Contents

Scenario: Instagram with Fewer DB Servers

Describe how you would design Instagram to run efficiently with a limited number of database servers.

Purpose

- Tests your ability to design for resource constraints
- Evaluates your knowledge of scaling, partitioning, and caching
- Assesses your prioritization and trade-off skills

Summary

Partition by user ID. Apply aggressive caching with invalidation. Archive old data. Use CDN for static content. Deprioritize analytics and secondary indexes.

Data Partitioning & Sharding

- **User-Based Partitioning:**
  - Distribute users across available DB servers by user ID hash/modulo
  - Ensures even load and avoids hotspots
- **Hot User Mitigation:**
  - Identify high-traffic users and split their data across multiple logical partitions if needed
- **Minimize Cross-Shard Operations:**
  - Design features to avoid joins or transactions across partitions

---

Caching & Content Delivery

- **Aggressive Caching:**
  - Use in-memory caches (e.g., Memcached, Redis) for timelines, user profiles, and frequently accessed data
  - Apply cache invalidation on writes/updates

- **CDN for Static Content:**
  - Store images, videos, and static assets in object storage (e.g., S3) and serve via CDN
  - Reduces DB and backend load

Data Lifecycle Management

- **Archiving:**
  - Move old posts, comments, and activity logs to cheaper, slower storage (cold storage)
  - Keep only recent/active data in the main DB
- **TTL Policies:**
  - Set time-to-live for ephemeral data (e.g., stories, notifications)

Query Optimization & Feature Trade-offs

- **Deprioritize Analytics:**
  - Run analytics and reporting jobs off-peak or on read replicas if available
- **Limit Secondary Indexes:**
  - Only index fields critical for user experience
  - Avoid expensive queries and full table scans
- **Batch & Rate Limit Writes:**
  - Buffer non-critical writes and process in batches to reduce DB load

---

Trade-offs & Limitations

- **Reduced Real-Time Analytics:** Some insights may be delayed or less granular
- **Eventual Consistency:** Accept some staleness in non-critical data for performance
- **Feature Degradation:** Temporarily disable or degrade non-essential features during peak load

---

Metrics for Success

- DB CPU and memory utilization
- Cache hit ratio
- User-perceived latency
- Error rate and data loss incidents
- Percentage of cold storage vs. hot storage data