NoSQL & Key-Value Databases

1. Distributed Databases and Concurrency Models

Traditional relational databases use **ACID transactions** with **pessimistic concurrency**, but NoSQL databases often use **optimistic concurrency** for better scalability.

Pessimistic Concurrency (ACID Model)

- **Assumes conflicts will occur**, so it **prevents** them by locking resources.
- **Locks can be read/write locks**, ensuring no two transactions interfere.
- Example: Borrowing a book from a library—if you have it, no one else can.
- **Downside:** Slower due to locks, especially in distributed environments.

Optimistic Concurrency

- **Assumes conflicts are rare** and allows multiple transactions to proceed.
- Instead of locking, transactions use **timestamps and version numbers** to detect conflicts.
- If a conflict is detected at commit time, the transaction is rolled back and retried.

Concurrency Model	Best for...
Pessimistic	High-conflict systems (OLTP, banking)
Optimistic	Low-conflict systems (analytics, backups)

2. Introduction to NoSQL

- **Origin of NoSQL**: First coined in **1998** by Carlo Strozzi for a relational database **without SQL**.
- **Modern Definition**: "Not Only SQL"—a broader term for non-relational databases.
- **Why NoSQL?**
    - Designed for **unstructured web data** (JSON, XML).
    - Supports **horizontal scaling** better than relational databases.
    - Optimized for **availability and scalability** over strong consistency.

3. CAP Theorem (Review)

In a distributed system, you can only achieve **two out of three** guarantees:

Guarantee	Description
Consistency (C)	Every user sees the latest data.
Availability (A)	The system always responds, even if it returns stale data.
Partition Tolerance (P)	System continues operating despite network failures.

Trade-offs:

- **CP (Consistency + Partition Tolerance)**: Drops availability when network issues occur (e.g., MongoDB with strong consistency).
- **AP (Availability + Partition Tolerance)**: Always available, but may return stale data (e.g., DynamoDB).
- **CA (Consistency + Availability)**: Possible only in non-distributed systems.

4. ACID vs. BASE

While relational databases rely on **ACID transactions**, distributed systems use the **BASE** model:

ACID (Traditional RDBMS)	BASE (NoSQL)
Atomicity	**Basically Available** (System remains operational, but responses might be unreliable)
Consistency	**Soft State** (State may change over time due to eventual consistency)
Isolation	**Eventual Consistency** (Data updates propagate gradually)
Durability	**Not guaranteed immediately**

Eventual Consistency

- **Nodes eventually converge** to a consistent state.

- Works well for systems where **absolute real-time accuracy is not required** (e.g., social media feeds, caching).

5. Categories of NoSQL Databases

NoSQL databases fall into different categories based on their data model:

Type	Description	Examples
Key-Value Store	Simple key-value pairs. Fast lookups.	Redis, DynamoDB
Document Store	Stores JSON, BSON, or XML documents.	MongoDB, CouchDB
Column-Family Store	Optimized for large-scale analytical queries.	Cassandra, HBase
Graph Database	Stores relationships between data as nodes and edges.	Neo4j, ArangoDB

6. Key-Value (KV) Databases

Key-value stores are **the simplest type of NoSQL database**.

Structure

- **Key**: A unique identifier.
- **Value**: Arbitrary data (string, JSON, binary).

Example:
user:123 → {"name": "John", "age": 30, "city": "Boston"}

-

Advantages

1. **Speed**: Lookup time is **O(1)** (constant time) using hash tables.
2. **Simplicity**: No complex queries or joins.
3. **Scalability**: Supports **horizontal scaling** by adding more nodes.
4. **Flexibility**: Works well for caching, session storage, and feature stores.

Challenges

- **Lack of Querying**: No structured querying like SQL.
- **Eventual Consistency**: In distributed environments, different nodes may return different values temporarily.

7. Use Cases for Key-Value Stores

Data Science & Machine Learning

- **EDA & Experimentation**: Store intermediate results and A/B testing outcomes.
- **Feature Store**: Cache frequently accessed features for **low-latency** retrieval.
- **Model Monitoring**: Track performance metrics during inference.

Software Engineering

- **Session Storage**: Store session data for fast access.
- **User Profiles & Preferences**: Retrieve user data in a **single GET request**.
- **Shopping Cart Data**: Persistent cart across different devices and sessions.
- **Caching Layer**: Speeds up read-heavy applications.

8. Redis - A Popular Key-Value Database

What is Redis?

- **Re**mote **Di**rectory **S**erver
- **In-memory** database for **ultra-fast performance**.
- Supports multiple data types:
    - **Strings**
    - **Lists**
    - **Sets**
    - **Hashes**
    - **Sorted Sets**
    - **Geospatial data**
    - **JSON (with plugins)**

Key Features

1. **Speed**: Can handle **100,000+ operations per second**.

2. **Durability**: Supports **snapshots** and **append-only file (AOF)** for persistence.
3. **No Secondary Indexes**: Lookups are **only by key**.
4. **Supports Pub/Sub Messaging**: Useful for real-time chat applications.

9. Redis Commands

Basic Key-Value Operations

SET user:1 "John Doe"
GET user:1
DEL user:1
EXISTS user:1

Incrementing & Decrementing

SET counter 0
INCR counter # Increments by 1
INCRBY counter 10 # Increments by 10
DECR counter # Decrements by 1
DECRBY counter 5 # Decrements by 5

10. Redis Data Structures

1. Hashes

- **Used for storing objects with multiple fields.**

Example:
HSET user:1 name "John" age "30" city "Boston"
HGET user:1 name
HGETALL user:1

-

2. Lists

- **Ordered collection (linked list).**
- Useful for **queues, message passing, and logs**.

Example:
LPUSH tasks "task1"
LPUSH tasks "task2"
RPOP tasks

-

3. Sets

- **Unordered collection of unique values**.
- Supports **set operations** like **union, intersection, and difference**.

Example:
SADD users "Alice" "Bob"
SISMEMBER users "Alice"
SCARD users

-

11. Summary

Concept	Description
NoSQL	Designed for unstructured, web-scale applications.
CAP Theorem	You can only have **two** of **C, A, or P**.
BASE Model	Focuses on **availability** over strict consistency.
Key-Value Stores	Simple, fast, scalable. Great for caching, session storage, etc.
Redis	High-performance in-memory KV store with rich data structures.