**CRUD Operations , SQL and NoSQL Databases**

* **Pratyush Jagdish Birole**

The databases I have used are:-

**SQL- SQLite3**

**NoSQL - MongoDB**

**GraphQL**



CRUD stands for Create, Read, Update, and Delete, and is a set of fundamental operations used in DBMS :

**Create**: Used to add new records to a database

**Read**: Used to access or retrieve existing records from a database Update: Used to modify existing records in a database

**Update:** Updating existing record from DB

**Delete**: Used to remove records from a database

**Advantages of CRUD:-**

### **1. Data Consistency:**

* **Uniformity:** CRUD operations enforce a standard way of interacting with data, preventing inconsistencies and errors that might arise from ad-hoc data manipulation.
* **Data Integrity:** By defining clear rules for creating, reading, updating, and deleting data, CRUD helps maintain data integrity and prevents unauthorized changes.

### **2. Efficiency:**

* **Optimization:** Database systems can optimize CRUD operations, leading to efficient data retrieval and manipulation.
* **Performance:** Well-designed CRUD operations can improve application performance by minimizing unnecessary data processing.

### **3. Maintainability:**

* **Modularity:** CRUD operations can be broken down into smaller, more manageable components, making code easier to understand and maintain.
* **Scalability:** As applications grow, CRUD operations can be scaled to accommodate larger datasets and increased user loads.

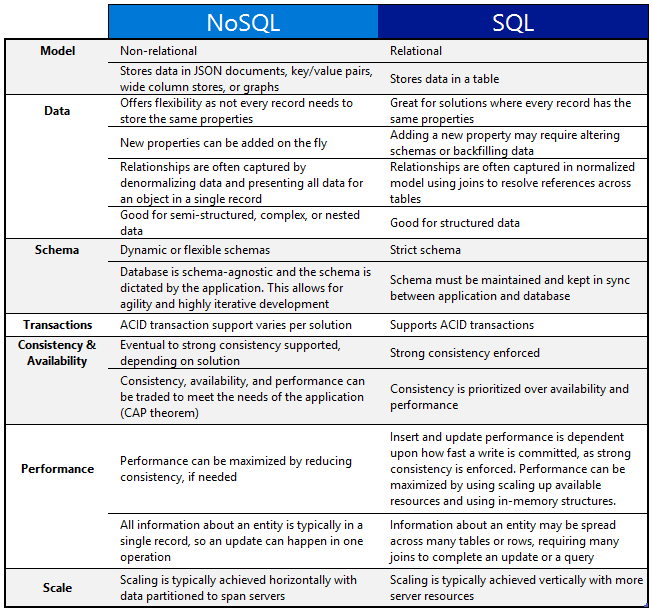
### **4. Security:**

* **Access Control:** CRUD operations can be used to implement granular access control, ensuring that only authorized users can perform specific actions on data.
* **Data Protection:** By limiting the ways in which data can be modified, CRUD helps protect sensitive information from unauthorized access or tampering.

### **5. Interoperability:**

* **Standard Interface:** CRUD operations provide a common interface for interacting with data, making it easier to integrate with other systems and applications.
* **Data Exchange:** CRUD can be used to facilitate data exchange between different systems, promoting interoperability and collaboration.

**Difference between SQL Databases and No SQL Databases**

****

**SQL Database:-**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases

SELECT, UPDATE, DELETE, INSERT, WHERE

**SQlite 3**

**What is a cursor in Python?**

It is an object that allows you to iterate over the results of a database query. It acts as a pointer that moves through the result set, one row at a time.

**Example usage :-**

1. **Create a cursor**

**conn = sqlite3.connect('mydatabase.db')**

**cursor = conn.cursor() # Execute a query cursor.execute("SELECT \* FROM mytable")**

****

**Creating a table In SQLite 3**

**command\_create = """**

**CREATE TABLE IF NOT EXISTS status (**

**id INTEGER PRIMARY KEY,**

**task TEXT,**

**date TEXT,**

**status TEXT**

**)**

**"""**

**cur.execute(command\_create)**

**print("Table created")**

**Inserting into table**

**command\_insert = """**

**INSERT INTO status (id, task, date, status)**

**VALUES**

**(1, 'Insert Operations for SQLite', '13-09-2024', 'done'),**

**(2, 'Create fetch command to fetch data', '14-09-2024', 'done')**

**"""**

**Fetching Data**

**command\_retrieve = "SELECT \* FROM status"**

**cur.execute(command\_retrieve)**

**rows = cur.fetchall()**

**Updating**

**command\_update = "UPDATE status SET status='done' WHERE id=2"**

**cur.execute(command\_update)**

**con.commit()**

**Deleting**

**command\_delete = "DELETE FROM status WHERE id=1"**

**cur.execute(command\_delete)**

**Task I performed :- Loaded a CSV File and then performed CRUD operations in SQLite3**

****

**No SQL DB - Pymongo**

**Connecting with MongoDB Client**

**client = MongoClient('localhost', 27017)**

**Creating or inserting documents**

**status\_1 = { "task": "Create fetch command to fetch data", "date": "14-09-2024", "status": "notdone" }**

**Updating a document**

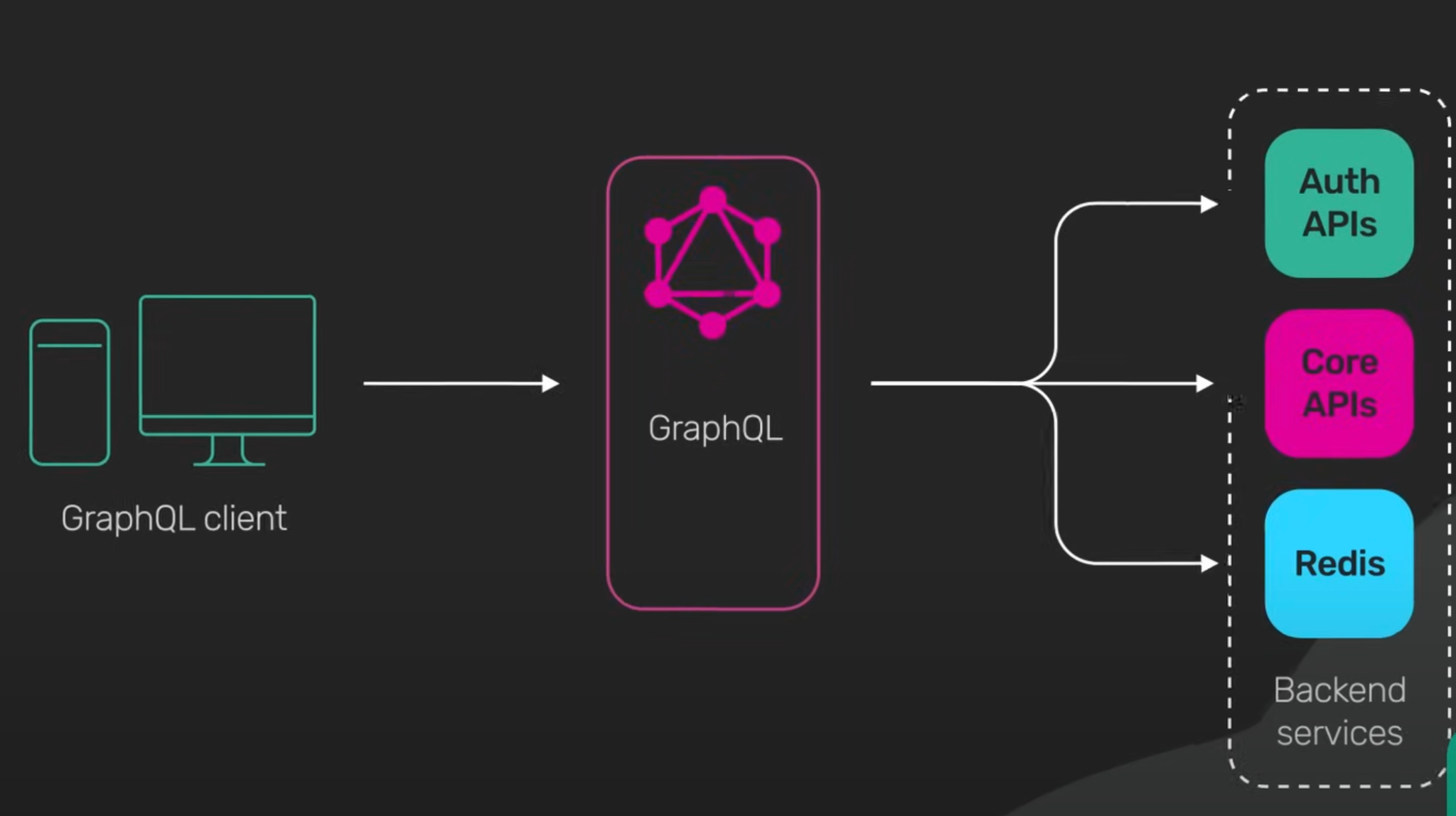
**status.update\_one({"task": "Create fetch command to fetch data"}, {"$set": {"status": "done"}})**

**Deleting a document**

**status.delete\_one({"task": "Reupdating in DB"})**

**Task Performed - Loaded a CSV file and performed CRUD Operations**

**Graph QL:-**

**Graph QL is a query language used for API .It provides the schema of the data **

Client-side Query: A client sends a query specifying exactly what data it needs

Server-side Execution: The server, using a schema that defines the data types and possible queries, resolves the query by fetching data from various sources (like a database or other APIs).

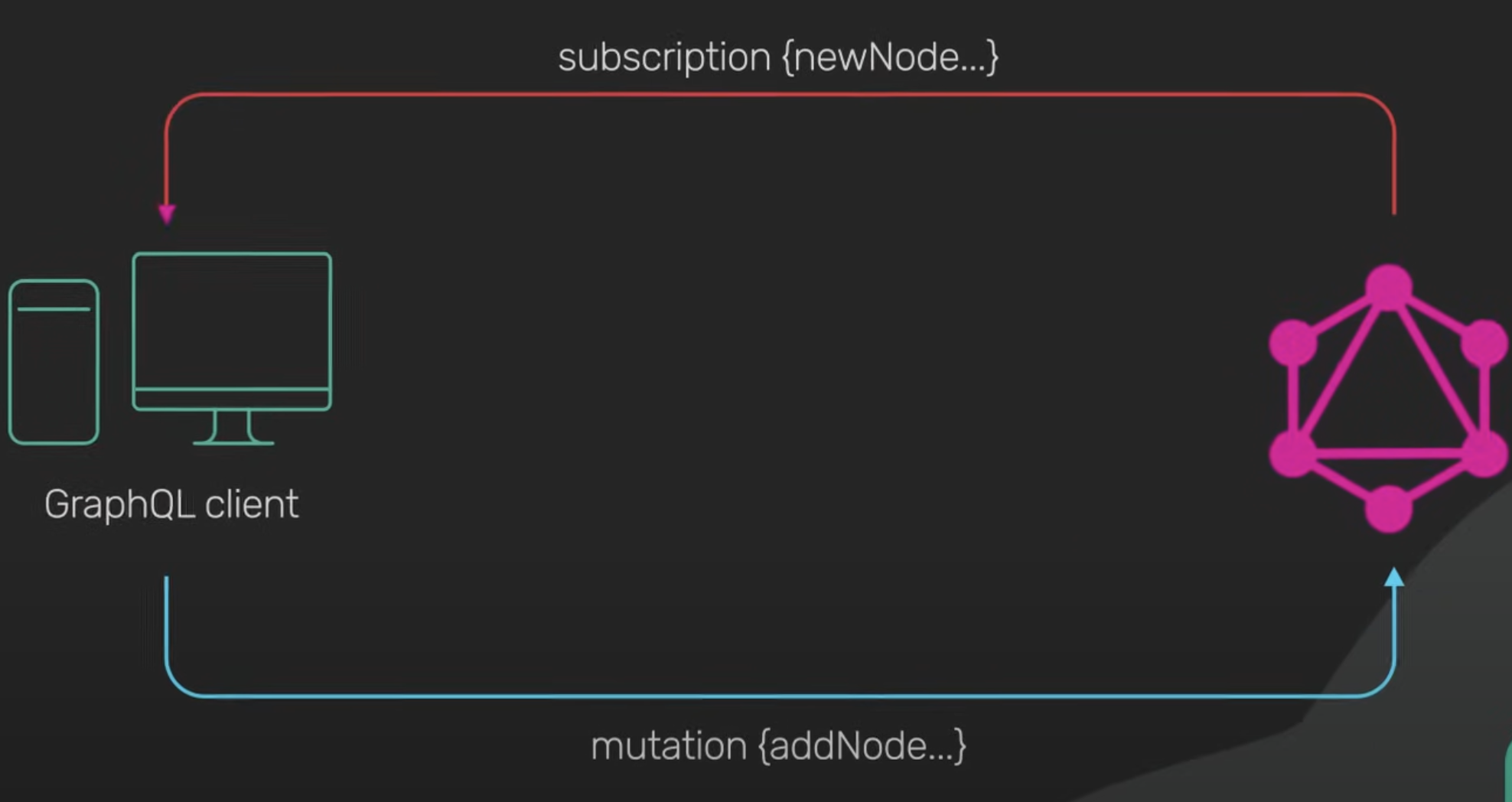
Response: The server returns a JSON response that matches the structure of the query.

1.Query (to fetch data):

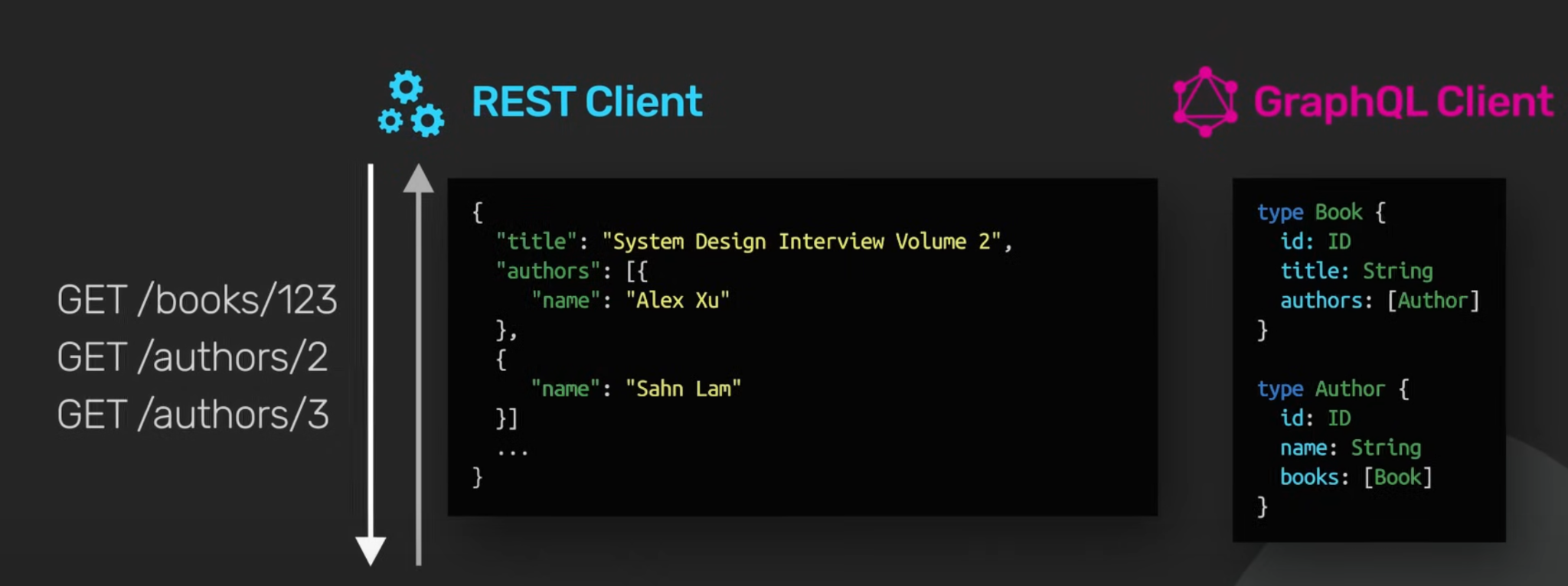
query { user(id: 1) { name email posts { title } } }

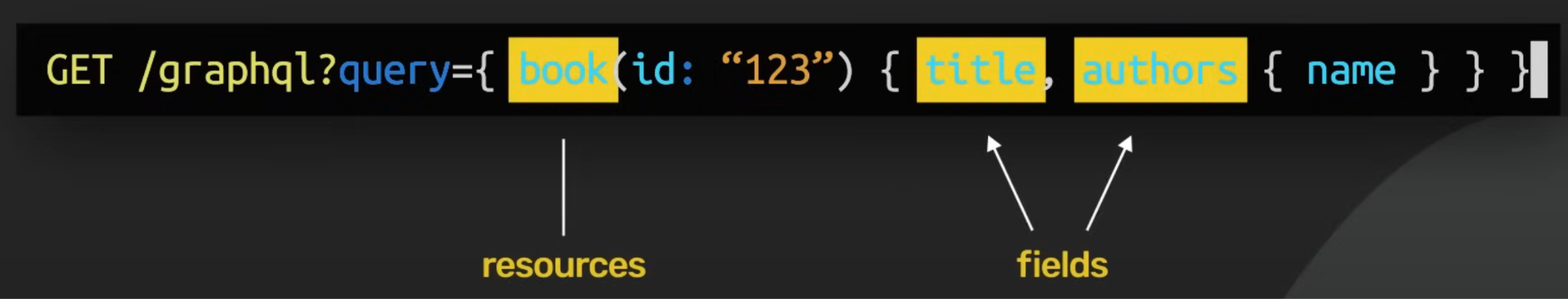
2.Mutation (to modify data):

mutation { createUser(name: "pratyush", email: "pratyush@example.com") { id name } }

****

**Difference between Rest API and Graph QL**

****

****

**What I have done**

**Made a simple task management API using GraphQL. It supports:**

* **Creating tasks**
* **Updating tasks**
* **Deleting tasks**
* **Retrieving a task by its ID**

1. Tasks Storage tasks: A list that acts as an in-memory storage for the task objects. Each task is a dictionary containing an id, task (task description), date (task date), and status (task completion status).

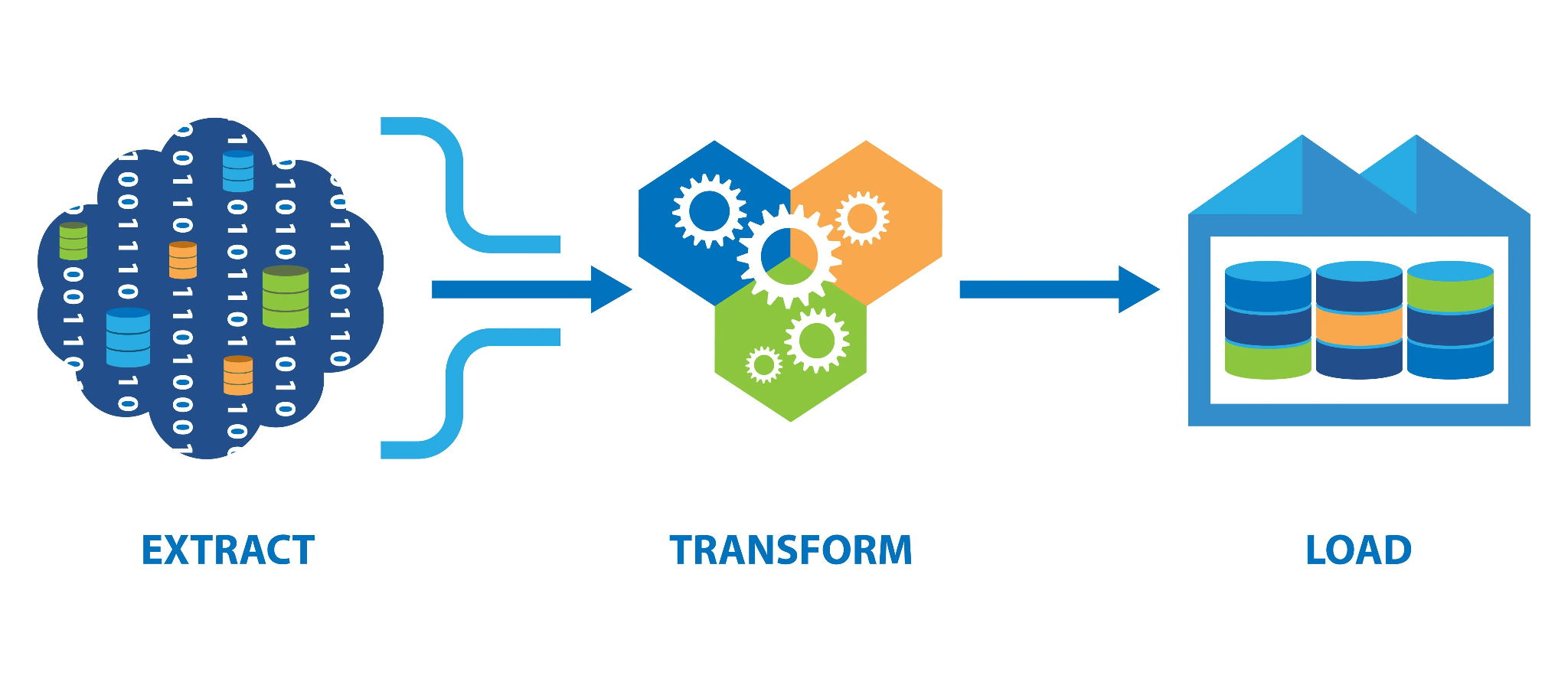
2. GraphQL Types TaskType: A GraphQL object type that defines the structure of a task. Each task has an id, task, date, and status as fields. TaskInput: An input type that specifies the required fields (task, date, status) when creating or updating tasks.

3. GraphQL Mutations CreateTask: A mutation that allows creating a new task. It accepts an input of type TaskInput. The new task is assigned an incremental ID and is appended to the tasks list. The mutation returns the newly created task. UpdateTask: A mutation for updating an existing task. It takes an id to identify the task and an optional input to update the task’s details. If the task with the given ID is found, its fields are updated.

DeleteTask: A mutation to delete a task by id. It removes the task from the tasks list and returns the deleted task.

4. GraphQL Queries Query: Defines the ability to retrieve a task by its id. If the task is found, it returns the task data.

**Data Loading and Performing CRUD operations:-**

****

**Graphene - A key component while developing a graphQL API**

Graphene is a Python library used to build GraphQL APIs. It provides a way to define your GraphQL schema and resolve queries and mutations

### **Key Features of Graphene:**

1. **Schema Definition:**
   * Allows you to define the GraphQL schema using Python classes. You specify types, queries, mutations, and their relationships in a straightforward and Pythonic way.
2. **Object Types:**
   * Define the types of data that can be queried. Each type can have fields that are themselves types.
3. **Queries and Mutations:**
   * Define the entry points for fetching and modifying data. Queries are used to fetch data, while mutations are used to modify data.