Enterprise Application Development

# LAB 03 - GRAPHQL

David O'Neill

C15737551

## Table of Contents

***Question One*** ..... ***2***

***Question Two*** ..... ***4***

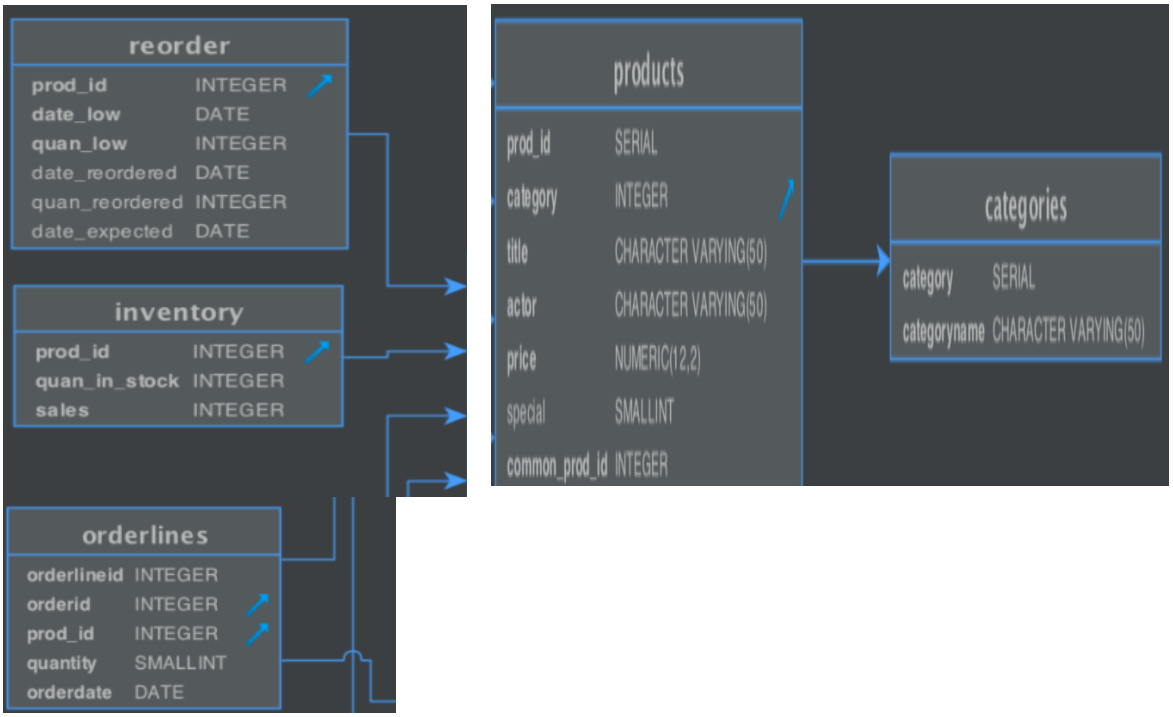
***Question 3:*** ..... ***6***

***Question 4:*** ..... ***7***

## Question One

| | |
|---|---|
| Using using graphql-yoga and the ERD below, construct a graphql schema using <u>any four relations</u> of your choice having the relationships depicted. | 25 Marks |

For this I chose these tables and relations:




This is the GraphQL schema that I created:

```
type Products {
  id: ID! @unique
  category: [Categories!]!
  title: String!
  actor: String!
  price: Int!
  special: Int!
  common_prod_id: Int
}

type Reorder {
  prod_id: [Products!]!
  date_low: DateTime!
  quan_low: Int!
  date_reordered: DateTime!
  quan_reordered: Int!
  date_expected: DateTime!
}

type Inventory {
  prod_id: [Products!]!
  quan_in_stock: Int!
  sales: Int!
}

type Categories {
  id: ID! @unique
  categoryname: String!
}
```

```
scalar DateTime

type Query{
   getProducts: [Products!]!
   getReorder: [Reorder!]!
   getInventory: [Inventory!]!
   getCategories: [Categories!]!
}

type Mutation {
  createProducts(
    title: String!
    category: ID
    actor: String!
    price: Int!
    special: Int!
    common_prod_id: Int
  ):Products

  createReorder (
    prod_id: ID
    date_low: DateTime!
    quan_low: Int!
    date_reordered: DateTime!
    quan_reordered: Int!
    date_expected: DateTime!
  ):Reorder

  createInventory (
    prod_id: ID
    quan_in_stock: Int!
    sales: Int!
  ):Inventory

  createCategories (
    categoryname: String!
  ):Categories
}
```

```
type Products {
  id: ID!
  category: Categories
  title: String!
  actor: String!
  price: Int!
  special: Int!
  common_prod_id: Int
}

type Reorder {
    prod_id: Products
    date_low: DateTime!
    quan_low: Int!
    date_reordered: DateTime!
    quan_reordered: Int!
    date_expected: DateTime!
}

type Inventory {
    prod_id: Products
    quan_in_stock: Int!
    sales: Int!
}

type Categories {
  id: ID!
  categoryname: String!
}
```

## Question Two

| | | |
|---|---|---|
| **2** | Build a GraphQL query resolver which returns some set of the the attributes from a single database relation. | 10 Marks |

```
const resolvers = {
    Query: {
        getProducts(root, args, context) {
            return context.prisma.productses()
        },
        getReorder(root, args, context) {
            return context.prisma.reorders()
        },

        getInventory(root, args, context) {
            return context.prisma.inventories()
        },

        getCategories(root, args, context) {
            return context.prisma.categorieses()
        },
    },
```

```
# Write your query or mutation here
query {
  getProducts {
    id
    title
    category {
      categoryname
    }
    actor
    price
    special
    common_prod_id
  }
}
```

QUERY VARIABLES HTTP HEADERS

```
{
  "data": {
    "getProducts": [
      {
        "id": "cjtrhct54008k07287saf7lri",
        "title": "Movie1",
        "category": null,
        "actor": "Actor1",
        "price": 25,
        "special": 20,
        "common_prod_id": null
      },
      {
        "id": "cjtrhdz0b008r072865ti8eo6",
        "title": "Movie1",
        "category": null,
        "actor": "Actor1",
        "price": 25,
        "special": 20,
        "common_prod_id": null
```

```
# Write your query or mutation here
query {
  getReorder {
    date_low
    date_expected
    date_reordered
    quan_low
    quan_reordered
  }
}
```

```
{
  "data": {
    "getReorder": [
      {
        "date_low": "2019-03-25T18:15:20.000Z",
        "date_expected": "2019-03-25T18:15:20.000Z",
        "date_reordered": "2019-03-25T18:15:20.000Z",
        "quan_low": 2,
        "quan_reordered": 2
      },
```

```
# Write your query or mutation here
query {
  getInventory {
    quan_in_stock
    sales
  }
}
```

```
{
  "data": {
    "getInventory": [
      {
        "quan_in_stock": 2,
        "sales": 2
      },
      {
        "quan_in_stock": 2,
        "sales": 2
      },
      {
        "quan_in_stock": 5,
        "sales": 2
      },
```

DOCS

SCHEMA

```
# Write your query or mutation here
query {
  getCategories {
    id
    categoryname
  }
}
```

```
{
  "data": {
    "getCategories": [
      {
        "id": "cjtrgsdkc00850728pd0qcep8",
        "categoryname": "Movies"
      }
    ]
  }
}
```

## Question Three:

| | | |
|---|---|---|
| **3** | Build a GraphQL query resolver which returns the attributes from 3 joined database relations having 2 levels of nesting in the resultant output<br><br>Briefly, describe an application of the query you have chosen to write as a comment in your resolver code | 20 Marks |

Below is the code for part 3: Nested Query. This query gets the category, product and inventory information from reordered stock.
This query gets the category and product information from reordered stock. An employee might want to view this information.

```
getReorder(root, args, context) {
    return context.prisma.reorders()
},
```

```
Products: {
    category(root, args, context) {
        return context.prisma.products({
            id: root.id
        }).category()
    }
},
Inventory: {
    products(root, args, context) {
        return context.prisma.inventory({
            id: root.id
        }).products()
    }
},
Reorder: {
    products(root, args, context) {
        return context.prisma.reorders({
            id: root.id
        }).products()

    }
}
```

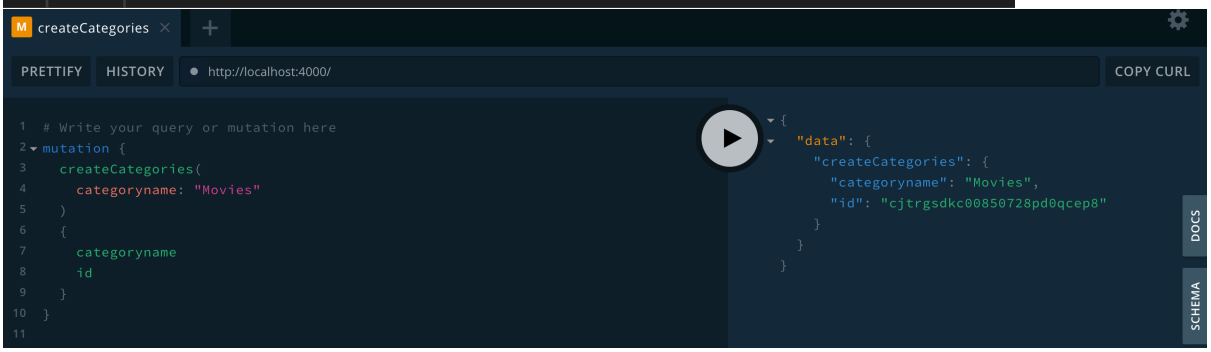
## Question Four:

| | | |
|---|---|---|
| **4** | Create a mutation resolver to add data the database. Your mutation should update at least two relations (of your choice)<br><br>Briefly, describe an application of the query you have chosen to write as a comment in your resolver code | 20 Marks |

Below is some of the migrations code. An employee could want to create more inventory and add it to a reorder.

```
createInventory(root, args, context) {
    return context.prisma.createInventory({
        prod_id: {
            connect: { id: args.prod_id }
        },
        quan_in_stock: args.quan_in_stock,
        sales: args.sales
    })
},

createReorder(root, args, context) {
    return context.prisma.createReorder({
            prod_id: {
                connect: { id: args.prod_id }
            },
            date_low: args.date_low,
            quan_low: args.quan_low,
            date_reordered: args.date_reordered,
            quan_reordered: args.quan_reordered,
            date_expected: args.date_expected
        }
```



PRETTIFY HISTORY http://localhost:4000/ COPY CURL

```
# Write your query or mutation here
mutation {
  createProducts(
    title: "Movie2"
    actor: "Actor2"
    price: 25
    special: 20
    category: "cjtrgsdkc00850728pd0qcep8"
  )
  {
    id
    title
    actor
    price
    special
    }
```

```
{
  "data": {
    "createProducts": {
      "id": "cjtrhp4e9009g0728uefkjlkb",
      "title": "Movie2",
      "actor": "Actor2",
      "price": 25,
      "special": 20
    }
  }
}
```

DOCS SCHEMA

```
# Write your query or mutation here
mutation {
  createInventory(
    prod_id: "cjtri5dzf00bm0728wd1ackyx"
    quan_in_stock:5
    sales: 2
  )
  {
    quan_in_stock
    sales
    }

    }
```

```
{
  "data": {
    "createInventory": {
      "quan_in_stock": 5,
      "sales": 2
    }
  }
}
```

M createReorder

PRETTIFY HISTORY http://localhost:4000/ COPY CURL

```
# Write your query or mutation here
mutation {
  createReorder(
    prod_id: "cjtri5dzf00bm0728wd1ackyx"
    date_low: "2019-03-25T18:15:20.000Z"
    quan_low: 2
    date_reordered: "2019-03-25T18:15:20.000Z"
    quan_reordered: 2
    date_expected: "2019-03-25T18:15:20.000Z"
  )
  {
    date_low
    quan_low
    date_reordered
    quan_reordered
    date_expected
    }
```

```
{
  "data": {
    "createReorder": {
      "date_low": "2019-03-25T18:15:20.000Z",
      "quan_low": 2,
      "date_reordered": "2019-03-25T18:15:20.000Z",
      "quan_reordered": 2,
      "date_expected": "2019-03-25T18:15:20.000Z"
    }
  }
}
```

DOCS SCHEMA

| | | |
|---|---|---|
| **5** | Set up a running GraphQLServer from the graphql-yoga library to test and demonstrate your resolver queries and mutations you implemented in sections 2-4 above | 25 Marks |

```
const { GraphQLServer } = require('graphql-yoga')
```

```
const server = new GraphQLServer({
    typeDefs: './schema.graphql',
    resolvers,
    context: {
        prisma
    },
})
server.start(() => console.log('Server is running on http://localhost:4000'))
```