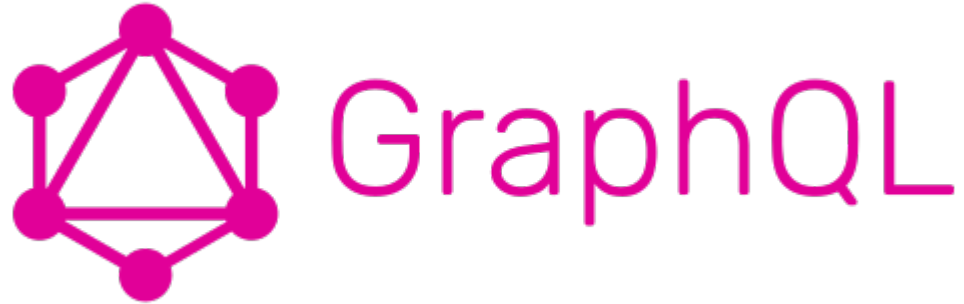


# SQL 활용

김보성

## SQL이란

구조적 쿼리 언어(SQL)는 관계형 데이터베이스에 정보를 저장하고 처리하기 위한 프로그래밍 언어입니다. 관계형 데이터베이스는 정보를 표 형식으로 저장하며, 행과 열은 다양한 데이터 속성과 데이터 값 간의 다양한 관계를 나타냅니다. SQL 문을 사용하여 데이터베이스에서 정보를 저장, 업데이트, 제거, 검색 및 검색할 수 있습니다. 데이터베이스 성능을 유지 관리하고 최적화하는 데 SQL을 사용할 수도 있습니다.



## GraphQL 이란

그래프QL은 페이스북이 2012년에 개발하여 2015년에 공개적으로 발표된 데이터 질의어이다. 그래프QL은 REST 및 부속 웹서비스 아키텍처를 대체할 수 있다. 클라이언트는 필요한 데이터의 구조를 지정할 수 있으며, 서버는 정확히 동일한 구조로 데이터를 반환한다. 그래프QL은 사용자가 어떤 데이터가 필요한 지 명시할 수 있게 해 주는 강타입 언어이다. 이러한 구조를 통해 불필요한 데이터를 받게 되거나 필요한 데이터를 받지 못하는 문제를 피할 수 있다.

주요 그래프QL 클라이언트로는 아폴로 클라이언트와 Relay 등이 있다. 그래프QL 서버는 여러 언어로 구현되어 있는데, 자바스크립트, 파이썬, 루비, 자바, C#, 스칼라, 고, 엘릭서, 얼랭, PHP, 클로저 등의 언어로 구현되어 있다.

데이터를 다루기 위해 graphql로 데이터를 다루는것을 설명하도록 하겠습니다.

```
문제   출력   디버그 콘솔   터미널

GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
$ npm init -y
█
```

1. 데이터작업을 위한 폴더를 열고 npm init -y 명령어로 package.json 파일을 생성합니다 .

```
GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
$ npm install

up to date, audited 1 package in 57ms

found 0 vulnerabilities
```

2. .npm install 명령어를 입력해 package-lock.json파일을 만들어 환경을 구축해줍니다.

```
GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
$ npm i -g nodemon
[#####.....] - reify:fsevents: sill reify mark deleted [
```

3. 데이터베이스의 자동 변화 감지를 위해 nodemon을 설치 해줍니다.

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "nodemon index_source.js"
}
```

4. package.json scripts 에 들어가서 "start":"nodemon index\_source.js" 를 설정 해줍니다.

```
GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
$ npm i graphql apollo-server
[#####.....] / idealTree:apollo-server-express: sill fetch manifest @types/node@*
```

5. graphql과 apollo-server를 동시에 설치 해줍니다.

```
GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
$ npm i convert-csv-to-json

added 1 package, and audited 2 packages in 345ms

found 0 vulnerabilities

GGG@DESKTOP-8N3GG40 MINGW64 /d/김보성/SQL_활용/GraphQL
```

6. 엑셀파일인 csv파일 변환기도 설치 해줍니다.

```

database.js > [?] <unknown>
const csvToJson = require('convert-csv-to-json')

const database = {
  teams: [],
  people: [],
  roles: [],
  softwares: [],
  equipments: [],
  supplies: []
}
Object.keys(database).forEach((key) => {
  database[key] = [
    ...database[key],
    ...csvToJson.fieldDelimiter(',')
      .getJsonFromCsv(`./data_files/${key}.csv`)
  ]
  if (database[key].length > 0) {
    const firstItem = database[key][0];
    Object.keys(firstItem).forEach((itemKey) => {
      if (database[key].every((item) => {
        return /^-?\d+$/.test(item[itemKey])
      })) {
        database[key].forEach((item) => {
          item[itemKey] = Number(item[itemKey])
        })
      }
    })
  }
})

module.exports = database

```

```

index_source.js > ...
const database = require('./database')
const { ApolloServer, gql } = require('apollo-server')

const typeDefs = gql`
  type Query {
    teams: [Team]
  }
  type Team {
    id: Int
    manager: String
    office: String
    extension_number: String
    mascot: String
    cleaning_duty: String
    project: String
  }
`

const resolvers = {
  Query: {
    teams: () => database.teams
  }
}

const server = new ApolloServer({ typeDefs, resolvers })

server.listen().then(({ url }) => {
  console.log(`Server ready at ${url}`)
})

```

7. database.js로 csv파일의 데이터를 가져와서 index.js파일에 연결시킬수 있도록 작업을 진행하고 경로나 require/module.exports가 확실하게 되어있는지 확인 해줍니다 .

```
$ npm start

> graphql@1.0.0 start
> nodemon index_source.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index_source.js`
Server ready at http://localhost:4000/
█
```

8. npm start 를 입력하여 노드몬을 실행시키면 아폴로서버의 로컬호스트 주소와함께 graphql 을 사용할수있게 됩니다.

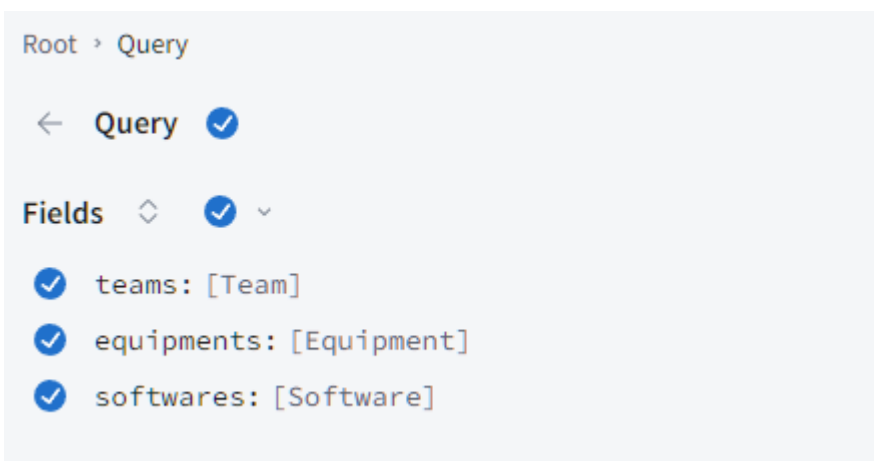


```
const typeDefs = gql`
  type Query {
    teams: [Team]
    equipments:[Equipment]
    softwares:[Software]
  }
  type Team {
    id: Int
    manager: String
    office: String
    extension_number: String
    mascot: String
    cleaning_duty: String
    project: String
  }
  type Equipment{
    id:String
    used_by:String
    count:Int
    new_or_used:String
  }
  type Software{
    id:String
    used_by:String
    developed_by:String
    description:String
  }
`
```

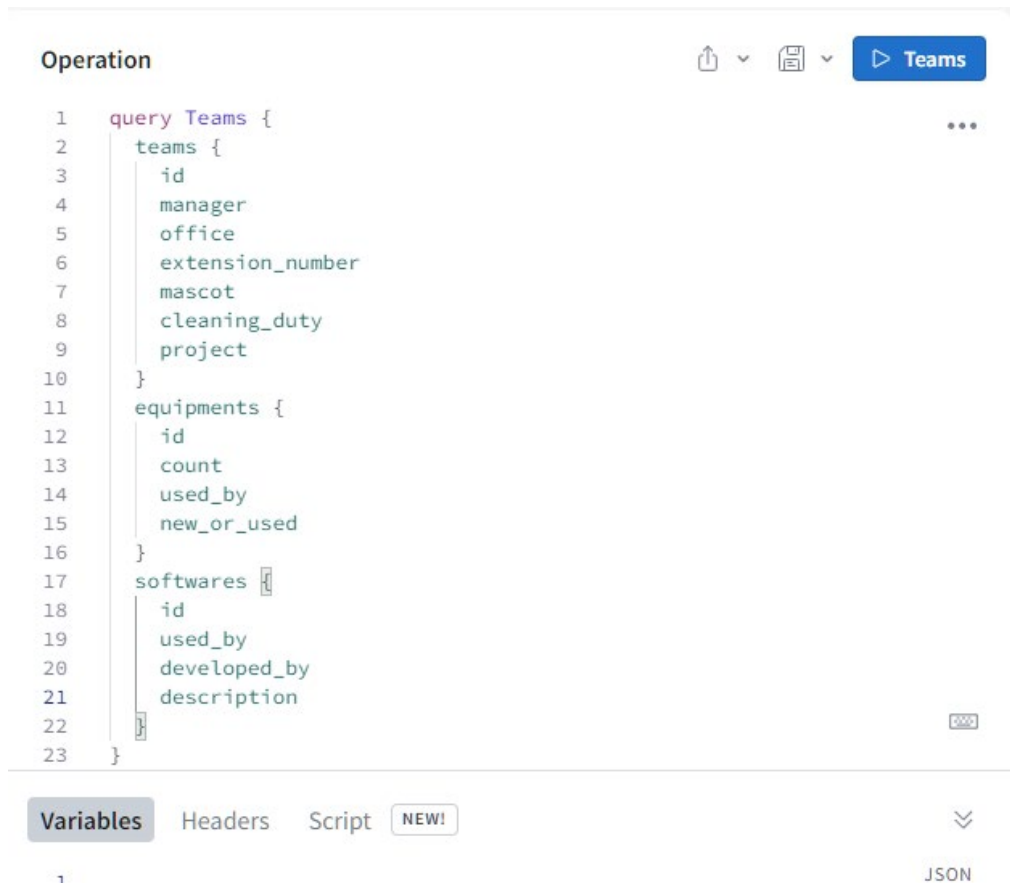
9. equipment.csv와 software.csv를 가져와서 사용하기 위해 스키마를 정의 해줍니다.

```
const resolvers = {  
  Query: {  
    teams: () => database.teams,  
    equipments: ()=> database.equipments,  
    softwares: ()=>database.softwares,  
  }  
}
```



10. 정의 해놓은 스키마를 resolvers 함수 쿼리에 정의해줍니다 .








11. 쿼리와 필드에서 필요한 데이터들을 체크해줍니다.



12. Operation 안에 본인이 체크한 데이터 필드가 들어와있는것을 확인한후 Teams 버튼(실행)을 누릅니다

Response   STATUS 200 | 16.0ms

Root > data  

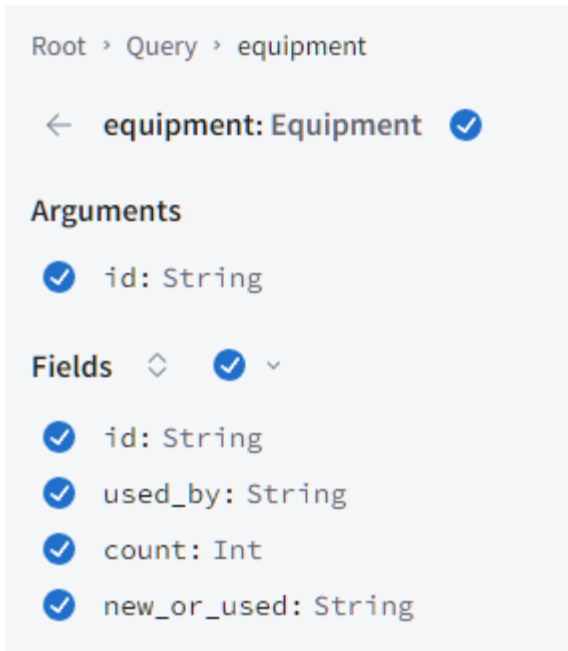
teams 							
	id	manager	office	extension_number	mascot	cleaning_duty	proj
0	1	Mandy Warren	101A	#5709	Panda	Monday	Hype
1	2	Stewart Grant	101B	#4012	Tadpole	Tuesday	Zen
2	3	Smantha Wheatly	102A	#3852	Falcon	Wednesday	Dura
3	4	Francis Buckley	103B	#1039	Beaver	Thursday	Geng
...reveal 1 rows ▼							
equipments 							
	id	used_by	count	new_or_used			
0	machanical keyboard		developer	24	used		
1	pen tablet		designer	15	used		
2	notebook		planner	37	new		
3	ergonomic mouse		designer	31	used		
...reveal 3 rows ▼							
softwares 							
	id	used_by	developed_by	description			
0	Eclipse	developer	Eclipse Foundation	integrated development environment			
1	Excel	planner	Microsoft	spreadsheet			
2	Illustrator	designer	Adobe	vector graphics editor			

13. Response에 데이터들이 확인되는것을 볼수있습니다.

```
const typeDefs = gql`  
  type Query {  
    teams: [Team]  
    equipments:[Equipment]  
    softwares:[Software]  
    equipment(id:Int):Equipment  
  }  
`
```

```
const resolvers = {  
  Query: {  
    teams: () => database.teams,  
    equipments: ()=> database.equipments,  
    softwares: ()=>database.softwares,  
    equipment:(parent,args,context,info)=>database.equipments  
      .filter((equipment)=>{  
        return equipment.id === args.id  
      })[0]  
  }  
}
```

14. Equipments안에서 id를 이용하여 아폴로서버 안에서 조회를 하기위해 스키마에 함수를 정의해주고 리졸버에 filter함수를 설정하여 줍니다.



15. 스키마와 리졸버함수에서 필터함수를 적용시킨 equipment 를 체크하고 arguments(입력값) 과 필드를 체크합니다.

## Operation



▶ Equipment

...

```
1 query Equipment($equipmentId: String) {  
2   equipment(id: $equipmentId) {  
3     id  
4     used_by  
5     count  
6     new_or_used  
7   }  
8 }
```

16. 오퍼레이션에 정의한 함수에 따라 조회되는 항목 등등이 표시됩니다.

17. Variables에 자신이 조회하고 싶은 id값을 입력해줍니다.

JSON

## Variables

Headers

Script

NEW!

⌵

```
1 {  
2   "equipmentId": "notebook"  
3 }
```

JSON

Response

STATUS 200 | 16.0ms | 921

Root > data > equipment

JSON

id	notebook
used_by	planner
count	37
new_or_used	new

18. Notebook 이라는 아이디를 가진 데이터가 조회되는것을 볼수있습니다.



```
type Equipment{  
  id:String  
  used_by:String  
  count:Int  
  new_or_used:String  
  softwares:[Software]  
}
```

```
const resolvers = {  
  Query: {  
    teams: () => database.teams,  
    equipments: () => database.equipments  
      .map((equipment)=>{  
        equipment.softwares = database.softwares  
          .filter((software)=>{  
            return software.used_by === equipment.used_by  
          })  
        return equipment  
      })  
  }  
}
```

19. Equipment 쿼리 안에 software 필드를 추가하기 위해 스키마를 정의해주고 used\_by 를 기준으로 조회를 하기 위해 함수를 작성해줍니다.

## Operation

```
1  query Equipments {  
2    equipments {  
3      used_by  
4      softwares {  
5        id  
6        used_by  
7        developed_by  
8        description  
9      }  
10   }  
11 }
```



▶ Equipments

...

Response ▾ ≡ 📄						STATUS 200   10.0ms   2.4		
Root ▸ data ▸ equipments						📄 📄 CSV 📄 JSON		
	used_by	softwares						
0	developer		id	used_by	developed_by	description		
		0	Eclipse	developer	Eclipse Foundation	integrated development environment		
		1	VS Code	developer	Microsoft	source code editor		
		2	Xcode	developer	Apple	integrated development environment		
1	designer		id	used_by	developed_by	description		
		0	Illustrator	designer	Adobe	vector graphics editor		
		1	Photoshop	designer	Adobe	raster graphics editor		
		2	Sketch	designer	Sketch B.V.	vector graphics editor		
2	planner		id	used_by	developed_by	description		
		0	Excel	planner	Microsoft	spreadsheet		
		1	PowerPoint	planner	Microsoft	presentation program		
		2	Word	planner	Microsoft	word processor		

20. 오퍼레이션에 알맞게 추가를 해준뒤 실행 해보면 used\_by를 기준으로 항목들이 조회되는것을 알수있습니다.

```

    type Mutation{
      insertPeople(
        id:Int
        first_name:String
        last_name:String
        sex:String
        blood_type:String
        serve_years:Int
        role:String
        team:Int
        from:String
      ):People
      deletePeople(id:Int):People
      editPeople(
        id:Int
        first_name:String
        last_name:String
        sex:String
        blood_type:String
        serve_years:Int
        role:String
        team:Int
        from:String
      ):People
    }
    type People{
      id:Int
      first_name:String
      last_name:String
      sex:String
      blood_type:String
      serve_years:Int
      role:String
      team:Int
      from:String
    }
  }
}

```

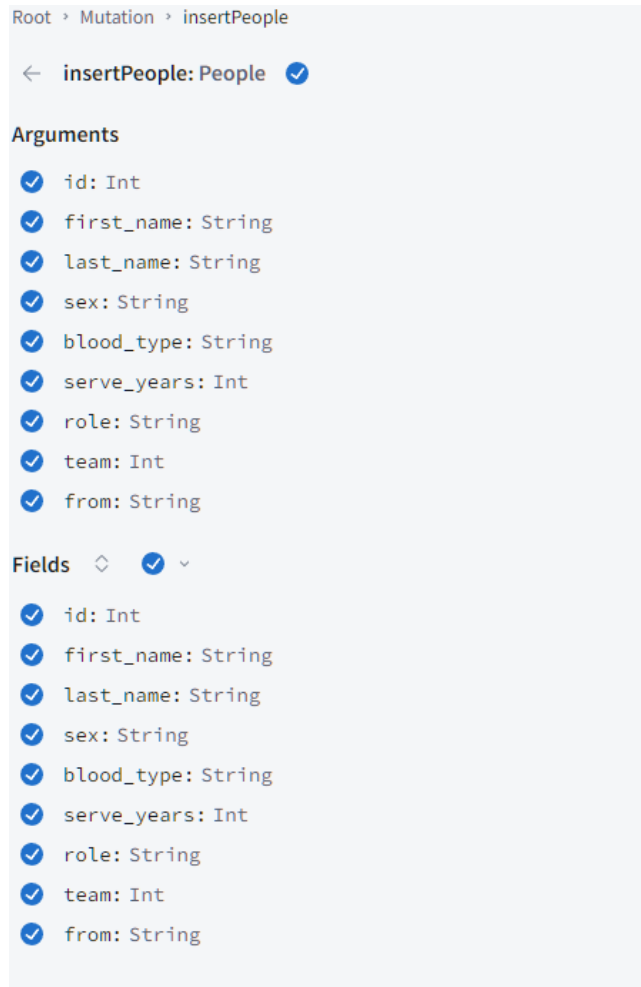
21. Peoples.csv 파일을 가져와서 삭제,삽입,수정을 하기위해 스키마에 정의해주고 mutation이라는 타입을 설정해 함수를 정의해줍니다.

```

Mutation:{
  insertPeople:(parent,args,context,info)=>{
    database.peoples.push(args)
    return args
  },
  deletePeople:(parent,args,context,info)=>{
    const deleted = database.peoples
      .filter((people)=>{
        return people.id === args.id
      })[0]
    database.peoples = database.peoples
      .filter((people)=>{
        return people.id !== args.id
      })
    return deleted
  },
  editPeople:(parent,args,context,info)=>{
    return database.peoples
      .filter((people)=>{
        return people.id === args.id
      }).map((people)=>{
        Object.assign(people,args)
        return people
      })[0]
  }
}

```

22. Mutation 삽입,삭제,수정 함수를 작성 해줍니다



23. 데이터 삽입입니다 . 아폴로서버에 접속해 Root>Mutation>insertPeople 에 들어가 Arguments 와 Fields 를 모두 체크해주고 Variables 에 삽입하고 싶은 정보를 형식에 맞게 입력후 실행을 눌러줍니다.

46	47	Leroy	Elliott	male	AB	2	developer
47	48	Barbara	Murphy	female	O	1	developer
48	49	Simon	Henderson	male	A	4	designer
49	50	Ned	Butler	male	O	2	planner
50	1	Jake	Kim	mail	A	1	developer

24. Qeury 에 들어와 peoples에서 필드를 체크한후 실행 시키면 제일 밑에 (id중복) 새로운 데이터가 들어온것을 확인할수있습니다.

← editPeople: People ✓

**Arguments**

- ✓ id: Int
- + first\_name: String
- + last\_name: String
- + sex: String
- ✓ blood\_type: String
- + serve\_years: Int
- + role: String
- + team: Int
- + from: String

**Fields** ⚙️ - ▾

- + id: Int
- + first\_name: String
- + last\_name: String
- + sex: String
- ✓ blood\_type: String
- + serve\_years: Int
- + role: String
- + team: Int
- + from: String

Operation 🔗 ▾ 📄 ▾ ▶ Query

```

1  mutation EditPeople($editPeopleId: Int, $bloodType: String) {
2    editPeople(id: $editPeopleId, blood_type: $bloodType) {
3      id
4      blood_type
5    }
6  }
7
8  query Query {
9    peoples {
10     id
11     first_name
12     last_name
13     sex
14     blood_type
15     serve_years
16     role
17     team
18     from
19   }
20 }

```

**Variables** Headers Script NEW! ⌵

```

1  {
2    "editPeopleId": 10,
3    "bloodType": "RH-O"
4  }

```

JSON

9	10	Tyler	Philips	male	RH-O	1	designer
---	----	-------	---------	------	------	---	----------

25. 데이터 수정입니다. 뮤테이션에서 에디트 함수를 체크한후 arguments 에서 id 와 blood\_type을 체크한후 필드도 같이 체크를 해준후 variables 에서 형식에 맞게 수정을 한후 query에 들어가서 peoples의 필드를 체크후 실행하면 10번째( index 9) 타일러의 혈액형이 RH-O 로 바뀐것을 볼수있습니다.

Root › Mutation › deletePeople

← deletePeople: People ✓

Arguments

✓ id: Int

Fields ⚡ -

✓ id: Int

+ first\_name: String

+ last\_name: String

+ sex: String

+ blood\_type: String

+ serve\_years: Int

+ role: String

+ team: Int

+ from: String

Variables Headers Script NEW!

```
1 {  
2   "deletePeopleId": 1  
3 }
```

Response ▾ ≡

Root › data › peoples STATUS 200 | 13.0ms

CSV JS

	id	first_name	last_name	sex	blood_type	serve_years	role
0	2	Lindsay	West	female	A	3	designer
1	3	Nathan	Jenkins	male	B	1	planner

26.삭제입니다. 뮤테이션에서 arguments 에서 id fields에서 아이디를 체크한후 실행을 누르고 query에 들어가 peoples의 모든 필드를 체크한후 실행하면 데이터가 삭제 된것을 확인 할수있습니다.