Name: ____________________________ eID:_____________________ Course: CIS560 CIS562

CIS560 & CIS562- Exam 2

You will have until the end of class to complete this exam, allowing **approximately 45 minutes**. It is a closed-book exam, so you should not have any materials next to you. **Please put away all papers, phones, and watches**.

Important: Please write your name and eID on each page in case the pages get separated. Also, please indicate which course you are in by circling either CIS560 or CIS562 in the upper right of each page.

You have two options for submitting your solutions:

1. Write your solution by hand on the following pages.
2. Typing them and submitting in Canvas. *Read carefully below.*

If typing your solution on a laptop, **you still need to turn in this paper exam** with your name, eID, and course circled, and under *Exam Submission* below, you have circled "Canvas" for the questions you are submitting online. On your laptop, you may have no more than two applications running: 1) a browser with the assignment open in Canvas and 2) an application for typing your solutions, limited to a basic text editor, SQL Server Management Studio, or SQL Operations Studio. **You must have no other applications running, and with each application, you may have only one tab or file open.**

Exam Submission

Indicate how you are submitting your answers by circling "Paper" or "Canvas" below for each question.

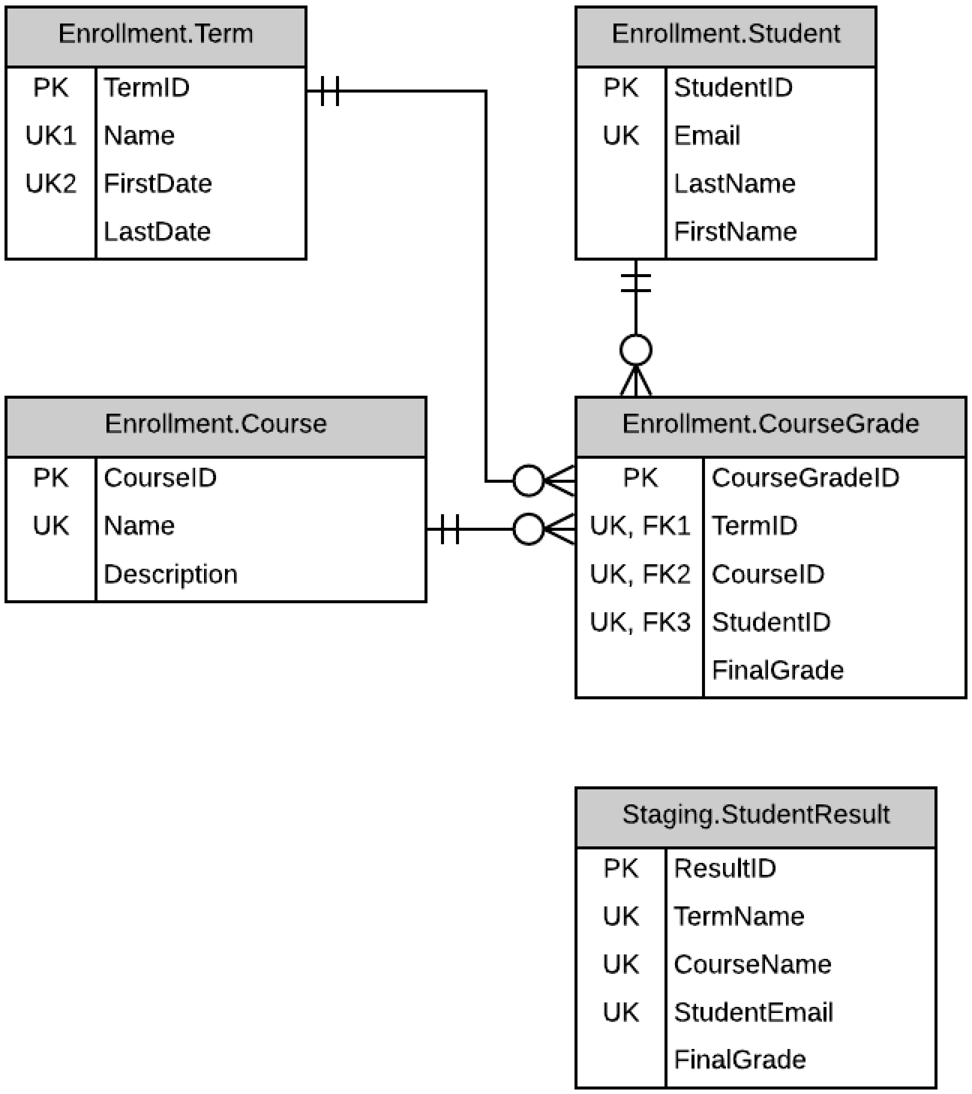
Question	Submission Type	
1	Paper	Canvas
2	Paper	Canvas
3	Paper	Canvas
4	Paper	Canvas
5	Paper	Canvas
6	Paper	Canvas

Circling "Paper" indicates your answer for that question is filled out on this paper exam.

Circling "Canvas" indicates your answer for that question was submitted online in Canvas.

Database Design

Some questions on this exam use tables from the below diagram. For the most part, the complete column specifications are unimportant for the purposes of this exam. More information may be provided as necessary in the questions that follow.



Enrollment.Term	
PK	TermID
UK1	Name
UK2	FirstDate
	LastDate

Enrollment.Student	
PK	StudentID
UK	Email
	LastName
	FirstName

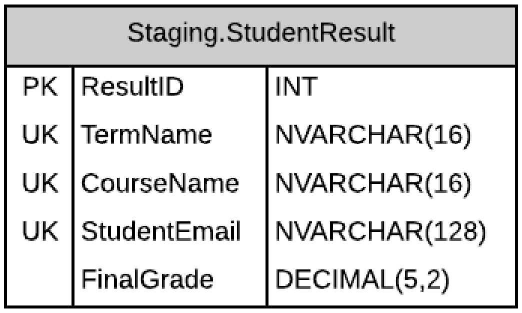
Enrollment.Course	
PK	CourseID
UK	Name
	Description

Enrollment.CourseGrade	
PK	CourseGradeID
UK, FK1	TermID
UK, FK2	CourseID
UK, FK3	StudentID
	FinalGrade

Staging.StudentResult	
PK	ResultID
UK	TermName
UK	CourseName
UK	StudentEmail
	FinalGrade

Name: ______________________________ eID:______________________ Course: CIS560 CIS562

1. **20 points** – Write a SQL statement that will create the `Staging.StudentResult` table. A more detailed diagram is shown here with the data types.



Staging.StudentResult		
PK	ResultID	INT
UK	TermName	NVARCHAR(16)
UK	CourseName	NVARCHAR(16)
UK	StudentEmail	NVARCHAR(128)
	FinalGrade	DECIMAL(5,2)

We want the `ResultID` column to automatically generate values as rows are inserted, starting with 1 and incrementing by 1. For simplicity, use a property on the `ResultID` column to achieve this desired behavior rather than using a sequence object. You may assume the schema, `Staging`, for the table already exists.

```
CREATE TABLE Staging.StudentResult
(
   ResultID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
   TermName NVARCHAR(16) NOT NULL,
   CourseName NVARCHAR(16) NOT NULL,
   StudentEmail NVARCHAR(128) NOT NULL,
   FinalGrade DECIMAL(5,2) NOT NULL,

   UNIQUE(TermName, CourseName, StudentEmail)
);
```

2. **20 points** – Using the tables as defined in the diagram on *Page 2*, complete the view implementation below that will return all rows from `Staging.StudentResult`, but translating `TermName`, `CourseName`, and `StudentEmail` into the appropriate `TermID`, `CourseID`, and `StudentID` respectively. The solution, then, should only return the columns `TermID`, `CourseID`, `StudentID`, and `FinalGrade`.

```
CREATE VIEW Staging.CourseGrade
AS

-- Provide your implementation here.
SELECT T.TermID, C.CourseID, S.StudentID, SR.FinalGrade
FROM Staging.StudentResult SR
   INNER JOIN Enrollment.Term T ON T.Name = SR.TermName
   INNER JOIN Enrollment.Course C ON C.Name = SR.CourseName
   INNER JOIN Enrollment.Student S ON S.Email = SR.StudentEmail;
```

3. **20 points** – Write a **MERGE** statement that will evaluate all the rows from the view defined in *Question 2*, `Staging.CourseGrade`, and either insert or update rows in the `Enrollment.CourseGrade` table. If the row already exists, you only need to update `FinalGrade`. You should assume the `CourseGradeID` column of the `Enrollment.CourseGrade` table has an IDENTITY property, thus an explicit value should not be provided upon inserting.

```
MERGE Enrollment.CourseGrade T
USING Staging.CourseGrade S ON S.TermID = T.TermID
   AND S.CourseID = T.CourseID
   AND S.StudentID = T.StudentID
WHEN MATCHED THEN
   UPDATE
   SET FinalGrade = S.FinalGrade
WHEN NOT MATCHED THEN
   INSERT(TermID, CourseID, StudentID, FinalGrade)
   VALUES(S.TermID, S.CourseID, S.StudentID, S.FinalGrade);
```

4. **10 points** – The table `Staging.StudentResult` currently has 10,000 rows.
   a. 3 points – Write a SQL statement that will truncate the table `Staging.StudentResult`, deleting all rows.

   ```
   TRUNCATE TABLE Staging.StudentResult;
   ```

   b. 1 point – After the table `Staging.StudentResult` is truncated, we insert 1,000 rows. What would be the first (minimum) value assigned for `ResultID`.

   The next value assigned after a truncate is 1.

   c. 1 point – What would be the last (maximum) value assigned for the 1,000 newly inserted rows?S

   Since we started the 1,000 rows with 1, the last value would be 1000.

   d. 5 points – Given all tables in the diagram on *Page 2*, which tables *could* be truncated should we need to? Circle all tables that would support truncation.

`Enrollment.Term`	`Enrollment.Student`	`Enrollment.Course`
<u>`Enrollment.CourseGrade`</u>	<u>`Staging.StudentResult`</u>	

   Only tables with no foreign key constraints referencing it can be truncated. It may have foreign keys referencing other tables.

5. **15 points** – Below is a relation and its known functional dependencies.

```
Schedule(Course, StartTime, Room, Instructor, Department, Building)

{ Course, StartTime } → { Instructor }
{ Instructor } → { Department }
{ Room } → { Building }
{ Room, StartTime } → { Course }
```

For each of the four provided functional dependencies above, X → Y, compute X^+. Once you have the closures computed, identify a key for the relation `Schedule`.

```
{ Course, StartTime }⁺ = { Course, StartTime, Instructor, Department }
{ Instructor }⁺ = { Instructor, Department }
{ Room }⁺ = { Room, Building }
{ Room, StartTime }⁺ = {Room, StartTime, Course, Instructor, Department, Building}
```

`{ Room, StartTime }` is a key.

6. **15 points** – Using the relation, functional dependencies, and calculated closures from *Question 5*, use the closure of `{ Course, StartTime }`, which violates BCNF, to decompose the `Schedule` relation into BCNF relations. Show all your work, including intermediate relations. Clearly mark the resulting BCNF relations along with the keys in each.

Iteration 1

X = { Course, StartTime }
Schedule_1(Course, StartTime, Instructor, Department)
Schedule_2(Course, StartTime, Room, Building)

Iteration 2

X = { Instructor }
$\text{Schedule}_{1.1}$(Instructor, Department) – BCNF
$\text{Schedule}_{1.2}$(Instructor, Course, StartTime) – BCNF

X = { Room }
$\text{Schedule}_{2.1}$(Room, Building) – BCNF
$\text{Schedule}_{2.2}$(Room, Course, StartTime) – BCNF

Solution

$\text{Schedule}_{1.1}$(<u>Instructor</u>, Department) – A better name would be "Instructor".
$\text{Schedule}_{1.2}$(Instructor, <u>Course</u>, <u>StartTime</u>) – BCNF
$\text{Schedule}_{2.1}$(<u>Room</u>, Building) – A better name would be "Room".
$\text{Schedule}_{2.2}$(<u>Room</u>, Course, <u>StartTime</u>)