

14 questions and answers

Fourteen original system design prompts with concise answer outlines and links to full worked walkthroughs. Free to download, with no account or email required.

Use each outline as a starting point. In an interview, clarify requirements and scale first; a sound design depends on the constraints you agree on.

Design a URL Shortener

Answer outline. Generate a unique key, store its mapping to the long URL durably, and keep redirects fast with a cache for popular keys. Record click analytics asynchronously so tracking does not slow the redirect path.

Practice focus. Estimate a read-heavy workload, then decide what to cache and how to count clicks without slowing redirects.

[Read the full worked answer](#)

Design an API Rate Limiter

Answer outline. A token bucket handles bursts while enforcing an average refill rate. A strict global limit needs shared state across servers; local counters are more available but can temporarily exceed the intended limit.

Practice focus. Compare counter and token-bucket designs, then decide how multiple servers share limits.

[Read the full worked answer](#)

Design a News Feed (Twitter Timeline)

Answer outline. Build most feeds by writing new posts into followers' timelines, then fetch posts on read for very large publishers. A hybrid avoids making celebrity posts fan out to millions of inboxes while keeping ordinary reads fast.

Practice focus. Choose between write fan-out and read fan-out when a few accounts have millions of followers.

[Read the full worked answer](#)

Design an Uber-Style Ride-Sharing System

Answer outline. Use a short-lived spatial index to find fresh, available drivers near a pickup, then rank a small candidate set by road ETA. Reserve the driver and ride with a conditional, versioned state transition before sending an offer; leases and idempotent replies make retries safe.

Practice focus. Design Uber-style dispatch: match nearby drivers quickly without stale locations or double assignments, then isolate a stadium hotspot.

[Read the full worked answer](#)

Design a Payment System

Answer outline. Give each payment attempt a durable idempotency key and move it through an explicit state machine. Retries can then return the existing result, while provider events are deduplicated and reconciled instead of assuming exactly-once delivery.

Practice focus. Make retries safe when a payment succeeds but the caller never receives the response.

[Read the full worked answer](#)

Design a Distributed Job Queue

Answer outline. Assume at-least-once delivery: acknowledge work only after its effect is durable, make handlers idempotent, and use expiring leases so another worker can recover a crashed job. Bound retries and route repeatedly failing jobs to a dead-letter queue.

Practice focus. Handle duplicate delivery, worker crashes, retries and messages that can never succeed.

[Read the full worked answer](#)

Design a Video Processing Pipeline

Answer outline. Upload video bytes directly and resumably to object storage, while a durable state machine tracks each video and transcode job. Workers claim jobs with expiring, fenced leases and write attempt-specific outputs, so a crash can trigger a retry without letting stale work overwrite the current result.

Practice focus. Separate upload, durable job state and transcoding so each can recover from failures independently.

[Read the full worked answer](#)

Design a Product Analytics System

Answer outline. Append validated events to a durable stream before acknowledging them, then batch them into a column store for queries by team, event and time. Stable event IDs make retries safe; an identity override table handles anonymous-to-known user merges without rewriting the event history.

Practice focus. Keep ingestion durable and deduplicated while making time-range analytics fast for each customer.

[Read the full worked answer](#)

Design a Notification System

Answer outline. Write product events through an outbox, then create notifications with stable deduplication keys. Delivery workers claim each notification through a state machine, recheck current preferences before sending, and isolate critical traffic from bulk traffic; retries must account for providers that may accept a message before timing out.

Practice focus. Prevent duplicate sends, honor preference changes and protect urgent messages when a provider slows down.

[Read the full worked answer](#)

Design a Distributed Cache (Memcache)

Answer outline. Keep the database authoritative and treat the cache as a disposable, demand-filled copy. Record invalidations with each database commit and deliver them after replication; use leases to stop an old read from restoring stale data, and a spare pool to absorb a cache-node failure.

Practice focus. Control stale reads and database load during writes, hot-key bursts and cache failures.

[Read the full worked answer](#)

Design Discord's Message Storage

Answer outline. Partition messages by channel and time bucket, then order them with time-sortable IDs so recent pages and history reads touch a small range. A routing layer keeps each channel's reads local and coalesces identical requests; tombstones, repair and dual writes make deletes and live database moves explicit.

Practice focus. Make channel history cheap to read while keeping hot channels and database migrations manageable.

[Read the full worked answer](#)

Design a Collaborative Editor (Google Docs)

Answer outline. Give each document one fenced owner to assign durable sequence numbers, while clients apply edits locally and resend unacknowledged operations after reconnecting. Use an operation merge algorithm so concurrent edits converge; keep transient presence separate from the durable edit log.

Practice focus. Preserve one document history through concurrent edits, reconnects, owner changes and fan-out.

[Read the full worked answer](#)

Design a Reactive Database (Live Queries)

Answer outline. Run each query at a versioned snapshot and record the index ranges it reads. When a commit changes one of those ranges, rerun only the affected queries and push results with the commit timestamp; clients apply updates only when their subscriptions agree on that timestamp.

Practice focus. Update subscribed query results without polling or showing data from mixed snapshots.

[Read the full worked answer](#)

Shard a Live Database Without Downtime

Answer outline. Shard by workspace because most reads and joins stay within one workspace, and map a fixed set of logical shards to database hosts. For migration, log writes, backfill versioned rows, catch up, verify with comparisons and dark reads, then cut over reversibly; handle cross-workspace features as a separate design problem.

Practice focus. Scale write volume without rehashing every row, and migrate live data without losing newer writes.

[Read the full worked answer](#)

Continue practising

Try the interactive questions and complete walkthroughs at sysgeeks.com/investigations.