

Node.js와 MongoDB II

00 수업 소개



커리큘럼

○ MongoDB 와 Mongoose

MongoDB의 특징과 장점에 대해 알아보고,
Mongoose 를 이용하여 Node.js 에서 MongoDB 를
사용하는 방법에 대해 학습합니다.

○ Express.js 와 MongoDB 로 웹서비스 만들기 1

Template Engine, 게시판 CRUD 만들기, 에러 핸들링,
Pagination 기법들에 대해 학습합니다.

커리큘럼



Express.js 와 MongoDB로 웹서비스 만들기 2

회원가입과 로그인, 회원관련 연동기능을 구성하는 방법에 대해 학습합니다.



Express.js 와 MongoDB로 웹서비스 만들기 3

JWT, Mail Server, OAuth2.0 등 웹 서비스를 구성하는 추가적인 기술들에 대해서 학습합니다.

추천대상

1. 웹 서비스 구성의 기본을 다지고 싶으신 분

웹 서비스를 구성하는 필수적인 기술들에 대해서 처음부터 단단하게 다지고 싶으신 분

2. 웹 서비스 개발 흐름을 이해하고 싶으신 분

웹 서비스의 기본적인 개발 방법 및 서비스에 기능을 추가하는 과정과 흐름에 대해서 이해하고 싶으신 분

3. 실제로 사용되는 웹 기술들을 시도하고 싶으신 분

교과서적인 내용을 벗어나 실제로 사용되는 웹 기술들을 프로젝트에 적용하고 싶으신 분

수강목표

1. MongoDB 와 Mongoose 사용하기

MongoDB 의 개념과 동작방식을 이해하고
Mongoose 패키지를 프로젝트에 적용하여 사용합니다.

2. Express.js 로 웹서비스 개발하기

Express.js 를 MongoDB와 연결하여 데이터베이스를 사용하는
동적인 웹 서비스를 개발합니다.

3. 다양한 웹 기술 사용하기

실제로 웹 서비스를 구성하는데 필요한 다양한 기술들을 프로젝트에
적용하는 방법에 대해 학습합니다.



Node.js와 MongoDB II

01 MongoDB와 Mongoose



목차

01. MongoDB
02. Mongoose ODM
03. Mongoose ODM 사용하기
04. Express.js + Mongoose
05. 추가 - Sequelize ORM

수강목표

1. NoSQL과 MongoDB 이해하기

MongoDB로 대표되는 NoSQL의 개념에 대해 학습하고 MongoDB를 사용해보며 NoSQL 동작에 대해 학습합니다.

2. Mongoose ODM 학습하기

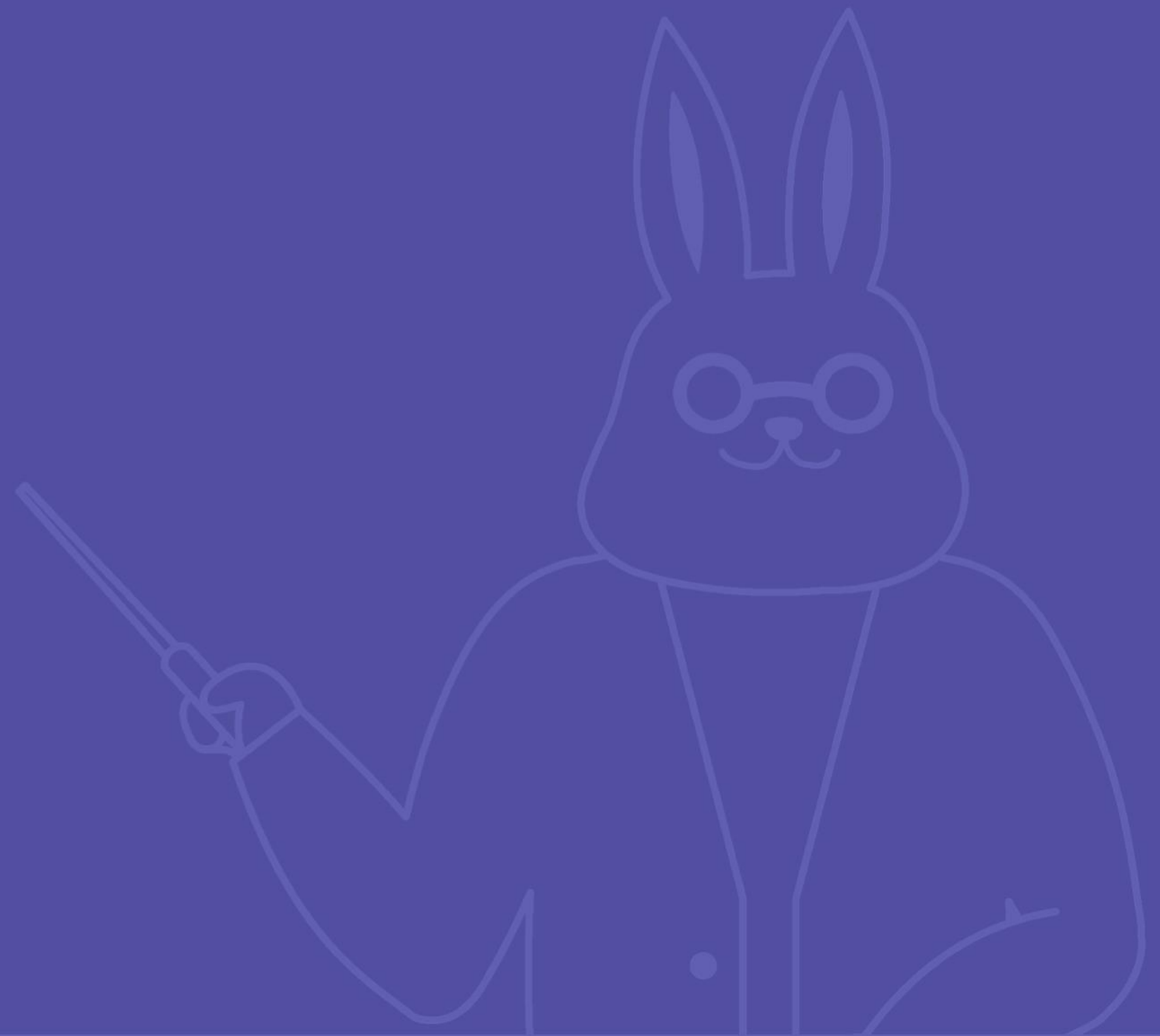
Node.js에서 MongoDB를 다루기 위해 Mongoose ODM에 대해 학습합니다.

3. Express.js에서 Mongoose ODM 사용하기

Mongoose ODM을 Express.js 어플리케이션 내에서 사용하는 방법에 대해 학습합니다.

01

MongoDB



✓ MongoDB란?

대표적인 **NoSQL, Document DB**

Mongo는 Humongous 에서 따온 말로, 엄청나게 큰 DB 라는 의미

→ **대용량 데이터**를 처리하기 좋게 만들어짐

✓ RDB와 NoSQL

NoSQL

VS

RDB

✓ RDB와 NoSQL

Relational Database

관계형 데이터베이스

자료들의 관계를 주요하게 다룸.

SQL 질의어를 사용하기 위해 데이터를 구조화해야 함

✓ RDB와 NoSQL

Non SQL 또는 Not Only SQL

구조화된 **질의어를 사용하지 않는** 데이터베이스
자료 간의 관계에 초점을 두지 않음
데이터를 구조화하지 않고, **유연하게 저장함**

✓ NoSQL을 사용하는 이유

SQL을 사용하기 위해서는 **데이터를 구조화**하는 것이 필수 (DDL)

→ **스키마에 정의된 데이터가 아니면** 저장할 수 없는 제약이 따름

NoSQL을 사용하면 **사전작업 없이** 데이터베이스를 사용할 수 있음

→ 데이터베이스 작업에 크게 관여하지 않고 **프로젝트를 빠르게 진행**할 수 있음

✓ MySQL(RDB) vs MongoDB(NoSQL)

MySQL

```
CREATE DATABASE simple_board

CREATE TABLE posts (
  id NOT NULL AUTO INCREMENT
  title VARCHAR(30),
  content TEXT,
  PRIMARY KEY(id)
);

INSERT INTO posts (title, content)
VALUES
('first title', 'first content'),
('second title', 'second content'),
```

MongoDB

```
use simple_board
db.posts.insert([
  {
    title: 'first title',
    content: 'first content'
  },
  {
    title: 'second title',
    content: 'second content'
  }
]);
```

✓ NoSQL과 Document DB

NoSQL은 다양한 종류가 있지만,
대표적으로 자료를 **Document**(문서) 로 저장하는 **Document DB**가 일반적
이 외에, key-value, Graph, large collection 등의 NoSQL DB가 존재

✓ MongoDB 기본 개념



✓ MongoDB 기본 개념 - Database

Database

하나 이상의 **collection**을 가질 수 있는 저장소
SQL에서의 **database**와 유사

✓ MongoDB 기본 개념 - Collection

Collection

하나 이상의 **Document**가 저장되는 공간

SQL 에서의 **table**과 유사

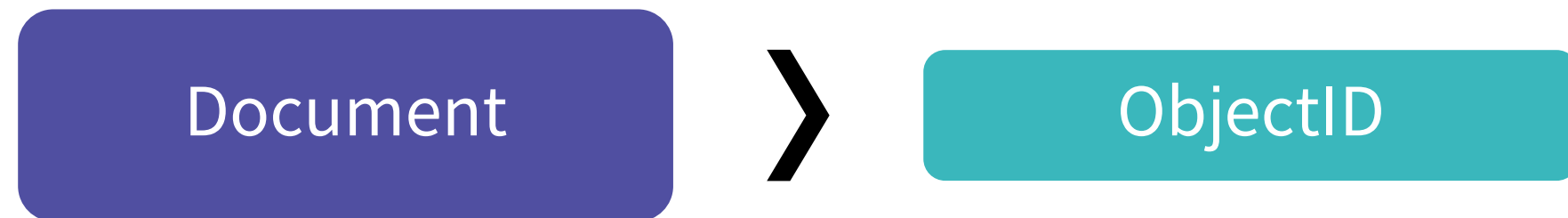
하지만, collection 이 document의 **구조를 정의하지 않음**

✓ MongoDB 기본 개념 - Document

Document

MongoDB에 **저장되는 자료**,
SQL에서 **row**와 유사하지만 구조제약 없이 **유연하게 저장** 가능
JSON과 유사한, BSON을 사용하여 **다양한 자료형**을 지원

✓ MongoDB 기본 개념 - Document - ObjectId



각 document의 **유일한 키 값**, SQL 의 **primary key**와 유사
하나씩 증가하는 값이 아닌 document를 생성할 때 **자동으로 생성되는 값**

※ timestamp + random value + auto increament

✓ MongoDB 사용 방법

MongoDB를 **직접 설치**하거나 **Cloud 서비스**를 사용할 수 있음

직접 설치하면 **귀찮고 어렵지만** 원하는 만큼 **얼마든지 데이터를 사용할** 수 있음

Cloud 를 사용하면 **쉽고 빠르게 시작** 가능하지만, **사용량에 따라 요금이 부과**됨

✔ MongoDB 사용 방법 - 직접 MongoDB 설치하기

직접 MongoDB 설치하기

직접 모든 데이터베이스 관련 설정을 해야 함

Sharding이나 Replication 등의 작업이 필요할 때 **운영지식과 노하우**가 요구됨
무료로 사용할 수 있는 **Community Version**을 제공 함

✓ MongoDB 사용 방법 - MongoDB Cloud 이용하기

MongoDB Cloud 이용하기

모든 데이터베이스 관련 기능을 **웹에서 관리 가능**

특별한 노하우 없이 데이터베이스 운용 가능

사용량에 따라 비용이 발생하지만, **512MB까지는 평생 무료로** 사용 가능

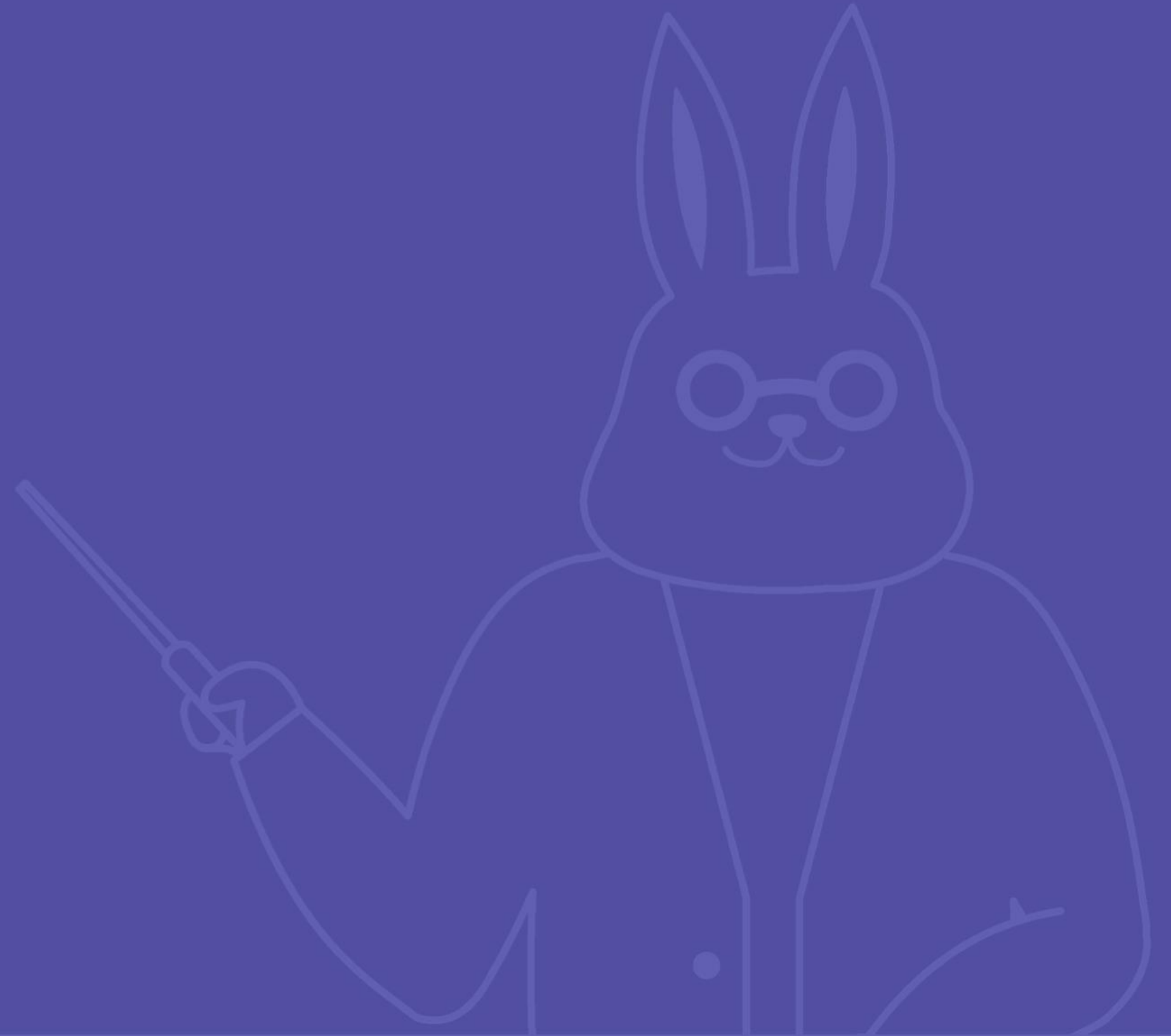
✓ MongoDB 사용 방법 - MongoDB Compass

MongoDB Compass

MongoDB에 접속하여 Database, Collection, Document 등을
시각화하여 관리할 수 있게 도와주는 도구
MySQL을 사용할 때 **MySQL Workbench**와 유사

02

Mongoose ODM



✓ Mongoose ODM이란?

Object **D**ata **M**odeling

MongoDB의 **Collection**에 **집중하여 관리**하도록 도와주는 패키지
Collection을 **모델화**하여, 관련 기능들을 **쉽게 사용할 수 있도록** 도와줌

✓ Mongoose ODM을 사용하는 이유

연결관리

MongoDB의 기본 Node.js 드라이버는 **연결상태를 관리하기 어려움**

Mongoose를 사용하면 **간단하게 데이터베이스와의 연결상태를 관리**해줌

✔ Mongoose ODM을 사용하는 이유

스키마 관리

스키마를 **정의하지 않고 데이터를 사용할 수 있는 것은 NoSQL의 장점**이지만,
데이터 형식을 미리 정의 해야 **코드 작성과 프로젝트 관리**에 유용함
Mongoose는 **Code-Level에서 스키마를 정의**하고 관리할 수 있게 해 줌

✓ Mongoose ODM을 사용하는 이유

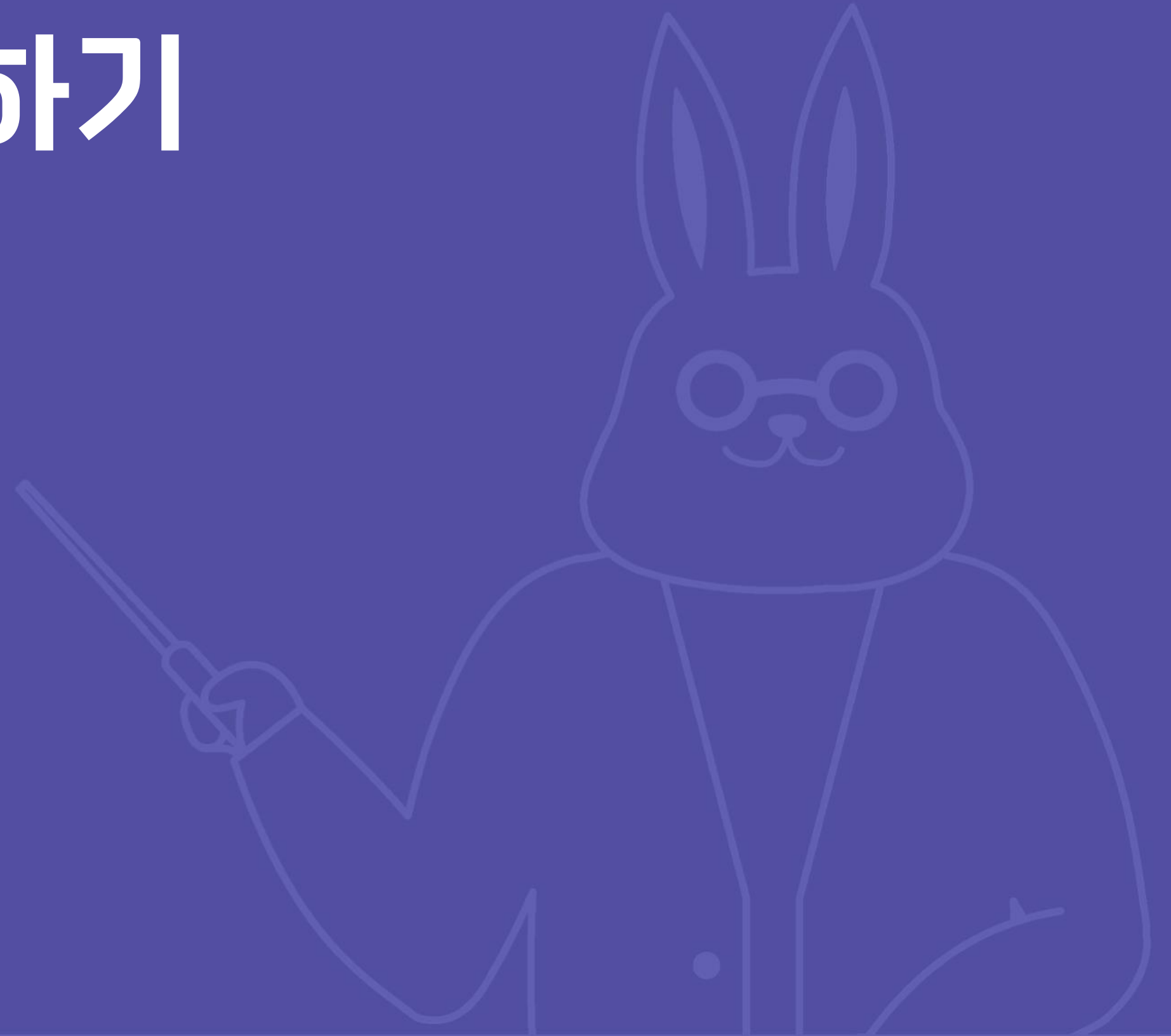
Populate

MongoDB는 기본적으로 **Join**을 제공하지 않음

Join과 유사한 기능을 사용하기 위해선 **aggregate** 라는 복잡한 쿼리를 해야 하지만,
Mongoose는 **populate**를 사용하여 간단하게 구현할 수 있음

03

Mongoose ODM 사용하기



✓ Mongoose ODM 사용 방법

Mongoose ODM 사용 순서

1. 스키마 정의
2. 모델 만들기
3. 데이터베이스 연결
4. 모델 사용

✓ 스키마 정의하기

./models/schemas/board.js

```
const { Schema } = require('mongoose');

const PostSchema = new Schema({
  title: String,
  content: String,
}, {
  timestamps: true,
});

module.exports = PostSchema;
```

Collection에 저장될 Document의 스키마를 **Code-Level에서 관리**할 수 있도록 Schema를 작성할 수 있음

다양한 **형식을 미리 지정**하여, 생성, 수정 작업 시 **데이터 형식을 체크**해주는 기능을 제공함

timestamps 옵션을 사용하면 **생성, 수정 시간을 자동으로 기록**해 줌

✓ 모델 만들기

./models/index.js

```
const mongoose = require('mongoose');  
  
const PostSchema = require('./schemas/board');  
  
exports.Post = mongoose.model('Post', PostSchema);
```

작성된 스키마를 mongoose에서
사용할 수 있는 **모델로 만들어야 함**

모델의 이름을 지정하여 Populate 등에서
해당 이름으로 모델을 호출할 수 있음

✓ 데이터베이스 연결하기

index.js

```
const mongoose = require('mongoose');  
  
const { Post } = require('./models');  
  
mongoose.connect('mongodb://localhost:27017/myapp');  
  
// Post 바로 사용 가능
```

connect 함수를 이용하여 간단하게
데이터베이스에 연결할 수 있음

mongoose는 **자동으로 연결을 관리**해 주어
직접 연결 상태를 체크하지 않아도
모델 사용 시 연결 상태를 확인하여 사용이
가능할 때 작업을 실행 함

✔ 모델 사용하기 - 간단한 CRUD

작성된 모델을 이용하여 CRUD를 수행할 수 있음

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

✓ 간단한 CRUD - CREATE

index.js

```
const { Post } = require('./models');

async function main() {
  const created = await Post.create({
    title: 'first title',
    content: 'second title',
  });

  const multipleCreated = await Post.create([
    item1,
    item2
  ]);
}
```

create 함수를 사용하여 **Document 생성**

create 함수에는 Document Object나

→ **단일 Document 생성**

Document Object의 Array 전달 가능

→ **복수 Document 생성**

create는 **생성된 Document를 반환**해 줌

✓ 간단한 CRUD - FIND (READ)

index.js

```
const { Post } = require('./models');

async function main() {
  const listPost = await Post.find(query);
  const onePost = await Post.findOne(query);
  const postById = await Post.findById(id);
}
```

find 관련 함수를 사용하여
Document를 검색

query를 사용하여 검색하거나
findById를 사용하면 **ObjectID**로
Document를 검색할 수 있음

✓ Query

MongoDB에도 SQL의 where와 유사한 **조건절 사용 가능**
MongoDB의 query는 **BSON 형식**으로,
기본 문법 그대로 mongoose에서도 사용 가능

✓ Query - 자주 사용되는 query

쿼리 예제

```
Person.find({
  name: 'kyubum',
  age: {
    $lt: 20,
    $gte: 10,
  },
  languages: {
    $in: ['ko', 'en'],
  },
  $or: [
    { status: 'ACTIVE' },
    { isFresh: true },
  ],
});
```

{ key: value } 로 **exact match**

\$lt, \$lte, \$gt, \$gte 를 사용하여
range query 작성 가능

\$in 을 사용하여 **다중 값으로 검색**

\$or 를 사용하여 **다중 조건 검색**

✓ 참고 - Mongoose ODM - \$in

Mongoose \$in

```
Person.find({ name: ['elice', 'bob'] });  
// { name: { $in: ['elice', 'bob'] } }
```

Mongoose는 쿼리 값으로 배열이 주어지면 자동으로 \$in 쿼리를 생성해 줌

✓ MongoDB Query Operators

MongoDB 홈페이지에서 다양한 Query Operator들 확인 가능

<https://docs.mongodb.com/manual/reference/operator/query/>

✓ 간단한 CRUD - UPDATE

index.js

```
async function main() {
  const updateResult = await Post.updateOne(query, {
    ...
  });
  const updateResults = await Post.updateMany(query, {
    ...
  });
  const postById = await Post.findByIdAndUpdate(id, {
    ...
  });
  const onePost = await Post.findOneAndUpdate(query, {
    ...
  });
}
```

update 관련 함수를 사용하여
Document를 수정

find~ 함수들은 **검색된 Document**를
업데이트를 반영하여 반환해 줌

mongoose 의 update 는 기본적으로
\$set operator 를 사용하여,
Document를 **통째로 변경하지 않음**

✓ 간단한 CRUD - DELETE

index.js

```
async function main() {  
  const deleteResult = await Post.deleteOne(query);  
  
  const deleteResults = await Post.deleteMany(query);  
  
  const onePost = await Post.findOneAndDelete(query);  
  
  const postById = await Post.findByIdAndDelete(query);  
}
```

delete 관련 함수를 사용하여
Document 삭제

find~ 함수들은
검색된 Document를 반환해 줌

✓ populate

populate.js

```
const Post = new Schema({
  ...
  user: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  },
  comments: [{
    type: Schema.Types.ObjectId,
    ref: 'Comment',
  }],
});

const post = await Post
  .find().populate(['user', 'comments']);

// post.user.name, post.comments[0].content
```

Document 안에 **Document**를 담지 않고,
ObjectID를 가지고 **reference**하여 사용할 수
있는 방법을 제공 함

Document에는
reference되는 ObjectId를 담고,
사용할 때 **populate**하여
하위 **Document처럼** 사용할 수 있게 해 줌

04

Express.js + Mongoose ODM



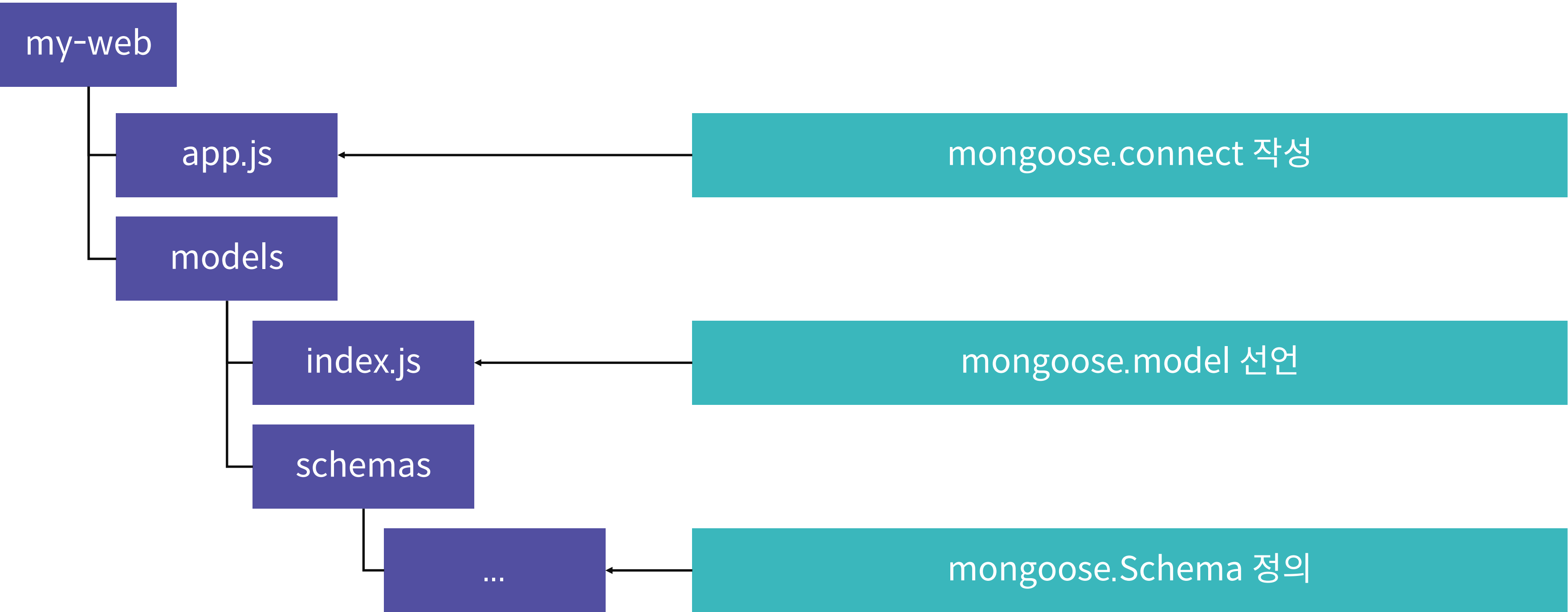
✓ Express.js에서 Mongoose ODM 사용하기

Express.js는 **프로젝트 구조를 자유롭게 구성**할 수 있기 때문에 어느 부분에 Mongoose ODM을 위치시키면 좋을지 **적절한 위치를 결정**하는 것이 중요

✓ Mongoose ODM 위치 정하기

일반적으로 **models** 디렉터리에 **Schema와 Model을 같이** 위치
app 객체는 어플리케이션 시작을 의미하는 부분이므로
해당 부분에 데이터베이스 연결을 명시하는 `mongoose.connect`를 위치

✔ Mongoose ODM 위치 정하기



✓ Mongoose ODM 커넥션 이벤트

Express.js 어플리케이션은 **종료되지 않고 동작**하기 때문에
계속해서 **데이터베이스가 정상적으로 동작하는지**를 파악하기 위해
동작 중에 발생하는 **데이터베이스 연결 관련 이벤트에 대한 처리**를 하는 것이 좋음

✓ Mongoose ODM 커넥션 이벤트

connection events

```
mongoose.connect('----');

mongoose.connection.on('connected', () => {
});

mongoose.connection.on('disconnected', () => {
});

mongoose.connection.on('reconnected', () => {
});

mongoose.connection.on('reconnectFailed', () => {
});
```

connected - 연결 완료

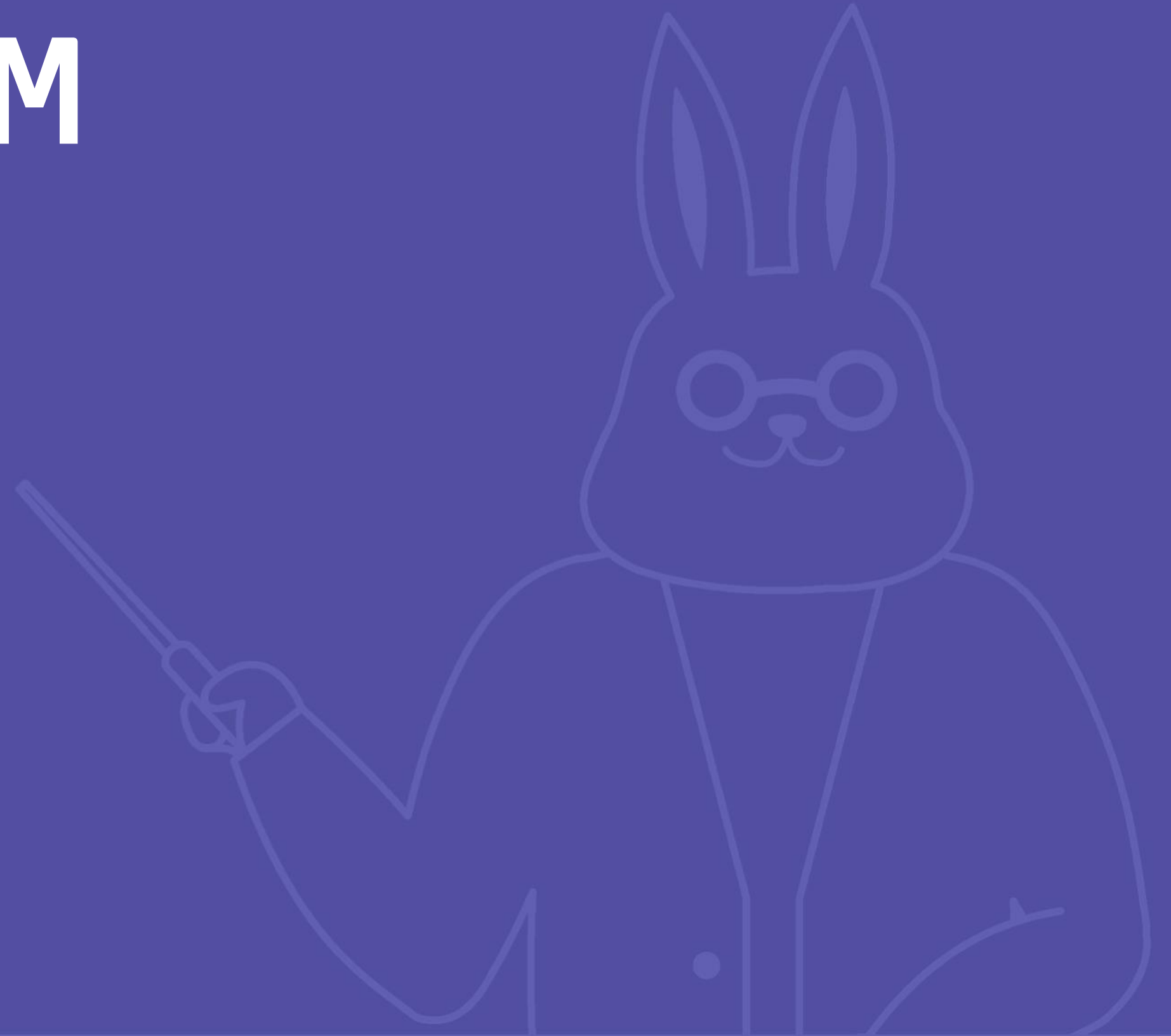
disconnected - 연결이 끊김

reconnected - 재연결 완료

reconnectFailed - 재연결 시도 횟수 초과

05

추가 - Sequelize ORM



✓ Sequelize ORM 이란?

Object-Relational Mapping

MySQL, PostgreSQL 등의 **RDBMS**를 이용하는 간단한 방법

ODM이 단순히 **모델에 집중**하여 관리하는 것에 반해,

ORM은 **테이블 관계**와 **쿼리** 등의 기능을 **더욱 단순화**하는 용도로 주로 사용

✓ Sequelize ORM 사용하기

본 강의는 **MongoDB에 집중**하기 위해
RDB와 ORM에 대한 **자세한 내용은 다루지 않음**
ORM의 **작성과 사용 방법의 예제코드**만 함께 살펴봄

✓ 디비 연결

connect db

```
const sequelize = new Sequelize('database', 'username', 'password', {  
  host: 'localhost',  
  dialect: 'mysql'  
});
```

sequelize도 **연결을 관리**하는 간단한 방법을 제공
mongoose가 MongoDB만 연결이 가능한 데에 반해
sequelize는 MySQL, PostgreSQL, SQLite 등 **다양한 RDBMS 에 연결 가능**

✓ 스키마 작성

define

```
const User = sequelize.define('User', {  
  name: {  
    type: DataTypes.STRING(10),  
    allowNull: false  
  },  
  age: {  
    type: DataTypes.Integer,  
  }  
}, {});
```

sequelize는 **define**을 통해
Schema를 생성

mongoose.Schema와 유사하지만

sequelize는 Schema가 **DDL도 생성**해 줌

✓ 스키마 작성 - 관계 정의

relation

```
User.hasMany(Post);  
Post.belongsTo(User);  
Foo.belongsToMany(Bar);  
Bar.belongsToMany(Foo);
```

sequelize를 이용하면 **테이블 간의 관계**를 **Code-Level로 관리**할 수 있음

이를 이용하면 **외래키 설정과 제약조건**까지 **DDL로 생성**해 줌

또한 **다대다 관계 설정**을 통해 **relation table**도 자동으로 생성해 줌

✓ 쿼리

query

```
User.findAll({
  where: {
    name: 'elice',
    age: {
      [Op.lt]: 20,
      [Op.gte]: 10,
    },
  },
  include: User,
});
```

Operator를 이용해
SQL 쿼리를 코드로 작성 가능

스키마의 관계 설정을 한 경우, **include**를
사용하여 **자동으로 join 쿼리 생성** 가능

✓ Synchronization

sync

```
sequelize.sync();
```

define된 model 데이터를 바탕으로 **DDL을 자동으로 실행**해 줌

-> 직접 데이터베이스에 접속하여 **테이블 생성 및 관리**를 할 필요가 없음

-> 자동으로 생성된 DDL을 따르지 않으면 **테이블 관리가 어려워 짐**

✓ Sequelize ORM 정리

Sequelize ORM 을 사용하면 데이터베이스에 **직접 DDL을 하지 않고 JavaScript 코드로** 테이블 및 관계를 관리할 수 있음
또한 RDB의 어려운 점 중 하나인 **join을 간단하게 사용**할 수 있음

크레딧

/* elice */

코스 매니저

이재성

콘텐츠 제작자

최규범

강사

최규범

감수자

최규범

디자이너

김루미

연락처

TEL

070-4633-2015

WEB

<https://elice.io>

E-MAIL

contact@elice.io

