# Database & Caching Study Notes

## I. Databases: SQL vs. NoSQL

The first question in many design interviews is: *"Which database would you use for this service, and why?"*

| Feature | SQL (Relational) | NoSQL (Non-Relational) |
|---|---|---|
| **Schema** | Rigid/Fixed (Tables) | Flexible (JSON-like, Key-Value) |
| **Scaling** | Vertical (Scaling Up) | Horizontal (Scaling Out/Sharding) |
| **Data Integrity** | ACID Compliant (Strong) | BASE (Eventual Consistency) |
| **Best For** | Finance, ERP, Complex Queries | Real-time Big Data, User Profiles, Chats |

Interview Talk Track:

> "I choose SQL when data integrity is the highest priority—like in a payment system where a transaction must either fully succeed or fully fail. I lean toward NoSQL when the schema is evolving rapidly or when we need to handle massive write volumes that require horizontal scaling across multiple servers."

---

## II. ACID Properties

ACID is the set of guarantees that ensure database transactions are processed reliably.

- **Atomicity:** The "All or Nothing" rule. If one part of a transaction fails, the whole thing rolls back.
- **Consistency:** The database moves from one valid state to another.
- **Isolation:** Transactions happening at the same time don't interfere with each other.
- **Durability:** Once a transaction is committed, it stays committed, even if the power goes out.

Interview Talk Track:

"Think of a bank transfer. Atomicity ensures money isn't debited from my account without being credited to yours. Durability ensures that once the app says 'Success,' that record is written to a disk and won't vanish if the server reboots a second later."

---

# III. Database Optimization: Indexing & Sharding

## 1. Indexing

An index is a data structure (usually a B-Tree) that improves the speed of data retrieval.

- **Pros:** Turns a full table scan (slow) into a targeted search (fast).
- **Cons:** Makes "Writes" (Insert/Update) slower because the index must be updated too.

## 2. Sharding (Horizontal Partitioning)

Splitting a large dataset into smaller chunks (shards) and storing them across different machines.

Interview Talk Track:

"If we're searching for a user by email among 100 million rows, an Index is like the index at the back of a textbook—you find the page number instantly instead of reading every page. If the textbook becomes so heavy it doesn't fit on one shelf, we Shard it by putting Volume A-M on one shelf and N-Z on another."

---

# IV. The CAP Theorem

In a distributed system, you can only provide two of the following three guarantees:

1. **Consistency (C):** Every node sees the same data at the same time.
2. **Availability (A):** Every request receives a response (even if it's not the latest data).
3. **Partition Tolerance (P):** The system continues to work even if the network fails between nodes.

**Note:** In the real world, you **must** choose 'P'. So the choice is usually between **CP** (Consistency) or **AP** (Availability).

Interview Talk Track:

> "In a distributed environment, network failures (Partitions) are inevitable. For a banking app, I'd choose Consistency (CP) because I'd rather show an error message than show a wrong balance. For a social media feed, I'd choose Availability (AP) because it's okay if a user sees a post 2 seconds late, as long as the app stays up."

---

# V. Caching with Redis

Caching is the process of storing copies of data in a high-speed memory layer (RAM) to serve future requests faster.

- **Cache Hit:** The requested data is in the cache (Fast).
- **Cache Miss:** The data isn't in the cache; we must go to the DB (Slow) and then update the cache.

### Cache Eviction (LRU)

Since RAM is expensive and limited, we need a way to delete old data. **Least Recently Used (LRU)** removes the data that hasn't been accessed for the longest time.

> Interview Talk Track:
>
> "We use a cache to protect our database from being overwhelmed. By using a 'Cache-Aside' pattern with Redis, we can reduce our read latency from 100ms (DB) down to 2ms (RAM). I'd implement an LRU eviction policy to ensure our cache only holds the most relevant, frequently accessed data."

---

# Quick Interview Check-up

- **Scenario:** You are building a 'Top 10 Trending' feature for a news site.
- **Analysis:** This needs high speed (Caching), can handle slight delays (AP over CP), and the data structure fits NoSQL or Redis perfectly.