**✅ 1. Relational Databases (SQL)**

These are databases like MySQL, PostgreSQL, Oracle DB, etc.

**🔹 What to Know:**

| **Concept** | **You should understand** |
| --- | --- |
| **What is RDBMS?** | Tables, rows, columns, keys |
| **Normalization** | 1NF, 2NF, 3NF (just basic idea) |
| **Primary Key vs Foreign Key** | Definitions and differences |
| **Joins** | INNER, LEFT, RIGHT, FULL |
| **Basic Queries** | SELECT, INSERT, UPDATE, DELETE |
| **Aggregate Functions** | COUNT, SUM, AVG, GROUP BY, HAVING |
| **Subqueries & Nested queries** | Basic usage |
| **Constraints** | NOT NULL, UNIQUE, DEFAULT, CHECK |
| **Indexing** | Very basic idea: speeds up search |

**🔹 Sample Question:**

What is the difference between WHERE and HAVING?

**✅ 2. NoSQL Databases**

Examples: MongoDB, Cassandra, Firebase, etc.

**🔹 What to Know:**

| **Concept** | **You should understand** |
| --- | --- |
| **Why NoSQL?** | Use cases where relational DBs don’t fit well |
| **Types** | Document (MongoDB), Key-Value, Column, Graph |
| **MongoDB Basics** | Collection, Document, CRUD operations |
| **Differences vs SQL** | Schema-less, horizontal scaling |
| **Data Representation** | JSON-like format in MongoDB |

**🔹 Sample Question:**

How does MongoDB store data? What is a document?

**✅ 3. Data Mapping**

This is often in the context of **mapping objects in code to data in databases**.

**🔹 What to Know:**

| **Term** | **You should understand** |
| --- | --- |
| **ORM (Object Relational Mapping)** | Maps classes/objects to DB tables |
| **Examples** | Hibernate (Java), Mongoose (Node.js + MongoDB) |
| **Data Mapping Use** | Converting API or form data to DB-compatible format |

**🔹 Sample Question:**

What is ORM and why is it used?

**✅ 4. REST-based Web Services**

REST APIs are how systems communicate over HTTP.

**🔹 What to Know:**

| **Concept** | **You should understand** |
| --- | --- |
| **What is REST?** | Representational State Transfer |
| **HTTP Methods** | GET, POST, PUT, DELETE |
| **Status Codes** | 200, 201, 400, 404, 500 |
| **JSON** | As data exchange format |
| **Postman Tool** | For testing APIs (basics) |
| **API Endpoint** | What a route looks like: /api/users/:id |

**🔹 Sample Question:**

What is the difference between PUT and POST in REST?

**✅ How Much Depth is Expected for Kyndryl (Non-Product Company)?**

| **Topic** | **Depth** |
| --- | --- |
| SQL | Moderate (queries, joins, constraints) |
| NoSQL | Basic understanding, mostly MongoDB |
| Data Mapping | Conceptual understanding of how data flows |
| REST API | Understand methods, status codes, maybe test an API |

**✅ Summary — What to Do Practically**

* ✅ Revise **SQL** queries and theory
* ✅ Watch a **30-min MongoDB crash course** (YouTube)
* ✅ Understand what **REST APIs** do, test one on **Postman**
* ✅ Learn what **ORMs** like Hibernate or Mongoose do (just the idea)

**✅ Today's Focus:**

**🔹 What is RDBMS?**

Let’s understand the basics, and then I’ll give you common interview questions + how to answer them **confidently as a fresher**.

**🔍 What is RDBMS?**

**🔸 RDBMS stands for:**

**Relational Database Management System**  
It stores data in **tables** (rows and columns), where each table is **related** to another through keys.

* **Table** → like a spreadsheet
* **Row (Tuple)** → one complete record
* **Column (Attribute)** → a property of the data

**🔸 Examples of RDBMS:**

* MySQL
* PostgreSQL
* Oracle DB
* SQL Server

**📌 Real-Life Analogy:**

Think of a **school database**.

* **Table: Students** | ID | Name | Class | |----|------|-------| | 1 | Rahul | 10A | | 2 | Priya | 9B |
* **Table: Marks** | StudentID | Subject | Marks | |-----------|---------|-------| | 1 | Math | 95 |

These tables can be **linked using a key** — the **StudentID** in "Marks" refers to the **ID** in "Students".

**✅ Most Common Interview Questions from This Concept:**

**🟢 1. What is RDBMS? How is it different from DBMS?**

✅ **Answer (Simple):**  
RDBMS stores data in tables with rows and columns and uses **relationships** between them.  
DBMS may not use relational structures and can store data in files, documents, etc.

**🟢 2. What is a table in SQL?**

✅ **Answer:**  
A table is a collection of data organized in rows and columns. Each row is a record, and each column represents a field.

**🟢 3. What is a record and a field?**

✅ **Answer:**

* A **record** is one complete row in a table (like a student’s full info).
* A **field** is a single column in the table (like “Name” or “ID”).

**🟢 4. What are keys in RDBMS?**

✅ **Answer (Briefly):**  
Keys are constraints used to identify rows in a table and maintain relationships.

You can say:

"The **Primary Key** uniquely identifies each record. A **Foreign Key** connects one table to another."

**🟢 5. Why do we need RDBMS instead of storing in Excel or files?**

✅ **Answer:**  
RDBMS supports:

* **Data integrity**
* **Efficient search & query**
* **Relationships across tables**
* **Security and scalability**

**✍️ Mini Task (Practice writing this):**

Create a simple table Employees with 3 columns (ID, Name, Salary):

sql

Copy code

CREATE TABLE Employees (

ID INT PRIMARY KEY,

Name VARCHAR(50),

Salary INT

);

Try inserting a few rows:

sql

Copy code

INSERT INTO Employees VALUES (1, 'Amit', 50000);

INSERT INTO Employees VALUES (2, 'Neha', 60000);

**✅ Topic 2: Normalization + Primary Key vs Foreign Key**

**🔸 What is Normalization?**

**✅ Definition (Simple):**

**Normalization** is the process of organizing data in a database to remove **redundancy** (duplicate data) and ensure **data integrity**.

**🔹 Why Normalize?**

| **Without Normalization** | **With Normalization** |
| --- | --- |
| Repeating data | No duplication |
| Waste of storage | Efficient storage |
| Risk of inconsistency | Data integrity |

**✅ Example:**

**🔸 Without Normalization (One table with repetition):**

| **StudentID** | **Name** | **Subject** | **Marks** |
| --- | --- | --- | --- |
| 1 | Rahul | Math | 90 |
| 1 | Rahul | Science | 85 |
| 2 | Priya | Math | 95 |

**🔸 After Normalization (Two tables):**

**Table 1: Students**

| **StudentID** | **Name** |
| --- | --- |
| 1 | Rahul |
| 2 | Priya |

**Table 2: Marks**

| **StudentID** | **Subject** | **Marks** |
| --- | --- | --- |
| 1 | Math | 90 |
| 1 | Science | 85 |
| 2 | Math | 95 |

**🔹 Normal Forms to Know (Just 1NF, 2NF, 3NF for interviews)**

| **Normal Form** | **Rule** |
| --- | --- |
| **1NF** | Remove repeating groups (each cell should have atomic data) |
| **2NF** | Remove partial dependencies (non-key attributes depend on the whole key) |
| **3NF** | Remove transitive dependencies (non-key attributes should depend only on the primary key) |

**🔸 Interview Questions on Normalization:**

**🟢 1. What is normalization and why is it important?**

✅ **Answer:**  
Normalization is the process of structuring a database to eliminate redundancy and ensure data integrity. It helps save space and avoid inconsistencies.

**🟢 2. What are the types of normalization?**

✅ **Answer (Short):** 1NF: Remove repeating data  
2NF: Remove partial dependencies  
3NF: Remove transitive dependencies

**🟢 3. Can you give a real-life example of normalization?**

✅ **Answer:**  
In a school database, instead of storing student details along with every subject they take (duplicating data), we separate them into two tables: one for students, and one for marks/subjects.

**✅ Now: Primary Key vs Foreign Key**

**🔸 Primary Key**

* Uniquely identifies each record in a table.
* Cannot be NULL.
* One per table.

✅ Example:

sql

CopyEdit

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

Name VARCHAR(50)

);

**🔸 Foreign Key**

* A column that **refers to the primary key** in another table.
* Helps maintain relationships.

✅ Example:

sql

CopyEdit

CREATE TABLE Marks (

StudentID INT,

Subject VARCHAR(50),

Marks INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID)

);

**🟢 Interview Question:**

**What is the difference between primary key and foreign key?**

✅ **Answer:**

* A **primary key** uniquely identifies a row in a table.
* A **foreign key** links one table to another by referring to the primary key.

Perfect! Let’s move on to one of the **most asked** and **super important topics** in SQL interviews — especially for freshers:

**✅ Topic 3: SQL JOINS**

**🔹 What are Joins?**

Joins are used to **combine rows** from two or more tables **based on a related column**, typically using a **foreign key**.

**✅ Real-Life Example:**

Imagine two tables:

**Customers**

| **CustomerID** | **Name** |
| --- | --- |
| 1 | Aman |
| 2 | Riya |

**Orders**

| **OrderID** | **CustomerID** | **Product** |
| --- | --- | --- |
| 101 | 1 | Laptop |
| 102 | 2 | Headphones |
| 103 | 1 | Keyboard |

We can join them using the CustomerID to find out which customer ordered what.

**🔸 Types of Joins (with Simple Explanation)**

| **Join Type** | **Description** |
| --- | --- |
| **INNER JOIN** | Returns only the matching rows from both tables |
| **LEFT JOIN** | Returns all rows from the **left** table, and matched rows from the right |
| **RIGHT JOIN** | Returns all rows from the **right** table, and matched rows from the left |
| **FULL JOIN** | Returns all rows when there is a match in one of the tables |

**🔍 Visual Summary**

| **Type** | **Returns** |
| --- | --- |
| INNER JOIN | Matches only (common in both) |
| LEFT JOIN | All from left + matches from right |
| RIGHT JOIN | All from right + matches from left |
| FULL OUTER | All from both, matched + unmatched |

**🔸 Syntax Examples**

**✅ INNER JOIN:**

sql

CopyEdit

SELECT Customers.Name, Orders.Product

FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**Result:**

| **Name** | **Product** |
| --- | --- |
| Aman | Laptop |
| Riya | Headphones |
| Aman | Keyboard |

**✅ LEFT JOIN:**

sql

CopyEdit

SELECT Customers.Name, Orders.Product

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

If any customer has **no order**, they will still appear, with NULL in product column.

**🔸 Most Asked Interview Questions on JOINS**

**🟢 1. What is a JOIN in SQL?**

✅ **Answer:**  
A JOIN is used to combine rows from two or more tables based on a related column, usually a foreign key.

**🟢 2. What is the difference between INNER JOIN and LEFT JOIN?**

✅ **Answer:**

* **INNER JOIN** returns only matching rows.
* **LEFT JOIN** returns all rows from the left table, even if there is no match in the right table.

**🟢 3. What if there’s no matching record in JOIN?**

✅ **Answer:**

* In **INNER JOIN**, the row is skipped.
* In **LEFT/RIGHT JOIN**, the non-matching part becomes NULL.

**🟢 4. Which JOIN will return unmatched records from both sides?**

✅ **Answer:**  
**FULL OUTER JOIN**.

But note: Not all RDBMS (like MySQL) support FULL JOIN directly — you may need UNION of LEFT and RIGHT JOIN.

**🧠 Quick Practice Challenge (Try this):**

**Tables:**

**Employees**

| **EmpID** | **Name** |
| --- | --- |
| 1 | Ravi |
| 2 | Sneha |
| 3 | Kabir |

**Departments**

| **EmpID** | **Dept** |
| --- | --- |
| 1 | HR |
| 2 | IT |

❓ Write a query to list all employees and their departments (even if they don't have one).

✅ Answer (LEFT JOIN):

sql

CopyEdit

SELECT Employees.Name, Departments.Dept

FROM Employees

LEFT JOIN Departments ON Employees.EmpID = Departments.EmpID;

**✅ Topic 4: Basic SQL Queries**

(*SELECT, WHERE, GROUP BY, ORDER BY, HAVING — the most used parts of SQL!*)

**🔹 1. SELECT Statement (Retrieving Data)**

sql

CopyEdit

SELECT column1, column2 FROM table\_name;

✅ Example:

sql

CopyEdit

SELECT Name, Age FROM Employees;

🧠 **Tip:** Use \* to select all columns:

sql

CopyEdit

SELECT \* FROM Employees;

**🔹 2. WHERE Clause (Filtering Rows)**

sql

CopyEdit

SELECT \* FROM Employees WHERE Age > 25;

📌 Supports operators:

* =, !=, <, >, <=, >=
* BETWEEN, IN, LIKE, IS NULL

✅ Examples:

sql

CopyEdit

SELECT \* FROM Employees WHERE Name = 'Ravi';

SELECT \* FROM Employees WHERE Age BETWEEN 20 AND 30;

SELECT \* FROM Employees WHERE Department IN ('HR', 'IT');

SELECT \* FROM Employees WHERE Name LIKE 'R%'; -- Starts with R

**🔹 3. ORDER BY (Sorting Results)**

sql

CopyEdit

SELECT \* FROM Employees ORDER BY Age ASC; -- or DESC

**🔹 4. GROUP BY (Grouping Rows)**

Used with aggregate functions like SUM, AVG, COUNT.

✅ Example:

sql

CopyEdit

SELECT Department, COUNT(\*)

FROM Employees

GROUP BY Department;

**🔹 5. HAVING Clause (Filter after Grouping)**

💡 WHERE is used before grouping, HAVING is used **after** grouping.

✅ Example:

sql

CopyEdit

SELECT Department, COUNT(\*)

FROM Employees

GROUP BY Department

HAVING COUNT(\*) > 2;

**🔥 Must-Know Aggregate Functions**

| **Function** | **Purpose** |
| --- | --- |
| COUNT() | Total records |
| SUM() | Add values |
| AVG() | Average |
| MAX() | Highest value |
| MIN() | Lowest value |

✅ Example:

sql

CopyEdit

SELECT AVG(Salary) FROM Employees;

**🔸 Most Asked Interview Questions**

**🟢 1. Difference between WHERE and HAVING?**

✅ **Answer:**

* WHERE filters rows **before** grouping.
* HAVING filters groups **after** GROUP BY.

**🟢 2. Write a query to find the average salary of employees in each department.**

✅ **Answer:**

sql

CopyEdit

SELECT Department, AVG(Salary)

FROM Employees

GROUP BY Department;

**🟢 3. Write a query to get employees whose name starts with ‘A’.**

sql

CopyEdit

SELECT \* FROM Employees WHERE Name LIKE 'A%';

**🟢 4. How do you find duplicate records in a table?**

✅ Example:

sql

CopyEdit

SELECT Name, COUNT(\*)

FROM Employees

GROUP BY Name

HAVING COUNT(\*) > 1;

**🟢 5. Find the highest salary in each department.**

sql

CopyEdit

SELECT Department, MAX(Salary)

FROM Employees

GROUP BY Department;

**✅ Topic 6: SQL Constraints**

**Constraints** are rules applied to columns in a table to enforce data integrity and consistency.

**🔹 Types of Constraints:**

| **Constraint** | **Description** |
| --- | --- |
| NOT NULL | Ensures a column cannot have NULL values |
| UNIQUE | Ensures all values in a column are unique |
| PRIMARY KEY | Uniquely identifies each record in a table (NOT NULL + UNIQUE) |
| FOREIGN KEY | Links two tables together |
| CHECK | Ensures values satisfy a specific condition |
| DEFAULT | Assigns a default value if no value is provided |
| AUTO\_INCREMENT | Automatically generates unique values (used in IDs) |

**✅ 1. NOT NULL**

sql

CopyEdit

CREATE TABLE Students (

ID INT NOT NULL,

Name VARCHAR(100) NOT NULL

);

🔹 Used when a column **must** have a value.

**✅ 2. UNIQUE**

sql

CopyEdit

CREATE TABLE Users (

Email VARCHAR(100) UNIQUE

);

🔹 Ensures no two users can have the same email.

**✅ 3. PRIMARY KEY**

sql

CopyEdit

CREATE TABLE Students (

RollNo INT PRIMARY KEY,

Name VARCHAR(100)

);

🔹 Combination of NOT NULL + UNIQUE.

**✅ 4. FOREIGN KEY**

sql

CopyEdit

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

🔹 Links Orders to the Customers table.

**✅ 5. CHECK**

sql

CopyEdit

CREATE TABLE Employees (

Age INT CHECK (Age >= 18)

);

🔹 Ensures values follow a condition.

**✅ 6. DEFAULT**

sql

CopyEdit

CREATE TABLE Products (

ProductName VARCHAR(50),

InStock INT DEFAULT 1

);

🔹 If no value is provided, InStock defaults to 1.

**🔸 BONUS: AUTO\_INCREMENT (for MySQL)**

sql

CopyEdit

CREATE TABLE Employees (

ID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100)

);

🔹 Automatically generates unique ID values for new rows.

**🔥 Most Asked Interview Questions**

**🟢 1. What is the difference between PRIMARY KEY and UNIQUE?**

✅ **Answer:**

* Both enforce uniqueness.
* **PRIMARY KEY** = UNIQUE + NOT NULL
* A table can have **only one primary key**, but multiple unique keys.

**🟢 2. Can a table have multiple FOREIGN KEYs?**

✅ **Answer:**  
Yes, a table can reference multiple other tables using multiple foreign keys.

**🟢 3. Can a column be both UNIQUE and NOT NULL?**

✅ **Answer:**  
Yes. This is basically what a Primary Key does.

**🟢 4. What is the purpose of CHECK constraint?**

✅ **Answer:**  
To restrict column values to satisfy a condition (e.g., Age > 18).

**🟢 5. What is the difference between DEFAULT and NULL?**

✅ **Answer:**

* **DEFAULT** provides a value **automatically** when none is given.
* **NULL** means the column is **left blank**.

**🔥 Interview Questions**

**🟢 1. What is indexing? Why is it used?**

✅ **Answer:**  
Indexing is a technique to improve query performance by reducing the amount of data scanned. It’s like creating a shortcut to data.

**🟢 2. What are the downsides of using too many indexes?**

✅ **Answer:**

* Increased storage
* Slower write operations (insert/update/delete)
* Indexes must be maintained

**🟢 3. When should you not use indexing?**

✅ **Answer:**

* On small tables (not needed).
* On columns with lots of duplicates.
* If the column is not used in WHERE, JOIN, ORDER BY

**🟢 4. What is the difference between clustered and non-clustered indexes?**

✅ **Answer:**

* **Clustered:** Reorders the actual data rows.
* **Non-clustered:** Creates a pointer to data rows. Most DBs use this by default.

**🧠 Practice Task:**

Create a table and:

1. Add an index on a frequently searched column (e.g., Name).
2. Try running a SELECT query with and without index — observe execution time if you're using a tool like MySQL Workbench or pgAdmin.

✅ That wraps up your **SQL Fundamentals** module — great work so far!

You’ve covered:

1. RDBMS Basics
2. Normalization
3. Keys
4. Joins
5. Subqueries
6. Constraints
7. Indexing

## ✅ ****NoSQL Databases (Beginner-Friendly Overview)****

NoSQL stands for "**Not Only SQL**". It’s used when traditional relational databases (SQL) are **not suitable** due to huge data size, unstructured data, or dynamic schema needs.

### 🔹 1. ****Types of NoSQL Databases****

| **Type** | **Example** | **Use Case** |
| --- | --- | --- |
| **Document** | MongoDB | JSON-like data |
| **Key-Value** | Redis | Caching, fast lookups |
| **Column-Family** | Cassandra | Big data, analytics |
| **Graph** | Neo4j | Social networks, recommendations |

As a fresher, **focus on Document (MongoDB)** — most beginner-friendly and popular.

### 🔹 2. ****MongoDB Basics (Document-Oriented)****

MongoDB stores data in JSON-like documents called **BSON**.

#### ✅ Example Document:

json

CopyEdit

{

"name": "Alice",

"age": 23,

"skills": ["Python", "SQL"],

"address": {

"city": "Mumbai",

"pincode": 400001

}

}

No rows or columns — data is flexible.

### 🔸 SQL vs NoSQL Quick Comparison

| **Feature** | **SQL (MySQL, etc.)** | **NoSQL (MongoDB)** |
| --- | --- | --- |
| Structure | Table-based | Document-based (JSON) |
| Schema | Fixed schema | Flexible schema |
| Relations | Supports joins | No joins (denormalized) |
| Scaling | Vertical | Horizontal (easier) |
| Transactions | Strong ACID | Mostly BASE (eventual consistency) |

### 🔹 3. MongoDB Basic Commands

| **Operation** | **SQL Equivalent** | **MongoDB Command** |
| --- | --- | --- |
| Insert | INSERT | db.users.insertOne({...}) |
| Read | SELECT | db.users.find({}) |
| Filtered Read | SELECT WHERE | db.users.find({age: 23}) |
| Update | UPDATE | db.users.updateOne({...}) |
| Delete | DELETE | db.users.deleteOne({...}) |

## 🔥 Most Expected Interview Questions on NoSQL

### 🟢 1. ****What is NoSQL and when would you use it?****

✅ **Answer:**  
NoSQL databases store unstructured or semi-structured data. They're best for large-scale, flexible, or fast-evolving applications (e.g., social apps, chat, sensor data).

### 🟢 2. ****Difference between SQL and NoSQL?****

✅ Covered above: structure, schema, joins, scaling.

### 🟢 3. ****What are Documents in MongoDB?****

✅ **Answer:**  
Documents are JSON-like records that store data as key-value pairs, often nested and flexible in structure.

### 🟢 4. ****Can you perform JOINs in NoSQL (MongoDB)?****

✅ **Answer:**  
Not directly like SQL. But MongoDB has a $lookup operator to mimic joins

Let's continue with the **REST API** topic now.

**✅ REST API (Representational State Transfer)**

REST is a **web architecture style** used to build APIs that allow **communication between client and server** over HTTP.

**🔹 Real-life Analogy**

Think of a **restaurant**:

* **Client** = You (placing an order)
* **API** = Waiter (communicates between you and kitchen)
* **Server** = Kitchen (prepares the data/response)
* **Database** = Ingredients storage

**🔹 1. REST Constraints (in simple terms)**

| **Constraint** | **Meaning** |
| --- | --- |
| **Stateless** | Each request is independent. No history is stored on the server. |
| **Client-Server** | UI (frontend) is separate from backend logic. |
| **Cacheable** | Responses can be stored temporarily to improve speed. |
| **Uniform Interface** | Consistent way to access resources using URLs. |
| **Layered System** | May use intermediaries like load balancers, proxies. |

**🔹 2. HTTP Methods**

| **Method** | **Use for** | **Example** |
| --- | --- | --- |
| GET | Read data | GET /users |
| POST | Create data | POST /users |
| PUT | Update/replace | PUT /users/1 |
| PATCH | Update partially | PATCH /users/1 |
| DELETE | Delete data | DELETE /users/1 |

**🔹 3. REST API URL Design (Endpoints)**

| **Resource** | **Endpoint** |
| --- | --- |
| All Users | GET /users |
| Single User | GET /users/1 |
| Create User | POST /users |
| Update User | PUT /users/1 |
| Delete User | DELETE /users/1 |

**🔹 4. JSON – Most common REST data format**

Example response from server:

json

CopyEdit

{

"id": 1,

"name": "Ravi",

"email": "ravi@example.com"

}

**🔹 5. Sample Interview Questions (with answers)**

**✅ 1. What is REST API?**

**Answer:**  
REST API is a way for client and server to communicate over HTTP using a set of rules like statelessness, uniform interface, and standard HTTP methods like GET, POST, PUT, DELETE.

**✅ 2. What’s the difference between PUT and PATCH?**

**Answer:**

* PUT replaces the entire resource.
* PATCH updates only the fields that are provided.

**✅ 3. Why is REST API stateless?**

**Answer:**  
Because each request from client to server must contain all the information needed. Server does not remember previous requests. This makes REST simple and scalable.

**✅ 4. What format is used to send/receive data?**

**Answer:**  
Mostly JSON (JavaScript Object Notation).

**✅ 5. What is the status code 200, 201, 404, 500?**

| **Code** | **Meaning** |
| --- | --- |
| 200 | OK (success) |
| 201 | Created (POST success) |
| 404 | Not Found |
| 500 | Server Error |