Explain in brief the following cache eviction policies -
1. Cache Aside
2. Read Through Cache
3. Write Around Cache
4. Write Through Cache
5. Write Behind Cache

Here's a brief explanation of each cache eviction policy:

1. Cache Aside (Lazy Loading)

- **Concept**: Application manages cache directly
- **Read**: App checks cache first → if miss, reads from DB → writes result to cache
- **Write**: App writes directly to DB → invalidates/updates cache entry
- **Use case**: Most common pattern, flexible but more complex

2. Read Through Cache

- **Concept**: Cache sits between app and DB, handles reads automatically
- **Read**: App always reads from cache → cache fetches from DB on miss
- **Write**: Similar to Cache Aside (direct DB write + cache invalidation)
- **Use case**: Simplified application logic, cache-aware libraries

3. Write Around Cache

- **Concept**: Writes bypass cache completely
- **Write**: Data written directly to DB, cache unaffected
- **Read**: Subsequent reads populate cache (cache miss → load from DB)
- **Use case**: Write-heavy workloads with infrequent reads of new data

4. Write Through Cache

- **Concept**: All writes go through cache to DB

- **Write**: App writes to cache → cache synchronously writes to DB
- **Read**: Always from cache (guaranteed fresh if write succeeded)
- **Use case**: Consistency-critical applications, simpler failure handling

5. Write Behind (Write Back)

- **Concept**: Writes to cache only, DB updated asynchronously
- **Write**: App writes to cache → cache queues/batches DB writes
- **Read**: Always from cache (may have unsynced data)
- **Use case**: High-write throughput, can tolerate some data loss risk
- **Risk**: Data loss if cache fails before DB sync

Key Trade-offs

- **Consistency**: Write Through > Cache Aside > Write Behind
- **Performance**: Write Behind > Write Around > Cache Aside
- **Complexity**: Cache Aside (app) > Write Through (cache)
- **Data Safety**: Write Through > Write Around > Write Behind