Systems Design Concepts Practice Test (Sample)

Study Guide

Everything you need from our exam experts!

Table of Contents

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here's how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don't just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There's no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

1. What characterizes a document-oriented database?

A. Stores data as fixed schemas in tables with normalized relationships.

B. Stores data as flexible JSON-like documents with embedded related data.

C. Stores data primarily as key-value pairs with little structure.

D. Replaces relational databases entirely for all workloads.

2. When should you choose WebSockets or Server-Sent Events (SSE)?

A. They are deprecated in favor of polling.

B. They are identical to traditional REST APIs.

C. They enable real-time data delivery via open connections for push updates.

D. They are best used for simple request-response interactions.

3. Which caching pattern is more common with Redis for application-level caching?

A. Read-Through

B. Write-Behind

C. Cache-Aside

D. In-Process Caching

4. Why can't B-tree indexes efficiently support certain LIKE patterns?

A. They can efficiently support patterns that match from the start of a string.

B. They rely on sorted order enabling range scans, but cannot efficiently handle middle wildcard patterns.

C. They can use the index for middle string patterns.

D. They are best for full-text search.

5. Which sharding strategy assigns contiguous ranges of shard key values to specific shards, enabling efficient range scans but risking hot spots when traffic concentrates on a single range?

A. Range-Based Sharding assigns contiguous ranges of shard key values to specific shards.

B. Directory-Based Sharding stores shard assignments in a lookup table for flexible movement.

C. Consistent Hashing uses a virtual ring to minimize data movement.

D. Hot Spot Problem describes unequal traffic across shards.

6. What is the recommended approach to discussing CAP in system design interviews?

A. Ignore CAP in interviews.

B. Raise CAP theorem during non-functional requirements and declare domain-specific priorities for consistency vs availability.

C. Always optimize for consistency regardless of domain.

D. CAP has no bearing on design decisions.

7. What is a key benefit of consistent hashing compared to simple modulo hashing?

A. It moves only a small fraction of keys when a node is added or removed.

B. It remaps almost all keys when topology changes.

C. It relies on modulo hashing for lookups.

D. It requires a fixed cluster size to function.

8. What best defines the hot key problem in caching?

A. A scenario where all keys are accessed in a round-robin fashion.

B. A scenario where one cache key receives disproportionately more traffic than others, overloading the single cache node.

C. A scenario where the cache memory slowly leaks.

D. A scenario where keys are rotated every minute to balance load.

9. Which statement best describes Protocol Buffers usage?

A. A binary serialization format used by gRPC to define service contracts.

B. A text-based data interchange format used by REST.

C. An API style that builds endpoints around actions.

D. A schema language for GraphQL contracts.

10. What RAM range do modern app servers have, and what is the first bottleneck?

A. 64-128 GB; memory is the primary bottleneck.

B. 64-512 GB; CPU is almost always the first bottleneck, not memory.

C. 1-2 TB; disk I/O is the bottleneck.

D. 16-32 GB; network is the bottleneck.

Answers

1. B
2. C
3. C
4. B
5. A
6. B
7. A
8. B
9. A
10. B

Explanations

1. What characterizes a document-oriented database?

A. Stores data as fixed schemas in tables with normalized relationships.

B. Stores data as flexible JSON-like documents with embedded related data.

C. Stores data primarily as key-value pairs with little structure.

D. Replaces relational databases entirely for all workloads.

Document-oriented databases store data as documents, usually JSON-like, where each document contains all the data for an entity and can include nested or embedded related data. There's no fixed schema, so fields can vary between documents and the structure can evolve over time. This setup enables retrieving a whole object in a single read without needing joins, which is ideal for complex or hierarchical data. In contrast to relational databases, which organize data into tables with predefined schemas and normalized relationships, document stores embrace flexible structures and embedded data. They're not meant to replace relational databases for every workload, but they excel in scenarios with dynamic schemas and nested data.

2. When should you choose WebSockets or Server-Sent Events (SSE)?

A. They are deprecated in favor of polling.

B. They are identical to traditional REST APIs.

C. They enable real-time data delivery via open connections for push updates.

D. They are best used for simple request-response interactions.

Real-time data delivery relies on an open, persistent connection that lets the server push updates to the client as soon as they happen. WebSockets give you a continuous, two-way channel where either side can send messages at any time. Server-Sent Events create a long-lived, server-to-client stream over HTTP, which lets the server push updates to the client automatically. Both are built for push-based updates rather than the usual request/response pattern of REST. You'd choose them whenever your app needs live data—think dashboards that refresh in real time, chat, live feeds, or notifications. If you need true bidirectional communication (client and server talking back and forth), go with WebSockets. If you only need the server to push updates to clients and keep things simple, SSE can be a lighter option. They're not deprecated and they're not the same as traditional REST polling or short-lived requests.

3. Which caching pattern is more common with Redis for application-level caching?

A. Read-Through

B. Write-Behind

C. Cache-Aside

D. In-Process Caching

This question tests how Redis is typically used for application-level caching. The common approach is Cache-Aside. In this pattern the application first checks Redis for the data. If the data is in the cache (a hit), you return it quickly. If it's not there (a miss), the application fetches the data from the primary data store, returns it to the caller, and then stores a copy in Redis for subsequent requests. This gives tight control over when data moves into the cache and how freshness is managed, which is ideal when Redis is a fast, shared cache in front of a database. On updates, the usual practice is to write to the primary data store and then invalidate or refresh the corresponding cache entry. This keeps the cache in sync with the source of truth without introducing complexity like asynchronous writes. Other patterns are less natural in this context. Read-Through would require the cache layer itself to automatically fetch from the data store on a miss, which typically means adding a caching layer or library that sits in front of Redis; plain Redis doesn't automatically provide this behavior. Write-Behind (asynchronous writes to the underlying store) introduces complexity and potential consistency issues, making it less common for straightforward application caching. In-Process caching lives inside a single application instance and doesn't leverage Redis as a shared cache, so it doesn't align with using Redis as the central application-level cache.

4. Why can't B-tree indexes efficiently support certain LIKE patterns?

A. They can efficiently support patterns that match from the start of a string.

B. They rely on sorted order enabling range scans, but cannot efficiently handle middle wildcard patterns.

C. They can use the index for middle string patterns.

D. They are best for full-text search.

B-tree indexes are designed to take advantage of the sorted order of keys to do fast range scans for patterns with a fixed prefix. When a LIKE pattern starts with a constant string, the database can translate that into a narrow range of index keys and scan just that portion, which is very efficient. But if the pattern has a wildcard in the middle or at the front, there isn't a single contiguous range of keys that will match. The match depends on characters after the wildcard, so the index can't prune the search effectively and would have to examine many entries or even the whole table. That's why B-tree indexes can't efficiently support certain LIKE patterns, particularly those with leading or middle wildcards. For those cases, other indexing methods or full-text/substring indexes are typically used.

5. Which sharding strategy assigns contiguous ranges of shard key values to specific shards, enabling efficient range scans but risking hot spots when traffic concentrates on a single range?

A. Range-Based Sharding assigns contiguous ranges of shard key values to specific shards.

B. Directory-Based Sharding stores shard assignments in a lookup table for flexible movement.

C. Consistent Hashing uses a virtual ring to minimize data movement.

D. Hot Spot Problem describes unequal traffic across shards.

Range-based sharding partitions data by ranges of the shard key values, assigning each range to a specific shard. This layout makes range queries efficient because all data for adjacent keys sits on the same shard, reducing the need to touch many shards. But the flip side is a potential hot spot when traffic concentrates on a single range, causing that shard to become overloaded while others are underutilized. Flexible lookup-based approaches exist, but they don't inherently optimize for range scans; consistent hashing distributes keys to balance load and minimize movement without preserving contiguous ranges; and "hot spot" describes the effect of uneven access patterns rather than a specific sharding method.

6. What is the recommended approach to discussing CAP in system design interviews?

A. Ignore CAP in interviews.

B. Raise CAP theorem during non-functional requirements and declare domain-specific priorities for consistency vs availability.

C. Always optimize for consistency regardless of domain.

D. CAP has no bearing on design decisions.

CAP is about trade-offs among Consistency, Availability, and Partition tolerance in a distributed system. In a system design interview, you don't sidestep this reality—you bring it up during the non-functional requirements discussion and state, for the given domain, which two goals you will optimize for and what that means for data freshness and user experience. For example, if the system must always process transactions correctly even during partitions, you'd lean toward Consistency and Partition tolerance, accepting that some requests may be slow or temporarily unavailable. If the goal is a highly responsive user experience with many reads, you might favor Availability and Partition tolerance, embracing eventual consistency and using strategies to resolve conflicts later. Explain how you would implement that choice: use strong consensus or quorum for strict consistency, or employ eventual consistency with conflict resolution, versioning, and idempotent operations. Tie your discussion to practical concerns like SLAs, RPO/RTO, and how you handle failures, latency, and data integrity. This approach shows you understand when and why CAP matters for different domains, rather than treating it as irrelevant or always optimizing for one dimension.

7. What is a key benefit of consistent hashing compared to simple modulo hashing?

A. It moves only a small fraction of keys when a node is added or removed.

B. It remaps almost all keys when topology changes.

C. It relies on modulo hashing for lookups.

D. It requires a fixed cluster size to function.

Consistent hashing minimizes data movement when the set of nodes changes. With simple modulo hashing, keys are assigned to nodes by hashing the key and taking modulo the number of nodes, so adding or removing a node changes N and can remap almost every key to a different node, causing a large data reallocation. In contrast, consistent hashing places both keys and nodes on a ring and maps each key to the first node clockwise from its hash. When a node is added or removed, only keys that fall into the small segment around that node are moved to neighboring nodes, leaving most mappings intact. This is why only a small fraction of keys get relocated, a major benefit for scalability and fault tolerance. The other statements describe properties that don't hold for consistent hashing: one implies widespread remapping, another wrongly claims reliance on modulo hashing, and another suggests a fixed cluster size is required.

8. What best defines the hot key problem in caching?

A. A scenario where all keys are accessed in a round-robin fashion.

B. A scenario where one cache key receives disproportionately more traffic than others, overloading the single cache node.

C. A scenario where the cache memory slowly leaks.

D. A scenario where keys are rotated every minute to balance load.

Hot key problem in caching happens when one key draws far more requests than all others. That creates a hotspot on the cache node or shard that stores that key, so that node becomes the bottleneck. With most requests funneling to a single location, memory, CPU, and network can saturate there, increasing latency for every request and causing churn as evictions happen and the backend is hit more often. In a balanced cache, traffic is spread across many keys and nodes, so no single component is overwhelmed. The other scenarios don't describe this skew: round-robin access would distribute load more evenly, avoiding a single hotspot; a memory leak is a different failure mode; rotating keys to balance load is a strategy to prevent hotspots, not the problem itself. To mitigate hot keys, you'd partition the cache across more nodes, replicate hot keys, or apply tactics like per-key throttling, pre-warming, or a secondary cache layer to absorb bursts.

9. Which statement best describes Protocol Buffers usage?

A. A binary serialization format used by gRPC to define service contracts.

B. A text-based data interchange format used by REST.

C. An API style that builds endpoints around actions.

D. A schema language for GraphQL contracts.

Protocol Buffers encode data in a compact binary wire format and provide a schema for both the data structures and the services that operate on them. In gRPC, you describe your messages and your service RPCs in .proto files, and the tooling uses that schema to generate client and server stubs while serializing all calls as the binary protobuf payloads. This combination is why this statement is the best fit: it captures that Protocol Buffers are a binary serialization format used by gRPC to define service contracts. By contrast, REST typically uses text formats like JSON or XML, not the binary protobuf wire format; GraphQL relies on its own schema language rather than Protocol Buffers.

10. What RAM range do modern app servers have, and what is the first bottleneck?

A. 64-128 GB; memory is the primary bottleneck.

B. 64-512 GB; CPU is almost always the first bottleneck, not memory.

C. 1-2 TB; disk I/O is the bottleneck.

D. 16-32 GB; network is the bottleneck.

The main idea is that even with plenty of RAM, the limiting factor for handling more requests on a modern app server is usually the CPU. Having lots of memory helps with caches, in-memory data, and heap sizing, but it doesn't by itself increase how many requests you can process per second if the CPU can't keep up. A practical RAM range you'll commonly see on modern app servers is about 64–512 GB. This amount supports large in-memory caches, application heaps, and working data sets without resorting to disk frequently. When memory is plenty, the next limiter is how fast the CPU can execute code, manage threads, and orchestrate I/O, not how much memory you have. So the statement that the CPU is almost always the first bottleneck, not memory, is the best fit. Disk I/O or network can become bottlenecks in specific workloads, but they're not typically the first constraint for everyday app-server throughput. And a RAM range like 16–32 GB is often too small for modern workloads that rely on caching and larger heaps, while 1–2 TB is usually unnecessary for many setups and would point toward other limits like storage bandwidth rather than RAM itself.

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://systemsdesignconcepts.examzify.com

We wish you the very best on your exam journey. You've got this!