstName, P.PenaltyName

HAVING COUNT(P.PenaltyName) > 1

ORDER BY P.StudentId, PenaltyCount DESC;

36, Calculate passing rate of each course:

SELECT C.CourseName,

COUNT(CASE WHEN R.Score >= A.PassingScore AND S.AverageScore >= 5.0 THEN 1 ELSE NULL END) AS PassCount,

COUNT(\*) AS TotalCount,

(COUNT(CASE WHEN R.Score >= A.PassingScore AND S.AverageScore >= 5.0 THEN 1 ELSE NULL END) \* 100.0) / NULLIF(COUNT(\*), 0) AS PassingRate

FROM Course AS C

LEFT JOIN Assessment AS A ON C.CourseId = A.CourseId

LEFT JOIN Result AS R ON C.CourseId = R.CourseId AND A.AssessmentId = R.AssessmentId

LEFT JOIN Study AS S ON C.CourseId = S.CourseId

GROUP BY C.CourseName;

37, Calculate the average fee collected from penalties for each penalty name:

SELECT PenaltyName, AVG(Fee) AS AverageFee

FROM Penalty

GROUP BY PenaltyName;

38, Find students who have received scholarships and penalties:

SELECT S.StudentId, S.LastName, S.FirstName

FROM Student AS S

WHERE S.Schoolarship > 0

AND S.StudentId IN (

SELECT P.StudentId

FROM Penalty AS P

);

39, Calculate the average score improvement for students between two specific semesters:

CREATE PROCEDURE DispPerfoByTimeBetween @se1 NVARCHAR(5), @se2 NVARCHAR(5)

AS

SELECT S.StudentId, S.LastName, S.FirstName,

AVG(R2.Score - R1.Score) AS AverageImprovement

FROM Student AS S

JOIN Result AS R1 ON S.StudentId = R1.StudentId

JOIN Result AS R2 ON S.StudentId = R2.StudentId

WHERE R1.Semester = @se1

AND R2.Semester = @se2

GROUP BY S.StudentId, S.LastName, S.FirstName

HAVING AVG(R2.Score - R1.Score) IS NOT NULL;

46, Calculate the GPA (Grade Point Average) for a specific student:

CREATE PROCEDURE CalcGPA @studentID NVARCHAR (10)

AS

SELECT s.FirstName, s.LastName, AVG(r.Score) AS GPA

FROM Student s

JOIN Result r ON s.StudentId = r.StudentId

WHERE s.StudentId = @studentID

GROUP BY s.FirstName, s.LastName;

47, Retrieve the names of all students who are majoring in a specific major:

CREATE PROCEDURE DisplayByMajor @majorID NVARCHAR (10)

AS

SELECT FirstName, LastName

FROM Student

WHERE MajorId = (SELECT MajorId FROM Major WHERE Name = @majorID);

48, Retrieve all students who have incurred a penalty of a certain type:

CREATE PROCEDURE DispPenStudent @type NVARCHAR (20)

AS

SELECT s.FirstName, s.LastName, p.PenaltyName

FROM Student s

JOIN Penalty p ON s.StudentId = p.StudentId

WHERE p.PenaltyName = @type;

50, Retrieve the courses a specific student has taken:

CREATE PROCEDURE StudentCourses @studentID NVARCHAR (10)

AS

SELECT s.FirstName, s.LastName, c.CourseName

FROM Student s

JOIN Study st ON s.StudentId = st.StudentId

JOIN Course c ON st.CourseId = c.CourseId

WHERE s.StudentId = @studentID;