Foundations of Large-Scale Information Storage and Retrieval

Searching

- The most common database operation.
- In SQL, `SELECT` is the most complex and versatile statement.
- **Linear Search** is the baseline for efficiency:
    - Starts at the beginning of a list and proceeds element by element.
    - Ends when the target is found or the last element is reached.

Database Concepts

- **Record**: A row in a table, a collection of attribute values for an entity.
- **Collection**: A table, or a set of records of the same entity type.
- **Search Key**: A value for an attribute (one or more attributes).

Memory Allocation in Lists

- **Contiguously Allocated List**:
    - All `n*x` bytes allocated in a single chunk.
    - Fast for random access but slow for insertions.
- **Linked List**:
    - Requires `x` bytes per record plus space for pointers.
    - Fast for insertions but slow for random access.

Binary Search

- Requires sorted data.
- Algorithm:
    - Find middle element.
    - If `mid == target`, return index.
    - If `mid < target`, search right half.
    - Else, search left half.
- **Time Complexity**:
    - **Linear Search**: O(n) in the worst case.
    - **Binary Search**: O(log n) in the worst case.

Database Searching & Indexing

- Searching by a **primary key** (e.g., `id`) is fast.

- Searching by a **non-indexed attribute** (e.g., `specialVal`) requires a linear scan.
- Indexing strategies:
    - **Sorted array of tuples** (`specialVal`, row number) → fast search but slow insertions.
    - **Linked list of tuples** (`specialVal`, row number) → slow search but fast insertions.
    - **Binary Search Tree (BST)** → balances fast insertions and searches.

Document Databases & MongoDB

What is a Document Database?

- A **NoSQL database** that stores data as structured documents.
- Uses **JSON** (JavaScript Object Notation) format.
- **Advantages**:
    - Simple, flexible, and scalable.
    - Well-suited for applications using JSON/XML as a transport layer.

JSON vs. BSON

- **JSON**:
    - Human-readable, lightweight.
    - Uses name/value pairs and ordered lists.
- **BSON** (Binary JSON):
    - Binary-encoded JSON.
    - Supports additional data types (Date, BinaryData).
    - Designed for efficiency, traversal, and storage optimization.

Why Use Document Databases?

- Avoids **impedance mismatch** between object-oriented programming and relational databases.
- Documents are **self-describing**, making them flexible for dynamic data storage.

MongoDB Overview

- Developed in **2007** by former DoubleClick engineers.
- **MongoDB Atlas** (2016): Fully managed cloud service.
- Structure:
    - **Database → Collections → Documents**.
    - Documents in a collection **don’t require a fixed schema**.

MongoDB Features

- **Rich Query Support**: Full CRUD operations.
- **Indexing**: Supports primary and secondary indices.
- **Replication**: Automatic failover via replica sets.
- **Load Balancing**: Built-in.

Interacting with MongoDB

- **CLI Tools**:
  - `mongosh` (MongoDB Shell).
  - `MongoDB Compass` (GUI).
- **Query Examples**:
  - `db.users.find({"name": "Davos Seaworth"})` → Filters documents by name.
  - `db.movies.find({ "rated": { $in: ["PG", "PG-13"] }})` → Retrieves movies with specific ratings.
  - `db.movies.find({ "countries": "Mexico", "imdb.rating": { $gte: 7 } })` → Finds movies released in Mexico with IMDb ≥ 7.

MongoDB in Python (PyMongo)

Connecting to MongoDB:
```
 from pymongo import MongoClient
client = MongoClient('mongodb://user_name:pw@localhost:27017')
```

-

Selecting a collection:
```
 db = client['ds4300']
collection = db['myCollection']
```

-

Inserting a document:
```
 post = {"author": "Mark", "text": "MongoDB is Cool!", "tags": ["mongodb", "python"]}
post_id = collection.insert_one(post).inserted_id
```

-

Counting documents:
```
 count = collection.count_documents({})
```

-

Introduction to Graph Data Models

What is a Graph Database?

- **Graph-based data model** using:
  - **Nodes** (entities).
  - **Edges** (relationships between nodes).
  - **Properties** (metadata on nodes and edges).
- Enables **graph-based queries** (traversals, shortest paths, etc.).

Where Are Graphs Used?

- **Social Networks**: Modeling relationships.
- **The Web**: Pages connected via hyperlinks.
- **Biology & Chemistry**: Interaction modeling.

Graph Theory Basics

- **Labeled Property Graph**:
  - Nodes have **labels** for grouping.
  - Nodes/edges have **properties** (key-value pairs).
  - **Edges must always connect nodes**.
- **Graph Terminology**:
  - **Path**: A sequence of connected nodes (no repetition).
  - **Connected Graph**: A path exists between any two nodes.
  - **Weighted Graph**: Edges have weights.
  - **Directed Graph**: Relationships have direction.
  - **Acyclic Graph**: No cycles.

Graph Algorithms

- **Pathfinding**:
  - Shortest path between nodes (e.g., Dijkstra's algorithm).
  - **Minimum spanning tree**, **cycle detection**, **max/min flow**.
- **Centrality & Community Detection**:
  - **Centrality**: Identifies influential nodes (e.g., social media influencers).
  - **Community Detection**: Clustering and partitioning of nodes.

Famous Graph Algorithms

1. **Dijkstra’s Algorithm**: Single-source shortest path for weighted graphs.
2. *A Algorithm**: Like Dijkstra’s but uses heuristics for efficiency.

3. **PageRank**: Ranks nodes based on incoming links.

Neo4j - A Graph Database System

- **Schema-optional** NoSQL database.
- Supports **ACID transactions** and distributed computing.
- Competes with **Amazon Neptune, Microsoft CosmosDB**.