

## Post-Services

### Create

The screenshot shows the Apollo Studio interface with a GraphQL mutation operation defined in the 'Operation' tab. The operation is a mutation that creates a new post. The 'Variables' tab shows the input variables for the mutation. The 'Response' tab shows the JSON response from the server.

```
1 mutation($title: String!, $content: String!){
2   createPost(title: $title, content: $content){
3     id
4     title
5     content
6   }
7 }
8
```

```
1 {
2   "title": "new assignment",
3   "content": "I did my assignment last Sunday."
4 }
```

```
{
  "data": {
    "createPost": {
      "id": 1,
      "title": "new assignment",
      "content": "I did my assignment last Sunday."
    }
  }
}
```

### Read

The screenshot shows the Apollo Studio interface with a GraphQL query operation defined in the 'Operation' tab. The operation is a query that retrieves a list of posts. The 'Variables' tab shows the input variables for the query. The 'Response' tab shows the JSON response from the server.

```
1 query{
2   posts {
3     id
4     title
5     content
6   }
7 }
8
```

```
1 {
2   "title": "new assignment",
3   "content": "I did my assignment last Sunday."
4 }
```

```
{
  "data": {
    "posts": [
      {
        "id": 1,
        "title": "new assignment",
        "content": "I did my assignment last Sunday."
      }
    ]
  }
}
```

### Update

studio.apollographql.com/sandbox/explorer

SANDBOX • http://localhost:4002/ Publish Log in

Unnamed × +

Documentation

Root > Query

Query

Fields

- post(\_): Post
- posts: [Post!]!

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 }
```

Response

```
{
  "data": {
    "updatePost": {
      "id": 1,
      "title": "bagong assignment",
      "content": "gibuhah nako akong assignment gabii."
    }
  }
}
```

Variables

```
1 {
2   "title": "bagong assignment",
3   "content": "gibuhah nako akong assignment gabii.",
4   "updatePostId": 1
}
```

Headers Pre-Operation Script Post-Operation Script JSON

+ Add files

27°C Partly cloudy 10:01 pm 16/03/2025

## Delete

studio.apollographql.com/sandbox/explorer

SANDBOX • http://localhost:4002/ Publish Log in

Unnamed × +

Documentation

Root > Query

Query

Fields

- post(\_): Post
- posts: [Post!]!

Operation

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

Response

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

Variables

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

Headers Pre-Operation Script Post-Operation Script JSON

+ Add files

27°C Partly cloudy 10:04 pm 16/03/2025

## User-Services

### Create

The screenshot shows the Apollo Studio interface with a GraphQL mutation executed successfully. The left sidebar displays the 'Query' section with fields for 'user' and 'users'. The main editor shows the following mutation:

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

The 'Variables' tab shows the input variables:

```
1 {
2   "name": "Maya",
3   "email": "mayamurakami@gmail.com"
4 }
```

The 'Response' tab shows the JSON output:

```
{
  "data": {
    "createUser": {
      "id": 1,
      "name": "Maya",
      "email": "mayamurakami@gmail.com"
    }
  }
}
```

The status bar at the bottom indicates a 200 status code, 126ms response time, and 808 bytes of data.

### Read

The screenshot shows the Apollo Studio interface with a GraphQL query executed successfully. The left sidebar displays the 'Query' section with fields for 'user' and 'users'. The main editor shows the following query:

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

The 'Variables' tab shows the input variables:

```
1 {
2   "name": "Maya",
3   "email": "mayamurakami@gmail.com"
4 }
```

The 'Response' tab shows the JSON output:

```
{
  "data": {
    "users": [
      {
        "id": 1,
        "name": "Maya",
        "email": "mayamurakami@gmail.com"
      }
    ]
  }
}
```

The status bar at the bottom indicates a 200 status code, 21.0ms response time, and 77B of data.

### Update

studio.apollographql.com/sandbox/explorer

SANDBOX http://localhost:4001/ Publish Log in

Documentation

Root > Query

Query

Fields

- user(:): User
- users: [User!]!

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 }
```

Response

```
{
  "data": {
    "updateUser": {
      "id": 1,
      "name": "MARIE",
      "email": "MARIEmurakami@gmail.com"
    }
  }
}
```

Variables

```
1 {
2   "name": "MARIE",
3   "email": "MARIEmurakami@gmail.com"
4   "updateUserId": 1
5 }
```

Headers Pre-Operation Script Post-Operation Script

+ Add files

27°C Partly cloudy 10:17 pm 16/03/2025

## Delete

studio.apollographql.com/sandbox/explorer

SANDBOX http://localhost:4001/ Publish Log in

Documentation

Root > Query

Query

Fields

- user(:): User
- users: [User!]!

Operation

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

Response

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

Variables

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

Headers Pre-Operation Script Post-Operation Script

+ Add files

27°C Partly cloudy 10:18 pm 16/03/2025

## Reflection

- What do database migrations do and why are they useful?
  - Database migrations manage changes to a database schema over time, such as adding or modifying tables and columns. They are useful for ensuring consistency between the database structure and application code, enabling seamless updates and collaboration across development environments.
- How does GraphQL differ from REST for CRUD operations?
  - GraphQL allows clients to request exactly the data they need in a single query, reducing over-fetching or under-fetching of data, whereas REST typically requires multiple endpoints for different resources, leading to fixed responses. Additionally, GraphQL uses a single endpoint for all operations (queries, mutations, and subscriptions), while REST relies on separate HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.