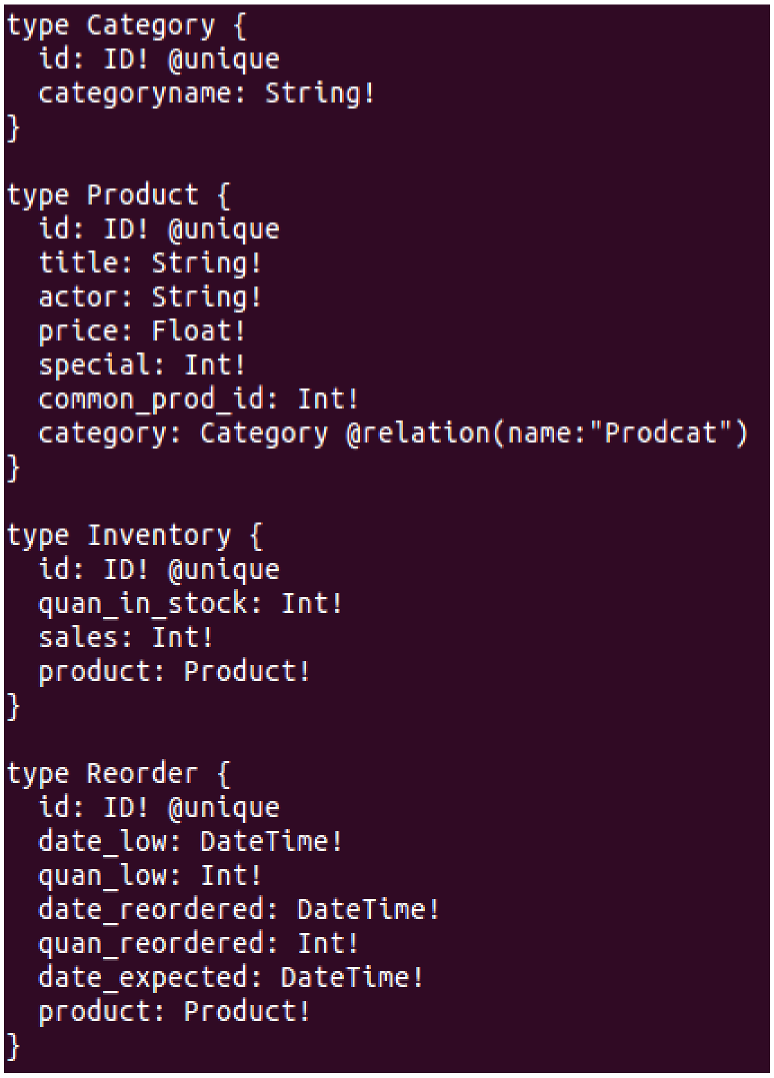
Problem Set One

I implemented the following four tables from the ER diagram and modelled their relationships accordingly.

datamodel.prisma



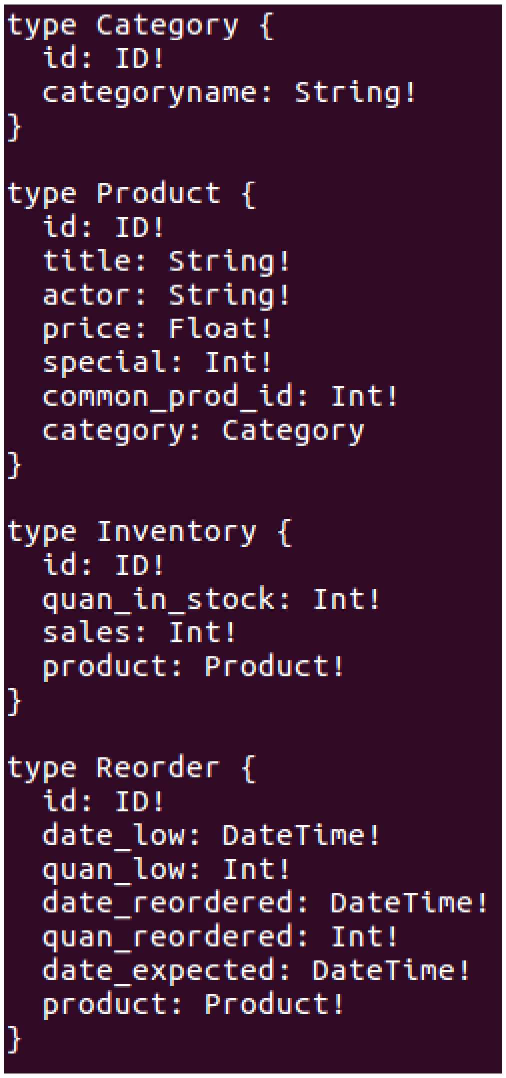
```
type Category {
  id: ID! @unique
  categoryname: String!
}

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

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!
}
```

schema.graphql



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

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

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

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

Problem Set Two

Index.js

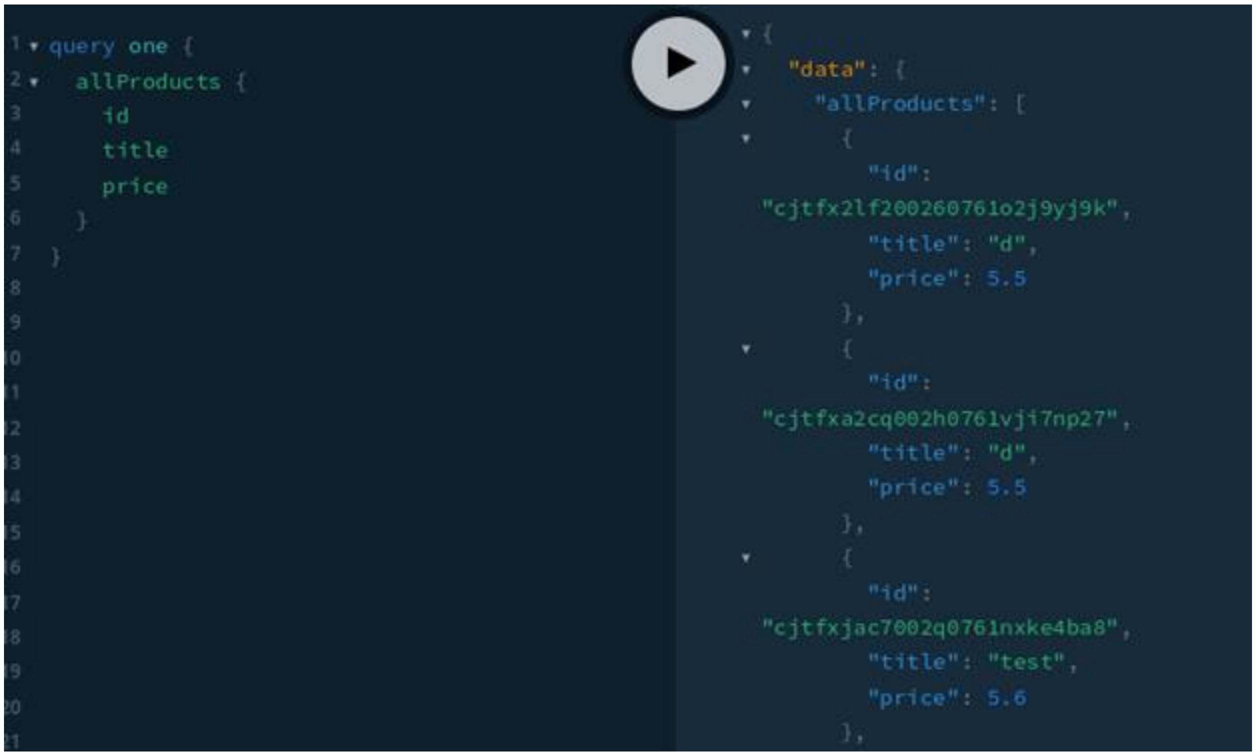
```
const resolvers = {
  Query: {
    allCategories: (root, args, context, info) => {
      return context.prisma.categories()
    },
    allProducts: (root, args, context, info) => {
      return context.prisma.products()
    },
    allInventories: (root, args, context, info) => {
      return context.prisma.inventories()
    },
    allReorders: (root, args, context, info) => {
      return context.prisma.reorders()
    },
  },
```

Schema.graphql

```
type Query {
  allCategories: [Category!]!
  allProducts: [Product!]!
  allInventories: [Inventory!]!
  allReorders: [Reorder!]!
}
```

Playground

This query returns a subset of attributes of all the products.

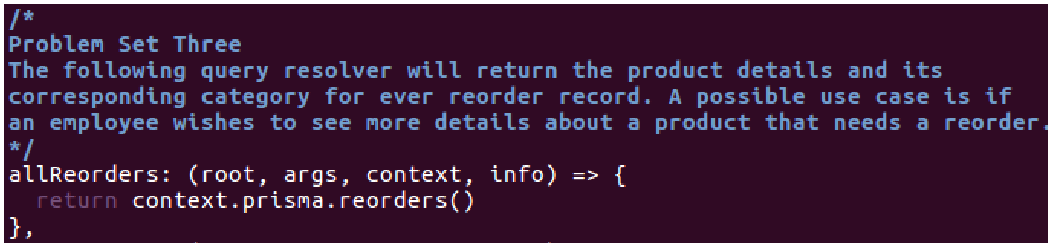


```
query one {
  allProducts {
    id
    title
    price
  }
}
```

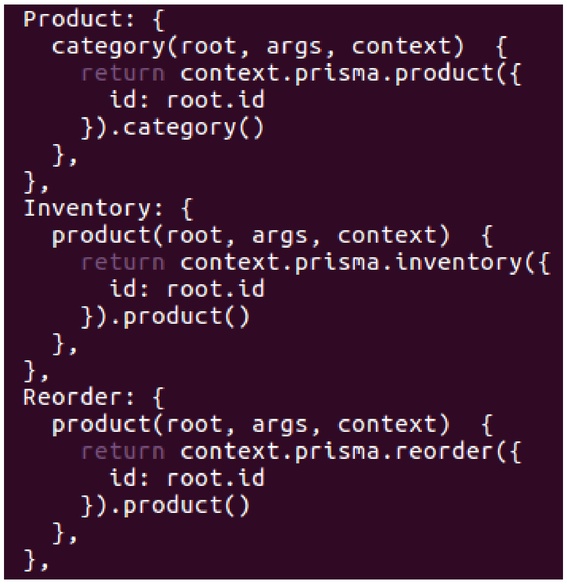
```
{
  "data": {
    "allProducts": [
      {
        "id":
"cjtfx2lf20026076lo2j9yj9k",
        "title": "d",
        "price": 5.5
      },
      {
        "id":
"cjtfxa2cq002h0761vji7np27",
        "title": "d",
        "price": 5.5
      },
      {
        "id":
"cjtfxjac7002q0761nxke4ba8",
        "title": "test",
        "price": 5.6
      },
```

Problem Set Three

Index.js

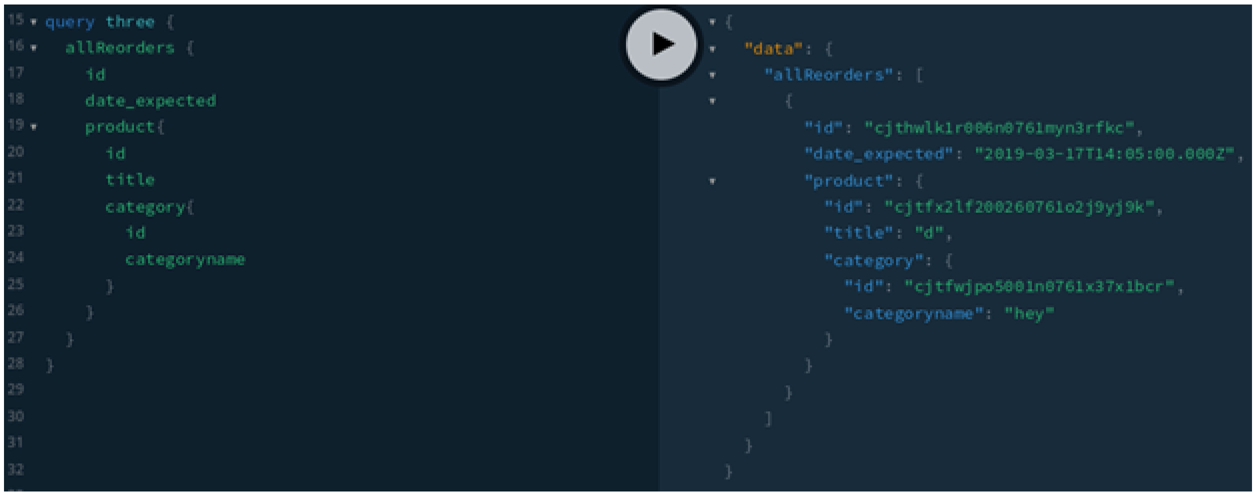


```
/*
Problem Set Three
The following query resolver will return the product details and its
corresponding category for ever reorder record. A possible use case is if
an employee wishes to see more details about a product that needs a reorder.
*/
allReorders: (root, args, context, info) => {
  return context.prisma.reorders()
},
```



```
 Product: {
   category(root, args, context)  {
     return context.prisma.product({
       id: root.id
     }).category()
   },
 },
 Inventory: {
   product(root, args, context)  {
     return context.prisma.inventory({
       id: root.id
     }).product()
   },
 },
 Reorder: {
   product(root, args, context)  {
     return context.prisma.reorder({
       id: root.id
     }).product()
   },
 },
```

Playground

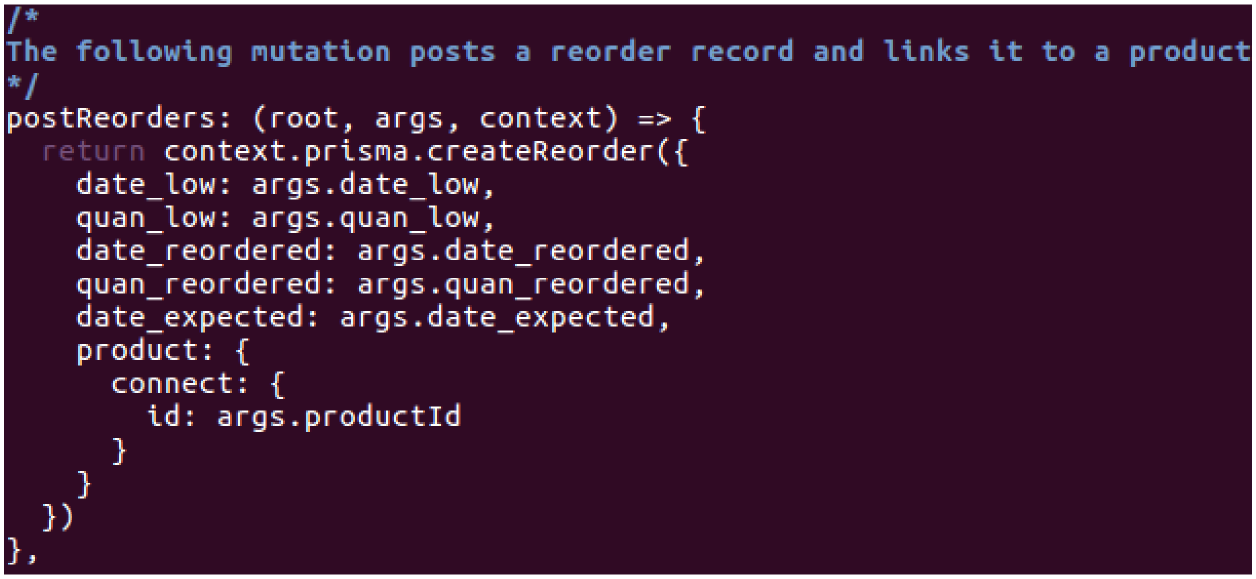


```
query three {
  allReorders {
    id
    date_expected
    product{
      id
      title
      category{
        id
        categoryname
      }
    }
  }
}
```

```
{
  "data": {
    "allReorders": [
      {
        "id": "cjthwlk1r006n0761myn3rfkc",
        "date_expected": "2019-03-17T14:05:00.000Z",
        "product": {
          "id": "cjtfx2lf200260761o2j9yj9k",
          "title": "d",
          "category": {
            "id": "cjtfwjpo5001n0761x37x1bcr",
            "categoryname": "hey"
          }
        }
      }
    ]
  }
}
```

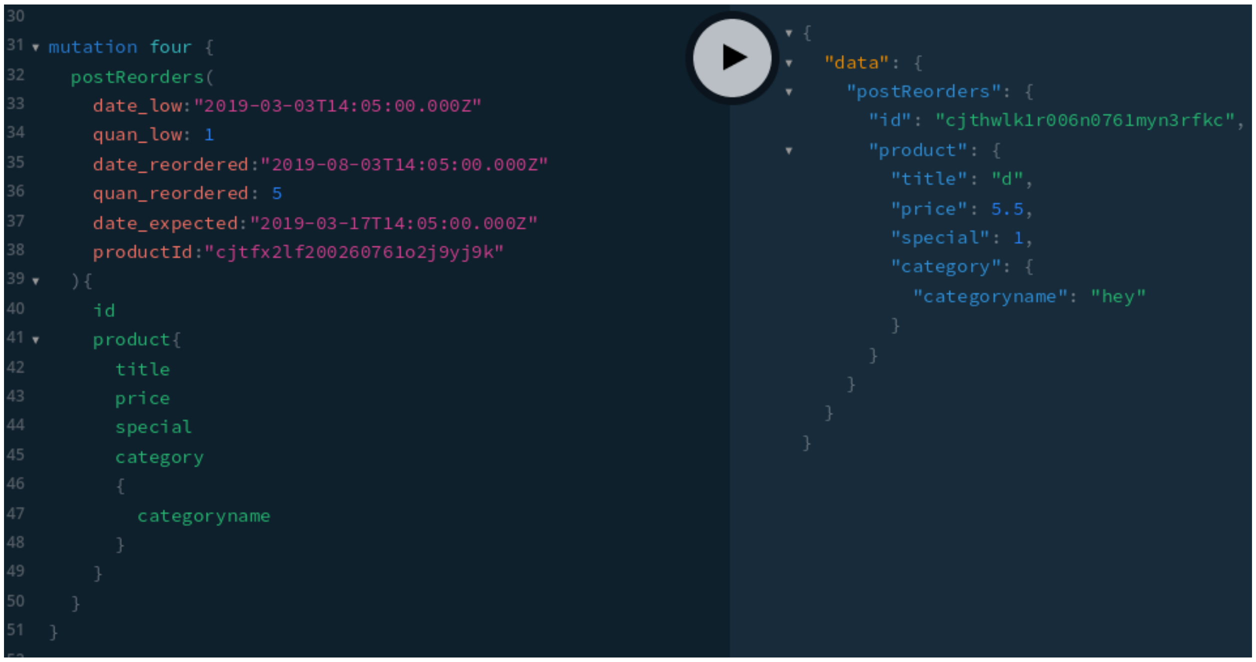
Problem Set Four

Index.js



```
/*
The following mutation posts a reorder record and links it to a product
*/
postReorders: (root, args, context) => {
  return context.prisma.createReorder({
    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,
    product: {
      connect: {
        id: args.productId
      }
    }
  })
},
```

playground

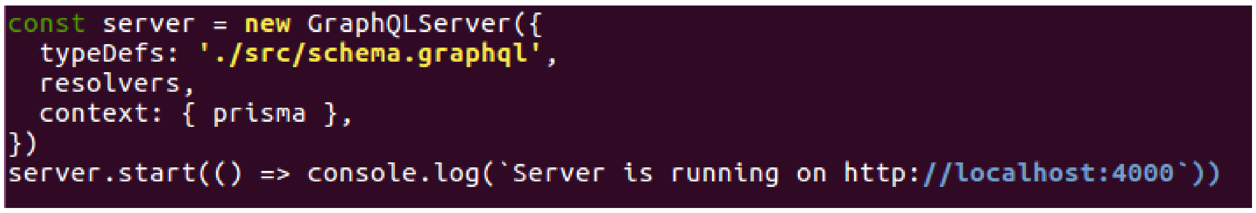


```
mutation four {
  postReorders(
    date_low:"2019-03-03T14:05:00.000Z"
    quan_low: 1
    date_reordered:"2019-08-03T14:05:00.000Z"
    quan_reordered: 5
    date_expected:"2019-03-17T14:05:00.000Z"
    productId:"cjtfx2lf20026076lo2j9yj9k"
  ){
    id
    product{
      title
      price
      special
      category
      {
        categoryname
      }
    }
  }
}
```

```
{
  "data": {
    "postReorders": {
      "id": "cjthwlk1r006n0761myn3rfkc",
      "product": {
        "title": "d",
        "price": 5.5,
        "special": 1,
        "category": {
          "categoryname": "hey"
        }
      }
    }
  }
}
```

Problem Set Five

As can be seen from the above screenshots, a graphql server is running correctly and the schema can be queried.



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