


```
type Project {
```

```
  name:  GraphQL
```

```
  tagline: "A query language for your  
API"
```

```
  contributors: ["Phil", "Sarah",  
                 "Patrick", "Perry"]
```

```
}
```

Project (name: GraphQL) {

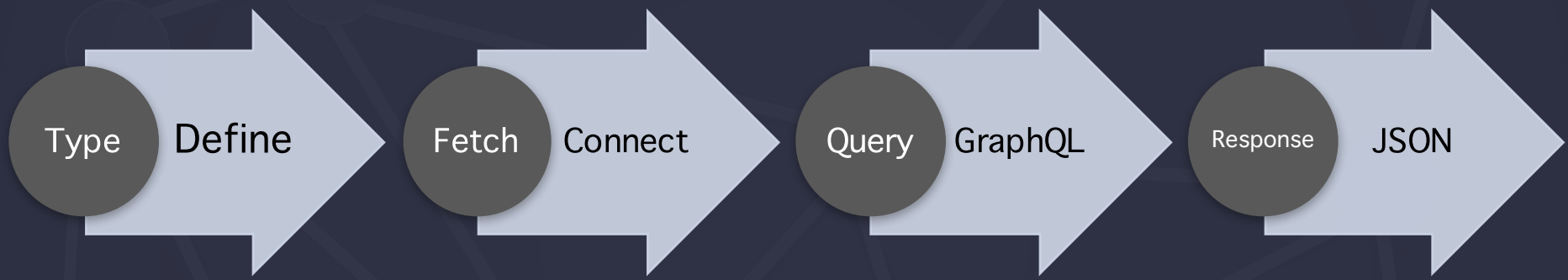
What is GraphQL and why is it useful?

}

- Created by Facebook in 2012
- Query language that serves as an interface between the client and server
- Data retrieval and manipulation
- Non-technical once configured
- Great for analysts

```
GET user/123?include=friend.name&friend.location
{
  user(id: 123) {
    name
    location
    friend {
      name
      location
    }
  }
}
```

Project (name: GraphQL) { Language structure }



```
type Query {  
  me: User  
}  
  
type User {  
  id: ID  
  name: String  
}
```

```
function Query_me(request) {  
  return request.auth.user;  
}  
  
function User_name(user) {  
  return user.getName();  
}
```

```
{  
  me {  
    name  
  }  
}
```

```
{  
  "me": {  
    "name": "Luke Skywalker"  
  }  
}
```

Project (name: GraphQL) { GraphQL and SQL }

sql-to-graphql

- Generate GraphQL schemas and server from table structure
- Give server credentials to your database

```
carl:sql-to-graphql ozarkp$ node cli.js --database "cs397" --port 5432 --user "ozarkp" --password "ozarkp" --backend "postgres" --output-dir "cs397"  
Demo app generated in /home/students/ozarkp/node_modules/sql-to-graphql/cs397. To run:  
cd /home/students/ozarkp/node_modules/sql-to-graphql/cs397  
npm install  
npm start
```

Then point your browser at <http://localhost:3000>

Project (name: GraphQL) {

GraphQL and SQL



}

Column	Type	Modifiers
abbreviation	character varying(5)	not null default nextval('degrees_id_seq'::regclass)
description	character varying(100)	
id	integer	not null default nextval('degrees_id_seq'::regclass)

Column	Type	Modifiers
id	integer	not null default nextval('majors_id_seq'::regclass)
name	character varying(100)	
department	character varying	

```

type Degree {
  abbreviation: String!
  description: String
  id: Int!
}

type Major {
  id: Int!
  name: String
  department: String
}

type RootQueryType {
  student(id: Int!): Student
  major(id: Int!): Major
  degree(id: Int!): Degree
  studentstodegree(id: Int!): Studentstodegree
  studentstomajor(id: Int!): Studentstomajor
}

type Student {
  id: Int!
  firstname: String
  lastname: String
  graduationYear: Int
}

type Studentstodegree {
  id: Int!
  degreeid: Int!
}

type Studentstomajor {
  id: Int!
  majorid: Int!
}

```

SQL Schema to GraphQL Schema

Column	Type	Modifiers
id	integer	not null default nextval('students_id_seq'::regclass)
firstname	character varying	
lastname	character varying	
graduation_year	integer	

Table "public.studentstodegrees"		
Column	Type	Modifiers
studentid	integer	not null
degreeid	integer	not null

Table "public.studentstomajors"		
Column	Type	Modifiers
studentid	integer	not null
majorid	integer	not null

Project (name: GraphQL) { GraphQL and SQL }

RootQueryType

- Defines entry points into the GraphQL schema
- Each query must start with one of the listed attributes

```
type RootQueryType {  
  student(id: Int!): Student  
  major(id: Int!): Major  
  degree(id: Int!): Degree  
  studentstodegree(id: Int!): Studentstodegree  
  studentstomajor(id: Int!): Studentstomajor  
}
```

sql-to-graphql

- Elementary method of wrapping SQL database
- Purposed for an introduction to GraphQL
- Defaults to providing client access to one table at a time

Project (name: GraphQL) {

GraphQL and SQL

}

- GraphQL does permit predetermined access across multiple tables
- Let's say you want to grant the client the ability to access students by graduation year

Defined in schema

```
getStudentMajorsByYear(_, args)
{
  return sql.raw(
    'SELECT firstname,lastname,name AS major
     FROM students,majors,studentstomajors
     WHERE students.id =
studentstomajors.studentid      AND
studentstomajors.majorid = majors.id
     AND graduation_year = %s', args);
}
```

Query

```
{
  getStudentMajorsByYear(
    graduationYear : 2017)
  {
    firstname
    lastname
  }
}
```


Project (name: GraphQL) { GraphQL and NoSQL (MongoDB) }

- Create server on NodeJS using express-graphql
 - express-graphql was created by Facebook to link Express and GraphQL
 - Processes HTTP requests and returns JSON responses

```
// app.js
import express from 'express';
import graphqlHTTP from 'express-graphql';
import mongoose from 'mongoose';

import schema from './graphql';

var app = express();

// GraphQL server route
app.use('/graphql', graphqlHTTP(req => ({
  schema,
  pretty: true
})));
```


Project (name: GraphQL) { GraphQL and NoSQL (MongoDB) }

Create Schema

- **Mongoose Model**
 - Mongoose is a Node.js library that provides MongoDB object mapping
 - Define fields and data types
- GraphQL types create objects for each field
- Mongoose model and GraphQL types create the schema

```
// graphql/index.js
import {
  GraphQLObjectType,
  GraphQLSchema
} from 'graphql';

import mutations from './mutations';
import queries from './queries';

export default new GraphQLSchema({
  query: new GraphQLObjectType({
    name: 'Query',
    fields: queries
  }),
  mutation: new GraphQLObjectType({
    name: 'Mutation',
    fields: mutations
  })
});
```

Project (name: GraphQL) { GraphQL and NoSQL (MongoDB) }

Mutation Schema

- Similar to a query schema
 - Define GraphQL types
 - Implement asynchronous callback to facilitate multi-threaded environment (NodeJS)
 - POST with `.save()`

```
// graphql/mutations/blog-post/add.js
import {
  GraphQLNonNull,
  GraphQLBoolean
} from 'graphql';

import blogPostInputType from '../../../types/blog-post-input';
import BlogPostModel from '../../../models/blog-post';

export default {
  type: GraphQLBoolean,
  args: {
    data: {
      name: 'data',
      type: new GraphQLNonNull(blogPostInputType)
    }
  },
  async resolve (root, params, options) {
    const blogPostModel = new BlogPostModel(params.data);
    const newBlogPost = await blogPostModel.save();

    if (!newBlogPost) {
      throw new Error('Error adding new blog post');
    }
    return true;
  }
};
```

Project (name: GraphQL) { GraphQL and REST API }

- Wrapping a JSON response
 - Create a new Query Object
 - Replace JSON type fields with GraphQL type equivalent
 - Create resolving functions for each data field

```
text: <string>  --> text: GraphQLString  
count: <integer> --> count: GraphQLInt  
follow: <bool>  --> follow: GraphQLBoolean
```

Project (name: GraphQL) { GraphQL and REST API }

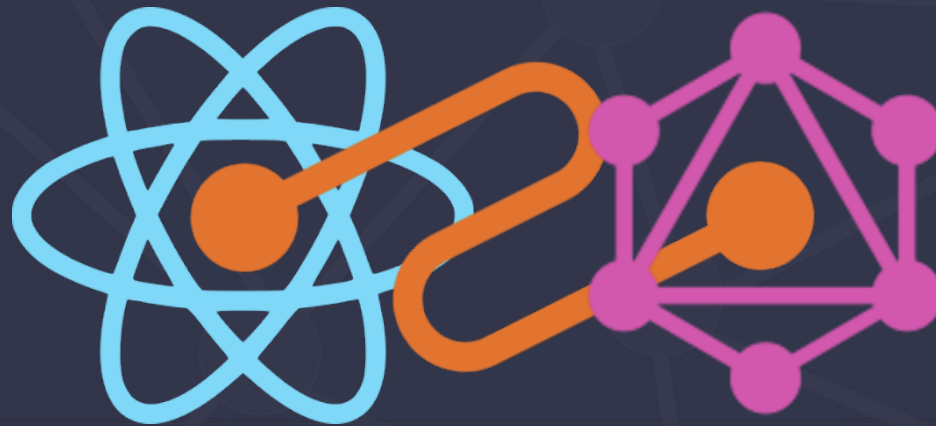
Creating Types

```
{  
  "Address": {  
    "street": <string>  
    "number": <integer>  
  }  
}
```

```
var AddressType = new GraphQLObjectType({  
  name: 'Address',  
  fields: {  
    street: { type: GraphQLString },  
    number: { type: GraphQLInt },  
    formatted: {  
      type: GraphQLString,  
      resolve(obj) {  
        return obj.number + ' ' + obj.street  
      }  
    }  
  }  
});
```

```
Project (name:  GraphQL) {  
  GraphQL and REST API  
}
```

Connecting and Visualizing – Data Exploration



Project (name: GraphQL) { GraphQL and REST API (SWAPI) }

- swapi-graphql
- Connect to StarWars API (SWAPI) through GraphQL
- Download from: <https://github.com/graphql/swapi-graphql.git>

```
Last login: Thu Apr  6 20:22:40 on ttys000
adminisatorsair:~ administrator$ cd Desktop/swapi-graphql-master/
adminisatorsair:swapi-graphql-master administrator$ npm install
adminisatorsair:swapi-graphql-master administrator$ npm start

> swapi-graphql@ start /Users/administrator/Desktop/swapi-graphql-master
> npm run download && babel-node src/server/main.js

> swapi-graphql@ download /Users/administrator/Desktop/swapi-graphql-master
> sh scripts/download

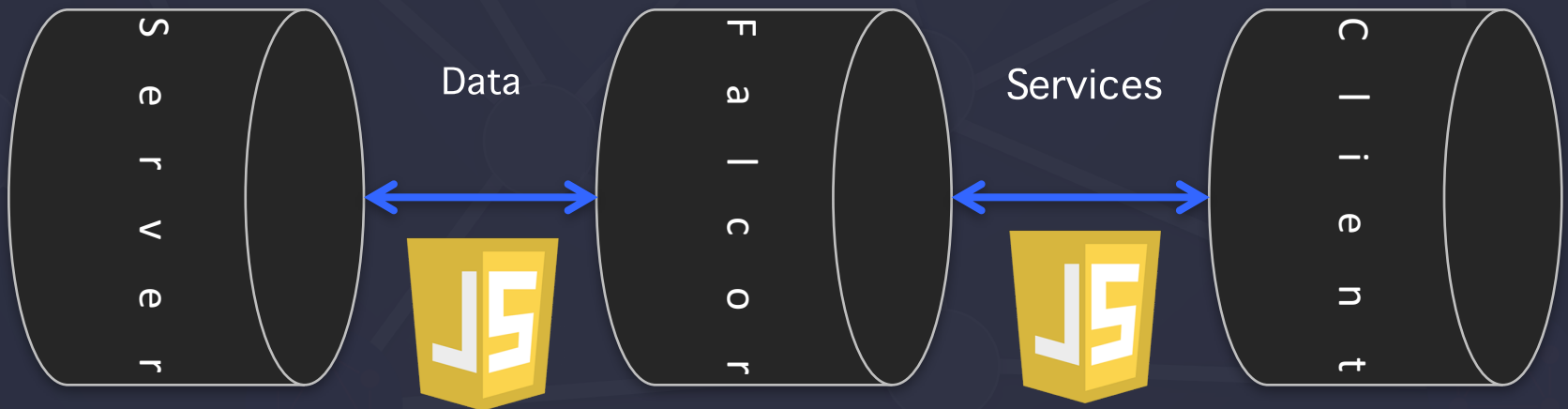
Listening at http://localhost:56457
```


Project (name: FALCOR) {

Falcor

}

- Created by Netflix in 2015
- Serves as an interface between client and server
- Aimed at managing the increasingly complex data requirements of modern applications



Project (name:  GraphQL) {

Falcor

}



GraphQL

- NOT a query language for graph databases
- Declarative, strongly typed application-level query language
- Allows for defining models for data and mapping data-model to backend



FALCOR

- All data is represented in a singular JSON model
- Merges requests and templates to fetch data, cached in the client
- Stored in JSON, runs on Falcor server
- Fetching routes defined as needed

```
Project (name:  GraphQL) {  
  GraphQL – looking forward  
}
```

- Unifies desperate data sources through one interface
- Growing commercial presence, lacks open-source following
- Needs time to mature

bazinga!

