1. Concurrency Control in Databases

- **ACID Model** (Atomicity, Consistency, Isolation, Durability):
    - Traditional relational databases use ACID properties to ensure transaction safety with pessimistic concurrency control. This model assumes that conflicts are likely, hence it locks resources (read/write locks) during transactions.
    - **Analogy**: Borrowing a library book — "If you have it, no one else can."
- **Optimistic Concurrency**:
    - This model assumes conflicts are rare, and does not lock resources. Conflicts are detected at the end of the transaction using versioning or timestamps, and if detected, the transaction can be rolled back and retried.
    - Suitable for low-conflict systems like backups and analytical databases.

2. Introduction to NoSQL Databases

- **Origins of NoSQL**:
    - Initially coined in 1998 for databases that did not use SQL. Now understood as "Not Only SQL" or non-relational databases.
    - NoSQL evolved to address challenges with unstructured data, especially from web-based applications.

3. The CAP Theorem

- The **CAP Theorem** posits that distributed systems can provide at most two of the following three guarantees:
    - **Consistency**: All users see identical data at any point in time.
    - **Availability**: The system always responds, even in failure scenarios.
    - **Partition Tolerance**: The system can still function when network partitions occur.
- **Trade-offs**:
    - **CA (Consistency + Availability)**: Data is always consistent, but may not handle network partitions well.
    - **CP (Consistency + Partition Tolerance)**: Consistent data, but with possible data loss in network partition situations.
    - **AP (Availability + Partition Tolerance)**: The system remains operational but may not always provide the latest data.

4. BASE as an Alternative to ACID

- For distributed systems, NoSQL databases often use **BASE** properties instead of ACID:
    - **Basically Available**: Guarantees availability, but responses may be unreliable due to inconsistent or changing states.
    - **Soft State**: The state of the system can change over time without external input, allowing for eventual consistency.
    - **Eventual Consistency**: Data will eventually synchronize across nodes/replicas.

5. Key-Value (KV) Databases

- **Core Concept**:
    - KV databases store data as key-value pairs, providing a very simple data model that allows for simple CRUD operations.
- **Design Principles**:
    - **Simplicity**: Only basic operations (e.g., GET, PUT, DELETE) are required.
    - **Speed**: Many KV stores are in-memory databases, allowing for fast retrieval (typically O(1) operations).
    - **Scalability**: They scale horizontally by adding more nodes to the system and typically use eventual consistency.
- **Use Cases**:
    - **EDA Results Store, Feature Store, Model Monitoring**, Session Information, User Profiles, Caching Layer, Shopping Cart Data.

6. Redis Overview

- **What is Redis?**
    - Redis is a popular open-source, in-memory KV store that supports a variety of data types beyond simple key-value pairs.
    - Redis is often referred to as a **Data Structure Store** as it supports structures such as lists, sets, and hashes.
- **Durability in Redis**:
    - Despite being in-memory, Redis offers persistence via snapshots and an append-only file (AOF) for recovery.
- **Performance**:
    - Can achieve high throughput (e.g., >100,000 SET operations per second).
- **Limitations**:
    - Does not support complex queries or secondary indexes. Only supports lookups by keys.

7. Redis Data Types and Commands

- **Strings**:
    - Used for simple caching, counting, and configuration. Commands: SET, GET, DEL, KEYS, INCR, DECR.
- **Hashes**:
    - Stores field-value pairs (ideal for objects, session data). Commands: HSET, HGET, HINCRBY.
- **Lists**:
    - Linked list structure for implementing stacks, queues, or message passing. Commands: LPUSH, RPOP, LLEN.
- **Sets**:
    - Unordered collections of unique strings, useful for social networks or access control. Commands: SADD, SINTER, SDIFF.

- **JSON**:
    - Redis supports JSON data types and offers commands like JSON.GET for retrieving data in JSON format.

8. Security Note

- **Redis Security**:
    - Exposing the Redis port (6379) without a proper password can create a major security vulnerability.