**Union in GraphQL with SpringBoot 3 – 2025**

When you define a union type, you declare which object types are included in the union:

union Media = Book | Movie

With a union, we can define a new type that lists out the different possible object types it can resolve to. Unlike an interface, a union doesn't enforce a set of common fields all of the object types need to implement. Imagine a union like a box containing some number of different, unrelated items.

Complete Example on Union is given below.

In **schema.graphqls**, define the below.

type Query {

**allCards: [AllCards]**

}

**union AllCards = CreditCard | DebitCard**

type CreditCard {

type: String

bankName: String

}

type DebitCard {

type: String

bankName: String

@Data @NoArgsConstructor @AllArgsConstructor

**public** **class** CreditCard **implements** Card {

**private** String type;

**private** String bankName;

}

}

Java code for the above given below.

**public** **interface** Card {

}

@Data @NoArgsConstructor @AllArgsConstructor

**public** **class** DebitCard **implements** Card {

**private** String type;

**private** String bankName;

}

Controller class is given below

@Controller

**public** **class** CardController {

@QueryMapping

**public** List<Card> **allCards**() {

**return** List.*of*(**new** DebitCard("Debit", "HDFC"), **new** CreditCard("Credit", "HSBC"));

}

}

**Actual GraphQL query** is given below.

**query** AllCards {

    allCards {

        ... **on** CreditCard {

**type**

            bankName

        }

        ... **on** DebitCard {

**type**

            bankName

        }

    }

}

**Response**

{

    "data": {

        "allCards": [

            {

                "type": "Debit",

                "bankName": "HDFC"

            },

            {

                "type": "Credit",

                "bankName": "HSBC"

            }

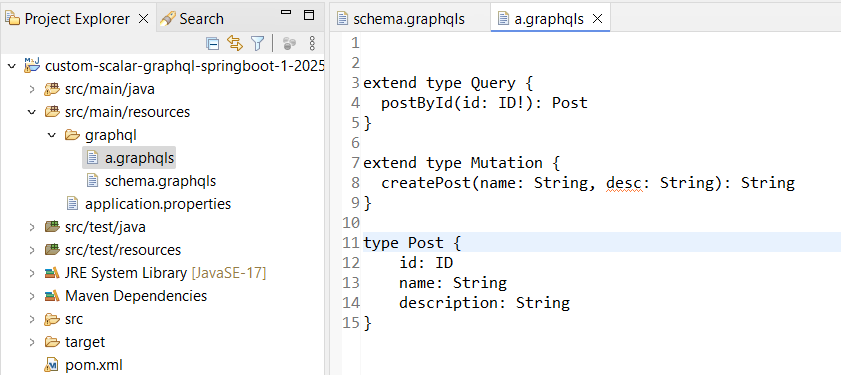
        ]

    }

}

extend type Query and extend type Mutation in GraphQL

This is application when you have multiple **\*.graphqls** files. We must have one **schema.graphqls** file and then based upon the functionality, we can have multiple .graphqls file like **a.graphqls**, **b.graphqls** files. The screenshot is given below.



Let us see the complete example.

We have one schema.graphqls file which is given below.

**type Query {**

**allCards: [AllCards]**

**}**

**union AllCards = CreditCard | DebitCard**

**type CreditCard {**

**type: String**

**bankName: String**

**}**

**type DebitCard {**

**type: String**

**bankName: String**

**}**

Now we create another file called a.graphqls as shown below.

**extend type Query {**

**postById(id: ID!): Post**

**}**

**extend type Mutation {**

**createPost(name: String, desc: String): String**

**}**

**type Post {**

**id: ID**

**name: String**

**description: String**

**}**

Corresponding Controller class is given below.

@Controller

**public** **class** PostController {

@QueryMapping

**public** Post postById(@Argument **long** id) {

System.***out***.println("Received ID: "+id);

Post post = **new** Post();

post.setId(id);

post.setDescription("Some description ...");

post.setName("some name");

**return** post;

}

@MutationMapping

**public** String createPost(@Argument String name, @Argument String desc) {

**return** "Post created successfully ...";

}

}

Actual GraphQL query is given below.

**query PostById {**

**postById(id: "111") {**

**id**

**name**

**description**

**}**

**}**

**How to implement Custom Scalar in GraphQL**

Scalars are equivalent to [primitive data types](https://en.wikipedia.org/wiki/Primitive_data_type) in a programming language. In GraphQL, there are five built-in scalar types:

* **Boolean**, true or false
* **Int**, a signed 32‐bit numeric non‐fractional value
* **Float**, a signed double‐precision fractional values
* **String**, a sequence of UTF‐8 characters
* **ID**, a unique identifier

A scalar simply represents a single value and are the basic building blocks of your schema. The important thing to notice here is that **leaf types do not have fields, so any field that returns a leaf type will not have a selection set**.

[graphql-java-extended-scalars](https://github.com/graphql-java/graphql-java-extended-scalars) adds many more scalars, including the following which are useful in Java based systems:

* Long aka GraphQLLong - a java.lang.Long based scalar
* Short aka GraphQLShort - a java.lang.Short based scalar
* Byte aka GraphQLByte - a java.lang.Byte based scalar
* BigDecimal aka GraphQLBigDecimal - a java.math.BigDecimal based scalar
* BigInteger aka GraphQLBigInteger - a java.math.BigInteger based scalar

**How will you create your own custom scalar in springBoot**

**Use Case**: Create a scalar called Void, it should return null in case of creating object in the server side while implementing Mutation Mapping.

**Step-1**: Define the name of the scalar in schema.graphqls as shown below.

scalar Void

**Step-2**: Create an Implementation class using GraphQLScalarType by overriding **serialize**, **parseValue**, **parseLiteral** methods.

**Step-3**: Create a Config class by returning RuntimeWiringConfigurer and add that Scalar Implementation class.

Complete example is given below.

In **schema.graphqls**, it has been defined as below.

**scalar Void**

**type Query {**

**ping: String @deprecated(reason: "https://stackoverflow.com/questions/59868942/graphql")**

**}**

**type Mutation {**

**createEmp(firstName: String, lastName: String): Void**

**}**

**type Employee {**

**id: ID**

**firstName: String**

**lastName: String**

**}**

GraphQl Scalar **Void implementation class** is given below.

**public** **class** GraphQLVoidScalar {

**public** **static** **final** GraphQLScalarType ***Void*** = GraphQLScalarType

.*newScalar*()

.name("Void")

.description("A custom scalar that represents the null value").coercing(**new** Coercing() {

@Override

**public** Object serialize(Object dataFetcherResult) {

**return** **null**;

}

@Override

**public** Object parseValue(Object input) {

**return** **null**;

}

@Override

**public** Object parseLiteral(Object input) {

**return** **null**;

}

}).build();

}

GraphQL config class is given below.

@Configuration

**public** **class** GraphQlConfig {

@Bean

**public** RuntimeWiringConfigurer runtimeWiringConfigurer() {

**return** wiringBuilder -> wiringBuilder.scalar(GraphQLVoidScalar.***Void***);

}

}

Controller class for the below mutation type is given below.

**type Mutation {**

**createEmp(firstName: String, lastName: String): Void**

**}**

@Controller

**public** **class** EmpController {

@MutationMapping

**public** **void** createEmp(@Argument String firstName, @Argument String lastName) {

System.***out***.println("Received First Name: "+firstName);

System.***out***.println("Received Last Name: "+lastName);

}

}

GraphQL Actual Query is given below.

**mutation CreateEmp {**

**createEmp(firstName: "Ram", lastName: "Shyam")**

**}**

**Response**

**{**

**"data": {**

**"createEmp": null**

**}**

**}**