Lab W03 - Requirements Analysis

Note: You may partner with one other class member (strongly encouraged) or work individually on these labs. The assignment requirements are the same in either case. You cannot change partners later, but you or your partner can be laid off.

Objective: Analyze the problem description and translate it into precise requirements.

Real-world database projects seldom start with a fully complete or perfectly detailed set of requirements. Instead, designers need to analyze the problem, determine the necessary system features, identify ambiguities, and make sensible assumptions. In this lab, you will review the CERMS project scenario to create a requirements document that specifies the system's functions and constraints.

This document will guide all future work in this lab. Vague descriptions and poorly justified designs and decisions at this stage will have consequences later in the semester.

Note: Do not use generative AI (considered a violation of academic integrity).

Requirements Document

You will create a written requirements document that clearly explains your understanding of the problem domain. This document should be written in complete sentences and organized using clear section headings. Use one of the templates provided.

1. System Overview

Provide a summary of the CERMS system in your own words. Describe the purpose of the system and the problems it is intended to solve.

2. Functional Requirements

List and describe the actions the system must support. These should focus on what the system does, not how it is implemented. Examples include, but are not limited to:

- Managing users and roles
- Creating and modifying events
- Reserving resources and detecting conflicts
- Handling approvals
- Supporting registration and attendance
- Functional requirements should be specific, testable, and written from a system perspective.

3. Non-Functional Requirements

Describe constraints and quality attributes that affect how the system operates. Examples include:

- Data integrity and consistency
- Concurrency and multi-user access
- Security and access control
- Performance considerations
- Auditability and logging

Avoid vague statements. Each requirement should explain why it matters.

4. Data Requirements

Identify the major types of data the system must store and manage. This section should describe entities at a high level without listing table structures. Examples include:

- Users and their roles
- Events and scheduling information
- Resources and availability
- Approvals and status tracking

This section should clearly connect system behavior to stored data.

5. Business Rules

This section describes the policies, constraints, and logical rules that govern the system's operation. Business rules are statements about the domain that must always be valid, regardless of the database or application implementation. They are not technical details but specify how the real-world system behaves and what the database must enforce or support. Include all business rules in this section, as there will be many. Avoid vague terms such as "usually," “often,” or “as needed,” and follow the examples from our lectures and textbook.

Business Rule Format: Each business rule should be stated as a clear, declarative statement. Number each rule so it can be referenced later in your ER diagram and design justification. Unlike the other report sections above, these generally require little explanation because they are simple, clear statements. Examples of acceptable formats:s:

1. “An event may reserve one or more resources.”
2. “An event has an ID, title, date, time, description, and room number.”
3. “A resource may not be reserved by more than one event during overlapping time periods.”
4. “Only users with an approver role may approve events requiring approval.”

Coverage: Your business rules must address at least the following areas, though additional rules are encouraged.

- *Users and Roles*
    - How users are associated with roles
    - Whether users may hold multiple roles
    - Restrictions on role-based actions
- *Events*
    - Event creation, modification, and status
    - Whether events can be associated with multiple organizations
    - Rules governing event approval
- *Resources and Reservations*
    - Resource availability and conflicts
    - Capacity or usage limits
    - Authorization requirements for restricted resources

- *Registration and Attendance*
    - Whether registration is required for certain events
    - Capacity enforcement
    - Rules governing waitlists or overbooking, if applicable
- *Approvals and Auditing*
    - Conditions that require approval
    - Ordering or dependency of approvals
    - Whether approval actions must be recorded and retained
- *ER Diagram Rules*
    - Entities and their attributes
    - Relationships, including cardinality, participation, and strength (remember that relationships are written in both directions)
    - Associative entities for resolving many-to-many relationships
    - Attribute constraints and identifiers

Example Business Rules (Illustrative Only)

- The following examples are provided for clarity. You should not copy them verbatim.
- A user may be associated with zero or more organizations.
- An event must be associated with at least one organization.
- An event may require one or more approvals before becoming active.
- A resource may be reserved by multiple events, but not during overlapping time periods.
- The assigned resource's capacity may limit registration for an event.
- Approval actions must be recorded and linked to the approving user.

6. Assumptions and Open Questions

Document any assumptions you made while interpreting the scenario. Identify ambiguities or missing information and explain how you chose to resolve them. This section is required and will be graded. Thoughtful assumptions are preferable to ignoring uncertainty.

Submission

- Use one of the provided templates to write your document.
- Name the document 'lastRequirements' where last is your last name. The filename extension will depend on the template you used.
    - If you are working in a group, name the file "lastlastRequirements" where lastlast is both last names in your group.
- Create a folder in your cis4000work repository named 'labs'.
    - The work only needs to be in one team member's repository.
- Upload and commit the document to the labs folder.
    - You may organize your labs folder how you wish as long as I can easily find your submission
- Place "done" in the Moodle submission textbox for Lab W03.
    - Both team members must do this.

Evaluation Criteria

The lab emphasizes requirements analysis rather than database implementation. Since there is not a single correct set of requirements, your work will be assessed mainly on the quality of your analysis and reasoning, not on

including specific features. Evaluation will be based on a mix of objective standards and professional judgment, reflecting how design documents are reviewed in actual technical settings.

Grading Breakdown

- Objective Criteria: 30%
- Subjective Evaluation: 70%

Both components matter. A well-written document that ignores the problem domain will not score well, nor will one that lists many items without clear reasoning or organization.

Objective Criteria (30 points)

The objective portion of the grade is based on whether your submission includes all required sections and meets baseline expectations. This includes:

- All the necessary sections are present
- The document is clearly organized and readable
- Requirements are written in complete, coherent sentences
- Business rules are clearly stated and numbered
- Assumptions and ambiguities are explicitly documented

Subjective Evaluation (70 points)

Most points are based on the quality of thought and effort demonstrated in your work. This portion of the grade answers the question, “Does this look like careful, senior-level analysis informed by the course material?”

The following qualities will be considered:

- Reasonableness
    - Do the requirements make sense given the project scenario?
    - Are assumptions realistic and defensible?
    - Are edge cases and constraints considered, rather than ignored?
- Thoroughness
    - Does the document address all major aspects of the system?
    - Are important behaviors, constraints, and interactions discussed?
    - Does it go beyond obvious surface-level requirements?
- Completeness
    - Do the requirements work together as a coherent whole?
    - Are dependencies and implications acknowledged?
    - Are there noticeable gaps that would cause problems later?
- Clarity and Writing Quality
    - Is the document easy to read and professionally written?
    - Are ideas expressed clearly and unambiguously?
    - Would another developer be able to understand the system based on this document?
- Evidence of Effort and Engagement
    - Does the work reflect careful reading of the scenario?
    - Are course concepts applied thoughtfully rather than copied mechanically?
    - Does the document show signs of revision, organization, and reflection?

What This Lab Is Not

- This is not a test of how many requirements you can list.
- This is not about guessing what the instructor “wants.”
- This is not graded on whether your assumptions match someone else’s.

Thoughtful and reasonable alternatives and clearly justified choices are all acceptable and encouraged.

How to Do Well

Strong submissions typically:

- Explain why requirements exist, not just what they are
- Acknowledge uncertainty and make justified assumptions
- Use precise language and consistent terminology
- Demonstrate that the author is thinking ahead to later design and implementation phases

Weak submissions often:

- Paraphrase the scenario without analysis
- List generic or obvious statements without justification
- Avoid making decisions to sidestep ambiguity
- Read like a rushed or first-draft effort

Example: Strong vs. Weak Requirements Thinking

Topic: Event Registration Capacity

Weak Example

The system should limit event registration based on capacity.

This statement repeats the scenario at a surface level. It does not explain what “capacity” means, how it is determined, or what happens when limits are reached.

Strong Example

The system needs to enforce a maximum registration limit for required sign-up events, based on the resource with the smallest reserved capacity. When multiple users try to register at the same time and the capacity is nearly full, the system should block further registrations and keep accurate count. Once full, it should reject any additional registration attempts and log them for auditing.

This statement:

- Clarifies how capacity is defined
- Anticipates concurrency issues
- Connects requirements to data integrity and auditing
- Demonstrates forward-thinking beyond the immediate scenario

Final Note

This lab sets the tone for the rest of the semester. It is an opportunity to demonstrate how you approach an open-ended technical problem when no one gives you a complete specification. That skill is as important as any specific database technique you will learn in this course.