# DBI202 Project

Pham Thi Minh Anh – SE04957

Department of Software Engineering

FPT University

Major Professor: Ngo Tung Son

In partial fulfillment of the requirements for the DBI202

# I. Introduction

## Background

This is a project to design and develop a database system for a university entrance system. The system must base on some following requirements: Applicants may apply to five different universities. Each university may or may not interview the applicant and then may make an offer to the applicant. The offer may be conditional or unconditional. If the offer is conditional, then the conditions are stored and communicated to the student. The applicant needs to decide which of the conditional offers he or she wishes to accept, up to a maximum of three. If the condition is met when the results come out at the end of the year, the offer becomes unconditional and applicants then may accept one of the unconditional offers.

In the condition of each university which makes the conditional offer, they can recruit only on 4 criterias: SAT score, IELTS score, TOELF score and Recommendation requirement. Universities can require one or more in these 4 criterias. Note: IELTS degree can replace for TOELF degree and and on the contrary in some university offer but some Universities can require both of them.

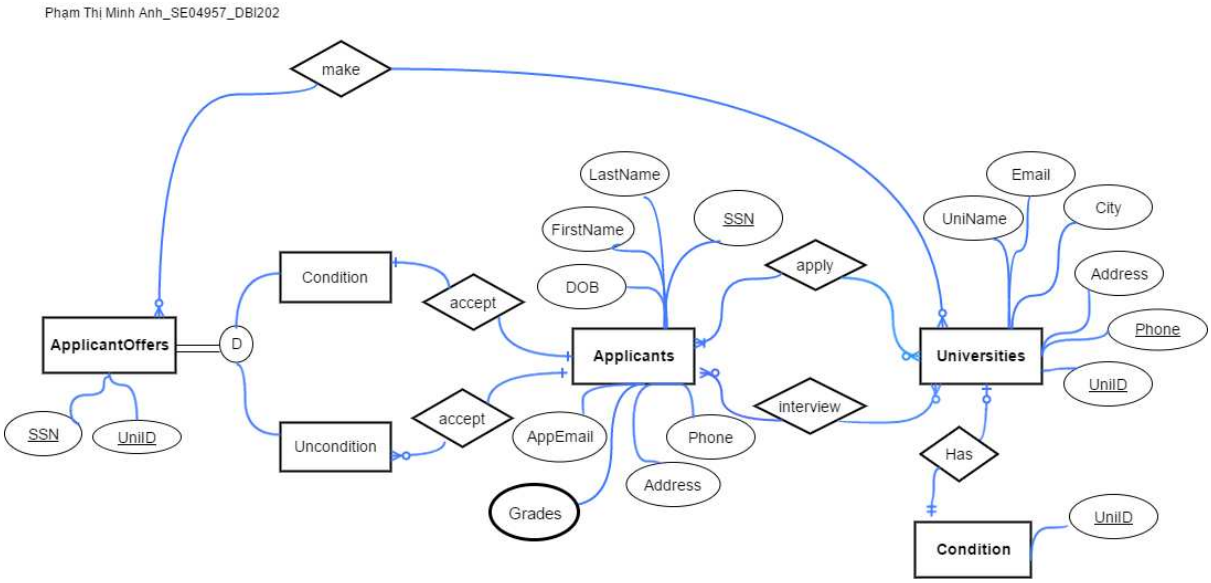
## Tools

The system  uses to store data and calidate data with no required extensions, such as java, javascript, or  html,….

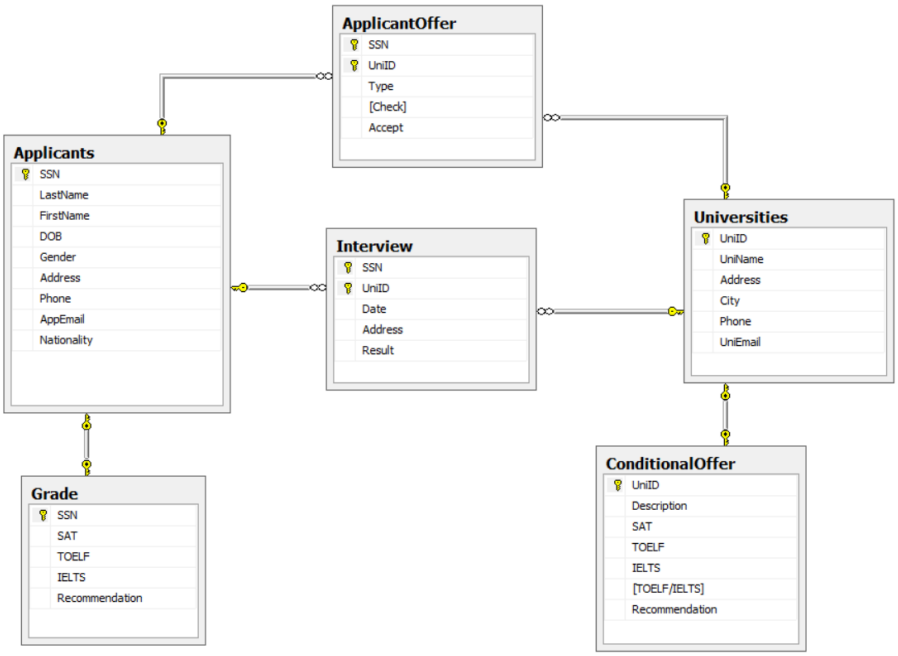
- SQL Server is used to design and create a set of database tables on .
- https://www.mockaroo.com/ is used to generating data.
- https://www.draw.io/ to create Entity Relationship (ER) model.

## II. ERD - Entity Relationship Diagram

- Use Chen's Notation to create the following ERD




- The following image show the relation between the set of tables of database.



## III. Relational Schema

Universities (UniID, UniName, Address, City, Phone, UniEmail)

Applicants (SSN, LastName, FirstName, DOB, Gender, Address, Phone, Email, Nationality)

ApplycantOffer (SSN, UniID)

Interview (SSN, UniID , Date, Address, Result)

ConditionalOffer (UniID, Description, SAT, TOELF, IELTS, TOELF/IELTS, Recommendation)

Grade (SSN, SAT, TOELF, IELTS, Recommendation)

## IV. Set of database statements

```
USE master;
GO
IF EXISTS(SELECT name FROM sys.databases
    WHERE name = 'DBI_SPRING2017_PROJECT_AnhPTM')
    DROP DATABASE DBI_SPRING2017_PROJECT_AnhPTM

GO
CREATE DATABASE [DBI_SPRING2017_PROJECT_AnhPTM];
GO
USE [DBI_SPRING2017_PROJECT_AnhPTM];
GO

/****** Object:  Table [dbo].[Class]   ******/
```

```
CREATE TABLE "Universities"
(
	"UniID" nvarchar(20) NOT NULL ,
	"UniName" nvarchar (50) NOT NULL ,
	"Address" nvarchar (60) NULL ,
	"City" nvarchar(30),
	"Phone" nvarchar (24) NULL ,
	"UniEmail" nvarchar(60),
	CONSTRAINT "PK_Universities" PRIMARY KEY  CLUSTERED
	(
		"UniID" ASC
	)
)


CREATE TABLE "Applicants"
(
	"SSN" "int" NOT NULL ,
	"LastName" nvarchar (50) NOT NULL ,
	"FirstName" nvarchar (30) NOT NULL ,
	"DOB" date NULL ,
	"Gender" bit,
	"Address" nvarchar (60) NULL ,
	"Phone" nvarchar (24) NULL ,
	"AppEmail" nvarchar(60),
	"Nationality" nvarchar (30) NULL ,
	CONSTRAINT "PK_Student Details" PRIMARY KEY  CLUSTERED
	(
		"SSN" ASC
	),
	CONSTRAINT "CK_Birthdate" CHECK (DOB < getdate())
)



CREATE TABLE "ApplicantOffer"
(
	"SSN" "int" NOT NULL ,
	"UniID" nvarchar(20) NOT NULL ,
	"Type" nvarchar(20) NOT NULL,
	"Check" bit,
	"Accept" bit,
	CONSTRAINT "PK_StuUniOffer" PRIMARY KEY  CLUSTERED
	(
		"SSN" ASC,
		"UniID" ASC
```

```
	),
	CONSTRAINT "FK_AppliOffer_Stu" FOREIGN KEY
	(
		"SSN"
	) REFERENCES "dbo"."Applicants" (
		"SSN"
	),
	CONSTRAINT "FK_AppliOffer_Uni" FOREIGN KEY
	(
		"UniID"
	) REFERENCES "dbo"."Universities" (
		"UniID"
	),
	CONSTRAINT "CK_Type_Apply" CHECK ("Type" IN ('Condition','Uncondition'))
)



CREATE TABLE "Interview"
(
	"SSN" "int" NOT NULL ,
	"UniID" nvarchar(20) NOT NULL ,
	"Date" date,
	"Address" nvarchar(50),
	"Result" nvarchar(50),
	CONSTRAINT "PK_Interview" PRIMARY KEY  CLUSTERED
	(
		"SSN" ASC,
		"UniID" ASC
	),
	CONSTRAINT "FK_Interview_Stu" FOREIGN KEY
	(
		"SSN"
	) REFERENCES "dbo"."Applicants" (
		"SSN"
	),
	CONSTRAINT "FK_Interview_Uni" FOREIGN KEY
	(
		"UniID"
	) REFERENCES "dbo"."Universities" (
		"UniID"
	)
)



CREATE TABLE "ConditionalOffer"
(
	"UniID" nvarchar(20) NOT NULL ,
```

```
	"Description" nvarchar(150),
	"SAT" float,
	"TOELF" float,
	"IELTS" float,
	"TOELF/IELTS" int,
	"Recommendation" bit,
	CONSTRAINT "PK_ConditionalOffer" PRIMARY KEY  CLUSTERED
	(
		"UniID"  ASC
	),
	CONSTRAINT "FK_ConditionalOffer_UniNo" FOREIGN KEY
	(
		"UniID"
	) REFERENCES "dbo"."Universities"  (
		"UniID"
	),
	CONSTRAINT "CK_TOELF_IELTS" CHECK ("TOELF/IELTS" IN (1,2)),
)



CREATE TABLE "Grade"
(
	"SSN" "int" NOT NULL ,
	"SAT" float,
	"TOELF" float,
	"IELTS" float,
	"Recommendation" bit,
	CONSTRAINT "PK_Transcript" PRIMARY KEY  CLUSTERED
	(
		"SSN" ASC
	),
	CONSTRAINT "FK_Transcript_Stu" FOREIGN KEY
	(
		"SSN"
	) REFERENCES "dbo"."Applicants" (
		"SSN"
	)
)



**** CAC TRIGGER --- PROC

CREATE TRIGGER trigger_Apply
ON "ApplicantOffer"
FOR INSERT
AS
BEGIN
```

```
	DECLARE @SSN int;
	DECLARE @UniID nvarchar(20)
	DECLARE @number int;
	SELECT @SSN = SSN FROM Inserted
	SELECT @UniID = UniID FROM Inserted
	SELECT @number = COUNT(*) FROM "ApplicantOffer" WHERE SSN = @SSN
	IF(@number >5)
	BEGIN
	print 'An applicant cannot apply more than 5 universities';
	rollback Tran
	END
END



CREATE TRIGGER trigger_Check
ON "ApplicantOffer"
FOR INSERT, UPDATE
AS
BEGIN
	DECLARE @SSN int;
	DECLARE @UniID nvarchar(20)
	DECLARE @number int;
	SELECT @SSN = SSN FROM Inserted
	SELECT @UniID = UniID FROM Inserted
	SELECT @number = COUNT(*) FROM "ApplicantOffer" WHERE
	SSN = @SSN AND "check" = 1 AND "TYPE" = 'Condition'
	IF(@number >3)
	BEGIN
	print 'A applicant cannot choose more than 3 universities';
	rollback Tran
	END
END



CREATE TRIGGER trigger_Accept
ON "ApplicantOffer"
FOR INSERT, UPDATE
AS
BEGIN

	DECLARE @SSN int;
	DECLARE @UniID nvarchar(20)
	DECLARE @number int;
	DECLARE @type nvarchar(20);
	DECLARE @accept bit;
	SELECT @SSN = SSN FROM Inserted
```

```
        SELECT @UniID = UniID FROM Inserted
        SELECT @type = "TYPE" FROM inserted
        SELECT @accept = "accept" FROM inserted
        SELECT @number = COUNT(*) FROM "ApplicantOffer" WHERE
        SSN = @SSN AND "Type" = 'Uncondition' AND "accept" = 1
        IF(@number >1)
        BEGIN
                print 'A applicant can accept only 1 university';
                rollback Tran
        END
        ELSE IF (@accept = 1 AND @type = 'Condition')
        BEGIN
                print 'Applicant can not accept conditional offer '
                rollback Tran
        END
END



CREATE PROC sp_DetailApplicant
        @SSN nvarchar(20)
AS
BEGIN
        SELECT * FROM Applicants WHERE SSN = @SSN
END



**** CREATE VIEW

CREATE VIEW Applicant_VIEW AS
SELECT SSN, FirstName + LastName as [FullName]
FROM  Applicants
```

## V. Queries

```
---Cau 1: Communicate Conditional offer to the apllicants.
---- Trao doi thong tin Offerletter cho applicant

SELECT a.SSN, a.AppEmail, co.UniID, co."Description" FROM
Applicants a INNER JOIN ApplicantOffer ao ON a.SSN = ao.SSN
INNER JOIN ConditionalOffer co ON ao.UniID = co.UniID
```

```
---Cau 2: A query that uses INNER JOINS
---Find applicants ID check universities which have condition:  SAT >1000

SELECT ao.SSN FROM ApplicantOffer ao INNER JOIN ConditionalOffer co
ON ao.UniID = co.UniID
WHERE co.SAT >1000 AND "CHECK" = 1



---Cau 3: A query that uses aggregate functions
---Select number of universites and number of Applicants enroll the Entrance system
SELECT * FROM
(SELECT COUNT(*) as [Number of Universities] FROM Universities)one,
(SELECT COUNT(*) as [Number of Applicants] FROM Applicants)two



---Cau 4: A query that uses the GROUP BY and HAVING clauses
--- SELECT name of universities which have number of Applicants check >2

SELECT one.UniID, one.UniName, two.[Number of Applicants] FROM
(SELECT u.UniID, u.UniName FROM Universities u)one
INNER JOIN
(SELECT UniID, COUNT(*) as [Number of Applicants] FROM ApplicantOffer
WHERE "CHECK" = 1
GROUP BY UniID
HAVING COUNT(*) > 2) two
ON one.UniID = two.UniID



---Cau 5: A query that uses a sub-query as a relation
--- Name of applicants enroll the most universities.

SELECT name.SSN, name.[FullName], num.[Number of Universities] FROM
(SELECT ao.SSN, COUNT(*) as [Number of Universities] FROM ApplicantOffer ao
GROUP BY ao.SSN) num
INNER JOIN
(
SELECT * FROM Applicant_VIEW
WHERE SSN NOT IN
(SELECT one.SSN FROM
(SELECT ao.SSN, COUNT(*) as [Number of Universities] FROM ApplicantOffer ao
GROUP BY ao.SSN)one,
(SELECT ao.SSN, COUNT(*) as [Number of Universities] FROM ApplicantOffer ao
GROUP BY ao.SSN)two
WHERE one.[Number of Universities] < two.[Number of Universities])) name
ON num.SSN = name.SSN
```

```
---Cau 6: A query that uses ORDER BY
---Order universities by number of student apply.

SELECT one.UniID, one.UniName, two.[Number of Applicants] FROM
(SELECT u.UniID, u.UniName FROM Universities u)one
INNER JOIN
(SELECT UniID, COUNT(*) as [Number of Applicants] FROM ApplicantOffer
GROUP BY UniID) two
ON one.UniID = two.UniID
ORDER BY two.[Number of Applicants] DESC



---Cau 7: A query that uses partial matching in the WHERE clause
---Applicant has firstName like "Annie" apply which universites

SELECT a.SSN, a.[FullName], u.UniName FROM Applicant_VIEW a INNER JOIN
ApplicantOffer ao ON a.SSN = ao.SSN
INNER JOIN Universities u ON ao.UniID = u.UniID
WHERE a.[FullName] like '%Annie%'



---Cau 8: A query that uses a self-JOIN
---Applicant (s) has the highest score of SAT

SELECT * FROM Applicants
Where SSN NOT IN
(SELECT one.SSN FROM Grade one, Grade two
WHERE one.SAT < two.SAT)



---Cau 9:
---Top 3 Universities have condition on  highest score of SAT.

SELECT one.UniID, one.UniID, two.SAT FROM
(SELECT u.UniID, u.UniName FROM Universities u)one
INNER JOIN
(
SELECT TOP 3 co.UniID, co.SAT FROM ConditionalOffer co
ORDER BY co.SAT DESC) two
ON one.UniID = two.UniID
```

```
---Cau 10:
--- Grade of applicant whose name is Carlos.

SELECT a.SSN, a.[FullName], g.SAT, g.IELTS, g.TOELF, g.Recommendation FROM
Applicant_VIEW a INNER JOIN Grade g
ON a.SSN = g.SSN
WHERE a.[FullName] like '%Carlos%'
```

## VI. Conclusion

In conclusion, I will add time in the database if I will do that again such as: enrolling students by year or semester, date to accept the application form and expiration date to take the decision of apllicant about the finally university he/she accept..