ECON485 Fall 2025 – Homework Assignment 1

Course Registration System – Database Design

PART 1: Extracting Concepts (Business Objects)

Based on careful analysis of Appendix 1, here is a comprehensive list of important concepts for a course registration system:

Core Academic Entities

- **Course** - The fundamental instructional unit
- **Course Code** - Unique identifier for courses (e.g., ECON 211)
- **Course Title** - Descriptive name of the course
- **Credit Value** - Both local credits and ECTS
- **Section** - Specific offerings of a course in a semester
- **Section Number** - Identifier for different sections of the same course
- **Syllabus** - Course content description

People

- **Student** - Individuals enrolling in courses
- **Instructor** - Faculty teaching courses
- **Co-Instructor** - Additional teaching faculty
- **Teaching Assistant** - Support staff for courses
- **Academic Advisor** - Faculty guiding student course selection
- **Department Chair** - Administrative authority
- **Program Coordinator** - Undergraduate program manager

Academic Structures

- **Department** - Academic organizational unit offering courses
- **Program** - Degree program (major)
- **Semester** - Time period for course offerings
- **Academic Year** - Yearly academic cycle

Registration Concepts

- **Registration** - Student enrollment in a section
- **Enrollment Capacity** - Maximum students per section
- **Student Status** - Active, suspended, etc.
- **CGPA (Cumulative GPA)** - Student's overall grade point average
- **Credit Load** - Total credits student is taking
- **Registration Priority** - Determines registration order

Course Relationships

- **Prerequisite** - Course that must be completed before another
- **Co-requisite** - Course that must be taken simultaneously
- **Cross-Listed Course** - Same course under multiple department codes
- **Joint Course** - Collaboratively offered course
- **Course Equivalence** - Mutually exclusive courses
- **Mutually Exclusive Courses** - Courses where only one counts for credit

Grading and Academic Progress

- **Grade** - Letter grade earned (A, B, C, D, F, etc.)
- **Minimum Grade Requirement** - Threshold for prerequisite satisfaction
- **Failed Course** - Course with F, FX, or U grade
- **Grade Replacement** - Policy for repeated courses
- **Transcript** - Official record of all courses and grades
- **Pass/Fail Course** - Alternative grading mode

Scheduling

- **Meeting Time** - When a section meets
- **Meeting Days** - Days of the week for class
- **Classroom** - Physical location for section
- **Online Meeting Link** - Virtual classroom location
- **Time Conflict** - Overlapping section schedules
- **Schedule** - Complete timetable

Registration Periods and Actions

- **Add/Drop Period** - Weeks 1-2 when changes are free
- **Withdrawal Period** - Weeks 3-10 when W grade is assigned
- **Add Action** - Adding a course to registration
- **Drop Action** - Removing a course during add/drop
- **Withdrawal** - Leaving course after add/drop period
- **Section Change** - Moving between sections of same course

History and Tracking

- **Registration History** - Complete record of all registration actions
- **Action Log** - Timestamped record of system actions
- **Action Type** - Category of registration action
- **Timestamp** - Date and time of action
- **Administrative Drop** - Forced removal from course
- **Forced Withdrawal** - Involuntary course exit

Special Permissions and Overrides

- **Departmental Override** - Permission to bypass rules
- **Instructor Consent** - Required approval from instructor
- **Time Conflict Approval** - Permission for overlapping schedules
- **Overload Permission** - Approval to exceed credit limits
- **Conditional Registration** - Registration pending prerequisite completion

Constraints and Limits

- **Credit Limit** - Maximum ECTS allowed per semester
- **Minimum Credit Load** - Required minimum credits
- **Attendance Policy** - Required class attendance rules
- **Enrollment Restriction** - Limits on who can register
- **Financial Restriction** - Holds on account preventing registration
- **Disciplinary Suspension** - Temporary ban from registration

Multi-Component Course Elements

- **Lecture** - Theoretical instruction component
- **Laboratory** - Hands-on practical component
- **Recitation** - Small group discussion sessions
- **Project Studio** - Project-based work sessions

Additional Concepts

- **Delivery Mode** - In-person, online, or hybrid
- **Graduation Requirements** - Courses needed to complete degree
- **Common Core Curriculum** - University-wide required courses
- **Course Repeat Attempts** - Number of times course has been taken
- **Academic Probation** - Warning status for poor performance
- **Exchange Student** - Visiting student from another institution
- **Scholarship Student** - Student with financial award
- **Final Semester** - Last term before graduation

PART 2: Selecting Basic Concepts (Minimum System)

For a very simple course registration system with **no restrictions, no rules, no history tracking**, we only need the following concepts:

Selected Basic Concepts:

1. **Student** - People who register for courses
2. **Course** - The instructional units being offered
3. **Section** - Specific offerings of courses (since one course can have multiple sections)
4. **Registration** - The link between students and the sections they're enrolled in
5. **Instructor** - Faculty assigned to teach sections (optional but useful)
6. **Semester** - Time period context (optional but helps organize data)

Why These Concepts Are Sufficient:

This minimal set allows us to:

- Create and store course information
- Create and store student information

- Offer multiple sections of the same course
- Link students to the sections they want to attend
- Track which instructor teaches which section
- Organize offerings by semester

What We're Excluding:

- Prerequisites, co-requisites, and all course dependencies
- Credit limits and overload permissions
- Time conflicts and schedule validation
- Add/drop history and action logging
- Grades and grade replacement policies
- Withdrawals and forced drops
- Mutually exclusive courses
- Registration priority and approval workflows
- All constraint checking and rule enforcement

PART 3a: 2NF Design and ER Diagram (Basic System)

Table Designs

1. STUDENTS Table

```
STUDENTS
---------
StudentID (PK)        INT
FirstName             VARCHAR(50)
LastName              VARCHAR(50)
Email                 VARCHAR(100)
StudentNumber         VARCHAR(20)
EnrollmentYear        INT
```

2. COURSES Table

```
COURSES
-------
CourseID (PK)          INT
CourseCode             VARCHAR(20)
CourseTitle            VARCHAR(200)
Credits                INT
ECTSCredits            INT
Department             VARCHAR(100)
Description            TEXT
```

3. SEMESTERS Table

```
SEMESTERS
---------
SemesterID (PK)        INT
SemesterName           VARCHAR(50)
AcademicYear           VARCHAR(10)
StartDate              DATE
EndDate                DATE
```

4. INSTRUCTORS Table

```
INSTRUCTORS
-----------
InstructorID (PK)      INT
FirstName              VARCHAR(50)
LastName               VARCHAR(50)
Email                  VARCHAR(100)
Department             VARCHAR(100)
Title                  VARCHAR(50)
```

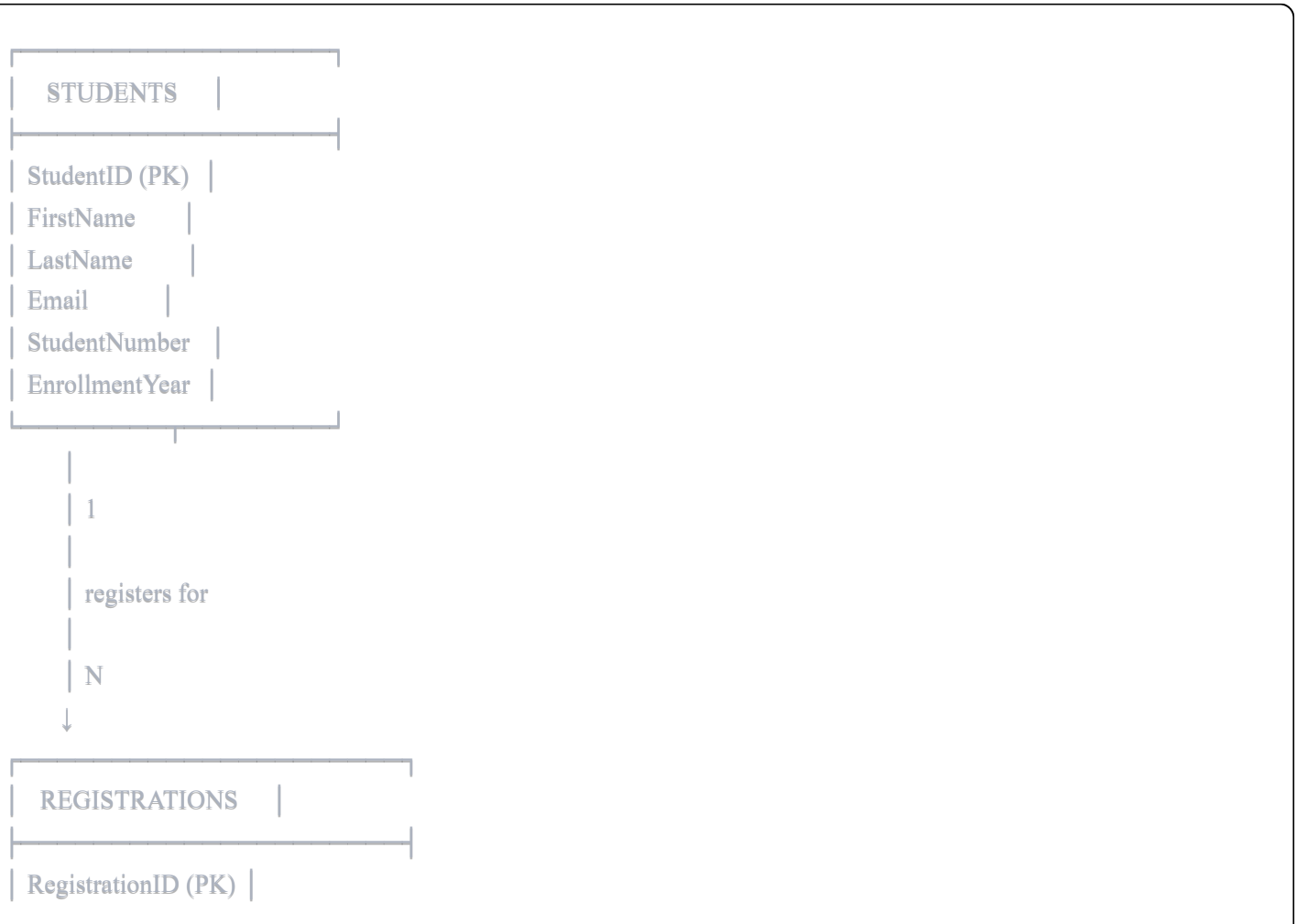
5. SECTIONS Table

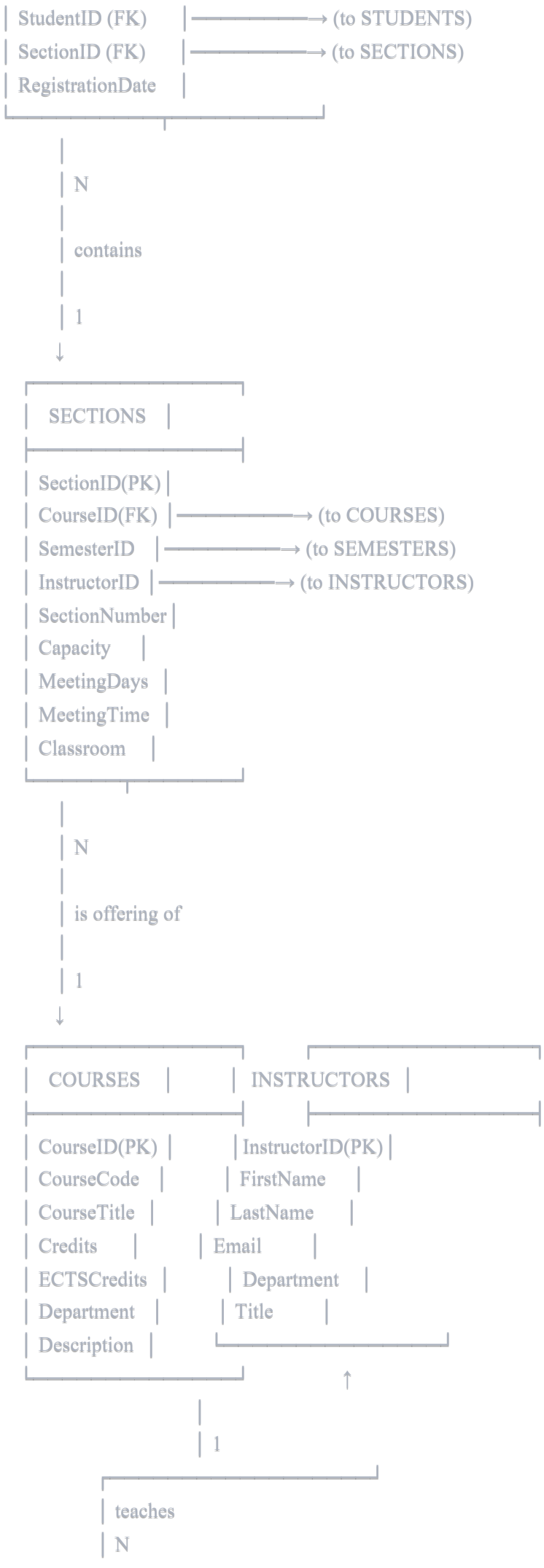
```
SECTIONS
--------
SectionID (PK)        INT
CourseID (FK)         INT  → References COURSES(CourseID)
SemesterID (FK)       INT  → References SEMESTERS(SemesterID)
InstructorID (FK)     INT  → References INSTRUCTORS(InstructorID)
SectionNumber         VARCHAR(10)
Capacity              INT
MeetingDays           VARCHAR(20)
MeetingTime           VARCHAR(50)
Classroom             VARCHAR(50)
```

6. REGISTRATIONS Table

```
REGISTRATIONS
-------------
RegistrationID (PK)    INT
StudentID (FK)         INT  → References STUDENTS(StudentID)
SectionID (FK)         INT  → References SECTIONS(SectionID)
RegistrationDate       DATETIME
```

ER Diagram




StudentID (FK)
SectionID (FK)
RegistrationDate
(to STUDENTS)
(to SECTIONS)
N
contains
1
SECTIONS
SectionID(PK)
CourseID(FK)
SemesterID
InstructorID
SectionNumber
Capacity
MeetingDays
MeetingTime
Classroom
(to COURSES)
(to SEMESTERS)
(to INSTRUCTORS)
N
is offering of
1
COURSES
CourseID(PK)
CourseCode
CourseTitle
Credits
ECTSCredits
Department
Description
INSTRUCTORS
InstructorID(PK)
FirstName
LastName
Email
Department
Title
1
teaches
N

Relationships:

1. **STUDENTS (1) ↔ (N) REGISTRATIONS** - One student can have many registrations
2. **SECTIONS (1) ↔ (N) REGISTRATIONS** - One section can have many student registrations
3. **COURSES (1) ↔ (N) SECTIONS** - One course can have many sections
4. **INSTRUCTORS (1) ↔ (N) SECTIONS** - One instructor can teach many sections
5. **SEMESTERS (1) ↔ (N) SECTIONS** - One semester contains many sections

N-M Relationship Resolution:

The **STUDENTS to SECTIONS** relationship is naturally many-to-many:

- One student can register for many sections
- One section can have many students

This N-M relationship is resolved using the **REGISTRATIONS** junction table, which contains:

- StudentID (FK to STUDENTS)
- SectionID (FK to SECTIONS)
- Additional registration metadata (RegistrationDate)

PART 3b: Explanation of Relations and Keys

Primary Key Selections

Why these primary keys were chosen:

Each table uses a surrogate integer primary key (StudentID, CourseID, SectionID, etc.) rather than natural keys. This design choice offers several advantages. Integer keys provide optimal performance for indexing and joining operations. They remain stable even if business data changes—for example, if a student's email address changes, the StudentID remains constant, preserving all relationships. Surrogate keys also simplify foreign key references and avoid composite key complexity.

For the REGISTRATIONS table, we use RegistrationID as the primary key rather than a composite key of (StudentID, SectionID). While the combination of StudentID and SectionID could serve as a natural primary key (since a student can only register once for a given section), using a surrogate key provides flexibility for future enhancements, such as tracking registration history or allowing re-registration after drops.

Foreign Key Selections

Why these foreign keys were chosen:

Foreign keys enforce referential integrity and establish clear relationships between entities. The SECTIONS table contains three foreign keys: CourseID links to COURSES (indicating which course this section offers), SemesterID links to SEMESTERS (indicating when the section is offered), and InstructorID links to INSTRUCTORS (indicating who teaches the section). These foreign keys ensure that a section cannot reference a non-existent course, semester, or instructor.

The REGISTRATIONS table contains two foreign keys: StudentID and SectionID. These establish the many-to-many relationship between students and sections. The foreign key constraints guarantee that registrations can only be created for valid students and valid sections, preventing orphaned records.

How Tables Are Connected

Explanation of table relationships:

The database design follows a clear hierarchical structure. At the foundation, we have independent entities: STUDENTS, COURSES, INSTRUCTORS, and SEMESTERS. These tables have no foreign keys and represent core business objects. The SECTIONS table sits at the next level, connecting COURSES, INSTRUCTORS, and SEMESTERS. This design reflects the real-world concept that a section is a specific offering of a course, taught by an instructor, during a particular semester. Finally, the REGISTRATIONS table connects STUDENTS to SECTIONS, completing the system by recording which students are enrolled in which sections.

Navigation through the database is straightforward. To find all courses a student is taking, we join STUDENTS → REGISTRATIONS → SECTIONS → COURSES. To find all students in a particular course, we reverse this path: COURSES → SECTIONS → REGISTRATIONS → STUDENTS. This star-like topology with SECTIONS and REGISTRATIONS as connection points provides flexible query patterns.

N-M Relationship Resolution

How many-to-many relationships are solved:

The primary N-M relationship in this system exists between STUDENTS and SECTIONS. Without proper resolution, this would require repeating groups or multiple values in a single column, violating first normal form. We resolve this using the REGISTRATIONS junction table, which creates two one-to-many relationships: one student has many registrations (1:N), and one section has many registrations (1:N). The junction table transforms the N-M complexity into two manageable 1-N relationships.

This pattern is standard for many-to-many relationships in relational databases. The REGISTRATIONS table serves exclusively as a junction table but can be enhanced with additional attributes describing the relationship itself, such as RegistrationDate. This additional data cannot belong to either STUDENTS or SECTIONS independently, as it describes the relationship, not the entities.

Why This Design Satisfies 2NF

Second Normal Form compliance:

This design satisfies Second Normal Form (2NF) requirements. First, all tables satisfy First Normal Form (1NF): there are no repeating groups, all attributes contain atomic values, and each table has a primary key. Second, there are no partial dependencies on composite keys. Since all tables use single-column surrogate primary keys rather than composite keys, partial dependencies cannot exist. Every non-key attribute in each table depends on the entire primary key.

In the SECTIONS table, for example, CourseID, InstructorID, SectionNumber, Capacity, and all other attributes depend entirely on SectionID. No attribute depends on only part of the primary key (because the primary key is not composite). The same logic applies to all tables. Even in REGISTRATIONS, where a composite key of (StudentID, SectionID) might seem natural, we use RegistrationID as the sole primary key, ensuring all other attributes depend on this single key. Therefore, the design is free from partial dependencies and fully complies with 2NF requirements.

PART 4a: Constraints That Force Students to Register for Courses

Discussion: Prerequisites, Co-requisites, and Required Chains

Why these constraints are important in a real system:

Prerequisites, co-requisites, and required course sequences are fundamental to academic integrity and pedagogical structure. Prerequisites ensure students have the necessary foundational knowledge before attempting advanced material—for example, students must understand Calculus I before tackling Calculus II. Without prerequisite enforcement, students may struggle or fail in courses they are unprepared for, wasting time and resources. Co-requisites ensure that theoretical and practical components are taken together, such as a chemistry lecture and its laboratory component. Required course chains enforce proper sequencing in programs where skills build progressively, such as Programming I → Programming II → Data Structures.

What new tables or fields would be needed:

To support prerequisite constraints, we would need a PREREQUISITES table with the following structure: PrerequisiteID (primary key), CourseID (foreign key to the course that has the requirement), RequiredCourseID (foreign key to the course that must be completed first), and MinimumGrade (the minimum acceptable grade, such as 'C' or 'D'). This table creates a self-referential relationship within the COURSES table, linking courses to their prerequisite courses. Additionally, we need a way to track student grades for completed courses. This could be a COMPLETED_COURSES table or a GRADES table with fields: StudentID, CourseID, Grade, SemesterID, and CompletionStatus. Without this grade history, the system cannot verify whether a student has satisfied prerequisites.

What information must be stored to check these constraints:

The system must store several pieces of information to validate prerequisites. First, it needs a complete mapping of all prerequisite relationships (which course requires which other courses). Second, it must store minimum grade thresholds for each prerequisite. Third, it needs a complete history of courses each student has taken, along with their final grades. Fourth, it should track courses currently in progress, as some systems allow conditional registration if the prerequisite is being taken concurrently. Finally, the system needs to distinguish between different types of constraints: hard prerequisites (absolutely required), recommended prerequisites (advisory), and co-requisites (must be taken simultaneously). For co-requisites specifically, the system must validate that both courses are in the student's current registration for the same semester.

Why these tables and fields are not required in the simple system:

The simplified system from Part 2 deliberately excludes all rule enforcement to focus on basic functionality. Without prerequisites, the database only needs to track current registrations, not historical course completion or grades. The absence of grade tracking significantly simplifies the schema—we don't need to store past performance, only current enrollments. This makes the system appropriate for understanding fundamental database concepts like entities, relationships, and foreign keys without the complexity of constraint validation. In a real university environment, prerequisite checking is critical, but for learning database design fundamentals, the simplified model provides a clearer introduction to core concepts before adding layers of business logic complexity.

PART 4b: Constraints That Limit What Students Can Register For

Discussion: Time Conflicts, Credit Limits, and Exclusionary Rules

Why these limiting rules matter in a real system:

Limiting rules protect students from impossible or inadvisable registration scenarios. Time conflict prevention ensures students cannot register for two sections that meet simultaneously—a physical impossibility that would force the student to miss one class. Credit load limits prevent students from overextending themselves academically, which could lead to poor performance across all courses. Research shows that students taking excessive credit loads have lower GPAs and higher dropout rates. Mutually exclusive course rules prevent students from receiving duplicate credit for substantially overlapping content, ensuring degree requirements are met with diverse coursework. Withdrawal restrictions prevent students from abandoning courses they've previously failed, forcing them to confront weak areas rather than repeatedly avoiding challenging material.

What extra tables or fields are needed to enforce them:

Time conflict checking requires detailed schedule information. The SECTIONS table would need structured fields for MeetingDays (perhaps stored as a bit pattern: Monday=1, Tuesday=2, Wednesday=4, etc.) and precise MeetingTime values (StartTime and EndTime). A more sophisticated design might use a MEETING_TIMES table with multiple entries per section to handle courses with non-standard schedules. For credit limits, we need to store both the institutional rules (maximum/minimum ECTS per semester) and student-specific overrides. This could be a CREDIT_LIMITS table with SemesterID, StudentCategoryID, MaxCredits, and MinCredits. For mutually exclusive courses, we need a COURSE_EXCLUSIONS table linking CourseID1 and CourseID2 as mutually exclusive pairs. Finally, to enforce withdrawal restrictions, we need historical data showing which courses a student has previously failed, which brings us back to requiring a grade history table.

What information the system needs to check before allowing registration:

Before allowing a registration, the system must gather and validate multiple pieces of information. For time conflict checking, it must retrieve all sections the student is currently registered for, extract their meeting times, and compare them against the proposed new section's schedule, accounting for travel time between classes if campuses differ. For credit limit validation, the system must calculate the student's current ECTS total for the semester, add the credits from the proposed section, and compare against maximum limits (potentially checking the student's CGPA to see if they qualify for overload permission). For mutual exclusion, the system must check if the student has already completed or is currently registered for any course that is mutually exclusive with the proposed course. For withdrawal-based restrictions, the system must query the student's academic history to determine if they've previously failed or withdrawn from this course, and if so, whether department policy permits another attempt.

Why these rules were removed from the basic system:

These rules were removed from the Part 2 design because they require complex validation logic and extensive additional data. The simplified system focuses on the fundamental database operations—creating entities and linking them through relationships—without the burden of implementing business rules. Time conflict checking requires sophisticated date/time parsing and comparison logic. Credit limit enforcement requires aggregate queries (summing ECTS across multiple registrations) and conditional logic based on student status. Mutual exclusion requires recursive or transitive checking if course equivalencies form chains. These advanced features are important in production systems but would obscure the core learning objectives of understanding primary keys, foreign keys, and basic relational design. By removing constraints, students can focus on proper normalization and relationship modeling before tackling the additional complexity of rule enforcement in subsequent assignments.

PART 4c: The Requirement to Keep History

Discussion: Action Logging and Registration History

Why a real registration system needs to keep a history of actions:

Action history is critical for multiple stakeholders and purposes in a university environment. For students, history provides transparency—they can review what changes they made and when, helping them understand

deadlines and track their decision-making process. For administrators, history is essential for auditing and dispute resolution. If a student claims they tried to drop a course but the system failed, the action log provides definitive evidence of what occurred. For academic advisors, history reveals patterns in student behavior, such as repeatedly dropping courses in the same subject, which might indicate placement issues or need for additional support. From a compliance perspective, universities may be legally required to maintain records of registration changes for accreditation reviews, financial aid verification, or legal proceedings. History also supports data analytics—universities can identify systematic problems like courses that experience high drop rates or registration bottlenecks during peak periods.

What extra tables or fields would be needed to store history:

A comprehensive history system requires an ACTION_LOG table with a detailed structure. Essential fields include: ActionLogID (primary key), StudentID (foreign key), ActionType (enumerated type: 'ADD', 'DROP', 'WITHDRAW', 'SECTION_CHANGE', 'OVERRIDE', etc.), CourseID and SectionID (identifying what course/section was affected), Timestamp (precise date and time of action), PerformedBy (user ID of who initiated the action—student, advisor, or administrator), PreviousValue and NewValue (for tracking changes, especially in section changes), ApproverID (if the action required approval), ApprovalReason (text field for override justifications), and SystemNotes (automated system messages). For certain action types, additional context may be needed: TimeConflictApproval actions should store details about which sections conflict and which instructors approved; Override actions should store the reason and authority level. This might lead to specialized tables like OVERRIDE_LOG or CONFLICT_APPROVALS as extensions of the main ACTION_LOG.

Why this requirement makes the system more complex:

History tracking fundamentally changes the database from recording current state to recording state transitions over time. Every registration operation becomes a multi-step process: first, update the current state (add or delete from REGISTRATIONS), then create a permanent record in ACTION_LOG. This dual-write pattern introduces potential consistency issues—what if the registration succeeds but the log insert fails? Transaction management becomes critical. Storage requirements grow continuously, as history tables accumulate data indefinitely while current state tables remain relatively static. Query complexity increases because answering questions like "Was this student registered for ECON211 on September 15?" requires either querying historical logs or implementing a temporal database design. User interface complexity also increases, as the system must display both current registrations and historical actions, with appropriate filtering and visualization. Performance considerations arise because the ACTION_LOG table grows without bound and could become a bottleneck if not properly indexed and potentially archived.

Why we did not include this history in the simplified design:

The Part 3 simplified design excludes history tracking to maintain focus on foundational concepts. History tracking is an advanced feature that requires understanding of temporal data modeling, transaction integrity, and audit trail design—topics beyond the scope of an introductory database design exercise. Including history would require explaining concepts like soft deletes (marking records as inactive rather than deleting them), temporal validity (tracking when data was valid versus when it was recorded), and event sourcing patterns. These concepts can confuse students who are still learning basic normalization and relationship modeling. The simplified design treats the database as a "current state" snapshot, which is conceptually easier to grasp. Students can understand that REGISTRATIONS contains "who is currently enrolled in what" without the added

complexity of "who was enrolled when, and who made changes." Once students master basic relational design principles in the simplified system, they are better prepared to extend their understanding to temporal and historical data tracking in more advanced coursework or real-world implementations.

END OF HOMEWORK 1