

# Golden Rules to answer in a System Design Interview

# Rule #1

---

If we are dealing with  
a read-heavy system,  
it's good to consider  
using a Cache

# Rule #2

---

If we need low latency  
in system, it's good to  
consider using a  
**Cache & CDN**

# Rule #3

---

If we are dealing with  
a write-heavy system,  
it's good to consider  
using a Message  
Queue for Async  
processing

# Rule #4

---

If we need a system to  
be ACID complaint,  
we should go for  
RDBMS or SQL  
Database

# Rule #5

---

If data is unstructured  
& doesn't require  
ACID properties, we  
should go for NO-SQL  
Database

# Rule #6

---

If the system has complex data in the form of videos, images, files etc, we should go for Blob/Object storage

# Rule #7

---

If the system requires complex pre-computation like a news feed, we should consider using a Message Queue & Cache

# Rule #8

---

If the system requires searching data in high volume, we should consider using a search index, tries or search engine like Elasticsearch

# Rule #9

---

If the system requires  
to Scale SQL  
Database, we should  
consider using  
Database Sharding

# Rule #10

---

If the system requires  
High Availability,  
Performance, and  
Throughput, we  
should consider using  
a Load Balancer

# Rule #11

---

If the system requires faster data delivery globally, reliability, high availability, and performance, we should consider using a CDN

# Rule #12

---

If the system has data with nodes, edges, and relationships like friend lists, and road connections, we should consider using a Graph Database

# Rule #13

---

If the system needs scaling of various components like servers, databases, etc, we should consider using **Horizontal Scaling**

# Rule #14

---

If the system requires high performing database queries, we should consider using Database Indexes

# Rule #15

---

If the system requires  
bulk job processing,  
we should consider  
using  
Batch Processing &  
Message Queues

# Rule #16

---

If the system requires reducing server load and preventing DOS attacks, we should consider using a Rate Limiter

# Rule #17

---

If the system has microservices, we should consider using an API Gateway (Authentication, SSL Termination, Routing etc)

# Rule #18

---

If the system has a  
single point of failure,  
we should implement  
Redundancy in that  
component

# Rule #19

---

If the system needs to be fault-tolerant, and durable, we should implement Data Replication (creating multiple copies of data on different servers)

# Rule #20

---

If the system needs user-to-user communication (bi-directional) in a fast way, we should consider using Websockets

# Rule #21

---

If the system needs the ability to detect failures in a distributed system, we should consider implementing Heartbeat

# Rule #22

---

If the system needs to ensure data integrity, we should consider implementing Checksum Algorithm

# Rule #23

---

If the system needs to transfer data between various servers in a decentralized way, we should go for Gossip Protocol

# Rule #24

---

If the system needs to scale servers with add/removal of nodes efficiently, no hotspots, we should implement Consistent Hashing

# Rule #25

---

If the system needs anything to deal with a location like maps, nearby resources, we should consider using Quadtree, Geohash etc

# Rule #26

---

Avoid using any specific technology names such as - Kafka, S3, or EC2. Try to use more generic names like message queues, object storage etc

# Rule #27

---

If High Availability is required in the system, it's better to mention that system cannot have strong consistency. Eventual Consistency is possible

# Rule #28

---

If asked how domain name query in the browser works and resolves IP addresses.

Try to sketch or mention about DNS (Domain Name System)

# Rule #29

---

If asked how to limit the huge amount of data for a network request like youtube search, trending videos etc. One way is to implement Pagination which limits response data

# Rule #30

---

If asked which policy you would use to evict a Cache. The preferred/asked Cache eviction policy is LRU (Least Recently Used) Cache. Prepare around its Data Structure and Implementation

# Rule #31

---

To handle traffic spikes: Implement  
Autoscaling to  
manage resources  
dynamically

# Rule #32

---

Need analytics and audit trails - Consider using data lakes or append-only databases

# Rule #33

---

Handling Large-Scale  
Simultaneous  
Connections - Use  
Connection Pooling  
and consider using  
Protobuf to minimize  
data payload

# Rule #34

---

To prevent  
overloading systems:  
**Use Token Bucket and  
Leaky Bucket  
algorithms**

# Rule #35

---

**Minimize deployment downtime -  
Implementing Blue-Green deployment to  
minimize downtime  
and reduce risks**

If you like the content  
Don't forget to  
follow us



Arslan Ahmed



Dinesh Varyani