Node.js + MySQL + Sequelize

백성애 2025-05-12

MySQL Server/Workbench 설치

02. MySQL Server Workbench 8.0.17 설치.pdf 파일 참고하세요

- MySQL Server와
- MySQL Workbench 만 설치합니다

Sequelize 란?

DB작업을 쉽게하도록 도와주는 ORM 라이브러리

ORM(Object-Relational Mapping) 이란 자바스크립트 객체와 관계형 데이터베이스를 매핑해주는 도구를 의미

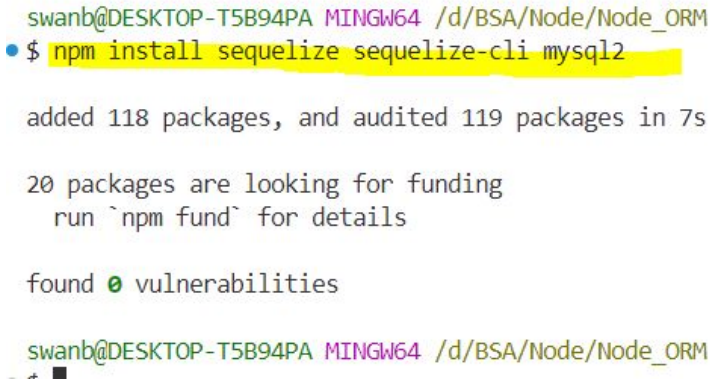
ORM을 사용하면 SQL이 아닌 클래스나 메서드를 통해 데이터베이스 CRUD 작업을 할 수 있게 된다

sequelize는 ORM 중 가장 인기 있고 여러 dbms를 지원하며 promise 패턴을 사용할 수 있도록 하여 비동기 처리도 async/await으로 할 수 있다.

sequelize 설치

```
npm install sequelize sequelize-cli mysql2
```

sequelize와 sequelize 명령어를 사용할 수 있게 해주는 sequelize-cli, MySQL 드라이버 역할을 하는 mysql2모듈을 작업 폴더 최상위에 설치한다



그외 필요한 모듈 설치

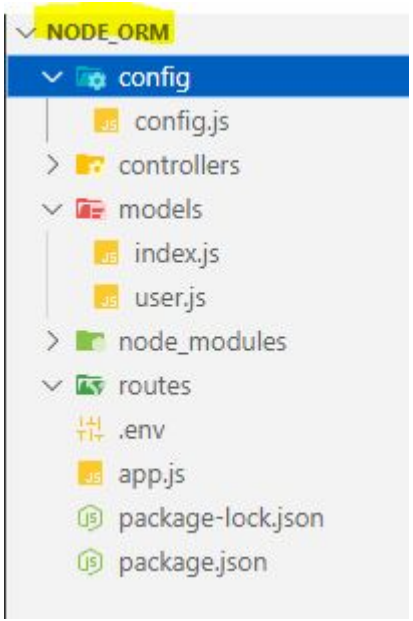
```
swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Node_ORM
$ npm i express dotenv morgan

$ npm i cors
```

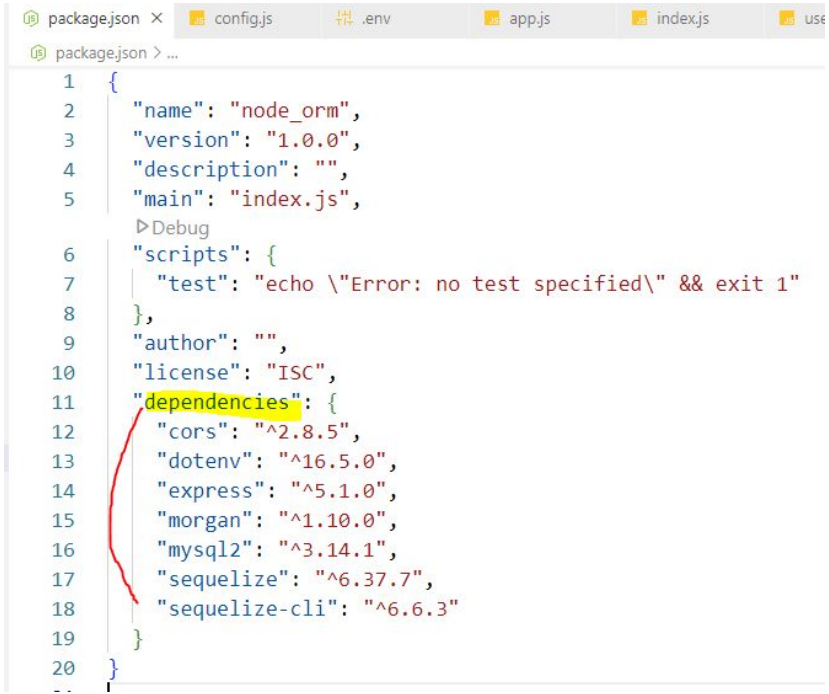
프로젝트 구조

NODE_ORM 플젝 구조 예시

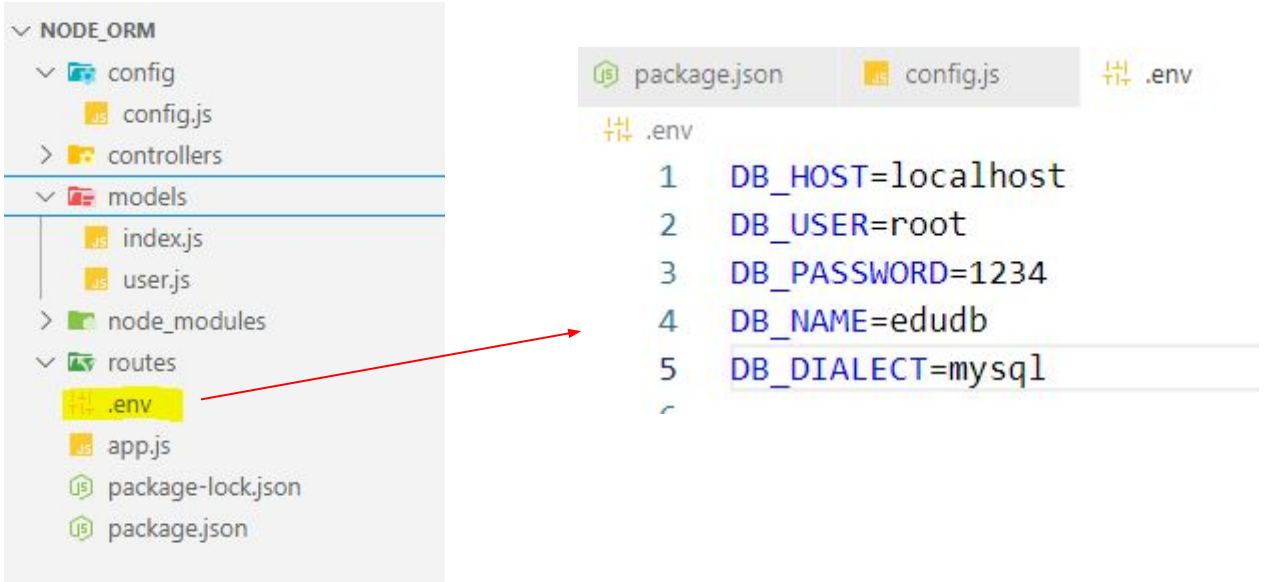
```
├── config/
│   └── config.js              ← DB 설정
├── models/
│   ├── index.js               ← Sequelize 초기화
│   └── user.js                ← User 모델 정의
├── routes/
│   └── userRouter.js          ← 회원가입 라우트
├── controllers/
│   └── userController.js      ← 회원가입 로직
├── app.js                     ← Express 앱 진입점
├── .env                       ← 환경변수 저장
└── package.json
```



package.json



[1] .env 파일 작성


NODE_ORM
config
config.js
controllers
models
index.js
user.js
node_modules
routes
.env
app.js
package-lock.json
package.json
package.json
config.js
.env
.env
1 DB_HOST=localhost
2 DB_USER=root
3 DB_PASSWORD=1234
4 DB_NAME=edudb
5 DB_DIALECT=mysql

[2] config/config.js

개발모드:development

테스트모드: test

운영모드: production

등으로 나눠 설정할 수 있다.

우리는 개발모드로 사용 예정

[3] models/index.js

이 파일의 역할은?

- Sequelize 인스턴스 생성
- 설정 정보를 기반으로 DB 연결
- 모델(User)을 등록
- 등록된 인스턴스와 모델을 하나의 객체(db)로 모아 export 한다
-

models > index.js > ...

```
1  const { Sequelize } = require('sequelize');
2  const config = require('../config/config').development; //개발모드 설정 적용
3
4  const sequelize = new Sequelize(config.database, config.username, config.password, config);
5  //이 인스턴스를 통해 DB 연결, 모델 정의, 쿼리 실행 등을 하게 됩니다
6  const db = {}; //모델들과 Sequelize 인스턴스를 모아둘 객체
7  db.sequelize = sequelize; //실제 연결 객체다 (DB 연결용).
8  db.Sequelize = Sequelize; //Sequelize 라이브러리 자체를 외부에서 필요할 때 사용하도록 넣어둔다
9
10 db.User = require('./user')(sequelize, Sequelize);
11 //user.js 모델 정의 파일을 가져와서 sequelize 인스턴스에 연결된 모델로 등록
12 //보통 module.exports = (sequelize, DataTypes) => { return sequelize.define(...) } 형식으로 정의되어 있음.
13 module.exports = db;
```

Sequelize 를 이용한 데이터베이스 연결 설정 및 모델 등록을 위한 초기화 코드

Sequelize클래스는 데이터베이스와의 연결을 관리하는 핵심객체

[4] models/user.js

```
// User 모델 정의
module.exports = (sequelize, DataTypes) => {
    const newUser = sequelize.define(
        // User 모델 정의
        'User',
        {
            id: {
                type: DataTypes.INTEGER.UNSIGNED,
                autoIncrement: true,
                primaryKey: true,
            },
            name: {
                type: DataTypes.STRING(50),
                allowNull: false,
            },
            email: {
                type: DataTypes.STRING(100),
                unique: true,
                allowNull: false,
            },
            passwd: {
                type: DataTypes.STRING(100),
                allowNull: false,
            },
            refresh_token: {
                type: DataTypes.TEXT,
                allowNull: true,
            },
            role: {
                type: DataTypes.STRING(10),
                allowNull: false,
                defaultValue: 'USER',
            },
        },
        {
            tableName: 'node_users',
            timetamp: true, // createdAt, updatedAt 자동 생성
            underscored: true, // created_at, updated_at으로 생성됨
        }
    );
    return newUser;
};
```

이 파일의 역할
Sequelize를 사용하는 프로젝트에서 **"User" 테이블에 대한 정의(모델)를 담당하는 파일**입니다.
이 파일은 해당 테이블의 **컬럼 구성, 자료형, 제약 조건** 등을 선언하며,
Sequelize가 이를 기반으로 **ORM 방식으로 데이터베이스를 다룰 수 있도록** 해줍니다.

[4] models/user.js

🔍 이 파일이 하는 역할 요약

역할	설명
테이블 모델 정의	Sequelize가 인식할 수 있는 `User` 모델 정의
컬럼 정의	id, name, email, passwd 등 컬럼의 이름, 타입, 제약 조건 등 설정
기본 옵션 설정	`timestamps`, `underscored` 같은 옵션으로 createdAt 등 자동 생성 여부 지정
Sequelize 연결	`sequelize.define()`에 인자로 받아 연결하는 방식으로 구성됨

📌 사용되는 곳 예시

`models/index.js`에서 다음과 같이 불러 사용합니다:

```
js                                                    복사  편집

db.User = require('./user')(sequelize, Sequelize);
```

이렇게 하면 `db.User`를 통해 앱 전체에서 `User` 테이블을 조작할 수 있습니다:

```
js                                                    복사  편집

await db.User.create({ name: '홍길동', email: 'test@test.com', passwd: '1234' });
```

[4] models/user.js

✅ **Sequelize에서 `timestamps: true`를 설정하면...**

```js
sequelize.define('User', { ... }, {
  timestamps: true,
  underscored: true,
});
```

이렇게만 설정해도 **아래 두 컬럼은 자동으로 생성 및 관리됩니다**:

컬럼 이름	설명
`created_at`	레코드 생성 시 자동으로 시간 저장
`updated_at`	레코드 수정 시 자동으로 시간 갱신

(※ `underscored: true`가 없다면 `createdAt`, `updatedAt` 형태로 생성됩니다.)

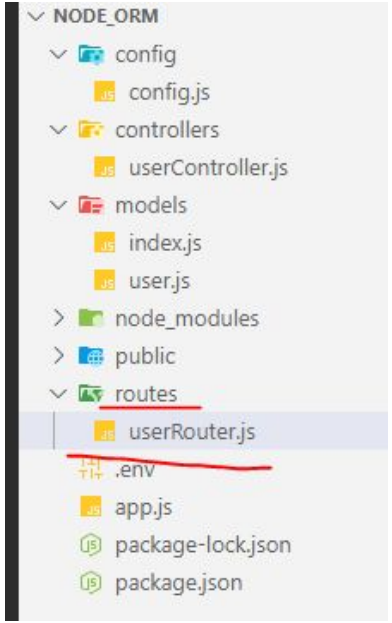
명시적으로 정의하지 않아도 생성되므로 생략해도 무방하다

[5] controllers/userController.js

controllers > userController.js > ...

```
1  const { User } = require('../models');
2
3  exports.createUser = async (req, res) => {
4      const { name, email, passwd } = req.body;
5      try {
6          const existingUser = await User.findOne({ where: { email } });
7          if (existingUser) {
8              return res.status(409).json({ message: 'Email already exists.' });
9          }
10
11         const user = await User.create({ name, email, passwd });
12         res.status(201).json({ result: 'success', message: '회원정보 등록 성공.', user });
13     } catch (err) {
14         res.status(500).json({ result: 'fail', message: 'Registration failed.', error: err.message });
15     }
16 };
```

[6] routes/userRouter.js



routes > userRouter.js > ...

```
1  const express = require('express');
2  const router = express.Router();
3  const userController = require('../controllers/userController');
4
5  router.post('/', userController.createUser);
6  module.exports = router;
7
```

[7] /app.js

package.json | config.js | .env | app.js | index.js | user.js

app.js > ...

```
1  const express = require('express');
2  require('dotenv').config();
3  const morgan = require('morgan');
4  const path = require('path');
5  const cors = require('cors');
6
7  const { sequelize } = require('./models');
8
9  //라우터 가져오기
10 const userRouter = require('./routes/userRouter');
11
12 const port = process.env.PORT || 7777;
13
14 const app = express();
15 //미들웨어 설정
16 app.use(express.json());
17 app.use(express.urlencoded({ extended: true }));
18 app.use(express.static(path.join(__dirname, 'public')));
19 app.use(cors());
20 app.use(morgan('dev'));
21
22 app.use('/api/users', userRouter);
```

```
23 /**
24  * sequelize.sync()는 Sequelize에서 모델 정의를 실제 데이터베이스 테이블로 동기화하는 역할을 합니다.
25  * 즉 자바스크립트로 정의한 모델을 실제 DB 테이블로 만들어주는 명령어
26  */
27
28 //force: true → 테이블을 강제로 삭제 후 재생성 (주의!  개발환경에서만 사용. 운영환경에서는 안된다)
29 //          기존 데이터를 보존하면서 모델과  DB스키마를 동기화 시킴
30
31 //force: false → 기존 테이블이 없으면 생성, 있으면 그대로 둠
32 sequelize.sync({ force: false }).then(() => {
33     console.log('DB connected');
34     app.listen(port, () => {
35         console.log('Server running on http://localhost:' + port);
36     });
37 });
38
```

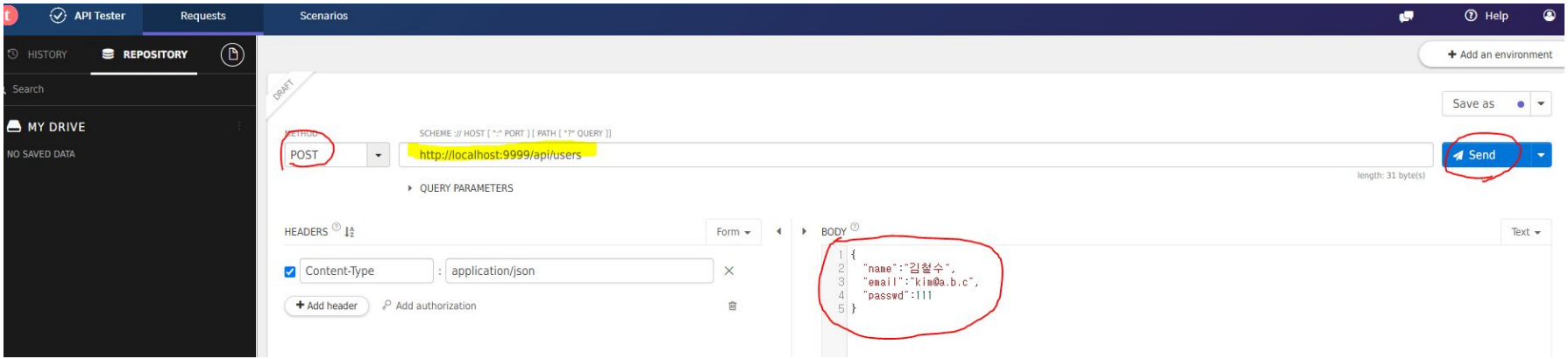
[8] node app.js로 서버 실행

```
swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Node_ORM
$ node app.js
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'node_users' AND TABLE_SCHEMA = 'edudb'
Executing (default): SHOW INDEX FROM `node_users` FROM `edudb`
DB connected
Server running on http://localhost:9999
```

[9] Talend API Tester로 회원가입 테스트

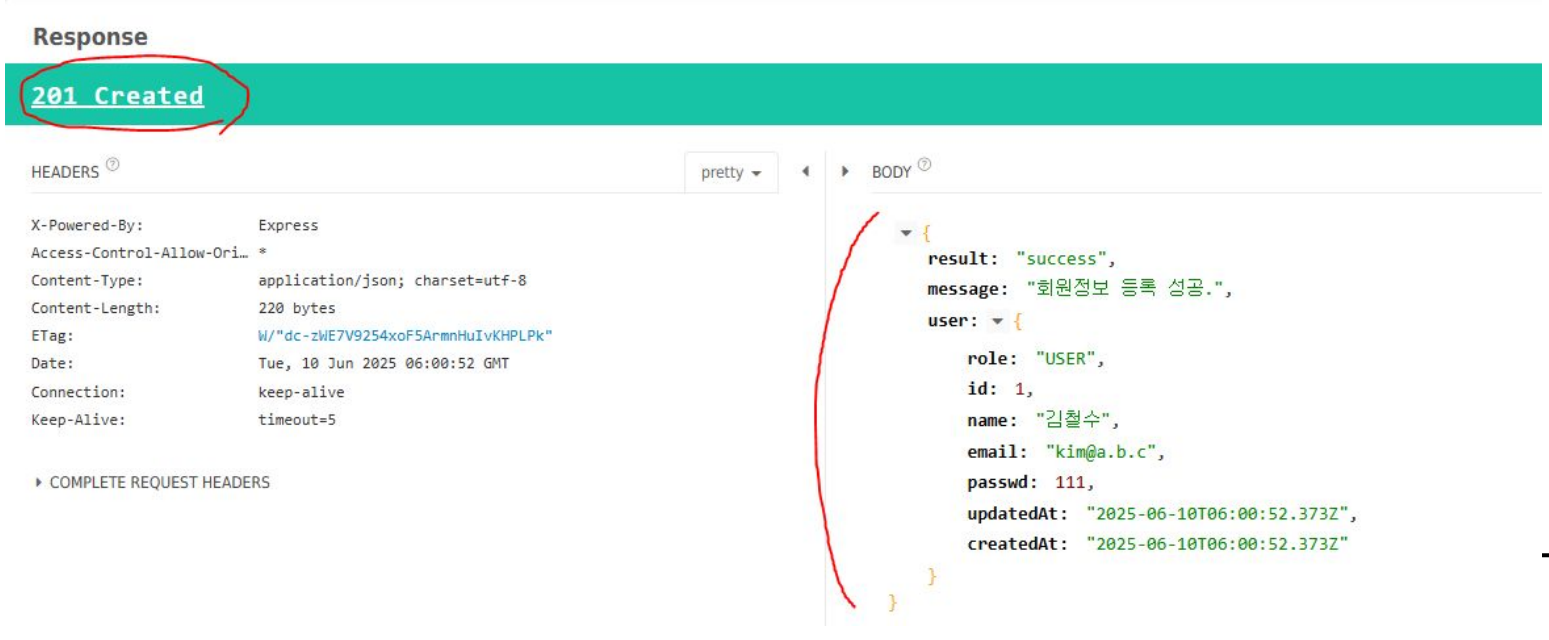
회원가입 요청 보내기

POST 방식으로 http://localhost:9999/api/users로 요청을 보내고 회원가입 데이터를 json형태로 body에 포함시켜

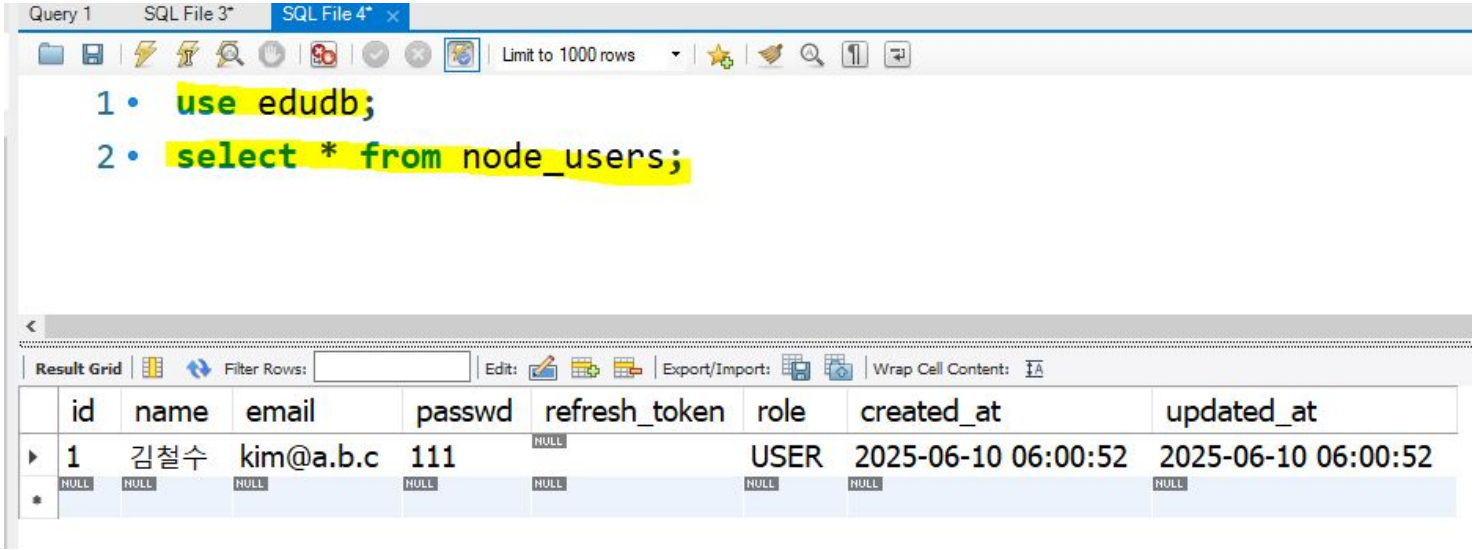


[9] Talend API Tester로 회원가입 테스트

회원가입 결과 응답



[10] DB에서 확인해본다



Sequelize 주요 함수

함수명	설명	예시
`sequelize.define()`	모델 정의	`sequelize.define('User', {...})`
`sequelize.sync()`	모델 기반으로 테이블 생성 (or 수정)	`sequelize.sync({ force: false })`
`Model.create()`	새 레코드 생성	`User.create({ name: '홍길동' })`
`Model.findAll()`	모든 레코드 조회	`User.findAll()`
`Model.findOne()`	조건에 맞는 하나 조회	`User.findOne({ where: { id: 1 } })`
`Model.findByPk()`	기본키로 조회	`User.findByPk(1)`
`Model.update()`	레코드 수정	`User.update({ name: '수정' }, { where: { id: 1 } })`
`Model.destroy()`	레코드 삭제	`User.destroy({ where: { id: 1 } })`

Sequelize 모델 정의 시 옵션

옵션명	설명	예시
`timestamps`	createdAt/updatedAt 자동 생성 여부	`timestamps: true`
`underscored`	created_at 형식으로 컬럼 생성	`underscored: true`
`tableName`	테이블명 직접 지정	`tableName: 'users'`
`freezeTableName`	복수형 자동 방지	`freezeTableName: true`
`paranoid`	삭제시 실제 삭제 대신 deletedAt 기록	`paranoid: true`
`defaultValue`	컬럼의 기본값 설정	`defaultValue: 'USER'`

Sequelize 필드 정의시 사용하는 옵션

속성	설명	예시
`type`	데이터 타입	`DataTypes.STRING(50)`
`allowNull`	null 허용 여부	`allowNull: false`
`unique`	고유 제약조건	`unique: true`
`primaryKey`	기본 키 여부	`primaryKey: true`
`autoIncrement`	자동 증가 여부	`autoIncrement: true`
`defaultValue`	기본값	`defaultValue: 'USER'`

Sequelize 모델 간 관계 설정(association)

관계	Sequelize 함수	설명	예시
1:N	`hasMany()`	부모가 여러 자식을 가짐	`User.hasMany(Post)`
N:1	`belongsTo()`	자식이 부모를 가짐	`Post.belongsTo(User)`
1:1	`hasOne()`	하나의 대상만 가짐	`User.hasOne(Profile)`
N:M	`belongsToMany()`	다대다 관계	`User.belongsToMany(Role, { through: 'UserRoles' })`

Sequelize 관계 설정 시 옵션

📌 5. 관계 설정 시 옵션

옵션명	설명	예시
`foreignKey`	외래 키 이름 지정	`foreignKey: 'user_id'`
`sourceKey`	원본 키 지정	`sourceKey: 'id'`
`targetKey`	연결 대상 키	`targetKey: 'id'`
`through`	중간 테이블 이름 (N:M)	`through: 'UserRoles'`
`onDelete`	삭제 시 동작	`onDelete: 'CASCADE'`
`as`	관계에 별칭 부여	`as: 'posts'`

실습

[1] User의 CRUD 구현해보기

[2] User와 Post모델 관계 맺기

참고 사이트 및 도서

https://any-ting.tistory.com/49

백견 불여일타 Node.js로 서버 만들기 -박민경 저(로드북)

Node.js 교과서 - 조현영 (길벗)