

# GraphQL, Apollo and Neo4j using React

Jorge Rios  
Miguel Uc

November 2, 2018

Ksquare

# Contents

- 1 Schemma postgresql
- 2 Schema Neo4j
- 3 Apollo
- 4 GraphQL
- 5 Aplicacion
- 6 Referencias

1 Schemma postgresql

2 Schema Neo4j

3 Apollo

4 GraphQL

5 Aplicacion

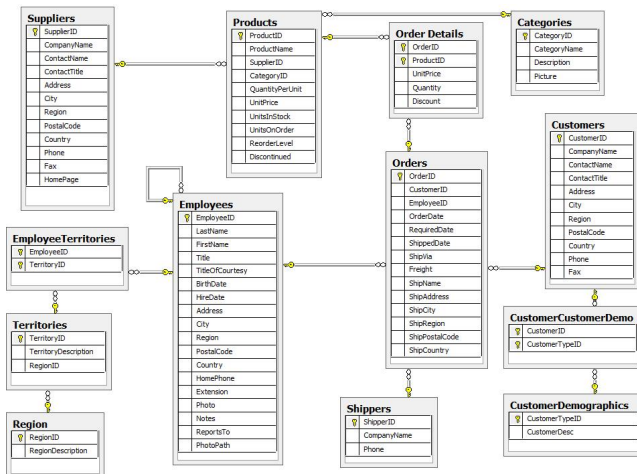
6 Referencias

# Schema postgresql

We will be using the NorthWind dataset, a commonly-used SQL dataset. Although the NorthWind dataset is often used to demonstrate SQL and relational databases, it is graphy enough to be interesting for us.

# Schema postgresql

- The following is an entity-relationship diagram of the Northwind dataset:



# Schema postgresql

```
northwind=# \dt
```

## List of relations

Schema	Name	Type	Owner
public	categories	table	postgres
public	customercustomerdemo	table	postgres
public	customerdemographics	table	postgres
public	customers	table	postgres
public	employees	table	postgres
public	employeeterritories	table	postgres
public	order_details	table	postgres
public	orders	table	postgres
public	products	table	postgres
public	region	table	postgres
public	shippers	table	postgres
public	shippers_tmp	table	postgres
public	suppliers	table	postgres
public	territories	table	postgres
public	usstates	table	postgres

(15 rows)

- 1 Schemma postgresql
- 2 **Schema Neo4j**
- 3 Apollo
- 4 GraphQL
- 5 Aplicacion
- 6 Referencias

# Schema Neo4j

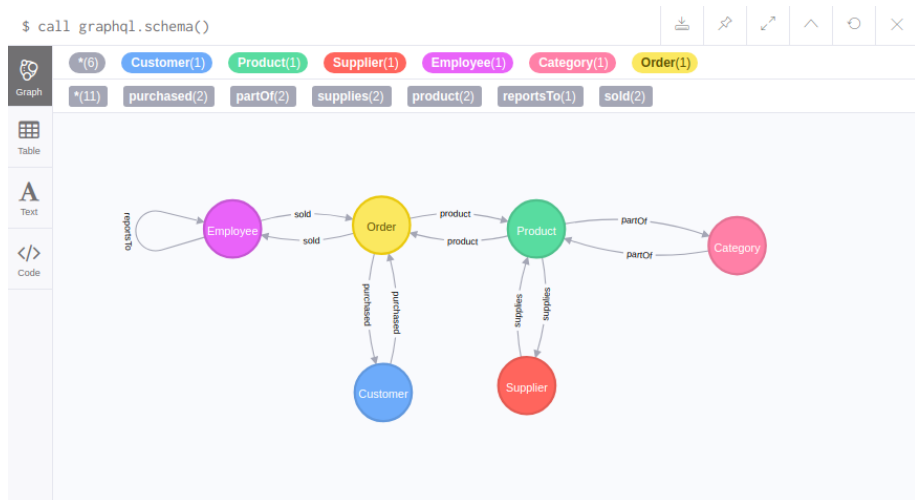
When deriving a graph model from a relational model, we should keep the following guidelines in mind:

- A row is a node
- A table name is a label name



# Schema Neo4j

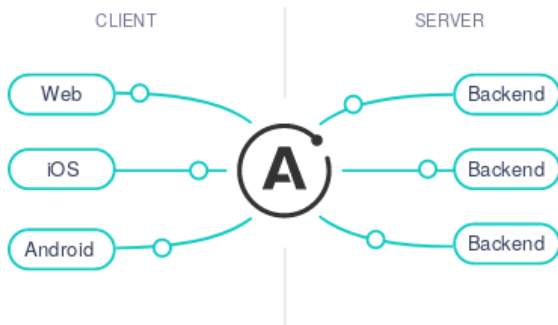
The following graph model represent the Northwind dataset:



- 1 Schemma postgresql
- 2 Schema Neo4j
- 3 Apollo**
- 4 GraphQL
- 5 Aplicacion
- 6 Referencias

# Apollo

- Apollo is a family of technologies you can incrementally add to your stack: Apollo Client to connect data to your UI, Apollo Engine for infrastructure and tooling, and Apollo Server to translate your REST API and backends into a GraphQL schema.



- 1 Schemma postgresql
- 2 Schema Neo4j
- 3 Apollo
- 4 GraphQL**
- 5 Aplicacion
- 6 Referencias

# GraphQL

- GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data.
- For example, a GraphQL service that tells us who the logged in user is (me) as well as that user's name might look something like this:

```
query {  
  user(login: "danbri") {  
    repositories(first: 2) {  
      nodes {  
        owner {  
          login  
        }  
      }  
    }  
  }  
}
```

(a) initial example query

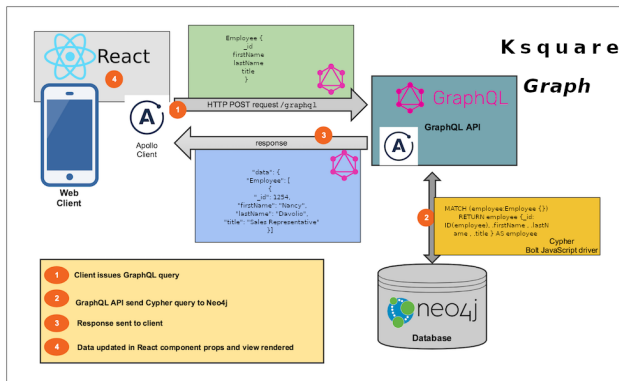
```
data: {  
  user: {  
    repositories: {  
      nodes: [  
        { owner: { login: "danbri" } },  
        { owner: { login: "danbri" } }  
      ]  
    }  
  }  
}
```

(b) result of initial example query

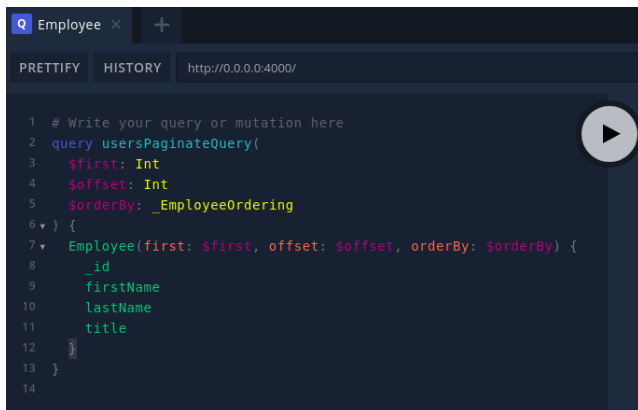
- 1 Schemma postgresql
- 2 Schema Neo4j
- 3 Apollo
- 4 GraphQL
- 5 Aplicacion**
- 6 Referencias

# Application

- This project is a starter for building a GraphKsquare (GraphQL, React, Apollo, Neo4j Database) application. There are two components to the starter, the UI application (a React app) and the API app (GraphQL server).







The screenshot shows a GraphQL IDE interface. At the top, there's a tab labeled 'Employee' with a search icon and a plus sign. Below the tab are two buttons: 'PRETTIFY' and 'HISTORY', followed by the URL 'http://0.0.0.0:4000/'. The main area contains a GraphQL query with line numbers 1 through 14. A play button icon is visible on the right side of the query editor.

```
1 # Write your query or mutation here
2 query usersPaginateQuery(
3   $first: Int
4   $offset: Int
5   $orderBy: _EmployeeOrdering
6 ) {
7   Employee(first: $first, offset: $offset, orderBy: $orderBy) {
8     _id
9     firstName
10    lastName
11    title
12  }
13 }
14
```

# API output

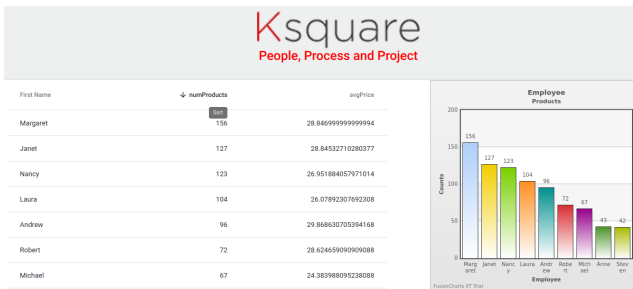
```
▼ {  
  ▼ "data": {  
    ▼ "Employee": [  
      ▼ {  
        "_id": 1254,  
        "firstName": "Nancy",  
        "lastName": "Davolio",  
        "title": "Sales Representative"  
      },  
      ▼ {  
        "_id": 1255,  
        "firstName": "Andrew",  
        "lastName": "Fuller",  
        "title": "Vice President, Sales"  
      },  
      ▼ {  
        "_id": 1256,  
        "firstName": "Janet",  
        "lastName": "Leverling",  
        "title": "Sales Representative"  
      },  
    ]  
  }  
}
```

SCHEMA

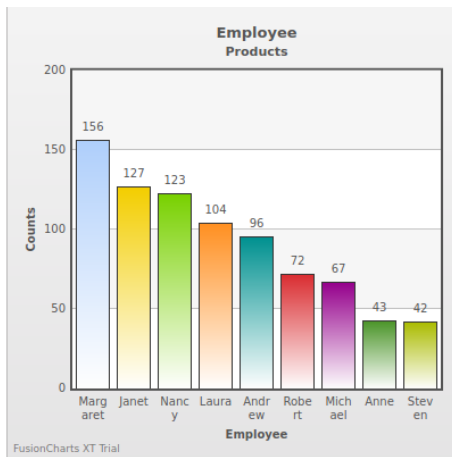
## Ksquare

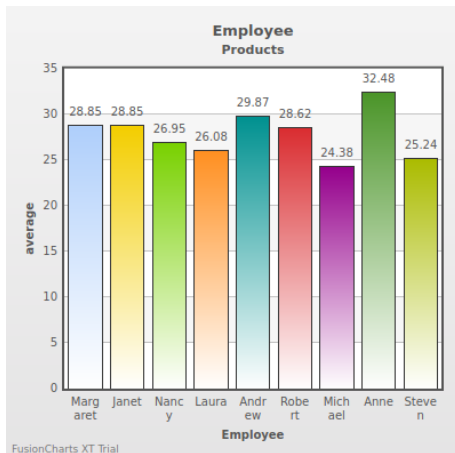
**People, Process and Projects**

First Name	Last Name	Title
Andrew	Fuller	Vice President, Sales
Anne	Dodsworth	Sales Representative
Janet	Leverling	Sales Representative
Laura	Callahan	Inside Sales Coordinator
Margaret	Peacock	Sales Representative
Michael	Suyama	Sales Representative
Nancy	Davolio	Sales Representative
Robert	King	Sales Representative



# UI application





# About Security

- Denial of Service Attack.
  - ① Whitelist of queries using Apollo's persistgraphql
  - ② Depth limiting using graphql-depth-limit
  - ③ Amount limiting (99999) with custom scalar created with graphql-input-number
  - ④ Query Cost Analysis with graphql-cost-analysis (Mutations are hard to estimate)

## About Security 2

An example of security scheme is the one used by GitHub:

- Token based authentication (or bearer authentication). It can be applied to operation level. It can use JWT format.
- Clients must supply a first or last argument on any connection.
- Values of first and last must be within 1-100.
- Individual calls cannot request more than 500,000 total nodes.
- Rate limit is 5,000 points per hour. A formula is applied to each query on parent and children arguments.



- 1 Schemma postgresql
- 2 Schema Neo4j
- 3 Apollo
- 4 GraphQL
- 5 Aplicacion
- 6 Referencias**

# Referencias

-  <https://graphql.org/learn/>
-  <https://www.apollographql.com/>
-  <https://neo4j.com/blog/>
-  <https://grandstack.io/>
-  <https://developer.github.com>