Question 1 of 15

What are the key architectural characteristics that make Redis particularly fast?

1. In-memory storage and single-threaded execution ✓
2. Distributed architecture and complex query optimization
3. Multi-threaded processing and disk-based storage
4. Sharded databases and connection pooling

Correct!

Redis achieves high performance through in-memory storage (avoiding disk I/O) and single-threaded execution (eliminating context switching and lock contention), making it very fast and easy to reason about.

Question 2 of 15

What is the underlying data model that Redis is built on?

1 Graph database with nodes and edges

2 Key-value store where keys are strings and values can be various data structures ✓

3 Relational tables with SQL queries

4 Document store with collections

✓ **Correct!**

Redis is fundamentally a key-value store where keys are always strings, but values can be various data structures like strings, hashes, lists, sets, sorted sets, and more.

Question 3 of 15

Redis provides the same durability guarantees as traditional relational databases for committed transactions.

1 True

2 False

Correct!

Redis intentionally trades durability for speed. While there are strategies like AOF (Append-Only File) to minimize data loss, Redis doesn't provide the same disk-based durability guarantees as traditional databases.

Question 4 of 15

In a Redis cluster, how do clients typically determine which node contains the data they need?

1 By caching hash slots that map keys to specific nodes

2 By querying a central coordinator service

3 By using a load balancer that handles routing

4 By broadcasting requests to all nodes

Correct!

Redis clients cache hash slot mappings to directly connect to the node containing their requested data, prioritizing performance by avoiding redirects and coordinator overhead.

Question 5 of 15

Redis clusters can automatically handle cross-node operations and joins across multiple keys stored on different nodes.

1 True

2 False

Correct!

Redis clusters have basic limitations - they expect all data for a given request to be on a single node. This is why choosing how to structure your keys is crucial for scaling Redis effectively.

Question 6 of 15

Which Redis data structure would be most appropriate for implementing a leaderboard that needs frequent score updates and top-K queries?

1 List

2 Set

3 Hash

4 Sorted Set

Correct!

Sorted Sets maintain ordered data and support logarithmic time operations for both updates and range queries, making them ideal for leaderboards where you need to frequently update scores and retrieve top players.

Question 7 of 15

For implementing a simple distributed lock with timeout in Redis, which command combination would you primarily use?

1 SET with NX and EX options

2 HSET with HDEL

3 INCR with TTL and DEL

4 SADD with EXPIRE

Incorrect.

A simple distributed lock can use atomic INCR operations: increment a key, check if the result is 1 (you own the lock), set a TTL for timeout, and DEL when finished to release the lock.

Question 8 of 15

Redis Pub/Sub persists messages so that offline subscribers can receive them when they reconnect.

1 True

2 False

Correct!

Redis Pub/Sub is ephemeral and provides 'at most once' delivery - if a subscriber is offline when a message is published, it will miss that message entirely. For persistence, you'd need Redis Streams or external message brokers.

Question 9 of 15

What is the 'hot key' problem in Redis clusters?

1. Uneven load distribution when one key receives disproportionate traffic ✓
2. Keys expiring too quickly under high load
3. Memory fragmentation from frequently accessed keys
4. Keys becoming corrupted due to high temperatures

Correct!

The hot key problem occurs when one key (and thus one node) receives much more traffic than others, potentially overwhelming that single server while leaving other nodes underutilized.

Question 10 of 15

Which approach is NOT a valid solution for Redis hot key issues?

1 Adding client-side caching to reduce requests

2 Automatically redistributing hot keys across more nodes ✓

3 Storing the same data in multiple keys and randomizing requests

4 Adding read replicas and dynamically scaling ✗

Incorrect.

Redis clusters cannot automatically redistribute individual hot keys across nodes. Solutions involve client-side caching, data replication with randomized access, or read replicas - but not automatic key redistribution.

Question 11 of 15

For implementing a fixed-window rate limiter in Redis, which operations would you use?

1 ZADD and ZCOUNT for time-based scoring

2 LPUSH and LLEN for request queue management

3 INCR and EXPIRE for counting with automatic reset

4 SADD and SCARD for unique request tracking

Correct!

A fixed-window rate limiter uses INCR to count requests and checks if the result exceeds the limit, with EXPIRE setting a TTL so the counter resets after the time window.

Question 12 of 15

Redis Streams provide similar functionality to Kafka topics for event sourcing and message processing with consumer groups.

1 True

2 False

Correct!

Redis Streams are append-only logs similar to Kafka topics, supporting producer-consumer patterns with consumer groups, message claiming for fault tolerance, and durable message processing workflows.

Question 13 of 15

Why might Redis be considered superior to SQL databases for certain high-frequency operations?

1. Redis has microsecond latency and can handle 100k+ writes per second ✓
2. Redis automatically optimizes query execution plans
3. Redis provides stronger consistency guarantees
4. Redis supports complex joins across multiple tables

Correct!

Redis's in-memory architecture enables microsecond-level read latency and over 100k writes per second, making operations that would be inefficient in SQL databases (like multiple small requests) actually feasible.

Question 14 of 15

When using Redis for proximity search with geospatial data, what is the time complexity of a radius search?

1 O(N + log M) where N is items in radius and M is total items

2 O(log M) where M is total items

3 O(N * M) where N and M are coordinate dimensions

4 O(1) constant time

Correct!

Redis geospatial searches run in O(N + log M) time, where N is the number of elements found within the search radius and M is the total number of items in the spatial index.

Question 15 of 15

The choice of how you structure Redis keys is the primary way you scale Redis clusters effectively.

1 True

2 False

Correct!

Since Redis clusters expect all data for a request to be on a single node and use key-based hash slot mapping, how you structure and organize your keys directly determines how well your system scales and distributes load.