FINAL REPORT

Name : Vũ Lê Băng Tâm
Roll number : HE160536
Course : DBI202

Instructor : Ngô Tùng Sơn

I. Database Description

A. *Problem Introduction*

- You are asked to provide a conceptual database design for Attendance Taking & Student Grade Management System at FPT University
- Every semester, Lecturers and students will receive their schedules for classes
- Each class, the lecture going to take attendance for student, all of their absent/present will be recorded
- For each subject that attended by the student, the lecture will give score to the assessment to each of their assessment
- Each Subject code, student can check their detailed result

B. *Entity & Relationship*

1. *Entity*

- Student: **StudentID**, SurName, MiddleName, GivenName, Email, Gender, Dob
- Instructor: **InstructorID**, FirstName, LastName, Email
- Course: **CourseID**, CourseName
- Time_Slot: **TimeSlotID,** StartTime, EndTime
- Group: **CourseID, GroupName**, InstructorID
  (Forgein Key:
  - CourseID from Course: each group can study many subject
  - InstructorID from Instructor: one group can only be instructed by one instructor)
- Group_Students: **CourseID, GroupName, StudentID**
  (Forgein Key:
  - CourseID and GroupName from Group: each student can be a part of many group of different subject
  - StudentID from Student)
- Lecture: **LectureID**, GroupName, CourseID, LectureName, InstructorID, TimeSlotID, Room
  (Forgein Key:
  - CourseID and GroupName from Group: the same lecture can be taught in different group with the same subject
  - InstructorID from Instructor: the instructor who (will) teach this lecture
  - TimeSlotID from Time_Slot: time when the lecture happens)
- Take_Attendance: **LectureID, InstructorID, StudentID**, Status, RecordTime
  (Forgein Key:
  - LectureID from Lecture

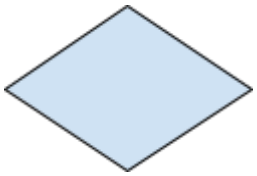
    - InstructorID from Instructor: the instructor who actually teaches the lecture (There are some cases which the instructor is changed due to circumstances)
    - StudentID from Student: student who attend the session)
- Assessment: **CourseID, Category**, Weight, CompletionCriteria
(Forgein Key:
    - CourseID from Course: which assessment belongs to which course)
- Grading: **CourseID, Category, StudentID**, Grade
(Forgein Key:
    - CourseID and Category from Assessment: grade base on which course and which assessment in that course
    - StudentID from Student)

2. *Relationship*
    - One student can belongs to many group of different subject
    - One student can only attend 1 lecture at a time
    - One instructor can instruct many group of different subject
    - One instructor can only teaches 1 lecture at a time
    - There can be a change in who actually teach the lecture due to some circumstances
    - Each course may have it owns grade assessment (Some course can have the same category name but as long as it belongs to different course)
    - Student can only have 1 record of grade belongs to each category of the course

II. ERD

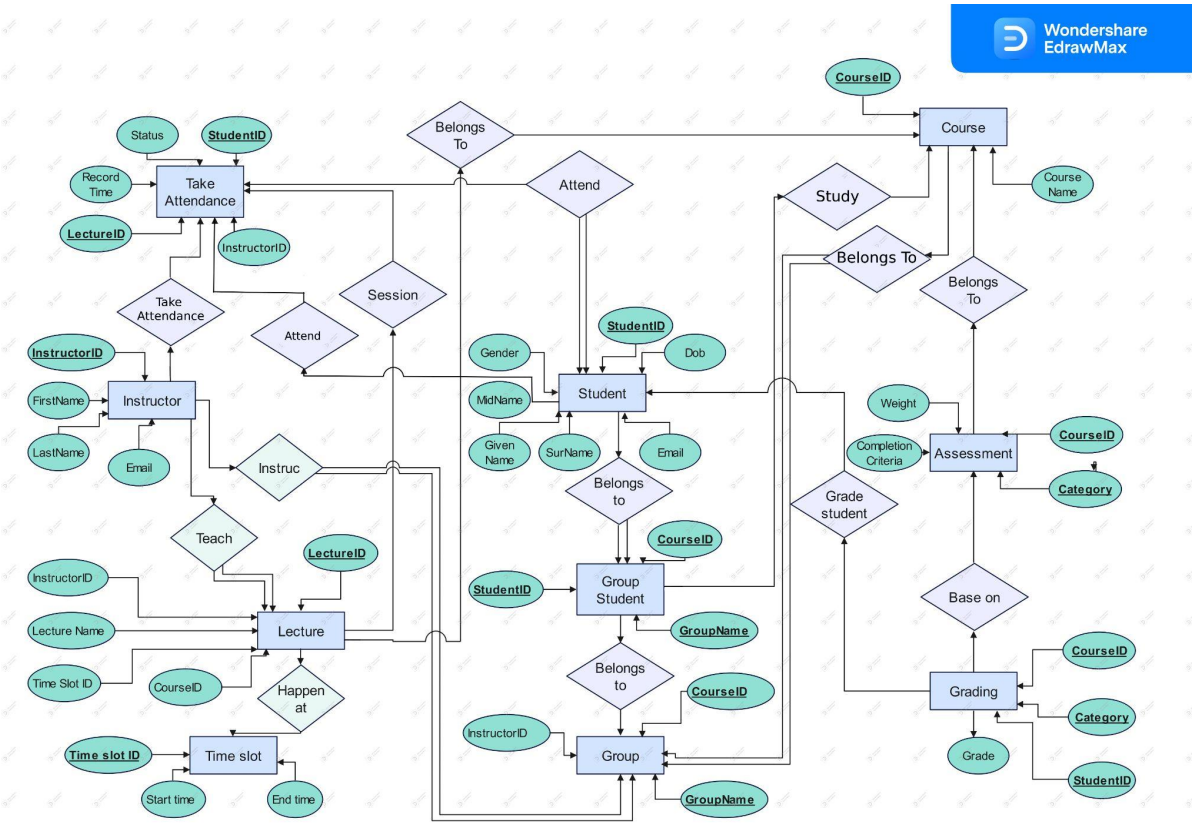
1. *ERD (Chen notation)*


: Entity

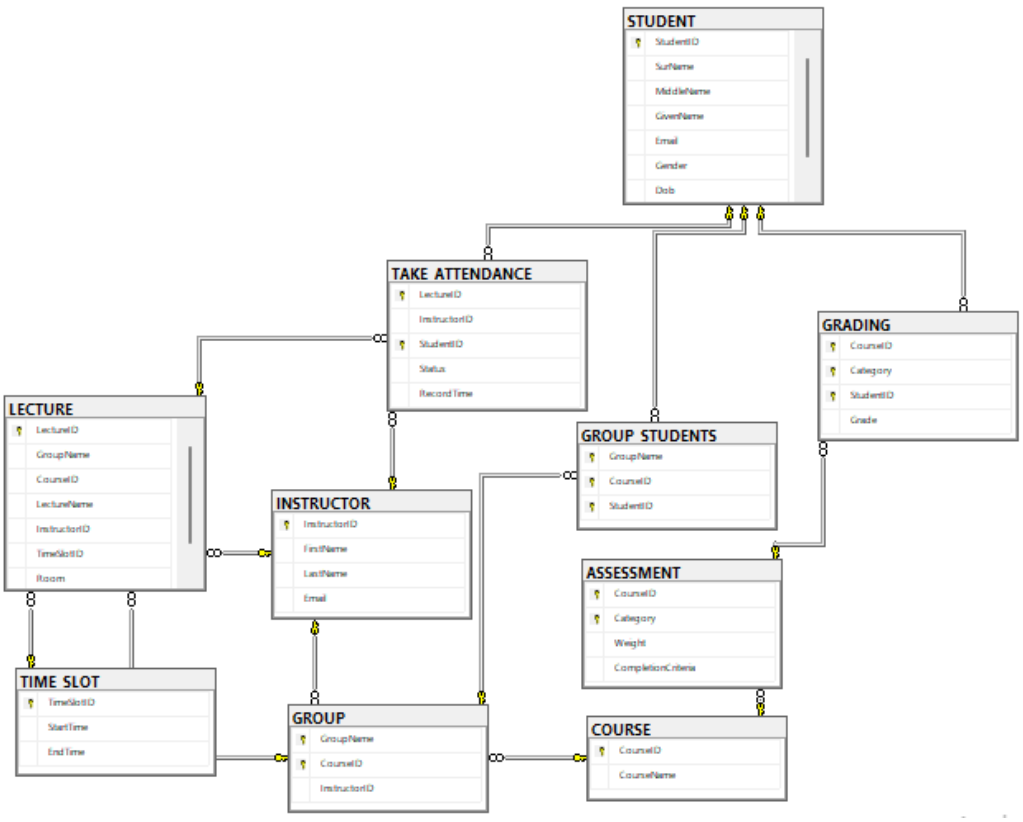

:Relationship


:Attribute

:Connectivity


Wondershare EdrawMax
CourseID
Course
Status
StudentID
Belongs To
Record Time
Take Attendance
Attend
Study
Course Name
LectureID
InstructorID
Belongs To
Belongs To
Take Attendance
Session
Attend
InstructorID
Gender
StudentID
Dob
FirstName
Instructor
MidName
Student
Weight
LastName
Email
Instruc
Given Name
SurName
Email
Completion Criteria
Assessment
CourseID
Belongs to
Grade student
Category
Teach
CourseID
LectureID
InstructorID
StudentID
Group Student
Base on
Lecture Name
Lecture
GroupName
CourseID
Time Slot ID
CourseID
Happen at
Belongs to
CourseID
Grading
Category
InstructorID
Group
Time slot ID
Time slot
Grade
StudentID
Start time
End time
GroupName

2. *Diagram*


STUDENT
StudentID
SurName
MiddleName
GivenName
Email
Gender
Dob
TAKE ATTENDANCE
LectureID
InstructorID
StudentID
Status
RecordTime
GRADING
CourseID
Category
StudentID
Grade
LECTURE
LectureID
GroupName
CourseID
LectureName
InstructorID
TimeSlotID
Room
GROUP STUDENTS
GroupName
CourseID
StudentID
INSTRUCTOR
InstructorID
FirstName
LastName
Email
ASSESSMENT
CourseID
Category
Weight
CompletionCriteria
TIME SLOT
TimeSlotID
StartTime
EndTime
GROUP
GroupName
CourseID
InstructorID
COURSE
CourseID
CourseName

III. Tables and Data

1. *Student table*

A. *Code*

```
---CREATE STUDENT TABLE
CREATE TABLE STUDENT(
	StudentID VARCHAR(8) PRIMARY KEY NOT NULL CHECK (StudentID LIKE 'HE[0-9][0-9][0-9][0-9][0-9][0-9]'),
	SurName VARCHAR(10) NOT NULL,
	MiddleName VARCHAR(10) NOT NULL,
	GivenName VARCHAR(10) NOT NULL,
	Email VARCHAR(50) NOT NULL CHECK (Email LIKE '%@fpt.edu.vn'),
	Gender BIT NOT NULL,
	Dob DATE NOT NULL CHECK (YEAR(GETDATE()) - YEAR(dob) >= 18),
)
```

B. *Example*

	StudentID	SurName	MiddleName	GivenName	Email	Gender	Dob
1	HE102623	Johanna	Hand	Lacie	Fae@fpt.edu.vn	1	2001-02-16
2	HE120192	Flo	Turner	Mune	Charlotte@fpt.edu.vn	0	2002-11-29
3	HE138474	Miller	Halvorson	Nocturne	Maryjane.Gutkowski@fpt.edu.vn	0	2000-12-09
4	HE143202	Damin	Kutcht	Sheng	Oda@fpt.edu.vn	0	2000-12-01
5	HE147255	Tiffany	Schumm	Bert	Lewis_Doyle@fpt.edu.vn	0	2002-09-03
6	HE150192	Rae	McLaughlin	Lilla	Maximillia.Grant@fpt.edu.vn	1	2002-05-25
7	HE151234	Assunta	Gottlieb	Lao	Domenico_Stehr@fpt.edu.vn	1	2001-07-26
8	HE155022	Presley	Sporer	Kayce	Jalon@fpt.edu.vn	1	2003-12-12
9	HE156788	Jarrell	Dibbert	Lin	Gavin@fpt.edu.vn	0	2002-08-11
10	HE160098	Leora	Collier	Oin	Kayley@fpt.edu.vn	1	2001-11-23
11	HE160666	Green	Swaniawski	Laz	Alexys_Legros@fpt.edu.vn	1	2000-07-15
12	HE161039	Keeley	DAmore	Jane	Russel.Waelchi@fpt.edu.vn	0	2001-12-10
13	HE167821	Zoey	Corwin	Alexa	Mina@fpt.edu.vn	0	2000-02-18

2. *Instructor table*

A. *Code*

```
---CREATE TABLE INSTRUCTOR
CREATE TABLE INSTRUCTOR(
	InstructorID VARCHAR(20) PRIMARY KEY NOT NULL,
	FirstName VARCHAR(15) NOT NULL,
	LastName VARCHAR(15) NOT NULL,
	Email VARCHAR(50) NOT NULL CHECK (Email LIKE '%@fe.edu.vn')
)
```

B. *Example*

	InstructorID	FirstName	LastName	Email
1	ChiLP	Chi	Le	chilp@fe.edu.vn
2	SonNT5	Son	Ngo	sonnt69@fe.edu.vn
3	TuNT57	Tu	Nguyen	TuNT57@fe.edu.vn

3. *Course table*

A. *Code*

```sql
---CREATE TABLE COURSE
CREATE TABLE COURSE(
	CourseID VARCHAR(6) PRIMARY KEY NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
	CourseName VARCHAR(50) NOT NULL,
)
```

B. Sample

	CourseID	CourseName
1	CSD201	Data Structures and Algorithms
2	DBI202	Introduction to Databases

4. Group table

A. Code

```sql
---CREATE TABLE GROUP
CREATE TABLE [GROUP](
	GroupName VARCHAR(6) NOT NULL,
	CourseID VARCHAR(6) NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
	InstructorID VARCHAR(20) NOT NULL,
	CONSTRAINT fk_Group_Course FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID),
	CONSTRAINT fk_Group_Instructor FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID),
	CONSTRAINT pk_Group PRIMARY KEY(GroupName,CourseID)
)
```

B. Sample

	GroupName	CourseID	InstructorID
1	AI1603	DBI202	ChiLP
2	AI1604	CSD201	TuNT57
3	AI1604	DBI202	SonNT5

5. Group_Student table

A. Code

```sql
---CREATE TABLE GROUP_MEMBER (GROUP - STUDENT)
CREATE TABLE GROUP_STUDENTS(
	GroupName VARCHAR(6) NOT NULL,
	CourseID VARCHAR(6) NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
	StudentID VARCHAR(8) NOT NULL CHECK (StudentID LIKE 'HE[0-9][0-9][0-9][0-9][0-9][0-9]'),
	CONSTRAINT fk_GroupStudent_Student FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
	CONSTRAINT fk_GroupStudent_Group FOREIGN KEY (GroupName,CourseID) REFERENCES [GROUP](GroupName,CourseID),
	CONSTRAINT pk_GroupStudents PRIMARY KEY(GroupName,CourseID,StudentID)
)
```

B. Sample

	GroupName	CourseID	StudentID
1	AI1603	DBI202	HE120192
2	AI1603	DBI202	HE138474
3	AI1603	DBI202	HE155022
4	AI1603	DBI202	HE160666
5	AI1603	DBI202	HE170101
6	AI1604	CSD201	HE102623
7	AI1604	CSD201	HE120192
8	AI1604	CSD201	HE143202
9	AI1604	CSD201	HE150192
10	AI1604	CSD201	HE151234
11	AI1604	CSD201	HE156788
12	AI1604	CSD201	HE160098

6. *TimeSlot table*

A. *Code*

```
---CREATE TABLE TIME_SLOT
CREATE TABLE TIME_SLOT(
      TimeSlotID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
      StartTime TIME NOT NULL,
      EndTime TIME NOT NULL,
)
```

B. *Sample*

	TimeSlotID	StartTime	EndTime
1	1	07:30:00.0000000	09:00:00.0000000
2	2	09:10:00.0000000	10:40:00.0000000
3	3	10:50:00.0000000	12:20:00.0000000
4	4	12:50:00.0000000	14:20:00.0000000
5	5	14:30:00.0000000	16:00:00.0000000
6	6	16:10:00.0000000	17:40:00.0000000

7. *Lecture table*

A. *Code*

```
---CREATE TABLE LECTURE
CREATE TABLE LECTURE(
      LectureID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
      GroupName VARCHAR(6) NOT NULL,
      CourseID VARCHAR(6) NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
      LectureName VARCHAR(50) NOT NULL,
      InstructorID VARCHAR(20) NOT NULL,
      TimeSlotID INT NOT NULL,
      Room VARCHAR(5) NOT NULL,
      CONSTRAINT fk_Lecture_Group FOREIGN KEY (GroupName,CourseID) REFERENCES [GROUP](GroupName,CourseID),
      CONSTRAINT fk_Lecture_Instructor FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID),
      CONSTRAINT fk_Lecture_TimeSlot FOREIGN KEY (TimeSlotID) REFERENCES TIME_SLOT(TimeSlotID)
)
```

B. *Sample*

	LectureID	GroupName	CourseID	LectureName	InstructorID	TimeSlotID	Room
1	1	AI1604	DBI202	The Worlds of Database Systems	SonNT5	2	B308
2	2	AI1604	CSD201	Course Introduction	TuNT57	1	A110
3	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207
4	4	AI1603	DBI202	The Relational Model of Data	SonNT5	3	A301
5	5	AI1604	CSD201	Using Arrays	TuNT57	4	B308
6	6	AI1603	DBI202	Design Theory for Relational Databases	SonNT5	5	A301

C. Trigger

```
---CREATE TRIGGER FOR LECTURE
CREATE TRIGGER trigger_Lecture ON LECTURE AFTER INSERT AS
     DECLARE @GroupName VARCHAR(20)
     DECLARE @CourseID VARCHAR(20)
     DECLARE @InstructorID VARCHAR(20)
     DECLARE @InsID2 VARCHAR(20)
     SELECT @GroupName=GroupName FROM inserted
     SELECT @CourseID=CourseID FROM inserted
     SELECT @InstructorID=InstructorID FROM inserted
     SELECT @InsID2=InstructorID FROM [GROUP] WHERE CourseID=@CourseID AND GroupName=@GroupName
     IF @InstructorID <> @InsID2
      BEGIN
       PRINT @InstructorID+' can not teachs this lecture'
       ROLLBACK TRANSACTION
      END
```

Trigger trigger_Lecture is used to check if the instructor can instruct this lecture or not

- Ex: This instructor does not teach course CSD201 belongs to Group AI1604

INSERT INTO LECTURE(GroupName,CourseID,LectureName,InstructorID,TimeSlotID, Room) VALUES ('AI1604','CSD201','The Worlds of Database Systems','SonNT5',2,'B308')

- Message:

```
SonNT5 can not teachs this lecture
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-03-15T11:30:13.4990825+07:00
```

8. Take_Attendance table

A. Code

```
 ---CREATE TABLE TAKE_ATTENDANCE
)CREATE TABLE TAKE_ATTENDANCE(
     LectureID INT NOT NULL,
     InstructorID VARCHAR(20) NOT NULL,
     StudentID VARCHAR(8) NOT NULL CHECK (StudentID LIKE 'HE[0-9][0-9][0-9][0-9][0-9][0-9]'),
     [Status] BIT NOT NULL,
     RecordTime TIME NOT NULL,
     CONSTRAINT fk_TakeAttendance_Instructor FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID),
     CONSTRAINT fk_TakeAttendance_Student FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
     CONSTRAINT fk_TakeAttendance_Lecture FOREIGN KEY (LectureID) REFERENCES LECTURE(LectureID),
     CONSTRAINT pk_TakeAttendance PRIMARY KEY(LectureID,StudentID)
)
```

B. Sample

	LectureID	InstructorID	StudentID	Status	RecordTime
1	1	SonNT5	HE102623	1	09:10:00.0000000
2	1	SonNT5	HE150192	1	09:11:00.0000000
3	1	SonNT5	HE156788	1	09:12:00.0000000
4	1	SonNT5	HE161039	1	10:10:00.0000000
5	1	SonNT5	HE167821	0	10:11:00.0000000
6	1	SonNT5	HE174829	0	11:10:00.0000000
7	2	TuNT57	HE102623	0	07:41:00.0000000
8	2	TuNT57	HE120192	1	07:40:00.0000000
9	2	TuNT57	HE143202	1	07:41:00.0000000
10	2	TuNT57	HE150192	0	07:42:00.0000000
11	2	TuNT57	HE151234	1	07:43:00.0000000
12	2	TuNT57	HE156788	0	07:41:00.0000000
13	2	TuNT57	HE160098	1	07:40:00.0000000

C. Trigger

```
---CREATE TRIGGER FOR TAKE ATTENDANCE
CREATE TRIGGER trigger_Attendance ON TAKE_ATTENDANCE AFTER INSERT AS
      DECLARE @LecID INT
      SELECT @LecID=LectureID FROM inserted

      ---Student
      DECLARE @StuID VARCHAR(10)
      SELECT @StuID=StudentID FROM inserted
      IF @StuID NOT IN (SELECT g.StudentID FROM LECTURE l INNER JOIN GROUP_STUDENTS g ON l.GroupName=g.GroupName
      AND l.CourseID=g.CourseID WHERE l.LectureID=@LecID)
        BEGIN
          PRINT @StuID+' does not in this Group'
          ROLLBACK TRANSACTION
        END

      ---Instructor
      DECLARE @InsID VARCHAR(10)
      SELECT @InsID=InstructorID FROM inserted
      IF @InsID <> (SELECT InstructorID FROM LECTURE WHERE LectureID=@LecID)
        BEGIN
          PRINT @InsID+' alternative teach this lecture'
        END

      ---Record time
      DECLARE @RecordTime TIME
      DECLARE @Start TIME
      SELECT @RecordTime=RecordTime FROM inserted
      SELECT @Start=t.StartTime FROM [LECTURE] l INNER JOIN TIME_SLOT t ON LectureID=@LecID AND l.TimeSlotID=t.TimeSlotID
      IF @RecordTime < @Start
        BEGIN
          PRINT 'Record Time('+CAST(@RecordTime AS VARCHAR(50))+') must not before '+CAST(@Start AS VARCHAR(50))
          ROLLBACK TRANSACTION
        END
```

Trigger trigger_Attendance is used to :

- check if a student can attend the lecture
  - + Ex: This student with roll number HE170101 does not attend course DBI202 of Instructor SonNT5
    INSERT INTO TAKE_ATTENDANCE(LectureID,InstructorID,StudentID,[Status],RecordTime) VALUES (1,'SonNT5','HE170101',1,'9:10:00')
  - + Message:

```
HE170101 does not in this Group
Msg 3609, Level 16, State 1, Line 3
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-03-15T14:55:05.9913834+07:00
```

- check if there is a change of instructor
  + Ex: Instructor SonNT5 alternative teach the lecture. Instructor ChiLP suppose to teach the lecture
    INSERT INTO TAKE_ATTENDANCE(LectureID,InstructorID,StudentID,[Status],RecordTime) VALUES (3,'SonNT5','HE138474',1,'9:40:00')
  + Message:

```
SonNT5 alternative teach this lecture

(1 row affected)

Completion time: 2022-03-15T15:06:42.6315256+07:00
```

- check if the record time is reasonable
  + Ex: The lecture has the TimeSlotID 1 (starts at 7:30 & ends at 9:00), but the attendance is taken at 6:10 before the lecture start
    INSERT INTO TAKE_ATTENDANCE(LectureID,InstructorID,StudentID,[Status],RecordTime) VALUES (1,'SonNT5','HE151234',1,'6:10:00')
  + Message:

```
Record Time(06:10:00.0000000) must not before 09:10:00.0000000
Msg 3609, Level 16, State 1, Line 7
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-03-15T15:15:14.0232379+07:00
```

9. *Assessment table*

A. *Code*

```
---CREATE TABLE COURSE_ASSESSMENT
CREATE TABLE ASSESSMENT(
     CourseID VARCHAR(6) NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
     Category VARCHAR(20) NOT NULL,
     [Weight] INT NOT NULL,
     CompletionCriteria INT NOT NULL,
     CONSTRAINT fk_Assessment_Course FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID),
     CONSTRAINT pk_Assessment PRIMARY KEY(CourseID,Category)
)
```

B. *Sample*

	CourseID	Category	Weight	CompletionCriteria
1	CSD201	Assignment 1	10	0
2	CSD201	Assignment 2	10	0
3	CSD201	Final Exam	30	4
4	CSD201	Practical Exam	30	0
5	CSD201	Progress Test 1	10	0
6	CSD201	Progress Test 2	10	0
7	DBI202	Assignment	20	0
8	DBI202	Final Exam	30	4
9	DBI202	Lab1	2	0
10	DBI202	Lab2	2	0
11	DBI202	Lab3	2	0
12	DBI202	Lab4	2	0
13	DBI202	Lab5	2	0
14	DBI202	Practical Exam	30	0

C. *Trigger*

```
---CREATE TRIGGER FOR ASSESSMENT
CREATE TRIGGER trigger_Assessment ON ASSESSMENT AFTER INSERT AS
     DECLARE @CourseID VARCHAR(10)
     DECLARE @Weight INT
     DECLARE @SumWeight INT
     SELECT @CourseID=CourseID FROM inserted
     SELECT @Weight=[Weight] FROM inserted
     SELECT @SumWeight = (SELECT SUM([Weight]) FROM ASSESSMENT WHERE CourseID=@CourseID)
     IF @Weight > (100-@SumWeight+@Weight)
       BEGIN
         PRINT 'This category weight can not exceed '+CAST((100-@SumWeight+@Weight) AS VARCHAR(10))+'%'
         ROLLBACK TRANSACTION
       END
```

Trigger trigger_Assessment is used to control the weight of each category of one course. The total weight of category of a course can not exceed 100%

+ Ex: Final Exam of course DBI202 can only have weight equal to 30%
  INSERT INTO ASSESSMENT(CourseID,Category,[Weight],CompletionCriteria) VALUES ('DBI202','Final Exam',40,4)
+ Message:

```
This category weight can not exceed 30%
Msg 3609, Level 16, State 1, Line 9
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-03-15T15:25:26.3390925+07:00
```

10. Grading table

A. *Code*

```sql
---CREATE TABLE GRADING
CREATE TABLE GRADING(
     CourseID VARCHAR(6) NOT NULL CHECK(CourseID LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]'),
     Category VARCHAR(20) NOT NULL,
     StudentID VARCHAR(8) NOT NULL CHECK (StudentID LIKE 'HE[0-9][0-9][0-9][0-9][0-9][0-9]'),
     Grade FLOAT NOT NULL,
     CONSTRAINT fk_Grading_Student FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
     CONSTRAINT fk_Grading_Assessment FOREIGN KEY (CourseID,Category) REFERENCES ASSESSMENT(CourseID,Category),
     CONSTRAINT pk_Grading PRIMARY KEY(CourseID,Category,StudentID)
)
```

B. Sample

	CourseID	Category	StudentID	Grade
1	CSD201	Assignment 1	HE102623	7
2	CSD201	Assignment 1	HE160098	5
3	CSD201	Assignment 1	HE174829	3
4	CSD201	Assignment 2	HE102623	8.5
5	CSD201	Assignment 2	HE160098	8
6	CSD201	Assignment 2	HE174829	4
7	CSD201	Final Exam	HE102623	6.5
8	CSD201	Final Exam	HE174829	5
9	CSD201	Practical E...	HE102623	8
10	CSD201	Practical E...	HE174829	2
11	CSD201	Progress T...	HE102623	7
12	CSD201	Progress T...	HE160098	6.5
13	CSD201	Progress T...	HE174829	4.5
14	CSD201	Progress T...	HE102623	9
15	CSD201	Progress T...	HE160098	6

C. View

Since FPT has many courses so I decided to create a sample of 2 views for 2 courses DBI202 and CSD201.

```sql
---CREATE VIEW MARK REPORT FOR DBI202
CREATE VIEW MARK_REPORT_DBI202 AS
     SELECT StudentID,Category,Grade FROM GRADING
     WHERE CourseID='DBI202'

---CREATE VIEW MARK REPORT FRO CSD201
CREATE VIEW MARK_REPORT_CSD201 AS
     SELECT StudentID,Category,Grade FROM GRADING
     WHERE CourseID='CSD201'
```

Sample:

- Mark report for DBI202:

	StudentID	Category	Grade
1	HE120192	Assignment	9
2	HE138474	Assignment	6.5
3	HE155022	Assignment	6.5
4	HE120192	Final Exam	6
5	HE155022	Final Exam	3.8
6	HE120192	Lab1	7.5
7	HE138474	Lab1	4
8	HE155022	Lab1	6
9	HE120192	Lab2	8.5
10	HE138474	Lab2	5
11	HE155022	Lab2	6.5
12	HE120192	Lab3	8

- Mark report for CSD201:

	StudentID	Category	Grade
1	HE102623	Assignment 1	7
2	HE160098	Assignment 1	5
3	HE174829	Assignment 1	3
4	HE102623	Assignment 2	8.5
5	HE160098	Assignment 2	8
6	HE174829	Assignment 2	4
7	HE102623	Final Exam	6.5
8	HE174829	Final Exam	5
9	HE102623	Practical Exam	8
10	HE174829	Practical Exam	2
11	HE102623	Progress Test 1	7
12	HE160098	Progress Test 1	6.5
13	HE174829	Progress Test 1	4.5

D. Trigger

```
---CREATE TRIGGER FOR GRADING
CREATE TRIGGER trigger_Grading ON GRADING AFTER INSERT AS
      DECLARE @StuID VARCHAR(10)
      DECLARE @CourseID VARCHAR(10)
      SELECT @StuID=StudentID FROM inserted
      SELECT @CourseID=CourseID FROM inserted
      IF @CourseID NOT IN (SELECT CourseID FROM GROUP_STUDENTS WHERE StudentID=@StuID)
        BEGIN
          PRINT @StuID+' does not enroll course name '+@CourseID
          ROLLBACK TRANSACTION
        END
```

Trigger trigger_Grading is used to check a student study/enroll in a course or not

+ Ex: The student with roll number HE161039 does not enroll in course CSD201
  INSERT INTO GRADING (CourseID,Category,StudentID,Grade)
  VALUES ('CSD201','Assignment 2','HE161039',9)

+ Message:

```
HE161039 does not enroll course name CSD201
Msg 3609, Level 16, State 1, Line 11
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2022-03-15T15:52:06.1786434+07:00
```

E. Proc

```
CREATE PROC spGrade
    @StudentID VARCHAR(10),
    @CourseID VARCHAR(10),
    @Average FLOAT OUT
AS
BEGIN
    DECLARE @avg FLOAT = 0
    DECLARE @countFromGrade INT
    DECLARE @countFromAssessment INT
    SET @countFromGrade = (SELECT COUNT(*) FROM GRADING WHERE StudentID=@StudentID AND CourseID=@CourseID)
    SET @countFromAssessment = ( SELECT COUNT(*) FROM ASSESSMENT WHERE CourseID=@CourseID)
    IF @countFromGrade != @countFromAssessment
      BEGIN
        PRINT 'Status: NOT (YET) PASS'
      END
    ELSE
      BEGIN
        DECLARE @Cate VARCHAR(50)
        DECLARE @Grade FLOAT
        DECLARE grade_cursor CURSOR FOR
        SELECT Category,Grade FROM GRADING WHERE StudentID=@StudentID AND CourseID=@CourseID
        OPEN grade_cursor
        FETCH NEXT FROM grade_cursor INTO @Cate,@Grade
        WHILE @@FETCH_STATUS = 0
         BEGIN
           IF @Grade < (SELECT CompletionCriteria FROM ASSESSMENT WHERE CourseID=@CourseID AND Category=@Cate)
             BEGIN
               PRINT 'Status: '+@StudentID+' fail course '+@CourseID
             END
           ELSE
             BEGIN
               SET @avg= @avg + @Grade*(SELECT [Weight] FROM ASSESSMENT WHERE CourseID=@CourseID AND Category=@Cate)/100
             END
           FETCH NEXT FROM grade_cursor INTO @Cate,@Grade
         END
        IF @avg < 5
          BEGIN
            PRINT 'Status: '+@StudentID+' fail course '+@CourseID
          END
        ELSE
          BEGIN
            PRINT 'Status: PASS'
          END
        CLOSE grade_cursor
        DEALLOCATE grade_cursor
      END
      SET @Average = @avg
END
```

Stored procedure spGrade is used to calculate the average and final result of a course of a student

- Input :
  + StudentID
  + CourseID
- Output :
  + Average grade
  + Status (PASS/FAILED/NOT (YET) PASS)
- Ex:

```
DECLARE @StudentID VARCHAR(10)
DECLARE @CourseID VARCHAR(10)
DECLARE @Average FLOAT
EXEC spGrade @StudentID='HE120192',@CourseID='DBI202',@Average=@Average OUT
SELECT @Average AS [AVERAGE]
```

- Message:

```
Status: PASS

(1 row affected)

Completion time: 2022-03-15T16:03:08.0319993+07:00
```

	AVERAGE
1	7.485

F. Index

```
---CREATE INDEX FOR GRADING
CREATE INDEX index_StudentGrade ON GRADING (StudentID,CourseID)
```

This index is used to receive mark report of a student more faster

IV. Use of database

1. Using ORDER BY to order schedule of lecture

- Query:
  SELECT * FROM LECTURE ORDER BY TimeSlotID
- Result:

	LectureID	GroupName	CourseID	LectureName	InstructorID	TimeSlotID	Room
1	2	AI1604	CSD201	Course Introduction	TuNT57	1	A110
2	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207
3	1	AI1604	DBI202	The Worlds of Database Systems	SonNT5	2	B308
4	4	AI1603	DBI202	The Relational Model of Data	SonNT5	3	A301
5	5	AI1604	CSD201	Using Arrays	TuNT57	4	B308
6	6	AI1603	DBI202	Design Theory for Relational Databases	SonNT5	5	A301

2. Using INNER JOIN to see attendances of group AI1603 that was teach by the same instructor

- Query:
  SELECT * FROM LECTURE l INNER JOIN TAKE_ATTENDANCE t ON l.LectureID=t.LectureID AND l.InstructorID=t.InstructorID AND l.GroupName='AI1603'
- Result:

	LectureID	GroupName	CourseID	LectureName	InstructorID	TimeSlotID	Room	LectureID	InstructorID	StudentID	Status	RecordTime
1	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207	3	ChiLP	HE120192	1	09:40:00.0000000
2	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207	3	ChiLP	HE155022	0	09:41:00.0000000
3	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207	3	ChiLP	HE160666	1	09:42:00.0000000
4	3	AI1603	DBI202	The Worlds of Database Systems	ChiLP	2	B207	3	ChiLP	HE170101	1	09:43:00.0000000
5	6	AI1603	DBI202	Design Theory for Relational Databases	ChiLP	5	A301	6	SonNT5	HE120192	1	14:35:00.0000000
6	6	AI1603	DBI202	Design Theory for Relational Databases	ChiLP	5	A301	6	SonNT5	HE138474	1	14:35:00.0000000
7	6	AI1603	DBI202	Design Theory for Relational Databases	ChiLP	5	A301	6	SonNT5	HE155022	1	14:36:00.0000000
8	6	AI1603	DBI202	Design Theory for Relational Databases	ChiLP	5	A301	6	SonNT5	HE160666	0	14:36:00.0000000
9	6	AI1603	DBI202	Design Theory for Relational Databases	ChiLP	5	A301	6	SonNT5	HE170101	1	14:37:00.0000000

3. Using aggregate functions to see the number of student attend lecture

- Query:
  SELECT LectureID, COUNT(*) AS [NumberOfAttendStudent] FROM TAKE_ATTENDANCE WHERE [Status]=1 GROUP BY LectureID
- Result:

	LectureID	NumberOfAttendStudent
1	1	4
2	2	6
3	3	3
4	4	4
5	5	7
6	6	4

4. Using GROUP BY and HAVING to query StudentID, CourseID and Number of lecture attended by the student who have attend the course at least 2 times
   - Query:
     SELECT l.CourseID,t.StudentID,COUNT(*) AS [NumberOfAttend] FROM TAKE_ATTENDANCE t INNER JOIN LECTURE l ON t.LectureID=l.LectureID
     WHERE t.[Status]=1
     GROUP BY l.CourseID, t.StudentID HAVING COUNT(*)>=2
   - Result:

	CourseID	StudentID	NumberOfAttend
1	CSD201	HE120192	2
2	DBI202	HE120192	3
3	DBI202	HE138474	2
4	CSD201	HE143202	2
5	CSD201	HE151234	2
6	CSD201	HE160098	2
7	CSD201	HE160666	2
8	DBI202	HE160666	2
9	DBI202	HE170101	3

5. Using sub-query as a relation to query CourseID, StudentID and Grade of student who has the highest grade in Final Exam in term of all Course
   - Query:
     SELECT TOP 1 * FROM
     (SELECT CourseID,StudentID,MAX(Grade) AS [Highest grade] FROM GRADING WHERE Category='Final Exam' GROUP BY CourseID,StudentID)tbl
   - Result:

	CourseID	StudentID	Highest grade
1	CSD201	HE102623	6.5

6. Using sub-query in WHERE clause to query student who enrol different group of different course
   - Query:
     SELECT * FROM
     (SELECT StudentID,COUNT(GroupName) AS [NumberOfGroup] FROM GROUP_STUDENTS GROUP BY StudentID) tbl
     WHERE tbl.NumberOfGroup>1
   - Result:

	StudentID	NumberOfGroup
1	HE102623	2
2	HE120192	2
3	HE150192	2
4	HE151234	2
5	HE156788	2
6	HE160666	2
7	HE170101	2
8	HE174829	2

7. Using partial matching in the WHERE clause to query students who have roll number starts with HE16....
   - Query:
     SELECT * FROM STUDENT WHERE StudentID LIKE 'HE16%'
   - Result:

	StudentID	SurName	MiddleName	GivenName	Email	Gender	Dob
1	HE160098	Leora	Collier	Oin	Kayley@fpt.edu.vn	1	2001-11-23
2	HE160666	Green	Swaniawski	Laz	Alexys_Legros@fpt.edu.vn	1	2000-07-15
3	HE161039	Keeley	DAmore	Jane	Russel.Waelchi@fpt.edu.vn	0	2001-12-10
4	HE167821	Zoey	Corwin	Alexa	Mina@fpt.edu.vn	0	2000-02-18

8. Using SELF-JOIN to query student information that NOT attend both course CSD201 and DBI202
   - Query:
     SELECT *FROM STUDENT WHERE StudentID NOT IN(
     SELECT g1.StudentID FROM GROUP_STUDENTS g1 INNER JOIN GROUP_STUDENTS g2 ON g1.StudentID=g2.StudentID AND g1.CourseID = g2.CourseID)
   - Result:

	StudentID	SurName	MiddleName	GivenName	Email	Gender	Dob
1	HE147255	Tiffany	Schumm	Bert	Lewis_Doyle@fpt.edu.vn	0	2002-09-03