

USERS CREATE:

The screenshot shows the Apollo Sandbox interface with the following details:

- URL:** `http://localhost:4001/`
- Documentation:** Shows the `Query` root field with `users: [User!]`.
- Operation:** A mutation is defined:

```
1 mutation($name: String!, $email: String!) {  
2   createUser(name: $name, email: $email) {  
3     id  
4     email  
5     name  
6   }  
7 }
```
- Variables:**

```
1 {  
2   "name": "Jericho",  
3   "email": "jericho@gmail.com"  
4 }
```
- Response:** A successful JSON response:

```
{  
  "data": {  
    "users": [  
      {  
        "email": "jericho@gmail.com",  
        "id": 1,  
        "name": "Jericho"  
      }  
    ]  
  }  
}
```
- Status:** 200, 40.0ms, 75B

USERS READ:

The screenshot shows the Apollo Sandbox interface with the following details:

- URL:** `http://localhost:4001/`
- Documentation:** Shows the `Query` root field with `users: [User!]`.
- Operation:** A query is defined:

```
1 query {  
2   users {  
3     email  
4     id  
5     name  
6   }  
7 }  
8
```
- Variables:**

```
1 {  
2   "name": "Jericho",  
3   "email": "jericho@gmail.com"  
4 }
```
- Response:** A successful JSON response:

```
{  
  "data": {  
    "users": [  
      {  
        "email": "jericho@gmail.com",  
        "id": 1,  
        "name": "Jericho"  
      }  
    ]  
  }  
}
```
- Status:** 200, 17.0ms, 75B

USERS UPDATE:

Unnamed × +

Operation

```
1 mutation ($updateUserId: Int!, $name: String!, $email: String!) {  
2   updateUser(id: $updateUserId, name: $name, email: $email) {  
3     id  
4     name  
5     email  
6   }  
7 }  
8
```

Run

Response

```
{  
  "data": {  
    "updateUser": {  
      "id": 1,  
      "name": "Jericho Update",  
      "email": "jerichoupdate@gmail.com"  
    }  
  }  
}
```

Variables Headers Pre-Operation Script Post-Operation Script

```
1 {  
2   "name": "Jericho Update",  
3   "email": "jerichoupdate@gmail.com",  
4   "updateUserId": 1  
5 }
```

JSON

+ Add files

Unnamed × +

Operation

```
1 query {  
2   users {  
3     name  
4     email  
5     id  
6   }  
7 }
```

Run

Response

```
{  
  "data": {  
    "users": [  
      {  
        "name": "Jericho  
Update",  
        "email":  
"jerichoupdate@gmail.  
com",  
        "id": 1  
      }  
    ]  
  }  
}
```

Variables Headers Pre-Operation Script Post-Operation Script

```
1 {  
2   "name": "Jericho Update",  
3   "email": "jerichoupdate@gmail.com",  
4   "updateUserId": 2  
5 }
```

JSON

+ Add files

USERS DELETE:

The screenshot shows the GraphQL Studio interface with a tab titled "Unnamed". The "Operation" tab is active, displaying a GraphQL mutation:

```
1 mutation($deleteUserId: Int!){
2   deleteUser(id: $deleteUserId){
3     id
4   }
5 }
```

The "Variables" tab is also active, showing the input variables:

```
1 {
2   "deleteUserId": 2
3 }
```

The "Response" tab shows the JSON response from the server:

```
{
  "data": {
    "deleteUser": {
      "id": 2
    }
  }
}
```

At the bottom of the interface, the system tray shows the time as 12:53 pm on 15/03/2025.

The screenshot shows the GraphQL Studio interface with a tab titled "Unnamed". The "Operation" tab is active, displaying a GraphQL query:

```
1 query{
2   users {
3     id
4     name
5     email
6   }
7 }
```

The "Variables" tab is also active, showing the input variables:

```
1 {
2   "deleteUserId": 2
3 }
```

The "Response" tab shows the JSON response from the server:

```
{
  "data": {
    "users": [
      {
        "id": 1,
        "name": "Jericho Update",
        "email": "jerichoupdate@gmail.com"
      }
    ]
  }
}
```

POST CREATE:

The screenshot shows the GraphQL Studio interface for a POST CREATE query. The 'Operation' tab on the left contains the following query:

```
1 mutation($title: String!, $content: String!){
2   createPost(title: $title,content: $content){
3     id
4     title
5     content
6   }
7 }
```

The 'Response' tab on the right shows the JSON response:

```
{
  "data": {
    "createPost": {
      "id": 1,
      "title": "testing",
      "content": "testing"
    }
  }
}
```

Below the query editor, the 'Variables' tab shows the input variables:

```
1 {
2   "title":"testing",
3   "content":"testing"
4 }
```

The status bar at the top right of the response tab indicates a 200 status code, 181ms execution time, and 71B response size. A '+ Add files' button is visible at the bottom left.

POST READ:

The screenshot shows the GraphQL Studio interface for a POST READ query. The 'Operation' tab on the left contains the following query:

```
1 query{
2   posts{
3     id
4     content
5     title
6   }
7 }
```

The 'Response' tab on the right shows the JSON response:

```
{
  "data": {
    "posts": [
      {
        "id": 1,
        "content": "testing",
        "title": "testing"
      }
    ]
  }
}
```

Below the query editor, the 'Variables' tab shows the input variables:

```
1 {
2   "title":"testing",
3   "content":"testing"
4 }
```

The status bar at the top right of the response tab indicates a 200 status code, 37.0ms execution time, and 68B response size. A '+ Add files' button is visible at the bottom left.

POST UPDATE:

Operation

```
1 mutation($updatePostId: Int!, $title: String!, $content: String!) {
2   updatePost(id: $updatePostId, title: $title, content: $content) {
3     id
4     title
5     content
6   }
7 }
```

Run

Response

```
{
  "data": {
    "updatePost": {
      "id": 1,
      "title": "testing
      success",
      "content": "testing
      success"
    }
  }
}
```

200 | 49.0ms | 87

Variables

Headers

Pre-Operation Script

Post-Operation Script

JSON

```
1 {
2   "title": "testing success",
3   "content": "testing success",
4   "updatePostId": 1
5 }
```

+ Add files

Operation

```
1 query{
2   posts{
3     content
4     id
5     title
6   }
7 }
```

Run

Response

```
{
  "data": {
    "posts": [
      {
        "content": "testing
        success",
        "id": 1,
        "title": "testing
        success"
      }
    ]
  }
}
```

200 | 29.0ms | 84

Variables

Headers

Pre-Operation Script

Post-Operation Script

JSON

```
1 {
2   "title": "testing success",
3   "content": "testing success",
4   "updatePostId": 1
5 }
```

+ Add files

POST DELETE:

Operation

Run

```
1 mutation($deletePostId: Int!){
2   deletePost(id: $deletePostId){
3     id
4   }
5 }
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

JSON

```
1 {
2   "deletePostId": 1
3 }
```

+ Add files

Response

200 | 61.0ms | 33B

```
{
  "data": {
    "deletePost": {
      "id": 1
    }
  }
}
```

Operation

Run

```
1 query{
2   posts{
3     content
4     id
5     title
6   }
7 }
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

JSON

```
1 {
2   "deletePostId": 1
3 }
```

+ Add files

Response

200 | 12.0ms | 22B

```
{
  "data": {
    "posts": []
  }
}
```

- What do database migrations do and why are they useful?

As I gathered, database migrations are essentially known as a “snapshot” in that it reflects your database state at that point in time. Whenever there is anything needed to change or add to the database, each migration builds over the previous. Migrations are helpful as it keeps a record of previous versions of each database. This is particularly useful if you ever need to revert to a previous version of the database.

- How does GraphQL differ from REST for CRUD operations?

Unlike RESTful APIs where multiple endpoints are created for each resource, GraphQL exposes a single endpoint, which helps clients to fetch their respective data as per the requirement. REST, on the other hand, is frequently tied to many endpoints which need to be fixed data, causing discrepancies. GraphQL’s type system also enhances client-server communication by helping both sides reach an agreement regarding the structure of the data exchanged.