Fall 2025
Y.Wu

CSDS 341: Introduction to Databases
Assignment #6

Due: Dec 2, 2025 (11:59 PM ET)

All assignments in this course are to be digitally produced and submitted to the canvas as pdf.

*Questions marked with * are optional and do not require answers. These questions are provided for you to practice with specific database systems and queries.*

1. [Graph DBs and NoSQL] (30).

Recall the five types of (NoSQL) database we have discussed in the class so far: Database types:
A. Key-Value Store (e.g., Redis)
B. Column Store (e.g., Cassandra, HBase)
C. Document Store (e.g., MongoDB)
D. Graph Database (e.g., Neo4j)
E. Vector Database (e.g., Pinecone, Milvus, FAISS)

(a). Given scenarios and query tasks below, for each case, select which type of NoSQL database best fits the application's needs (fill in A, B, C, D, E; one choice for each case suffices).

Scenario / Application	Query/ Task	Best-Fit DB Type?
Real-time personalization and caching for an online retail website, storing session tokens and user carts for millions of active users.	Fast lookup by session ID or user ID; extremely low latency reads/writes for ephemeral data.	
Global-scale time-series analytics platform monitoring IoT devices, energy grids, or financial tick data.	Range scans and aggregations over time intervals for billions of sensor readings.	
Content management and flexible JSON-like data for an e-commerce platform storing product catalogs, reviews, and metadata with different fields per item.	Search by product category, flexible schema, and nested document queries.	
Social network and recommendation graph connecting users, influencers, and shared interests.	Multi-hop traversal queries like "friends-of-friends," "who influences whom," or "shortest path between two users."	
AI-powered semantic search system for images, research papers, or user queries.	Retrieve top-k items with the most similar vector embeddings (cosine similarity or Euclidean distance).	

(b). Taylor is a popular tech reviewer on YouTube. Last week, Taylor posted a video titled "*Why the VisionPro 2 Changes Everything*", which was uploaded on Feb 12, 2025, and received 2.3 million views in 48 hours. In the video, Taylor reviews the new VisionPro 2 device, manufactured by Apple, which includes features like Eye-Track Pro and SpatialCast.

After the video went viral, Lin Zhang, a software engineer at Meta, commented:
"*Super interesting breakdown! I'm curious how this compares to Meta Quest 4."*

Several viewers liked Lin's comment. Taylor later collaborated with Sam Rivera, another tech creator, to make a follow-up video comparing VisionPro 2 and Meta Quest 4.

Using the **property graph model**, create a graph that includes:

- Nodes with appropriate labels (e.g., Person, Video, Device, Company, Feature, Person, Comment).
- Meaningful properties for nodes (e.g., name, role, job_title; video_title, date, views; device_name; company_name; feature_name; text).
- Relationships with meaningful types and directions, including properties when appropriate (e.g., POSTED, REVIEWED, MANUFACTURED_BY, COMMENTED, LIKED, COLLABORATED_WITH, WROTE, HAS_FEATURE, COMPARES).
- Enough structure such that the result is useful for actual recommendation, influence, or social-graph tasks.

Draw or describe the resulting graph clearly. For example,
Nodes (examples): (:Person {name: "Taylor Chen", role: "Tech Reviewer"})
Relationships (examples): (Taylor)-[:POSTED]->(Video) …

(c). Consider example Cypher queries. Describe in natural language what this query does.
For example: the query
MATCH (t:Person {name: "Taylor Chen"})-[:POSTED]->(v:Video) RETURN v.title, v.date
is to "Find all videos that Taylor Chen has posted, and return each video's title and upload date."

- **MATCH (d:Device {name: "VisionPro 2"})-[:HAS_FEATURE]->(f:Feature)**
  RETURN f.name;

- **MATCH (p:Person)-[:COMMENTED_ON]->(v:Video {title: "Why the VisionPro 2 Changes Everything"})**
  RETURN p.name;

- **MATCH (t:Person {name: "Taylor Chen"})-[:COLLABORATED_WITH]->(s:Person)**
  RETURN s.name;

*(d) *. Install Neo4j. Try creating the graph above, and play with search with Cypher queries to retrieve different nodes and edges with patterns, you don't need to submit anything for this (d).*

2. [Vectors Similarity] (30). Modern recommendation systems (like Netflix, YouTube, Spotify) represent items as embedding numeric vectors that capture properties such as genre, style, mood, or audience patterns. Suppose you are building a mini movie recommender system. Each movie is encoded as a 3-dimensional embedding vector learned from user-watch history and metadata:

- **Movie A: "Laser Quest" (Sci-Fi Action)** a = (1, 2, 3)
- **Movie B: "Sunny Days" (Family Comedy):** b=(2, 1, 0)
- **Movie C: "Nebula Dreams" (Sci-Fi Drama):** c=(1, 2, 2)

The vectors represent latent features such as sci-fi intensity, emotional depth, and action level. Consider the similarity measures of vectors below. Here

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

is exactly the dot product; and $\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ is the norm of vector x.

Cosine similarity: $\cos\_\text{sim}(x, y) = \dfrac{x \cdot y}{\|x\| \|y\|}$

Euclidean distance: $d_E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$

Manhattan distance: $d_{L1}(x, y) = \sum_i |x_i - y_i|$

Dot product: $\text{dot}(x, y) = \sum_i x_i y_i$

(a). Compute the similarities or distances for the following queries.

- How similar is "Laser Quest" to "Nebula Dreams" in Cosine Similarity?

- How different is "Laser Quest" from "Sunny Days" in Euclidean distance?

- Compare "Laser Quest" and "Nebula Dreams" using Manhattan distance?

- How "aligned" are the themes of "Sunny Days" and "Nebula Dreams", in dot product?

(b). You are given the following set of 1-dimensional data points representing the "engagement scores" of users on a social media platform:

D={2, 4, 5, 8, 12, 13}

The company wants to cluster users into **k = 2** groups (low-engagement vs high-engagement) using the **k-means algorithm.** Initially, the two centroids are chosen as: $\mu 1(0)=4$, $\mu 2(0)=12$

Describe the first two rounds of major steps of the k-means algorithm running over this input; in each round, specify: 1. How a cluster is created based on distance two current centroids; 2. Update of centroids; and 3. How clusters are reassigned, if it is not a first step.

(c). K-NN search. A recommendation system embeds users into a 2D space based on their interests. You are given the following points (users) and their **genres of interest**:

User	Coordinates (x, y)	Favorite Genre
A	(1, 2)	Sci-Fi
B	(2, 4)	Comedy
C	(3, 3)	Sci-Fi
D	(6, 5)	Drama
E	(7, 3)	Drama

A **new user X** has embedding: X = (3,1). Using Euclidean distance, find the 3 nearest neighbors of X from {A, B, C, D, E}. Using the neighbors you found, predict X's favorite genre using majority vote (choosing the most frequent favorite genre of users that are similar with X).

3. [IVF and PQ] (40) A streaming service stores movie embeddings in 2D space ("high/low rating on a scale 1-10 in intensity; lightweight/epic rating on a scale of 1-10") for fast similarity search. Below are four movies and their 2-dimensional embedding vectors:

Movie	Embedding (x, y)	Genre
Laser Quest (Movie A)	(1, 2)	Sci-Fi Action
Sunny Days (Movie B)	(2, 1)	Family Comedy
Nebula Dreams (Movie C)	(4, 5)	Sci-Fi Drama
Shadow Empire (Movie D)	(7, 6)	Fantasy / Drama

(a) The IVF index is built with two coarse centroids:

- c1=(1.5, 1.5) (Low-intensity / lightweight movies)
- c2=(6.0, 5.5) (High-intensity / epic movies)

Assign each movie to its nearest centroid using Euclidean distance. List the resulting clusters: Cluster 1 (near c1), and Cluster 2 (near c2).

(b) Consider a new movie Q with embedding Q = (3,3.5). We want a nearest neighbor of Q. An IVF supports this query by: finding nearest centroid(s), and only searching for movies in those clusters. Describe how it can be used to support a fast search to find the NN of Q. You may just list the result for one probe.

(c) Is the above answer the true nearest neighbor? Complete the comparison with a "brute force" distance computation between Q and all the four movies to find out. If not, check how close your answer in (b) is to the true NN of Q in a ranked list.

(d)* Install and play with pgvector (PostgreSQL), to test the above example, you don't need to submit anything for this (d).