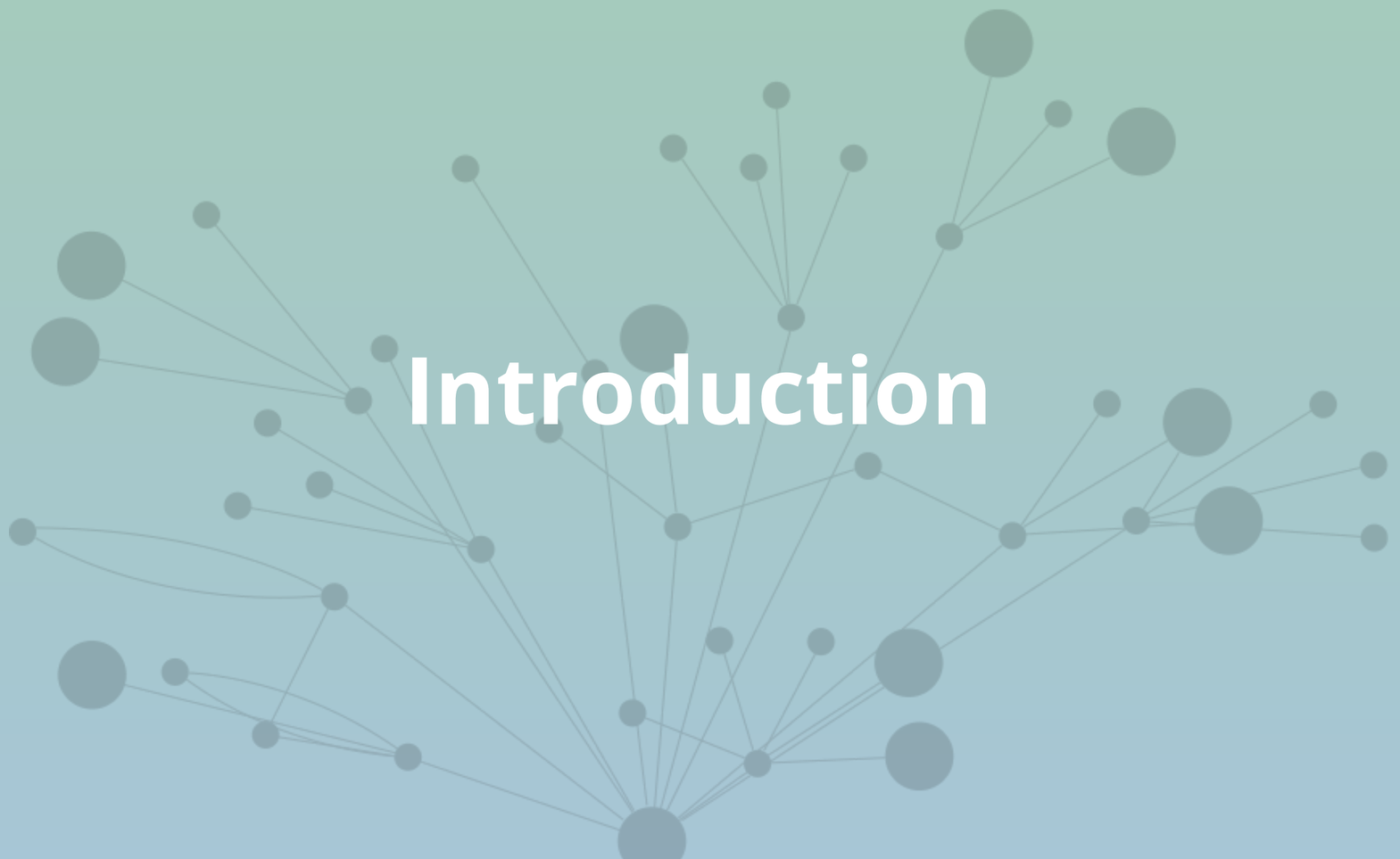
GRAND Stack

Benoit Simard (@logisima)

neo4j



Introduction

Benoit Simard

- **Graph addict**
- Data liberator
- Web developer
- Works at Neo4j
- benoit@neo4j.com
- @logisima

Agenda

- What is this Stack ?
- Why a new stack?
- How to use it ?

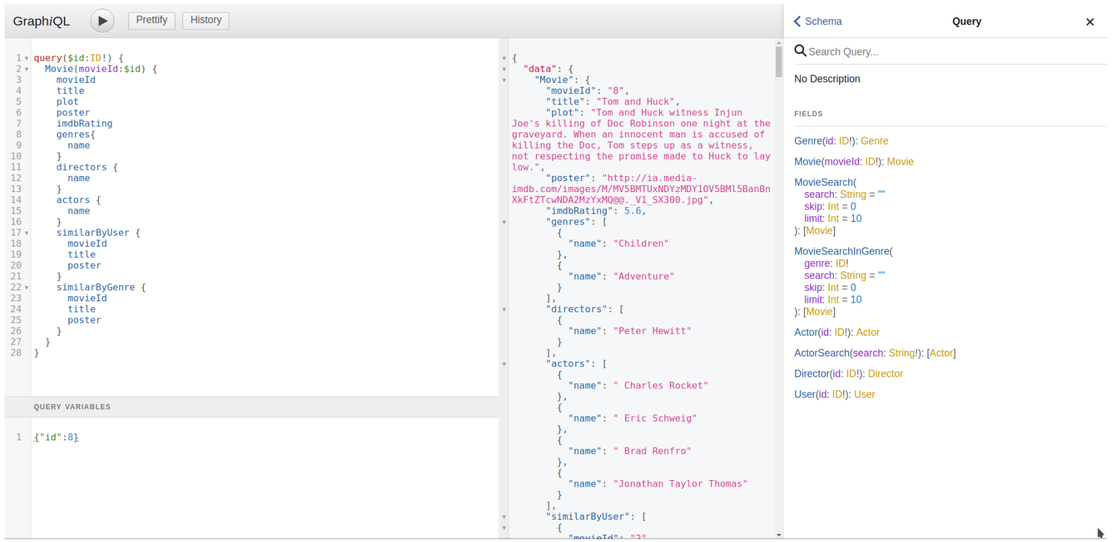
La stack

http://grandstack.io

GRAND

GATAND

GraphQL



- A new paradigm for **building APIs**
- **Schema** definition
- Query language for APIs
- Community of tools

Angular

- JavaScript library for building user interfaces
- Component based
  - Reusable
  - Composable
- Written in Typescript (type matters !)

```
import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'display-object',
  templateUrl: './display-object.component.html',
  styleUrls: ['./display-object.component.scss']
})
export class DisplayObjectComponent  implements OnInit{

  @Input() object:any;
  @Input() blacklist:Array<String> = [];
  @Input() inlineSeparator:any = null;
  keys:any;

  constructor() {}

  ngOnInit() {
    // find keys that are not blacklisted and wherethe key has a value
    this.keys = Object.keys(this.object).filter( (key) =>{
      let result = true;

      if(key == '__typename'){
        result = false;
      }

      if(this.blacklist.includes(key)) {
        result = false;
      }

      if(this.object[key] == null || this.object[key] == '') {
        result = false;
      }

      return result;
    })
  }

}
```

Apollo

"A set of tools designed to leverage GraphQL and work together to create a great workflow"

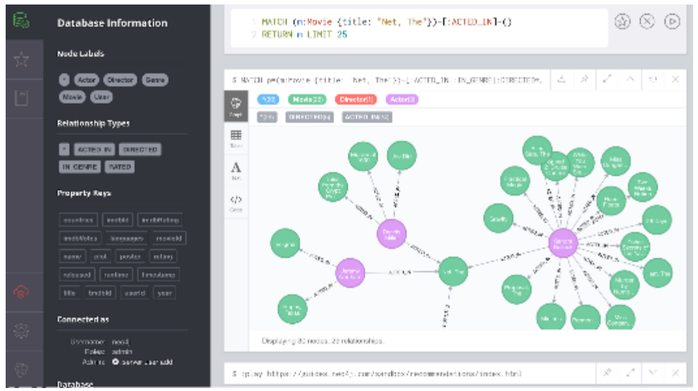
Client-side tooling

- Frontend framework integrations
- Caching
- Code generation

Server-side tooling

- Schema creation
- Mocking
- Schema stitching
- Performance monitoring

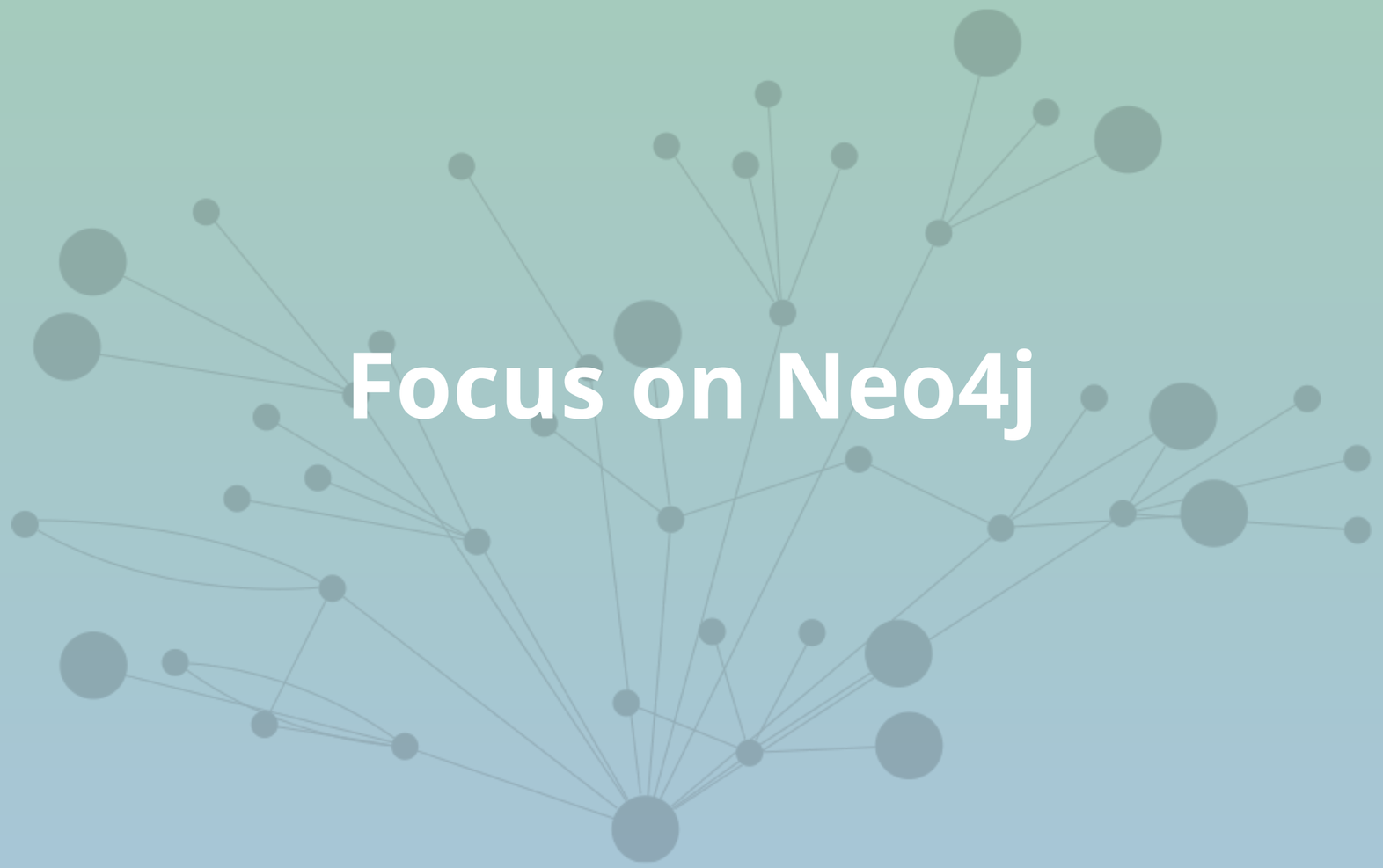
Neo4j



- Is a graph **database** (ACID compliant)
- Is a **graph** database (Native)
- Schema less
- Exists since 2010

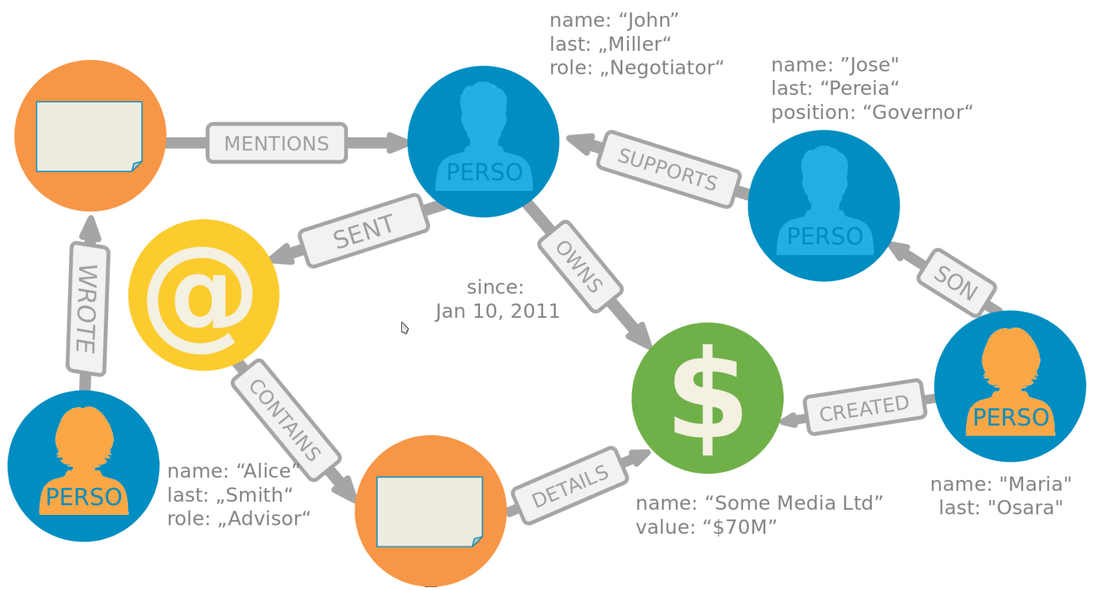
Example : a movie application

https://github.com/sim51/grand-movies-example



Focus on Neo4j

A graph of properties



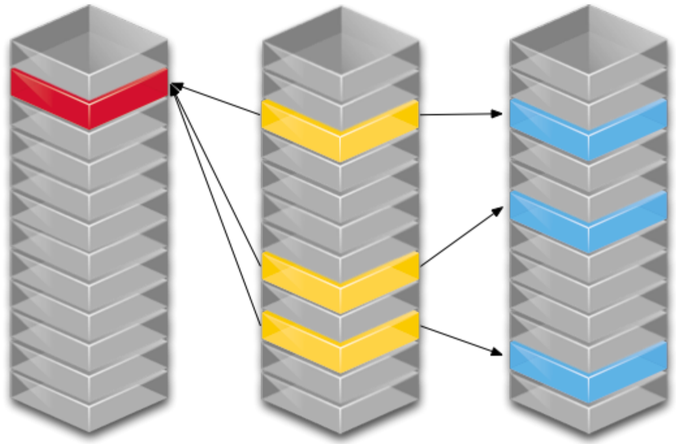
Nodes

- The entity of your model
- Can have labels
- Can have properties

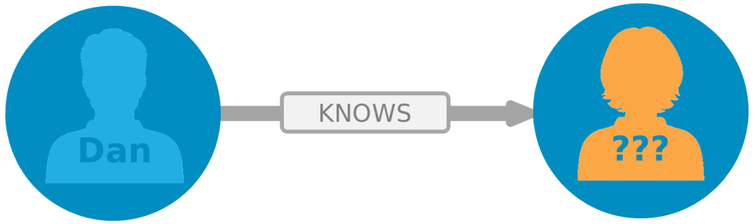
Relationships

- Links two nodes, with a **direction** and a **type**
- Can have properties

A local approach



Cypher

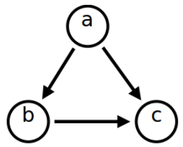


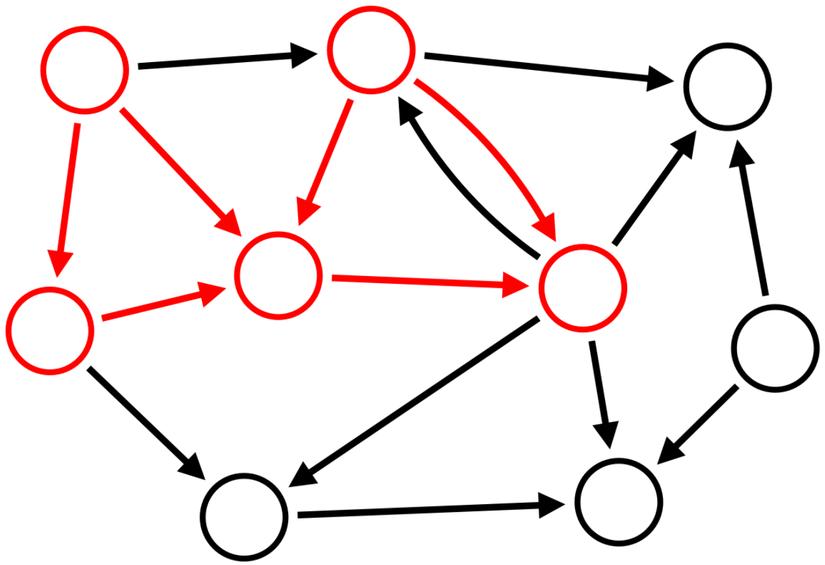
NODE NODE

```
MATCH (:Person { name:"Dan"} ) -[:KNOWS]-> (who:Person)
```

LABEL PROPERTY LABEL

```
RETURN who
```



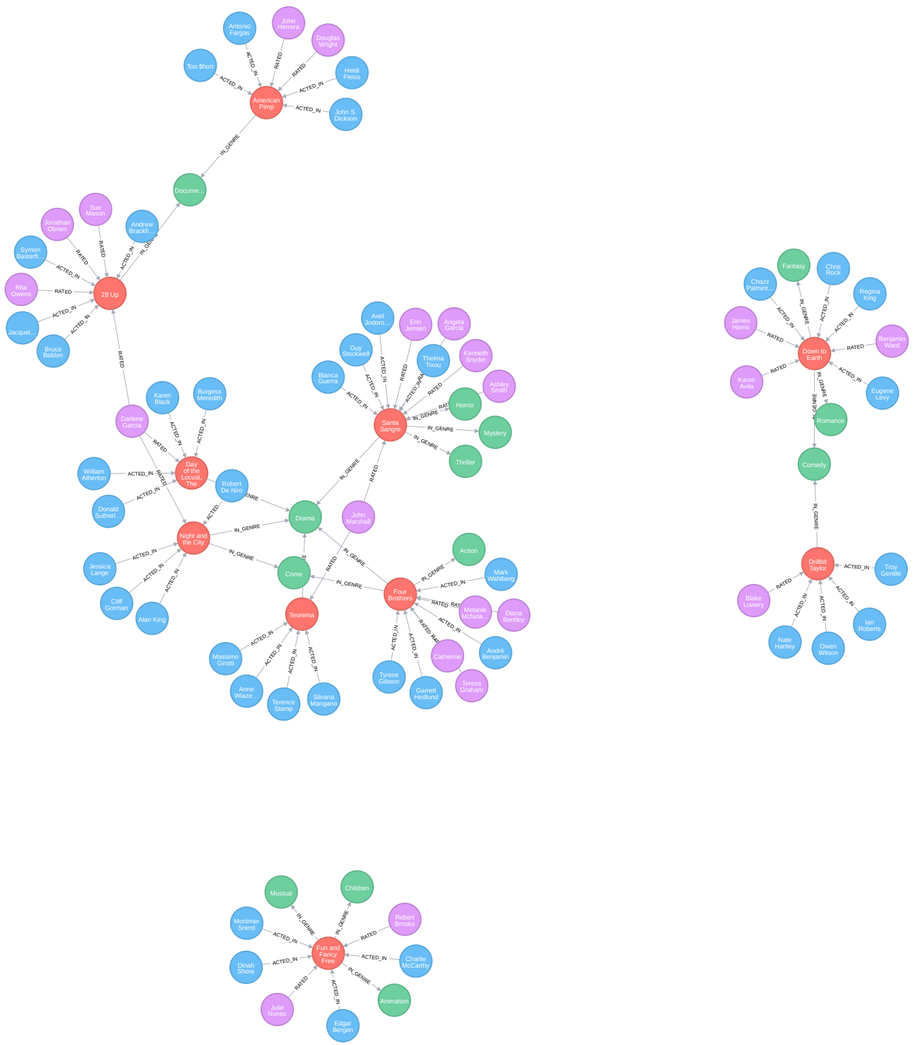


The movie dataset

https://neo4j.com/sandbox-v2/

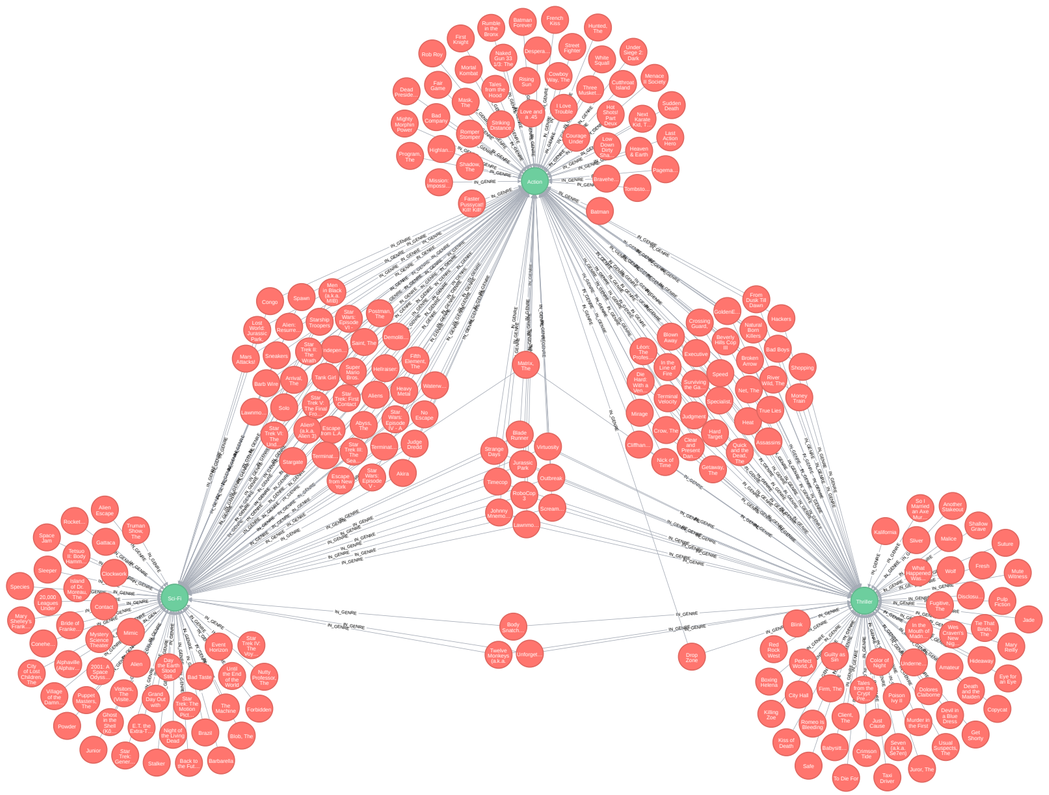
Recommendations

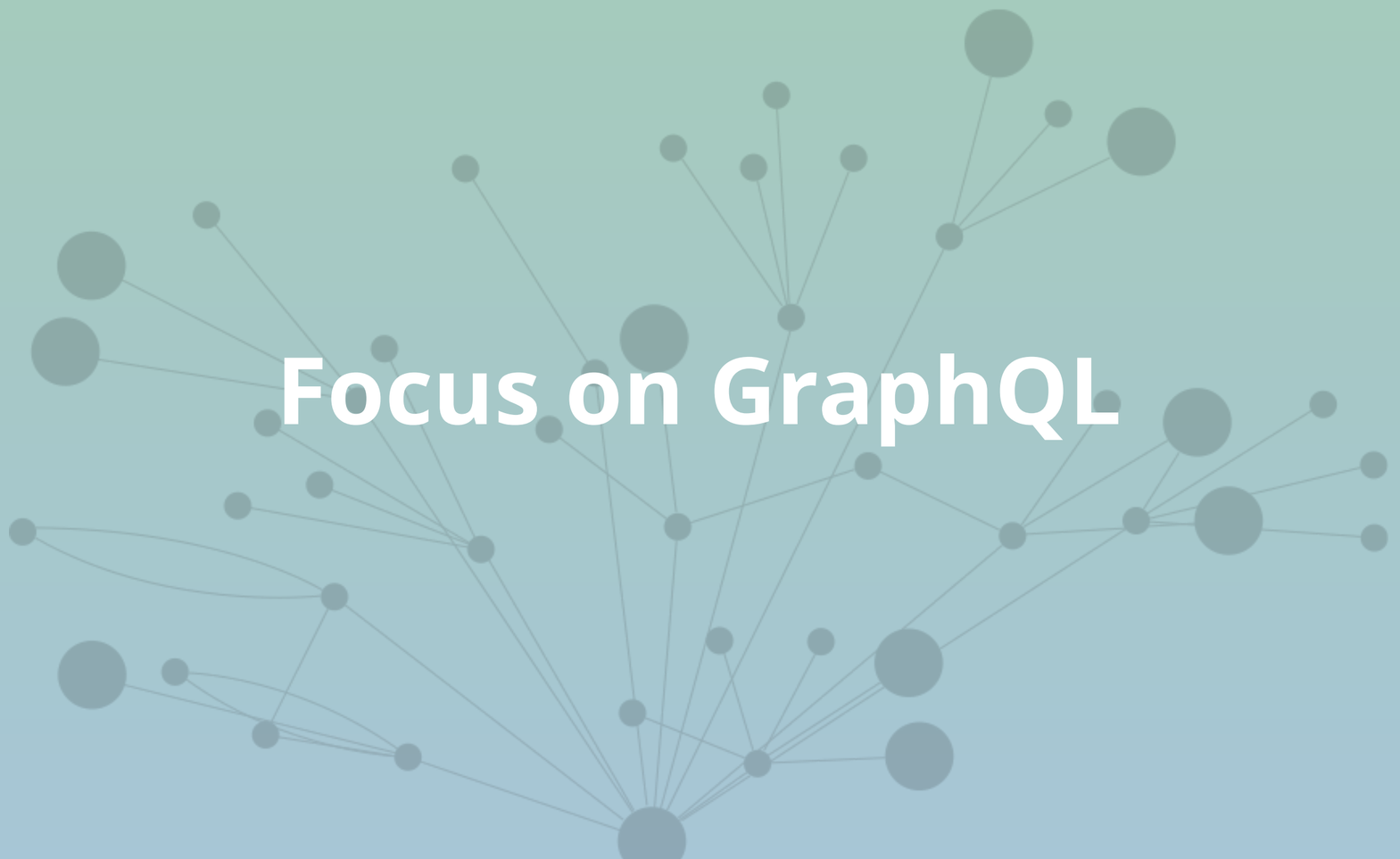
Generate personalized real-time recommendations using a dataset of movie reviews.



Recommendation

```
MATCH
  (m:Movie {title:'Matrix, The'}),
  (m)-[:IN_GENRE]->(:Genre)<-[:IN_GENRE]-(movie:Movie)
WITH m, movie, COUNT(*) AS genreOverlap

  MATCH (m)<-[:RATED]-(:User)-[:RATED]->(movie)
  WITH movie, genreOverlap, COUNT(*) AS userRatedScore
  RETURN movie
  ORDER BY
    (0.9 * genreOverlap) + (0.1 * userRatedScore)
    DESC
  LIMIT 3
```





Focus on GraphQL

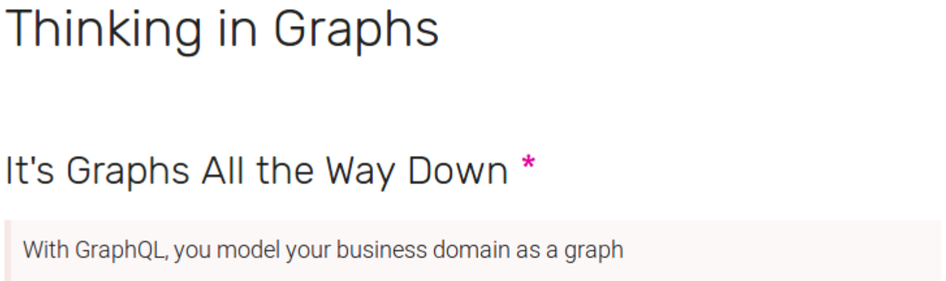
A large adoption

GraphQL

“A query language for your API, and a server-side runtime for executing queries by using a type system you define for your data”

“GraphQL isn’t tied to any specific database or storage engine”

“A GraphQL service is created by defining types and fields on those types, then providing resolver functions for each field on each type”



Your application model is a Graph !

```
query($id:ID = 2571) {
  Movie(movieId:$id) {
    movieId
    title
    plot
    poster
    imdbRating
    genres{
      name
    }
    directors {
      name
    }
    actors {
      name
      actedIn {
        title
      }
    }
  }
}
```

Your application model is a Graph !

```
query($id:ID = 2571) {
  Movie(movieId:$id) {
    movieId
    title
    plot
    poster
    imdbRating
    genres{
      name
    }
    directors {
      name
    }
    actors {
      name
      actedIn {
        title
      }
    }
  }
}
```

```
{
  "Movie": {
    "movieId": "2571",
    "title": "Matrix, The",
    "plot": "A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.",
    "poster": "http://ia.media-imdb.com/images/M/MV5BMTkxNDYxOTA4M15BMl5BanBnXkFtZTgwNTk0NzQxMTE@._V1_SX300.jpg",
    "imdbRating": 8.7,
    "genres": [ { "name": "Thriller" }, { "name": "Sci-Fi" }, { "name": "Action" } ],
    "directors": [ { "name": " Lana Wachowski" }, { "name": "Andy Wachowski" } ],
    "actors": [
      {
        "name": "Keanu Reeves",
        "actedIn": [ { "title": "John Wick" }, { "title": "47 Ronin" }, ...]
      },
      ...
    ]
  }
}
```

Schema Definition Language

GraphQL Cheat Sheet

- **Type :** The graph model definition
- **Resolvers :** How to fetch data from datasource
- **Query :** What queries you can do
- **Mutations :** What changes you can do

Type

Movie

```
type Movie {
  movieId: ID!
  title: String
  released: Date
  plot: String
  poster: String
  imdbRating: Float
  actors: [Actor]
  recommendations(skip: Int = 0, limit: Int =
5): [Movie]
}
```

Actor

```
interface Person {
  name: ID!
}

type Actor implements Person {
  name: ID!
  actedIn(skip: Int = 0, limit: Int = 5):
[Movie]
}
```

Queries / Mutation

Movie

```
type Query {
  Movie(movieId: ID!): Movie
  MovieSearch(search: String = "", skip: Int = 0, limit: Int = 10): [Movie]
}

type Mutation {
  MovieMutation(movieId: ID, title: String!, actors:[String], ...): Movie
}
```

Actor

```
type Query {
  Actor(id: ID!): Actor
  ActorSearch(search:String!): [Actor]
}

type Mutation {
  ActorMutation(name:String!): Actor
}
```

Resolvers

```
Movie {

  recommendations: async ( current, _, context ) => {
    let result = await Neo4j.run( context.driver.session(), queries.RECO, current, Neo4j.mappingNodeN );
    return result;
  }

},

Query: {

  Movie( root, params, context, resolveInfo ) {
    let result = await Neo4j.run( context.driver.session(), queries.GET, params, Neo4j.mappingNodeN );
    return result;
  },

  MovieSearch: async ( root, params, context, resolveInfo  ) => {
    let result = await Neo4j.run( context.driver.session(), queries.SEARCH, params, Neo4j.mappingNodeN );
    return result;
  },

}
```

Use The Schema, Luke

neo4j-graphql-js

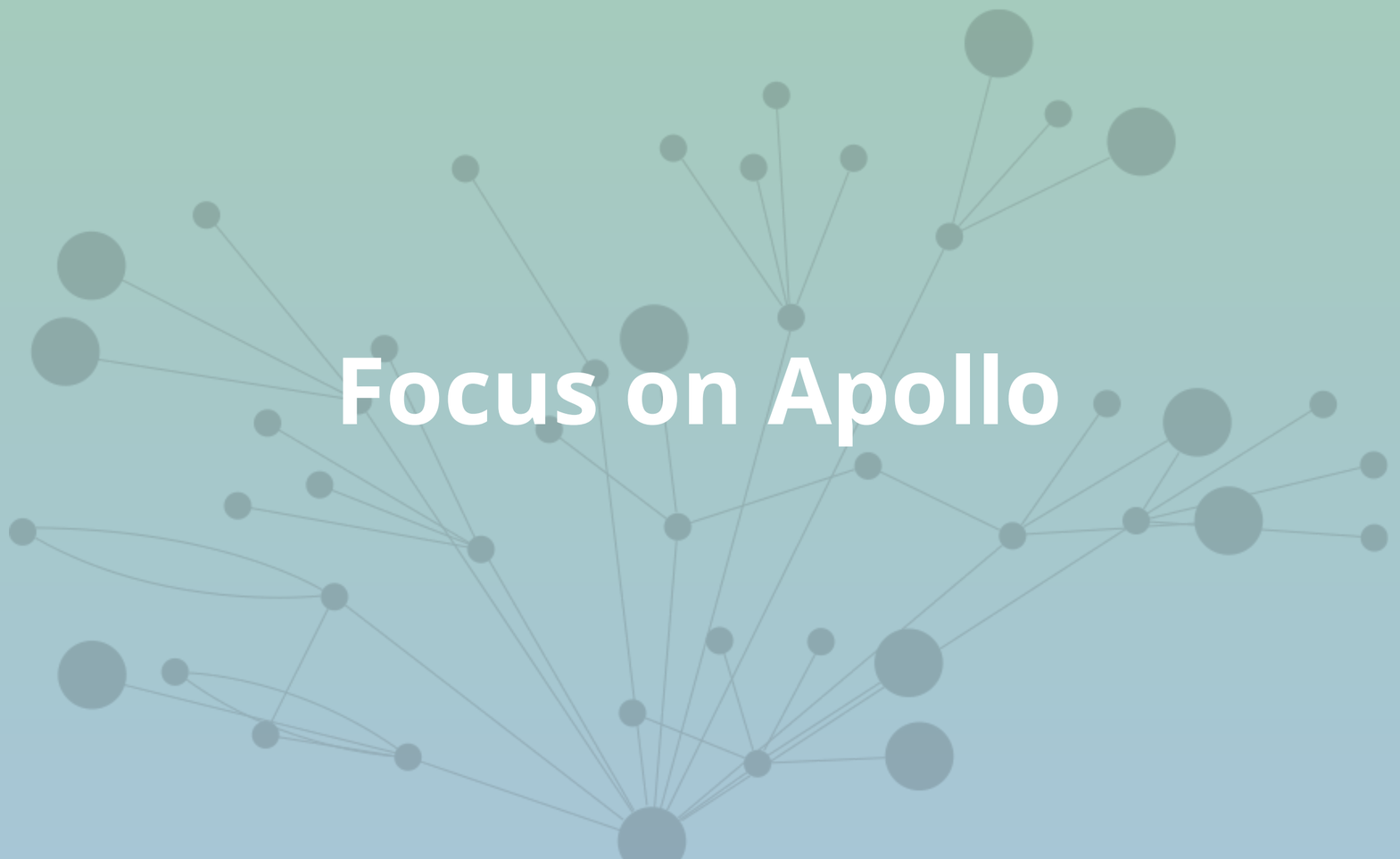
Translate your schema directly in Cypher, thanks to annotation.

```
{
  Movie(title: "River Runs Through It, A") {
    title
    year
    imdbRating
    actors {
      name
    }
  }
}
```

```
MATCH (movie:Movie {title:"River Runs Through It, A"})
RETURN movie { .title , .year , .imdbRating,
  actors: [(movie)<-[ACTED_IN]-(movie_actors:Actor) | movie_actors { .name }] }
AS movie
SKIP 0
```

openCypher

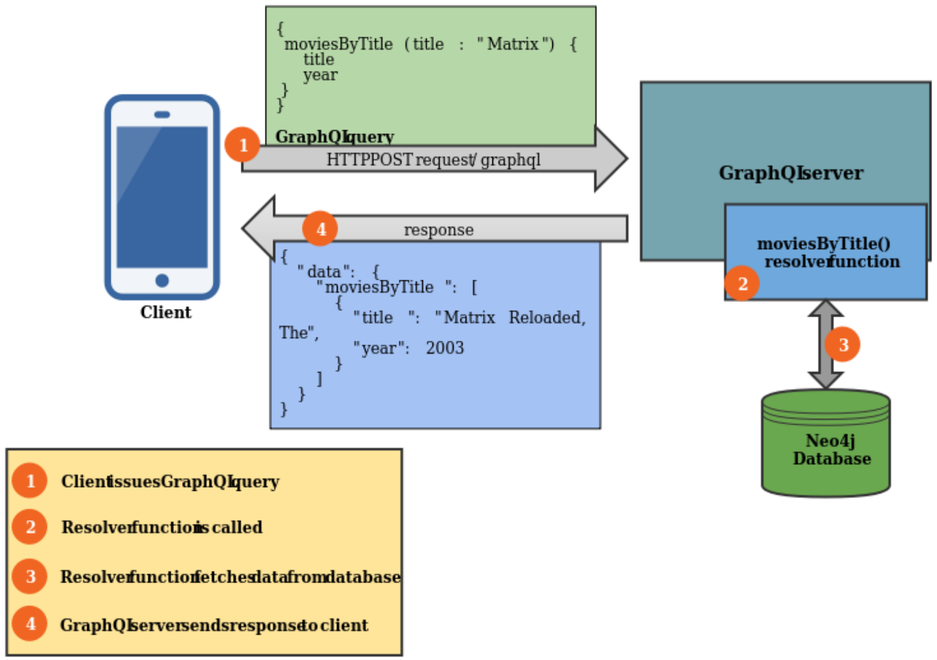
```
type Movie {
    movieId: ID!
    title: String
    year: Int
    plot: String
    poster: String
    imdbRating: Float
    genres: [String]
    similar(first: Int = 3, offset: Int = 0): [Movie] @cypher(statement: "WITH {this} AS this MATCH (this)--(:Genre)--(o:Movie) RETURN o")
    mostSimilar: Movie @cypher(statement: "WITH {this} AS this RETURN this")
    degree: Int @cypher(statement: "WITH {this} AS this RETURN SIZE((this)--())")
    actors(first: Int = 3, offset: Int = 0): [Actor] @relation(name: "ACTED_IN", direction:"IN")
    avgStars: Float
    filmedIn: State @relation(name: "FILMED_IN", direction: "OUT")
}
```



Focus on Apollo

GraphQL tool box

- **Server :** Apollo server helps you to create GraphQL server
- **Client:** Apollo client helps you to communicate with a GraphQL server



Server

Just an express.js server with some custom endpoints.

```
import express from 'express';
import { graphqlExpress, graphiqlExpress } from 'apollo-server-express';
import bodyParser from 'body-parser';

const PORT = 3000;
const server = express();

/**
 * GraphQL endpoint
 */
server.use( '/graphql', bodyParser.json(), graphqlExpress( async ( request ) => {  return {
  // your graphql schema
  schema: schema,
  // some context for the endpoint (here the Neo4j driver)
  context: Neo4j.context( request.headers, process.env )
}}));

/**
 * Create the graphiql endpoint.
 */
server.use( '/graphiql', graphiqlExpress( { endpointURL: '/graphql', query:'' }));

/**
 * Run the server.
 */
server.listen( PORT, () => { console.log( `GraphQL Server is now running on http://localhost:${PORT}/graphql` ) } );
```

Server - II

```
/**
 * This part is usefull for angular.
 * It enable CORS + OPTIONS request for the graphql endpoint.
 */
server.use(
  '/graphql',
  ( req, res, next ) => {
    res.header( 'Access-Control-Allow-Credentials', true );
    res.header( 'Access-Control-Allow-Headers', 'content-type, authorization, content-length, x-requested-with, accept, origin' );
    res.header( 'Access-Control-Allow-Methods', 'POST, GET, OPTIONS' );
    res.header( 'Allow', 'POST, GET, OPTIONS' );
    res.header( 'Access-Control-Allow-Origin', '*' );

    if ( req.method === 'OPTIONS' ) {
      res.sendStatus( 200 );
    } else {
      next();
    }
  }
);
```

Client - Angular - I

In your application module :

```
// Apollo & GraphQl
import { HttpModule } from '@angular/http';
import { HttpClientModule, HttpClient } from '@angular/common/http';
import { ApolloModule, Apollo } from 'apollo-angular';
import { HttpLinkModule, HttpLink } from 'apollo-angular-link-http';
import { InMemoryCache } from 'apollo-cache-inmemory';


@NgModule({
  declarations: [ AppComponent ],
  imports: [ ...,  HttpClientModule, ApolloModule, HttpLinkModule, ... ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule {

  constructor( apollo: Apollo, httpLink: HttpLink ) {
    apollo.create({
      // By default, this client will send queries to the
      // `/graphql` endpoint on the same host
      cache: new InMemoryCache()
    });
  }

}
```

Client - Angular - II

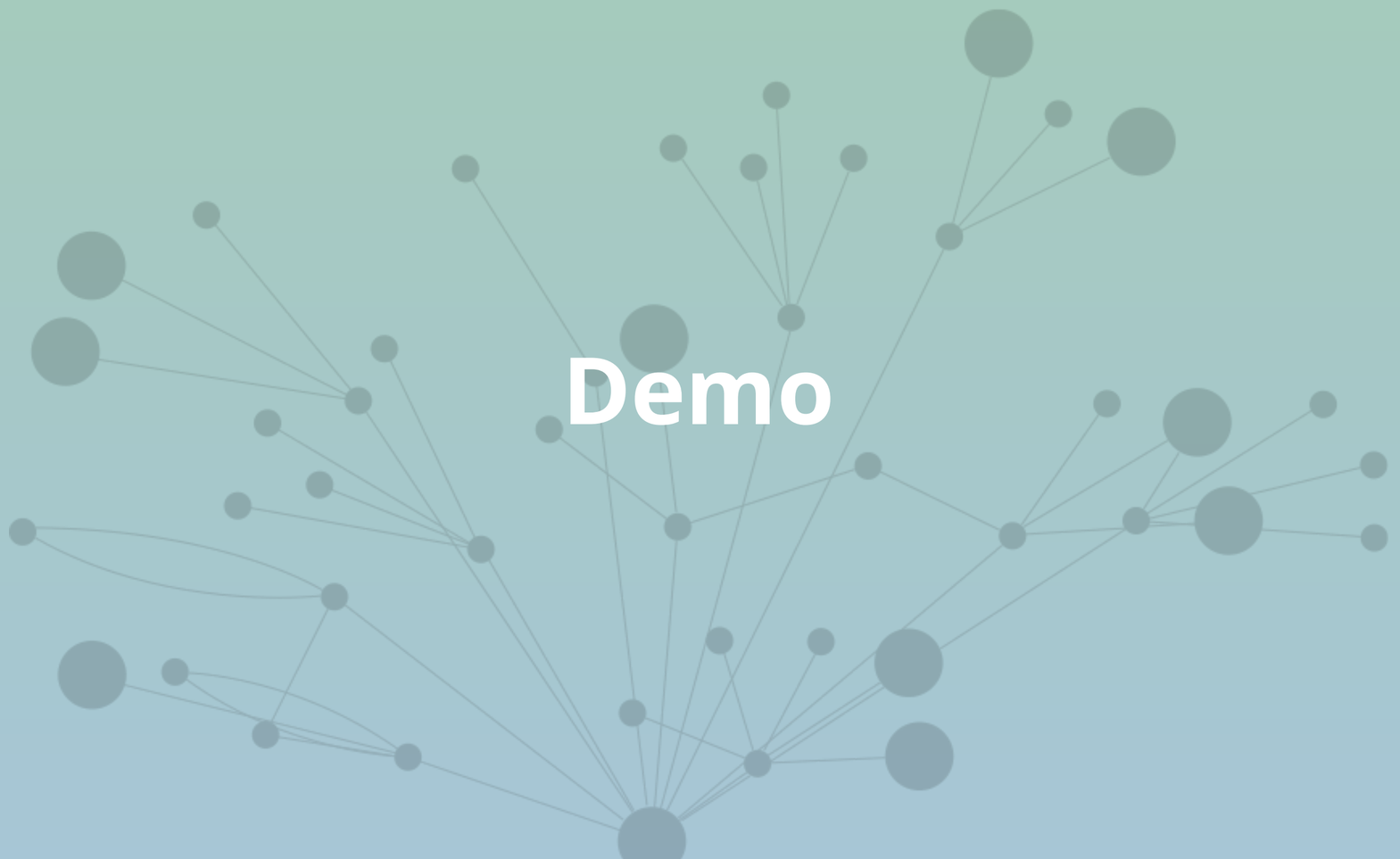
In your components :

```
import { Apollo } from 'apollo-angular';

export class MovieSearchComponent implements OnInit {

  // GraphQL query
  query :any;

  // Graphql result
  movies :[Movie];

  // search fields
  search :string = '';
  limit :number = 20;
  offset :number = 0;

  /**
   * Default constructor.
   * @param  {Apollo} apollo - Apollo service
   */
  constructor(private apollo :Apollo) {
    // Nothing
  }

  ngOnInit() {
    this.query = this.apollo.watchQuery(
      {
```

Typescript generation

https://github.com/apollographql/apollo-codegen

```
cd ~/worspaces/grandstack/grand-movies-examples/
apollo-codegen introspect-schema http://localhost:3000/graphql --output schema.json
cd front-angular/src/app/movie/search/
apollo-codegen generate ./search.graphql.ts --schema ~/worspaces/grandstack/grand-movies-examples/schema.json --target
typescript --output search.graphql.type..ts
```



Demo

Any questions ?

Thanks

Resource

Thanks to Will Lyon from Neo4j for the help