ECON485 – Homework 1

Part 1 – Identifying Key Concepts

Reading the university's registration regulations, several core ideas appear repeatedly in the description of how students choose and manage courses. These ideas include both simple data objects and more complex process-related concepts that the information system has to track.

Important concepts for a course registration platform include:

- Student profile
- Degree program / major track
- Academic term and academic year
- Course definition in the catalog
- Individual course section (specific class offering)
- Teaching staff (instructors, co-instructors, assistants)
- Credit values (local and ECTS)
- Enrollment record (linking student to section)
- Prerequisite rules
- Co-requisite rules
- Mutually exclusive or equivalent courses
- Grade records and transcripts
- Repeated course attempts
- Grade improvement / replacement policies
- Add and drop operations
- Voluntary withdrawal
- Administrative or enforced withdrawal
- Time and room schedule for sections
- Maximum credit load rules
- Registration priority and time windows
- Overrides and special approvals
- Academic advising interactions
- Action history and audit logs

This list captures the main academic entities, the constraints that connect them, and the administrative events that occur throughout the registration process.

Part 2 – Core Concepts for the Minimal System

For the simplified version of the registration system, all rules and limits are intentionally removed. There are no prerequisite checks, no time conflict checks, no maximum credit limits, and no tracking of add/drop history. Under these assumptions, the system only needs to support four fundamental functions: creating students, defining courses, offering sections, and recording which student attends which section.

In this minimal environment, only the following concepts are strictly necessary:

1. **StudentRecord** – represents each person who can sign up for classes.
2. **CourseUnit** – represents each catalog course, such as "ECON 101 – Introduction to Economics."
3. **SectionOffering** – represents an instance of a course in a particular term with a section code and teacher.
4. **EnrollmentLink** – represents the fact that a given student attends a given section.

Other ideas such as prerequisites, co-requisites, schedule management, or repeated attempts are not included, because the stripped-down system never blocks registration and does not enforce any academic rules.

Part 3a – 2NF Table Design and ER Diagram

Using the minimum set of concepts above, the database design for the basic system can be represented with four tables. Field names are chosen to be clear and distinct from other versions.

Table: StudentRecords

- StudentID (PK) – internal identifier
- UniversityID – institutional student number
- FirstName

- LastName
- ProgramCode – degree program or department
- CohortYear – first year of enrollment

Table: CatalogCourses

- CatalogID (PK)
- CourseCode – e.g. “ECON101”
- CourseName – descriptive title
- LocalCredits
- EctsCredits

Table: OfferedSections

- OfferedID (PK)
- CatalogID (FK → CatalogCourses.CatalogID)
- TermLabel – e.g. “Fall” or “Spring”
- YearValue – integer year
- SectionLabel – e.g. “01”, “A2”
- CapacityLimit – maximum number of seats
- MainInstructor – name of the instructor

Table: RegistrationsBasic

- RegID (PK)
- StudentID (FK → StudentRecords.StudentID)
- OfferedID (FK → OfferedSections.OfferedID)
- RegisteredOn – date of registration

The relationships are:

- One catalog course can have many offered sections (1:N).
- One student can attend many offered sections, and each offered section can have many students; this many-to-many relationship is handled by the RegistrationsBasic table.

In an ER diagram, StudentRecords, CatalogCourses, and OfferedSections are separate entities, and RegistrationsBasic appears as a linking entity connecting students to sections.

Part 3b – Explanation of Keys, Relationships, and 2NF

Primary keys use artificial integer identifiers to avoid depending on human-facing codes. StudentRecords uses StudentID rather than the university number, so the university can change numbering conventions without affecting foreign keys. CatalogCourses uses CatalogID instead of CourseCode, and OfferedSections uses OfferedID instead of a composite of term, year, and section label. RegistrationsBasic has its own RegID to uniquely identify each enrollment row.

Foreign keys are chosen to reflect natural connections between the entities:

- CatalogID in OfferedSections points back to the course definition in CatalogCourses.
- StudentID and OfferedID in RegistrationsBasic reference the student and the section that the enrollment belongs to.

This produces a one-to-many relationship from CatalogCourses to OfferedSections, and a many-to-many relationship between StudentRecords and OfferedSections resolved by RegistrationsBasic. All non-key columns in each table depend on the full primary key and not on part of it, because the primary keys are simple (single-column) surrogate keys. There are no repeating groups; for example, multiple sections are represented as multiple rows in OfferedSections rather than as repeated columns. For these reasons, the design satisfies Second Normal Form.

Part 4a – Constraints that Force or Enable Course Registration

In the real system, some rules are designed to ensure that students build knowledge in a logical order. Prerequisite constraints require successful completion of one or more courses before taking an advanced course, while co-requisite constraints require simultaneous enrollment, such as a lecture and an associated lab. Certain programs also define mandatory course chains where foundational courses must be taken before intermediate and advanced ones.

To support these types of rules, the database needs to store explicit prerequisite relationships and grade outcomes. One common approach is to introduce a **CoursePrerequisites** table with fields such as RuleID, TargetCatalogID, RequiredCatalogID, and MinimumAllowedGrade. A separate **CompletedCourseRecords** table (or an extended registration table with a Grade column) stores which student finished which course and what grade was obtained. With this information, the application can evaluate whether all required courses have been completed with acceptable grades before allowing registration into the target course.

These additional tables are not part of the minimal system because the basic model never automatically checks whether a student is prepared for a course. The simplified design focuses only on recording enrollments and assumes that all registrations are allowed, so prerequisite-related structures are unnecessary there.

Part 4b – Constraints that Limit What Students Can Register For

Real-world registration processes also contain rules that prevent students from choosing certain combinations of classes. Time conflict rules ensure that a student does not enroll in two sections that meet at overlapping times. Maximum credit load rules limit how many local or ECTS credits a student can take in a single term, often based on academic standing. Some courses are mutually exclusive because they cover similar material or belong to alternative tracks.

Supporting these rules requires additional structures in the database. A **SectionSchedule** table could store, for each OfferedID, the specific days and time ranges when the class meets. A **MutualExclusionPairs** table could list pairs of CatalogIDs that cannot be taken together. A **CreditPolicy** or **EnrollmentLimits** configuration could define upper bounds on total credits per term. When a student attempts to register, the application would need to query these tables to detect schedule overlaps, credit overloads, or disallowed course combinations before confirming the enrollment.

These limiting rules are excluded from the minimal design because they add complexity to both the schema and the registration logic. For the basic educational example in Homework 1, the goal is to understand core entities and relationships, not to build a full rules engine.

Part 4c – The Need to Keep a History of Actions

A fully functional registration system must maintain a detailed history of what has happened over time, not just the current state of enrollments. Historical information about add attempts, drops, withdrawals, overrides, section changes, and time conflict approvals is important for audits, disputes, and advising decisions. Without such logs, administrators cannot easily reconstruct why a student ended up in or out of a particular course.

To store this history, the database could use an **RegistrationEvents** table with fields such as EventID, StudentID, EventType, CatalogID, OfferedID, Timestamp, and a free-text Details field. Some implementations would further normalize the metadata into separate tables, but even a single event table already introduces more joins, more writing activity, and more storage. Every time a student interacts with the system, an additional row must be added to the event log, which increases the volume of data and the complexity of queries that need to reconstruct timelines.

This historical requirement is left out of the simplified design in Part 3 because the introductory model focuses on the present configuration of students, courses, and sections. Including a full event history from the start would make the ER diagram more complicated and distract from the learning objective of practicing basic normalization and key selection.