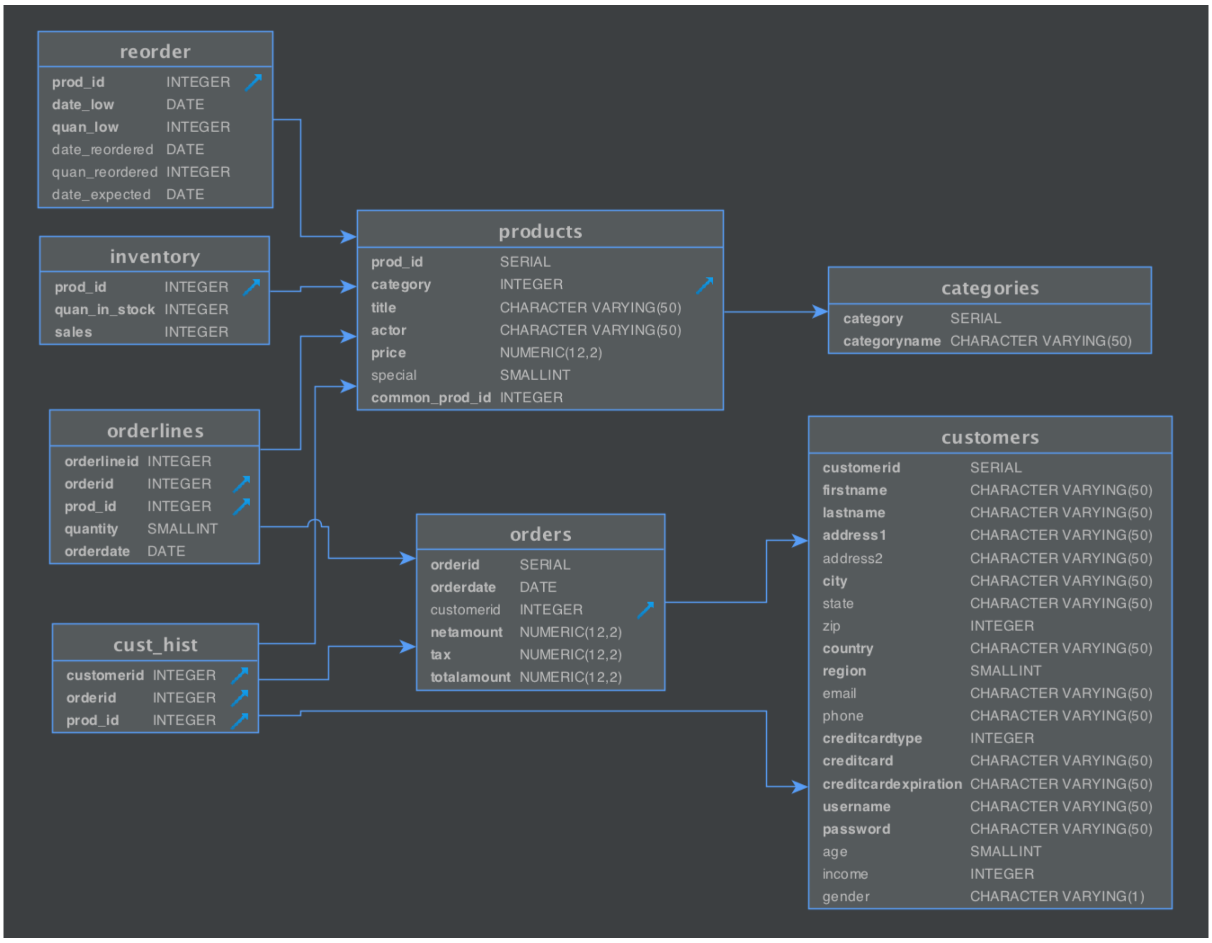
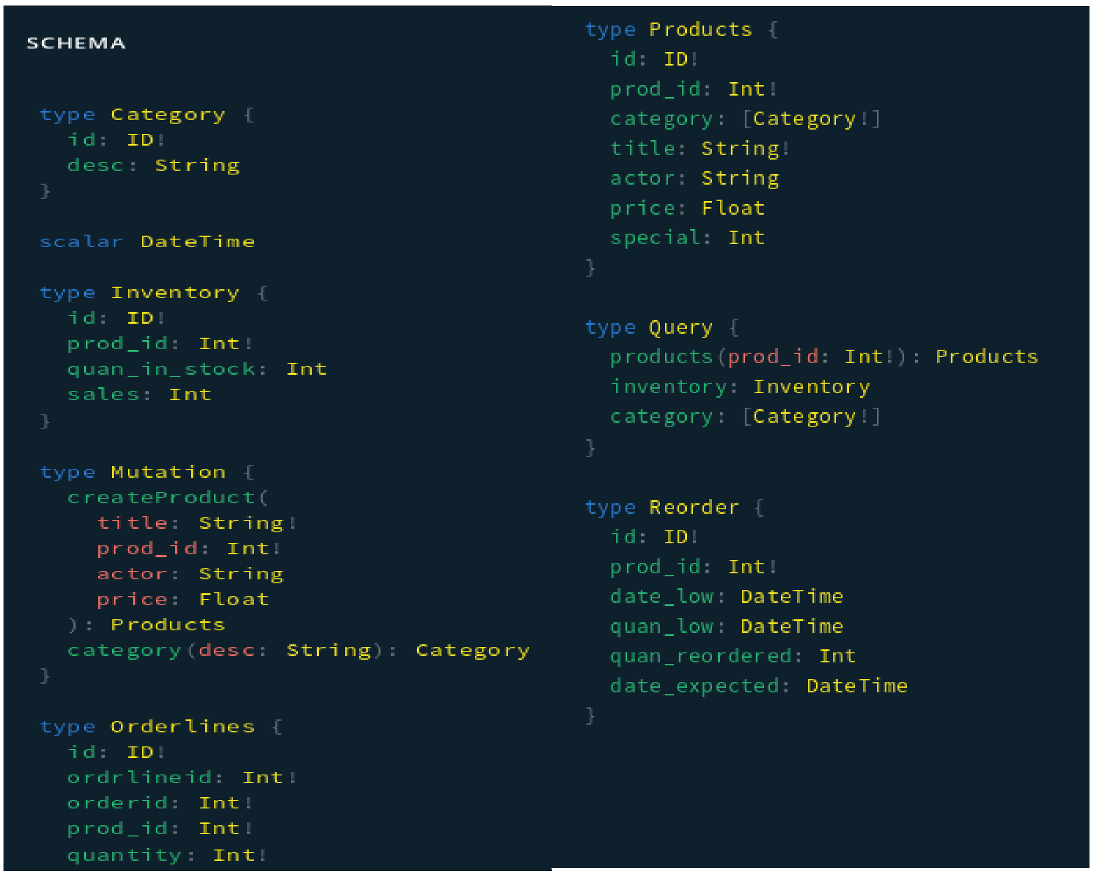
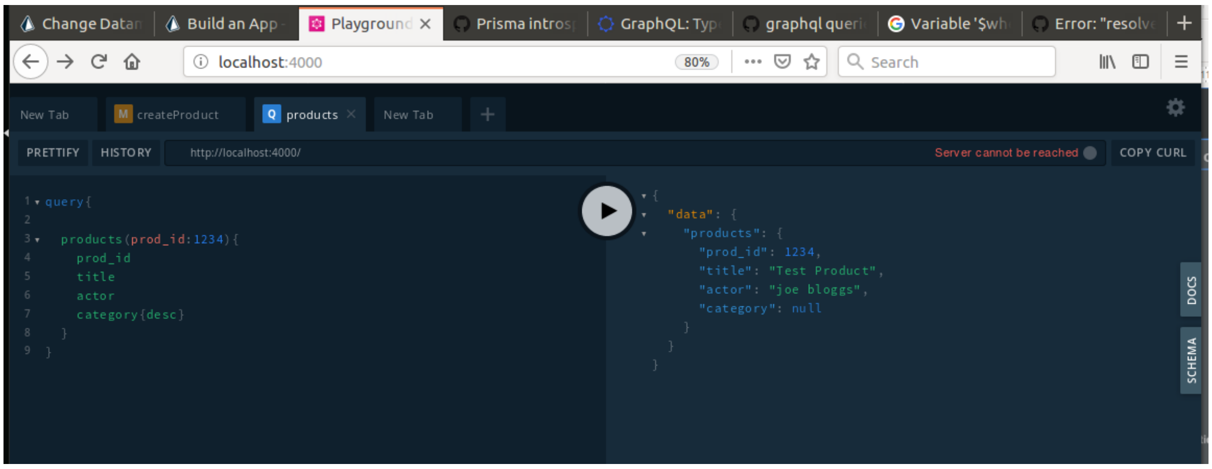
1

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



25 Marks

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

```
SCHEMA

type Category {
  id: ID!
  desc: String
}

scalar DateTime

type Inventory {
  id: ID!
  prod_id: Int!
  quan_in_stock: Int
  sales: Int
}

type Mutation {
  createProduct(
    title: String!
    prod_id: Int!
    actor: String
    price: Float
  ): Products
  category(desc: String): Category
}

type Orderlines {
  id: ID!
  ordrlineid: Int!
  orderid: Int!
  prod_id: Int!
  quantity: Int!
```

```
type Products {
  id: ID!
  prod_id: Int!
  category: [Category!]
  title: String!
  actor: String
  price: Float
  special: Int
}

type Query {
  products(prod_id: Int!): Products
  inventory: Inventory
  category: [Category!]
}

type Reorder {
  id: ID!
  prod_id: Int!
  date_low: DateTime
  quan_low: DateTime
  quan_reordered: Int
  date_expected: DateTime
}
```

localhost:4000

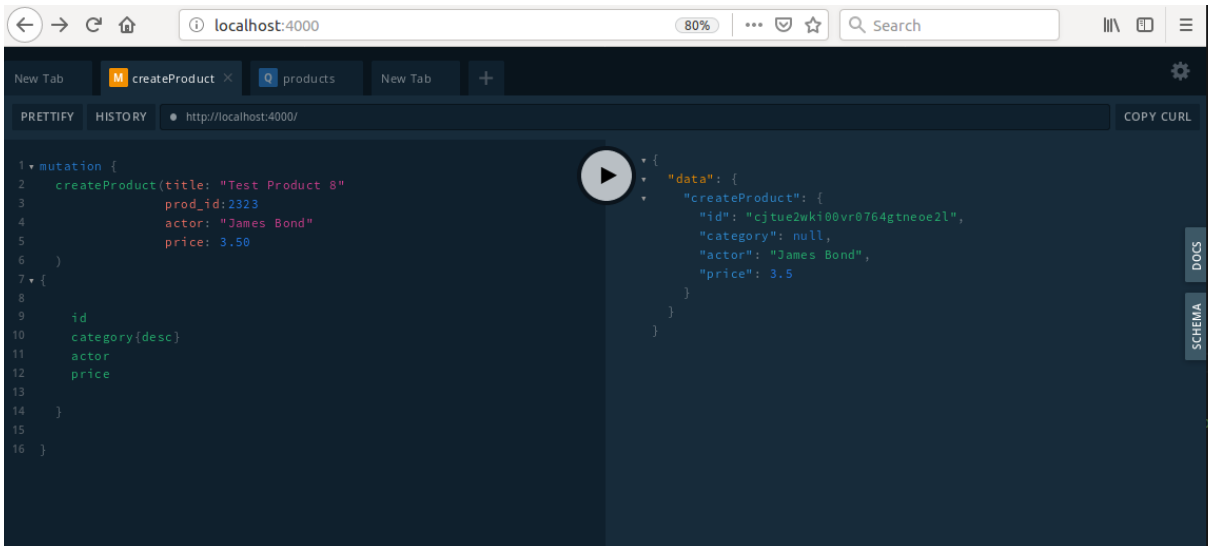
New Tab | createProduct | products | New Tab

PRETTIFY | HISTORY | http://localhost:4000/ | Server cannot be reached | COPY CURL

```
query{

  products(prod_id:1234){
    prod_id
    title
    actor
    category{desc}
  }
}
```

```
{
  "data": {
    "products": {
      "prod_id": 1234,
      "title": "Test Product",
      "actor": "joe bloggs",
      "category": null
    }
  }
}
```

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	20 Marks
4	Create a mutation resolver to add data the database. Your mutation should update at least two relations (of your choice) 	20 Marks

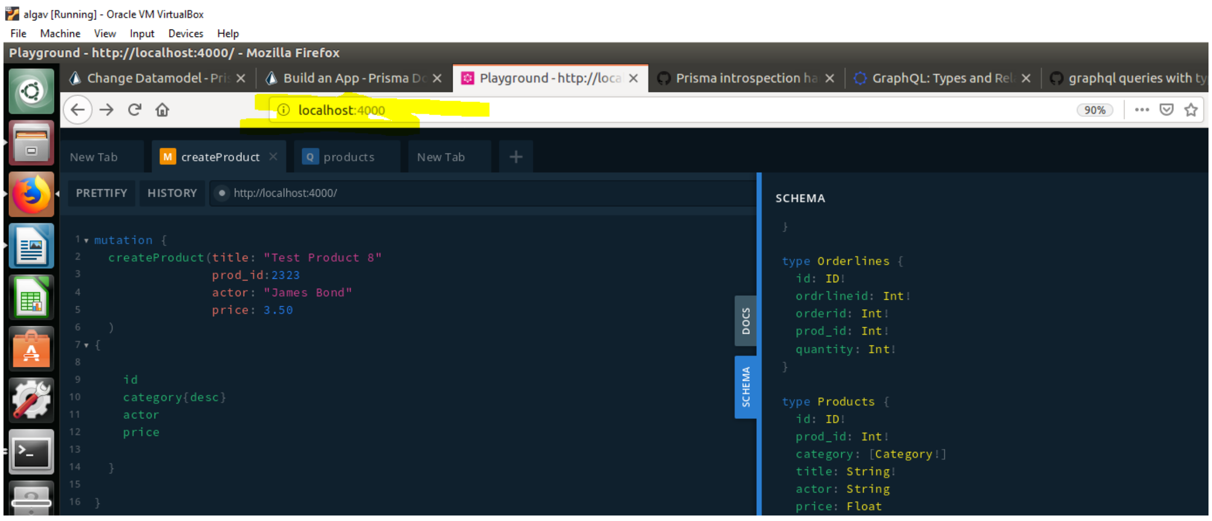
```
mutation {
  createProduct(title: "Test Product 8"
                prod_id:2323
                actor: "James Bond"
                price: 3.50
  )
{

    id
    category{desc}
    actor
    price

  }

}
```

```
{
  "data": {
    "createProduct": {
      "id": "cjtue2wki00vr0764gtneoe2l",
      "category": null,
      "actor": "James Bond",
      "price": 3.5
    }
  }
}
```

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

algav [Running] - Oracle VM VirtualBox

Playground - http://localhost:4000/ - Mozilla Firefox

```
mutation {
  createProduct(title: "Test Product 8"
                prod_id:2323
                actor: "James Bond"
                price: 3.50
  )
{

    id
    category{desc}
    actor
    price

  }

}
```

```
SCHEMA

}

type Orderlines {
  id: ID!
  ordrlineid: Int!
  orderid: Int!
  prod_id: Int!
  quantity: Int!
}

type Products {
  id: ID!
  prod_id: Int!
  category: [Category!]
  title: String!
  actor: String
  price: Float
```