

# Course Project: Question Answering System

CS480: Database Systems

Fall 2025

## 1 Overview

In this project, you will design and implement a **document question-answering system** that combines two major parts:

1. **Relational Database Part:** You will design entities, relationships, and schemas for managing different entities of the system, like users, roles, documents, and logs. You will create an ER diagram, translate it into a relational schema (SQL), and implement CRUD operations on that.
2. **Vector Database Part:** You will implement a pipeline similar to **Retrieval-Augmented Generation (RAG)**.<sup>1</sup> Specifically, you will process unstructured documents by chunking them, generating embeddings, building vector indices, and running top- $k$  retrieval queries. A language model will then generate answers with citations drawn from the retrieved passages.

### 1.1 Project Goals

- Learn ER diagram design and schema creation.
- Implement CRUD over a relational DB.
- Gain hands-on experience with document embedding vectors.
- Using popular vector indices (like IVF or HNSW).
- Generate answers with citations from retrieved passages.

### Questions?

If you have any questions, feel free to contact the TA (Mohsen: [mdehgh2@uic.edu](mailto:mdehgh2@uic.edu)) or visit during office hours on Wednesdays from 3:30–5:30 pm in CDRLC 2402.

## 2 Timeline & Phases

The project will be completed in several phases. Each phase has a deliverable that must be submitted on the due date. Late submissions will follow the course late policy.

---

<sup>1</sup>What is RAG? See this blog post.

## Phase 0: Teams (Deadline: Monday, September 15)

- Select a dataset (containing approximately 10-30 documents.<sup>2</sup>)
- Submit the following:
  1. Description of the dataset and its source.
  2. List of team members (at most 3).
  3. Chosen team name.
  4. GitHub usernames of all members.

## Phase 1: ER Model (Deadline: October 5th)

- Create an ER diagram for the system.<sup>3</sup>
- The diagram should clearly include all required entities and their relationships.
- Submit screenshots or image files of the ER diagram.

## Phase 2: Relational Schema (Deadline: October 19th)

- Translate the ER model into a relational schema, implemented as an SQL script using PostgreSQL's dialect.
- Submit a `.sql` file that generates the relational schema.

## Phase 3: Vector Pipeline (Deadline: November 10th)

- Ingest and chunk the dataset documents (Task 1).
- Generate embeddings for all document chunks (Task 2).
- Build a flat (brute-force) index to enable basic retrieval.
- Implement initial CRUD operations on the relational database.
- Commit all implementation code to the team's GitHub repository.

## Phase 4: Full System Integration (Deadline: December 1st)

- Implement an advanced index (e.g., IVF or HNSW) on top of the embeddings (Task 3).
- Develop a user interface that supports login and query execution (Task 4).
- Integrate the relational database with the vector pipeline.
- Integrate the queries with an LLM for a full RAG pipeline<sup>4</sup>.
- Demonstrate the complete system, showcasing all required functionalities.

---

<sup>2</sup>If you choose a larger dataset, you may work with a smaller sample.

<sup>3</sup>See the following section: *Entities and Requirements*.

<sup>4</sup>Bonus point

## 3 Relational Database

### 3.1 Entities

Design an **ER diagram** and the corresponding relational schema, including the following entities:

- **Users** – Represents all system users, storing attributes such as **id**, **name**, and **email**. Each user is uniquely identified by the *id*. Users fall into three roles:
  - **EndUser** – submits queries to the system. For each EndUser, also store the timestamp of their most recent activity.
  - **Admin** – responsible for managing and overseeing user accounts.
  - **Curator** – responsible for adding, updating, and deleting documents.
- **Document** – stores metadata for each document, including **id**, **title**, **type**, **source**, **added\_by**, and **timestamp**. Also, for each document, record whether it has been processed by the Vector Pipeline.
- **QueryLog** – records details of user queries, such as the query text, the issuing user, timestamp, and the IDs of the retrieved documents.

### 3.2 Requirements

The system should support the following functionalities for different user roles.

#### 3.2.1 General

All users must be able to log into the system using a username and password (credentials are created when the user account is generated).

#### 3.2.2 Admin

Admins can perform full CRUD operations on users: create new users, view a list of all users, update user information, and delete user accounts.

#### 3.2.3 Curator

Curators can perform CRUD operations on documents. However, a curator may only update or delete documents they originally created (not those of other curators).

#### 3.2.4 EndUser

EndUsers can submit queries (i.e., ask questions to the system). The system should use the Vector Pipeline to retrieve relevant documents and return the top-*k* most relevant results. Each query must also be recorded as a **QueryLog** entity.

## 4 Vector Database

### 4.1 Task 1: Ingestion & Chunking

- Ingest documents from the chosen dataset.
- Chunk into passages (256–512 tokens with overlap).
- Store chunks with metadata in the relational DB.

### 4.2 Task 2: Embeddings

- Choose an embedding model (open-source or API).
- Generate embeddings for all chunks.

### 4.3 Task 3: Indexing

- Build two indices:
  1. **Flat index** (brute force).
  2. **One advanced index** (IVF or HNSW).
- Justify index parameter choices.

### 4.4 Task 4: Retrieval & Generator

- Implement top- $k$  retrieval for a given query (question).
- Provide a query interface returning ranked chunks with scores.
- Generator:
  - Baseline: concatenate retrieved chunks into an answer **with citations**.
  - Optional: use a free/open LLM to paraphrase (citations required).

## 5 Constraints & Tips

- Keep dataset modest to run locally.
- Use only public or course-provided content.
- Ensure reproducibility of your code (in GitHub Repository).
- If using an API, ensure there's a free tier or fallback template-only generator.

## 6 Suggested Tools

- **Relational Database:** PostgreSQL
- **ER Diagram Design:** draw.io, Lucidchart, dbdiagram.io
- **Document Parsing:** pdfminer.six, PyPDF2, textract, readability-lxml
- **Embeddings:** SentenceTransformers (e.g., `all-MiniLM-L6-v2`, `bge-small-en`), Hugging Face API models
- **Vector Index / Vector Database:** FAISS, pgvector (Postgres extension), Qdrant, Weaviate, Milvus
- **Backend / Application:** Python (CLI with argparse or click; minimal API with Flask or FastAPI)
- **LLM for Generation:**
  - OpenAI API (`gpt-4o-mini`, `gpt-4o`, or similar free-tier model)
  - Hugging Face Inference API (e.g., `mistralai/Mistral-7B-Instruct-v0.2`, `tiuuue/falcon-7b-instruct`)
  - Local open-source models via `llama.cpp` or `Ollama` (if no API is used)