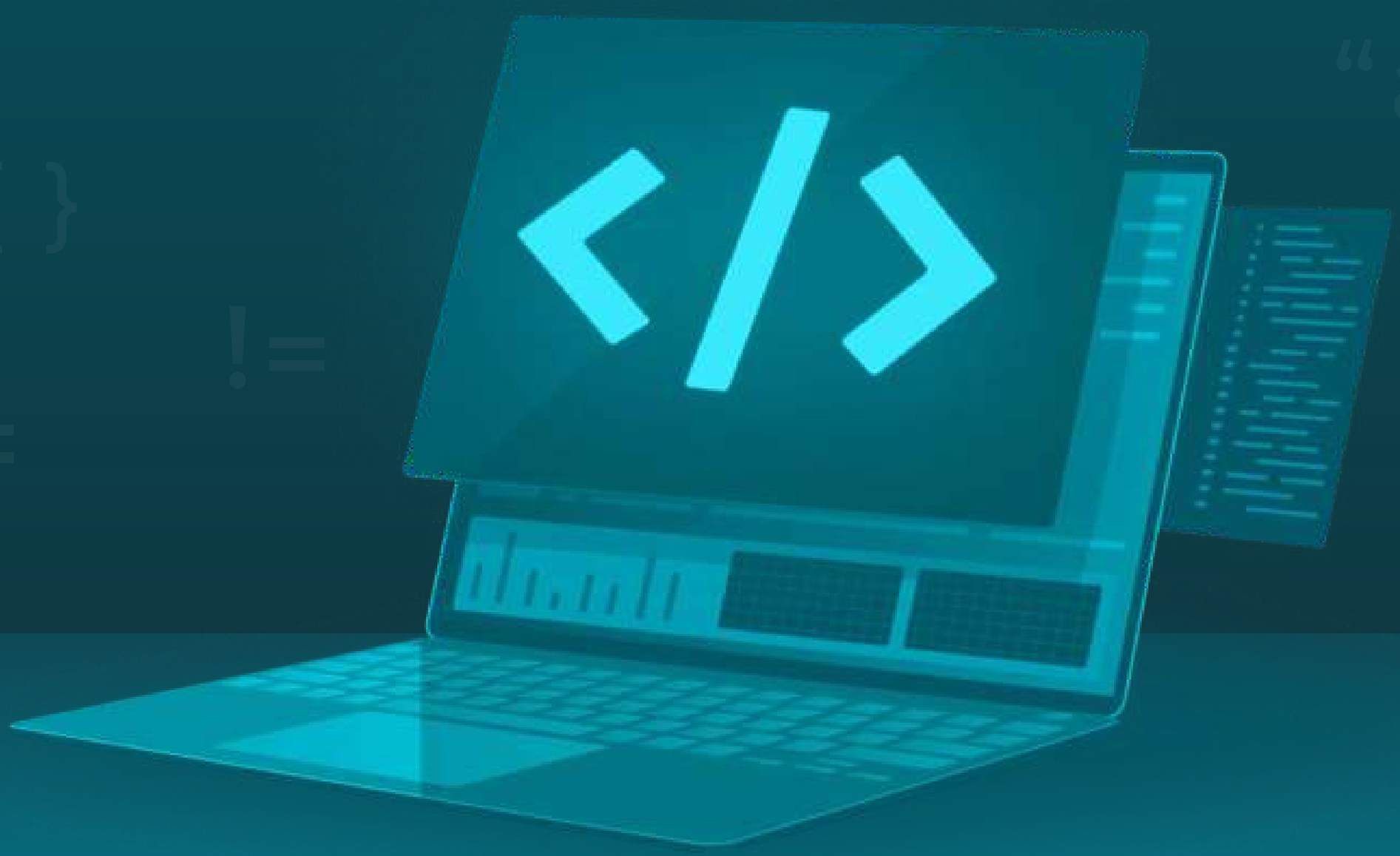
BOSSCODER ACADEMY

— IMPORTANT —

BACKEND

Concepts For Interviews



Special focus on SQL & REST API

Disclaimer

Everyone learns uniquely.

Learn Backend in a structured manner and master it by practically applying your skills.

This Doc will help you with the same.

Q.1

EASY

What are the different languages present in DBMS?

The four types of DBMS languages are as follows:

Data Manipulation Language (DML): It is used to manipulate the data and consists of the command for the same. E.g.: SELECT, INSERT, DELETE, UPDATE, etc.

Data Definition Language (DDL): It is used to define and update the data. E.g.: TRUNCATE, ALTER, DROP, CREATE, RENAME, etc.

Data Control Language (DCL): It is used to control the access to the data. E.g.: GRANT, REVOKE, etc.

Transaction Control Language (TCL): It is used to handle the data transactions. E.g.: COMMIT, ROLLBACK, etc.

Q.2

EASY

What are ACID properties?

ACID properties are a set of properties that ensure reliable and secure transactions among databases. To maintain data consistency, ACID properties are followed. ACID stands for Atomicity, Consistency, Isolation, Durability.

Atomicity: Either the entire transaction takes place at once or not at all.

Consistency: The database must be consistent before and after a transaction

Isolation: No other transaction can alter the data during a transaction is in progress

Durability: The transactions made should be durable and must persist

Q.3

MEDIUM

What is normalization? Explain the different types of normal forms.

Normalization is the technique that reduces data redundancy and eliminates insertion, updation and deletion anomalies. Normalization is the process of dividing the larger table into smaller tables and linking them through relationships. It is the process of organizing data in a database.

- **Insertion Anomaly:** Insertion Anomaly is when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** Delete anomaly is where the deletion of data results in the unintended loss of some other important data.
- **Updation Anomaly:** Updation anomaly is when an update of a single data value requires multiple rows of data to be updated.

Different types of normal forms:

- **1NF:** It is known as the first normal form. A relation is said to be in 1NF if it contains an atomic value.
- **2NF:** It is known as the second normal form. A relation is said to be in 2NF if it is in 1NF and each non-prime attribute is fully functionally dependent on the primary key.
- **3NF:** It is known as the third normal form. A relation is said to be in 3NF if it is in 2NF and there is no transitive dependency.

- **BCNF:** It is known as Boyce Codd Normal Form which is a strict version of 3NF. A relation is said to be in BCNF if it is in 3NF and for every functional dependency X->Y, X is a super key of the table. It is also called the 3.5 Normal Form.
- **4NF:** It is known as the fourth normal form. A relation is said to be in 4NF if it is in BCNF and there is no multivalued dependency in the table.
- **5NF:** It is known as the fifth normal form. A relation is said to be in 5NF if it is in 4NF and it cannot be further decomposed into smaller tables.

Q.4

EASY

What is an ER diagram?

ER diagrams stand for **Entity Relationship Diagram**. It is a diagram that displays the different entities and the relationship among them stored inside the database.

ER diagram provides a logical structure of the database. Creating an ER diagram for the database is a standardized procedure and is done before implementing the database.

ER diagram is based on three concepts:

Entities: It can be defined as an object having some properties. They are represented using rectangles. Eg: Car

Attributes: The properties of an entity are called attributes. They are represented using ellipses. Eg: Car name, car mileage, car type, etc

Relationships: Relationships are how the entities are related to each other. They are represented using lines.

Q.5

MEDIUM

Give the resulting tables arising from applying Joins on the following tables in SQL

Employees Table:

id	name	department_id
1	Alice	101
2	Bob	102
3	Charlie	101
4	David	103

Departments Table:

id	department_id
101	HR
102	IT
103	Marketing
104	Sales

Inner Join:

- Returns only the rows with matching values in both tables.
- Filters out rows with no match.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id =
departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing

Left Join (Left Outer Join):

- Returns all rows from the left table and the matched rows from the right table.
- If there is no match in the right table, NULL values are returned.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing

Right Join (Right Outer Join):

- Returns all rows from the right table and the matched rows from the left table.
- If there is no match in the left table, NULL values are returned.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments ON employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing
NULL	Sales

Full Outer Join:

- Returns all rows when there is a match in either the left or right table.
- Includes rows with no match in either table with NULL values.

SQL Query:

```SQL
SELECT employees.name, departments.department_name
FROM employees
FULL OUTER JOIN departments ON
employees.department_id = departments.id;
```

Output:

name	department_name
Alice	HR
Bob	IT
Charlie	HR
David	Marketing
NULL	Sales

Self Join:

- Combines rows from a single table, treating it as two separate tables.
- Often used for hierarchical data.

SQL Query:

```SQL
SELECT e1.name, e2.name AS manager
FROM employees e1
LEFT JOIN employees e2 ON e1.manager_id = e2.id;
```

Output:

name	manager
Alice	NULL
Bob	NULL
Charlie	Alice
David	NULL

Q.6

EASY

What is statelessness in REST?

In **REST (Representational State Transfer)**, statelessness is a fundamental architectural constraint. It means that each request from a client to a server must contain all the information needed to understand and process the request. The server should not rely on any information from previous requests or sessions stored on the server.

This ensures that each request is independent and can be processed in isolation, making the system more scalable, reliable, and easier to maintain. Statelessness simplifies the communication between clients and servers, as there is no need for the server to store or manage the client's state between requests.

Each request is self-contained, enhancing the overall flexibility and scalability of the RESTful system.

Q.7

HARD

What are Idempotent methods in REST?

Idempotent implies that the outcome of a single request remains the same, even if the request is called multiple times.

In **REST API** design, it is crucial to create idempotent APIs to handle potential duplicate requests from consumers and ensure fault tolerance.

REST inherently provides idempotent methods, which guarantee consistent responses regardless of the number of times a request is made.

GET, **OPTIONS**, **TRACE**, and **HEAD** are idempotent as they are designed for resource retrieval without altering server resource states.

PUT methods, used for resource updates, are idempotent because subsequent requests simply overwrite the same resource without changing its state.

DELETE methods are considered idempotent since the first request successfully deletes the resource (Status Code 200).

Subsequent **DELETE** requests return a Status Code 204, indicating no change in server resources.

DELETE may not be idempotent if it leads to multiple deletions of the same resource with each request (e.g., **DELETE** /user/last).

Q.8

MEDIUM

What is CAP Theorem?

The **CAP theorem** (Brewer's theorem) states that a distributed system or database can provide only two out of the following three properties:

Consistency: Similar to ACID Properties, Consistency means that the state of the system before and after transactions should remain consistent.

Availability: This states that resources should always be available, there should be a non-error response.

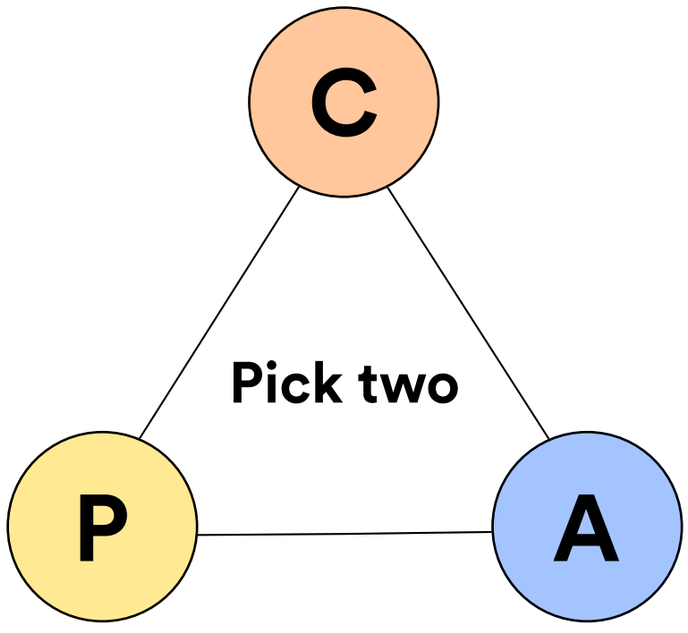
Partition tolerance: Even when the network communication fails between the nodes in a cluster, the system should work well.

By the CAP theorem, all of these three properties cannot be achieved at the same time.

Consistency

CP Category

There is a risk of some data becoming unavailable
Ex: MongoDB Hbase
Memcache Big table Redis



CA Category

Network Problem might stop the system
Ex:RDBMS
(Oracle SQL Server MySQL)

AP Category

Clients may read inconsistent data
Ex: Cassandra RIAK CouchDB

Q.9

MEDIUM

What is CAP Theorem?

SQL Injection:

SQL injection is a cyber attack where an attacker injects malicious SQL code into a website's input fields, exploiting vulnerabilities in the code. The aim is to manipulate the executed SQL query, gaining unauthorized access to, modifying, or deleting data, and potentially executing administrative operations on the database.

Example:

In a login form with the SQL query:

```SQL
SELECT * FROM users WHERE username = 'input_username'
AND password = 'input_password';
```

An attacker might input:

```SQL
input_username = 'admin' OR 1=1 --
```

Resulting in:

```SQL
SELECT * FROM users WHERE username = 'admin' OR 1=1
--' AND password = 'input_password';
```

The double hyphen (--) comments out the rest of the query, allowing unauthorized access.

Prevention

1. Input Validation
2. Cautious Error Messages
3. Logging and Monitoring
4. Web Application Firewalls (WAFs)
5. Security Audits

Q.10

MEDIUM

What is the difference between clustered and non clustered indexes?

Feature	Clustered Index	Non-Clustered Index
Speed	Faster	Slower
Memory Usage	Requires less memory	Requires more memory
Data Storage	Main data is the clustered index itself	Index is a copy of data
Number of Indexes Allowed	Only one per table	Multiple per table
Disk Storage	Stores data on disk	Does not inherently store data on disk
Storage Structure	Stores pointers to blocks, not data	Stores both values and pointers to data
Leaf Nodes	Actual data in leaf nodes	Leaf nodes may contain included columns, not data
Order Definition	Clustered key defines order in the table	Index key defines order in the index
Physical Order of Rows	Matches order of the clustered index	Does not necessarily match physical order on disk
Size	Large	Comparatively smaller (for non-clustered index)
Default for Primary Keys	Primary keys are clustered indexes by default	Composite keys with unique constraints act as non-clustered indexes

Q.11

EASY

What is a web server?

A web server is a **software application or hardware device** that stores, processes, and delivers web pages to users' browsers. It serves as the foundation for hosting websites and handling client requests by responding with the appropriate web content.

Examples of web servers include:

1. Apache HTTP Server:
2. Nginx
3. Microsoft Internet Information Services (IIS):
4. LiteSpeed Web Server:
5. Caddy

Q.12

MEDIUM

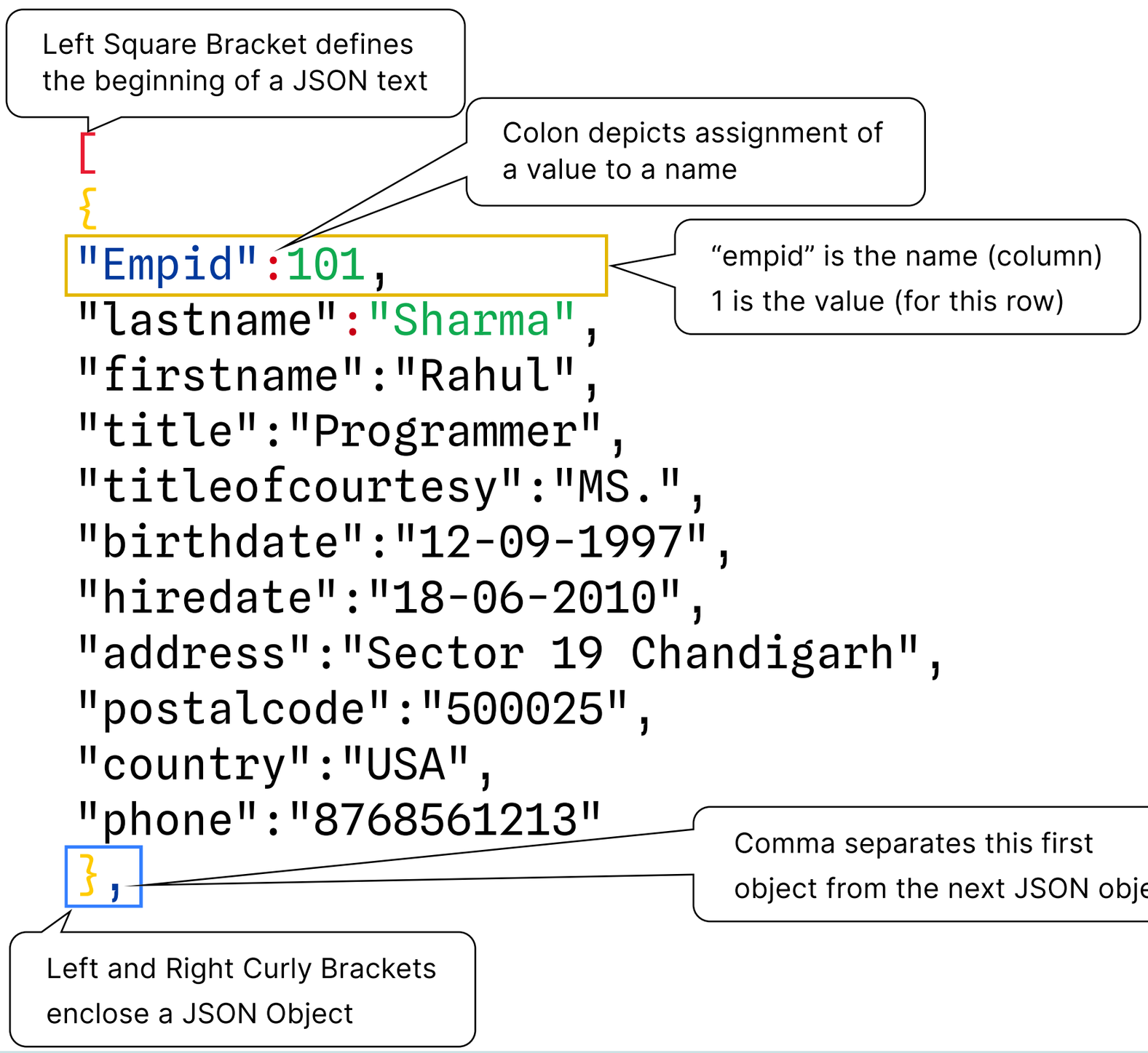
What is SQL injection? How can we prevent it?

NoSQL is a **Non-relational or Distributed Database**. Non-relational databases store their data in a non-tabular form. Instead, non-relational databases have different storage models based on specific requirements of the type of data being stored.

For example, data may be stored as

→ Document databases

Data is stored as documents in a format such as JSON or XML
Each document assigned its own unique key

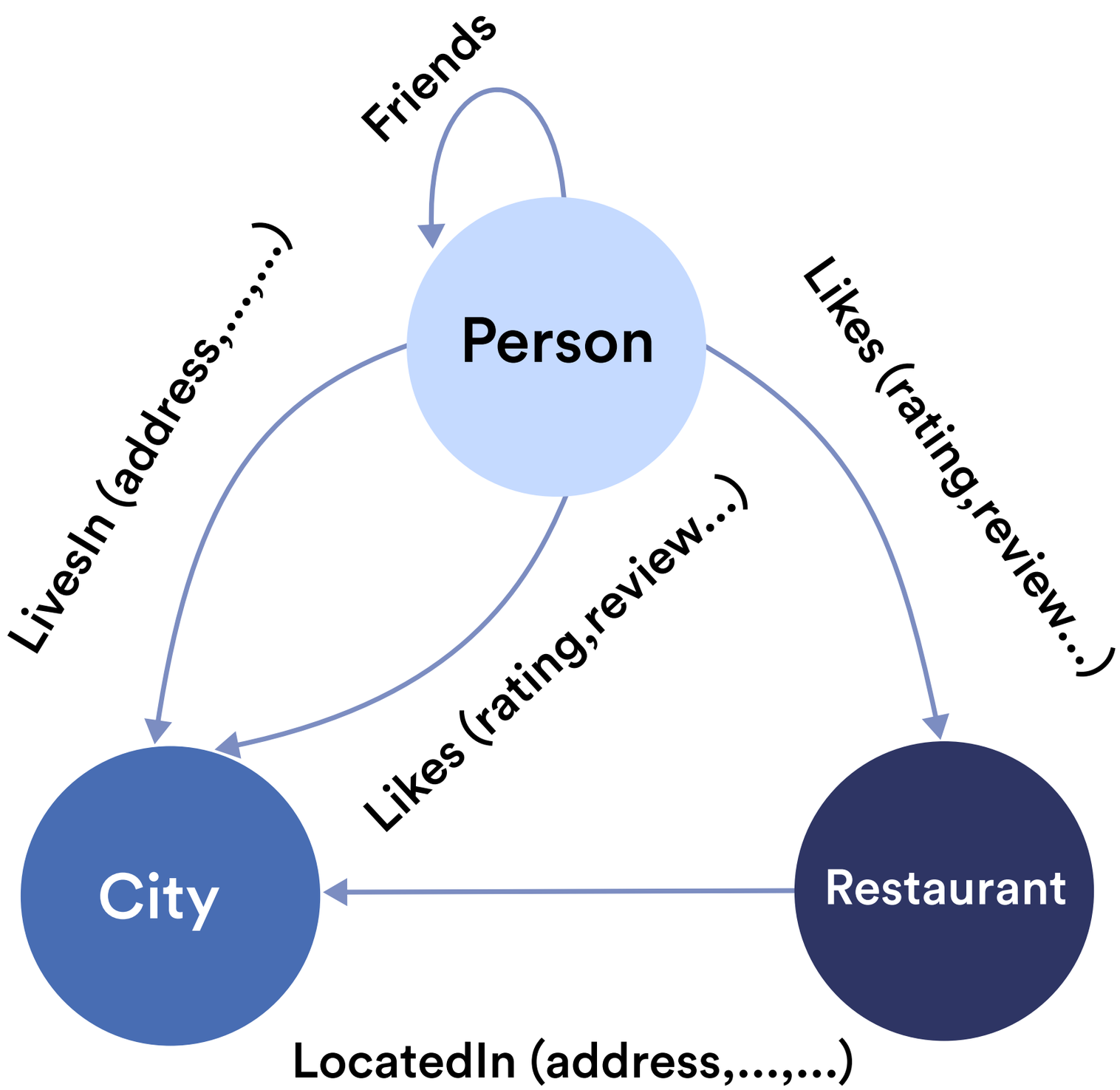


Example products include MongoDB, CouchDB, and BaseX.

→ Example products include MongoDB, CouchDB, and BaseX.

Each element is stored as a node. It stores the data itself.
(Example: A person in a social media graph).

The Edge explains the relationship between two nodes. Edges can also have their own pieces of information, like relationship between two nodes.



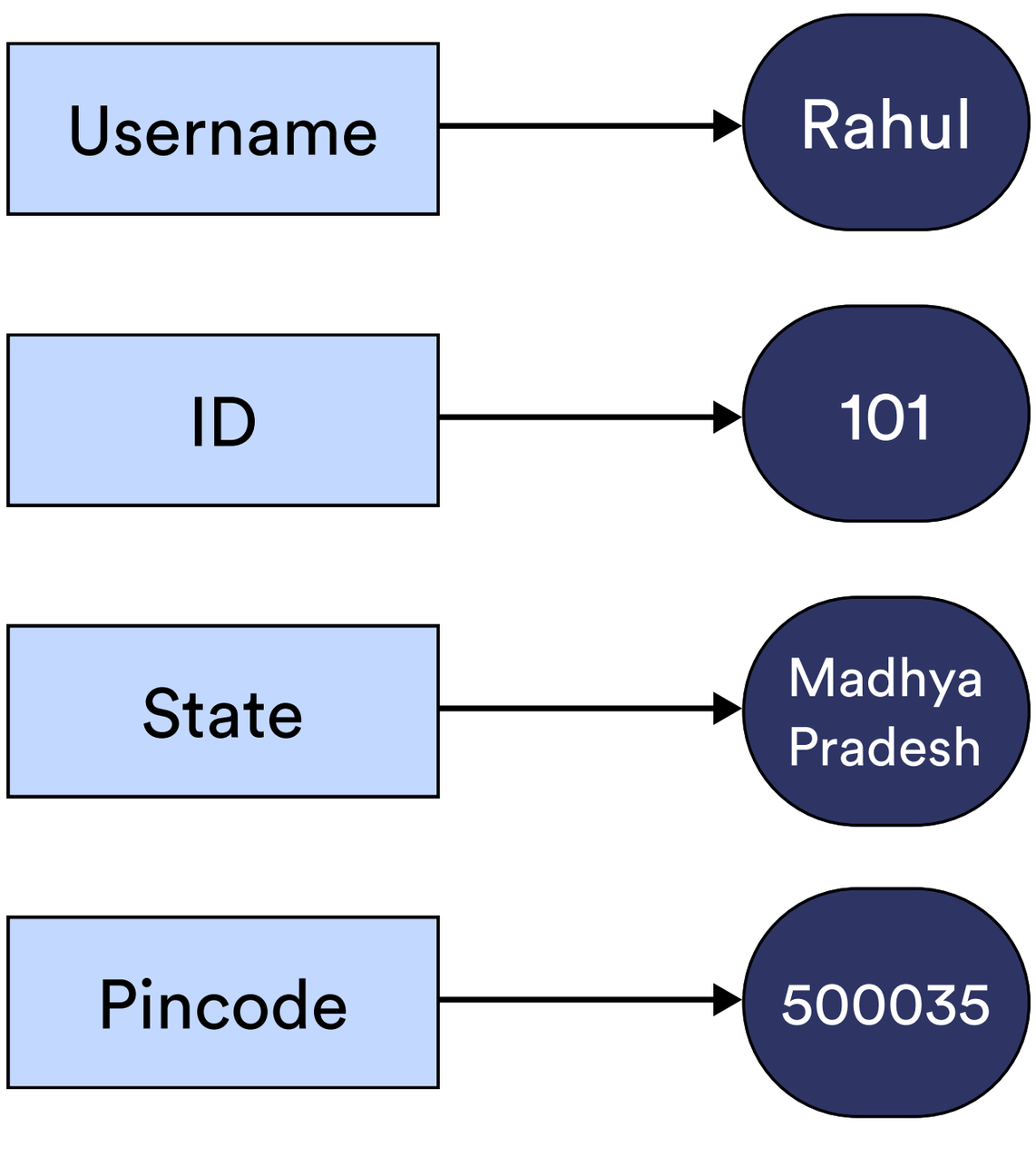
Examples include Neo4J, and InfiniteGraph.

→ Key Value Data-Model

In this model every data element in the database is stored as a key value pair.

The pair consists of an attribute or “key” and its corresponding value.

We can sort of consider this model to be similar to a relational database with only two columns, the key and and the value.



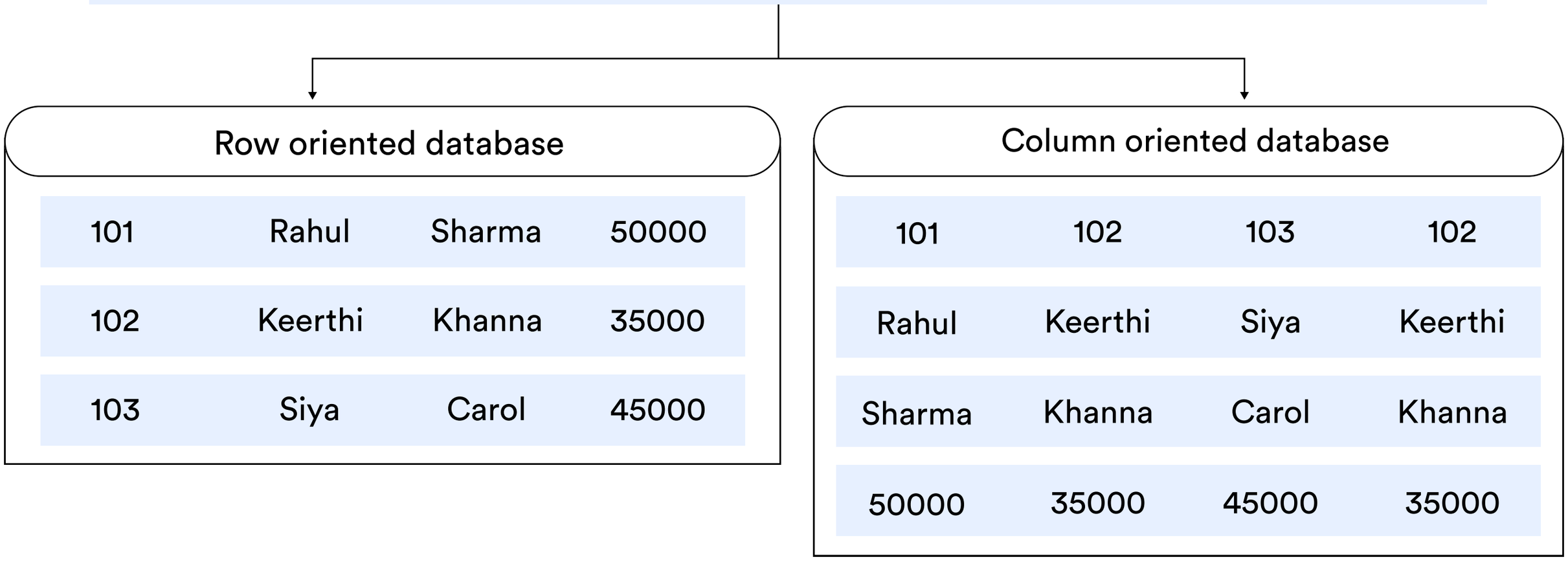
Example products include Redis, Berkeley DB, and Amazon DynamoDB.

→ Column Oriented Databases

A column oriented database is organised as a set of columns.

Column-oriented storage is used to improve analytic query performance. It reduces the overall disk I/O requirements and reduces the amount of data you need to load from disk.

Id Number	First Name	Last Name	Bonus
101	Rahul	Sharma	50000
102	Keerthi	Khanna	35000
103	Siya	Carol	45000



Example products include Redis, Berkeley DB, and Amazon DynamoDB.

Q.13

EASY

How do you create a simple server in Node.js that returns Hello World?

```
// Import the 'http' module
const http = require("http");
// Create an HTTP server
http.createServer((req, res) => {
  // Set the response headers (HTTP status code 200 OK and content type as plain text)
  res.writeHead(200, {'Content-Type': 'text/plain'});


  // Send the response content ('Hello World' followed by a newline)
  res.end('Hello World\n');
}).listen(3000); // Listen on port 3000
```

Q.14

MEDIUM

What is MVC Architecture?

The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller. It comprises three main components: Controller, Model, and View.

Controller: The controller focuses on processing business logic and handling incoming requests. The controller instructs the model, manipulates data, and collaborates with the view to produce the final output.

View: Responsible for the application's UI logic, the view generates the user interface based on data collected through the controller. It interacts solely with the controller, ensuring separation of concerns.

Model: The model handles data-related logic and manages interactions with the database, responding to controller requests and providing necessary data.

MVC Design Principles:

1. **Divide and conquer:** The three components can be independently designed.
2. **Increase cohesion:** The components exhibit strong layer cohesion.
3. **Reduce coupling:** Communication channels between components are minimal and clear.
4. **Increase reuse:** Views and controllers make use of reusable components for UI controls, promoting reusability.
5. **Design for flexibility:** Changing the UI is easily achievable by modifying the view, controller, or both.

Q.15

MEDIUM

What is API Rate Limiting? Give a few rate limiting algorithms.

API Rate Limiting:

API Rate Limiting is a technique used to control the number of requests a client (or user) can make to an API within a specified time frame. It helps prevent abuse, protect server resources, and ensure fair usage of the API. Without rate limiting, a malicious or overly aggressive client could overwhelm the API server, leading to degraded performance or even denial of service.

Rate Limiting Algorithms:

1. Token Bucket Algorithm:

- Clients are assigned tokens at a fixed rate.
- Each request consumes a token.
- Requests are allowed only if the client has tokens available.

2. Leaky Bucket Algorithm:

- Requests are added to the "bucket" at a fixed rate.
- The bucket has a maximum capacity.
- Requests are processed if there is capacity; otherwise, they are delayed or discarded.

3. Fixed Window Counter:

- Counts the number of requests within fixed time windows (e.g., 1 second, 1 minute).
- Resets the counter at the beginning of each window.

4. Sliding Window Log:

- Keeps a log of timestamps for each request.
- Counts requests within a sliding time window.

5. Adaptive Rate Limiting:

- Adjusts the rate limit dynamically based on the recent traffic patterns.
- Reacts to sudden spikes or drops in traffic.

Q.16

MEDIUM

How can you select which webservice to use between REST and SOAP?

When deciding between SOAP and REST for web services, consider the following factors:

1. Nature of Data/Logic Exposure:

- **SOAP:** Used for exposing business logic.
- **REST:** Used for exposing data.

2. Formal Contract Requirement:

- **SOAP:** Provides strict contracts through WSDL.
- **REST:** No strict contract requirement.

3. Data Format Support:

- SOAP: Limited support.
- REST: Supports multiple data formats.

4. AJAX Call Support:

- SOAP: No direct support.
- REST: Supports XMLHttpRequest for AJAX calls.

5. Synchronous/Asynchronous Requests:

- SOAP: Supports both sync and async.
- REST: Supports only synchronous calls.

6. Statelessness Requirement:

- SOAP: No.
- REST: Yes.

7. Security Level:

- SOAP: Preferred for high-security needs.
- REST: Security depends on underlying implementation.

8. Transaction Support:

- SOAP: Provides advanced support for transactions.
- REST: Limited transaction support.

9. Bandwidth/Resource Usage:

- SOAP: High bandwidth due to XML data overhead.
- REST: Uses less bandwidth.

10. Development and Maintenance Ease:

- SOAP: More complex.
- REST: Known for simplicity, easy development, testing, and maintenance.

Q.17

EASY

What is DRY principle in software development?

In software development, the DRY principle stands for "Don't Repeat Yourself." It's a best practice that emphasizes avoiding duplicate code. Imagine writing the same instructions for doing something twice in different parts of your program. If you need to make a change later, you'd have to update both places, increasing the risk of inconsistencies and bugs.

Here's an example:

Without DRY:

```
def validate_email(email):
  if not email or "@" not in email:
    return False
  return True
def update_profile(user, email):
  if not validate_email(email):
    raise ValueError("Invalid email address")
  user.email = email
def send_confirmation_email(email):
if not validate_email(email):
    raise ValueError("Invalid email address")
  # send email...
```

In this example, the email validation logic is repeated three times. Any change to this logic would require three edits, increasing the risk of errors and inconsistencies.

With DRY:

```
def validate_email(email):
  if not email or "@" not in email:
    return False
  return True

def update_profile(user, email):
  if not validate_email(email):
    raise ValueError("Invalid email address")
  user.email = email

def send_confirmation_email(email):
  if not validate_email(email):
    raise ValueError("Invalid email address")
  # send email... using validate_email(email)
```

Here, we extracted the email validation logic into a separate function, `validate_email`. Now, any changes to this logic only need to be done in one place, ensuring consistency and reducing error-prone duplication.

Q.18

MEDIUM

What is the difference between first party and third party cookies?

Both first-party and third-party cookies are small files stored on your computer by websites you visit. They track your activity and preferences, but they do so in different ways.

First-party cookies are created by the website you're on and can only be accessed by that website. They're like a little note that the website leaves on your computer to remember you next time you visit. They're used for things like:

- Keeping track of your login information so you don't have to type it in every time
- Remembering what items you've added to your shopping cart
- Tailoring the website to your preferences, such as language or font size

Third-party cookies are created by a different domain than the website you're on. They're like little spies that follow you around the internet, tracking your activity on different websites. They're used for things like:

- Showing you targeted advertising based on your interests
- Tracking your activity across different websites to build a profile of your interests

Feature	First-party cookies	Third-party cookies
Who creates them?	The website you're on	A different domain than the website you're on
Who can access them?	Only the website that created them	Any website that uses the same third-party code
What are they used for?	Remembering your preferences, keeping you logged in, etc.	Tracking your activity for advertising and other purposes

Privacy concerns

Third-party cookies have raised concerns about privacy, as they can be used to track your activity across the internet without your knowledge or consent. Many browsers now allow you to block or delete third-party cookies.

Q.19

MEDIUM

Describe the RESTful API design principles.

RESTful APIs follow six guiding principles:

1. **Uniform Interface:** Consistent resource naming and actions using HTTP methods (GET, POST, PUT, DELETE).
2. **Client-Server:** Separation of concerns between clients making requests and servers handling them.
3. **Statelessness:** Each request contains all information needed, servers don't "remember" past requests.
4. **Cacheable:** Resources can be cached by clients or intermediaries for better performance.
5. **Layered System:** Intermediaries can be placed between clients and servers without affecting communication.
6. **Code on Demand (Optional):** Servers can send executable code to clients to extend functionality.

These principles lead to well-designed, predictable, and scalable APIs.

Q.20

MEDIUM

Describe the RESTful API design principles.

The SOLID principles are a set of five principles in object-oriented design that aim to enhance the maintainability, flexibility, and scalability of software:

1. **Single Responsibility Principle (SRP):**
    - A class should have only one responsibility, promoting modular and understandable code.
2. **Open/Closed Principle (OCP):**
    - Software entities should be open for extension but closed for modification, facilitating adaptability through interfaces and abstract classes.
3. **Liskov Substitution Principle (LSP):**
    - Objects of a superclass should be replaceable with objects of a subclass without affecting program correctness, ensuring consistency in polymorphism.
4. **Interface Segregation Principle (ISP):**
    - A class should not be forced to implement interfaces it does not use, promoting focused and non-bloated interfaces.
5. Dependency Inversion Principle (DIP):
    - High-level modules should not depend on low-level modules; both should depend on abstractions, reducing coupling and improving flexibility.

Q.21

HARD

What are the advantages and disadvantages of microservices architecture?

Microservices are an architectural style that structures an application as a collection of small, loosely coupled, and independently deployable services.

Key Concepts:

- Independence: Each service has specific business function. Developed & scaled separately.
- Modularity: Breaking down a large, monolithic application into smaller, manageable pieces.

Advantages of Microservices:

- **Agility and Speed:** Faster development and deployment cycles due to independent services.
- **Scalability:** Individual services can be scaled up or down independently based on demand.
- **Resilience:** Failure of one service doesn't cripple the entire app.
- **Technology Choice:** Each service can use the best tool for the job without affecting others.

Disadvantages of Microservices:

- **Complexity:** Increased overhead in managing infrastructure, communication, and monitoring.
- **Testing:** Testing complex distributed systems can be challenging and time-consuming.
- **Debugging:** Identifying and fixing issues across services can be difficult.
- **Cost:** Initial setup and ongoing maintenance can be more expensive than monolithic.

Q.22

MEDIUM

What is the difference between horizontal and vertical scaling?

Horizontal scaling involves adding more machines or nodes to a system to distribute the load and increase performance.Vertical scaling involves increasing the resources (CPU, RAM, storage, etc.) on a single machine to improve its performance.

Feature	Horizontal Scaling	Vertical Scaling
Method	Add more machines	Add more resources to existing machine
Work distribution	Distributed across nodes	Single machine handles all workload
Flexibility & Resilience	High	Low
Management complexity	High	Low
Cost-effectiveness	High (long run)	Low (low workload)

Choosing between horizontal and vertical scaling depends on your specific needs. Here are some general guidelines:

Use horizontal scaling for:

- Handling high workloads and surges in traffic.
- Building highly resilient and available systems.

- Processing large datasets efficiently.

Use vertical scaling for:

- Simple workloads and applications.
- Rapid deployment and testing.
- Cost-efficiency for low workloads.

Q.23

MEDIUM

What is the difference between HTTP methods GET and POST?

- **GET:** Used to request data from a specified resource, with parameters appended to the URL. It is idempotent and suitable for data retrieval.

Example GET request:

```
GET /example/resource?param1=value1&param2=value2 HTTP/1.1
Host: example.com
```

- **POST:** Used to submit data to a specified resource, with data sent in the request body. It is non-idempotent and suitable for actions causing side effects, like form submissions.

Example POST request:

```
POST /example/resource HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
param1=value1&param2=value2
```

Feature	GET	POST
Purpose	Retrieve data	Send data
Data transfer	URL parameters	Request body
Visibility	Public (in URL)	Private (hidden)
Caching	Yes	No (usually)
Bookmarks	Yes	No
Idempotency	Yes	No
Data limitation	Yes	No
Security	Less secure	More secure

Q.24

MEDIUM

How can you maintain API Security?

Maintaining API security is crucial in today's digital landscape, where data breaches and unauthorized access can have severe consequences. Here are some key practices to keep your APIs secure:

- **Implement Token-based Authentication:** Ensure secure access to services and resources by assigning tokens to trusted identities.
- **Employ Encryption and Signatures:** Safeguard your data with encryption, such as TLS, and require signatures to verify the legitimacy of users accessing and modifying the data.
- **Identify and Address Vulnerabilities:** Stay vigilant by regularly updating operating systems, networks, drivers, and API components. Utilize sniffers to detect security issues and potential data leaks.
- **Implement Quotas and Throttling:** Set usage limits on API calls and monitor historical usage patterns. Unusual spikes in API calls may indicate misuse or errors, and implementing throttling rules can protect against abuse and potential Denial-of-Service attacks.
- **Utilize an API Gateway:** Deploy an API gateway as a central point for managing and securing API traffic. A robust gateway enables authentication, control, and analysis of API usage.

Q.25

HARD

What happens when you search for something on www.google.com?

Here's a breakdown of what happens when you search on Google, focusing on the backend aspects:

1. Query Submission:

- You enter your search term on the Google homepage.
- The browser sends a **GET request** to Google's servers, including your search query and other information (IP address, browser type, etc.).

2. Processing and Parsing:

- Google's web crawlers and indexing systems have already built a massive database of web pages and their content.
- The query is parsed and analyzed to understand its meaning and intent. This might involve stemming, synonymization, and entity recognition.

3. Ranking and Retrieval:

- The parsed query is matched against the indexed pages, using sophisticated algorithms like PageRank and BM25. These consider factors like relevance, authority, and freshness of the content.
- A ranked list of results is generated, prioritizing the most relevant and helpful pages.

4. Serving the Results:

- The search engine selects the top results and retrieves the necessary data from the database.
- This data is formatted into HTML snippets with titles, descriptions, and links to the original pages.
- The HTML response is sent back to your browser.

5. Displaying the Results:

- Your browser receives and interprets the HTML response, displaying the search results page with the ranked snippets.
- You can then click on the snippets to visit the relevant websites.

Additional Backend Considerations:

- **Load balancing:** Google distributes requests across its vast server network to handle high search volume efficiently.
- **Caching:** Frequently accessed data is cached to improve response times.
- **Personalization:** Search results can be personalized based on your location, search history, and other factors.
- **Security:** Google implements various security measures to protect user data and prevent malicious activities.

BOSSCODER ACADEMY

WHY BOSSCODER?

- **750+** Alumni placed at Top Product-based companies.
- More than **136% hike** for every **2 out of 3** working professional.
- Average package of **24LPA.**

> The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**
>
> **Lavanya**
> Meta

> Course is very well **structured** and **streamlined** to crack any **MAANG** company
>
> **Rahul .**
> Google

EXPLORE MORE