## Complexity

> warning **Warning** This chapter applies only to the code first approach.

Query complexity allows you to define how complex certain fields are, and to restrict queries with a **maximum complexity**. The idea is to define how complex each field is by using a simple number. A common default is to give each field a complexity of `1`. In addition, the complexity calculation of a GraphQL query can be customized with so-called complexity estimators. A complexity estimator is a simple function that calculates the complexity for a field. You can add any number of complexity estimators to the rule, which are then executed one after another. The first estimator that returns a numeric complexity value determines the complexity for that field.

The `@nestjs/graphql` package integrates very well with tools like graphql-query-complexity that provides a cost analysis-based solution. With this library, you can reject queries to your GraphQL server that are deemed too costly to execute.

### Installation

To begin using it, we first install the required dependency.

```
$ npm install --save graphql-query-complexity
```

### Getting started

Once the installation process is complete, we can define the `ComplexityPlugin` class:

```
import { GraphQLSchemaHost } from "@nestjs/graphql";
import { Plugin } from "@nestjs/apollo";
import {
  ApolloServerPlugin,
  GraphQLRequestListener,
} from 'apollo-server-plugin-base';
import { GraphQLError } from 'graphql';
import {
  fieldExtensionsEstimator,
  getComplexity,
  simpleEstimator,
} from 'graphql-query-complexity';

@Plugin()
export class ComplexityPlugin implements ApolloServerPlugin {
  constructor(private gqlSchemaHost: GraphQLSchemaHost) {}

  async requestDidStart(): Promise<GraphQLRequestListener> {
    const maxComplexity = 20;
    const { schema } = this.gqlSchemaHost;

    return {
```

```
      async didResolveOperation({ request, document }) {
        const complexity = getComplexity({
          schema,
          operationName: request.operationName,
          query: document,
          variables: request.variables,
          estimators: [
            fieldExtensionsEstimator(),
            simpleEstimator({ defaultComplexity: 1 }),
          ],
        });
        if (complexity > maxComplexity) {
          throw new GraphQLError(
            `Query is too complex: ${complexity}. Maximum allowed
complexity: ${maxComplexity}`,
          );
        }
        console.log('Query Complexity:', complexity);
      },
    };
  }
}
```

For demonstration purposes, we specified the maximum allowed complexity as `20`. In the example above, we used 2 estimators, the `simpleEstimator` and the `fieldExtensionsEstimator`.

- `simpleEstimator`: the simple estimator returns a fixed complexity for each field
- `fieldExtensionsEstimator`: the field extensions estimator extracts the complexity value for each field of your schema

> info **Hint** Remember to add this class to the providers array in any module.

**Field-level complexity**

With this plugin in place, we can now define the complexity for any field by specifying the `complexity` property in the options object passed into the `@Field()` decorator, as follows:

```
@Field({ complexity: 3 })
title: string;
```

Alternatively, you can define the estimator function:

```
@Field({ complexity: (options: ComplexityEstimatorArgs) => ... })
title: string;
```

**Query/Mutation-level complexity**

In addition, `@Query()` and `@Mutation()` decorators may have a `complexity` property specified like so:

```
@Query({ complexity: (options: ComplexityEstimatorArgs) =>
options.args.count * options.childComplexity })
items(@Args('count') count: number) {
  return this.itemsService.getItems({ count });
}
```