Part 1 – Concept list **(Task 1)**

Important concepts for a university course registration system include: Student, Course, Course Section, Instructor, Teaching Assistant, Department, Semester/Term, Course Credit (local credit and ECTS), Course Delivery Mode (in-person, online, hybrid), Registration (enrollment in a section), Prerequisite, Co-requisite, Mutually Exclusive Courses or Course Equivalence, Grade, Failed Course, Course Repeat, Grade Replacement Policy, Course Replacement, Add Action, Drop Action, Withdrawal (W grade), Forced Withdrawal or Administrative Drop, Registration Credit Limit, Time Conflict between Sections, Registration Eligibility (active status, disciplinary and financial status), Registration Priority, Override or Special Permission, Academic Advisor Approval, Registration History or Action Log, and Academic Program Requirements and Course Sequences.

Part 2 – Minimum system concepts **(Task 2)**

In the minimum system without any rules, only four concepts are required: Student, Course, Course Section, and Registration. Students represent the people who take courses. Courses represent abstract offerings such as “ECON 101”. Course Sections represent concrete offerings of a course in a specific semester with a section number and schedule. Registrations represent the relationship that a student is enrolled in a particular section. More advanced concepts like prerequisites, withdrawals, credit limits, or overrides are not needed in this simplified version because there are no restrictions or history tracking.

Part 3a – 2NF table design **(Task 3a)**

The following 2NF design is used for the basic system.

Students

- StudentID (PK)
- StudentNumber
- FirstName

- LastName
- Major
- EntryYear

Courses

- CourseID (PK)
- CourseCode
- CourseTitle
- LocalCredits
- ECTS

Sections

- SectionID (PK)
- CourseID (FK → Courses.CourseID)
- Semester
- Year
- SectionNumber
- Capacity
- InstructorName

Registrations

- RegistrationID (PK)
- StudentID (FK → Students.StudentID)
- SectionID (FK → Sections.SectionID)
- RegistrationDate
- Courses – Sections: 1-to-many
- Students – Registrations: 1-to-many
- Sections – Registrations: 1-to-many
- Students – Sections: many-to-many, Registrations ile çözülüyor.

Part 3b – Description **(PK, FK, 2NF) (Task 3b)**

The primary key of Students is StudentID because each student needs a unique and stable identifier. The primary key of Courses is CourseID, which uniquely identifies each course independently of its code or title. The primary key of Sections is SectionID because each section is a specific offering of a course in a particular semester and year. The primary key of Registrations is RegistrationID, giving each registration record a unique identifier.

CourseID is used as a foreign key in Sections to link each section to its underlying course. StudentID and SectionID are used as foreign keys in Registrations to connect each registration to a specific student and a specific section. The relationship between Courses and Sections is one-to-many, since a single course may have multiple sections but each section belongs to exactly one course. The relationship between Students and Sections is many-to-many and is implemented via the Registrations table, which serves as a bridge between Students and Sections.

This design satisfies Second Normal Form because all non-key attributes in each table depend on the whole primary key and there are no partial dependencies. Each table uses a single-column surrogate key as its primary key, and attributes such as student names, course titles, or instructor names depend only on their respective primary keys. There are no repeating groups, and multi-valued relationships are modeled using separate tables rather than multiple columns.

Part 4a – Compelling constraints **(prereq / co-req / chains)**

In a real system, constraints such as prerequisites, co-requisites, and required course chains are important for maintaining academic quality and proper sequencing. Prerequisites ensure that students have the necessary background knowledge before taking advanced courses. Co-requisites force students to take lecture and laboratory components together when both are required for learning outcomes. Required chains, such as a sequence of mathematics or programming courses, prevent students from skipping essential foundation material. To support these constraints, the database would need additional tables and fields, for example a Prerequisites table that stores pairs of courses and a MinimumGrade field that records the required grade level. The system must store which courses are required before others and what grade thresholds apply, as well as each student's past course completions and grades to evaluate whether the constraints

are satisfied. These additional structures are not required in the simplified system from Part 2 because that design does not enforce any registration rules, and therefore does not need to check or store prerequisite relationships.

Part 4b – Limiting constraints (time conflict, credit load, mutually exclusive)

Limiting constraints such as time conflict checks, maximum credit load, and mutually exclusive course rules matter because they protect both the academic process and the logistics of scheduling. Time conflict rules prevent students from enrolling in sections that overlap in meeting time. Maximum credit limits keep students from taking an unsustainable workload in a single semester. Mutually exclusive courses stop students from earning double credit for courses that cover the same content. To enforce these rules, the database needs extra structures, such as a Schedule table that stores meeting days and times for each section, and a MutualExclusion or CourseEquivalence table that stores pairs of courses that cannot both count toward graduation. The system must also know each student's current registered credits in a semester to check credit limits before confirming new registrations. These rules were omitted from the basic system in Part 2 to keep the design simple and focused only on core registration functionality, so detailed schedule and restriction tables are not necessary in that version.

Part 4c – History

A real registration system needs to keep a detailed history of actions to support auditing, conflict resolution, and appeals. Records of add attempts, drops, withdrawals, overrides, forced withdrawals, and time conflict approvals provide essential evidence when students or administrators question how a final registration state was reached. To store this information, the database would typically include an ActionLog table with fields such as a unique log ID, student ID, action type, timestamp, course ID, section ID, and optional notes or approval data. Additional columns can record who approved an override or the reason for an administrative withdrawal. This requirement makes the system more complex because every registration-related action must generate a history record and some queries must combine current registration data with past events. In the simplified

design in Part 3, history is not included so that the model remains small and easier to understand, focusing only on current registrations instead of full temporal behavior.