PART 2

To generate a list of the 3 most popular colors per year based on the total purchase quantities from the PurchaseOrder collection, I will do the following:

1) Group the PurchaseOrder collection by year using the purchaseDate field.

2) For each year group:

a. Join the PurchaseOrder records with the corresponding Product records using the ean field.

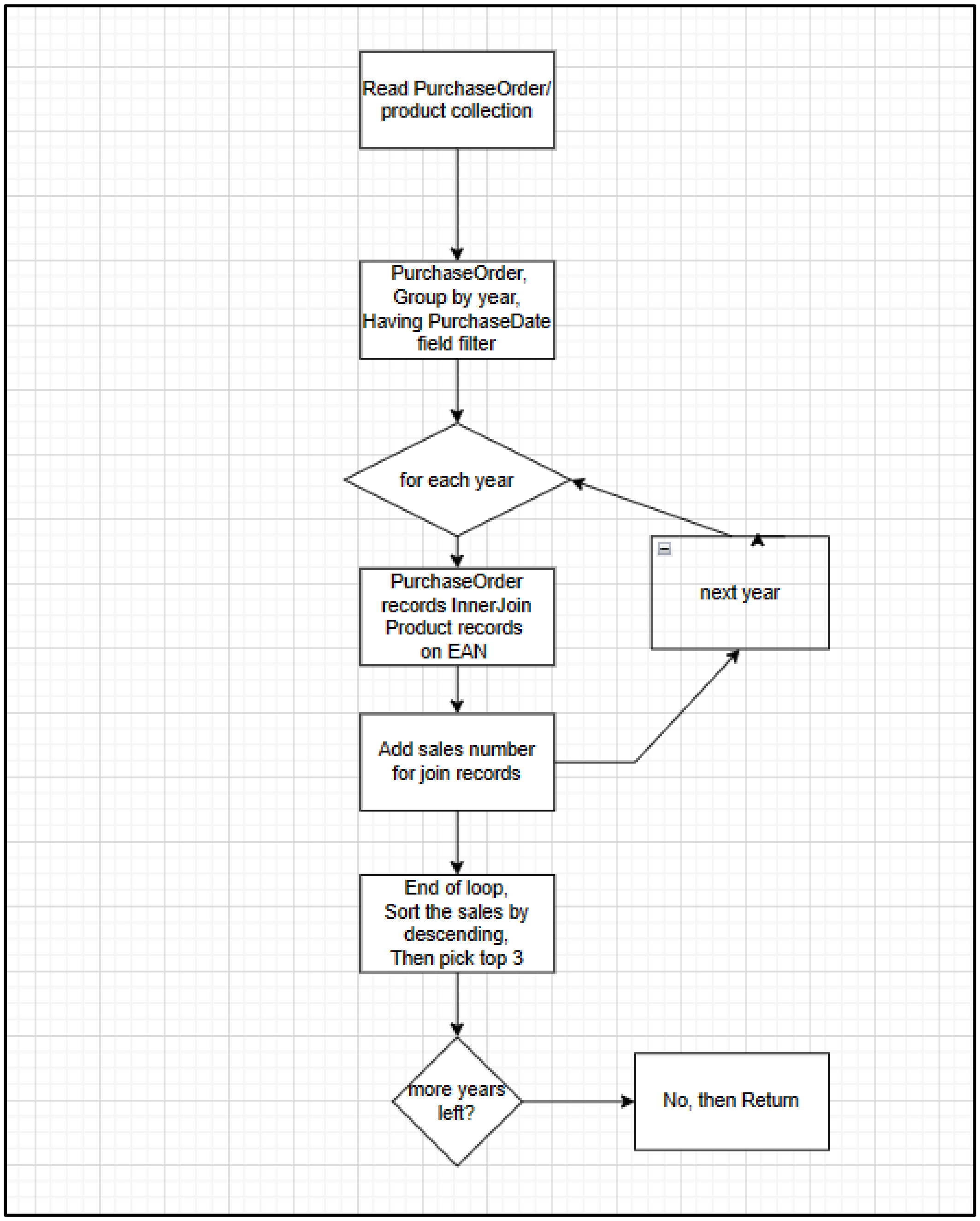
b. Sum the quantity for each color value from the joined Product records.

c. Sort the color values based on the summed quantity in descending order.

d. Take the top 3 color values and their corresponding summed quantity.

3) Return the list of top 3 colors and their total purchase quantities for each year.

Here is the diagram:


Read PurchaseOrder/ product collection
PurchaseOrder, Group by year, Having PurchaseDate field filter
for each year
next year
PurchaseOrder records InnerJoin Product records on EAN
Add sales number for join records
End of loop, Sort the sales by descending, Then pick top 3
more years left?
No, then Return

As I mentioned in the part 1, coding, I would prefer graphql schema for the same purpose, hence I am explaining a bit about graphene schema here.

To implement this solution using GraphQL with GrapheneGraphQL, I will follow these steps:

1) Define your data models (e.g., Product and PurchaseOrder) as GraphQL object types using Graphene. (In part 1, I have defined only product models, not the purchase models)
2) Create a GraphQL query that retrieves the PurchaseOrder and Product data, and applies the necessary filters and sorting.
3) After that, I will generate resolver for GraphQL, follow the logic outlined in chart above, to process the data and generate the expected results.

Here are some code snippet:

```
class ProductNode(DjangoObjectType):

    class Meta:

        model = Product

        interfaces = (relay.Node,)


class PurchaseOrderNode(DjangoObjectType):

    class Meta:

        model = PurchaseOrder

        interfaces = (relay.Node,)
```

And queries:

```
class Query(graphene.ObjectType):

    products = DjangoListField(ProductNode)

    purchase_orders = DjangoListField(PurchaseOrderNode)


    def resolve_products(self, info, **kwargs):

        # Retrieve all products with filters if needed
        return Product.objects.all()


    def resolve_purchase_orders(self, info, **kwargs):
        # Retrieve all purchase orders with filters if needed
        return PurchaseOrder.objects.all()
```

By using GraphQL with GrapheneGraphQL, we can efficiently retrieve the data we need and apply the necessary filters and sorting directly in your GraphQL queries. This can help reduce the amount of data transferred over the network and improve the overall performance of your application. And I believe, the important reason to choose over GraphQL is, we only fetch only the data we need unlike other API like RESTful.