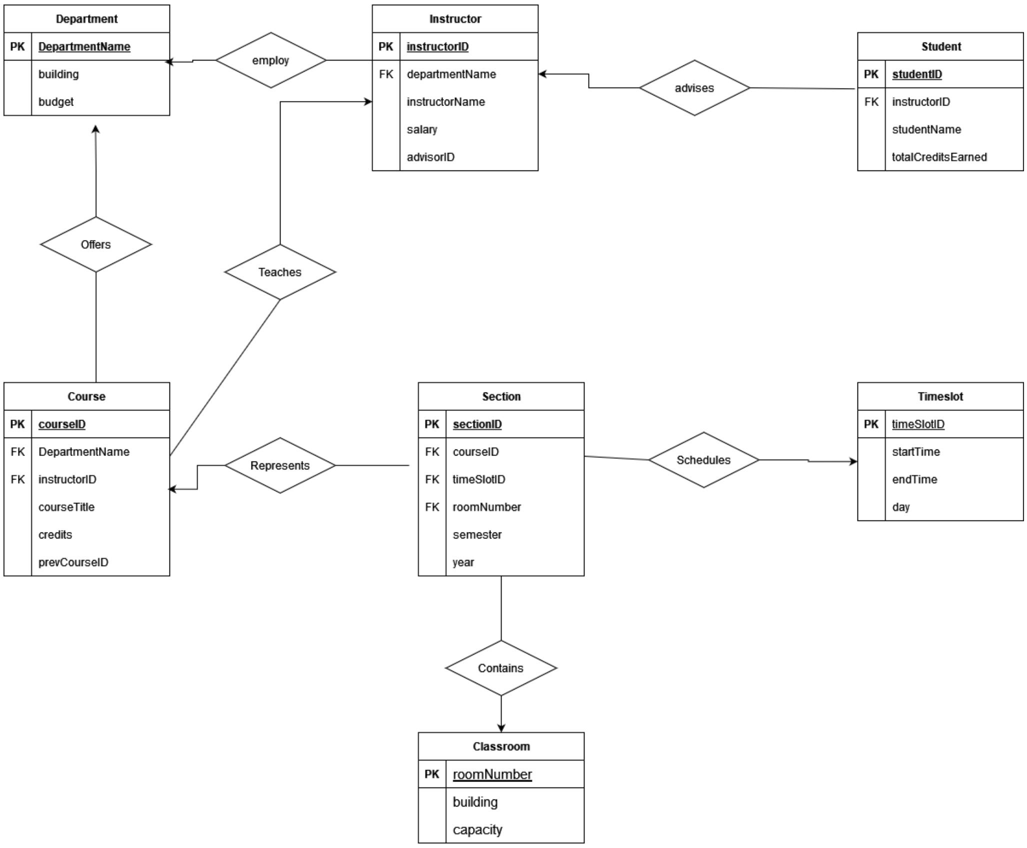
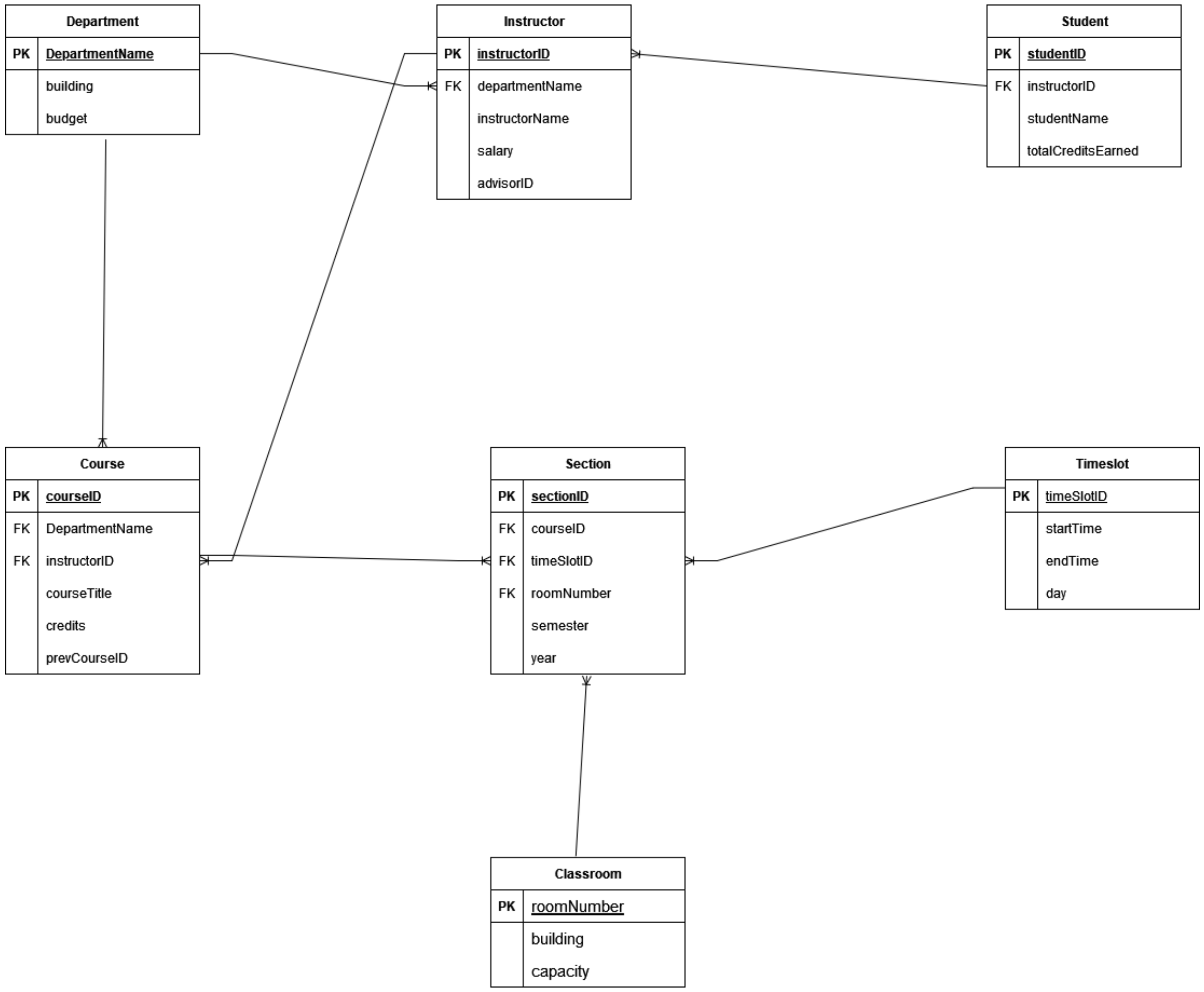
1.)

- Department(departmentName(PK), building, budget)
- Instructor(instructorID(PK), instructorName, salary, departmentName(FK), advisorID)
- Student(studentID(PK), studentName, totalCreditsEarned, instructorID(FK))
- Course(courseID(PK), courseTitle, credits, prevCourseID, instructorID(FK))
- Section(sectionID(PK), courseID(FK), semester, year, timeSlotID(FK), roomNumber(FK))
- TimeSlot(timeSlotID(PK), startTime, endTime, day)
- Classroom(building, roomNumber(PK), capacity)

2.)



3.a)



3.b)

Example: Instructor Table
Instructor(instructorID, instructorName, salary, departmentName, advisorID)

The instructor table is both 1NF and 3NF. It can be 1NF because the instructor table does not contain any repeating data in the column; containing atomic values/attributes and a primary key.

It is also 3NF because it starts off as 2NF and there are no transitive dependencies, meaning that any non-key attribute should only depend on the primary key. For example, salary does not depend on any other non-key attributes and only the primary key.

Example: Section Table
Section(sectionID, courseID, semester, year, timeSlotID, roomNumber)

The section table starts off with 1NF structure and all non-key attributes are fully functionally dependent entirely on the primary key(s). This primarily applies to tables with composite

keys, but since all the functional dependencies only have one primary key, this automatically meets the requirement for 2NF.
The section table is also in 3rd normal form because there are no transitive dependencies. A table is in 3NF if it's in 2NF and all non-key attributes are directly dependent on the primary key, with no dependencies between non-key attributes. An example of this is the many foreign keys in the table. For example if credits was in the section table then it wouldn't be 3NF because the credits attribute wouldn't be dependent on the primary key (sectionID) and instead on another key(courseID). This removes transitive dependencies and ensures all non-key attributes depend directly on the primary key(sectionID), satisfying 3NF.

Example Course Table:
Course(courseID(PK), courseTitle, credits, prevCourseID, instructorID(FK))

1NF requires that each table's attributes contain only atomic (indivisible) values, meaning no repeating groups or arrays within a field. Each column should have a single value per row, and each record should be unique. Each couseID identifies a specific course and there are no repeating fields.
2NF requires that all non-key attributes must spend solely on the primary key and nothing else, this is seen above as everything is reliant on courseID.
3NF requires that there are no transitive dependencies and there are no partial keys; courses follow this as course ID is the only key it is reliant on.

4.)

```
CREATE database UniSystem;
USE UniSystem;

CREATE TABLE DEPARTMENT (
DepartmentName VARCHAR(150) PRIMARY KEY,
Building VARCHAR(50) NOT NULL,
Budget numeric(25) NOT NULL);

CREATE TABLE INSTRUCTOR (
InstructorID VARCHAR(10) PRIMARY KEY,
InstructorName VARCHAR(35) NOT NULL,
Salary numeric(10) NOT NULL,
AdvisorID VARCHAR(10) NOT NULL,
DepartmentName VARCHAR(150) NOT NULL,
CONSTRAINT FK_ADVISORTO foreign key (AdvisorID)
references INSTRUCTOR(InstructorID),
CONSTRAINT FK_DEPART_INSTRUCT foreign key (DepartmentName)
references DEPARTMENT(DepartmentName));

CREATE TABLE STUDENT (
StudentID VARCHAR(10) PRIMARY KEY,
StudentName VARCHAR(35) NOT NULL,
totalCredsEarned numeric(5) default 0 NOT NULL,
InstructorID VARCHAR(10) NOT NULL,
CONSTRAINT FK_STUDENT_INSTRUCT foreign key (InstructorID)
references INSTRUCTOR(InstructorID));

CREATE TABLE COURSE (
CourseID VARCHAR(10) PRIMARY KEY,
CourseTitle VARCHAR(25) NOT NULL,
Credits Numeric(5) NOT NULL,
prevCourseID VARCHAR(10),
DepartmentName VARCHAR(150) NOT NULL,
InstructorID VARCHAR(10) NOT NULL,
CONSTRAINT FK_prevCourse foreign key (prevCourseID)
references COURSE(CourseID),
CONSTRAINT FK_DEPART_COURSE foreign key (DepartmentName)
references DEPARTMENT(DepartmentName),
CONSTRAINT FK_INSTUCT_COURSE foreign key (InstructorID)
references INSTRUCTOR(InstructorID));

CREATE TABLE TIMESLOT (
timeSlotID VARCHAR(10) PRIMARY KEY,
startTime TIME(0) NOT NULL,
endTime TIME(0) NOT NULL,
day VARCHAR(15) NOT NULL);

CREATE TABLE CLASSROOM (
roomNumber VARCHAR(10) PRIMARY KEY,
building VARCHAR(50) NOT NULL,
capacity int(3) NOT NULL);

CREATE TABLE SECTION (
sectionID VARCHAR(10) PRIMARY KEY,
semester VARCHAR(10) NOT NULL,
year int(5) NOT NULL,
courseID VARCHAR(10) NOT NULL,
timeSlotID VARCHAR(10) NOT NULL,
roomNumber VARCHAR(10) NOT NULL,
CONSTRAINT FK_SECT_COURSE foreign key (CourseID)
references COURSE(CourseID),
CONSTRAINT FK_SECT_TIME foreign key (timeSlotID)
references TIMESLOT(timeSlotID),
CONSTRAINT FK_SECT_ROOM foreign key (roomNumber)
references CLASSROOM(roomNumber));
```

5)

1)
```
SELECT InstructorName, CourseID
From INSTRUCTOR i
inner join COURSE c ON i.InstructorID = c.InstructorID;
```

2)
```
SELECT InstructorName, CourseTitle
From INSTRUCTOR
natural join COURSE;
```

3)
```
SELECT InstructorName
From INSTRUCTOR
WHERE Salary > 90000 AND Salary < 100000;
```

4)
```
SELECT CourseID, CourseTitle
From SECTION
NATURAL JOIN COURSE
WHERE Year = "2009" AND Semester = "Fall"
OR Year = "2010" AND Semester = "Spring";
```

5)
```
SELECT COUNT(InstructorID), DepartmentName
From INSTRUCTOR
NATURAL JOIN COURSE
NATURAL JOIN SECTION
WHERE semester = "Spring" AND year = 2010
GROUP BY DepartmentName;
```

6)
```
SELECT AVG(totalCredsEarned), semester
FROM STUDENT
NATURAL JOIN DEPARTMENT
NATURAL JOIN COURSE
NATURAL JOIN SECTION
WHERE year = 2009
GROUP BY semester;
```

7)
```
SELECT *
FROM COURSE
WHERE CourseID IN (
SELECT CourseID
FROM SECTION
WHERE (semester = "Fall" AND year = 2009))
AND CourseID NOT IN (
SELECT CourseID
FROM SECTION
WHERE (semester = "Spring" AND year = 2010));
```

8)
```
SELECT DEPARTMENT.DepartmentName, AVG(INSTRUCTOR.Salary)
FROM INSTRUCTOR
```

```
INNER JOIN DEPARTMENT ON INSTRUCTOR.DepartmentName =
DEPARTMENT.DepartmentName
GROUP BY DEPARTMENT.DepartmentName
HAVING AVG(INSTRUCTOR.Salary) >= ALL(
SELECT AVG(Salary)
FROM INSTRUCTOR);
```

```
9)  UPDATE STUDENT
    SET totalCredsEarned = (
    select SUM(COURSE.Credits)
    FROM COURSE
    INNER JOIN INSTRUCTOR ON COURSE.InstructorID = INSTRUCTOR.InstructorID
    WHERE COURSE.InstructorID = Student.InstructorID);
```

```
10) SELECT STUDENT.StudentID, SECTION.sectionID, SECTION.CourseID
    FROM SECTION
    INNER JOIN COURSE ON SECTION.CourseID = COURSE.CourseID
    INNER JOIN INSTRUCTOR ON COURSE.InstructorID = INSTRUCTOR.InstructorID
    INNER JOIN STUDENT ON INSTRUCTOR.InstructorID = STUDENT.InstructorID
    WHERE COURSE.DepartmentName = "Comp.Sci"
    AND SECTION.semester = "Spring"
    AND SECTION.year = 2009;
```

6)

Shaun	Stan Identified the functional dependencies in the database. Fiachra made both the ER diagram, the specific database structure diagram and helped Stan with explaining the normalised forms. I made the database and the queries. Fiachra created some test data and I helped him test my SQL queries. Stan(32%), Shaun(35%), Fiachra(33%)
Stan	Shaun ensured that the SQL queries were functional for the appropriate functional dependencies and diagram (ERD) both for table creation and data manipulation. Fiachra has helped with constructing the Entity Relationship Diagram based on what I wrote for the Functional Dependencies. In addition, Fiachra and I completed the database structure including Normal Form examples and explanations. Shaun(35%) Stan(32.5%), Fiachra(32.5%)
Fiachra Fiachra Bohan	Shaun created the database and wrote the queries, while I helped him test his SQL queries. Stan identified the functional dependencies in the database. I also created the ER diagram, the specific database structure diagram, and some test data. Stan(32%), Shaun(34.9%), Fiachra(33.1%)