interview questions *Advanced Database Topics*

✅ 1. What is the difference between horizontal and vertical scaling? When would you prefer one over the other?

- **Vertical Scaling** (Scale-Up) involves increasing the resources (CPU, RAM, SSD) of a single server. It's easier to implement and often used with traditional **SQL databases**.
- **Horizontal Scaling** (Scale-Out) means adding more servers or nodes to handle increased load. It's common with **NoSQL databases** that support distributed architecture.
- **Preference**:
    - Use **vertical scaling** when simplicity and consistency are top priorities and your workload can still fit on a single server.
    - Use **horizontal scaling** when dealing with large-scale systems where data and traffic exceed a single server's capacity.

✅ 2. Explain leader-follower replication. How does it impact consistency and availability?

- In **leader-follower replication**, all writes go to a **leader node**, which then replicates data to one or more **follower nodes**.
- **Reads** can be served from followers to reduce load on the leader.
- **Consistency Impact**:
    - If replication is **asynchronous**, followers might be slightly behind, leading to **eventual consistency**.
    - **Synchronous** replication provides strong consistency but can impact performance.
- **Availability Impact**:

- If the leader fails, a new leader must be elected, which might cause temporary downtime.

✅ 3. What are the pros and cons of using read replicas?

- **Pros**:
  - Improves read scalability by offloading traffic from the primary DB.
  - Increases fault tolerance—if the primary goes down, some reads can continue.
- **Cons**:
  - Data on replicas may be **eventually consistent** due to replication lag.
  - Complexities in routing read vs. write operations.
  - No improvement in write scalability.

✅ 4. Compare range-based and hash-based sharding. What are the trade-offs of each?

- **Range-Based Sharding**:
  - Data is divided based on a key range (e.g., user ID 1-1000).
  - **Pros**: Easy range queries and predictable key placement.
  - **Cons**: Can lead to **hot spots** where one shard gets overloaded.
- **Hash-Based Sharding**:
  - Uses a hash function on the sharding key to determine the shard.
  - **Pros**: Better load distribution and avoids hot spots.
  - **Cons**: Difficult to perform range queries and harder to debug data locality.

✅ 5. Why is consistent hashing important in distributed databases?

- **Consistent Hashing** minimizes the number of keys that need to be re-assigned when nodes are added or removed.
- In standard hashing (`key % N`), adding/removing a node affects most key mappings.
- With consistent hashing:
    - Only a small portion of keys are remapped.
    - It improves **resilience**, **elasticity**, and **availability**.
- Used in systems like **Cassandra**, **DynamoDB**, and **Redis clusters** for partitioning and rebalancing.

✅ 6. How does the CAP theorem influence the design of distributed databases? Can you give an example of a CP or AP system?

- **CAP Theorem** states that in a distributed system, you can only guarantee **two** out of the three:
    - **Consistency**: Every read receives the latest write.
    - **Availability**: Every request receives a (non-error) response.
    - **Partition Tolerance**: System continues functioning despite network failures.
- **CP System**: Prioritizes consistency and partition tolerance. Example: **MongoDB (with strong consistency settings)**.
- **AP System**: Prioritizes availability and partition tolerance. Example: **Cassandra**, **DynamoDB**.

✅ 7. What is polyglot persistence? Why might an architecture choose to use multiple types of databases?

- **Polyglot Persistence** is the practice of using **different types of databases** within the same system, each optimized for a specific use case.
- **Reasons**:

    - A **relational DB** for structured data (users, transactions).
    - A **document store** (e.g., MongoDB) for flexible product catalogs.
    - A **search engine** (e.g., Elasticsearch) for full-text search.
    - A **key-value store** (e.g., Redis) for fast session caching.
- Improves **performance**, **scalability**, and **developer productivity** by leveraging strengths of specialized databases.

✅ 8. How do systems like Netflix or Uber use a mix of database technologies in production?

- **Netflix**:
    - Uses **Cassandra** for write-heavy workloads and geo-distribution.
    - **MySQL** for billing and transactional data.
    - **Elasticsearch** for real-time search.
    - **DynamoDB** for metadata storage and resilience.
- **Uber**:
    - **PostgreSQL** and **MySQL** for core business data.
    - **Redis** for geolocation and session caching.
    - **BigQuery** and **Apache Hadoop** for analytics.
- These systems use polyglot persistence to handle diverse data needs across **scale, structure, and speed**.