GraphQL Type System

GraphQL is a strongly typed language. A programming language is known as a strongly typed programming language if each type of data such as integer, character, hexadecimal, packed decimal, etc. is predefined in that language and all constants or variables defined for a program is described with one of the data types.

The [GraphQL](https://www.javatpoint.com/graphql) Type system is used to describe the potential of a GraphQL server. It is used to determine if a query is valid or not. It defines various data types that we can use in a GraphQL application to check the input types of query variables at run time and check their validity.

There are mainly five kinds of named type definitions in GraphQL:

|  |  |  |
| --- | --- | --- |
| **Index** | **Types** | **Explanation** |
| 1. | Scalar | Scalar types represent primitive data types in a GraphQL type system and store only a single value. |
| 2. | Object | It shows what type of objects can be fetched. |
| 3. | Query | It specifies an entry point type to other specific types. |
| 4. | Mutation | It is used to write the query. It specifies entry point for data manipulation (i.e. create, update or delete data). |
| 5. | Enum | Enum type is used where you need the user to pick from a prescribed list of options. |

Scalar Types

Scalar types represent primitive data types in a GraphQL type system. The responses of GraphQL make a hierarchical tree form, and the **leaves on these trees** are GraphQL scalars. The scalar types can store only a single value.

**GraphQL provides the following default scalar types:**

**Int -** The Int scalar type is used to represent a signed 32-bit numeric non-fractional value.

**Float -** The Float scalar type is used to represent signed double-precision fractional values as specified by IEEE 754.

**String -** The String scalar type is used to represent textual data, represented as UTF-8 character sequences. This is mainly used by GraphQL to represent free-form human-readable text.

**Boolean -** The Boolean scalar type is used to represent true or false.

**ID -** The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache.

Syntax

Following is the syntax to define a scalar type:

1. field: data\_type

Example:

1. hello: String

Object Type

The responses of GraphQL make a hierarchical tree form where the **leaves on these trees are GraphQL scalars**, and **intermediate levels** are specified as Objects. It shows the kind of objects that you can fetch.

This is the most common type used in a schema and used to represent a group of fields. It allows nested types where each field inside an object type maps to another type. So, you can say that an object type is made of multiple scalar types or object types.

Syntax:

Following is the syntax to define an object type:

1. type object\_type\_name
2. {
3. field1: data\_type
4. field2:data\_type
5. ....
6. fieldn:data\_type
7. }

Example:

1. --To define an object type--
2. type Employee {
3. emp\_id:ID
4. firstname: String
5. age: Int
6. salary:Float
7. }
8. --To define a GraphQL schema--
9. type Query
10. {
11. emp\_details:[Employee]
12. }

The above example specifies an object data-type Employee. The emp\_details field in the root Query schema will return a list of Employee objects.

Query Type

The Query type is a simple request sent from the client to the GraphQL server. It is used to retrieve data from the GraphQL server. It specifies an entry point type to other specific types. It is one of the root-level types in GraphQL.

Syntax:

Following is the syntax to define a query:

1. type Query {
2. field1: data\_type
3. field2:data\_type
4. field2(param1:data\_type,param2:data\_type,...paramN:data\_type):data\_type
5. }

Example:

1. type Query  {
2. hello: String
3. }

Mutation Type :operations

Mutation type is used for data manipulation such as create, update or delete data. It is one of the root-level data-types in GraphQL that specifies the entry points for data-manipulation operations.

Syntax:

Following is the syntax to define a Mutation type:

1. type Mutation {
2. field1: data\_type
3. field2(param1:data\_type,param2:data\_type,...paramN:data\_type):data\_type
4. }

Example:

If you want to add a new Employee to the predefined object type "Employee" then you have to define a mutation type:

1. type Mutation {
2. addEmployee(firstName: String, lastName: String): Employee
3. }

Enum Type

The GraphQL Enum types are similar to scalar types and also represent leaf values in a GraphQL type system. Enum types are used where you need the user to pick from a prescribed list of options.

Syntax:

Following is the syntax to define an Enum type:

1. type enum\_name{
2. value1
3. value2
4. }

Example:

See how enum types are defined:

1. type Days\_of\_Week{
2. SUNDAY
3. MONDAY
4. TUESDAY
5. WEDNESDAY
6. THURSDAY
7. FRIDAY
8. SATURDAY
9. }

List Type

The GraphQL list type is a special collection type that is used to specify each item in the List. It is a representation of a specific type array of values. List values are serialized as ordered lists according to their item type. The list type is used with a type modifier [] that wraps object types, scalars, and enums.

Syntax:

Following is the syntax to define a list type -

1. field:[data\_type]

Example:

An example of the list type named "todo":

1. type Query {
2. todo: [String]
3. }

Non-Nullable Type

By default, all GraphQL types are nullable and set to null because the null value is a valid response for all of the GraphQL types. GraphQL types can either return a value of the specified type or can have no value.

A field must be defined with an exclamation mark (!) to define a non-nullable type. This exclamation mark is used to specify a field that uses a Non-Null type such as name: String!.

Syntax:

Following is the syntax to define a non-nullable field:

1. field:data\_type!

Example:

1. type Employee {
2. emp\_id:ID!
3. firstName:String
4. lastName:String
5. }