# Assignment 3

## Database Construction and Processing

Caitlin Ross
#040750891
CST 2355 Database Systems
Section 451

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

# Contents

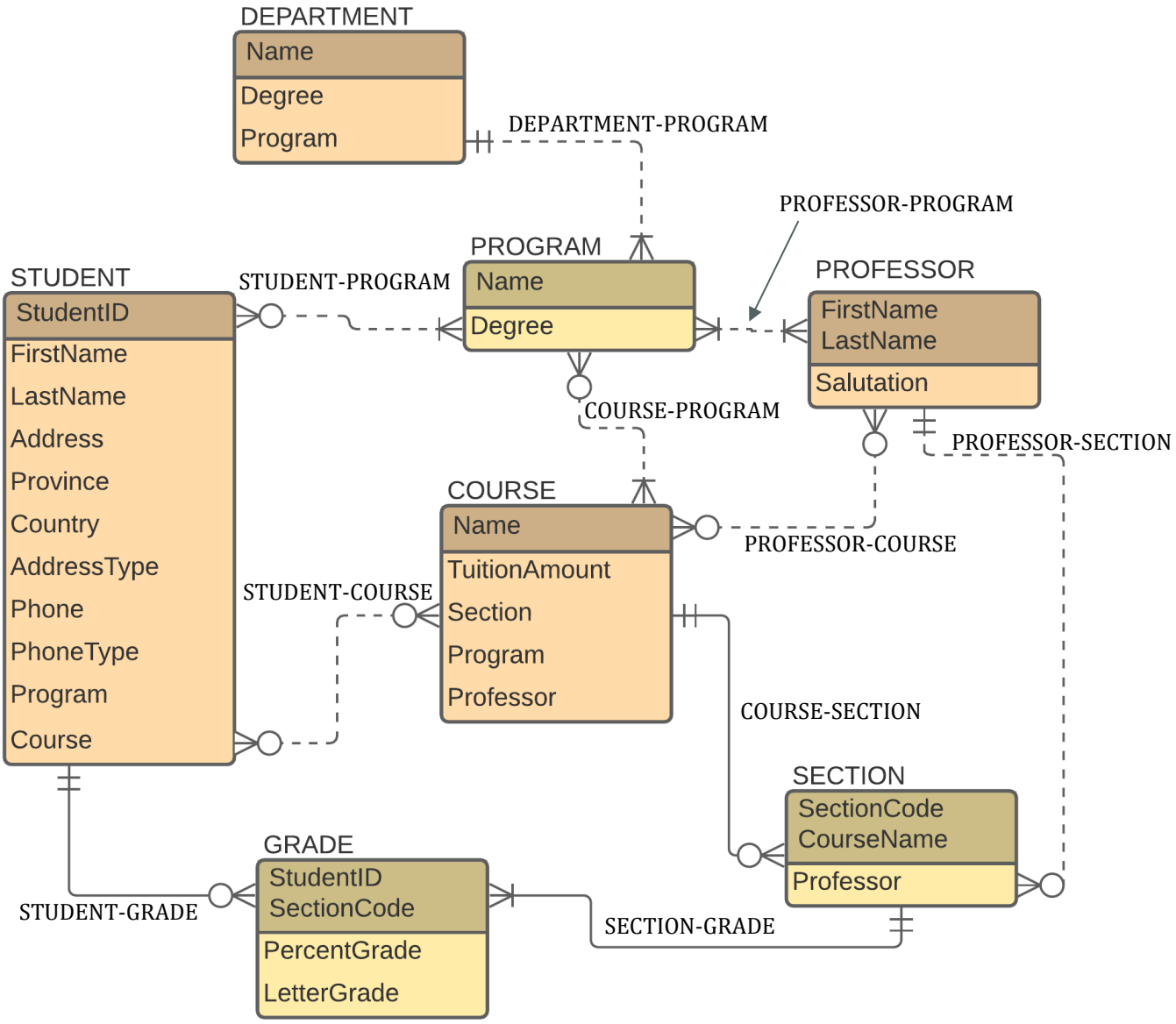
Data Model ..... 4
Transformation Decisions ..... 5
Cardinality Charts ..... 6
Action Charts ..... 7
STUDENT:GRADE ..... 7
PROGRAM:STUDENT ..... 7
PROGRAM:PROFESSOR ..... 7
PROGRAM:COURSE ..... 8
COURSE:GRADE ..... 8
COURSE:SECTION ..... 8
PROFESSOR:SECTION ..... 9
PROFESSOR:COURSE ..... 9
SECTION:GRADE ..... 9
Table Characteristics ..... 10
STUDENT ..... 10
PROFESSOR ..... 10
PROGRAM ..... 10
COURSE ..... 11
SECTION ..... 11
GRADE ..... 11
COURSE_PROG_INT ..... 11
PROF_TEACHES_INT ..... 12
Database Design Diagram ..... 13
Database Scripts ..... 14
View Scripts ..... 14
Create Student-Professor-Course View ..... 14
Create Update Trigger on Student-Professor-Course View ..... 15
Create Delete Trigger on Student-Professor-Course View ..... 17

Table Scripts ..... 18
Create STUDENT Table ..... 18
Add CHECK Constraint to PhoneNo ..... 18
Create PROFESSOR Table ..... 18
Add CHECK Constraint to Salutation ..... 18
Create Update Trigger on PROFESSOR ..... 19
Create PROGRAM Table ..... 20
Create Update Trigger on PROGRAM ..... 20
Create COURSE Table ..... 21
Create Update Trigger on COURSE ..... 21
Create SECTION Table ..... 22
Create Insert Trigger on SECTION ..... 22
Create Update Trigger on SECTION ..... 23
Create Delete Trigger on SECTION ..... 25
Create GRADE Table ..... 25
Create Update Trigger on GRADE ..... 26
Create COURSE_PROG_INT Table ..... 27
Create PROF_TEACHES_INT Table ..... 27
Export Database ..... 28
Import Database ..... 29
Delete All Data (Keeping Structure) ..... 30
Procedure/Function Scripts ..... 32
Calculate Letter Grade from Percent Grade ..... 32
Section Reassignment Based on Last Name ..... 32
Reports ..... 34
Program Enrollment Report ..... 34
Course Enrollment Report ..... 34
Program/Course Enrollment Report ..... 34
Course/Program Matrix ..... 35
Student Grade Average ..... 35

Request Scripts ..... 36
Change PhoneType's Default Value ..... 36
Change SectionCode Default Value ..... 36
Change TuitionAmount's Datatype ..... 36
Change SectionCode Column Contents ..... 36
Appendix 1 ..... 37

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

# Data Model



# Transformation Decisions

| Entity/Relationship | Table/Relationship | Comments |
|---|---|---|
| STUDENT | STUDENT | Use StudentID as PK |
| PROFESSOR | PROFESSOR | Add ProfID as PK |
| DEPARTMENT | N/A | Becomes a field in the PROGRAM table |
| COURSE | COURSE | |
| PROGRAM | PROGRAM | Includes DEPARTMENT entity |
| SECTION | SECTION | |
| GRADE | GRADE | Despite being its own entity, it's also an intersection between STUDENT and SECTION |
| STUDENT-GRADE | N/A | Becomes a field in the GRADE table |
| STUDENT-COURSE | N/A | Becomes a field in the GRADE table |
| STUDENT-PROGRAM | N/A | Becomes a field in the STUDENT table |
| PROFESSOR-PROGRAM | N/A | Becomes a field in the PROFESSOR table |
| PROFESSOR-COURSE | PROF_TEACHES_INT | Combine with PROFESSOR-SECTION relationship |
| PROFESSOR-SECTION | PROF_TEACHES_INT | Combine with PROFESSOR-COURSE relationship |
| DEPARTMENT-PROGRAM | N/A | Becomes a field in the DEPARTMENT table |
| COURSE-PROGRAM | COURSE_PROG_INT | Create new intersection table |
| COURSE-SECTION | N/A | Becomes a field in the SECTION table |
| SECTION-GRADE | N/A | Becomes a field in the GRADE table |

- It's unclear whether STUDENT or PROFESSOR entities can have more than one PROGRAM. In the model I assumed they could have multiple, but in order to simplify the design I'll assume they can only have one. To implement STUDENT and PROFESSOR entities having multiple PROGRAMs they would each need a new intersection table.

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

# Cardinality Charts

| Relationships | | Cardinality | | |
|---|---|---|---|---|
| Parent | Child | Type | Min | Max |
| STUDENT | GRADE | Weak, GRADE is ID-dependant | M-O | 1:N |
| PROGRAM | STUDENT | Weak, non-ID-dependant | M-O | 1:N |
| PROGRAM | PROFESSOR | Weak, non-ID-dependant | M-O | 1:N |
| PROGRAM | COURSE | Weak, non-ID-dependant | M-M | N:M |
| COURSE | GRADE | Weak, GRADE is ID-dependant | M-O | 1:N |
| COURSE | SECTION | Weak, SECTION is ID-dependant | M-O | 1:N |
| PROFESSOR | SECTION | Weak, non-ID-dependant | M-O | 1:N |
| PROFESSOR | COURSE | Strong | O-O | N:M |
| SECTION | GRADE | Weak, non-ID-dependant | O-M | 1:N |

- Despite the STUDENT, PROFESSOR, and PROGRAM entities all being strong, I described the relationships as weak because in the context of the business I'm assuming a STUDENT or PROFESSOR cannot exist without being part of a PROGRAM. These would be HAS-A relationships.
- The SECTION-GRADE relationship is the only way to tell which students are enrolled in each course. PercentGrade and LetterGrade would default to 0.0%/F. However, after a course is finished the SECTION part of the GRADE table would become NULL. There is currently no way of determining if a student is currently on a break or has completed their studies. That could be corrected in the future by adding a DegreeObtained field to the STUDENT table.

# Action Charts

## STUDENT:GRADE

| STUDENT required; GRADE optional | Action on STUDENT (Parent) | Action on GRADE (Child) |
|---|---|---|
| **Insert** | None | StudentID must match existing STUDENT record |
| **Modify Key** | Disallow for business reasons | Disallow for business reasons |
| **Delete** | Disallow for business reasons | Disallow for business reasons |

## PROGRAM:STUDENT

| PROGRAM required; STUDENT optional | Action on PROGRAM (Parent) | Action on STUDENT (Child) |
|---|---|---|
| **Insert** | None | Program must match existing PROGRAM record |
| **Modify Key** | Trigger updates on STUDENT records with matching Program | Disallow for business reasons |
| **Delete** | Only allow if no STUDENT records have matching Program | Disallow for business reasons |

## PROGRAM:PROFESSOR

| PROGRAM required; PROFESSOR optional | Action on PROGRAM (Parent) | Action on PROFESSOR (Child) |
|---|---|---|
| **Insert** | None | Program must match existing PROGRAM record |
| **Modify Key** | Trigger updates on PROFESSOR records with matching Program | Disallow for business reasons |
| **Delete** | Only allow if no PROFESSOR records have matching Program | Disallow for business reasons |

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

## PROGRAM:COURSE

| PROGRAM required; COURSE required | Action on PROGRAM (Parent) | Action on COURSE (Child) |
|---|---|---|
| **Insert** | None | Program must match existing PROGRAM record |
| **Modify Key** | Trigger updates on COURSE records with matching Program | None |
| **Delete** | Only allow if no COURSE records have matching Program | Disallow for business reasons |

## COURSE:GRADE

| COURSE required; GRADE optional | Action on COURSE (Parent) | Action on GRADE (Child) |
|---|---|---|
| **Insert** | None | Course must match existing COURSE record |
| **Modify Key** | Trigger updates on GRADE records with matching Course | Disallow for business reasons |
| **Delete** | Disallow for business reasons | Disallow for business reasons |

## COURSE:SECTION

| COURSE required; SECTION optional | Action on COURSE (Parent) | Action on SECTION (Child) |
|---|---|---|
| **Insert** | None | Ensure SectionCode is sequential for all SECTION records with matching Course |
| **Modify Key** | Trigger updates on SECTION records with matching Course | Disallow |
| **Delete** | Disallow for business reasons | Trigger updates on other SECTION records with matching Course |

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

## PROFESSOR:SECTION

| PROFESSOR required; SECTION optional | Action on PROFESSOR (Parent) | Action on SECTION (Child) |
|---|---|---|
| **Insert** | None | ProfessorName must match one of the entries in the PROF_TEACHES_INT with a matching Course |
| **Modify Key** | Disallow for business reasons | ProfessorName must match one of the entries in the PROF_TEACHES_INT with a matching Course |
| **Delete** | Disallow for business reasons | None (in the context of this relationship) |

## PROFESSOR:COURSE

| PROFESSOR optional; COURSE optional | Action on PROFESSOR (Parent) | Action on COURSE (Child) |
|---|---|---|
| **Insert** | None | None |
| **Modify Key** | Disallow for business reasons | None |
| **Delete** | Disallow for business reasons | Disallow for business reasons |

## SECTION:GRADE

| SECTION optional; GRADE required | Action on SECTION (Parent) | Action on GRADE (Child) |
|---|---|---|
| **Insert** | Must have a GRADE to be created | None |
| **Modify Key** | Trigger updates on GRADE records with matching SECTION | None |
| **Delete** | None | If last GRADE for a given SECTION, delete corresponding SECTION |

- Assume that STUDENT, GRADE, PROFESSOR, and COURSE records cannot be deleted (this would depend on business policies). In practice they would likely be moved to a legacy table of some sort.
- Assume that PROGRAM records can only be deleted if no PROFESSOR, STUDENT, or COURSE records match (very unusual, would probably only occur if a record was created by accident).

# Table Characteristics

## STUDENT

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| StudentID | PK, NN | Auto Increment | INT | Int | INT |
| FirstName | NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| LastName | NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| StreetAddress | | | VARCHAR2(128) | Varchar(128) | VARCHAR(128) |
| Province | | | CHAR(2) | Char(2) | CHAR(2) |
| Country | | | VARCHAR2(32) | Varchar(32) | VARCHAR(32) |
| AddressType | | | VARCHAR2(32) | Varchar(32) | VARCHAR(32) |
| PhoneNo | Only numerals | | CHAR(10) | Char(10) | CHAR(10) |
| PhoneType | | | VARCHAR2(32) | Varchar(32) | VARCHAR(32) |
| Program | NN, FK | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |

## PROFESSOR

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| FirstName | CK.1, NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| LastName | CK.2, NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| Salutation | NN | 'Dr.' | VARCHAR2(4) | Varchar(4) | VARCHAR(4) |
| Program | NN, FK | 'Data Science' | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |

## PROGRAM

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| ProgramName | PK, NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| Degree | NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| Department | NN | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |

# ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

## COURSE

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| CourseName | PK, NN | | VARCHAR2(128) | Varchar(128) | VARCHAR(128) |
| TuitionAmount | NN | 0 | NUMBER(6,0) | SmallInt | SMALLINT |

## SECTION

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| SectionCode | CK.1, NN | Use a trigger, not DEFAULT | NUMBER (2,0) | TinyInt | TINYINT |
| CourseName | CK.2, NN, FK | | VARCHAR2 (128) | Varchar(128) | VARCHAR(128) |
| ProfFirstName | FK.1 | | VARCHAR2 (64) | Varchar(64) | VARCHAR(64) |
| ProfLastName | FK.2 | | VARCHAR2 (64) | Varchar(64) | VARCHAR(64) |

## GRADE

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| StudentID | CK.1, NN, FK | | INT | Int | INT |
| CourseName | CK.2, NN, FK.1 | | VARCHAR2(128) | Varchar(128) | VARCHAR(128) |
| SectionCode | FK.2 | | NUMBER(2,0) | TinyInt | TINYINT |
| PercentGrade | | 0.0 | NUMBER(4,1) | Decimal(4,1) | DECIMAL(3,1) |
| LetterGrade | First char in (A,B,C,D,F), second in (+,-) | 'F' | CHAR(2) | Char(2) | CHAR(2) |

## COURSE_PROG_INT

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| Program | CK.1, NN, FK | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| Course | CK.2, NN, FK | | VARCHAR2(128) | Varchar(128) | VARCHAR(128) |

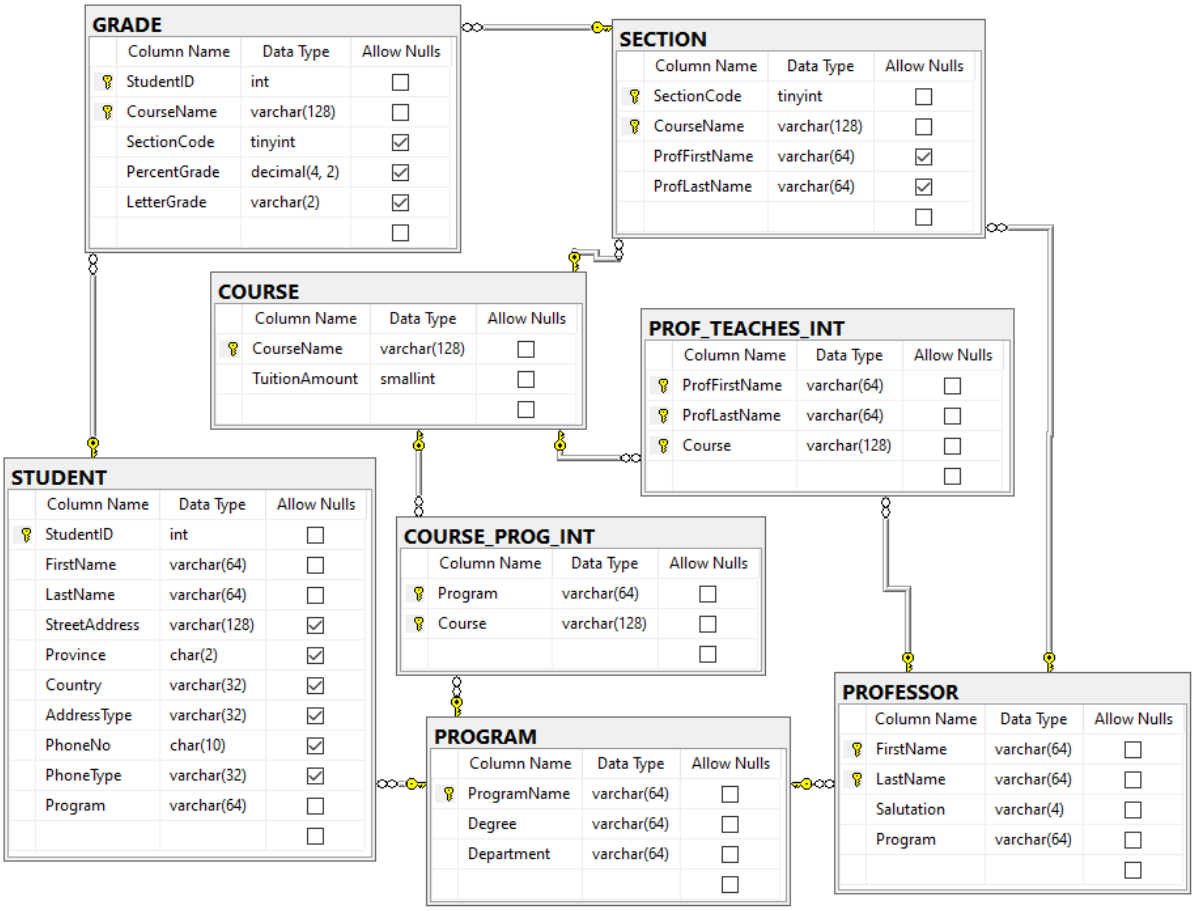
## PROF_TEACHES_INT

| Column ID | Constraints | Default | Oracle Datatype | SQL Server Datatype | MySQL Datatype |
|---|---|---|---|---|---|
| ProfFirstName | CK.1, NN, FK.1 | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| ProfLastName | CK.2, NN, FK.2 | | VARCHAR2(64) | Varchar(64) | VARCHAR(64) |
| Course | CK.3, NN, FK | | VARCHAR2(128) | Varchar(128) | VARCHAR(128) |

- Using abbreviations based on MySQL: Primary Key (PK), Composite Key (CK.1, CK.2, etc.), Foreign Key (FK), Not Null (NN), Unique (UQ). If NN isn't present, that column can be null.
- Moved Auto Increment to Default column instead of Constraints column. Seemed more logical to me. However, for the SECTION.SectionCode I will have to use a trigger instead of a DEFAULT value.
- Changed Address field to StreetAddress (STUDENT table) since SQL Server (and possible the others) uses Address as a reserved keyword
- Changed Salutation field (PROFESSOR table) from CHAR(4) to VARCHAR(4)/VARCHAR2(4) since some options have 3 characters ('Dr.', 'Mr.', 'Ms.') and some have 4 ('Mrs.', 'Miss')
- For the same reason, changed LetterGrade field (GRADE table) from CHAR(2) to VARCHAR(2) since the +/- in the second spot isn't consistent
- Removed Unique constraint from primary keys since it's redundant.
- In order to use the multicolumn primary key in the PROFESSOR table, I had to switch SECTION.Professor to SECTION.ProfFirstName and SECTION.ProfLastName. This also required changing all tables (PROF_TEACHES_INT, SECTION) referring to PROFESSOR to have a multicolumn foreign key. I'm assuming that there aren't enough professors at this institution to worry about duplicate names. If the business decided that it wasn't a safe assumption, a surrogate key would be needed.

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

# Database Design Diagram




- Made using SQL Server after being created with SQL statements
- Despite the order of the scripts in the next section, PROGRAM and COURSE tables were created first, then STUDENT and PROFESSOR, then SECTION, then GRADE, then COURSE_PROG_INT and PROF_TEACHES_INT. This was done so that the foreign key constraints were being created on existing tables.

# Database Scripts

## View Scripts

### Create Student-Professor-Course View

```
1.    CREATE VIEW dbo.VW_StudProfCourse AS
2.     SELECT
3.       PROF_TEACHES_INT.Course,
4.       PROF_TEACHES_INT.ProfFirstName,
5.       PROF_TEACHES_INT.ProfLastName,
6.       enrolled.StudentFirstName,
7.       enrolled.StudentLastName
8.     FROM PROF_TEACHES_INT
9.     INNER JOIN
10.      (SELECT
11.       GRADE.CourseName,
12.       GRADE.StudentID,
13.       STUDENT.FirstName AS StudentFirstName,
14.       STUDENT.LastName AS StudentLastName
15.      FROM GRADE
16.      INNER JOIN STUDENT
17.      ON GRADE.StudentID = STUDENT.StudentID) enrolled
18.    ON PROF_TEACHES_INT.Course = enrolled.CourseName;
```

## Create Update Trigger on Student-Professor-Course View

```
1.    CREATE TRIGGER dbo.TR_StudProfCourseUpdate
2.      ON dbo.VW_StudProfCourse
3.      INSTEAD OF UPDATE AS
4.    BEGIN
5.      -- Declare variables
6.      DECLARE @OldCourse Varchar(128);
7.      DECLARE @OldProfFName Varchar(64);
8.      DECLARE @OldProfLName Varchar(64);
9.      DECLARE @OldStudFName Varchar(64);
10.     DECLARE @OldStudLName Varchar(64);
11.     DECLARE @NewCourse Varchar(128);
12.     DECLARE @NewProfFName Varchar(64);
13.     DECLARE @NewProfLName Varchar(64);
14.     DECLARE @NewStudFName Varchar(64);
15.     DECLARE @NewStudLName Varchar(64);
16.     -- Initialize variables
17.     SELECT @OldCourse = DELETED.Course FROM DELETED;
18.     SELECT @OldProfFName = DELETED.ProfFirstName FROM DELETED;
19.     SELECT @OldProfLName = DELETED.ProfLastName FROM DELETED;
20.     SELECT @OldStudFName = DELETED.StudentFirstName FROM DELETED;
21.     SELECT @OldStudLName = DELETED.StudentLastName FROM DELETED;
22.     SELECT @NewCourse = INSERTED.Course FROM INSERTED;
23.     SELECT @NewProfFName = INSERTED.ProfFirstName FROM INSERTED;
24.     SELECT @NewProfLName = INSERTED.ProfLastName FROM INSERTED;
25.     SELECT @NewStudFName = INSERTED.StudentFirstName FROM INSERTED;
26.     SELECT @NewStudLName = INSERTED.StudentLastName FROM INSERTED;
27.     -- If the Course has changed, update COURSE, SECTION, GRADE,
        COURSE_PROG_INT, PROF_TEACHES_INT
28.     IF @NewCourse != @OldCourse
29.     BEGIN
30.       UPDATE COURSE
31.         SET COURSE.CourseName = @NewCourse
32.         WHERE COURSE.CourseName = @OldCourse;
33.       UPDATE SECTION
34.         SET SECTION.CourseName = @NewCourse
35.         WHERE SECTION.CourseName = @OldCourse;
36.       UPDATE GRADE
37.         SET GRADE.CourseName = @NewCourse
38.         WHERE GRADE.CourseName = @OldCourse;
39.       UPDATE COURSE_PROG_INT
40.         SET COURSE_PROG_INT.Course = @NewCourse
41.         WHERE COURSE_PROG_INT.Course = @OldCourse;
42.       UPDATE PROF_TEACHES_INT
43.         SET PROF_TEACHES_INT.Course = @NewCourse
44.         WHERE PROF_TEACHES_INT.Course = @OldCourse;
45.     END;
```

```
46.     -- If the Professor's first name has changed, update PROFESSOR,
        PROF_TEACHES_INT, SECTION
47.     IF @NewProfFName != @OldProfFName
48.     BEGIN
49.       UPDATE PROFESSOR
50.         SET PROFESSOR.FirstName = @NewProfFName
51.         WHERE PROFESSOR.FirstName = @OldProfFName;
52.       UPDATE PROF_TEACHES_INT
53.         SET PROF_TEACHES_INT.ProfFirstName = @NewProfFName
54.         WHERE PROF_TEACHES_INT.ProfFirstName = @OldProfFName;
55.       UPDATE SECTION
56.         SET SECTION.ProfFirstName = @NewProfFName
57.         WHERE SECTION.ProfFirstName = @OldProfFName;
58.     END;
59.     -- If the Professor's last name has changed, update PROFESSOR,
        PROF_TEACHES_INT, SECTION
60.     IF @NewProfLName != @OldProfLName
61.     BEGIN
62.       UPDATE PROFESSOR
63.         SET PROFESSOR.LastName = @NewProfLName
64.         WHERE PROFESSOR.LastName = @OldProfLName;
65.       UPDATE PROF_TEACHES_INT
66.         SET PROF_TEACHES_INT.ProfLastName = @NewProfLName
67.         WHERE PROF_TEACHES_INT.ProfLastName = @OldProfLName;
68.       UPDATE SECTION
69.         SET SECTION.ProfLastName = @NewProfLName
70.         WHERE SECTION.ProfLastName = @OldProfLName;
71.     END;
72.     -- If the Student's first name has changed, update STUDENT
73.     IF @NewStudFName != @OldStudFName
74.     BEGIN
75.       UPDATE STUDENT
76.         SET STUDENT.FirstName = @NewStudFName
77.         WHERE STUDENT.FirstName = @OldStudFName;
78.     END;
79.     -- If the Student's last name has changed, update STUDENT
80.     IF @NewStudLName != @OldStudLName
81.     BEGIN
82.       UPDATE STUDENT
83.         SET STUDENT.LastName = @NewStudLName
84.         WHERE STUDENT.LastName = @OldStudLName;
85.     END;
86.   END;
```

## Create Delete Trigger on Student-Professor-Course View

```
1.    CREATE TRIGGER dbo.TR_StudProfCourseDelete
2.      ON dbo.VW_StudProfCourse
3.      INSTEAD OF DELETE AS
4.    BEGIN
5.      -- Declare variables
6.      DECLARE @OldProfFName Varchar(129);
7.      DECLARE @OldProfLName Varchar(129);
8.      -- Initialize variables
9.      SELECT @OldProfFName = DELETED.ProfFirstName FROM DELETED;
10.     SELECT @OldProfLName = DELETED.ProfLastName FROM DELETED;
11.     -- Unassign all courses for that professor
12.     DELETE FROM PROF_TEACHES_INT
13.       WHERE
14.         PROF_TEACHES_INT.ProfFirstName = @OldProfFName AND
15.         PROF_TEACHES_INT.ProfLastName = @OldProfLName;
16.   END;
```

# Table Scripts

## Create STUDENT Table

```
1.    CREATE TABLE assign_3.dbo.STUDENT (
2.      StudentID Int IDENTITY(1,1) NOT NULL,
3.      FirstName Varchar(64) NOT NULL,
4.      LastName Varchar(64) NOT NULL,
5.      StreetAddress Varchar(128),
6.      Province Char(2),
7.      Country Varchar(32),
8.      AddressType Varchar(32),
9.      PhoneNo Char(10),
10.     PhoneType Varchar(32),
11.     Program Varchar(64) NOT NULL,
12.     CONSTRAINT PK_Stud PRIMARY KEY (StudentID),
13.     CONSTRAINT FK_StudProg FOREIGN KEY (Program) REFERENCES
        PROGRAM(ProgramName)
14.   );
```

## Add CHECK Constraint to PhoneNo

```
1.    ALTER TABLE assign_3.dbo.STUDENT
2.      ADD CHECK (PhoneNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-
        9][0-9]');
```

## Create PROFESSOR Table

```
1.    CREATE TABLE assign_3.dbo.PROFESSOR (
2.      FirstName Varchar(64) NOT NULL,
3.      LastName Varchar(64) NOT NULL,
4.      Salutation Varchar(4) DEFAULT 'Dr.' NOT NULL,
5.      Program Varchar(64) DEFAULT 'Data Science' NOT NULL,
6.      CONSTRAINT PK_Prof PRIMARY KEY (FirstName, LastName),
7.      CONSTRAINT FK_ProfProg FOREIGN KEY (Program) REFERENCES
        PROGRAM(ProgramName)
8.    );
```

## Add CHECK Constraint to Salutation

```
1.    ALTER TABLE assign_3.dbo.PROFESSOR
2.      ADD CHECK (Salutation IN ('Dr.', 'Ms.', 'Mr.', 'Mrs.', 'Miss'));
```

## Create Update Trigger on PROFESSOR

```
1.    CREATE TRIGGER dbo.TR_ProfNameUpdate
2.      ON dbo.PROFESSOR INSTEAD OF UPDATE AS
3.    BEGIN
4.      -- Create variables and initialize them
5.      DECLARE @OldFirstName Varchar(64);
6.      DECLARE @OldLastName Varchar(64);
7.      DECLARE @NewFirstName Varchar(64);
8.      DECLARE @NewLastName Varchar(64);
9.      DECLARE @NewSalutation Varchar(4);
10.     DECLARE @NewProgram Varchar(64);
11.     SELECT @OldFirstName = DELETED.FirstName FROM DELETED;
12.     SELECT @OldLastName = DELETED.LastName FROM DELETED;
13.     SELECT @NewFirstName = INSERTED.FirstName FROM INSERTED;
14.     SELECT @NewLastName = INSERTED.LastName FROM INSERTED;
15.     SELECT @NewSalutation = INSERTED.Salutation FROM INSERTED;
16.     SELECT @NewProgram = INSERTED.Program FROM INSERTED;
17.     -- Cascade update to SECTION table
18.     UPDATE SECTION
19.       SET
20.         SECTION.ProfFirstName = @NewFirstName,
21.         SECTION.ProfLastName = @NewLastName
22.       WHERE
23.         SECTION.ProfFirstName = @OldFirstName AND
24.         SECTION.ProfLastName = @OldLastName;
25.     -- Cascade update to PROF_TEACHES_INT table
26.     UPDATE PROF_TEACHES_INT
27.       SET
28.         PROF_TEACHES_INT.ProfFirstName = @NewFirstName,
29.         PROF_TEACHES_INT.ProfLastName = @NewLastName
30.       WHERE
31.         PROF_TEACHES_INT.ProfFirstName = @OldFirstName AND
32.         PROF_TEACHES_INT.ProfLastName = @OldLastName;
33.     -- Finally, complete update on PROFESSOR table
34.     UPDATE PROFESSOR
35.       SET
36.         PROFESSOR.FirstName = @NewFirstName,
37.         PROFESSOR.LastName = @NewLastName,
38.         PROFESSOR.Salutation = @NewSalutation,
39.         PROFESSOR.Program = @NewProgram
40.       WHERE
41.         PROFESSOR.FirstName = @OldFirstName AND
42.         PROFESSOR.LastName = @OldLastName;
43.   END;
```

## Create PROGRAM Table

```
1.    CREATE TABLE assign_3.dbo.PROGRAM (
2.      ProgramName Varchar(64) NOT NULL,
3.      Degree Varchar(64) NOT NULL,
4.      Department Varchar(64) NOT NULL,
5.      CONSTRAINT PK_Prog PRIMARY KEY (ProgramName)
6.    );
```

## Create Update Trigger on PROGRAM

```
1.    CREATE TRIGGER dbo.TR_ProgNameUpdate
2.      ON dbo.PROGRAM INSTEAD OF UPDATE AS
3.    BEGIN
4.      -- Create variables and initialize them
5.      DECLARE @OldProgName Varchar(64);
6.      DECLARE @NewProgName Varchar(64);
7.      DECLARE @NewDegree Varchar(64);
8.      DECLARE @NewDept Varchar(64);
9.      SELECT @OldProgName = DELETED.ProgramName FROM DELETED;
10.     SELECT @NewProgName = INSERTED.ProgramName FROM INSERTED;
11.     SELECT @NewDegree = INSERTED.Degree FROM INSERTED;
12.     SELECT @NewDept = INSERTED.Department FROM INSERTED;
13.     -- Cascade update to STUDENT table
14.     UPDATE STUDENT
15.       SET STUDENT.Program = @NewProgName
16.       WHERE STUDENT.Program = @OldProgName;
17.     -- Cascade update to PROFESSOR table
18.     UPDATE PROFESSOR
19.       SET PROFESSOR.Program = @NewProgName
20.       WHERE PROFESSOR.Program = @OldProgName;
21.     -- Cascade update to COURSE_PROG_INT table
22.     UPDATE COURSE_PROG_INT
23.       SET COURSE_PROG_INT.Program = @NewProgName
24.       WHERE COURSE_PROF_INT.Program = @OldProgName;
25.     – Finally, complete update on PROGRAM table
26.     UPDATE PROGRAM
27.       SET
28.         PROGRAM.ProgramName = @NewProgName,
29.         PROGRAM.Degree = @NewDegree,
30.         PROGRAM.Department = @newDept
31.       WHERE PROGRAM.ProgramName = @OldProgName;
32.   END;
```

## Create COURSE Table

```
1.    CREATE TABLE assign_3.dbo.COURSE (
2.      CourseName Varchar(128) NOT NULL,
3.      TuitionAmount SmallInt NOT NULL DEFAULT 0,
4.      CONSTRAINT PK_Course PRIMARY KEY (CourseName)
5.    );
```

## Create Update Trigger on COURSE

```
1.    CREATE TRIGGER dbo.TR_CourseNameUpdate
2.      ON dbo.COURSE INSTEAD OF UPDATE AS
3.    BEGIN
4.      -- Create variables and initialize them
5.      DECLARE @OldCourseName Varchar(64);
6.      DECLARE @NewCourseName Varchar(64);
7.      DECLARE @NewTuition Smallint;
8.      SELECT @OldCourseName = DELETED.CourseName FROM DELETED;
9.      SELECT @NewCourseName = INSERTED.CourseName FROM INSERTED;
10.     SELECT @NewTuition = INSERTED.TuitionAmount FROM INSERTED;
11.     -- Cascade update to SECTION table
12.     UPDATE SECTION
13.       SET SECTION.CourseName = @NewCourseName
14.       WHERE SECTION.CourseName = @OldCourseName;
15.     -- Cascade update to PROF_TEACHES_INT table
16.     UPDATE PROF_TEACHES_INT
17.       SET PROF_TEACHES_INT.Course = @NewCourseName
18.       WHERE PROF_TEACHES_INT.Course = @OldCourseName;
19.     -- Cascade update to COURSE_PROG_INT table
20.     UPDATE COURSE_PROG_INT
21.       SET COURSE_PROG_INT.Course = @NewCourseName
22.       WHERE COURSE_PROG_INT.Course = @OldCourseName;
23.     -- Finally, complete update on COURSE table
24.     UPDATE COURSE
25.       SET
26.         COURSE.CourseName = @NewCourseName,
27.         COURSE.TuitionAmount = @NewTuition
28.       WHERE COURSE.CourseName = @OldCourseName;
29.   END;
```

## Create SECTION Table

```
1.    CREATE TABLE assign_3.dbo.SECTION (
2.      SectionCode TinyInt NOT NULL,
3.      CourseName Varchar(128) NOT NULL,
4.      ProfFirstName Varchar(64),
5.      ProfLastName Varchar(64)
6.      CONSTRAINT PK_Section PRIMARY KEY (CourseName, SectionCode),
7.      CONSTRAINT FK_SectionOfCourse FOREIGN KEY (CourseName) REFERENCES
        COURSE(CourseName),
8.      CONSTRAINT FK_SectionProf FOREIGN KEY (ProfFirstName, ProfLastName)
        REFERENCES PROFESSOR(FirstName,LastName)
9.    );
```

## Create Insert Trigger on SECTION

```
1.    CREATE TRIGGER dbo.TR_SectionInsert
2.      ON dbo.SECTION AFTER INSERT AS
3.    BEGIN
4.      -- Declare variables
5.      DECLARE @CurrentSectCode Tinyint;
6.      DECLARE @NewSectCode Tinyint;
7.      DECLARE @NewCourse Varchar(128);
8.      -- Initialize variables
9.      SELECT @CurrentSectCode = INSERTED.SectionCode FROM INSERTED;
10.     SELECT @NewCourse = INSERTED.CourseName FROM INSERTED;
11.     -- Determine new section code
12.     SELECT * FROM SECTION WHERE CourseName = @NewCourse;
13.     IF @@ROWCOUNT = 1
14.       SELECT @NewSectCode = 1;
15.     ELSE
16.       SELECT @NewSectCode = 1 + MAX(SectionCode)
17.         FROM SECTION
18.         WHERE CourseName = @NewCourse;
19.     -- Finally, set the new SECTION's number
20.     UPDATE SECTION
21.       SET SECTION.SectionCode = @NewSectCode
22.       WHERE
23.         SECTION.SectionCode = @CurrentSectCode AND
24.         SECTION.CourseName = @NewCourse;
25.   END;
```

## Create Update Trigger on SECTION

```
1.    CREATE TRIGGER dbo.TR_SectionUpdate
2.      ON dbo.SECTION INSTEAD OF UPDATE AS
3.    BEGIN
4.      -- Declare variables
5.      DECLARE @OldSectCode Tinyint;
6.      DECLARE @OldCourse Varchar(128);
7.      DECLARE @OldProfFName Varchar(64);
8.      DECLARE @OldProfLName Varchar(64);
9.      DECLARE @SectCounter Tinyint;
10.     DECLARE @NewProfFName Varchar(64);
11.     DECLARE @NewProfLName Varchar(64);
12.     DECLARE @TempSectCode Tinyint;
13.     DECLARE @TempCourse Varchar(128);
14.     DECLARE @TempProfFName Varchar(64);
15.     DECLARE @TempProfLName Varchar(64);
16.     -- Ensure that only ONE row is being updated at a time
17.     SELECT * FROM INSERTED;
18.     IF @@ROWCOUNT > 1
19.       RETURN;
20.     -- Initialize variables
21.     SELECT @OldSectCode = DELETED.SectionCode FROM DELETED;
22.     SELECT @OldCourse = DELETED.CourseName FROM DELETED;
23.     SELECT @OldProfFName = DELETED.ProfFirstName FROM DELETED;
24.     SELECT @OldProfLName = DELETED.ProfLastName FROM DELETED;
25.     SELECT @NewProfFName = INSERTED.ProfFirstName FROM INSERTED;
26.     SELECT @NewProfLName = INSERTED.ProfLastName FROM INSERTED;
27.     SET @SectCounter = 1;
28.     -- Find all records with same course name and ensure sequential codes
29.     DECLARE Cursor1 CURSOR LOCAL FOR
30.       SELECT SectionCode, CourseName, ProfFirstName, ProfLastName
31.       FROM SECTION WHERE CourseName = @OldCourse;
32.     OPEN Cursor1;
33.     FETCH NEXT FROM Cursor1
34.       INTO @TempSectCode, @TempCourse, @TempProfFName, @TempProfLName;
35.     WHILE @@FETCH_STATUS = 0
36.     BEGIN
37.       UPDATE SECTION
38.         SET
39.           SECTION.SectionCode = @SectCounter,
40.           SECTION.CourseName = @TempCourse,
41.           SECTION.ProfFirstName = @TempProfFName,
42.           SECTION.ProfLastName = @TempProfLName
43.         WHERE
44.           SECTION.SectionCode = @TempSectCode AND
45.           SECTION.CourseName = @TempCourse AND
46.           SECTION.ProfFirstName = @TempProfFName AND
```

```
47.         SECTION.ProfLastName = @TempProfLName;
48.       SET @SectCounter = @SectCounter + 1;
49.       FETCH NEXT FROM Cursor1
50.         INTO @TempSectCode, @TempCourse, @TempProfFName, @TempProfLName;
51.     END;
52.     CLOSE Cursor1;
53.     DEALLOCATE Cursor1;
54.   END;
```

## Create Delete Trigger on SECTION

```
1.    CREATE TRIGGER dbo.TR_SectionDelete
2.      ON dbo.SECTION AFTER DELETE AS
3.    BEGIN
4.      -- Declare variables
5.      DECLARE @OldSectCode Tinyint;
6.      DECLARE @OldCourse Varchar(128);
7.      -- Initialize variables
8.      SELECT @OldSectCode = DELETED.SectionCode FROM DELETED;
9.      SELECT @OldCourse = DELETED.CourseName FROM DELETED;
10.     -- Update GRADE records with matching SectionCode
11.     UPDATE GRADE
12.       SET GRADE.SectionCode = NULL
13.       WHERE
14.         GRADE.SectionCode = @OldSectCode AND
15.         GRADE.CourseName = @OldCourse;
16.     -- Update records with higher SectionCodes
17.     WHILE (SELECT COUNT(*) FROM SECTION WHERE CourseName = @OldCourse AND
        SectionCode > @OldSectCode) > 0
18.     BEGIN
19.       UPDATE SECTION
20.         SET SECTION.SectionCode = @OldSectCode
21.         WHERE
22.           SECTION.CourseName = @OldCourse AND
23.           SECTION.SectionCode = @OldSectCode + 1;
24.       SELECT @OldSectCode = @OldSectCode + 1;
25.     END;
26.   END;
```

## Create GRADE Table

```
1.    CREATE TABLE assign_3.dbo.GRADE (
2.      StudentID Int NOT NULL,
3.      CourseName Varchar(128) NOT NULL,
4.      SectionCode TinyInt,
5.      PercentGrade Decimal(4,2) DEFAULT 0.0,
6.      LetterGrade Varchar(2) DEFAULT 'F',
7.      CONSTRAINT PK_Grade PRIMARY KEY (StudentID, CourseName),
8.      CONSTRAINT FK_StudentGrade FOREIGN KEY (StudentID) REFERENCES
        STUDENT(StudentID),
9.      CONSTRAINT FK_GradeForCourse FOREIGN KEY (CourseName, SectionCode)
        REFERENCES SECTION(CourseName, SectionCode)
10.   );
```

## Create Update Trigger on GRADE

```
1.    CREATE TRIGGER dbo.TR_GradeUpdate
2.      ON dbo.GRADE INSTEAD OF UPDATE AS
3.    BEGIN
4.      -- Declare variables
5.      DECLARE @OldStudID Int;
6.      DECLARE @OldCourse Varchar(128);
7.      DECLARE @OldSection Tinyint;
8.      DECLARE @NewStudID Int;
9.      DECLARE @NewCourse Varchar(128);
10.     DECLARE @NewSection Tinyint;
11.     DECLARE @NewPercent Decimal(4,2);
12.     DECLARE @NewLetter Varchar(2);
13.     -- Initialize variables
14.     SELECT @OldStudID = DELETED.StudentID FROM DELETED;
15.     SELECT @OldCourse = DELETED.CourseName FROM DELETED;
16.     SELECT @OldSection = DELETED.SectionCode FROM DELETED;
17.     SELECT @NewStudID = INSERTED.StudentID FROM INSERTED;
18.     SELECT @NewCourse = INSERTED.CourseName FROM INSERTED;
19.     SELECT @NewSection = INSERTED.SectionCode FROM INSERTED;
20.     SELECT @NewPercent = INSERTED.PercentGrade FROM INSERTED;
21.     EXEC @NewLetter = dbo.CalculateLetterGrade @Percent = @NewPercent;
22.     -- Check if corresponding SECTION has any GRADEs remaining (enrolled
        students)
23.     SELECT * FROM GRADE WHERE CourseName = @OldCourse AND SectionCode =
        @OldSection;
24.     IF @@ROWCOUNT = 0
25.       DELETE FROM SECTION WHERE SECTION.SectionCode = @OldSection AND
          SECTION.CourseName = @OldCourse;
26.     -- Finally, update the LetterGrade to match to PercentGrade
27.     UPDATE GRADE
28.       SET
29.         GRADE.StudentID = @NewStudID,
30.         GRADE.CourseName = @NewCourse,
31.         GRADE.SectionCode = @NewSection,
32.         GRADE.PercentGrade = @NewPercent,
33.         GRADE.LetterGrade = @NewLetter
34.       WHERE
35.         GRADE.StudentID = @OldStudID AND
36.         GRADE.CourseName = @OldCourse;
37.   END;
```

## Create COURSE_PROG_INT Table

```
1.    CREATE TABLE assign_3.dbo.COURSE_PROG_INT (
2.      Program Varchar(64) NOT NULL,
3.      Course Varchar(128) NOT NULL,
4.      CONSTRAINT PK_CourseProg PRIMARY KEY (Program, Course),
5.      CONSTRAINT FK_CoursesForProgram FOREIGN KEY (Program) REFERENCES
        PROGRAM(ProgramName),
6.      CONSTRAINT FK_ProgramHasCourses FOREIGN KEY (Course) REFERENCES
        COURSE(CourseName)
7.    );
```

## Create PROF_TEACHES_INT Table

```
1.    CREATE TABLE assign_3.dbo.PROF_TEACHES_INT (
2.      ProfFirstName Varchar(64) NOT NULL,
3.      ProfLastName Varchar(64) NOT NULL,
4.      Course Varchar(128) NOT NULL,
5.      CONSTRAINT PK_ProfTeaches PRIMARY KEY (ProfFirstName, ProfLastName,
        Course),
6.      CONSTRAINT FK_ProfTeaching FOREIGN KEY (ProfFirstName, ProfLastName)
        REFERENCES PROFESSOR(FirstName, LastName),
7.      CONSTRAINT FK_CourseTaught FOREIGN KEY (Course) REFERENCES
        COURSE(CourseName)
8.    );
```

## Export Database

```
1.    -- I have no idea how to write to files, but these are the SELECT
      statements that would be used within that logic
2.    -- COURSE Table
3.    SELECT * FROM dbo.COURSE;
4.    -- COURSE_PROG_INT Table
5.    SELECT * FROM dbo.COURSE_PROG_INT;
6.    -- GRADE Table
7.    SELECT * FROM dbo.GRADE;
8.    -- PROF_TEACHES_INT Table
9.    SELECT * FROM dbo.PROF_TEACHES_INT;
10.   -- PROFESSOR Table
11.   SELECT * FROM dbo.PROFESSOR;
12.   -- PROGRAM Table
13.   SELECT * FROM dbo.PROGRAM;
14.   -- SECTION Table
15.   SELECT * FROM dbo.SECTION;
16.   -- STUDENT Table
17.   SELECT * FROM dbo.STUDENT;
```

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

## Import Database

```
1.    -- Again, I have no idea how to read from files, but these are sample
      INSERT statements for each table
2.    -- COURSE Table
3.    INSERT INTO dbo.COURSE
4.          VALUES ('Underwater Basketweaving', 500);
5.    -- COURSE_PROG_INT Table
6.    INSERT INTO dbo.COURSE_PROG_INT
7.          VALUES ('Non-Terrestrial Arts', 'Underwater Basketweaving');
8.    -- GRADE Table
9.    INSERT INTO dbo.GRADE
10.         VALUES (1, 'Underwater Basketweaving', 1, 61.8, 'C-');
11.   -- PROF_TEACHES_INT Table
12.   INSERT INTO dbo.PROF_TEACHES_INT
13.         VALUES ('Ariel', 'Fisher', 'Underwater Basketweaving');
14.   -- PROFESSOR Table
15.   INSERT INTO dbo.PROFESSOR
16.         VALUES ('Ariel', 'Fisher', 'Dr.', 'Non-Terrestrial Arts');
17.   -- PROGRAM Table
18.   INSERT INTO dbo.PROGRAM
19.         VALUES ('Non-Terrestrial Arts', 'Bachelor of Silliness',
      'Humanities');
20.   -- SECTION Table
21.   INSERT INTO dbo.SECTION
22.         VALUES (1, 'Underwater Basketweaving', 'Ariel', 'Fisher');
23.   -- STUDENT Table
24.   INSERT INTO dbo.STUDENT
25.         VALUES ('Jim', 'Bobson', '123 Nonsense', 'NS', 'Canada', 'Home',
      '5555555555', 'Cell', 'Non-Terrestrial Arts');
```

## Delete All Data (Keeping Structure)

```
1.  -- Drop all foreign keys, primary keys and checks don't need to be
    dropped
2.  ALTER TABLE dbo.PROF_TEACHES_INT
3.    DROP CONSTRAINT
4.      FK_ProfTeaching,
5.      FK_CourseTaught;
6.  ALTER TABLE dbo.COURSE_PROG_INT
7.    DROP CONSTRAINT
8.      FK_CoursesForProgram,
9.      FK_ProgramHasCourses;
10. ALTER TABLE dbo.GRADE
11.   DROP CONSTRAINT
12.     FK_StudentGrade,
13.     FK_GradeForCourse;
14. ALTER TABLE dbo.SECTION
15.   DROP CONSTRAINT
16.     FK_SectionOfCourse,
17.     FK_SectionProf;
18. ALTER TABLE dbo.STUDENT
19.   DROP CONSTRAINT FK_StudProg;
20. ALTER TABLE dbo.PROFESSOR
21.   DROP CONSTRAINT FK_ProfProg;
22. -- Wipe tables clean
23. TRUNCATE TABLE dbo.COURSE;
24. TRUNCATE TABLE dbo.COURSE_PRO_INT;
25. TRUNCATE TABLE dbo.GRADE;
26. TRUNCATE TABLE dbo.PROF_TEACHES_INT;
27. TRUNCATE TABLE dbo.PROFESSOR;
28. TRUNCATE TABLE dbo.PROGRAM;
29. TRUNCATE TABLE dbo.SECTION;
30. TRUNCATE TABLE dbo.STUDENT;
31. -- Reinstate foreign keys
32. ALTER TABLE dbo.PROF_TEACHES_INT
33.   ADD CONSTRAINT FK_ProfTeaching
34.     FOREIGN KEY (ProfFirstName, ProfLastName)
35.     REFERENCES PROFESSOR(FirstName, LastName),
36.   CONSTRAINT FK_CourseTaught
37.     FOREIGN KEY (Course)
38.     REFERENCES COURSE(CourseName);
39. ALTER TABLE dbo.COURSE_PROF_INT
40.   ADD CONSTRAINT FK_CoursesForProgram
41.     FOREIGN KEY (Program)
42.     REFERENCES PROGRAM(ProgramName),
43.   CONSTRAINT FK_ProgramHasCourses
44.     FOREIGN KEY (Course)
45.     REFERENCES COURSE(CourseName);
```

```
46.  ALTER TABLE dbo.GRADE
47.    ADD CONSTRAINT FK_StudentGrade
48.      FOREIGN KEY (StudentID)
49.      REFERENCES STUDENT(StudentID),
50.    CONSTRAINT FK_GradeForCourse
51.      FOREIGN KEY (CourseName, SectionCode)
52.      REFERENCES SECTION(CourseName, SectionCode);
53.  ALTER TABLE dbo.SECTION
54.    ADD CONSTRAINT FK_SectionOfCourse
55.      FOREIGN KEY (CourseName)
56.      REFERENCES COURSE(CourseName),
57.    CONSTRAINT FK_SectionProf
58.      FOREIGN KEY (ProfFirstName, ProfLastName)
59.      REFERENCES PROFESSOR(FirstName,LastName);
60.  ALTER TABLE dbo.STUDENT
61.    ADD CONSTRAINT FK_StudProg
62.      FOREIGN KEY (Program)
63.      REFERENCES PROGRAM(ProgramName);
64.  ALTER TABLE dbo.PROFESSOR
65.    ADD CONSTRAINT FK_ProfProg
66.      FOREIGN KEY (Program)
67.      REFERENCES PROGRAM(ProgramName);
```

## Procedure/Function Scripts

### Calculate Letter Grade from Percent Grade

```
1.    CREATE FUNCTION dbo.CalculateLetterGrade
2.      (@Percent Decimal(4,2))
3.    RETURNS Varchar(2)
4.    AS
5.    BEGIN
6.      IF (@Percent >= 90.0)
7.        RETURN 'A+';
8.      IF (@Percent >= 85.0)
9.        RETURN 'A';
10.     IF (@Percent >= 80.0)
11.       RETURN 'A-';
12.     IF (@Percent >= 77.0)
13.       RETURN 'B+';
14.     IF (@Percent >= 73.0)
15.       RETURN 'B';
16.     IF (@Percent >= 70.0)
17.       RETURN 'B-';
18.     IF (@Percent >= 67.0)
19.       RETURN 'C+';
20.     IF (@Percent >= 63.0)
21.       RETURN 'C';
22.     IF (@Percent >= 60.0)
23.       RETURN 'C-';
24.     IF (@Percent >= 57.0)
25.       RETURN 'D+';
26.     IF (@Percent >= 53.0)
27.       RETURN 'D';
28.     IF (@Percent >= 50.0)
29.       RETURN 'D-';
30.     RETURN 'F';
31.   END;
```

### Section Reassignment Based on Last Name

```
1.    CREATE PROCEDURE dbo.SectionReassignment
2.      (@CourseName Varchar(128),
3.      @ProfFName Varchar(64),
4.      @ProfLName Varchar(64))
5.    AS
6.    BEGIN
7.      -- Declare variables
8.      DECLARE @CurrentSection Int;
9.      DECLARE @CurrentStudent Int;
10.     DECLARE @TempID Int;
```

```
11.     DECLARE @TempCourse Varchar(128);
12.     DECLARE @TempFirstName Varchar(64);
13.     DECLARE @TempLastName Varchar(64);
14.     -- Before reassigning students, delete all existing sections
15.     DELETE FROM SECTION WHERE CourseName = @CourseName;
16.     DECLARE EnrolledCursor CURSOR LOCAL FOR
17.       SELECT
18.         GRADE.StudentID,
19.         GRADE.CourseName,
20.         STUDENT.FirstName,
21.         STUDENT.LastName
22.       FROM dbo.GRADE
23.       INNER JOIN dbo.STUDENT
24.       ON GRADE.StudentID = STUDENT.StudentID
25.       WHERE GRADE.CourseName = @CourseName
26.       ORDER BY STUDENT.LastName, STUDENT.FirstName;
27.     -- Open cursor and begin reading rows
28.     OPEN EnrolledCursor;
29.     FETCH NEXT FROM EnrolledCursor
30.       INTO
31.         @TempID,
32.         @TempCourse,
33.         @TempFirstName,
34.         @TempLastName;
35.     SET @CurrentSection = 1;
36.     WHILE @@FETCH_STATUS = 0
37.     BEGIN
38.       -- Create a new section before assigning students
39.       INSERT INTO dbo.SECTION
40.         VALUES (@CurrentSection, @CourseName, @ProfFName, @ProfLName);
41.       SET @CurrentStudent = 0;
42.       WHILE @CurrentStudent < 30
43.       BEGIN
44.         UPDATE dbo.GRADE
45.           SET GRADE.SectionCode = @CurrentSection,
46.           WHERE
47.             GRADE.StudentID = @TempID AND
48.             GRADE.CourseName = @TempCourse;
49.         SET @CurrentStudent = @CurrentStudent + 1;
50.       END;
51.       SET @CurrentSection = @CurrentSection + 1;
52.     END;
53.     CLOSE EnrolledCursor;
54.     DEALLOCATE EnrolledCursor;
55.   END;
```

# Reports

## Program Enrollment Report

```
1.    SELECT Program, COUNT(*)
2.      FROM dbo.STUDENT
3.      GROUP BY Program;
```

## Course Enrollment Report

```
1.    SELECT
2.      COURSE_PROG_INT.Program,
3.      COURSE_PROG_INT.Course,
4.      COUNT(enrolled.StudentID) AS NumberOfStudents,
5.      SUM(enrolled.TuitionAmount) AS Tuition
6.    FROM COURSE_PROG_INT
7.    INNER JOIN
8.      (SELECT
9.        COURSE.CourseName,
10.       COURSE.TuitionAmount,
11.       GRADE.StudentID
12.     FROM COURSE
13.     INNER JOIN GRADE
14.     ON COURSE.CourseName = GRADE.CourseName
15.     ) enrolled
16.   ON COURSE_PROG_INT.Course = enrolled.CourseName
17.   GROUP BY ROLLUP(COURSE_PROG_INT.Course, COURSE_PROG_INT.Program);
```

## Program/Course Enrollment Report

```
1.    SELECT
2.      COURSE_PROG_INT.Program,
3.      COURSE_PROG_INT.Course,
4.      COUNT(enrolled.StudentID) AS NumberOfStudents,
5.      SUM(enrolled.TuitionAmount) AS Tuition
6.    FROM COURSE_PROG_INT
7.    INNER JOIN
8.      (SELECT
9.      COURSE.CourseName,
10.     COURSE.TuitionAmount,
11.     GRADE.StudentID
12.     FROM COURSE
13.     INNER JOIN GRADE
14.     ON COURSE.CourseName = GRADE.CourseName
15.     ) enrolled
16.   ON COURSE_PROG_INT.Course = enrolled.CourseName
17.   GROUP BY CUBE(COURSE_PROG_INT.Course, COURSE_PROG_INT.Program);
```

ASSIGNMENT 3: DATABASE CONSTRUCTION AND PROCESSING

## Course/Program Matrix

```
1.    SELECT * FROM dbo.COURSE_PROG_INT
2.    PIVOT (
3.      COUNT(COURSE_PROG_INT.Course)
4.      FOR COURSE_PROG_INT.Program IN
5.        (SELECT Program FROM COURSE_PROG_INT)
6.    )
7.    ORDER BY COURSE_PROG_INT.Course;
```

## Student Grade Average

```
1.    SELECT
2.      stud.StudentName,
3.      COUNT(CourseName) AS NumOfCourses,
4.      AVG(PercentGrade) AS GradeAverage
5.    FROM GRADE
6.    INNER JOIN
7.      (SELECT
8.        CONCAT_WS(' ', STUDENT.FirstName, STUDENT.LastName) AS StudentName,
9.        StudentID
10.     FROM STUDENT) stud
11.   ON GRADE.StudentID = stud.StudentID
12.   GROUP BY stud.StudentName;
```

# Request Scripts

## Change PhoneType's Default Value

```
1.    ALTER TABLE dbo.STUDENT
2.      ADD CONSTRAINT DF_PhoneType DEFAULT 'Not Provided' FOR PhoneType;
3.    UPDATE dbo.STUDENT
4.      SET PhoneType = 'Not Provided'
5.      WHERE
6.        PhoneType IS NULL OR
7.        PhoneNo IS NULL;
```

## Change SectionCode Default Value

```
1.    ALTER TABLE dbo.SECTION
2.      ADD CONSTRAINT DF_SectCode DEFAULT 450 FOR SectionCode;
3.    -- SectionReassignment procedure and SECTION triggers would have to be
      re-written
4.    -- After re-writing SectionReassignment, it could be called on all
      courses
```

## Change TuitionAmount's Datatype

```
1.    ALTER TABLE dbo.COURSE
2.      ALTER COLUMN TuitionAmount Decimal(6,2);
```

## Change SectionCode Column Contents

```
1.    ALTER TABLE dbo.SECTION
2.      DROP CONSTRAINT PK_Section;
3.    ALTER TABLE dbo.SECTION
4.      ADD CourseSection Varchar(150);
5.    MERGE INTO dbo.SECTION AS S1 USING dbo.SECTION AS S2
6.      ON (S1.SectionCode = S2.SectionCode
7.      AND S1.CourseName = S2.CourseName)
8.      WHEN MATCHED THEN
9.        UPDATE SET S1.CourseSection =
10.         CONCAT_WS(' - Section ', S1.CourseName, S1.SectionCode);
11.   ALTER TABLE dbo.SECTION
12.     ADD CONSTRAINT PK_Section PRIMARY KEY (CourseSection);
13.   ALTER TABLE dbo.SECTION
14.     DROP COLUMN SectionCode;
15.   ALTER TABLE dbo.SECTION
16.     DROP COLUMN CourseName;
```

# Appendix 1

- For the Cardinality and Action charts I omitted the intersection tables from the relationships. The intersection tables are relationships, so describing cardinality/action between an entity and its relationship to another entity (rather than to another entity) seems odd.
- I realize the UPDATE Trigger on the SECTION table will not work. It's probably causing some sort of recursive infinite loop, and not actually updating ProfFirstName or ProfLastName at all. Unfortunately I ran out of time and energy.
- Everything has been done in SQL Server, I ran out of time to look into the differences for Oracle and MySQL.