**Task 1:**

**Functional Dependencies**

Department(departmentName (PK), building, budget)

**Functional Dependencies:**

departmentName → {building, budget}

Instructor(instructorID (PK), name, salary, departmentName (FK), advisoryID (FK))

**Functional Dependencies:**

instructorID → {name, salary, departmentName, adisoryID}

advisorID → instructorID

Student(studentID (PK), name, totalCredits, advisorID (FK))

**Functional Dependencies:**

studentID → {name, totalCredits, advisorID}

advisorID → studentID

Course(courseID (PK), title, credits, departmentName (FK))

**Functional Dependencies:**

courseID → {title, credits, departmentName}

departmentName → courseID

TimeSlot(timeSlotID (PK), startTime, endTime, dayOfWeek)

**Functional Dependencies:**

timeSlotID → {startTime, endTime, dayOfWeek}

Section(sectionID (PK), semester, year, courseID (FK), timeSlotID (FK), classroomID (FK))

**Functional Dependencies:**

sectionID → {semester, year, courseID, timeSlotID, classroomID}

courseID → sectionID

Classroom(classroomID (PK), building, roomNO, capacity)

**Functional Dependencies:**

classroomID → building, roomNO, capacity

{building, roomNO} → {classroomID, capacity}

Prerequisite(courseID (PK), prerequisiteID (FK))

**Functional Dependencies:**

courseID → prerequisiteID

Enrollment(studentID (FK), sectionID (FK))

**Functional Dependencies:**

N/A

**Task 2:**

[University Database Design - ER Diagram.drawio.png](https://uniofleicester-my.sharepoint.com/:i:/g/personal/kp422_student_le_ac_uk/EfWi9pMpUN9Cqi_ho1-VNHkBfcCA4iPP7iuV0hpsPusr6Q?e=cgL0CD)

**Task 3:**

**1)**

[University Database Design - Database Structure Diagram.drawio.png](https://uniofleicester-my.sharepoint.com/:i:/g/personal/kp422_student_le_ac_uk/EVXRU8DUmHJPrE7Sw6TrEhUBkxXLYd0OfOGK5G2_7Wq6XA?e=lV2hQe)

**2)**

Below I have linked PNGs for the ERD in both 1NF and 2NF.

**1NF:** All entities within the diagram contain atomic attributes meaning each attribute within the database contains a single value.

For example, the Classroom entity only contains attributes that holds one value.

[University Database Design - ER Diagram 1NF.drawio.png](https://uniofleicester-my.sharepoint.com/:i:/g/personal/kp422_student_le_ac_uk/ERY81mNlhHtHs2mkLowD7QcBQsdjBP4_GIAm7IqbdfW-eg?e=ZO0b1r)

**2NF:** Diagram is already in 1NF, and every non-key attribute is functionally dependent on the whole primary key. Each table either has a primary key or a composite primary key.

For example, the Enrollment and Prerequisite tables use composite keys, and each non-key attribute is fully dependent on both parts of the composite key. Additionally, every other table has a primary key.

[University Database Design - ER Diagram 2NF.drawio.png](https://uniofleicester-my.sharepoint.com/:i:/g/personal/kp422_student_le_ac_uk/EV0Kpta8rrpHgQJfiMGrgb4BiTUJKJcPUA8540xxIRpUHA?e=eSf5mE)

**3NF:** Diagram is already in 2NF, and all non-key attributes are non-transitively dependent on the primary key meaning that there’s not a chain of more than 2 functionally dependent attributes. For example, attribute A determines the value of attribute B but attribute B doesn’t determine the value of attribute C.

Each non-key attribute for every entity in the diagram is directly dependent on the primary key and non on any other non-key attribute. For example, in the Instructor table, all non-key attribute are directly dependent on the primary key, instructorID.

**Task 4:**

CREATE DATABASE uniproject;

USE uniproject;

CREATE TABLE Department (

departmentName VARCHAR(20),

building VARCHAR(20),

budget DECIMAL(10, 2),

PRIMARY KEY (departmentName)

);

CREATE TABLE Instructor (

instructorID VARCHAR(20),

name VARCHAR(20),

salary DECIMAL(10, 2),

departmentName VARCHAR(20),

advisorID VARCHAR(20) NULL,

PRIMARY KEY (instructorID),

FOREIGN KEY (departmentName) REFERENCES Department(departmentName),

FOREIGN KEY (advisorID) REFERENCES Instructor(instructorID)

);

CREATE TABLE Student (

studentID VARCHAR(20),

name VARCHAR(20),

totalCredits INT,

advisorID VARCHAR(20),

PRIMARY KEY (studentID),

FOREIGN KEY (advisorID) REFERENCES Instructor(instructorID)

);

CREATE TABLE Course (

courseID VARCHAR(20),

title VARCHAR(50),

credits INT,

departmentName VARCHAR(20),

PRIMARY KEY (courseID),

FOREIGN KEY (departmentName) REFERENCES Department(departmentName)

);

CREATE TABLE Prerequisite (

courseID VARCHAR(20),

prerequisiteID VARCHAR(20),

PRIMARY KEY (courseID, prerequisiteID),

FOREIGN KEY (courseID) REFERENCES Course(courseID),

FOREIGN KEY (prerequisiteID) REFERENCES Course(courseID)

);

CREATE TABLE TimeSlot (

timeSlotID VARCHAR(20),

startTime TIME,

endTime TIME,

dayOfWeek VARCHAR(10),

PRIMARY KEY (timeSlotID)

);

CREATE TABLE Classroom (

classroomID VARCHAR(20),

building VARCHAR(20),

roomNO INT,

capacity INT,

PRIMARY KEY (classroomID)

);

CREATE TABLE Section (

sectionID VARCHAR(20),

semester VARCHAR(10),

year INT,

courseID VARCHAR(20),

timeSlotID VARCHAR(20),

classroomID VARCHAR(20),

PRIMARY KEY (sectionID),

FOREIGN KEY (courseID) REFERENCES Course(courseID),

FOREIGN KEY (timeSlotID) REFERENCES TimeSlot(timeSlotID),

FOREIGN KEY (classroomID) REFERENCES Classroom(classroomID)

);

CREATE TABLE Enrollment (

studentID VARCHAR(20),

sectionID VARCHAR(20),

PRIMARY KEY (studentID, sectionID),

FOREIGN KEY (studentID) REFERENCES Student(studentID),

FOREIGN KEY (sectionID) REFERENCES Section(sectionID)

);

-- Select statements to view table contents

SELECT \* FROM Department;

SELECT \* FROM Instructor;

SELECT \* FROM Student;

SELECT \* FROM Course;

SELECT \* FROM Prerequisite;

SELECT \* FROM Section;

SELECT \* FROM TimeSlot;

SELECT \* FROM Classroom;

SELECT \* FROM Enrollment;

**Task 5:**

1. **For all instructors in the university who have taught some courses, find their names and the course ID of all courses they taught.**

SELECT Instructor.name, Section.courseID

FROM Instructor

JOIN Section ON Instructor.instructorID = Section.instructorID;

1. **List the names of instructors along with the titles of courses that they teach (using NATURAL JOIN).**

SELECT Instructor.name, Course.title

FROM Instructor

NATURAL JOIN Section

NATURAL JOIN Course;

1. **Find the names of instructors with salary amounts between $90,000 and $100,000.**

SELECT name

FROM Instructor

WHERE salary BETWEEN 90000 AND 100000;

1. **Find the set of all courses taught either in Fall 2009 or in Spring 2010, or both.**

SELECT DISTINCT courseID

FROM Section

WHERE (semester = 'Fall' AND year = 2009)

OR (semester = 'Spring' AND year = 2010);

1. **Find the number of instructors in each department who teach a course in the Spring 2010 semester.**

SELECT Instructor.departmentName, COUNT(DISTINCT Instructor.instructorID) AS numInstructors

FROM Instructor

JOIN Section ON Instructor.instructorID = Section.instructorID

WHERE Section.semester = 'Spring' AND Section.year = 2010

GROUP BY Instructor.departmentName;

1. **For each course section offered in 2009, find the average total credits (tot\_cred) of all students enrolled in the section, if the section had at least 2 students.**

SELECT Section.sectionID, AVG(Student.totalCredits) AS avgCredits

FROM Section

JOIN Enrollment ON Section.sectionID = Enrollment.sectionID

JOIN Student ON Enrollment.studentID = Student.studentID

WHERE Section.year = 2009

GROUP BY Section.sectionID

HAVING COUNT(Student.studentID) >= 2;

1. **Find all the courses taught in the Fall 2009 semester but not in the Spring 2010 semester (using NOT IN).**

SELECT courseID

FROM Section

WHERE semester = 'Fall' AND year = 2009

AND courseID NOT IN (

SELECT courseID

FROM Section

WHERE semester = 'Spring' AND ye**ar = 2010**

**);**

1. **Find the departments that have the highest average salary. (Use the ALL keyword).**

SELECT departmentName

FROM Instructor

GROUP BY departmentName

HAVING AVG(salary) >= ALL (

SELECT AVG(salary)

FROM Instructor

GROUP BY departmentName

);

1. **Consider an update where we set the tot\_cred attribute of each student tuple to the sum of the credits of courses successfully completed by the student.**

UPDATE Student

SET totalCredits = (

SELECT SUM(Course.credits)

FROM Enrollment

JOIN Course ON Enrollment.courseID = Course.courseID

WHERE Enrollment.studentID = Student.studentID

AND Enrollment.status = 'completed' -- Assuming a status column indicating completion

);

1. **Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections from Spring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section.**

SELECT Student.name, Section.sectionID

FROM Section

LEFT JOIN Enrollment ON Section.sectionID = Enrollment.sectionID

LEFT JOIN Student ON Enrollment.studentID = Student.studentID

WHERE Section.semester = 'Spring' AND Section.year = 2009

AND (Student.departmentName = 'Comp. Sci.' OR Student.departmentName IS NULL);

**Task 6:**

Kairo contributed 33.3% to the project. He designed the ERD for the database as well as the database’s structure using a Database Structure Diagram.

Michael contributed 33.3% He programmed the database system using the ERD and Database Structure Diagram using mySQL.

John contributed 33.3%. He wrote the SQL queries to complete the questions for task 5.