1. ACID and Concurrency

Pessimistic Concurrency (Traditional ACID)

- Focuses on **data safety**
- Assumes things will go wrong
- Prevents conflicts via **locks** (read/write)
    - Analogy: borrowing a library book – exclusive access

Optimistic Concurrency

- Assumes **conflicts are rare**
- Doesn't lock on read/write
- Uses **timestamps and version numbers** to detect conflict at commit time
- Works well in **read-heavy/low-conflict** environments (e.g., analytics)
- Locking is better for **high-conflict** systems

2. What is NoSQL?

- Originally meant "No SQL," now means "Not Only SQL"
- Often refers to **non-relational databases**
- Created to handle **semi/unstructured web data**

3. CAP Theorem (Review)

- A distributed system **can't guarantee all three**:
    - **Consistency**: Every read reflects the latest write
    - **Availability**: System continues to respond to all requests
    - **Partition Tolerance**: System operates despite network failures
- Trade-offs:
    - C + A: Can't tolerate partitioning
    - C + P: Might drop availability
    - A + P: May serve stale data

4. BASE – The ACID Alternative for Distributed Systems

- **Basically Available**: System is usually available
- **Soft State**: State can change over time without input
- **Eventual Consistency**: System will become consistent over time

5. Key-Value (KV) Databases

- Simple **key = value** data model
- Prioritizes:
    - **Simplicity**: CRUD ops and API-friendly
    - **Speed**: Often in-memory; O(1) access using hash tables
    - **Scalability**: Easy horizontal scaling; embraces eventual consistency

6. Use Cases for Key-Value Stores

- **Data Science**: EDA results, experiment tracking
- **ML**: Feature stores, real-time model metrics
- **Web Dev**:
    - Session storage
    - User preferences
    - Shopping carts
    - Caching layer

7. Redis – A Popular Key-Value Store

- **Open-source, in-memory database**
- Also supports:
    - Graphs, Full Text Search, Time Series, Geospatial
- **Durability** via:
    - Snapshots
    - Append-only file
- Over 100k+ ops/sec
- Lookup only by key (no secondary indexes)

8. Redis Data Types

- **String**: basic type, maps string → string
- **Hash**: object-like structures (field-value pairs)
- **List**: linked list of strings; supports stacks & queues
- **Set**: unordered, unique strings; supports set operations
- **JSON**: full support, uses JSONPath for querying

9. Redis Setup

- **Docker**:
  - Search + run Redis image
  - Optional: expose port 6379 (default Redis port)
- **DataGrip**:
  - File > New > Data Source > Redis
  - Set port to 6379 and test connection

10. Redis Basics – String Type

- Use for caching, config settings, token mgmt, counters
- Example Commands:
  - `SET key value`, `GET key`, `EXISTS key`, `DEL key`, `KEYS pattern`
  - `INCR key`, `DECRBY key amt`
  - `SETNX key value`: only sets if key doesn't exist

11. Redis Hash Type

- Use to represent objects (e.g., a bike with model, brand, price)
- Example Commands:
  - `HSET bike:1 model Demios`
  - `HGETALL bike:1`, `HINCRBY bike:1 price 100`

12. Redis List Type

- Great for:
    - Stacks, queues, logging, task queues, chat/message history
- Key Commands:
    - `LPUSH`, `RPUSH`, `LPOP`, `RPOP`
    - `LRANGE key start stop`, `LLEN key`

13. Redis Set Type

- Use for:
    - Tracking unique values (IP addresses, user IDs)
    - Access control, friends lists, groups
- Supports full **set ops**:
    - `SADD`, `SISMEMBER`, `SCARD`
    - `SINTER`, `SDIFF`, `SREM`, `SRANDMEMBER`