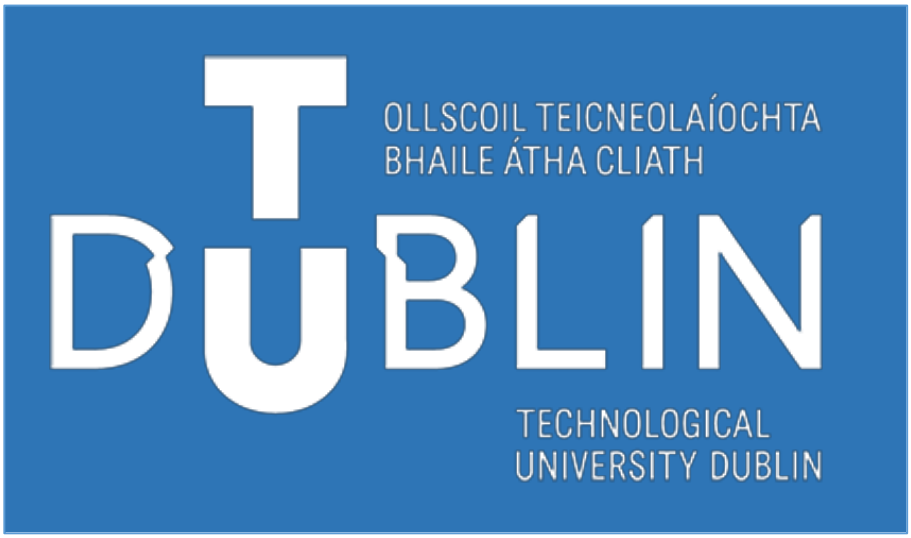


LAB 3 REPORT

Enterprise App Development

- Implemented a GraphQL interface to a relational DB using a GraphQL schema builder and tested it within a graphical client

Eric Strong
C15708709@mydit.ie
DT211C/4

Contents

Video Demo of Lab

I made a video demonstration of the entire lab. It can be viewed here:
https://drive.google.com/open?id=12FOgJOzJKbwR8Jx0JHnopj1_VgO0pb1v
All of the work below is included in a walk-through demonstration.

Setting Up

Installation of Docker

```
eric:$docker --version
Docker version 18.09.0, build 4d60db4
```

Docker Compose

```
eric:$ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
```

Docker machine

```
eric:$docker-machine --version
docker-machine version 0.16.0, build 702c267f
```

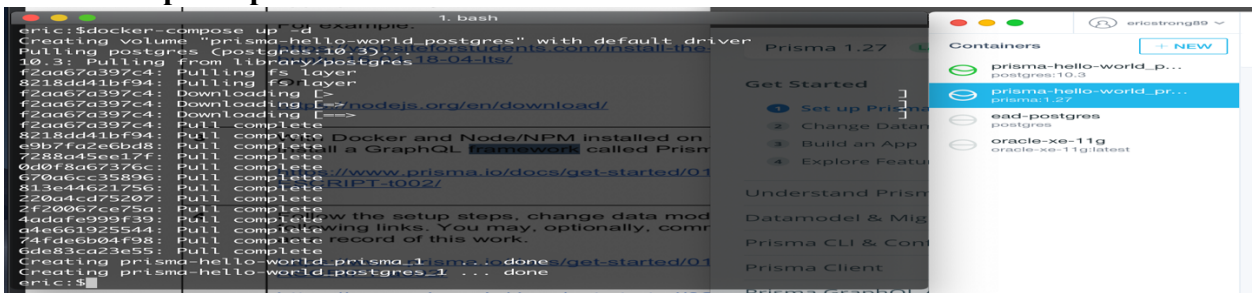
Node Project
Npm init

Install libraries
Npm install express –save
Npm install graphql --save

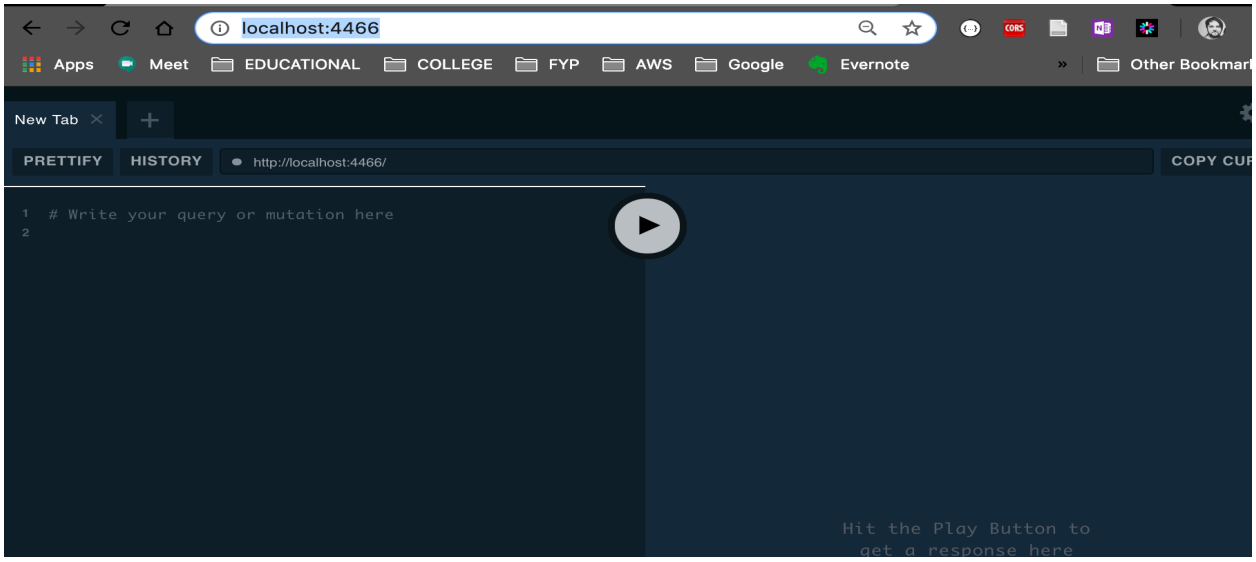
Prisma Framework
npm install -g prisma

creating the docker-compose.yml file
paste the yml config for postgres into docker-compose.yml

docker-compose up -d



View the Postgres DB and Prisma server via localhost:4466



Bootstrap the config files:
prisma init --endpoint http://localhost:4466

```
eric:$prisma init --endpoint http://localhost:4466

Created 2 new files:

  prisma.yml            Prisma service definition
  datamodel.prisma      GraphQL SDL-based datamodel (foundation for database)

Next steps:

  1. Deploy your Prisma service: prisma deploy
eric:$
```

Deploy the Prisma Model
prisma deploy

```
eric:$prisma deploy
Creating stage default for service default ✓
Deploying service `default` to stage `default` to server `local` 820ms

Changes:

  User (Type)
  + Created type `User`
  + Created field `id` of type `ID!`
  + Created field `name` of type `String!`
  + Created field `updatedAt` of type `DateTime!`
  + Created field `createdAt` of type `DateTime!`

Applying changes 1.1s

Your Prisma GraphQL database endpoint is live:

  HTTP:  http://localhost:4466
  WS:    ws://localhost:4466
```

Generating a client – add code to the prisma.yml file
generate:
  - generator: javascript-client
    output: ./generated/prisma-client/
prisma generate

create our index.js
add dependency to package.json
npm install --save prisma-client-lib
copy code to the index.js

```
const { prisma } = require('./generated/prisma-client')

// A `main` function so that we can use async/await
async function main() {

  // Create a new user called `Alice`
  const newUser = await prisma.createUser({ name: 'Alice' })
  console.log(`Created new user: ${newUser.name} (ID: ${newUser.id})`)

  // Read all users from the database and print them to the console
  const allUsers = await prisma.users()
  console.log(allUsers)

  //fetch list
  const usersCalledAlice = await prisma
  .users({
    where: {
      name: 'Alice'
    }
  })
  console.log('-----------' + usersCalledAlice);

  //update users
  const updatedUser = await prisma
  .updateUser({
    where: { id: 'cjst9lekp002s0a92a9w1p3r0' },
    data: { name: 'Bob' }
  })
  console.log('-------------' + updatedUser)


  //delete user
  const deletedUser = await prisma
  .deleteUser({ id: 'cjst9n8v9003d0a92u8s3nn8f' })

}



main().catch(e => console.error(e))
```

Run the server
Npm run dev (I already had a dev script using nodemon)

The above example gets you set up with Prisma and GraphQL. This is the sample hello world project.

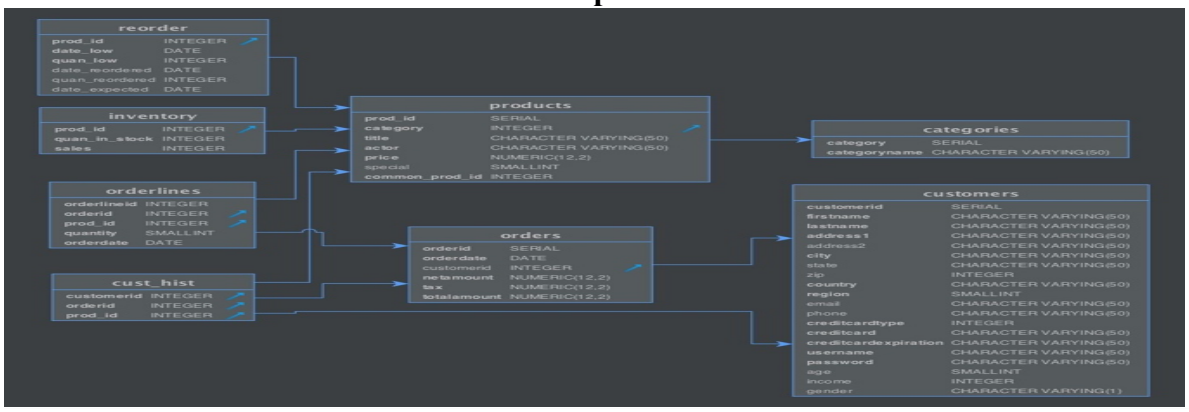
Problem Sets

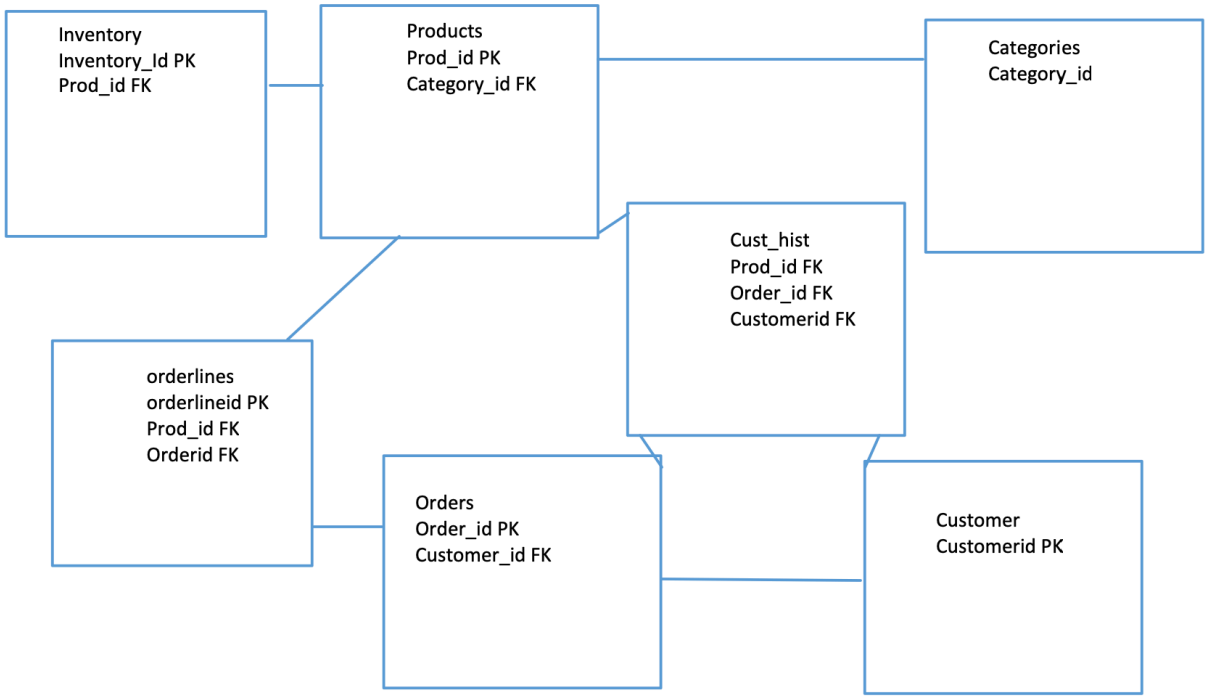
Part 1

Using using graphql-yoga and the ERD below, construct a graphql schema using any four relations of your choice having the relationships depicted.

npm install --save graphql-yoga
npm install --save graphql
npm install –save prisma-client-lib

Next I decided what entities and relationships to model. I chose the 7 below:





Relations of choice with depicted relationships are below

- Customers – joins **cust_hist, Orders**
- Orders – joins **cust_hist, customer**

- Cust_hist – **joins orders, customer, products**
- Products – joins **categories, cust_hist, orderlines, inventory**
- Categories – joins **products**
- Inventory – joins **products**
- Orderlines – joins **products, orders**

GraphQL Datamodel code:

Prisma init (added the following to datamodel.prisma)

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

type Customer{
  id: ID! @unique
  firstname: String!
  lastname: String!
  address1: String!
  address2: String!
  city: String!
  state: String!
  zip: String!
  county: String!
  region: String!
  email: String! @unique
  phone: String!
  creditcardtype: String
  creditcard: String
  creditcardexpiration: String
  username: String! @unique
  password: String!
  age: Int
  income: Float
  gender: String!
}

type Products{
  id: ID! @unique
  title: String
  actor: String
  price: Float
  special: Float
  common_prod_id: Int
  category: Category @relation(name:"ProductCat")
}
```

```
type Orders{
  id: ID! @unique
  orderdate: String
  netamount: Float!
  tax: Float!
  totalamount: Float!
  customer: Customer
}


type Cust_Hist{
  id: ID! @unique
  customer: Customer
  order: Orders
  product: Products
}
```

Prisma deploy
Prisma generate

GraphQL Schema code:
In the schema.graphql file:

```
scalar DateTime

type Query{
   getCategoriesEndpoint: [Category!]!
   getCategoriesWhereEndpoint(catId: ID!): [Category!]!
   getCustomersEndpoint: [Customer!]!
   getProductsEndpoint: [Products!]!
   getOrdersEndpoint: [Orders!]!
   getCust_HistsEndpoint: [Cust_Hist!]!
   getProductsByCategoryEndpoint(categoryId: ID!): Products!

}

type Mutation{
  createCategoryEndpoint(
    categoryname: String!
  ): Category

  createCustomerEndpoint(
    firstname: String!,
    lastname: String!,
    address1: String!,
    address2: String!,
    city: String!,
```

```
    state: String!,
    zip: String!,
    county: String!,
    region: String!,
    email: String!,
    phone: String! ,
    creditcardtype: String,
    creditcard: String,
    creditcardexpiration: DateTime,
    username: String!,
    password: String!,
    age: Int,
    income: Float,
    gender: String!
  ): Customer

  createProductsEndpoint(title: String!,actor: String,price: Float!,special:
Float,common_prod_id: Int,categoryId: ID): Products


  createOrdersEndpoint(
    orderdate: DateTime,
    netamount: Float!,
    tax: Float!,
    totalamount: Float!,
    customerId: ID!
  ): Orders

  createCust_HistEndpoint(
  customerId: ID!,
  orderId: ID!,
  productId: ID!
  ): Cust_Hist

}

type Category{
  id: ID!
  categoryname: String!
}

type Customer{
  id: ID!
  firstname: String!
  lastname: String!
  address1: String!
  address2: String!
  city: String!
```

```
  state: String!
  zip: String!
  county: String!
  region: String!
  email: String!
  phone: String!
  creditcardtype: String
  creditcard: String
  creditcardexpiration: DateTime
  username: String!
  password: String!
  age: Int
  income: Float
  gender: String!
}

type Products{
  id: ID!
  title: String!
  actor: String
  price: Float!
  special: Float
  common_prod_id: Int
  category: Category

}


type Orders{
  id: ID!
  orderdate: DateTime
  netamount: Float!
  tax: Float!
  totalamount: Float!
  customer: Customer
}

type Cust_Hist{
  id: ID!
  customer: Customer
  order: Orders
  product: Products
}
```

Part 2

Build a GraphQL query resolver which returns some set of the attributes from a single database relation.

```
Query: {
    /*
        part 1
        Single Table Queries
    */
    getCategoriesEndpoint(root, args, context) {
      console.log('-------- SELECTING CATEGORIES ------------')
      return context.prisma.categories()
    },

    getCategoriesWhereEndpoint(root, args, context) {
      console.log('-------- SELECTING CATEGORIES WHERE ID = ' + args.catId + ' -------
-----')
      return context.prisma.categories({
        where: {
          id: args.catId
        }
      })
    },

    getCustomersEndpoint(root, args, context) {
      console.log('-------- SELECTING CUSTOMERS ------------')
      return context.prisma.customers()
    },

    getProductsEndpoint(root, args, context) {
      console.log('-------- SELECTING Products ------------')
      return context.prisma.productses()
    },

    getOrdersEndpoint(root, args, context) {
      console.log('-------- SELECTING ORDERS ------------')
      return context.prisma.orderses()
    },

    getCust_HistsEndpoint(root, args, context) {
      console.log('-------- SELECTING CUST_HIST ------------')
      return context.prisma.cust_Hists()
    },
}
```

Screenshots

Customers relation

```
query{
  getCustomersEndpoint{
    id
    firstname
    username
    creditcardexpiration

  }
}
```


```
{
  "data": {
    "getCustomersEndpoint": [
      {
        "id": "cjsuytptg00bq0a04303626ny",
        "firstname": "Daveo1",
        "username": "db69o1",
        "creditcardexpiration": "02-03-2020"
      },
      {
        "id": "cjsuz8lrb00ft0a04zqa1p2ts",
        "firstname": "Eric",
        "username": "es89",
        "creditcardexpiration": "02-03-2020"
      },
      {
        "id": "cjsuzbkaj00ga0a04fyao27il",
        "firstname": "Eriddc",
        "username": "es8d9",
        "creditcardexpiration": "02-03-2020"
      }
    ]
  }
}
```


Categories relation

```
query{
  getCategoriesEndpoint{
    id
    categoryname
  }
}
```


```
{
  "data": {
    "getCategoriesEndpoint": [
      {
        "id": "cjsuyp94f00as0a04g95dq2m6",
        "categoryname": "newnew"
      },
      {
        "id": "cjsuyqi5r00b50a046mwr0v69",
        "categoryname": "newnew"
      }
    ]
  }
}
```


products relation

```
query{
  getProductsEndpoint{
    id
    title
    price


  }
}
```


```
{
  "data": {
    "getProductsEndpoint": [
      {
        "id": "cjsuypij100az0a041qui05yv",
        "title": "drumkit2",
        "price": 650
      },
      {
        "id": "cjsuyqq2w00bb0a04a2z2dqu2",
        "title": "drumkit2",
        "price": 650
      },
      {
        "id": "cjsuyqy5400bj0a04t68mswh9",
        "title": "drumkit2",
        "price": 650
      },
      {
        "id": "cjsuyxuds00cc0a04i9rhjtez",
        "title": "drumkit2",
        "price": 650
      }
    ]
  }
}
```


Orders relation

```
query{
  getOrdersEndpoint{
    id
    totalamount


  }
}
```

```
{
  "data": {
    "getOrdersEndpoint": [
      {
        "id": "cjsuyu19c00bw0a04ozww7i7u",
        "totalamount": 887.5
      },
      {
        "id": "cjsuyu7dd00c40a04pp42zyz5",
        "totalamount": 887.5
      }
    ]
  }
}
```

Part 3

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

Application example:
The query in this case would be a good example of a many to many join. We can easily get data from 3 or 4 tables. In the case of an application, we could get all orders by a customer and all of the products on that order also. We could also return all of the categories each of those products belong to.

Return:
All Orders of Customer “eric strong”
All Products that belong to that order
All of the categories for each of those products

```
query{
  getCust_HistsEndpoint{
    id
    product{
      id
      title
      category{
        categoryname
      }
    }
    customer{
      id
      firstname
      address1
    }
    order{
      id
      totalamount
    }
  }
}
```


```
"data": {
  "getCust_HistsEndpoint": [
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {↔},
    {
      "id": "cjsvn5tlp00k90a04g2ww89gu",
      "product": {
        "id": "cjsvmy8dy00i80a040t71hlza",
        "title": "Python 101",
        "category": {
          "categoryname": "ComputerScienceBooks"
        }
      },
      "customer": {
        "id": "cjsvmuxu500hv0a0475f0reww",
        "firstname": "Eric",
        "address1": "11 village green, "
      },
      "order": {
        "id": "cjsvn0nwl00ii0a04il07uv6y",
        "totalamount": 12.49
      }
    }
  ]
}
}
```


```
// this gets the products.category value defined in the schema
Products: {
  category(root, args, context) {
    return context.prisma.products({
      id: root.id
    }).category()
  }
},

Orders: {
  customer(root, args, context) {
    return context.prisma.orders({
      id: root.id
    }).customer()
  }
},

Cust_Hist: {
  customer(root, args, context) {
    return context.prisma.cust_Hist({
      id: root.id
    }).customer()
  },
  order(root, args, context) {
    return context.prisma.cust_Hist({
      id: root.id
    }).order()
```

```
    },
    product(root, args, context) {
      return context.prisma.cust_Hist({
        id: root.id
      }).product()
    }

  }
} // end resolvers
```

Part 4

Create a mutation resolver to add a new order to the database. Your mutation should update at least two relations
Briefly, describe an application of the query you have chosen to write as a comment in your resolver code

```
Application example:
The query in this case would be a good example of a transactional relational system.
When a customer purchases an item the inventory stock is affected. This can be
modified to have triggers to alert the manager to order more stock or automate this
process. This type of query is very common with ecommerce websites such as amazon,
tesco, river island etc..
```

For part 4, I create an OrderLines entry, while updating the quantity_in_stock value for that product in the Inventory table.

In order to do this, I had to make some changes and add in addition Inventory and OrderLines table

```
type Inventory{
  id: ID!
  quantity_in_stock: Int
  sales: Int
  product: Products
  testupdate: String
}

type Orderlines{
  id: ID!
  product: Products
  order: Orders
  quantity: Int
  orderdate: DateTime
  inventoryId: String
}
```

I then created endpoints and mutation. Here is the code the actual mutation

```
  //part 4 - mutation that adds new order to the database
    createOrderlinesEndpoint(root, args, context) {
      console.log('-------- INSERTING ORDERLines ------------')
      console.log(args)

      //when orderlines created - take the quatity, and subtract this from the
inventory quantity of that product
      console.log('------ You are ordering ' + args.quantity + ' of product:' +
args.productId)

      let check = update(context,args);
      check.then(data=>{
        console.log(data.quantity_in_stock)
      })

      console.log('---------- INVENTORY QUANTITY UPDATED FOR PRODUCT :' +
args.productId)

      return context.prisma.createOrderlines({
        product: {
          connect: {
            id: args.productId
          }
        },
        order: {
          connect: {
            id: args.orderId
          }
        },
        quantity: args.quantity,
        orderdate: args.orderdate,
        inventoryId: args.inventoryId


      }, )
    },
```

```
function update(context,args){
  console.log('Updating the inventory quantity')

  return context.prisma.updateInventory({
    data: {
      quantity_in_stock:15,
      sales:1,
      testupdate: "UPDATED!"
    },
    where: {
      id: args.inventoryId
```

```
    }
  })
}
```

This is a hardcoded example as I found it difficult to execute atomic actions on the database with Prisma. But it still works and updates the testupdate string as well as quantity and sales. In order to test this part out I used the following mutation

```
mutation{
  createOrderlinesEndpoint(
    productId:"cjsyqrjod00be0a994jh6j48j",
    orderId:"cjsyqt4j800bu0a99tg5vxekm",
    quantity:5,
    orderdate:"06-03-19",
    inventoryId:"cjsyqsf1300bm0a99kzemwtpg"
  ){
    id
  }
}
```

And query to view the changes

```
query{
  getInventoryEndpoint{
  id
  sales
  quantity_in_stock
  testupdate
    product{
      id
      title
      price
    }


  }
}
```

Below are the remaining Mutation resolvers. These only perform a single relation create

```
  Mutation: {
    //create category
    createCategoryEndpoint(root, args, context) {
      console.log('-------- INSERTING CATEGORY ------------')
      console.log(args)
      return context.prisma.createCategory({
        categoryname: args.categoryname
      }, )
```

```
  },

  //create category
  createCustomerEndpoint(root, args, context) {
    console.log('-------- INSERTING CUSTOMER ------------')
    console.log(args)
    return context.prisma.createCustomer({
      firstname: args.firstname,
      lastname: args.lastname,
      address1: args.address1,
      address2: args.address2,
      city: args.city,
      state: args.state,
      zip: args.zip,
      county: args.county,
      region: args.region,
      email: args.email,
      phone: args.phone,
      creditcardtype: args.creditcardtype,
      creditcard: args.creditcard,
      creditcardexpiration: args.creditcardexpiration,
      username: args.username,
      password: args.password,
      age: args.age,
      income: args.income,
      gender: args.gender
    }, )
  },

  createProductsEndpoint(root, args, context) {
    console.log('-------- INSERTING Product ------------')
    console.log(args)
    return context.prisma.createProducts({
      title: args.title,
      actor: args.actor,
      price: args.price,
      special: args.special,
      common_prod_id: args.common_prod_id,
      category: {
        connect: {
          id: args.categoryId
        }
      }
    })
  },

  createOrdersEndpoint(root, args, context) {
    console.log('-------- INSERTING ORDERS ------------')
```

```
      console.log(args)
      return context.prisma.createOrders({
        orderdate: args.orderdate,
        netamount: args.netamount,
        tax: args.tax,
        totalamount: args.totalamount,
        customer: {
          connect: {
            id: args.customerId
          }
        }
      }, )
    },

    createCust_HistEndpoint(root, args, context) {
      console.log('-------- INSERTING Cust_Hists ------------')
      console.log(args)
      return context.prisma.createCust_Hist({
        customer: {
          connect: {
            id: args.customerId
          }
        },
        order: {
          connect: {
            id: args.orderId
          }
        },
        product: {
          connect: {
            id: args.productId
          }
        }

      })
    },
  }, //end mutatio
```

Screenshot

```
Mutating two relations (product + categories)
TODO
```

Part 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

```
//part 5
const server = new GraphQLServer({
```

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

testing queries

customer mutation

```
mutation {
  createCustomerEndpoint(
        firstname: "Eric",
        lastname: "Strong"
        address1: "11 village green, "
        address2: "kilbreck"
        city: "stamullen"
        state: "meath"
        zip: "01"
        county: "Meath"
        region: "Leinster"
   username:"es1989"
        email: "c15708709@mydit.ie"
        phone: "+353851077975"
   creditcardtype: "visa"
   creditcard: "12345"
        gender: "male"
        creditcardexpiration: "02-03-2020"
        password: "password1"
        age: 30
        income: 65000.00

  ) {
        firstname
   lastname
   id

  }
}
```

Customer query

```
query{
  getCustomersEndpoint{
    id
            firstname
             username
    creditcardexpiration
```

```
}
}
```

Categories Mutation

```
mutation{
   createCategoryEndpoint(
            categoryname: "ComputerScienceBooks"
  ) {
            categoryname
    id

  }
}
```

Categories Query

```
query{
  getCategoriesEndpoint{
    id
    categoryname
  }
}
```

Products Mutation

```
mutation{
  createProductsEndpoint(
  title: "Python 101"
  actor: "book"
  price: 9.99
  special: 5.00
  common_prod_id: 1
  categoryId: "cjsvmwdgg00i10a04vn2tagrd"
  ){
    id
    title
    price
    actor
    special
    category{
        id
            categoryname
      }


  }
}
```

Products Query

```
query{
  getProductsEndpoint{
    id
    title
    price
    category{
      id
      categoryname
    }

  }
```

```
}
```

Orders Mutation

```
mutation{
  createOrdersEndpoint(
          orderdate:"05-03-2019",
          netamount: 9.99,
          tax: 2.50,
          totalamount: 12.49,
                     customerId:"cjsvmuxu500hv0a0475f0reww"

  ){
    id
    customer{
      id
      firstname
    }

  }
}
```

Orders Query

```
query{
  getOrdersEndpoint{
    id
    totalamount
    customer{
      id
    }

  }
}
```

Cust_Hist Mutation

```
mutation{
  createCust_HistEndpoint(
  customerId: "cjsvmuxu500hv0a0475f0reww"
  orderId: "cjsvn0nwl00ii0a04il07uv6y"
  productId:"cjsvmy8dy00i80a040t71hlza"
  ){
id
    customer{
      id
      firstname
    }
    product{
      id
      title
      category{
        id
        categoryname
      }
    }
    order{
      id
      orderdate
    }
  }
}
```

Cust_Hist Query

```
query{
  getCust_HistsEndpoint{
    id
    product{
      id
    }
    customer{
      id
      firstname
    }
    order{
      id
    }
  }
}
```

Inventory Mutation

```
mutation{
  createInventoryEndpoint(
         quantity_in_stock:10
               sales: 0
                productId:"cjsyqrjod00be0a994jh6j48j"

  ){
    id
     quantity_in_stock
              sales
             product{
        id
        title
        price
      }

  }
}
```

Inventory Query

```
query{
  getInventoryEndpoint{
  id
  sales
  quantity_in_stock
  testupdate
    product{
      id
      title
      price
    }


  }
}
```

Orderlines Mutation

```
mutation{
  createOrderlinesEndpoint(
    productId:"cjsyqrjod00be0a994jh6j48j",
    orderId:"cjsyqt4j800bu0a99tg5vxekm",
```

```
    quantity:5,
    orderdate:"06-03-19",
    inventoryId:"cjsyqsf1300bm0a99kzemwtpg"
  ){
    id
  }
}
```

Orderlines Query

```
query{
  getOrderlinesEndpoint{
    id
    product{
      id
      title
    }
    order{
      id
      tax
    }
  }
}
```