Database Searching & Indexing - Notes

Database Searching & Indexing

Basic Concepts

- Record: A collection of values for attributes of a single entity.
- Collection: A set of records representing the same entity type.
- Search Key: A single value for an attribute from an entity type.
- Storage:
- Information is stored in records of X bytes across N memory locations.
- Contiguous Allocation:
- Data stored in a continuous memory block.
- Linked List:
- Each record stores X bytes + space for memory addresses.
- Records are linked via memory addresses.
- Pros: Fast insertion anywhere in the list.
- Cons: Slower for random access.
- Arrays:
- Pros: Faster for random access.
- Cons: Slow insertions since other records need shifting.

Searching Algorithms

Binary Search

- Divides the dataset in half each step (left-biased).
- Time Complexity:
- Best Case: O(1)

- Worst Case: O(log n)

Linear Search

- Sequentially checks each record.

- Time Complexity:

- Best Case: O(1)

- Worst Case: O(n)

Database Searching

- Data is stored on disk, indexed by ID values.

- Data cannot be sorted simultaneously by multiple attributes.

- Indexes help improve search performance.

Binary Search Trees (BST)

Example of a BST:

23

/ 17 31

/ \ 14 20 90

Tree Traversal Methods:

- Pre-order

- Post-order

- In-order

- Level-order traversal

AVL Trees (Self-Balancing BSTs)

- A BST that maintains balance using rotations.

- Rotation Cases:
- LL (Left-Left) -> Single Right Rotation.
- RR (Right-Right) -> Single Left Rotation.
- LR (Left-Right) -> Double Rotation.
- RL (Right-Left) -> Double Rotation.

Memory Hierarchy & Optimization

Key Concepts

- Minimizing Disk Accesses:
- HDD/SSD storage is slow.
- We want to reduce secondary storage accesses.
- Key-Value Pointer:
- Uses 64-bit (8-byte) integers for pointers.
- SSD/HDD:
- Persistent (survives power loss).
- Reads in blocks, not individual bytes.

B-Trees & B+ Trees

B-Trees

- Balanced search tree for efficient indexing.
- Internal Nodes:
- Store keys and pointers to children.
- Leaf Nodes:
- Store actual data values.

B+ Trees

- Optimized for disk-based indexing.

- Properties:

- Root node does not need to be full.

- Insertions always happen at the leaf level.

- Minimizes disk accesses.

- Leaves are stored in a Doubly Linked List (DLL).

- Keys in nodes are kept sorted.

Beyond Relational Databases (RDBMS)

Transactions

- A sequence of CRUD operations executed as one unit.

- ACID Properties:

1. Atomicity: All-or-nothing execution.

2. Consistency: Transactions maintain a valid DB state.

3. Isolation: No interference between concurrent transactions.

4. Durability: Committed transactions persist.

Concurrency Issues

- Dirty Read: A transaction reads uncommitted data.

- Non-Repeatable Read: Data changes mid-transaction.

- Phantom Read: New rows appear or disappear mid-transaction.

Scaling Strategies

- Vertical Scaling: Adding more CPU/RAM to one machine.

- Horizontal Scaling: Adding more machines to handle the load.

- Distributed Systems:

- A group of independent machines acting as one system.

- No shared global clock.

- Sharding: Splitting data across nodes.

- Replication: Keeping multiple copies of data.

CAP Theorem

- In distributed databases, you can only have two of:

1. Consistency (C): All nodes return the latest data.

2. Availability (A): Every request gets a response.

3. Partition Tolerance (P): The system functions despite network failures.

Concurrency Control

Pessimistic Concurrency

- Transactions lock data before reading/writing.

- Best for high-conflict environments (e.g., banking).

Optimistic Concurrency

- Transactions assume low conflicts, using timestamps.

- Best for low-conflict environments.

NoSQL Databases

BASE Model (NoSQL Alternative to ACID)

- Basically Available: System works most of the time.

- Soft State: The system can change without input.

- Eventual Consistency: Updates will propagate eventually.

Types of NoSQL Databases

1. Key-Value Stores (e.g., Redis)

2. Graph Databases (e.g., Neo4j)

3. Columnar Databases (e.g., Apache Cassandra)

4. Document Stores (e.g., MongoDB)

5. Vector Databases (e.g., Pinecone)

Redis (Key-Value Store)

- Keys: Strings.

- Values: Strings, lists, sets, hashes, sorted sets.

- Very fast and scalable.

- Stores frequently accessed data.

Final Thoughts

- RDBMS: Strong consistency, well-structured queries (SQL).

- NoSQL: Flexible, scalable, and optimized for distributed systems.

- B-Trees & Hash Tables: Key for efficient search & indexing.

- Distributed Systems: Balancing availability, consistency, and fault tolera