05 - NoSQL & Key-Value Databases

Distributed Databases and ACID - Pessimistic Concurrency

ACID transactions are used in traditional databases to ensure data safety. They follow a pessimistic concurrency model, assuming that if something can go wrong, it will.

- Conflicts are prevented by locking resources until a transaction is complete.
- Write Lock Analogy → Borrowing a book from a library. If you have it, no one else can until you return it.

Optimistic Concurrency

Unlike pessimistic concurrency, transactions do not lock data when they read or write. This method assumes conflicts are unlikely to occur and, even if they do, they can be handled safely.

- How it works:
    - Each table has a last update timestamp and version number.
    - Before committing a change, the system checks if another transaction has modified the data. If so, the transaction is rolled back and retried.

Optimistic vs. Pessimistic Concurrency

- Optimistic Concurrency is better for low-conflict systems (e.g., analytical databases, backups).
- Pessimistic Concurrency is better for high-conflict systems where frequent transaction rollbacks would be inefficient.

NoSQL: Introduction

- The term “NoSQL” was first used in 1998 by Carlo Strozzi for a relational database system that did not use SQL.
- Today, NoSQL means "Not Only SQL" and often refers to non-relational databases.
- NoSQL databases emerged to handle unstructured, web-based data at scale.

CAP Theorem

A distributed database can only guarantee two out of the three:

1. Consistency – Every user sees the same data at any given time.
2. Availability – The system remains operational even if some components fail.
3. Partition Tolerance – The system can function even when network partitions occur.

CAP Trade-offs

- Consistency + Availability → The system always returns up-to-date data, but it may not handle network failures well.
- Consistency + Partition Tolerance → Guarantees latest data across a distributed system, but requests may be dropped if a partition occurs.
- Availability + Partition Tolerance → The system always responds but data might be slightly outdated.

BASE: An Alternative to ACID for Distributed Systems

BASE is a more flexible model for handling distributed databases, trading strict consistency for performance.

- Basically Available → Ensures high availability, even if the response might not always be reliable.
- Soft State → The database state may change over time without new inputs, due to eventual consistency.
- Eventual Consistency → Given enough time, all nodes will converge to the same data state.

Key-Value Databases (KV DBs)

A key-value store is the simplest form of NoSQL database, storing data as a key = value pair.

Why Use Key-Value Stores?

- Simplicity → The data model is very basic compared to relational tables.
- Speed → Often deployed as in-memory databases, where retrieval is an O(1) operation using hash tables.
- Scalability → Designed for horizontal scaling, allowing more machines to be added easily.

Trade-offs

- NoSQL KV stores are great for speed, but lack complex queries or relationships between data.

Use Cases for Key-Value Stores

Data Science & Machine Learning Use Cases

- EDA/Experimentation → Store intermediate results from data preprocessing.
- Feature Store → Store frequently accessed ML features for low-latency retrieval.
- Model Monitoring → Track real-time model performance metrics.

Software Engineering Use Cases

- Session Storage → Store session data efficiently.
- User Profiles & Preferences → Store user-specific settings with fast retrieval.
- Shopping Carts → Maintain cart data across devices and sessions.
- Caching Layer → Improve performance by storing frequently accessed data in-memory.

Redis: A Popular Key-Value Store

Redis (Remote Dictionary Server) is a widely used open-source in-memory key-value store.

Features of Redis

- Supports various data structures beyond simple key-value pairs, such as lists, sets, hashes, and sorted sets.
- Durability Options:
    - Snapshotting → Saves a full copy to disk periodically.
    - Append-Only File (AOF) → Logs every change for recovery.
- Extremely Fast → Handles 100,000+ operations per second.
- Limitations →
    - No complex queries or relationships.
    - Only supports lookups by key (no secondary indexes).

Redis Data Types

String Type

- Simplest type – Stores a single value under a key.
- Use cases:
    - Caching HTML/CSS/JS fragments.
    - Storing user settings and configuration.
    - Counting web page views or rate limiting.

Hash Type

- Stores multiple field-value pairs under one key.
- Use cases:
    - User profiles (e.g., store name, email, preferences).
    - Session tracking (e.g., active login data).

Example Commands

HSET user:100 name "Alice" age 25 email "alice@example.com"
HGET user:100 name
HGETALL user:100

Redis Lists

A list in Redis is a linked list of string values.

Use Cases

- Implementing queues and stacks.
- Social media feeds (e.g., storing posts in chronological order).
- Chat message history.
- Batch processing (queueing tasks for later execution).

Example Commands

```
LPUSH messages "Hello"
LPUSH messages "How are you?"
RPOP messages  # Retrieves "Hello"
```

Redis Sets

A set in Redis is an unordered collection of unique strings.

Use Cases

- Tracking unique visitors to a website.
- Managing access control lists for users.
- Storing friendships and group memberships in social networks.
- Performing set operations like intersection and difference.

Example Commands

```
SADD users:online "Alice" "Bob" "Charlie"
SISMEMBER users:online "Alice"  # Returns true
SCARD users:online  # Counts members in set
```

Redis Sorted Sets

A sorted set is similar to a set but keeps elements in order based on a score.

Use Cases

- Leaderboards (e.g., sorting players by score).
- Task scheduling (e.g., prioritizing jobs).
- Ranking systems (e.g., most popular articles).

Example Commands

```
ZADD leaderboard 100 "Alice" 150 "Bob"
ZRANGE leaderboard 0 -1 WITHSCORES
```

JSON Support in Redis

- Redis supports storing JSON documents.
- Uses JSONPath syntax for querying.
- Stored in a binary tree structure for fast access to sub-elements.

Example Command

```
JSON.SET user:1 . '{"name": "Alice", "age": 25}'
JSON.GET user:1
```

Final Takeaways

- NoSQL databases are designed for flexibility, scalability, and speed.
- Key-value stores provide fast lookups but lack complex querying.
- Redis is an efficient key-value store that supports various data structures.
- Choosing the right concurrency model depends on the system's needs (Optimistic vs. Pessimistic).