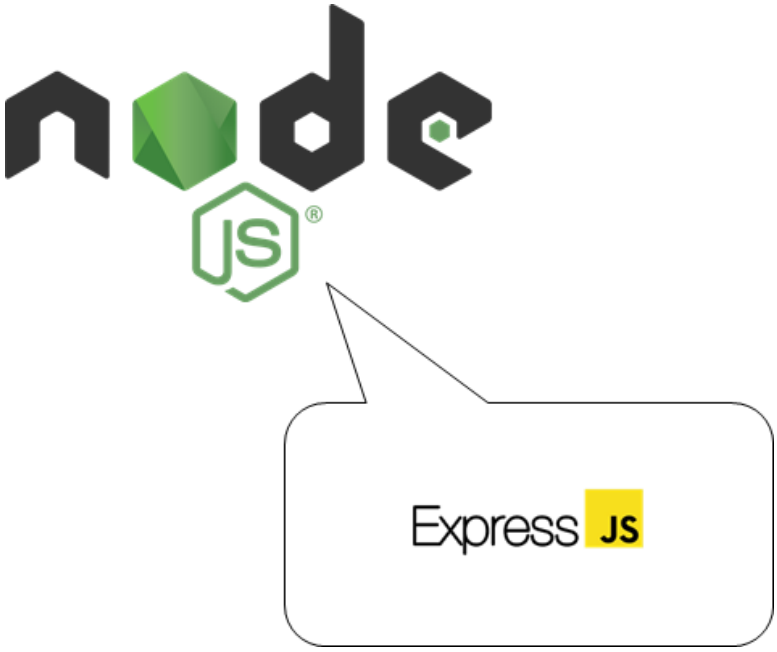
221228_2반_실습



Datebase

학생 비상연락망

일자	20 년 월 일					학교명			
번호	학년	반	성명	휴대폰	주소	비상연락처			
						휴대폰	관계	휴대폰	관계

NoSQL

NoSQL(Not Only SQL)

- 기존의 관계형 데이터베이스(Relational database, `RDBMS`)의 한계를 극복하기 위해 제안된 모델
  - `RDBMS`는 데이터 관계를 외래키 등으로 정의하고 JOIN 연산을 수행할 수 있지만, NoSQL은 JOIN 연산이 불가능
  - NoSQL - 수평적 확장성
    : 서버를 늘리기만 하면은 스케일이 늘어남. 그러면서 응답속도는 보장
  - RDBMS - 수직적 확장성
    : ssd, cpu 업그레이드 하거나, 확장하는데 돈과 손이 많이 듦
  - SQL(MySQL)과 NoSQL(MongoDB) 비교

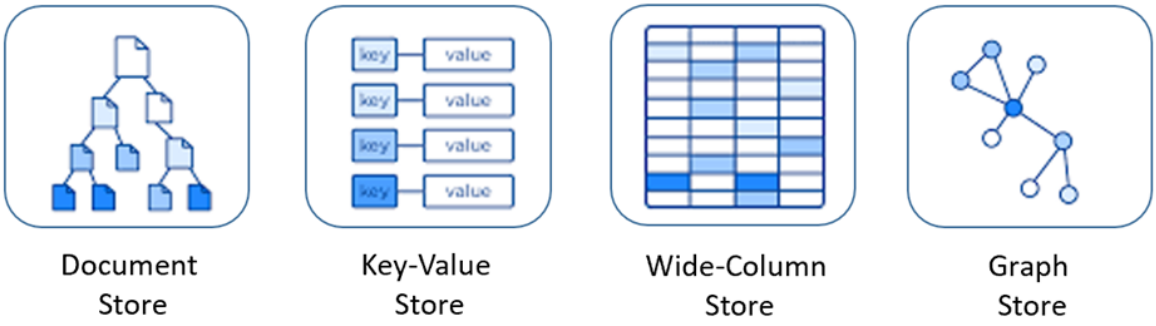
MySQL	MongoDB
database	database
table	collection
index	index
row	JSON document
column	JSON field (Key/Field)
join	Database Server & Client, Database Server & Client
primary key	_id field
group by	aggregation
mysqld	mongod
mysql	mongo

  - 대량의 분산된 데이터를 저장하고 조회하는데 특화
  - 대용의 데이터 저장 가능(페타바이트 급의 대용량)
  - 스키마 없이 혹은 유동적인 스키마를 제공
    - 데이터를 저장하는 칼럼이 각기 다른 이름과 다른 데이터 타입을 가지는 것이 허용
- NoSQL의 장단점

장점	단점

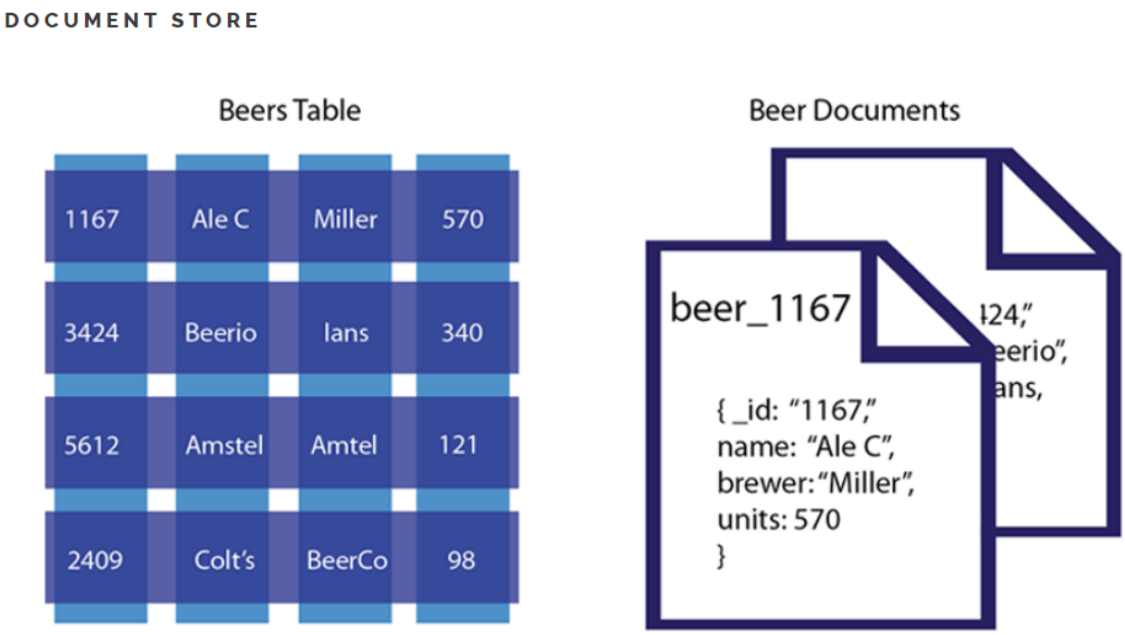
장점	단점
RDBMS에 비해 저렴한 비용으로 분산·병렬 처리 가능	데이터 업데이트 중 장애가 발생하면 데이터 손실 발생 가능
비정형 데이터 구조 설계로 설계 비용 감소	많은 인덱스를 사용하려면 충분한 메모리가 필요. 인덱스 구조가 메모리에 저장
Big Data 처리에 효과적	데이터 일관성이 항상 보장되지 않음
가변적인 구조로 데이터 저장이 가능	
데이터 모델의 유연한 변화가 가능	

- NoSQL의 종류



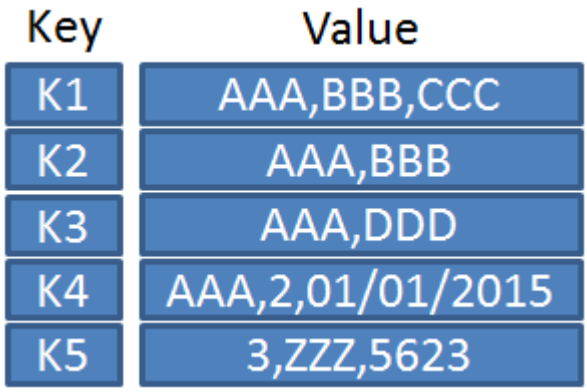
NoSQL의 4가지 일반적인 모델

1. `Document Database`



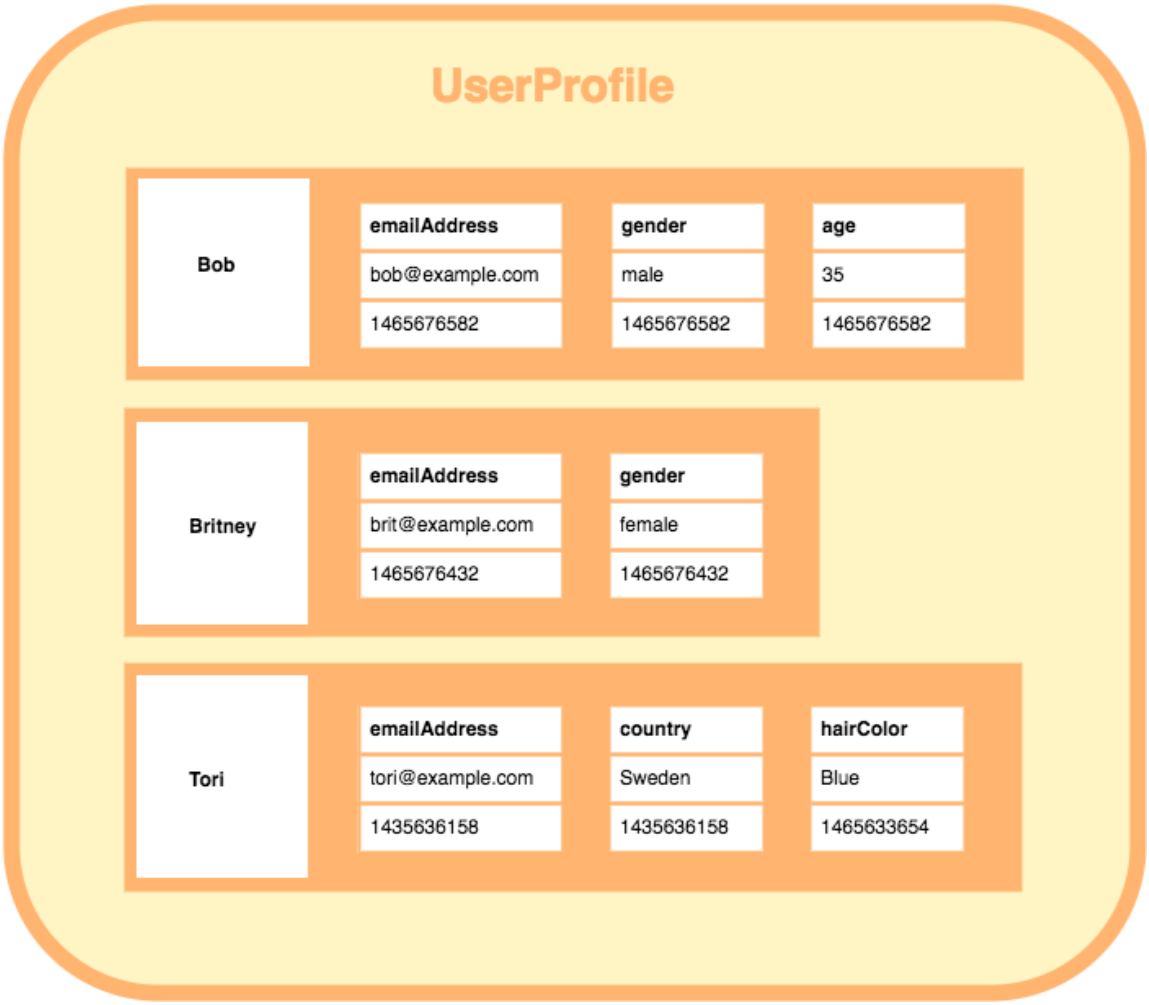
- 테이블의 스키마가 유동적(레코드마다 각각 다른 스키마를 가질 수 있음)
- 보통 XML, JSON과 같은 DOCUMENT를 이용해 레코드를 저장
- 트리형 구조로 레코드를 저장하거나 검색하는 데 효과적
- Ex) `MongoDB`, CouchDB, Azure Cosmos DB

2. Key-Value Database



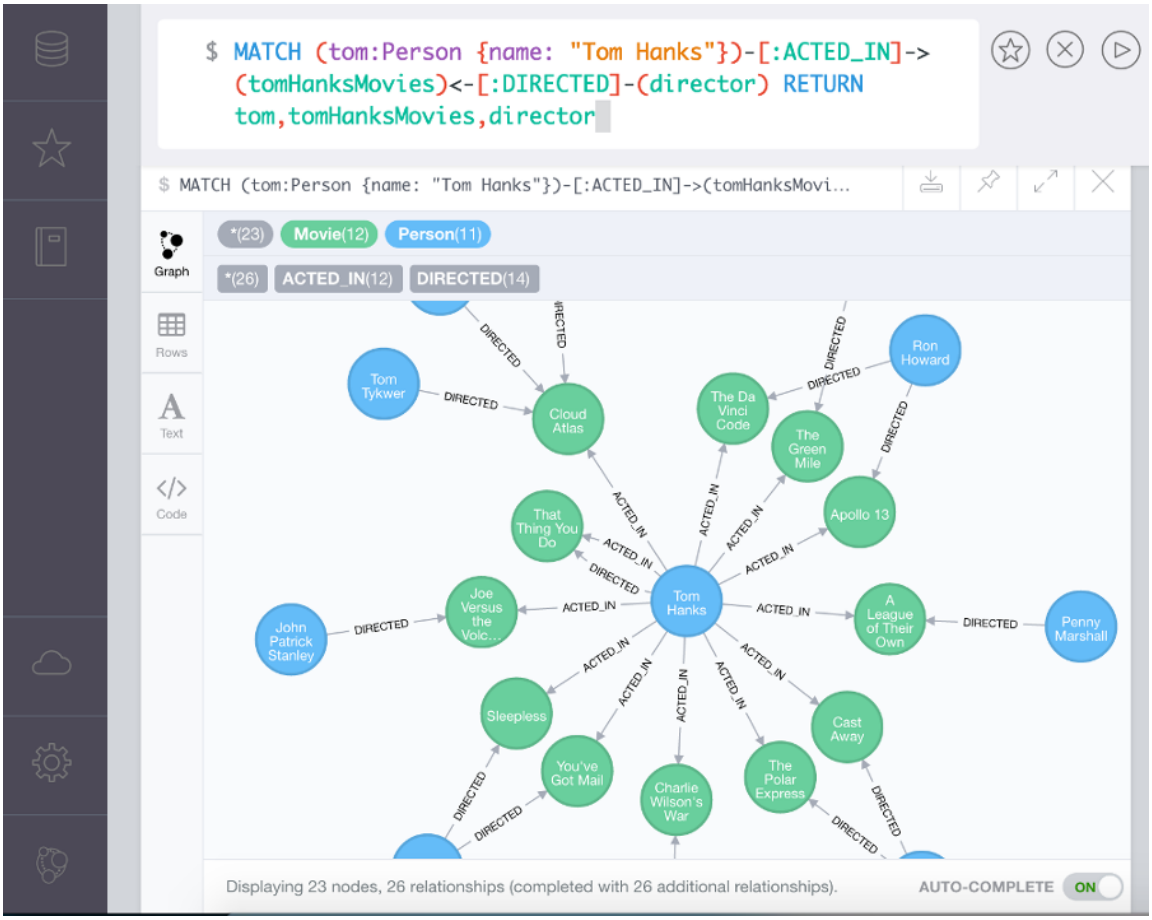
- 기본적인 패턴으로 KEY-VALUE 하나의 묶음(Unique)으로 저장되는 구조로 단순한 구조 → 속도가 빠르며 분산 저장시 용이
- Key 안에 (COLUMN, VALUE) 형태로 된 여러 개의 필드, 즉 COLUMN FAMILIES 구조
- 주로 SERVER CONFIG, SESSION CLUSTERING등에 사용(엑세스 속도는 빠르지만, SCAN에는 용이하지 않음)
- Ex) Dynamo, Redis, Oracle NoSQL Database, VoldeMorte

3. Wide-Column Database



- 행마다 키와 해당 값을 저장할 때마다 각각 다른값의 다른 수의 스키마를 가질 수 있음
    - 위 그림에서 사용자의 이름(key)에 해당하는 값에 스키마들이 각각 다름
- 대량의 데이터의 압축, 분산처리, 집계 쿼리 (SUM, COUNT, AVG 등)및 쿼리 동작 속도 그리고 확장성이 뛰어남
- EX) Apache Cassandra, Hbase, GoogleBigTable, Vertica

4. Graph Database



- 데이터를 노드로(그림에서 파란, 녹색 원) 표현
- 노드 사이의 관계를 엣지(그림에서 화살표)로 표현
- 일반적으로 RDBMS 보다 성능이 좋고 유연하며 유지보수에 용이
- Social networks, Network diagrams 등에 사용
- Ex) Neo4j, BlazeGraph, OrientDB

MongoDB

- C++로 작성된 오픈소스 문서지향(Document-Oriented)적 Cross-plaform 데이터베이스
- `NoSQL` , `Document Database`
- schema-less → 비 구조적
- `Document`
  - RDBMS의 record와 비슷한 개념
  - 데이터 구조는 한개 이상의 key-value pair로 이루어져 있음

- Ex)

```
{
    _id: 19643165,
    username: 'Dae-Yong Kim',
    email: 'daeyong@snoopy.com'
}
```

- 설치
  - Window
    - https://www.mongodb.com/try/download/enterprise
    - 참고 : https://joytk.tistory.com/m/74, https://khj93.tistory.com/m/entry/MongoDB-Window에-MongoDB-설치하기
  - MAC OS
    - Tap the MongoDB Homebrew Tap

```
brew tap mongodb/brew
```

    - Update Homebrew Tap

```
brew update
```

    - Install MongoDB

```
brew install mongodb-community
```

```
/usr/local/etc/mongod.conf        // Config 파일
/usr/local/var/log/mongdb         // log 파일
/usr/local/var/mongodb            // 데이터 파일
```

    - Start And Stop

```
brew services start mongodb-community     // start
brew services stop mongodb-community      // stop
```

```
mongod --config /usr/local/etc/mongod.conf --fork //백그라운드로 실행(Intel Chip)
mongod --config /opt/homebrew/etc/mongod.conf --fork //백그라운드로 실행(Apple Chip)
//mongosh를 통해 mongod와 연결한 후에 "shutdown" 명령어를 입력한다.
```

참고 : https://wildeveloperetrain.tistory.com/m/83, https://choboit.tistory.com/m/95

- Linux

참고 : https://velog.io/@seungsang00/Ubuntu-MongoDB-설치하기-Ubuntu-20.04, https://www.traderharu.com/2021/07/12/우분투20-ubuntu20-mongodb-설치/

- 명령어
  - MongoDB 실행 : mongo

```
mongo
```

  - Database 생성, 사용 : use [DB명]

```
use sample //없을 경우 생성
```

  - DB 확인 : db, show dbs
  - 현재 접속중인 DB 확인 : db

```
db
//결과로 DB 명 출력
```

  - Database 제거: db.dropDatabase()

```
use sample //제거할 DB 접속
db.dropDatabase(); //DB 삭제
```

  - Collection 생성: db.createCollection*(name, [options])*
    - createCollection()을 사용하지 않아도 Document 추가 시 자동으로 컬렉션 생성

```
db.sample.insert({"name": "snoopy"})
//collection 생성전 해당 콜렉션에 데이터를 생성할 경우 자동으로 collection 생성
```

- Options

Field	Type	설명
capped	Boolean	이 값을 true 로 설정하면 capped collection 을 활성화 시킵니다. Capped collection 이란, 고정된 크기(fixed size) 를 가진 컬렉션으로서, size 가 초과되면 가장 오래된 데이터를 덮어씁니다. 이 값을 true로 설정하면 size 값을 꼭 설정해야합니다.
autoIndex	Boolean	이 값을 true로 설정하면, _id 필드에 index를 자동으로 생성합니다. 기본값은 false 입니다.
size	number	Capped collection 을 위해 해당 컬렉션의 최대 사이즈 (maximum size)를 ~ bytes로 지정합니다.
max	number	해당 컬렉션에 추가 할 수 있는 최대 갯수를 설정합니다.

```
db.createCollection("user") //user collection을 옵션 없이 생성
//{ "ok" : 1 } ->결과

//article collection을 다음 옵션을 설정하여 생성
db.createCollection("articles", {
 capped: true,
 autoIndex: true,
 size: 6142800,
 max: 10000
})
```

- Collection 제거: db.COLLECTION_NAME.drop()

```
db.sample.drop()
//현재 접속중인 DB의 sample collection 삭제
```

- Document 추가: db.COLLECTION_NAME.insert(document)

```
db.sample.insert({"name": "Snoopy", "species": "Beagle"})
//현재 접속한 DB의 sample collection에 document 추가

db.sample.insert([
 {"name": "Pingu", "species": "Penguin"},
 {"name": "Ggambi", "species": "Russianblue"}
]);
//현재 접속한 DB의 sample collection에 여러개의 documents 추가
```

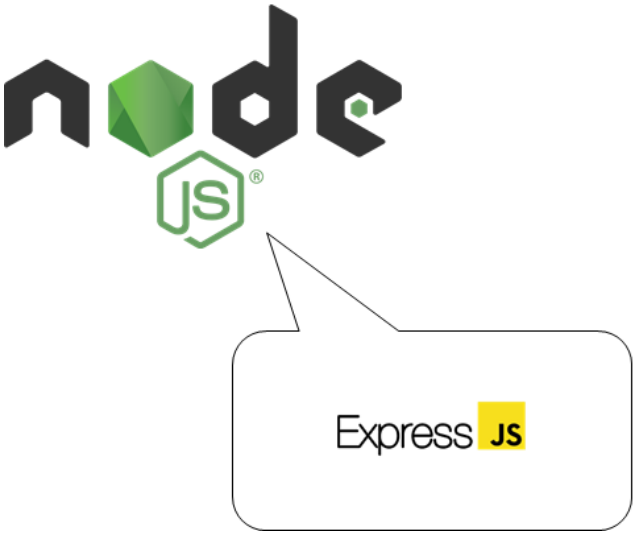
- Document 확인: db.COLLECTION_NAME.find()

```
db.books.find()
//collection의 document 리스트 출력
```

- Document 제거: db.COLLECTION_NAME.remove(criteria, justOne)

parameter	type	설명
*criteria	document	삭제 할 데이터의 기준 값 (criteria) 입니다. 이 값이 { } 이면 컬렉션의 모든 데이터를 제거합니다.
justOne	boolean	선택적(Optional) 매개변수이며 이 값이 true 면 1개 의 다큐먼트만 제거합니다. 이 매개변수가 생략되면 기본 값은 false 로 서, criteria에 해당되는 모든 다큐먼트를 제거합니다.

```
db.sample.remove({"name": "Snoopy"})
//현재 접속중인 DB의 sample collection에서 name이 Snoopy인 documents 삭제
```



NodeJS

Mongoose

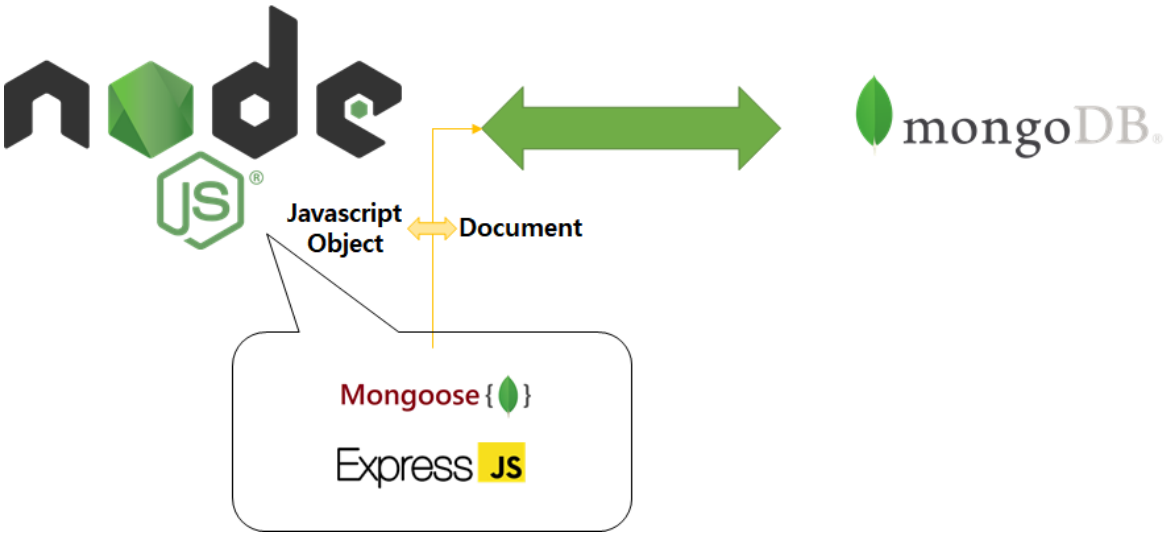
- Node.js와 MongoDB를 위한 `ODM` (Object Data Mapping) 라이브러리

ODM(Object Data Mapping)

- 말 그대로 객체와 문서를 1대1 매칭
- 문서를 DB에서 조회할 때 자바스크립트 객체로 바꿔주는 역할

- mongoose의 장점
  - 스키마와 모델 정의를 통해 data를 불러온 후, 그 데이터를 객체화시키는 것에 빠름
- mongoose 설치

```
npm install mongoose
```



Node js(Mongoose) ↔ MongoDB 연결

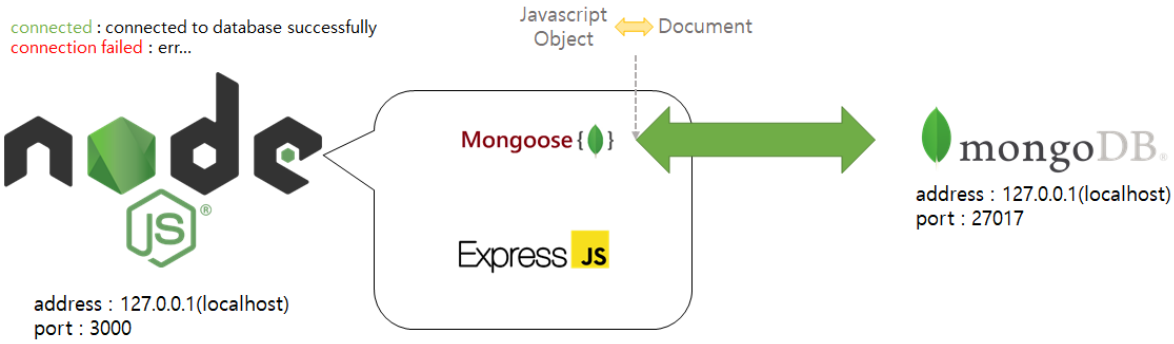
- express, mongoose 설치

```
npm install express mongoose
```

- server.js
  - mongoose를 통해 express(node js)와 mongoDB를 연결하고 성공 여부 확인
  - mongoDB의 default port number는 27017
  - localhost = 127.0.0.1 → 자기 자신의 주소

- 설정 값 : 웹 서버의 port(`port_number`), mongoDB 서버의 정보 (`mongo_connection_info`)

```
// Require express
const express = require('express')

// Require mongoose
const mongoose = require("mongoose");

// Initialize express
const app = express()

// parse json objects
app.use(express.json())

//set port number
const port_number = 8080

//set mongo connection information([mongoDB name]:[address]:[mongoDB port])
const mongo_connection_info = 'mongodb://localhost:27017'

// parse url encoded objects- data sent through the url
app.use(express.urlencoded({ extended: true}))

// create a server
const PORT = port_number
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT  }`)
  mongoose
  .connect(
    mongo_connection_info
  )
  .then(() => console.log('connected to database successfully'))
  .catch((err) => {
    console.log(err);
  });

});
```



Schema

- 스키마는 데이터베이스의 테이블, 컬렉션의 타입과 속성을 정의해 각각의 필드에 저장하는 값에 의존성을 부여합니다.
- **mongoDB에는 스키마가 없기 때문에 몽구스를 사용해서 스키마를 생성합니다.**
- **스키마는 컬렉션에 들어가는 문서 내부의 각 필드가 어떤 형식으로 되어 있는지 정의하는 객체이다.**
- 스키마를 만들 때 mongoose모듈은 Schema를 사용하여 정의한다.
- 각 필드 이름과 필드의 데이터 타입 정보가 들어 있는 객체를 작성한다.
- 필드의 기본값으로는 default 값을 설정하면 된다.
- 스키마 내부에 다른 스키마를 내장 시킬 수도 있다.
- 스키마 타입
  - `String`: 문자열
  - `Number`: 정수
  - `Schema.Types.ObjectId`: 명시적으로 id 타입을 사용할 때는 이렇게 사용해야 한다. 스키마에 정의를 안해도 자동적으로 몽고에서는 생성하며 행을 구분하는 아이디 값으로 사용된다.
  - `Date`: 날짜
  - `Buffer`: 바이너리 타입
  - `Boolean`: 참과 거짓만 값을 저장하고 나타낸다.
  - `Schema.Types.Mixed`: 이름 그대로 다양한 타입을 저장할 수 있다. 객첸 배열 JSON으로 사용합니다. 타입이 없다고 생각해도 된다.
  - `Array`: `[]`사용해서 배열을 표시 ex) `[Number]` 정수 배열
- 스키마 속성
  - `required`: 필수 입력
  - `unique`: 다른 행과 중복되면 안됨.
  - `trim`: 공백 제거(문자열 타입에 사용)
  - `default`: 문서가 생성되면 기본값으로 저장된다.
  - `lowercase`: 대문자를 소문자로 저장
  - `match`: 정규식으로 저장하려는 값과 비교
  - `validate`: 함수로 개발자가 조건을 만듦.

- `set`: 값을 입력할 때 함수로 조건을 만듦.
- `get`: 값을 출력할 때 함수로 조건을 만듦.
- `ref`: 해당하는 모델을 참조할 때 사용.

Model

- **모델은 스키마를 사용하여 만드는 인스턴스로, 데이터베이스에서 실제 작업을 처리할 수 있는 함수들을 지니고 있는 객체이다.**
- 모델을 만들 때는 mongoose.model 함수를 사용한다.
- model() 함수는 두 개의 파라미터를 필요로 한다.
- 첫 번째 파라미터는 스키마 이름이고, 두 번째 파라미터는 스키마 객체이다.
- 데이터베이스는 스키마 이름을 정해주면 그 이름의 복수 형태로 데이터베이스에 컬렉션 이름을 만든다.(스키마 이름이 'Post'이면 컬렉션은 'posts'로 생성)
- MongoDB에서 컬렉션 이름을 만들 때, 권장되는 컨벤션은 구분자를 사용하지 않고 복수 형태로 사용하는 것이다.
- 컨벤션 규칙을 따르고 싶지 않다면, model()함수의 세 번째 파라미터에 원하는 컬렉션 이름을 넣으면 된다.

Schema와 Model 예제

- 예제 코드

```
const mongoose from 'mongoose';

const { Schema } = mongoose;

// 스키마 객체 생성
const PostSchema = new Schema({
  title: String,
  body: String,
  tags: [String], // 문자열 배열
  publishedDate: {
    type: Date,
    default: Date.now // 현재 날짜를 기본값으로 지정
})

// 모델 생성
const Post = mongoose.model('Post', PostSchema);
export default Post;
```

Mongoose 함수

CRUD	함수명
CREATE	create
READ	find, findById, findOne
UPDATE	updateOne, updateMany, findByIdAndUpdate, findOneAndUpdate
DELETE	deleteOne, deleteMany, findByIdAndDelete, findOneAndDelete

Data Type	Description
create()	Model로부터 call하는 메소드, model.create()
save()	instance로부터 call하는 메소드, instance.save()
find()	조건에 해당하는 모든 documents 조회, 메소드의 인자로 필터링할 조건
findOne()	조건에 해당하는 하나의 document만 조회, 메소드의 인자로 필터링할 조건
findById()	request의 파라미터로 넘어오는 id를 통해 document를 조회, 메소드의 인자로 필터링할 조건
updateOne()	조건에 해당하는 하나의 데이터만 수정
updateMany()	조건에 해당하는 모든 데이터를 수정

CRUD 해보기

- server.js

```
const express = require('express')
const mongoose = require("mongoose");
var users = require('./routes/users');
const createUser = require("./middlewares/create-user");


// Initialize express
const app = express()

// parse json objects
app.use(express.json())

//set port number
const port_number = 8080

//set mongo connection information([mongoDB name]:[address]:[mongoDB port])
const mongo_connection_info = 'mongodb://172.17.0.1:27017'

// parse url encoded objects- data sent through the url
app.use(express.urlencoded({ extended: true}))

// create a server
const PORT = port_number
app.listen(PORT, () => {
```

```
  console.log(`Server running on port ${PORT  }`)
  mongoose
  .connect(
    mongo_connection_info
  )
  .then(() => console.log('connected to database successfully'))
  .catch((err) => {
    console.log(err);
  });

});

app.use('/users', users);
app.use("/user/create", createUser); //~/user/create?name=Snoopy&species=Dog
```

- ./models/user.js

```
const mongoose = require('mongoose')
const userSchema =
mongoose.Schema({
  name: {
  type: String,
  required: true
  },
  species: {
    type: String,
    required: true
  }
})

// Export model
module.exports = mongoose.model('User',
userSchema)
```

- ./routes/users.js

```
var express = require("express");
var User = require("../models/user");
var router = express.Router();

router.use('/', async (req, res) => {
    const users = await User.find();
    res.send(users);
});

module.exports = router;
```

- ./middlewares/create-user.js

```
var User = require("../models/user");

const createUser = async function (req, res, next) {
    const { name, species } = req.query;
    const user = await User.create({name, species});
    if (user.error) {
        res.status(500).json({
          message: user.error
        })
    }
}

module.exports = createUser;
```

실습

초기 설정

구조

project

├node_modules

│ └…

├models

│ └pokemon.js

├index.js

├package.json

└package-lock.json

node js 프로젝트 생성

```
npm init -y
```

express 설치

```
npm install express
```

nodemon 설치

```
npm install -g nodemon
```

mongoose 설치

```
npm install mongoose
```

Express& Mongoose 연결

- index.js

```
const express = require('express');
const mongoose = require("mongoose");
const pokemons = require('./routes/pokemon_route');
const createPokemon = require('./middlewares/createPokemon');

const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
});

mongoose
  .connect('mongodb://localhost:27017/sample')
  .then(() => console.log('connected to database successfully'))
  .catch(err => {
    console.log("Error : ", err);
  });

app.use('/pokemons', pokemons);
app.use("/add_pokemons", createPokemon);
```

Schema 정의

- ./models/Pokemon.js

```
const mongoose = require('mongoose');

const pokemonSchema = new mongoose.Schema({
    number : {type: Number, required: true, unique: true},
```

```
    name : String,
    type : [String]
});

module.exports = mongoose.model('Pokemon', pokemonSchema);
```

Data 생성

- ./data/pokemon.json

```
{
    "pokemon" : [
            {
                "number" : 1,
                "name" : "이상해씨",
                "type" : ["풀", "독"]
            },
            {
                "number" : 2,
                "name" : "이상해풀",
                "type" : ["풀", "독"]
            },
            {
                "number" : 3,
                "name" : "이상해꽃",
                "type" : ["풀", "독"]
            },
            {
                "number" : 4,
                "name" : "파이리",
                "type" : ["불꽃"]
            },
            {
                "number" : 5,
                "name" : "리자드",
                "type" : ["불꽃"]
            },
            {
                "number" : 6,
                "name" : "리자몽",
                "type" : ["불꽃", "비행"]
            },
            {
                "number" : 7,
                "name" : "꼬부기",
                "type" : ["물"]
            },
            {
                "number" : 8,
                "name" : "어니부기",
                "type" : ["물"]
            },
            {
                "number" : 9,
```

```
                "name" : "거북왕",
                "type" : ["물"]
            }

    ]
}
```

- ./middlewares/createPokemon.js

```
var express = require("express");
var Pokemon = require("../models/pokemon");
const fs = require('fs');

const jsonFile = fs.readFileSync('data/pokemon.json', 'utf8');
const jsonData = JSON.parse(jsonFile);
const pokemonData = jsonData.pokemon;

const createPokemon = async function (req, res, next) {
  pokemonData.forEach(async (pokemon) => {
    console.log(pokemon);
    const { number, name, type } = pokemon;
    const processData = await Pokemon.create({number, name, type});
    if (processData.error) {
        res.status(500).json({
          message: processData.error
        })
    }
  });
}

module.exports = createPokemon;
```

Data 조회

- ./routes/pokemon_route.js

```
var express = require("express");
var Pokemon = require("../models/pokemon");
var router = express.Router();

router.use('/all', async (req, res) => {
  console.log("pokemons");
  const pokemons = await Pokemon.find().lean();
  res.send(JSON.stringify(pokemons));
});

router.use('/', async (req, res) => {
  const pokemon = await Pokemon.findOne({number:req.query.number}).lean();
  res.send(pokemon);
});
```

```
module.exports = router;
```