[Caching]

↳ Tempesory store copies of data in high-speed storage

↓ ↳ E.g RAM

To decrease access time

---

1. Improves Performance
2. Reduced Load on Backend Systems
3. Increased Scalability
4. Cost efficiency
5. Enhanced User experience

Where can Caching be Done?

1. In-Memory Cache (RAM)
   - Redis software
2. Distributed Cache
   - Multiple servers of Cache
   - Redis cluster, Amazon ElastiCache
3. Client-side Cache
   - Cookies, Session/local storage
   - Mobile application specific cache
4. Database Cache
   - Cache frequently used query result
5. CDN

Caching Strategies

- **Read-Through Cache**: The application first checks the cache for data. If it's not there (a cache miss), it retrieves the data from the database and updates the cache.
- **Write-Through Cache**: Data is written to both the cache and the database simultaneously, ensuring consistency but potentially impacting write performance.
- **Write-Back Cache**: Data is written to the cache first and later synchronized with the database, improving write performance but risking data loss.
- **Cache-Aside (Lazy Loading)**: The application is responsible for reading and writing from both the cache and the database.

Cache Eviction Policies / Cache Replacement policies

To manage the limited size of a cache, eviction policies are used to determine which data should be removed when the cache is full.

1. Least Recently Used (LRU)

LRU evicts the least recently accessed data when the cache is full. It assumes that recently used data will likely be used again soon.

2. Least Frequently Used (LFU)

LFU evicts data that has been accessed the least number of times, under the assumption that rarely accessed data is less likely to be needed.

3. First In, First Out (FIFO)

FIFO evicts the oldest data in the cache first, regardless of how often or recently it has been accessed.

4. Time-to-Live (TTL)

TTL is a time-based eviction policy where data is removed from the cache after a specified duration, regardless of usage.

Challenges and Considerations

1. **Cache Coherence**: Ensuring that data in the cache remains consistent with the source of truth (e.g., the database).
2. **Cache Invalidation**: Determining when and how to update or remove stale data from the cache.
3. **Cold Start**: Handling scenarios when the cache is empty, such as after a system restart.
4. **Cache Eviction Policies**: Deciding which items to remove when the cache reaches capacity (e.g., Least Recently Used, Least Frequently Used).
5. **Cache Penetration**: Preventing malicious attempts to repeatedly query for non-existent data, potentially overwhelming the backend.
6. **Cache Stampede**: Managing situations where many concurrent requests attempt to rebuild the cache simultaneously.

Best Practices for Implementing Caching

- **Cache the Right Data:** Focus on caching data that is expensive to compute or retrieve and that is frequently accessed.
- **Set Appropriate TTLs:** Use TTLs to automatically invalidate cache entries and prevent stale data.
- **Consider Cache Warming**: Preload essential data into the cache to avoid cold starts.
- **Monitor Cache Performance:** Regularly monitor cache hit/miss ratios and adjust caching strategies based on usage patterns.
- **Use Layered Caching:** Implement caching at multiple layers (e.g., client-side, server-side, CDN) to maximize performance benefits.
- **Handle Cache Misses Gracefully:** Ensure that the system can handle cache misses efficiently without significant performance degradation.