**LABWORK 1**

**PART 1.**

Task 1.1

Relation: Registration(StudentID, CourseCode, Section, Semester, Year, Grade, Credits)

Minimum attributes for primary key:

StudentID + CourseCode + Section + Semester + Year

Why each attribute is needed:

StudentID: identify the student.

CourseCode: identify the course.

Section: the same course can have many sections.

Semester + Year: same course and section can repeat in different semesters/years. Together they guarantee one row per actual registration.

Additional candidate keys: If the system has a unique SectionID (section identifier that already encodes semester+year), then {StudentID, SectionID} could be a candidate. But with given attributes the full set above is the safe primary key.

Task 1.2

Tables: Student(StudentID, Name, Email, Major, AdvisorID)

Professor(ProfID, Name, Department, Salary)

Course(CourseID, Title, Credits, DepartmentCode)

Department(DeptCode, DeptName, Budget, ChairID)

Enrollment(StudentID, CourseID, Semester, Grade)

Foreign key relationships (list):

Student.AdvisorID → Professor.ProfID

Professor.Department → Department.DeptCode (or Professor.DepartmentCode → Department.DeptCode)

Course.DepartmentCode → Department.DeptCode

Department.ChairID → Professor.ProfID

Enrollment.StudentID → Student.StudentID

Enrollment.CourseID → Course.CourseID (Each arrow means “references”.)

**PART 4.**

Task 4.1

Table: StudentProject(StudentID, StudentName, StudentMajor, ProjectID, ProjectTitle, ProjectType, SupervisorID, SupervisorName, SupervisorDept, Role, HoursWorked, StartDate, EndDate)

**1) Functional dependencies (FDs)**

StudentID → StudentName, StudentMajor

ProjectID → ProjectTitle, ProjectType, StartDate, EndDate

SupervisorID → SupervisorName, SupervisorDept

ProjectID → SupervisorID *(assume each project has a supervisor)*

(StudentID, ProjectID) → Role, HoursWorked

**2) Problems / Redundancy / Anomalies**

Redundancy: StudentName and StudentMajor repeat for the same StudentID over many rows. SupervisorName repeats for many projects supervised by same supervisor. ProjectTitle repeats for many students on same project.

Update anomaly: If SupervisorName changes, we must update many rows. If we miss one row, data is inconsistent.

Insert anomaly: We cannot insert a new Supervisor without a student-project row (if schema forced Supervisor in same row).

Delete anomaly: If the last student leaves a project and we delete the row, we lose Project and Supervisor info.

**3) 1NF check and fix**

Table is in 1NF if all attributes atomic. If Role or Phone had multiple values in one cell, fix by splitting rows. Here we assume atomic, so it is 1NF.

**4**) 2NF

Primary key: (StudentID, ProjectID) (one student works on many projects; one project has many students).

Partial dependencies: StudentName, StudentMajor depend only on StudentID (part of the key). ProjectTitle depends only on ProjectID.

Decompose to 2NF:

Tables after 2NF decomposition:

1. Student(StudentID PK, StudentName, StudentMajor)
2. Project(ProjectID PK, ProjectTitle, ProjectType, StartDate, EndDate, SupervisorID)
3. Supervisor(SupervisorID PK, SupervisorName, SupervisorDept)
4. Participation(StudentID FK, ProjectID FK, Role, HoursWorked) — PK = (StudentID, ProjectID)

**5) 3NF**

Check transitive dependencies: In Project table SupervisorID → SupervisorName if we kept SupervisorName there. We moved SupervisorName to Supervisor table, so no transitive dependencies remain.

Final 3NF tables (same as above). All FKs shown:

Student(StudentID PK, StudentName, StudentMajor)

Supervisor(SupervisorID PK, SupervisorName, SupervisorDept)

Project(ProjectID PK, ProjectTitle, ProjectType, StartDate, EndDate, SupervisorID FK → Supervisor.SupervisorID)

Participation(StudentID FK → Student.StudentID, ProjectID FK → Project.ProjectID, Role, HoursWorked) PK=(StudentID,ProjectID)

This is clean and avoids anomalies.

Task 4.2

Table: CourseSchedule(StudentID, StudentMajor, CourseID, CourseName, InstructorID, InstructorName, TimeSlot, Room, Building)

Business rules recap:

Each student has exactly one major. → StudentID → StudentMajor

Each course has a fixed name. → CourseID → CourseName

Each instructor has exactly one name. → InstructorID → InstructorName

Each time slot in a room determines the building. (Room → Building) (rooms unique across campus)

Each course section taught by one instructor at one time in one room.

A student can enroll in many course sections.

**1) Primary key (tricky)**

We need a key that identifies one enrollment row. I choose (StudentID, CourseID, TimeSlot) as the primary key. Reason: a student enrolls in a specific course section at a specific time. CourseID alone is not unique because same course can have many sections; TimeSlot and Room help identify section.

(Alternative: if there is SectionID, then (StudentID, SectionID) is simpler.)

**2) Functional dependencies (FDs)**

StudentID → StudentMajor

CourseID → CourseName

InstructorID → InstructorName

Room → Building *(rooms unique across campus)*

(CourseID, TimeSlot) → InstructorID, Room *(a course section at a time has one instructor and one room)*

From above we also get (CourseID, TimeSlot) → Building by composition.

**3) Is table in BCNF?**

No. Example violation: Room → Building is a non-trivial FD where Room is not a superkey of the whole table. So not BCNF.

Also StudentID → StudentMajor is FD where StudentID is not a superkey. So not BCNF.

**4) Decompose to BCNF (step by step)**

We decompose into these relations:

1. Student(StudentID PK, StudentMajor)

From FD: StudentID → StudentMajor

1. Course(CourseID PK, CourseName)
2. Instructor(InstructorID PK, InstructorName)
3. Room(Room PK, Building)
4. Section(CourseID, TimeSlot, InstructorID FK, Room FK)

PK = (CourseID, TimeSlot)

This table stores that a course at a time has one instructor and one room.

1. Enrollment(StudentID FK, CourseID, TimeSlot)

PK = (StudentID, CourseID, TimeSlot)

References Section(CourseID, TimeSlot) and Student(StudentID)

This decomposition removes FDs that break BCNF. Each table has keys that determine other attributes.

**5) Potential loss of information**

There is **no loss** of information if we keep all FKs and PKs correctly. We can join Enrollment with Section and Student to recreate the original rows. This is a lossless decomposition.