CSc 460 — Database Design
Spring 2026 (McCann)

http://www2.cs.arizona.edu/classes/cs460/spring26/

Program #4: Database Design and Implementation

Due Dates:

Team Members:	April 21st, 2026, at the beginning of class
Draft E–R Diagram:	April 28th, 2026, at the beginning of class
Final Product:	May 5th, 2026, at the beginning of class

Designed by *Jianwei (James) Shen and Muhammad Bilal*

Overview: In this assignment, you will build a database-driven information management system from the ground up. We will give you an application domain to work on. Your goal is to design the underlying database, define the application functionalities you will provide, and implement this application using Oracle within a text–based JDBC program.

Assignment: In this assignment you are to implement a two–tier client–server architecture.

1. **Database Back–End**, which runs the Oracle DBMS on `aloe.cs.arizona.edu`. You must design the relational schema, justify 3NF/BCNF, and populate it with initial data.
2. **JDBC Front–End**, which is the client's user interface running on `lectura`.

Application Domain: The LLM User–Facing Ecosystem

What is a "User–Facing Ecosystem?" To understand this domain, distinguish between the *AI Model* (the complex neural network that generates text) and the *Ecosystem* (the software infrastructure users actually interact with).

Think of a professional user, "Alice," a software consultant. Alice doesn't just "chat" with an AI; she manages a digital workspace. She has a **Pro Subscription** (Billing), uses a **"Senior Architect" Persona** to style her responses (Personalization), organizes her research into a **"Cloud Migration" Workspace** (Organization), and occasionally contacts **Tech Support** when her usage limits are exceeded (Support). Your database is the "brain" that tracks Alice's settings, her money, her collaborative folders, and her history — not the AI that generates the words.

Application Description:

The project focuses on the user–facing ecosystem of a modern Large Language Model (LLM) platform. While the AI's "intelligence" resides in massive neural networks, the **relational database** serves as the system's memory, organizational structure, and personalization engine.

Users must register for the service to maintain a persistent history of their conversations. We collect basic information: name, email, account creation date, and a preferred UI language. Each user have a membership tier (e.g., Free, Plus, Enterprise). Each tier has different rate limits, such as a maximum number of messages per day or access to advanced "pro" models.

(Continued...)

The core interaction unit is the conversation. A conversation is a logical container for a specific dialogue thread, identified by a user–defined title. Within each conversation are multiple Messages. To maintain the "chat" experience, the database must track the role of each message sender (User vs. AI Assistant), the timestamp, and the content. To facilitate model improvement, the system tracks user feedback on every AI–generated message. This includes a rating and optional text. Users also have the ability to bookmark specific messages for quick retrieval.

To provide a tailored experience, the system utilizes Personas (System Instructions). A Persona is a pre–defined set of behavioral guidelines (e.g., "Act as a technical writer"). When a user starts a conversation, they can attach a Persona to it. The database must ensure that if a Persona is updated or deleted, the historical context of existing conversations remains coherent.

For professional workflows, the application supports Workspaces. A Workspace acts as a shared or private environment where multiple conversations are grouped. To boost efficiency, the platform includes a Prompt Library. This is a repository of Prompt Templates — reusable, high–quality boilerplate "prompts" (e.g., "Summarize this legal document"). Users can save, edit, and categorize these templates, tracking which are private and which are shared within a Workspace.

To scale the business, the system now requires robust billing profiles and invoices. Each user must have a billing record (payment method, billing address) and a history of generated invoices showing the amount, date, and payment status. Furthermore, to maintain user satisfaction, the system tracks support tickets. When a user encounters an issue, they open a ticket. These tickets are assigned to support agents and record the topic (e.g., Billing, Model Error, etc.), the duration of the resolution process, and the final outcome (Resolved/Escalated).

Finally, stakeholders require data–driven insights. The system must support queries that aggregate usage, such as identifying popular Personas, tracking message volume per membership tier, and monitoring AI success rates based on user feedback.

Required functionalities: Specifically, the JDBC application's interface should enable users to:

1. **Manage User Accounts:** Add, update (e.g., change tier), or delete users. Deleting a user must fail if there are unpaid invoices or open support tickets.
2. **Handle Conversations & Messages:** Start a new conversation, add messages to it, and update message feedback. When a user is deleted, their message history must be handled appropriately (deleted or anonymized).
3. **Workspace Organization:** Create or modify workspaces. The system must verify that a user belongs to a workspace before they can move a conversation into it.
4. **Persona Management:** Create or delete custom instructions/personas. A persona cannot be deleted if it is currently the "active" template for more than five ongoing conversations.
5. **Prompt Library:** Add or update saved prompt templates.
6. **Subscription Tracking:** Update a user's subscription level and verify they haven't exceeded their tier's message limits before allowing a new "Record Insertion" for a message.
7. **Billing Operations:** Generate a new invoice for a user's monthly tier fee or mark an existing invoice as "Paid."
8. **Support Ticket Lifecycle:** Create a support ticket for a user, assign it to an agent, and update its resolution status and duration upon completion.

(Continued...)

Required queries: Here are the queries that your application is to be able to answer:

1. For a given User, list all their Bookmarked messages across all conversations, including the conversation title and the timestamp.
2. List all users who have "Unpaid" invoices, including their email, the total amount owed, and the date of their last conversation.
3. Identify the "Most Helpful" Persona: List the persona name that has received the highest percentage of "Thumbs Up" feedback across all conversations linked to it.
4. One additional non–trivial query of your own design, with these restrictions:
   - The question must use more than two relations.
   - It must be constructed using at least one piece of information gathered from the user.

Working in Groups: In industry, such a project is usually the work of multiple developers, because it involves several different components. Good communication is a vital key to the success of the project. This assignment provides an opportunity for just this sort of teamwork. Therefore, we are accepting team sizes of between two and four members (inclusive), due to the desire to add to your teamwork experiences, the scope of the project, and the stresses students usually experience at this time of the semester (camaraderie helps!).

Early on, you will need to agree on a reasonable workload distribution plan for your team, with well–defined responsibilities, deliverables, and expected completion dates. Such a plan will minimize conflicts and debugging effort in the actual implementation.

Late days: Late days can be used on this assignment, but only on the third due date. How many a team has to use is determined as follows: Team members total their remaining late days, and divide by the number of members in the team (integer division), producing the number of late days the team has available, **to a max of two days late**. (Why? The TAs need to get grading done soon after the due date, you need time to study for your final exams, and the department has a rule about assignments needing to be due before the start of finals.)

For example, a team whose three members have 1, 1, and 3 late days remaining have $\lfloor \frac{1+1+3}{3} \rfloor = 1$ late day to use, if needed.

Hand In: Here are the 'deliverables' for each of the assignment's three due dates:

1. *Team Composition*: By the first due date (see the top of the front page of this handout), one member of your team must add a new row to the shared Google Sheet with the names and email addresses of all team members. The link to the Google Sheet has, or will soon be, posted on Piazza. Failure to do so by the start of class on this date will cost your team the corresponding points listed in the Grading Criteria section (below).
2. *E–R Diagram*: As stated in the Assignment section, your team will need to create an E–R diagram that describes your database design. Before the second due date, your team will need to prepare a draft of your E–R diagram **and** a member of your team will need to submit it through `turnin` to the `cs460p4` folder. The purpose of this requirement is to allow the TAs to review your schema and make suggestions for improvement. The sooner you create your design and discuss it with the TAs, the more time you will have to refine your final E–R diagram. If TAs need further explanation of your E–R Diagram, they'll send out an email to make an appointment to have an additional meeting.

(Continued...)

3. *Final Product*: On or before the third due date, a member of your team must submit a `.tar` file of your well–documented application program file(s) via turnin to the folder `cs460p4`. The tar file should contain all of the following:
   (a) The source code for your application.
   (b) A PDF file named "design.pdf" containing the following sections, in this order:
      i. *Conceptual database design*: Your final E–R diagram along with your design rationale and any necessary high–level text description of the data model (e.g., constraints, or anything you were not able to show in the E–R diagram but that is necessary to help people understand your database design).
      ii. *Logical database design*: The conversion of your E–R schema into a relational database schema. Provide the schemas of the tables resulting from this step.
      iii. *Normalization analysis*: For each of your entity sets (tables), provide all of the FDs of the table and justify why your the table adheres to 3NF / BCNF.
      iv. *Query description*: Describe your self–designed query. Specifically, what question is it answering, and what is the utility of including such a query in the system?
   (c) A `ReadMe.txt` describing:
      i. Compilation and execution instructions, to enable the TAs to execute your application and exercise the required functionalities.
      ii. The workload distribution among team members (that is, which people were responsible for which parts of the project).

   In addition, each team must schedule a time slot (∼15 – 20 minutes) to meet with a TA, demonstrate your system, and perhaps answer some questions about it. Closer to the final due date, we will let you know how to sign up.

Grading Criteria: Total: 100 points

1. Team Composition (1st due date): 5
2. Complete E–R Diagram Draft (2nd due date): 20
3. Final Submission (3rd due date): 75
   (a) Coding / Implementation: 55
      - Documentation 15
      - Style and organization 10
      - Record insertion: 5
      - Record deletion: 5
      - Record update: 10
      - Record query: 10
   (b) Database design: 20
      - Final E–R diagram: 10
      - Normalization analysis: 10

Grading Notes:

1. Absent verifiable complaints about inadequate contributions, each member of a team will receive the same score on this assignment.
2. We won't put much weight at all on the appearance of the text application; concern yourselves with the application's design and functionality instead. The main point of the assignment is the DB design and how well it supports the use cases (the DML operations and queries, in particular).