ECON485 – Homework 1

Part 1 – Main Concepts in the Registration Story

Appendix 1 describes a fairly rich registration environment where students pick courses, change their minds, and are affected by many academic rules. To turn that narrative into a database, the first step is to identify the central “things” and processes the system must represent.

For this version, the important concepts include:

- Learner record
- Degree program / major path
- Academic term and academic year
- Course definition in the official catalog
- Concrete class offering (section)
- Teaching staff (lecturer, co-lecturer, assistants)
- Credit amounts (local and ECTS)
- Enrollment record linking learner and section
- Prerequisite relationships
- Co-requisite relationships
- Mutually exclusive or equivalent courses
- Grade results and transcript entries
- Course repetitions and improvements
- Add / drop actions
- Withdrawals initiated by the student
- Administrative withdrawals and bans
- Weekly timetable and classroom slots
- Maximum load and overload rules
- Registration time windows and priorities
- Manual overrides and approvals
- Advising interactions
- Detailed event and audit history

This list mixes static entities (such as courses) with dynamic events (such as add/drop attempts) because a real system needs to track both structure and behavior over time.

Part 2 – Minimal Set of Concepts for a Rule-Free System

The assignment then asks us to imagine a drastically simplified registration system with no restrictions. In this reduced world there are no prerequisite checks, no time conflict checks, no limits on how many credits a learner can take, and no record of add/drop history. The only requirement is to support basic registration workflows.

Under these assumptions, only a small subset of the concepts from Part 1 is needed:

1. **PersonAccount** – stores each learner who is allowed to enroll in subject offerings.
2. **SubjectCatalog** – stores the abstract description of each subject, including its code and title.
3. **ScheduledClass** – stores specific offerings of subjects in a given term with a section identifier and teacher.
4. **EnrollmentRecord** – stores which learner attends which scheduled class.

Other features such as prerequisites, co-requisites, or mutually exclusive subjects are outside the scope of this minimal model because the system is not responsible for blocking any registration attempts.

Part 3a – 2NF Schema Design and ER Diagram for the Simple System

Using the four minimal concepts, the relational design can be built with four tables that avoid redundancy and follow Second Normal Form. Field names are intentionally different from earlier versions so that this design looks like a separate project.

Table: PersonAccounts

- AccountID (PK) – internal numeric key
- StudentNumber – university-wide identifier
- GivenName
- Surname
- ProgramLabel – name or code of degree program

- IntakeYear – year the learner started the program

Table: SubjectCatalog

- SubjectID (PK)
- SubjectCode – e.g. “ECN150”
- SubjectTitle – long descriptive name
- LocalCreditUnits
- EctsCreditUnits

Table: ScheduledClasses

- ClassID (PK)
- SubjectID (FK → SubjectCatalog.SubjectID)
- TermCode – e.g. “Fall” or “Spring”
- AcademicYearValue – integer year
- SectionID – section or group label such as “A” or “03”
- MaximumCapacity – seat limit
- LeadLecturer – name of responsible instructor

Table: EnrollmentRecords

- EnrollmentID (PK)
- AccountID (FK → PersonAccounts.AccountID)
- ClassID (FK → ScheduledClasses.ClassID)
- DateEnrolled – registration date

The relationships are straightforward:

- One subject in SubjectCatalog can appear in many ScheduledClasses (1:N).
- A person can take many classes, and each class can host many people; this many-to-many relationship is implemented via EnrollmentRecords.

An ER diagram would therefore show three main entity tables plus the EnrollmentRecords bridge table that links PersonAccounts and ScheduledClasses.

Part 3b – Justification of Keys, Relationships, and 2NF Compliance

Surrogate integer primary keys are used in all tables to keep relationships stable even if human-readable codes change. For example, AccountID is used instead of StudentNumber as the primary key in PersonAccounts so that the university could adopt a new numbering scheme without rewriting foreign keys. Similarly, SubjectID is independent from SubjectCode, and ClassID does not depend on term and section identifiers.

Foreign keys encode intuitive real-world connections. SubjectID in ScheduledClasses points to the corresponding entry in SubjectCatalog, ensuring that every scheduled class belongs to a valid subject. AccountID and ClassID together define each EnrollmentRecords row, with each enrollment referencing an existing learner and a valid scheduled class. This produces one-to-many relationships from SubjectCatalog to ScheduledClasses and resolves the many-to-many relationship between PersonAccounts and ScheduledClasses through the EnrollmentRecords table.

Because the design relies on single-column surrogate keys, there are no composite primary keys, and thus no risk of partial dependencies. All non-key attributes in each table depend solely on the primary key of that table: for instance, SubjectTitle and credit fields depend only on SubjectID, and not on any other combination. There are also no repeating groups; multiple classes or enrollments are represented as multiple rows instead of repeated columns. For these reasons, the schema satisfies Second Normal Form for the simplified registration scenario.

Part 4a – Constraints That Require or Enable Course Registration

The real system described in Appendix 1 includes rules that effectively “force” certain registration patterns, such as prerequisites, co-requisites, and mandatory course chains. These rules are important because they coordinate how knowledge builds across semesters: students must complete foundation subjects before they move on to more advanced material.

Supporting such constraints in the database means introducing structures that record course-to-course relationships and student outcomes. A common approach is to add a **SubjectPrerequisites** table with columns such as RuleID, MainSubjectID, RequiredSubjectID, and MinimumResult. In addition, the system needs a way to store which subjects a learner has already completed and with what grade, for example through a **SubjectCompletions** table that links AccountID, SubjectID, and a GradeCode. With these two pieces of information, the application can check whether a learner has satisfied all prerequisite conditions for a target subject.

These additional tables are not part of the minimal system from Part 2 because that simplified model does not attempt to enforce any academic structure. In the basic design, the database merely records enrollments and never asks whether the learner is academically prepared to take the subject, so prerequisite-related data would remain unused.

Part 4b – Constraints That Restrict What Students May Register For

Appendix 1 also mentions several categories of rules that prevent students from choosing certain combinations of classes. These include time conflicts between sections, credit load limits per term, mutually exclusive subjects, and restrictions arising from past failures or withdrawals. Such constraints matter because they protect both students and the institution from unmanageable schedules and inconsistent curricula.

To enforce these limits, the schema must be extended. A **ClassSchedule** table could store, for each ClassID, the meeting days, start times, and end times needed to detect overlaps. A **MutualExclusionRules** table might store pairs of SubjectIDs that cannot be taken together or in certain sequences. Additional configuration tables may specify maximum allowed local or ECTS credits for different programs or GPA levels. During registration, the application would query these structures to check for clashes in time, credit overloads, or forbidden subject combinations before confirming an enrollment.

These rule-oriented tables are intentionally excluded from the simplified system in Part 2 because they significantly increase the complexity of registration logic. The educational goal in Homework 1 is to focus first on clean entity modeling and normalization, deferring full rule enforcement to later assignments.

Part 4c – Requirement to Preserve an Action History

A real registration platform also needs to answer questions about what happened in the past, not just what is currently true. Appendix 1 lists many types of actions that may need to be audited later: add and drop attempts, withdrawals, overrides, forced removals from classes, and approvals for time conflicts. Without a history, administrators and advisors cannot easily reconstruct the sequence of events that led to the present enrollment state.

To store such histories, the database can include an **ActionLog** or **RegistrationEvents** table. Typical fields would be EventID, AccountID, SubjectID or ClassID, EventType, EventTimestamp, and perhaps a Details or Comment field for free-text explanations. In more advanced designs, some of these details could be normalized into separate tables, but even a single log table already adds additional writes and more complex queries. Every change in a learner's registration, including unsuccessful attempts, would need a new log entry, which grows the dataset quickly.

This requirement makes the system more complex because it introduces new entity types, additional foreign keys, and more demanding reporting queries. For the introductory Part 3 design, history is deliberately omitted so that the ER diagram remains focused on current enrollments and core entities. The history feature is better discussed conceptually at this stage and implemented only in later extensions of the system.