- particularly fast?
- In-memory storage and single-threaded execution

Distributed architecture and complex query optimization

- 3 Multi-threaded processing and disk-based storage
 - 4 Sharded databases and connection pooling
 - Carrest

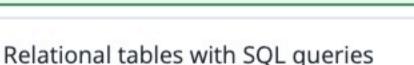
Correct!

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.

Qu	esti	ion	2	of	1

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

- 1 Graph database with nodes and edges
- Key-value store where keys are strings and values can be various



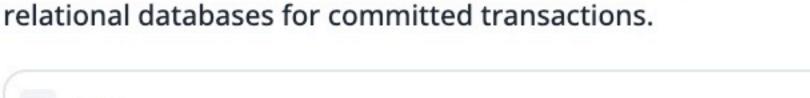
4 Document store with collections

data structures

Correct!

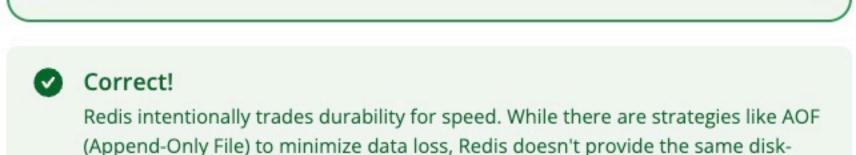
Redis is fundamentally a key-value store where keys are always

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.

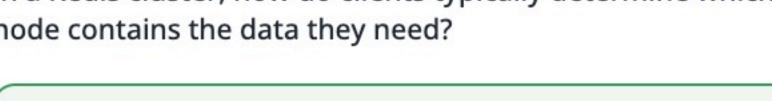


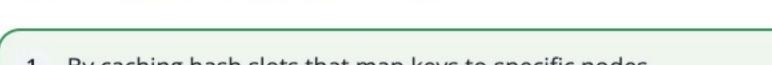


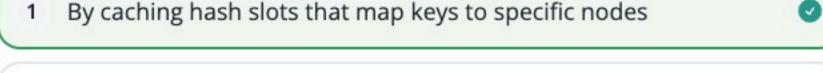
False



based durability guarantees as traditional databases.







- By querying a central coordinator service
 - By using a load balancer that handles routing
- 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	
D 1:				

False

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





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.

Que	st	ion	6	of	1

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

- updates and top-K queries?

 1 List
 - 2 Set
 - 3 Hash
 - 4 Sorted Set



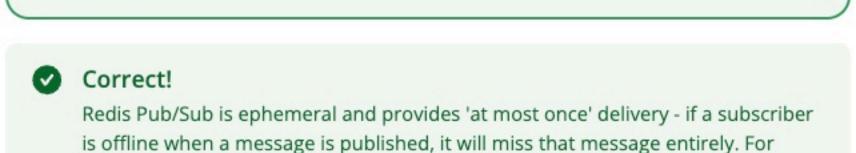
Sorted Sets maintain ordered data and support logarithmic time operations for both updates and range queries, making them ideal for leaderboards where you need to frequently update scores and retrieve top players.

Q	uest	tion 7 of 15	F
		implementing a simple distributed lock with timeout in is, which command combination would you primarily use	?
	1	SET with NX and EX options	
	2	HSET with HDEL	
(3	INCR with TTL and DEL	2

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.

can receive them when they reconnect.							
1	True						

False



persistence, you'd need Redis Streams or external message brokers.

Uneven load distribution when one key receives disproportionate traffic

Keys expiring too quickly under high load

Memory fragmentation from frequently accessed keys

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.

W	hi	ch approach is NOT a valid solution for Redis hot key es?	
	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	

! Incorrect.
Redis clusters cannot automatically redistribute individual hot keys across nodes.
Solutions involve client-side caching, data replication with randomized access, or

Adding read replicas and dynamically scaling

read replicas - but not automatic key redistribution.

Q	ues	tion 11 of 15	F
		implementing a fixed-window rate limiter in Redis, which rations 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	2



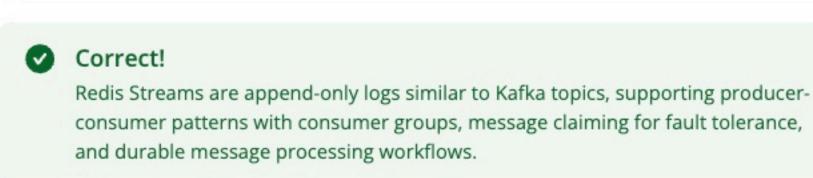
SADD and SCARD for unique request tracking

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.

Redis Streams pro	vide similar functionality to Kafka topics for
And the first section of the section	message processing with consumer groups

Question 12 of 15





Question 13 of 15
Why might Redis be considered superior to SQL databases for

certain high-frequency operations?

- Redis has microsecond latency and can handle 100k+ writes per
 - second
 - 2 Redis automatically optimizes query execution plans

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

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

- 2 O(log M) where M is total items

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	
he choice of how you structure Redis keys is the pr	imary way
ou scale Redis clusters effectively.	

