System Design

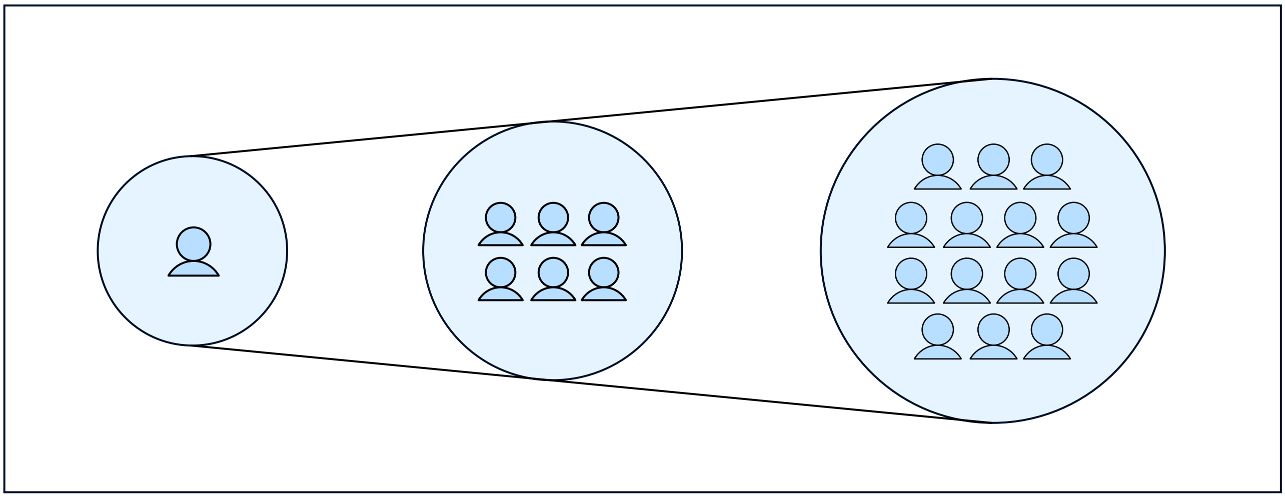
System design is a crucial component of software engineering. It's a process of defining the architecture, components, interfaces, and data for a system to satisfy specified requirements. Good system design can improve efficiency, scalability, and performance.

Core Concepts in System Design

Scalability

Navigating increased load with grace. Think Amazon on Black Friday.

Pro Tip: *Remember, the key techniques here are Load balancing, Database partitioning, and Caching.*



✓ Availability

Your system should always be there for you. Imagine Google Search being down, unthinkable, right?

Pro Tip: *The tricks of the trade are Replication, Failover, and Recovery.*

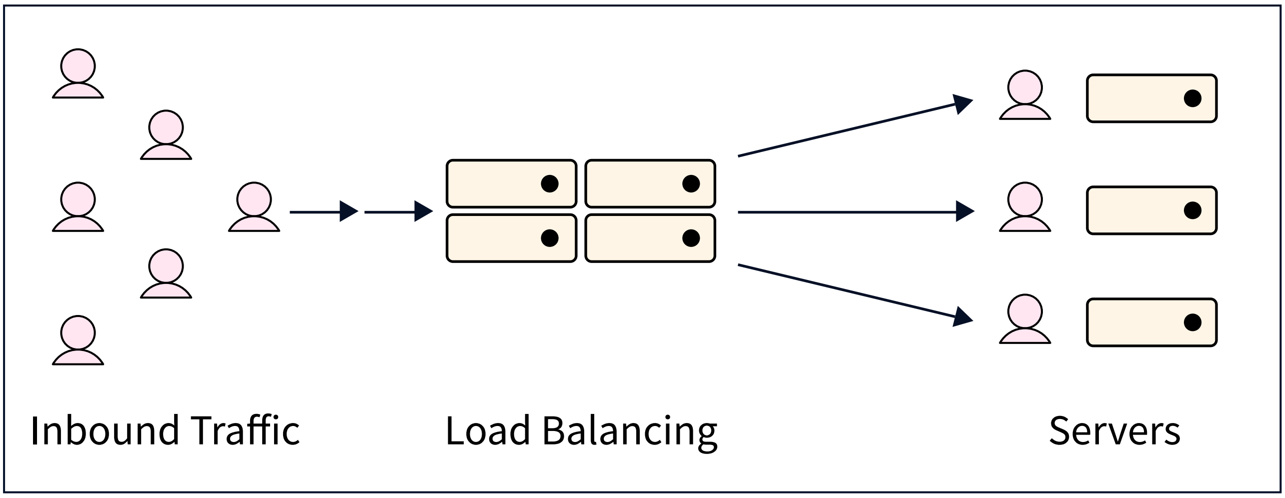
500. That's an error.

There was an error. Please try again later. That's all we know.

Load Balancing

Spreading the load for a smooth run. It's like assigning the right amount of homework to students.

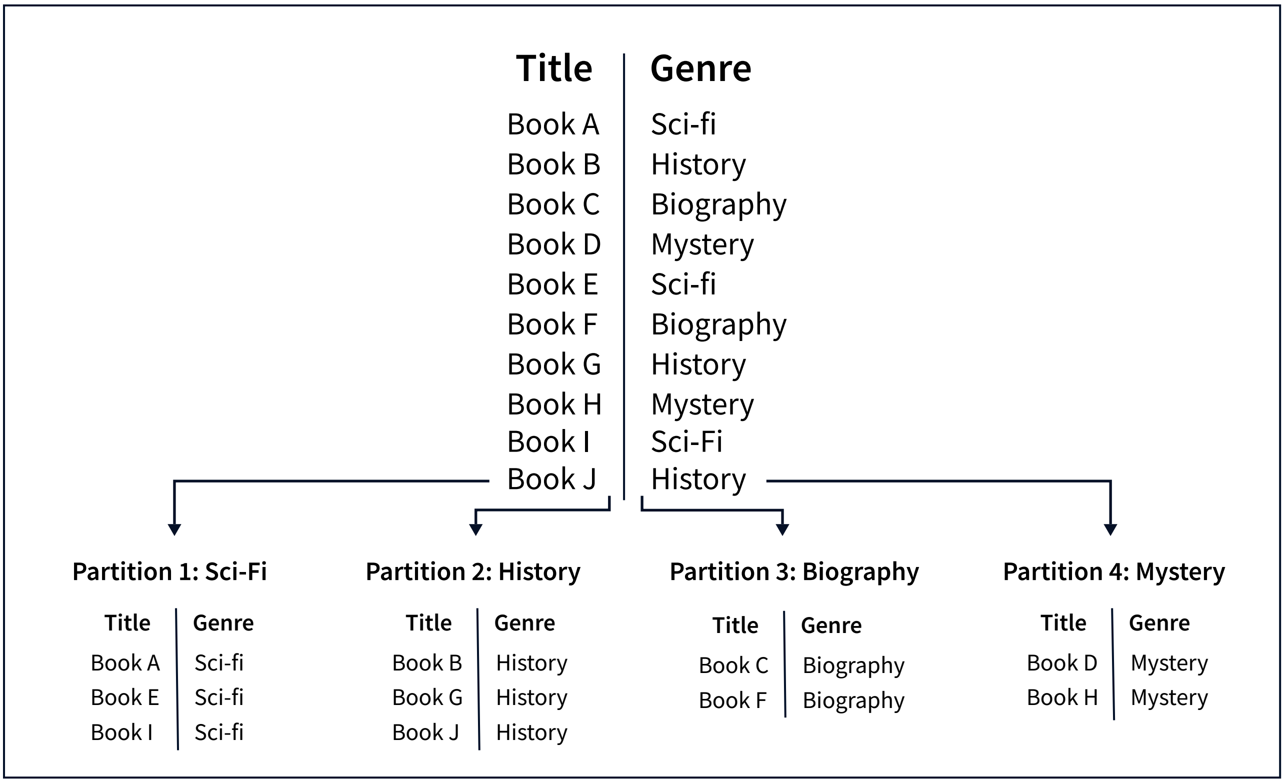
Pro Tip: *Use algorithms such as Round-Robin, Least Connections, or IP Hashing for efficient load balancing.*



Data Partitioning

Breaking the database for a better outcome. Imagine trying to find a book in an unorganized library.

Pro Tip: *Range and Hash partitioning are common techniques used in data partitioning.*



Title	Genre
Book A	Sci-fi
Book B	History
Book C	Biography
Book D	Mystery
Book E	Sci-fi
Book F	Biography
Book G	History
Book H	Mystery
Book I	Sci-Fi
Book J	History

Partition 1: Sci-Fi

Title	Genre
Book A	Sci-fi
Book E	Sci-fi
Book I	Sci-fi

Partition 2: History

Title	Genre
Book B	History
Book G	History
Book J	History

Partition 3: Biography

Title	Genre
Book C	Biography
Book F	Biography

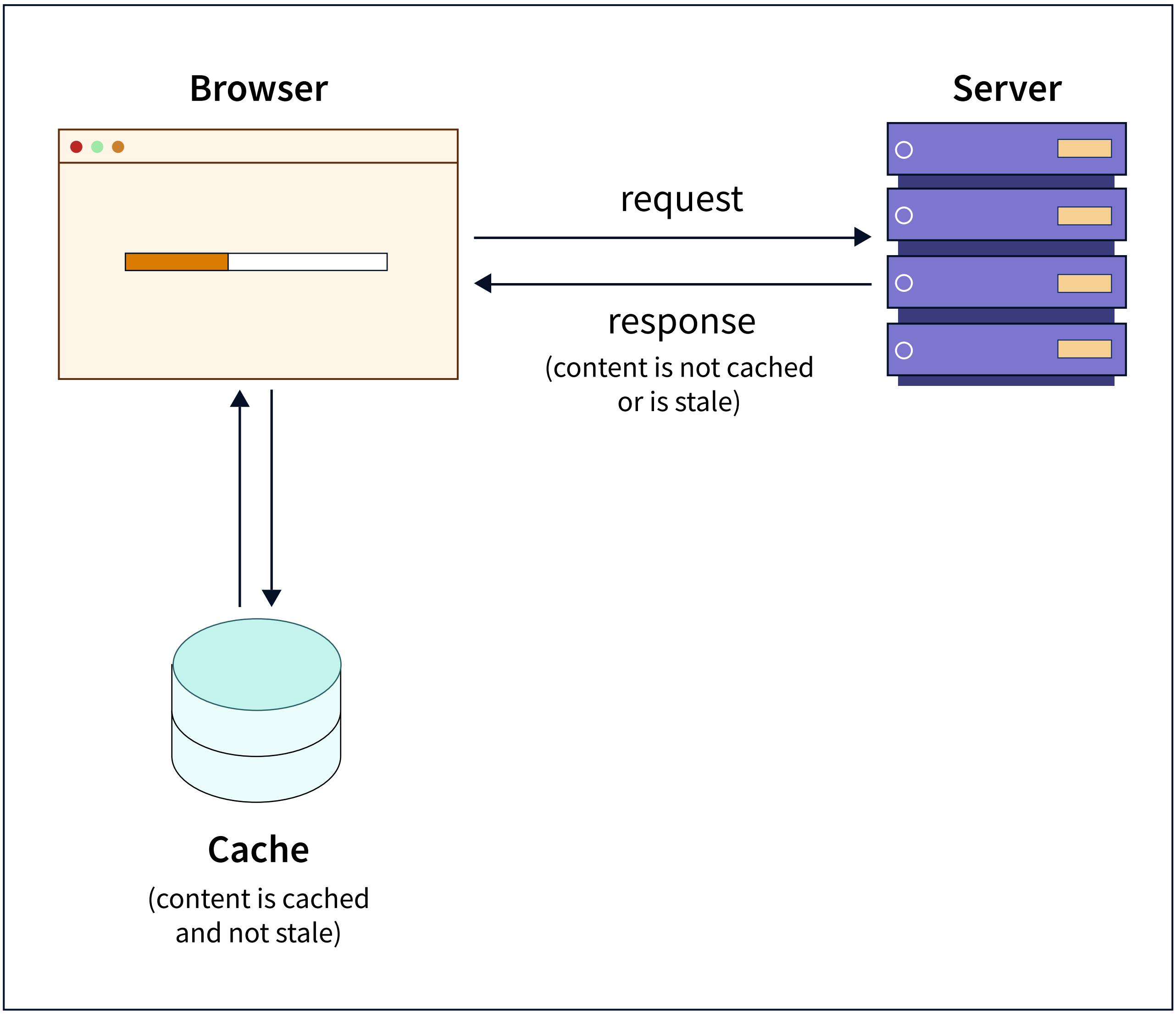
Partition 4: Mystery

Title	Genre
Book D	Mystery
Book H	Mystery

Caching

Because remembering is quicker than asking. Ever noticed how quick Google Autocomplete is?

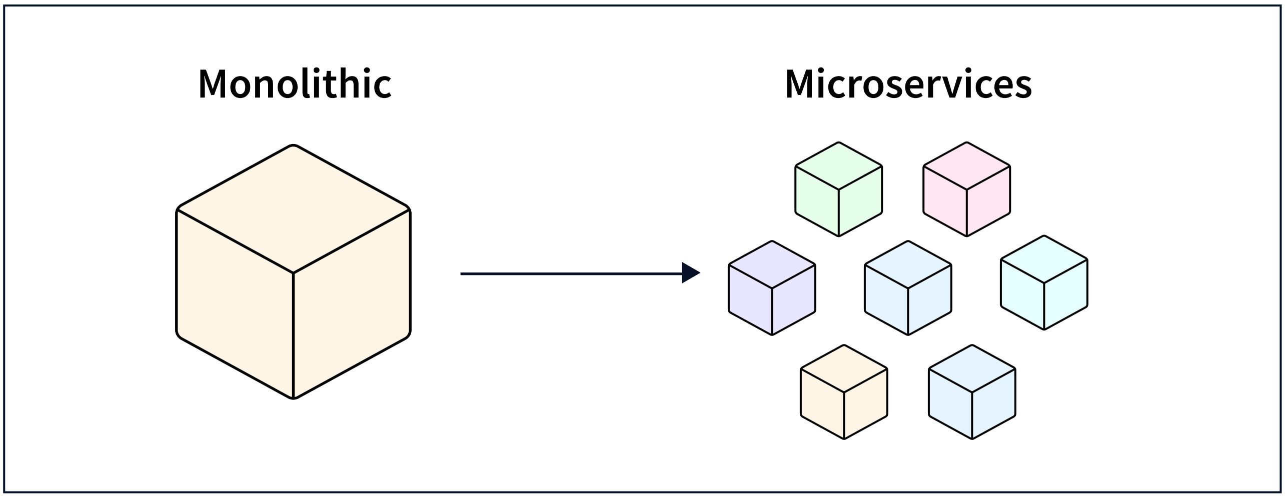
Pro Tip: *Decide caching strategies based on the use case - LRU cache can be a good start.*



Microservices

The charm of working in small teams, but for your applications.

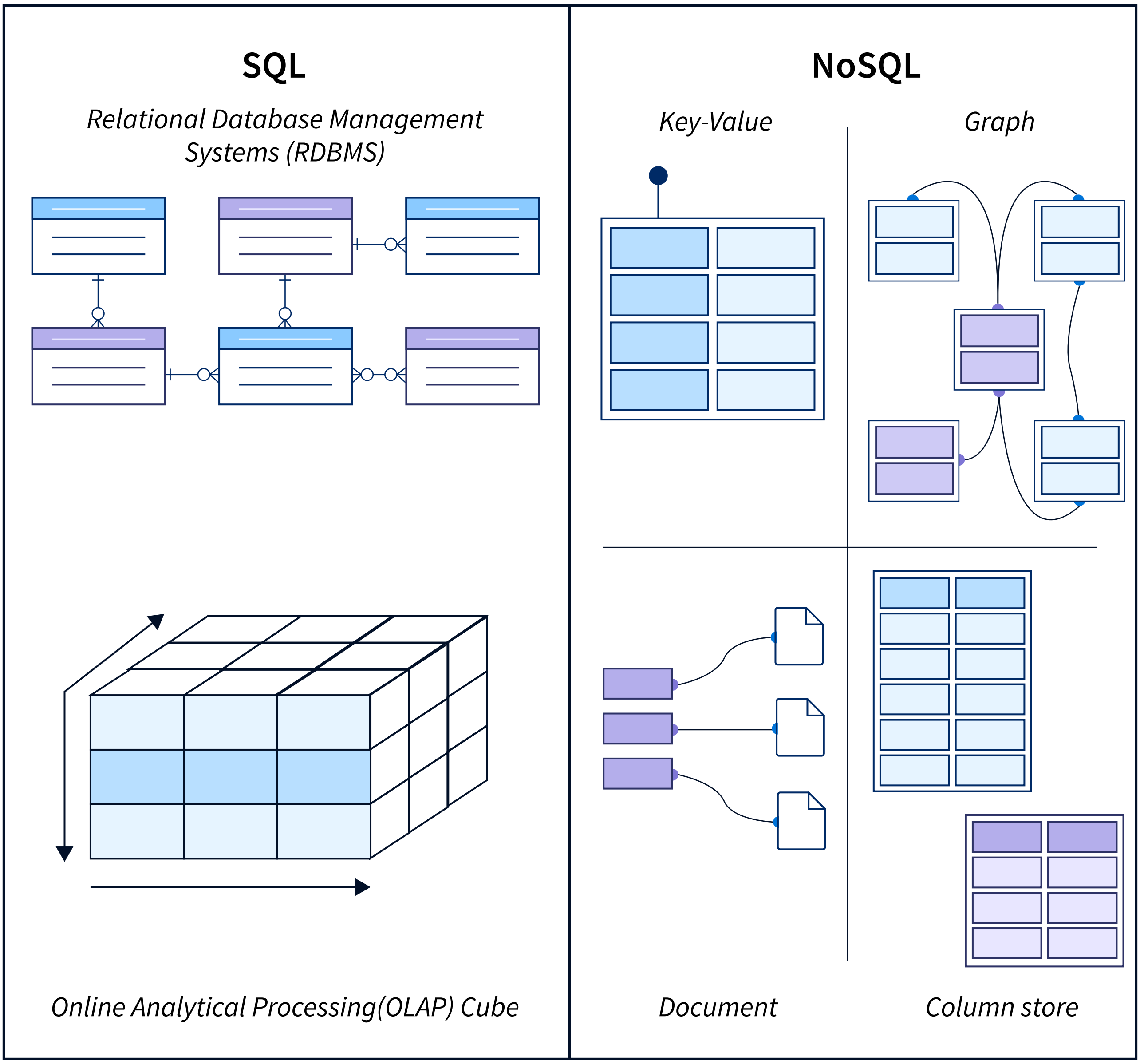
Pro Tip: _Loose coupling and high cohesion make for an effective microservices architecture._



Databases and Storage

Like picking the right vehicle for your road trip. SQL or NoSQL? Disk or In-memory?

Pro Tip: *Evaluate based on factors like consistency requirements, data model, read/write load, and scalability.*



SQL
Relational Database Management Systems (RDBMS)
Online Analytical Processing(OLAP) Cube
NoSQL
Key-Value
Graph
Document
Column store

Didn't understand a term used above? Refer the glossary section at the end of the cheat sheet for a quick overview!

Effective System Design Strategies

- **Understanding User Needs:**

  Design keeping in mind who will use the system and for what.

- **Designing for Scalability:**

  Like preparing for a guest list that can go from 5 to 500, unpredictability is the name of the game.

- **Emphasizing Flexibility and Extensibility:**

  Today's 'extra' could be tomorrow's 'essential'. Your design should accommodate changing requirements.

- **Prioritizing Security and Privacy:**

  Because the trust of the user is paramount.

- **Ensuring Performance:**

  Because no one likes waiting.

Real-World Case Studies in System Design

Twitter	Netflix
Handling a massive number of posts and followers and displaying feeds in real-time. The underlying concept? Sharding the user and tweet databases.	Managing video storage and streaming and handling recommendations. It's all about data-driven personalization and high-speed content delivery.

Examples of System Design Interview Questions and Tips for Answering Them

- **Design a URL shortening service:**

  Consider how to generate unique URLs, how to handle redirects, and how to manage the storage of the URLs.

- **Design a social media platform like Twitter:**

  Consider how to manage user profiles, how to handle massive numbers of posts and followers, and how to display feeds in real-time.

- **How to design Netflix:**

  Think about how to manage video storage and streaming, how to handle recommendations, and how to manage user subscriptions and profiles.

System Design Glossary

API (Application Programming Interface):

A set of protocols and tools for building software applications.

It's like a menu in a restaurant. It tells you what services a program can perform.

Availability:

The time a system remains operational to perform its required function in a specific period.

If a website is available, it means you can reach it on your browser.

Cache:

A hardware or software component that stores data to serve future requests faster.

It's like a small storage box you keep your most used tools in, so you can access them faster than going to your large tool shed (database).

CAP Theorem:

A concept that a distributed computing system can only achieve two of the three following: Consistency, Availability, and Partition tolerance.

It's like juggling, you can only keep two balls in the air and have to drop one.

Consistency:

Ensures that data is the same across all nodes in a distributed system.

If a book's price is changed, all online bookstores show the new price instantly.

Data Partitioning:

The process of breaking down a database into multiple parts to improve manageability, performance, and availability.

It's like breaking down a big book into smaller volumes for easy management and quick access.

Failover:

The process of switching automatically to a redundant or standby system upon the failure or abnormal termination of the previously active system.

If a bulb fails, an automatic switch turns on an extra bulb.

Horizontal Scaling:

Adding more machines to the existing pool of resources.

It's like adding more cash registers to a busy grocery store.

Idempotency:

An operation that produces the same results no matter how many times it's executed.

No matter how many times you turn a switch off, the light remains off.

Load Balancing:

The process of distributing network traffic across multiple servers to ensure no single server bears too much demand.

Like a receptionist who distributes incoming visitors to different service counters to avoid overloading one counter.

Microservices:

An architectural style that structures an application as a collection of loosely coupled services.

If an application were a restaurant, each service (booking, cooking, serving) would be separate, so if one fails, the others still work.

Middleware:

Software that lies between an operating system and the applications running on it, enabling communication and data management.

The delivery guy (middleware) who takes your order (request) from the frontend (user interface) to the kitchen (backend), and brings the food (response) back.

NoSQL Database:

A non-relational database that allows for high-performance, agile processing of information at massive scale.

It's like a toolbox where you can toss in tools (data) without worrying about their size or shape.

Rate Limiting:

A technique for limiting network traffic. It sets a limit on how many requests a client can make to a server in a given amount of time.

It's like a librarian who says you can only borrow a certain number of books at a time.

Replication:

The process of sharing information to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

Like having copies of a document in multiple locations so you can access it even if you lose one copy.

REST (Representational State Transfer):

A software architectural style that defines a set of constraints to be used for creating Web services.

It's like traffic rules for safe and efficient transportation.

Scalability:

The ability of a system to handle an increasing amount of work by adding resources to the system.

It's like a train that can add more coaches (resources) to carry more passengers (load) when required.

Sharding:

A method for distributing data across multiple machines. It is used in databases to manage large amounts of data.

It's like organizing different categories of your belongings in different sections of a large almirah (wardrobe). Imagine having a section for shirts, another for trousers, another for shoes, and so on.

SQL Database:

A relational database that uses SQL (Structured Query Language) to query a database.

It's like a library with books (data) arranged in a specific order.

Throughput:

The number of processes that pass through a system from beginning to end in a given period.

Similar to the number of cars passing through a toll booth in an hour.

Vertical Scaling:

Adding more resources (like CPU, RAM) to your existing machine.

Replacing a regular engine with a turbocharged one in the same car.

WebSockets:

A communication protocol that provides full-duplex communication channels over a single TCP connection.

A two-way road where both cars (client and server) can move at the same time, sending and receiving data.

SCALER TOPICS

Unlock your potential in software development with **FREE COURSES** from **SCALER TOPICS!**

Register now and take the first step towards your future Success!

C++ for Beginners

5.9k enrolled ₹ Free

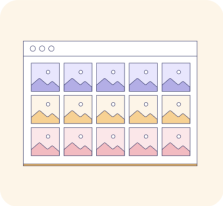
Java for Beginners

6.8k enrolled ₹ Free

That's not it. Explore 20+ Courses by clicking below

Explore Other Courses

Practice **CHALLENGES**
and become 1% better everyday

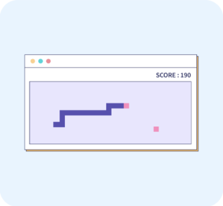


CIFAR-10 Image Classification Using PyTorch

Article

No. Of Questions : 3

Go to Challenge >



How to Build a Snake Game in JavaScript?

Article

No. Of Questions : 3

Go to Challenge >

Explore Other Challenges