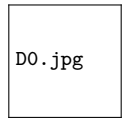
Lecture 19: Modern & AI Data Systems

NoSQL, Cloud Warehouses, Vectors, and RAG Pipelines

Z2004: Database Management Systems



IIT Madras Zanzibar

Monday, 13 April 2026

Learning Objectives

Bridging the Gap to the Modern Industry

So far, we have covered the foundations: introduction to DBMS, ER modelling, relational modelling (SQL), storage, normalization, and the transaction stack (ACID, concurrency control, and recovery). Today we connect those ideas to the modern industry: systems built for web-scale data and for AI workloads.

Today's Agenda:

1. **The Scaling Crisis:** Why Web 2.0 forced us to invent NoSQL.
2. **The 4 Pillars of NoSQL:** Document, Key-Value, Column, and Graph DBs.
3. **Cloud-Native DBs:** Separation of Compute and Storage (Snowflake).
4. **The AI Revolution:** How machines understand language (Vector Embeddings).
5. **Vector Databases:** How to perform mathematical semantic searches.
6. **RAG Pipelines:** Integrating private databases with LLMs like ChatGPT.

Scaling Up vs. Scaling Out

Scaling a Database Management System (DBMS) means increasing capacity to handle higher traffic, larger datasets, and faster query performance.

Vertical scaling (scale up)

- **How:** bigger CPU/RAM/SSD on one server.
- **Why used:** simplest (no app changes), strong consistency.
- **Limit:** expensive, and there is a hardware ceiling.

Horizontal scaling (scale out)

- **How:** add machines to a cluster.
- **Why used:** cheaper per node, elastic growth, higher availability.
- **Common strategies:**
  - *Read replicas:* copy data to serve more reads.
  - *Partitioning:* split data by range/hash to reduce per-node load.
  - *Sharding:* partitioning *across machines*; each shard stores a subset of the data.

Sharding (one-liner): divide a large table into many smaller pieces (shards) on different servers, typically by a *shard key* (e.g., `user_id`).

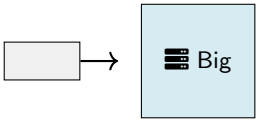
The Relational Scaling Crisis

Vertical vs. Horizontal Scaling

In 2005, Facebook and Twitter started generating millions of rows per second. Relational databases (like PostgreSQL) hit a physical wall.

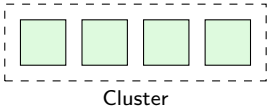
Vertical Scaling (SQL)

- To handle more data in SQL, you must buy a bigger, more powerful server.
- **The Limit:** Eventually, you cannot buy a CPU fast enough or a motherboard with enough RAM slots. The hardware maxes out.



Horizontal Scaling (NoSQL)

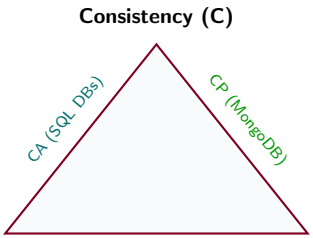
- Add hundreds of cheap, standard laptops/servers together into a "cluster".
- **The Problem:** SQL `JOIN`s break when table data is split across 50 different machines over a network.



The CAP Theorem (The Database Dilemma)

You can only pick TWO

In 2000, Dr. Eric Brewer proved that any distributed database system can only guarantee **two out of three** properties. To scale horizontally, NoSQL sacrifices Consistency.



- **Consistency:** Every client sees the exact same data at the same time.
- **Availability:** The system never goes down; every request gets a response.
- **Partition Tolerance:** If the network cable between Server A and Server B breaks, the system still functions.

Example (AP): On Instagram, if someone likes your photo, Server A might update instantly, but Server B (in another country) might take 5 seconds to show it. It's *Available*, but not *Consistent*.

What is NoSQL?

Databases Beyond the Relational Model

NoSQL (*Not Only SQL*) is a family of database systems designed for **distributed, horizontally scalable** workloads, often prioritizing flexible data models and high availability.

Key characteristics

- **Flexible schema:** Allows storing diverse data types without predefining structures (schema-less).
- **Scale-out first:** designed to run on clusters.
- **Replication & partitioning:** built-in distribution features.
- **Trade-offs:** may relax consistency (eventual consistency) for availability/latency.

Benefits & when to use

- **High throughput** reads/writes at web scale.
- **Low-latency** access for simple queries.
- **Handles semi-structured data** (rapidly changing fields).
- **Geo-distribution** and fault tolerance.

Common use cases: user profiles and sessions, product catalogs, big data management, event logging/IoT streams, social graphs, caching, and large-scale content feeds.

Abandoning the Rigid Table (MongoDB)

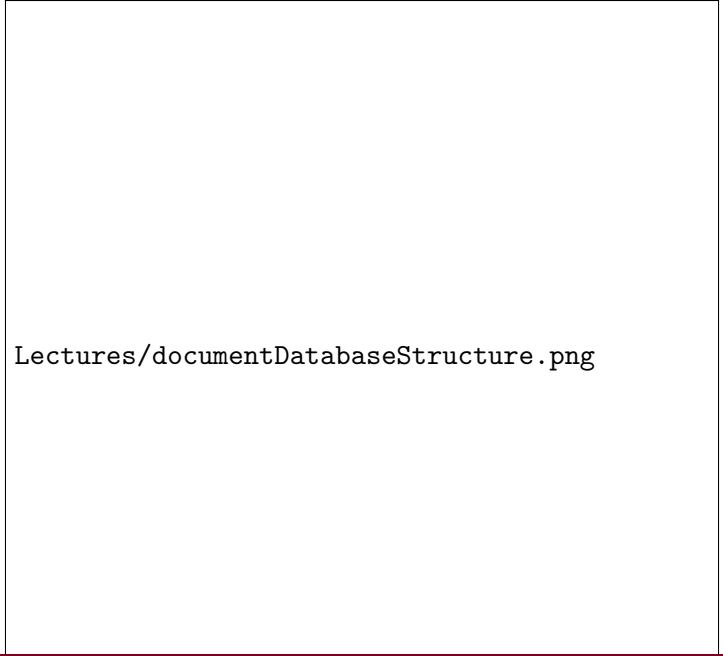
Relational DBs force 1NF (Atomic data). But E-commerce apps have highly unstructured data. A laptop has "RAM" specs; a T-shirt has "Size" specs. Storing them in a rigid SQL table results in hundreds of empty `NULL` columns.

The Document Solution

- Data is stored as **JSON** objects.
- Schemaless: Document 1 can have 3 fields; Document 2 can have 10 fields.
- **Arrays are allowed!** We don't need a separate "Reviews" table; we just nest the reviews inside the document.

```
{
  "_id": "PROD-992",
  "name": "Wireless Mouse",
  "price": 25.99,
  "specs": {
    "dpi": 4000,
    "battery": "AA"
  },
  "tags": ["electronics", "office"],
  "reviews": [
    {"user": "Alice", "rating": 5}
  ]
}
```

doc.png


Lectures/documentDatabaseStructure.png

NoSQL Pillar 2: Key-Value Stores

Pure RAM for Extreme Speed (Redis)

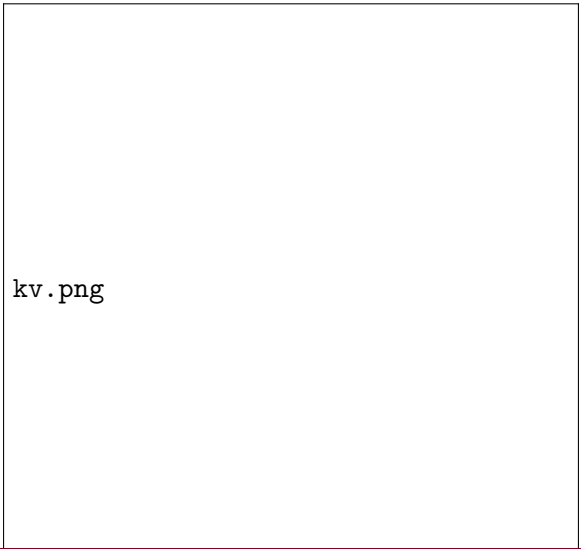
Sometimes, you don't need complex queries. You just need to retrieve a piece of data instantly based on an ID. Key-Value stores are massive, distributed Hash Tables that often live entirely in **RAM**.

Key (String)	**Value** (Blob/JSON)
session:user12	{"theme":"dark", "cart":2}
game:highscore	995,400

Use Cases

- **Session Caching:** When you log into Amazon, your session ID is checked against Redis in 0.1 milliseconds so the site doesn't have to query the main database on every single page load.
- **Shopping Carts:** Fast, temporary storage that disappears if inactive.

Key–Value Store Example



NoSQL Pillar 3 & 4: Column & Graph DBs

Wide-Column (Cassandra)

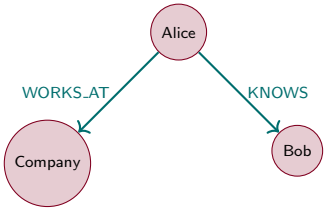
Optimized for massive volumes of time-series data spread across global servers.

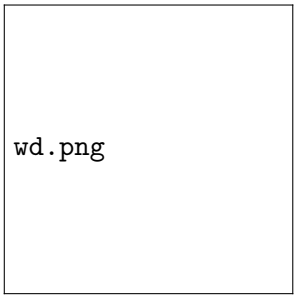
- **Example:** Netflix tracks every second of video you watch. This creates billions of rows a day.
- Cassandra groups data by columns on disk, so aggregating "Total minutes watched" across millions of users takes one quick disk read, rather than scanning whole rows.

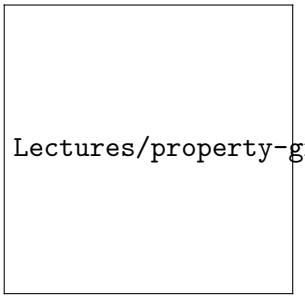
Graph Databases (Neo4j)

Standard SQL is terrible at deep relationships. A query finding "Friends of Friends of Friends" requires 3 expensive `JOIN`s.

- Graph DBs store **Nodes** (Entities) and **Edges** (Relationships).

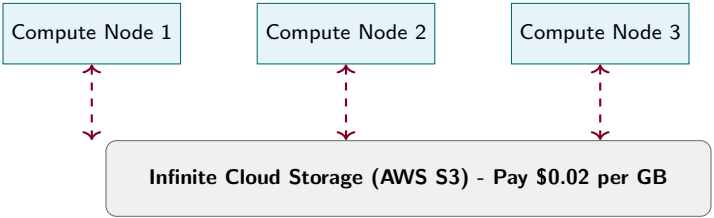



wd.png


Lectures/property-graph.png

Decoupling Storage from Compute (Snowflake)

In old databases, the Hard Drive and the CPU were trapped in the same physical box. If your disk got full, you had to buy a whole new server, even if your CPU was bored. Modern Cloud Data Warehouses (like Snowflake and AWS Aurora) split them via the network.



The Power: On Black Friday, you can spin up 100 Compute Nodes to analyze data for 1 hour, then shut them off. You only pay for what you use, and the data remains safe in storage!

cluouddb.png

The AI Revolution in Data

Why SQL fails at "Meaning"

Traditional databases use **Lexical Search** (Exact keyword matching).

- Query: `SELECT * FROM Products WHERE desc LIKE '%comfortable%';`
- If a product description says "Plush and cozy sofa", SQL will **fail** to return it because the exact string "comfortable" is missing.

The Human Brain Approach (Semantic Search)

Humans know "cozy" and "comfortable" mean the same thing. To make databases smart enough to understand meaning, we have to translate human language into the only thing computers truly understand: **Math and Geometry**.

This brings us to the greatest breakthrough in modern data: The Vector Embedding.

What is a Vector Embedding?

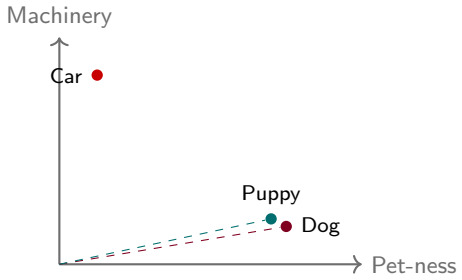
Translating Concepts into Numbers

A Neural Network (like OpenAI's models) reads a word, a sentence, or an image, and maps its "meaning" onto a multi-dimensional mathematical grid. It outputs an array of numbers called a **Vector**.

Example Array (1536 dimensions):

"Puppy" → [0.45, -0.12, 0.88, ... 0.03]
"Dog" → [0.44, -0.11, 0.89, ... 0.04]
"Car" → [-0.99, 0.75, 0.11, ... 0.88]

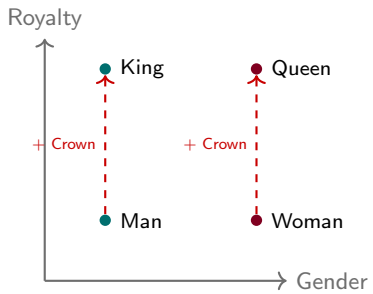
Notice how the numbers for Puppy and Dog are almost identical. In the mathematical space, they are plotted extremely close to each other!



The Magic of Vector Math

King - Man + Woman = Queen

Because embeddings map relationships geographically, we can perform actual arithmetic on concepts.



Why this matters for Databases:

- If a user uploads an image of a red dress and searches "Find me shoes that match this vibe."
- The DB converts the image to a vector.
- It mathematically searches for products in the database that sit in the exact same geometric region of the multidimensional space.

Enter the Vector Database

Pinecone, Milvus, Qdrant, & pgvector

If embeddings are just arrays of numbers, why can't we store them in a standard SQL database?

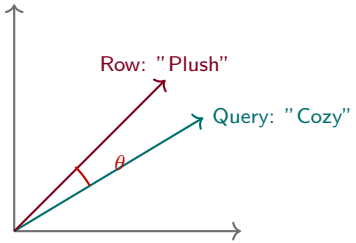
- **Size:** A single OpenAI embedding has 1,536 dimensions. Storing 10 million arrays of 1,536 floating-point numbers requires massive specialized storage.
- **Math Speed:** To find a match, the database must calculate the distance between the user's search vector and **every single row** in the database.
- Traditional B-Tree indexes (from Lecture 17) cannot index 1,536 dimensions.

Vector DB Architecture:
Specialized Vector DBs use completely different indexing algorithms (like **HNSW** - Hierarchical Navigable Small World) to group vectors into geometric clusters, allowing them to search millions of documents in milliseconds.

Measuring Semantic Distance

Cosine Similarity

When you execute a Vector Search, the database calculates the geometric angle between the User Query arrow and the Database Record arrow. This is called **Cosine Similarity**.



- **Cosine(0°) = 1.0 (Perfect Match):** The arrows point in the exact same direction. The concepts are identical.
- **Cosine(90°) = 0.0 (Unrelated):** The arrows are perpendicular. (e.g., "Cozy" and "Carburetor").
- **Cosine(180°) = -1.0 (Opposite):** The arrows point in opposite directions. (e.g., "Cozy" and "Agonizing pain").

```
SELECT chunk_text FROM knowledge_base
ORDER BY embedding <-> '[0.4, 0.1, ...]'  LIMIT 3;
```

The LLM Limitation: Hallucination

Why ChatGPT can't do your job (yet)

Large Language Models (LLMs) are incredibly smart, but they suffer from two fatal database flaws:

1. **Knowledge Cutoff:** They are trained on static data. ChatGPT doesn't know what happened yesterday unless it searches the web.
2. **Data Privacy:** An LLM does not have access to your company's private HR policies, financial records, or secret source code.

The Danger: If you ask an LLM a specific question about private data, it will try to please you by inventing a very confident, fake answer. This is called a **Hallucination**.

How do we connect the reasoning power of an LLM to the truthful data of a Database?

The Solution: RAG Pipelines

Retrieval-Augmented Generation (RAG)

RAG is the standard industry architecture for building Enterprise AI. It merges Vector Databases with LLMs. Instead of relying on the LLM's memory, we force it to act as a summarizer of *our* retrieved data.

1. Retrieval

- User asks a question.
- We convert it to a Vector.
- We search the Vector DB for the top 3 most relevant factual paragraphs.

2. Augmentation

- We take the user's question, and we staple the 3 factual paragraphs to it as "Context".

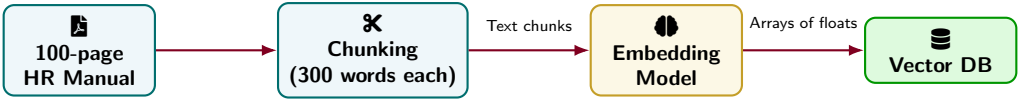
3. Generation

- We send the massive prompt to the LLM.
- The LLM reads the facts and generates a perfect, hallucination-free answer.

RAG Step 1: The Ingestion Phase

Preparing the Private Data

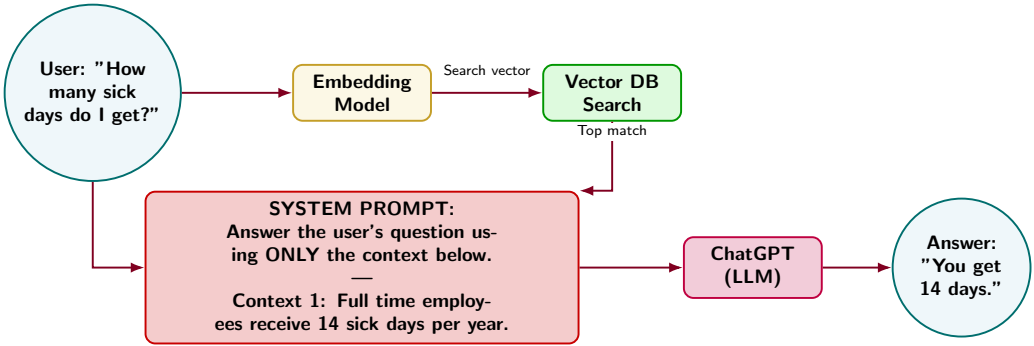
Before anyone can search, we must prepare the company's private data.



- **Chunking:** You cannot embed a whole 100-page PDF at once. We slice it into small chunks (paragraphs).
- **Embedding:** Every chunk is sent to an AI model to be mapped into geometric space.
- **Storage:** The Vector DB stores the text chunk AND its corresponding mathematical array.

RAG Step 2: The Retrieval & Query Phase

Answering the User



Merging Vectors with SQL

Vector search is amazing for meaning, but terrible for exact facts.

- If a user says "Show me cozy sweaters under $50".
- Vector search will find "cozy sweaters".
- But Vector math doesn't understand the strict numerical boundary of "under $50". It might return a $200 sweater because the text descriptions are similar!

The Modern Standard: Hybrid Search

We combine standard SQL indexing (from Lecture 17) with Vector Search.

```
SELECT text FROM products
WHERE price < 50   – Lexical SQL Filter (B-Tree)
ORDER BY embedding <-> '[user_vector]';   – Semantic Vector Search
```

rag.png

Modern Tools of the Trade

What you will use in the industry

As you graduate and build AI applications, these are the dominant database technologies in the industry today:

Cloud Data Warehouses

- **Snowflake / Google BigQuery:** Massive analytics. Petabytes of data.
- **AWS DynamoDB:** Serverless NoSQL.

Dedicated Vector DBs

- **Pinecone:** The standard managed vector database for RAG.
- **Milvus / Qdrant:** Open-source vector engines.

The Hybrid Kings (SQL + Vectors)

- **pgvector (PostgreSQL):** An extension that allows the world's most popular relational DB to store and search vectors. Extremely popular for startups!

Orchestration Frameworks

- **LangChain / LlamaIndex:** Python libraries that wire together your database, your embedding model, and ChatGPT to build RAG pipelines effortlessly.

Lecture Summary

Key Takeaways

- **NoSQL:** Born from the need to scale horizontally and handle flexible schemas. Sacrifices some ACID compliance for scale.
- **Cloud Architecture:** Separates physical storage (cheap) from compute power (expensive, spun up on-demand).
- **Vector Embeddings:** Translating human concepts (text/images) into multidimensional arrays of math.
- **Semantic Search:** Finding data not by keywords, but by measuring the angle (Cosine Similarity) between vectors.
- **RAG Pipelines:** Searching a Vector DB for facts, and injecting those facts into an LLM prompt to prevent hallucinations.

Wrap-Up

So far, we've built the database theory pipeline end-to-end: from the introduction, through ER modeling, the relational model and schemas, normalization, relational algebra/calculus, storage and indexing, SQL engine processing, query optimization, and backup & recovery, and now this lecture on AI & modern data systems.

What we've covered (so far):

- Introduction & data models
- ER modeling
- Relational model: schemas, keys, constraints
- Normalization (1NF to 4NF)
- Relational algebra & relational calculus
- Storage, pages, buffer pools
- Indexing (B trees)
- Query processing & optimization
- Transactions, ACID, logging, backup & recovery
- NoSQL and vector databases

Assessments this week

ICA 2: Tuesday, 14 April 2026 (9:00–9:50).
Quiz 1: Thursday, 16 April 2026.
Coverage: All topics from the introduction up to **Normalization**.

What's next: After Quiz 1, we start **SQL basics**.