Part One

I implemented all the tables in the ERD and included the relationships where appropriate as can be seen in datamodel.prisma and schema.graphql.

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

type Product {
  id: ID! @unique
  title: String!
  actor: String!
  price: Float!
  special: Int!
  common_prod_id: Int!
  category: Category
}

type Inventory {
  id: ID! @unique
  quan_in_stock: Int!
  sales: Int!
  product: Product!
}
```

```
type Reorder {
  id: ID! @unique
  date_low: DateTime!
  quan_low: Int!
  date_reordered: DateTime!
  quan_reordered: Int!
  date_expected: DateTime!
  product: Product!
}

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

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

type Orderline {
  id: ID! @unique
  order: Order!
  product: Product!
  quantity: Int!
  orderdate: DateTime!
}

type Cust_hist {
  customer: Customer!
  order: Order!
  product: Product!
}
```

Part Two

The following query in playground returns the first and last names of all customers.

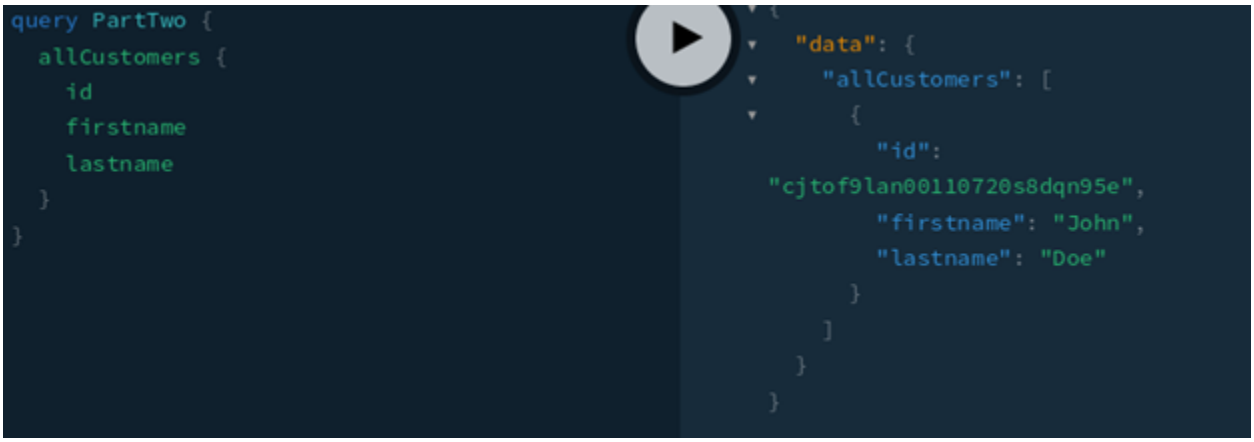
Schema.graphql

```
type Query {
  allCategories: [Category!]!
  allProducts: [Product!]!
  allInventories: [Inventory!]!
  allReorders: [Reorder!]!
  allOrders: [Order!]!
  allOrderlines: [Orderline!]!
  allCustomers: [Customer!]!
  allCust_hist: [Cust_hist!]!
  lineDetails(id: ID!): Orderline
}
```

Index.js resolver

```
allCustomers: (root, args, context, info) => {
  return context.prisma.customers()
},
```

Playground



Part Three

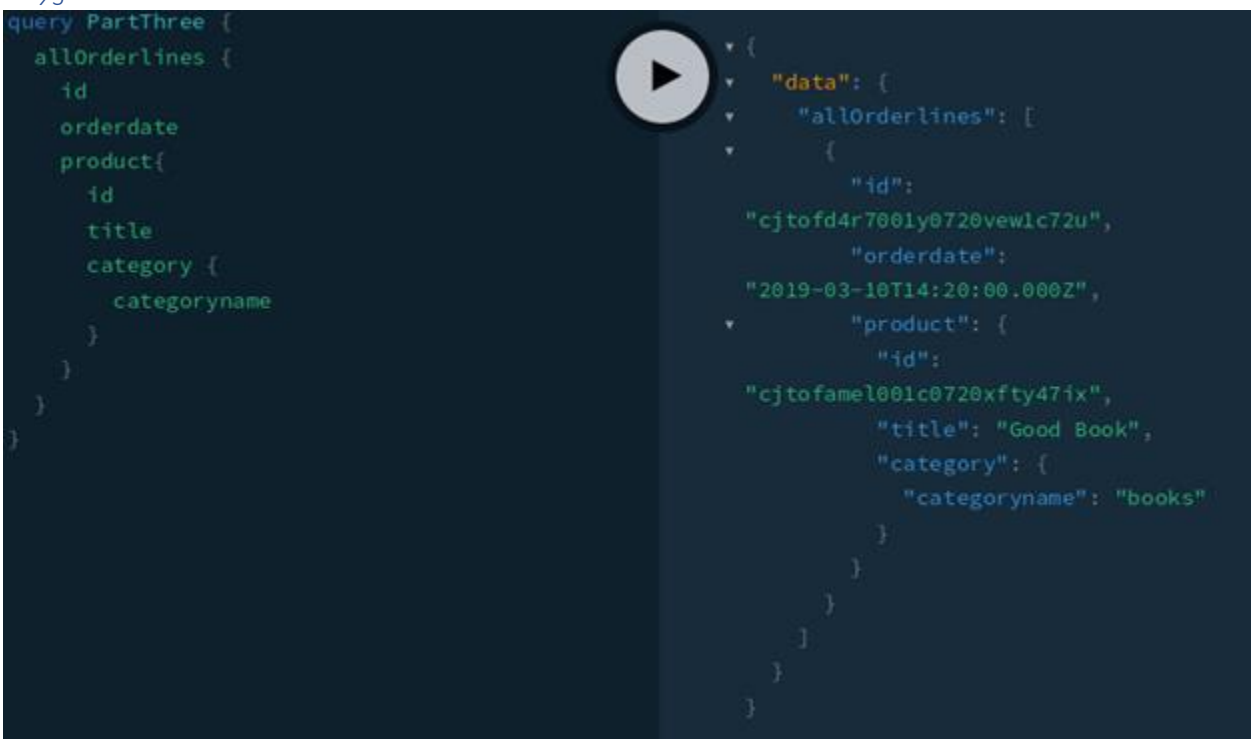
Schema.graphql

```
allOrderlines: [Orderline!]!
```

Index.js

```
//Part Three. This query returns Orderline information, product information,
//and the product category. It can be used to see product details for each
//line of an order. The lineDetails resolver performs the same function but
//an ID is used to specify the order line.
allOrderlines: (root, args, context, info) => {
  return context.prisma.orderlines()
},
```

Playground



Part Four

Schema.graphql

```
createOrders(
  orderdate: DateTime!
  netamount: Float!
  tax: Float!
  totalamount: Float!
  customerId: ID!
): Order!
```

Index.js

```
//This mutation can be used to create an order and add the relation to
//customer table.
createOrders: (root, args, context) => {
  return context.prisma.createOrder({
orderdate: args.orderdate,
netamount: args.netamount,
tax: args.tax,
totalamount: args.totalamount,
    customer: {
  connect: {
    id: args.customerId
  }
}
  })
},
```

Playground

```
mutation PartFour {
  createOrders(
    orderdate:"2019-03-17T16:57:31.000Z"
    netamount: 150.50
    tax: 13
    totalamount: 163.50
    customerId:"cjtof9lan00110720s8dqn95e"
  ){
    id
    customer{
      firstname
      lastname
      address1
      city
    }
  }
}
```

```
{
  "data": {
    "createOrders": {
      "id":
"cjtofb243001j0720vq6df4uj",
      "customer": {
        "firstname": "John",
        "lastname": "Doe",
        "address1": "50 generic
road",
        "city": "dublin"
      }
    }
  }
}
```

Part Five

Index.js

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