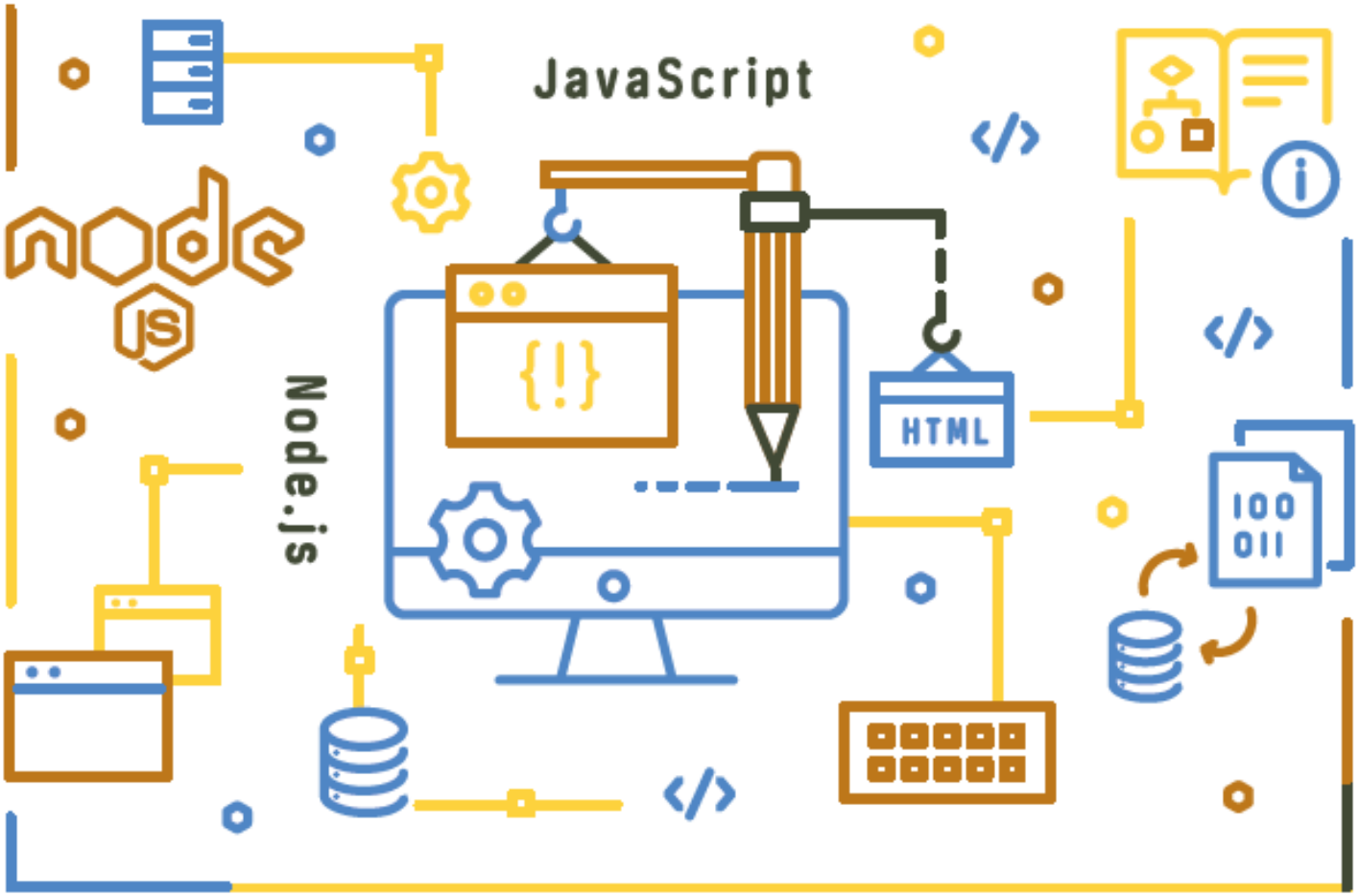


〈12장〉 몽고DB 연동

목차

01

데이터베이스

1. 데이터베이스

데이터와 정보

데이터

- 관찰이나 측정을 통해 수집된 사실이나 값
- 가공되지 않은 상태의 원시 데이터(로우(raw) 데이터)
- 형태 : 숫자, 문자, 이미지 등

정보

- 어떠한 목적이나 의도에 맞게 데이터를 가공한 것
- 사람이나 시스템이 이해하고 활용할 수 있는 형태
- 형태 : 문장, 표, 그래프 등

1. 데이터베이스

■ 데이터와 정보 예시

❖ 데이터 : 주식 시장에서 수집된 가공되지 않은 숫자나 사실

- 삼성전자 ○○○○년 ○월 ○일
  - 시가: 72,500원
  - 종가: 75,300원
  - 거래량: 3,100,500주
  - 고가: 77,300원
  - 저가: 72,700원

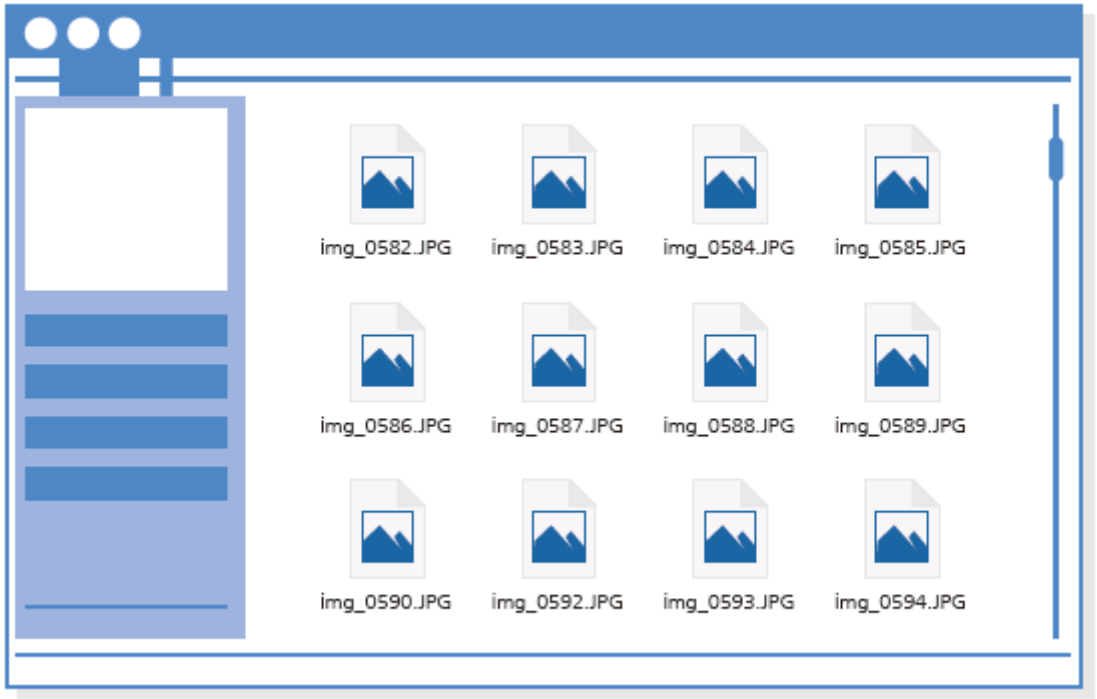
❖ 정보 : 데이터를 분석하거나 정리해서 이해할 수 있는 형태로 만든 것

- "삼성전자 주가는 오늘 2.3% 상승해 75,300원에 마감했습니다."
- "거래량이 급증하며 투자자들의 매수세가 강해진 것으로 보입니다. “
- "오늘 주가 상승은 반도체 업황 개선 기대감에 따른 것으로 분석됩니다."

1. 데이터베이스

파일의 사용

- 가장 일반적으로 사용하는 데이터 보관 방법
- 데이터를 파일에 저장하고 읽어오는 형태가 바로 파일 시스템을 사용하고 있는 것

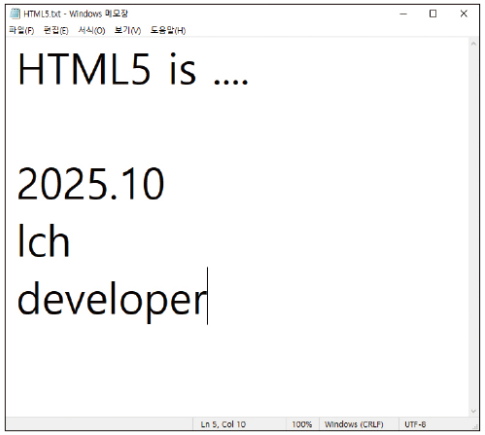


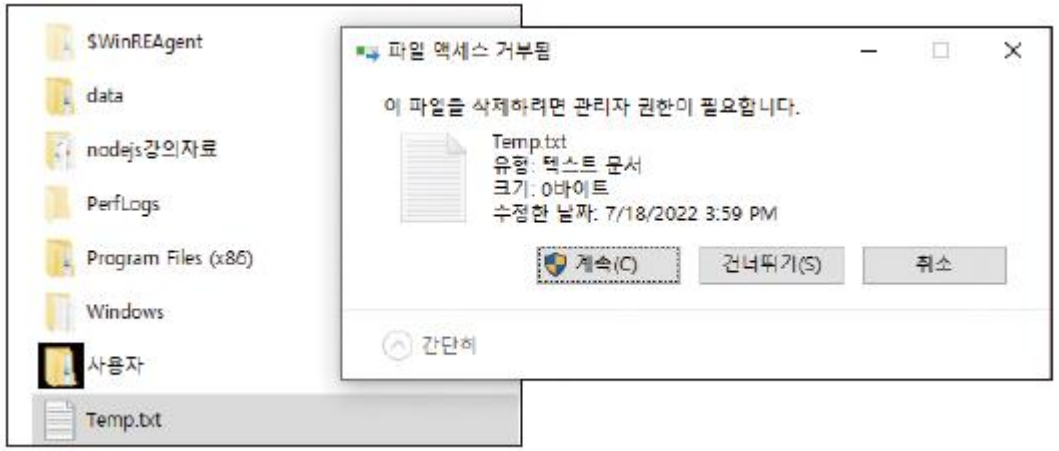
1. 데이터베이스

파일의 사용

파일의 한계점

- 특정 데이터를 검색해서 찾아내기 힘들다는 것
- 파일은 누구나 접근하여 삭제 가능하기 때문에 보안에 취약한 것

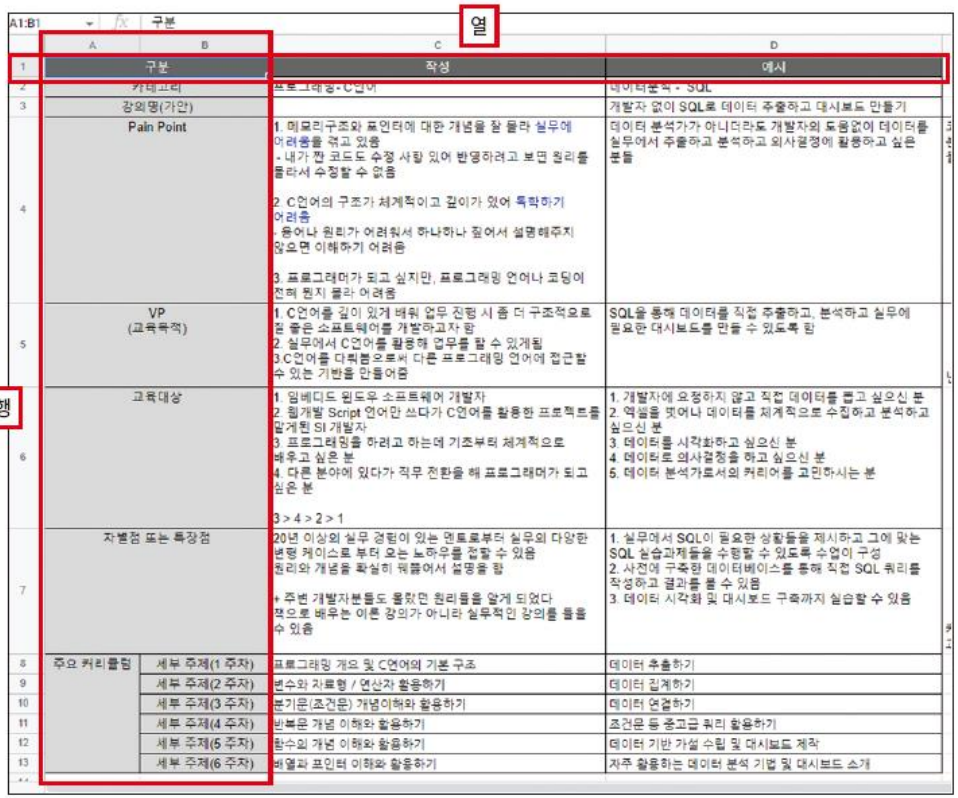




1. 데이터베이스

■ 표의 구조 등장

- 파일을 사용할 때 데이터가 구조화되어 있지 않아 발생하는 한계점을 극복하기 위해 표(Table)라는 방법이 등장
  - 엑셀(Excel), 스프레드시트, MS-ACCESS 등

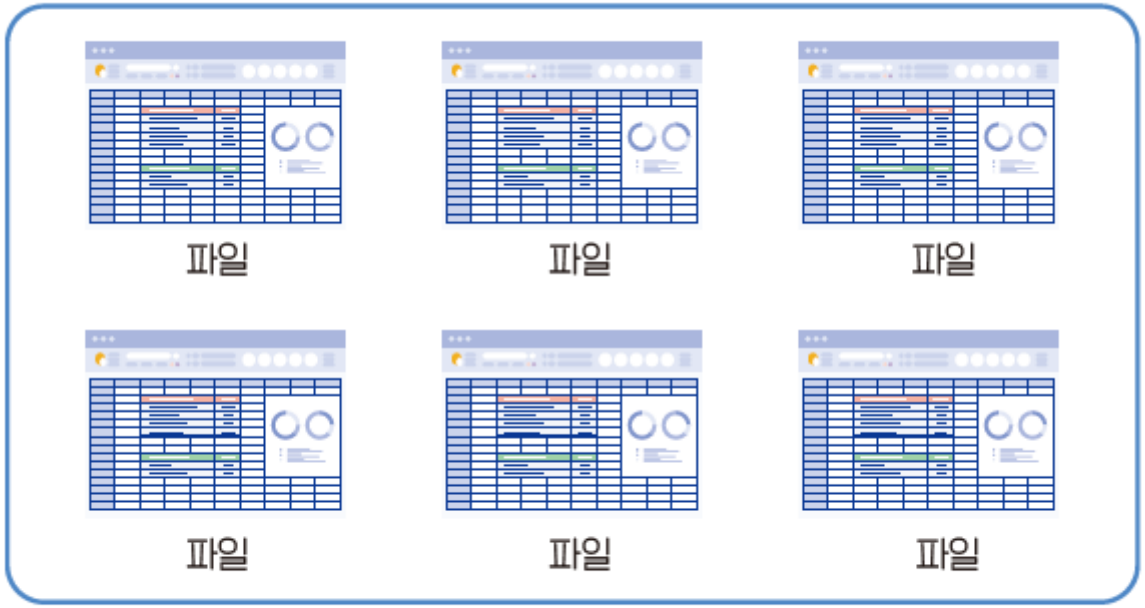


	A	B	C	D
1	구분		작성	예시
2	카테고리		프로그래밍-C언어	데이터분석 - SQL
3	강의명(가안)			개발자 없이 SQL로 데이터 추출하고 대시보드 만들기
4	Pain Point		1. 메모리구조와 포인터에 대한 개념을 잘 몰라 실무에 어려움을 겪고 있음 - 내가 짠 코드도 수정 사항 있어 반영하려고 보면 원리를 몰라서 수정할 수 없음 2. C언어의 구조가 체계적이고 깊이가 있어 독학하기 어려움 - 용어나 원리가 어려워서 하나하나 짚어서 설명해주지 않으면 이해하기 어려움 3. 프로그래머가 되고 싶지만, 프로그래밍 언어나 코딩이 전혀 뭔지 몰라 어려움	데이터 분석가가 아니더라도 개발자의 도움없이 데이터를 실무에서 추출하고 분석하고 의사결정에 활용하고 싶은 분들
5	VP (교육목적)		1. C언어를 깊이 있게 배워 업무 진행 시 좀 더 구조적으로 잘 짜인 소프트웨어를 개발하고자 함 2. 실무에서 C언어를 활용해 업무를 할 수 있게됨 3.C언어를 다뤄봄으로써 다른 프로그래밍 언어에 접근할 수 있는 기반을 만들어줌	SQL을 통해 데이터를 직접 추출하고, 분석하고 실무에 필요한 대시보드를 만들 수 있도록 함
6	고객대상		1. 임베디드 윈도우 소프트웨어 개발자 2. 웹개발 Script 언어만 쓰다가 C언어를 활용한 프로젝트를 맡게된 SI 개발자 3. 프로그래밍을 하려고 하는데 기초부터 체계적으로 배우고 싶은 분 4. 다른 분야에 있다가 직무 전환을 해 프로그래머가 되고 싶은 분 3 > 4 > 2 > 1	1. 개발자에 요청하지 않고 직접 데이터를 뽑고 싶으신 분 2. 엑셀을 뗐어나 데이터를 체계적으로 수집하고 분석하고 싶으신 분 3. 데이터를 시각화하고 싶으신 분 4. 데이터로 의사결정을 하고 싶으신 분 5. 데이터 분석가로서의 커리어를 고민하시는 분
7	차별점 또는 특장점		20년 이상의 실무 경험이 있는 멘토로부터 실무의 다양한 변형 케이스로 부터 오는 노하우를 전할 수 있음 원리와 개념을 확실히 해줘서 설명을 함 + 주변 개발자분들도 몰랐던 원리들을 알게 되었다 책으로 배우는 이론 강의가 아니라 실무적인 강의를 들을 수 있음	1. 실무에서 SQL이 필요한 상황들을 제시하고 그에 맞는 SQL 실습과제들을 수행할 수 있도록 수업이 구성 2. 사전에 구축한 데이터베이스를 통해 직접 SQL 쿼리를 작성하고 결과를 볼 수 있음 3. 데이터 시각화 및 대시보드 구축까지 실습할 수 있음
8	주요 커리큘럼	세부 주제(1 주차)	프로그래밍 개요 및 C언어의 기본 구조	데이터 추출하기
9		세부 주제(2 주차)	변수와 자료형 / 연산자 활용하기	데이터 집계하기
10		세부 주제(3 주차)	분기문(조건문) 개념이해와 활용하기	데이터 연결하기
11		세부 주제(4 주차)	반복문 개념 이해와 활용하기	조건문 등 중고급 쿼리 활용하기
12		세부 주제(5 주차)	함수의 개념 이해와 활용하기	데이터 기반 가설 수립 및 대시보드 제작
13		세부 주제(6 주차)	배열과 포인터 이해와 활용하기	자주 활용하는 데이터 분석 기법 및 대시보드 소개

1. 데이터베이스

데이터베이스의 등장

- 파일의 한계를 극복한 방식
- 데이터의 양이 1억 개 이상 계속 늘어나도 성능에 크게 영향을 받지 않음
- 구조화된 형식의 데이터인 스프레드시트 파일이 여러 개 모여 있는 것이 데이터베이스



1. 데이터베이스

■ 데이터베이스의 분류

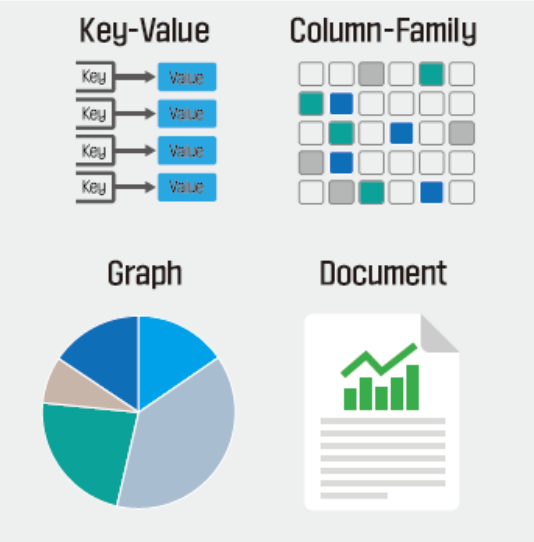
- 관계형 데이터베이스(Relational Database)와 NoSQL(Not Only SQL)로 분류
  - 관계형 데이터베이스

<table>
<tr><th>분류</th><th>특징</th><th>형태</th></tr>
<tr><td>RDB</td><td>가장 일반적이고 많이 사용되는 데이터베이스로써 행(Row)과 열(Column)로 구성된 테이블 간의 관계를 나타날 때 사용되며, SQL을 통해서만 데이터베이스에 접근할 수 있다. 대표적으로 오라클(Oracle), MySQL, MS-SQL 등이 RDB(관계형 데이터베이스)다.</td><td>두 테이블간에 관계는 교수님 번호로 결합되어 있다.
<table>
<tr><th>교수님 번호</th><th>이름</th></tr>
<tr><td>0</td><td>이귀엽</td></tr>
<tr><td>1</td><td>스티브</td></tr>
<tr><td>2</td><td>아이유</td></tr>
</table>
<table>
<tr><th>과목번호</th><th>이름</th><th>교수님 번호</th></tr>
<tr><td>0</td><td>컴공</td><td>0(이귀엽)</td></tr>
<tr><td>1</td><td>물리</td><td>0(이귀엽)</td></tr>
<tr><td>2</td><td>경영</td><td>1(스티브)</td></tr>
</table></td></tr>
</table>

1. 데이터베이스

데이터베이스의 분류

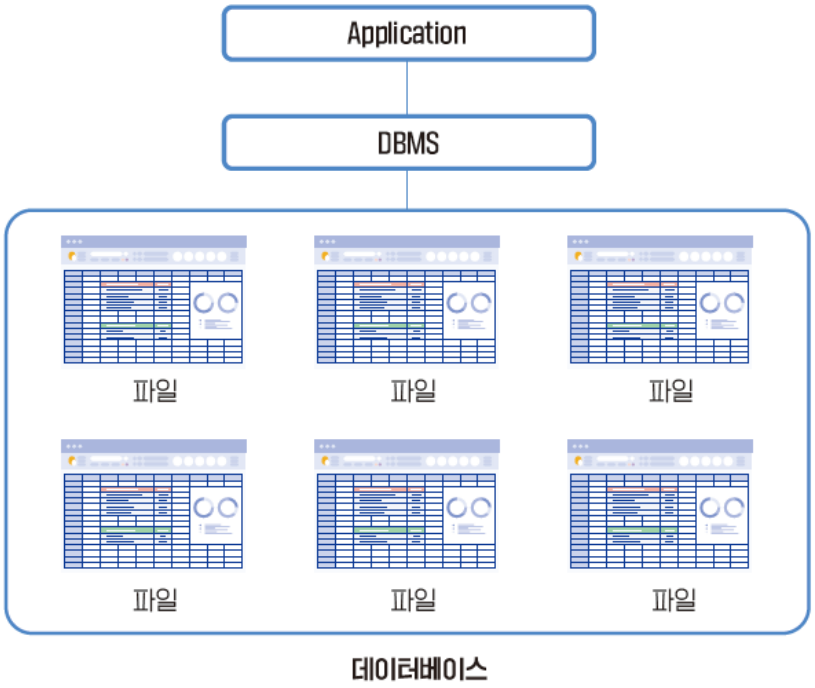
- NoSQL(Not Only SQL)

NoSQL	"Not Only SQL"의 약자로 SQL만을 사용하지 않는 데이터베이스라는 의미다. 즉, RDB의 특성뿐만 아니라 다른 유형의 데이터베이스도 지원한다는 말이다. **1. 확장이 쉽다.** 관계형 데이터베이스는 확장이 어렵지만, NoSQL 데이터베이스는 데이터 분산 저장을 기본적으로 지원함으로써 확장이 용이하다. **2. 다루기 쉽다.** SQL을 배우지 않고 자바스크립트 object 자료형 사용하듯이 데이터를 입출력할 수 있으므로 사용이 편리하다. **3. 스키마 정의 없이도 사용 가능하다.** 대표적으로 몽고DB(MongoDB), 레디스(Redis), 카산드라(Cassandra) 등이 있다.	테이블에 국한되지 않고 자유로운 형식으로 데이터를 쉽게 분산 저장할 수 있다.  1. Key-value **모델** : object, json 형태로 데이터를 쉽게 저장 및 출력 가능하다. 2. Document **모델** : 테이블 대신 컬렉션이라는 문서 기반으로 데이터를 분류하고 저장한다. 3. Graph **모델** : 데이터를 노드의 형태로 저장하고, 노드 간의 흐름 또는 관계를 저장한다. 4. Column-Family **모델** : 한 행마다 각각 다른 수, 다른 종류의 열을 가질 수 있다.

1. 데이터베이스

데이터베이스의 등장

- 이 데이터를 관리해 주는 소프트웨어 DBMS(Database Management System : 데이터베이스 관리 시스템)
- 데이터베이스에는 사용자나 응용프로그램이 직접 접근할 수 없고, DBMS를 통해서만 접근 가능



1. 데이터베이스

DBMS의 종류

- 흔히 부르는 데이터베이스 이름들은 사실 DBMS를 가리키는 것

DBMS	특징
MySQL	오픈 소스로 무료 라이센스 DBMS 중 가장 많이 사용되고 있다. 중소기업 층에서 많이 사용되며 상업적 사용 시에는 비용이 발생한다. 원래 MySQL 사에서 개발했고, 현재는 오라클에서 인수하여 서비스 중이다. 호환성이 좋아 다양한 운영체제에서 사용 가능하다.
Oracle	전 세계적으로 사용 점유율이 가장 높은 DBMS다. 오라클 사에서 만들었고, 성능이 MySQL이나 MSSQL보다 조금 더 우수하다. 고객의 필요에 맞게 커스터마이징이 가능하므로 비용이 비싸고 대기업에서 주로 사용한다.
MS-SQL	마이크로소프트 사에서 만든 상업용 DBMS이다. 다양한 운영체제에서 사용은 가능하지만 윈도우에 특화되어 있어 윈도우 서버를 운용하거나 MS 기반의 애플리케이션 연동 시에 주로 사용한다.
MongoDB	가장 대표적인 NoSQL 기반의 DBMS이다. 도큐먼트 방식으로 Json, XML과 같은 컬렉션(Collection) 데이터 모델 구조를 갖는다. 다양한 형태의 데이터 저장이 가능하고 상대적으로 사용법이 쉽다.

02

몽고DB 설치

2. MongoDB

MongoDB란?

- NoSQL 데이터베이스의 한 종류
- 문서(Document) 지향 데이터 모델을 사용하는 데이터베이스
  - 문서 지향 : XML 또는 JSON과 같은 형식의 데이터를 저장하는 것
- {key : value} 형식의 객체 형식인 JSON 형태로 데이터를 저장

```
{
    "name" : "lee",
    "age" : "25",
    "email" : "lee@gmail.com"
}
```

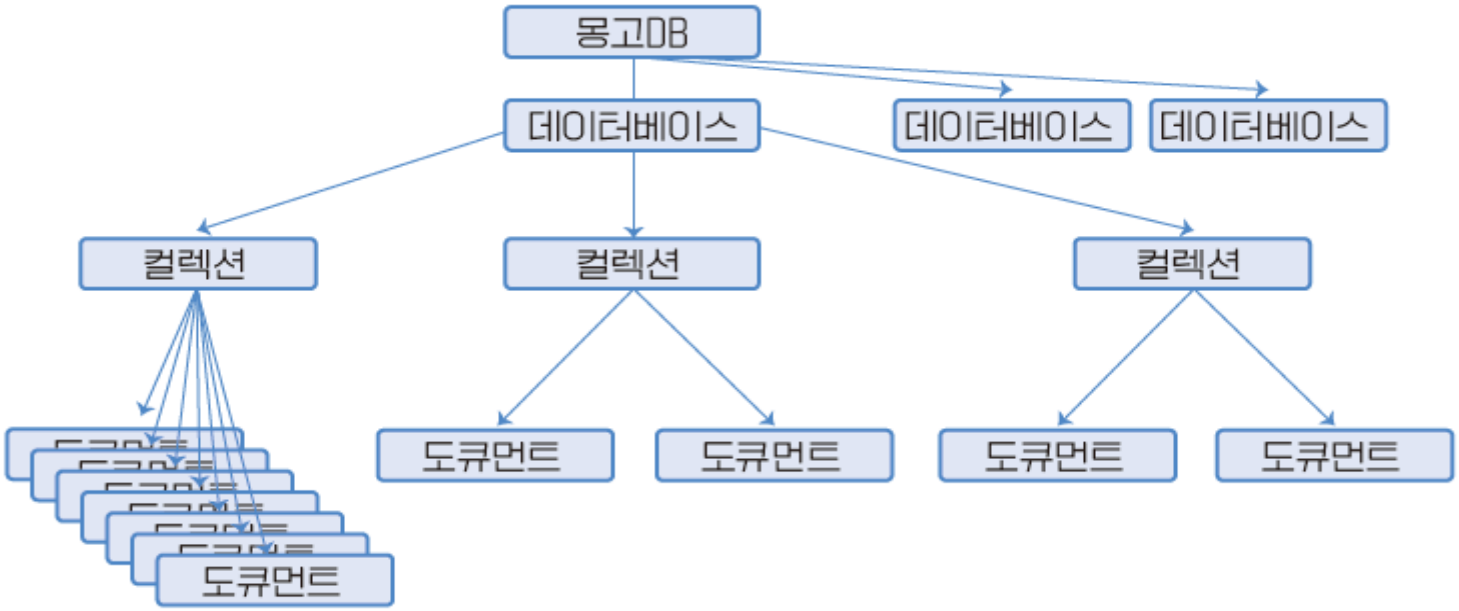
한 명의 정보를 하나의 문서로 저장

- 서버에 데이터베이스를 만들 수도 있고, 클라우드에서 데이터베이스를 사용할 수도 있음.

2. MongoDB

MongoDB의 특징

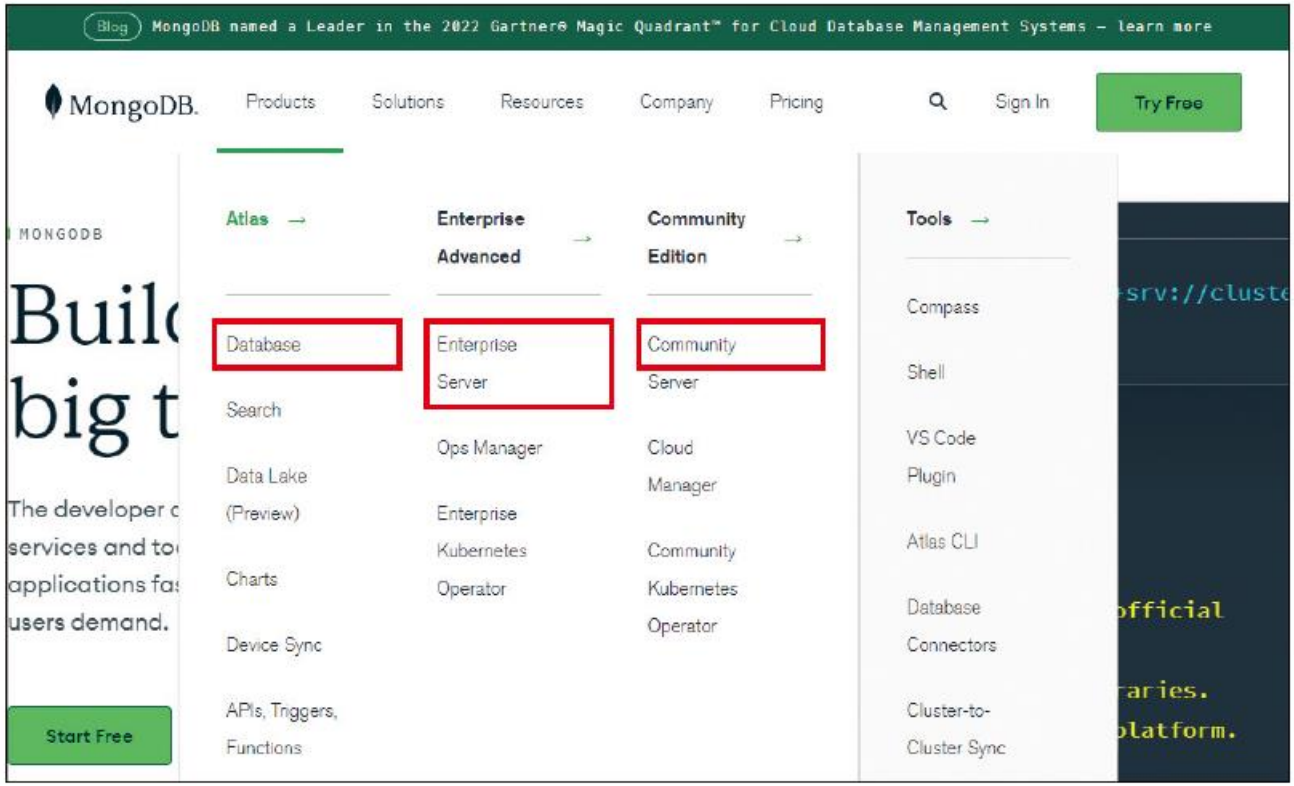
- MongoDB 는 RDB의 테이블과 같은 스키마가 고정된 구조 대신 JSON 형태의 동적 스키마형 도큐먼트 사용
- 도큐먼트는 RDB에서 행(row)에 해당
- 도큐먼트(Document)가 모여서 컬렉션(Collection)을 이룸
- RDB의 테이블과 비슷한 개념



2. MongoDB

몽고DB의 종류

- 설치형 : 몽고DB Community Server, Enterprise Server
- 클라우드 형태 : 몽고DB Atlas에 회원가입하여 사용

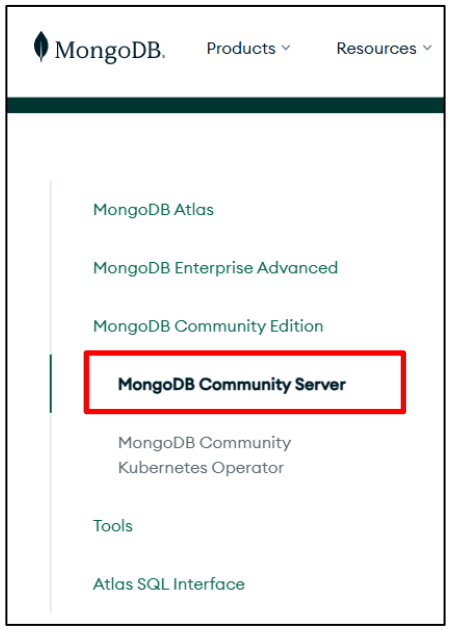


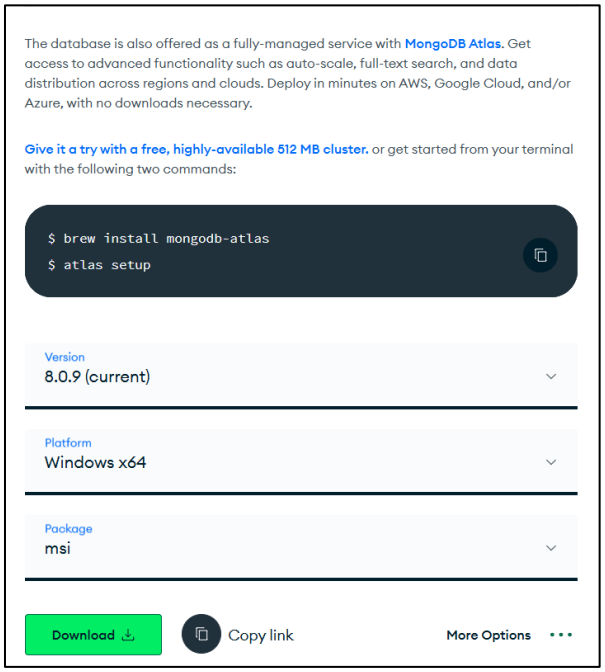
2. MongoDB

몽고DB 설치 : 로컬 컴퓨터에 직접 설치

1. MongoDB 다운로드

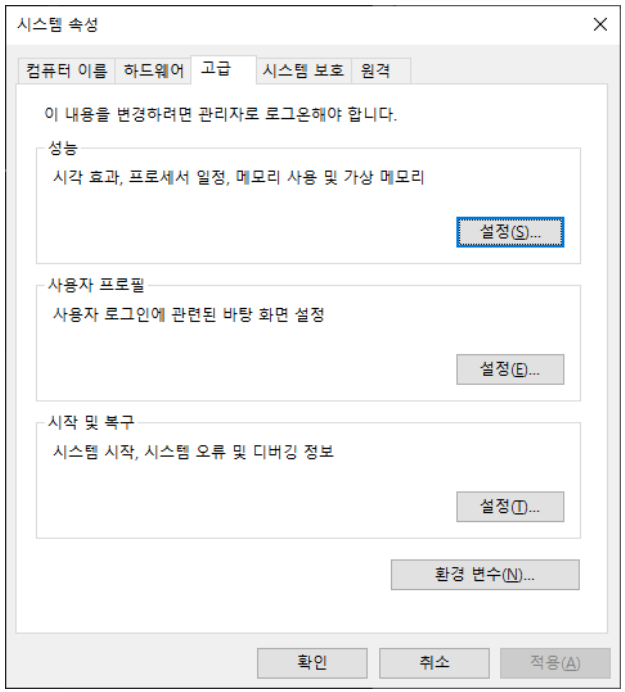
- https://www.mongodb.com/try/download/community
  - Community Server 선택
  - 운영체제에 맞게 Windows / macOS / Linux 선택
  - "MSI 설치 파일" 또는 "ZIP" 선택 후 다운로드

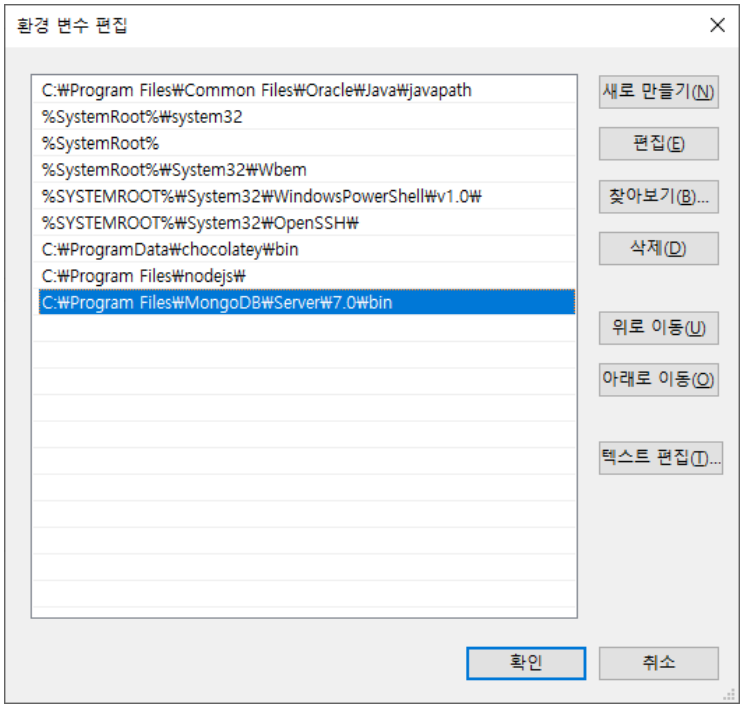




2. 환경 변수 설정

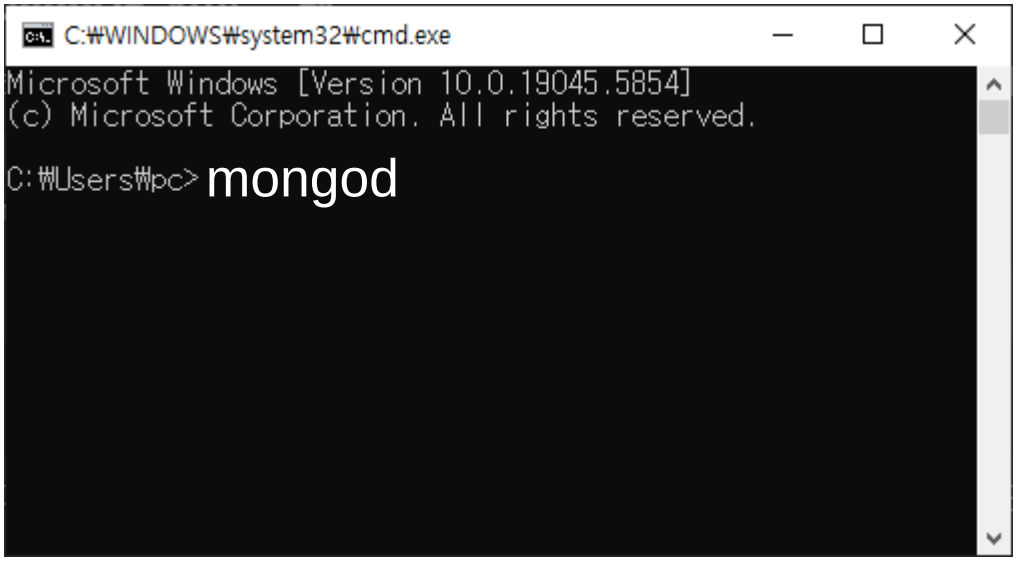
◈ 설치 후, mongod와 mongo 명령어를 터미널에서 바로 사용하려면:
MongoDB 설치 경로(ex: C:\Program Files\MongoDB\Server\7.0\bin)를 시스템 환경 변수의 Path에 추가(윈도우+x → 시스템 → 고급 시스템 설정)





3. MongoDB 서버 실행

서버(Client-Server 구조) 방식이기 때문에 먼저 서버를 실행해야 한다.



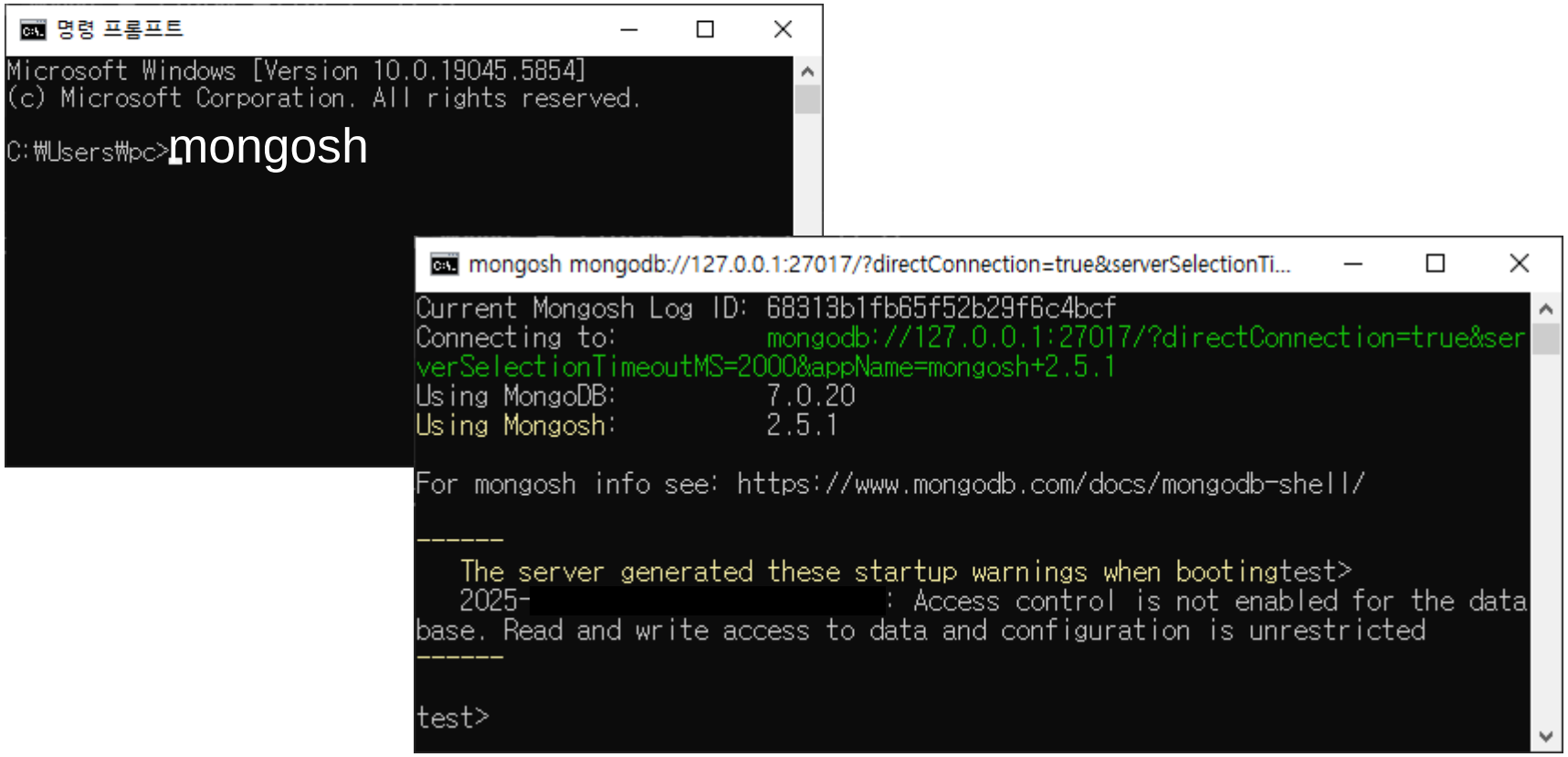
mongod :
mongoDB 실행 명령어

- 기본 포트 27017에서 서버가 실행
- 자동으로 c:₩data₩db 폴더를 찾는다.
- mkdir c:₩data₩db

2. MongoDB

4. MongoDB 클라이언트 실행

새 터미널 창에서 mongosh 실행
Node.js에서는 mongoose나 mongodb 모듈로 연결해서 사용



mongosh:
MongoDB와 대화할 수 있는 명령어 기반 shell

2. MongoDB

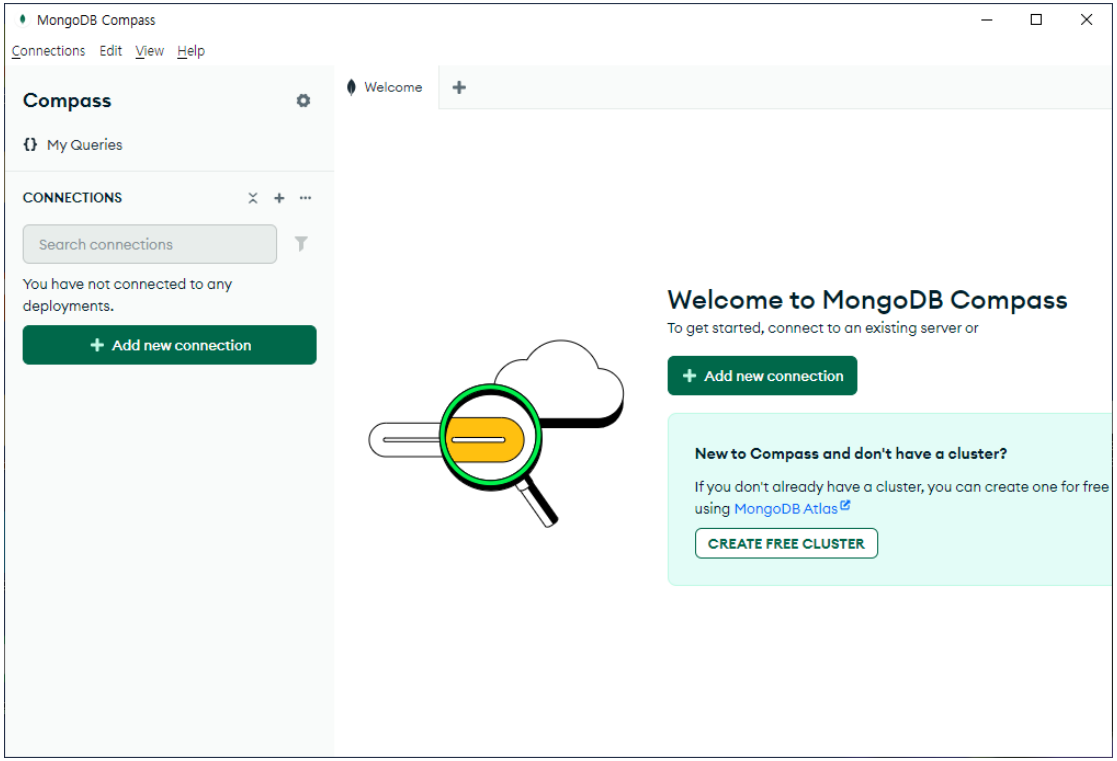
데이터베이스 사용 예

```
use myDB                // myDB라는 데이터베이스로 이동 (없으면 자동 생성)
db.myCollection.insertOne({ name: " Ace", age: 28 })   // 문서 삽입
db.myCollection.find()      // 문서 조회
```

2. MongoDB

MongoDB Compass(GUI 툴) 사용

- MongoDB의 데이터를 시각적으로 관리하고 조작할 수 있게 도와주는 그래픽 도구(GUI)
- 명령어(cmd, 터미널)없이 마우스로 클릭하며 MongoDB를 사용



2. MongoDB

MongoDB Atlas(몽고디비 아틀라스) 란?

- MongoDB를 클라우드에서 쉽게 사용할 수 있도록 만든 서비스
- MongoDB 데이터베이스를 직접 설치하지 않고도 사용할 수 있게 해주는 DBaaS (Database as a Service)
- 몽고DB Atlas 사용

1. MongoDB Atlas 사이트에서 가입
2. 클러스터(데이터베이스 서버) 생성
3. 컬렉션과 문서(데이터) 추가
4. Node.js, Python 등에서 연결하여 데이터 사용

03

몽고DB 데이터베이스와 컬렉션 생성

3. 몽고DB 데이터베이스와 컬렉션 생성

데이터베이스와 컬렉션Collection) 생성

컬렉션 정의

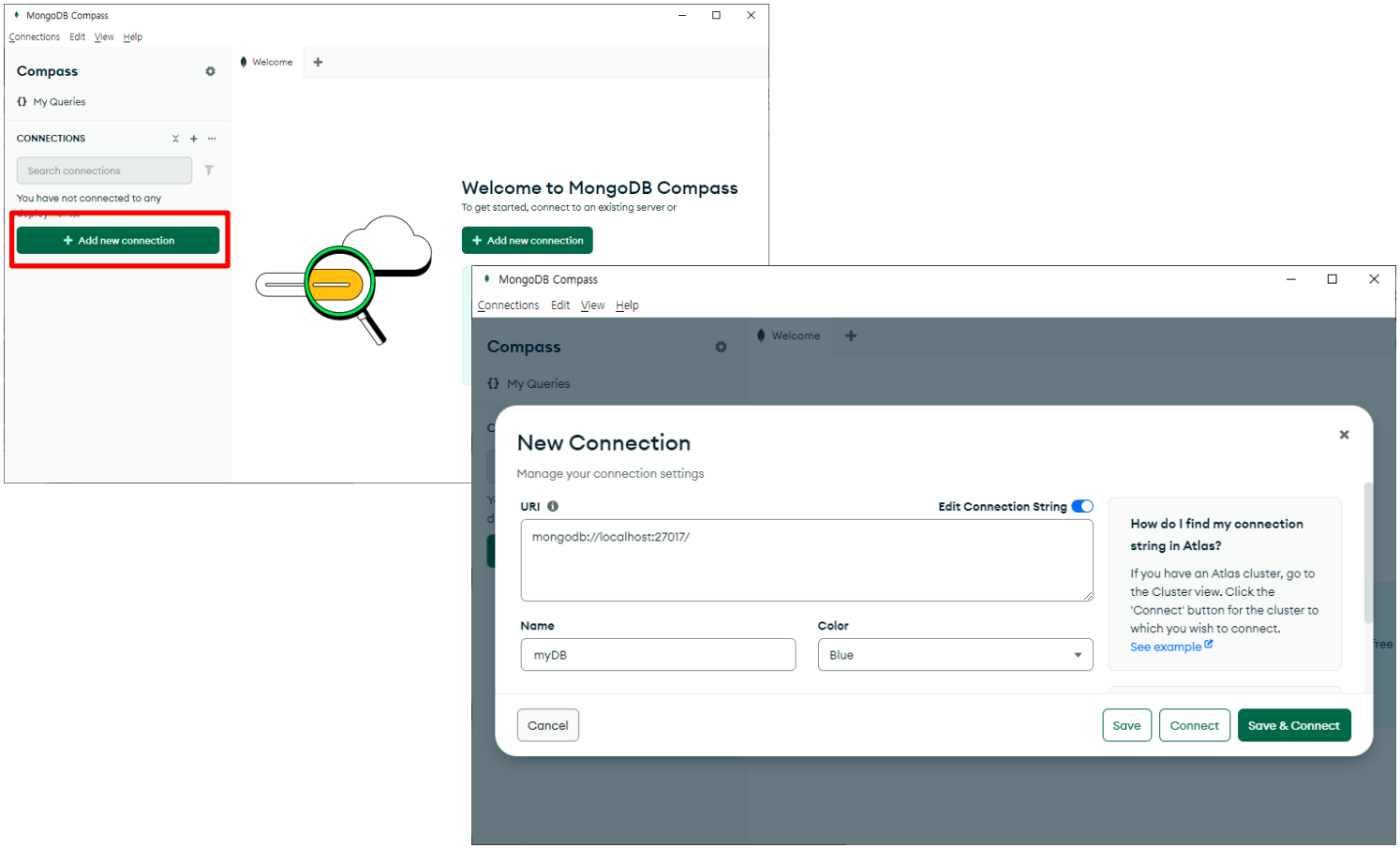
- MongoDB에서 도큐먼트(Document)들을 모아놓은 **그룹**
- 관계형 DB의 table에 해당
- 같은 형식의 데이터를 모아놓음
- 하나의 데이터베이스 안에 여러 컬렉션이 존재

3. 몽고DB 데이터베이스와 컬렉션 생성

데이터베이스와 컬렉션 생성

1. MongoDB 서버에 연결

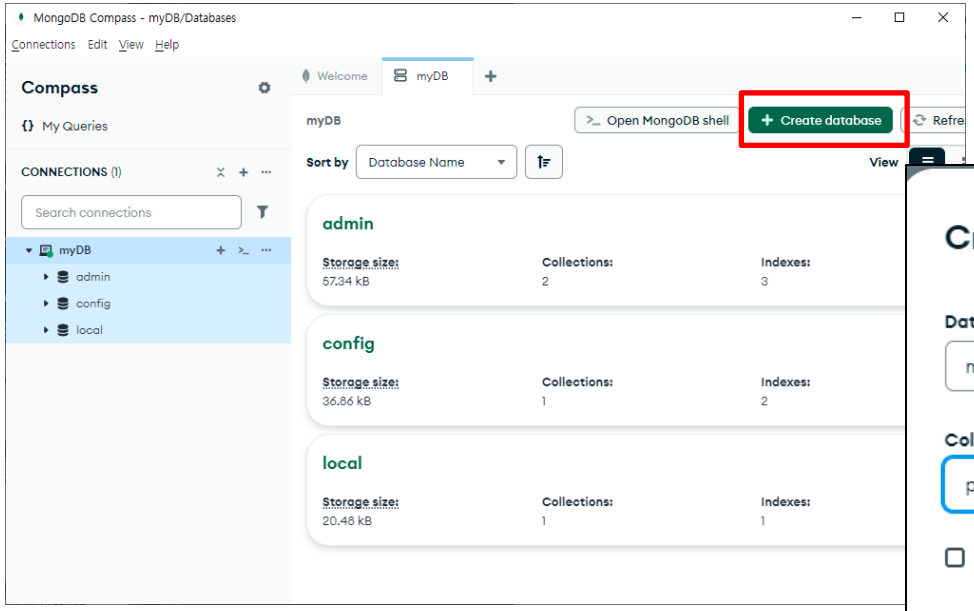
- "Add New Connection" 버튼 클릭

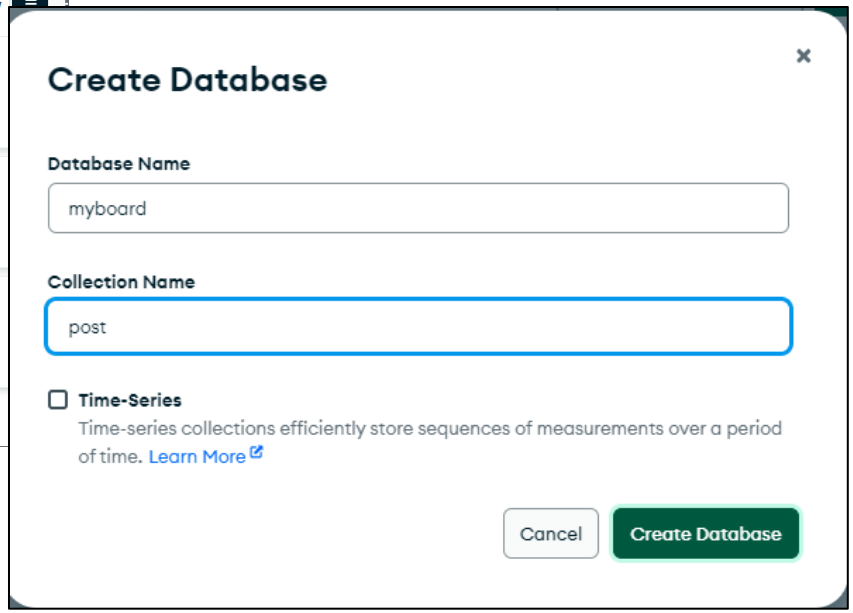


2. 몽고DB 데이터베이스와 컬렉션 생성

데이터베이스 생성

- Create Database 클릭
- DB 이름과 기본 컬렉션 이름을 입력
- 완료하면 DB가 생성되고 바로 사용할 수 있다.





04

도큐먼트 다루기

4. 도큐먼트(Document) 다루기

- 도큐먼트 정의
  - MongoDB에서 하나의 데이터를 저장하는 단위
  - 관계형 DB의 행(row) 또는 레코드(record)에 해당
  - 도큐먼트는 JSON과 유사한 형식 BSON으로 저장
  - 각 도큐먼트에는 자동으로 생성되는 _id 필드가 있어 고유하게 식별됨

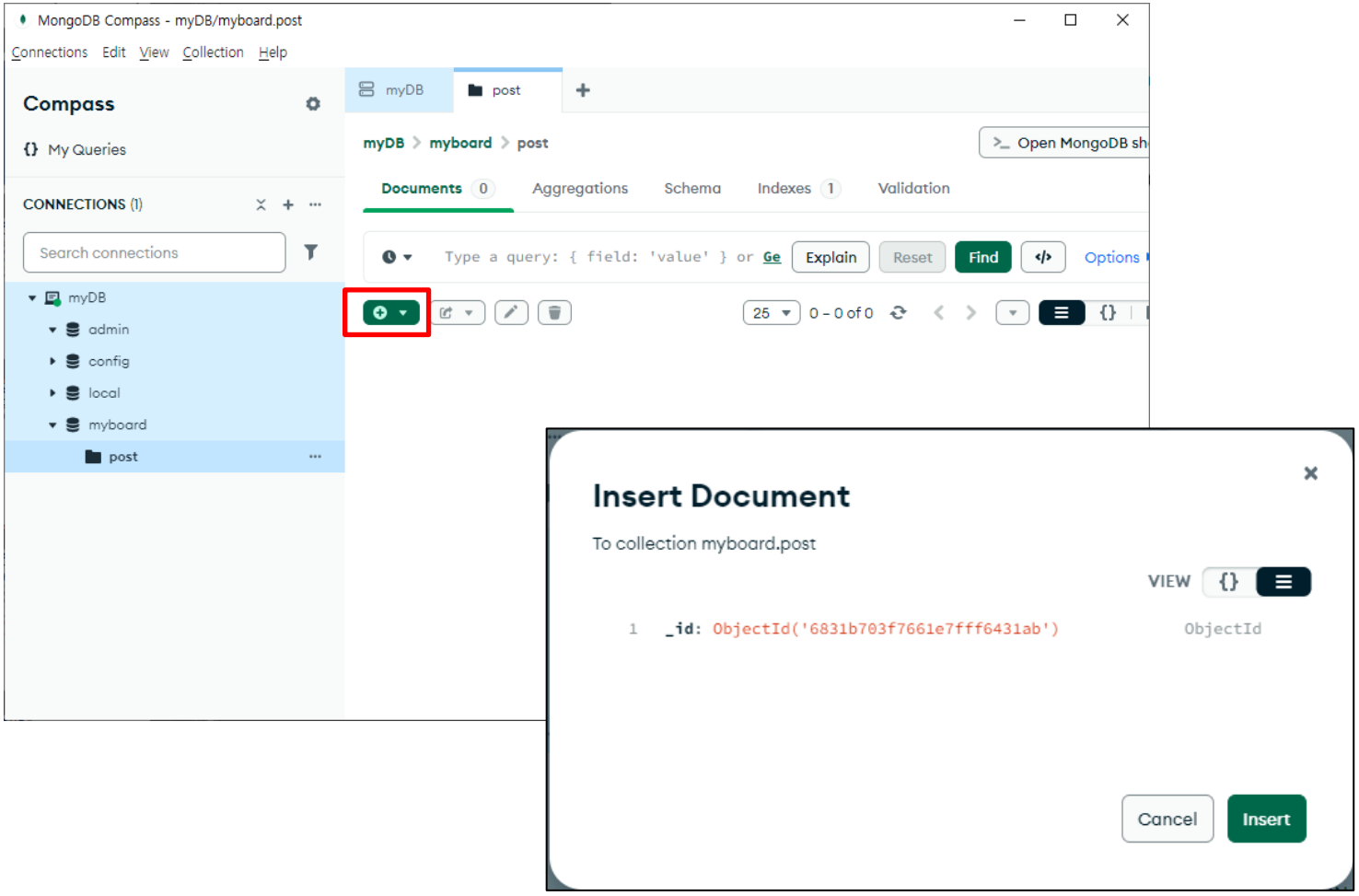
BSON

- Binary JSON의 약자
- JSON(JavaScript Object Notation)을 이진(binary) 형식으로 인코딩한 것
- MongoDB에서 주로 사용

4. 도큐먼트 다루기

데이터 추가

- 데이터를 채우기 위해서 〈add data〉 버튼 클릭



4. 도큐먼트 다루기

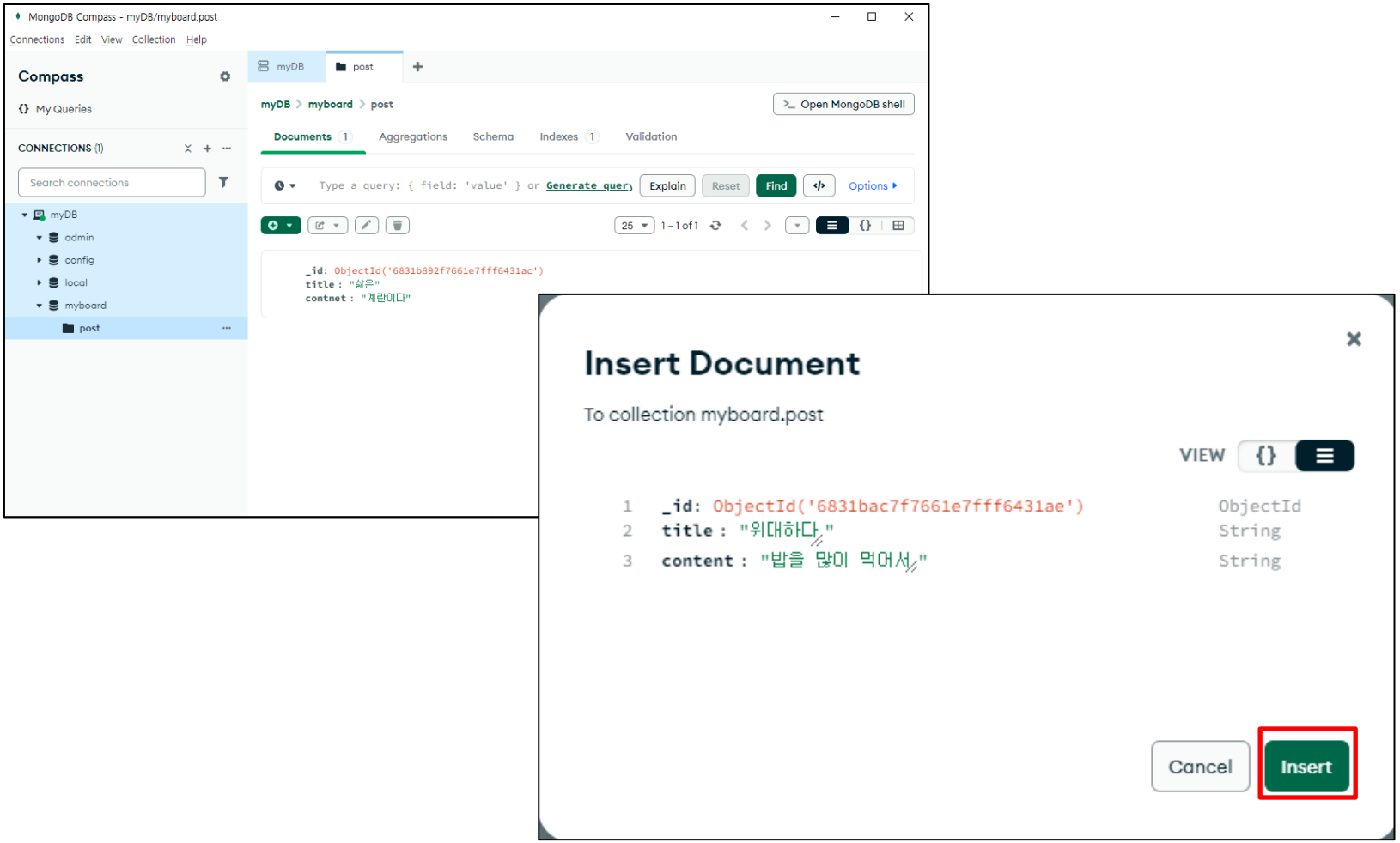
데이터 추가

- _id 아래 2번째 줄에 :를 기준으로 왼쪽에 'key', 오른쪽에 'value' 입력
- 데이터를 추가하기 위해 해당 줄에 마우스 커서를 위치하면 '+' 아이콘이 표시되는데 이때 이 아이콘을 클릭하면 데이터 추가 가능

4. 도큐먼트 다루기

데이터 추가

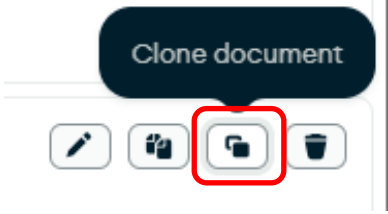
- 데이터가 추가된 것을 확인하고 다시 〈Insert Document〉 버튼을 클릭하여 같은 방법으로 추가

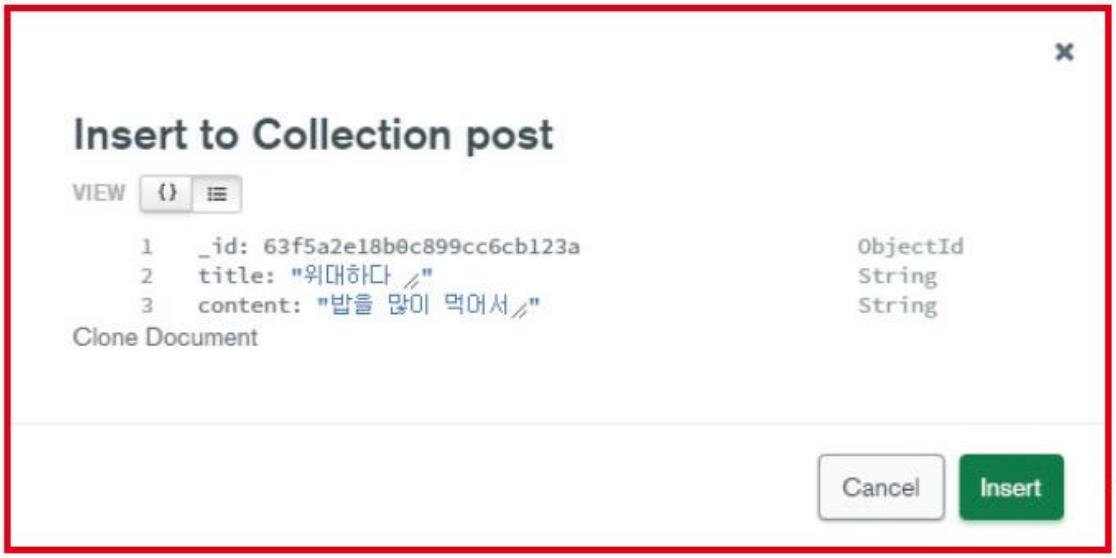


4. 도큐먼트 다루기

데이터 복제

- ‘clone document’ 아이콘을 클릭 클릭하여 데이터 복제
- 같은 형식의 데이터를 여러 개 추가하고 싶을 때 사용하면 편리



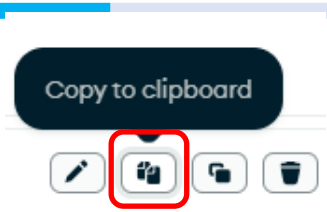


클릭

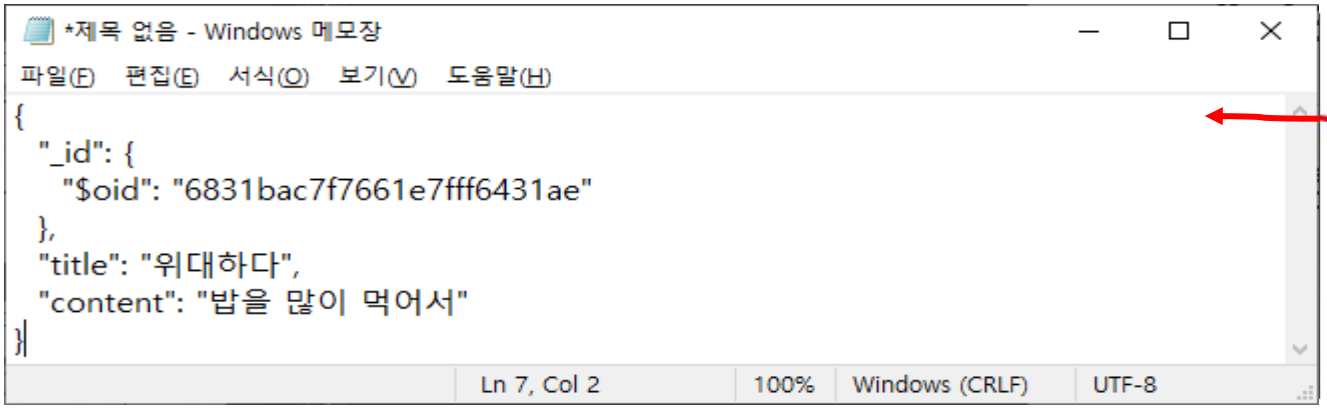
4. 도큐먼트 다루기

데이터 복사

- ‘Copy document’ 아이콘을 클릭하여 데이터 복사
- 복사한 데이터를 메모장에 붙여넣으면 JSON 형식의 도큐먼트 데이터 표시됨



Ctrl + V

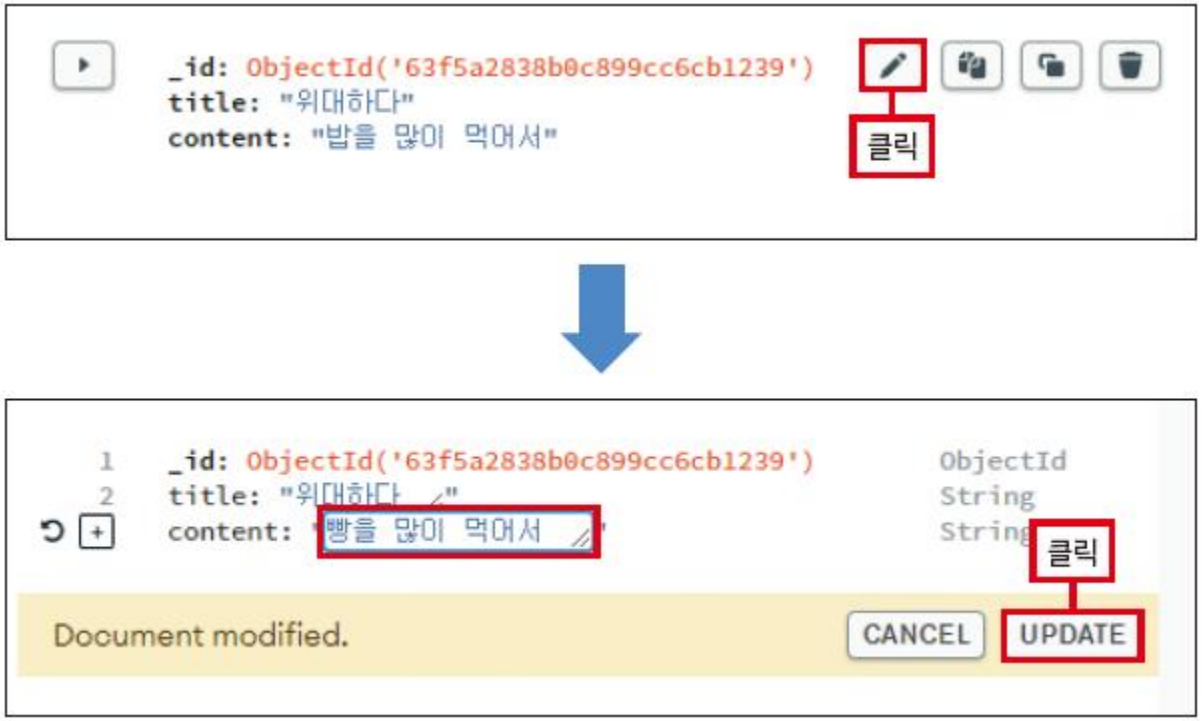


클릭

4. 도큐먼트 다루기

데이터 수정

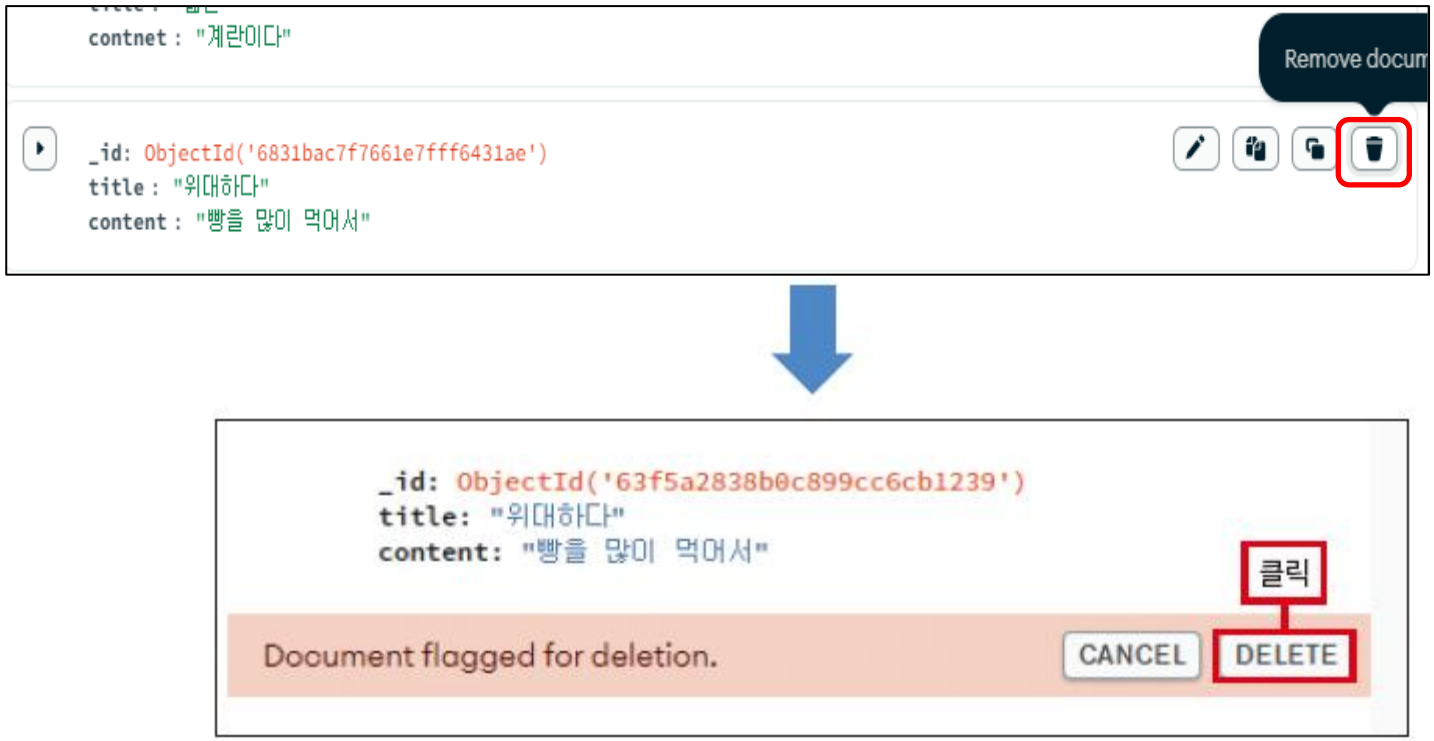
- ‘Edit document’ 아이콘을 클릭하면 기존 데이터 수정 가능
- 데이터 수정을 완료할 때 〈UPDATE〉 버튼 클릭



4. 도큐먼트 다루기

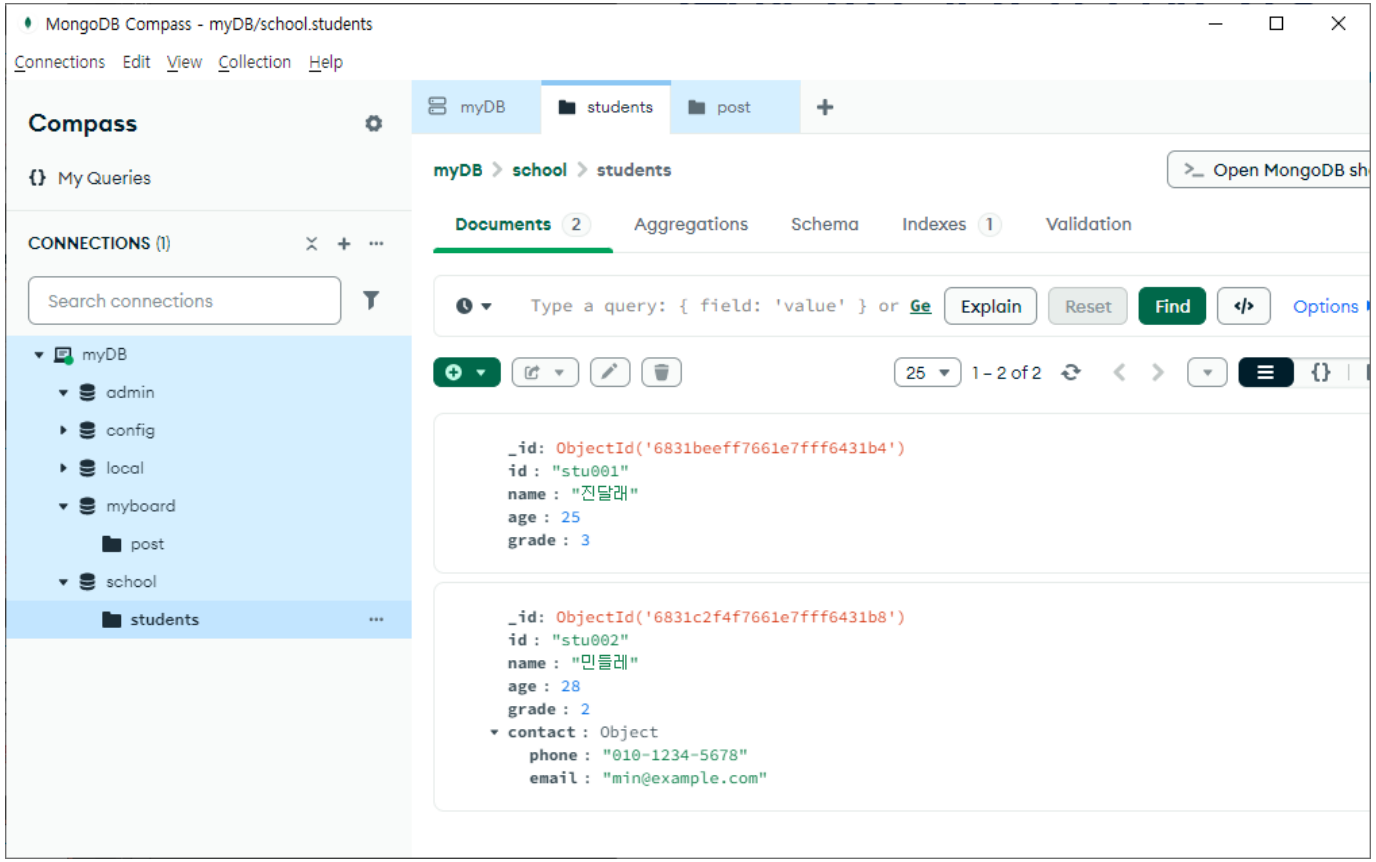
데이터 삭제

- 삭제할 데이터에서 ‘Remove document’ 아이콘 클릭



〈학생 정보 관리 시스템〉 실습

- 데이터베이스 : school
- 컬렉션 : students
- 도큐먼트



05

몽고DB와 Node.js 연결

5. 몽고DB와 Node.js 연결

mongodb 모듈

- MongoDB에서 공식으로 제공하는 Node.js용 기본 드라이버
- MongoDB와 직접 통신하여 데이터베이스 작업을 할 수 있다.
- Mongoose보다 간단, 모델/스키마 기능은 없음.

```
$ npm install mongodb --save
```

5. 몽고DB와 Node.js 연결

MongoClient 객체

- MongoDB에 연결을 생성하는 역할을 하는 객체
- MongoDB 서버와 연결을 하고, 데이터베이스와 컬렉션에 접근하거나 조작할 수 있게 해주는 객체
- MongoClient 주요 메서드
  - connect() : MongoDB 서버에 연결
  - db(dbName) : 연결된 클라이언트에서 특정 데이터베이스에 접근
  - db.collection(collectionName) : 해당 데이터베이스의 컬렉션 객체에 접근

```
const { MongoClient } = require('mongodb');
```

실습) MongoDB에 연결하여 데이터베이스(school)의 컬렉션(students)에서 데이터를 읽고 출력하는 코드 작성

```
const { MongoClient } = require("mongodb");

//// MongoDB 서버 주소 및 설정
const url = "mongodb://127.0.0.1:27017";
const dbName = "school";
const collectionName = "students";
```

서버 없이 MongoDB에서 데이터 읽고 확인

Promise 방식

```
const client = new MongoClient(url);
// MongoDB 연결
client.connect()
  .then((client) => {
    const db = client.db(dbName); //데이터베이스 선택
    const collection = db.collection(collectionName); // 컬렉션 접근
    // .find() : 데이터 읽기, .toArray() : 배열 형태로 가져오기
    return collection.find().toArray()
    .then((result) => {
        console.log("데이터:", result);
        return client; // client를 다음 then에 넘김
      });
  })
  .then((client) => {
    return client.close();
  })
  .catch((err) => {
    console.error("에러 발생:", err);
  });
```

〈실행 결과〉

```
데이터: [
  {
    _id: new ObjectId('6831beeff7661e7fff6431b4'),
    id: 'stu001',
    name: '진달래',
    age: 25,
    grade: 3
  },
  {
    _id: new ObjectId('6831c2f4f7661e7fff6431b8'),
    id: 'stu002',
    name: '민들레',
    age: 28,
    grade: 2,
    contact: { phone: '010-1234-5678', email: 'min@example.com' }
  }
]
```

5. 몽고DB와 Node.js 연결

async 함수

- JavaScript에서 비동기 작업을 보다 간결하고 직관적으로 처리할 수 있게 해주는 함수
- async 키워드를 함수 앞에 붙이면 함수가 항상 Promise를 반환
- 함수 내부에서 await 키워드를 사용하여 비동기 작업이 동기처럼 순차적으로 처리되도록 할 수 있음.
- 서버 요청, 파일 읽기, DB 작업 등 비동기 작업을 순서대로 처리하고 싶을 때
- await 키워드 :
  - Promise가 끝날 때까지 기다렸다가 결과 반환
  - 반드시 async 함수 안에서만 사용

실습) MongoDB에 연결하여 데이터베이스(school)의 컬렉션(students)에서 데이터를 읽고 출력하는 코드 작성

```
const { MongoClient } = require("mongodb");

//// MongoDB 서버 주소 및 설정
const url = "mongodb://127.0.0.1:27017";
const dbName = "school";
const collectionName = "students";
```

async/await 방식

- ✓ try-catch로 에러 처리 간편
- ✓ 위에서 아래로 순서대로 실행

```
async function readData() {
  const client = new MongoClient(url);
  try {
    await client.connect();
    const db = client.db(dbName);
    const collection = db.collection(collectionName);

    const result = await collection.find().toArray();
    console.log("데이터:", result);
  } catch (err) {
    console.error("에러 발생:", err);
  } finally {
    await client.close();
  }
}

readData();
```

실습) Express와 MongoDB를 사용하여 데이터베이스(school)의 컬렉션(students)에 접근하기 위한 설정 코드

```
1   // mongodb 모듈에서 MongoClient 불러오기
2   //const mongoclient = require("mongodb").MongoClient
3   const { MongoClient } = require("mongodb");
4   const express = require("express");
5
6   const app = express();
7
8   // MongoDB 서버 주소 및 설정
9   const url = "mongodb://127.0.0.1:27017";
10  const dbName = "school";
11  const collectionName = "students";
```

```
13  let db; //데이터베이스 접근 객체(확장성 고려하여 외부에 선언)
14  let collection;
15
16  // MongoDB 연결
17  const client = new MongoClient(url);
18  client.connect(url)
19    .then((client) => {
20      console.log("MongoDB 연결 성공");
21      db = client.db(dbName); //데이터베이스 선택
22      collection = db.collection(collectionName); // 컬렉션 접근
23
24      // 서버 실행
25      app.listen(8080, () => {
26        console.log("서버 실행 중...");
27      });
```

Node.js 서버가 실행될 때 MongoDB 데이터베이스에 연결하고, 연결이 성공하면 Express 웹 서버를 시작

```
28      //.find() : 데이터 읽기, .toArray() :배열 형태로 가져오기
29      return db.collection(collectionName).find().toArray();
30    })
31    .then((result) => {
32      console.log("students 컬렉션 내용:");
33      console.log(result);
34    })
35    .catch((err) => {
36      console.error("에러 발생:", err);
37    });
```

〈실행 결과〉

```
PS C:\Users\pc\Desktop\backend_programming\ch_12> node mongodb_doc
1. MongoDB 연결 성공
2. 8080 서버 실행 중...
students 컬렉션 내용:
[
  {
    _id: new ObjectId('6831beeff7661e7fff6431b4'),
    id: 'stu001',
    name: '진달래',
    age: 25,
    grade: 3
  },
  {
    _id: new ObjectId('6831c2f4f7661e7fff6431b8'),
    id: 'stu002',
    name: '민들레',
    age: 28,
    grade: 2,
    contact: { phone: '010-1234-5678', email: 'min@example.com' }
  }
]
```

5. 몽고DB와 Node.js 연결

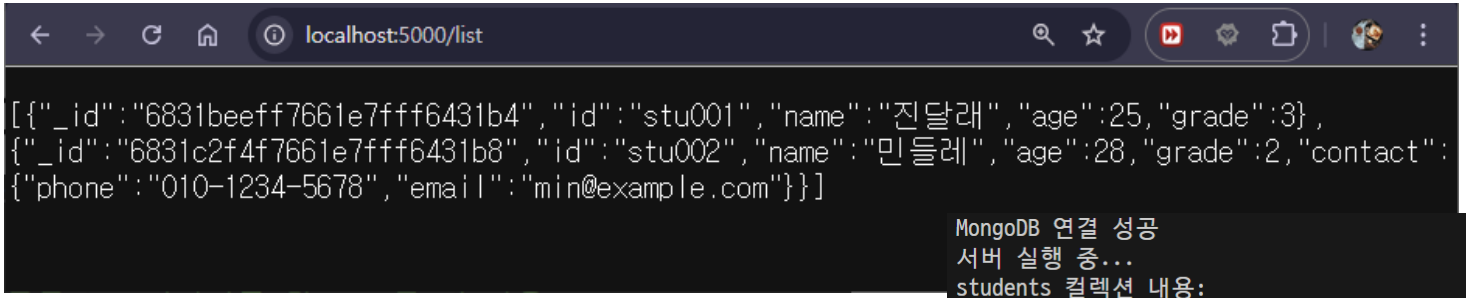
- /list 요청 시 데이터 조회(p399)
  - /list 요청 시 students 컬렉션의 데이터를 조회해서 브라우저로 응답하는 코드

```
1   const { MongoClient } = require("mongodb");
2   const express = require("express");
3
4   const app = express();
5
6   // MongoDB 설정
7   const url = "mongodb://127.0.0.1:27017";
8   const dbName = "school";
9   const collectionName = "students";
10
```

```
let db;
let collection;

// MongoDB 연결
const client = new MongoClient(url);
client.connect()
  .then((connectedClient) => {
    console.log("MongoDB 연결 성공");
    db = connectedClient.db(dbName);
    collection = db.collection(collectionName);

    // 서버 실행
    app.listen(5000, () => {
      console.log("서버 실행 중...");
    });
  })
  .catch((err) => {
    console.error("MongoDB 연결 실패:", err);
  });
```

```
31  // /list 요청 시 데이터 조회
32  app.get("/list", (req, res) => {
33    if (!collection) {
34      return res.status(500).send("컬렉션에 접근할 수 없습니다");
35    }
36
37    collection.find().toArray()
38      .then((result) => {
39        console.log("students 컬렉션 내용:");
40        console.log(result); // 서버 콘솔 출력
41        res.json(result); // 브라우저에 JSON 응답
42      })
43      .catch((err) => {
44        console.error("데이터 조회 실패:", err);
45        res.status(500).send("데이터 조회 중 에러 발생");
46      });
47  });
```

```
PS C:\Users\pc\Desktop\backend_programming\ch_12> nodemon mongodb_4
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node mongodb_4.js`
MongoDB 연결 성공
서버 실행 중...
```




```
MongoDB 연결 성공
서버 실행 중...
students 컬렉션 내용:
[
  {
    _id: new ObjectId('6831beeff7661e7fff6431b4'),
    id: 'stu001',
    name: '진달래',
    age: 25,
    grade: 3
  },
  {
    _id: new ObjectId('6831c2f4f7661e7fff6431b8'),
    id: 'stu002',
    name: '민들레',
    age: 28,
    grade: 2,
    contact: { phone: '010-1234-5678', email: 'min@example.com' }
  }
]
```

6. Mongoose

Mongoose

- Node.js 환경에서 MongoDB를 쉽게 다룰 수 있도록 도와주는 ODM(Object Data Modeling) 라이브러리
  - ODM : 자바스크립트 객체와 MongoDB의 문서(Document)를 연결해주는 도구
- MongoDB는 스키마가 없지만, Mongoose를 통해 데이터 구조를 만들 수 있다.
- Node.js와 MongoDB 사이에서 데이터 구조를 명확히 하고, 효율적이고 안정적인 방식으로 데이터를 다루게 도와주는 도구

```
$ npm install mongoose
```

6. Mongoose

MongoDB vs Mongoose

	MongoDB	Mongoose
연결 객체	mongoClient	mongoose.connect()
스키마	없음	new mongoose.Schema() 사용
모델	직접 컬렉션 접근	mongoose.model()로 정의 후 사용
종료	client.close()	mongoose.connection.close()

실습) MongoDB에 연결하고, 한 명의 학생 데이터를 저장하기 위한 Mongoose 코드

```
1   const mongoose = require("mongoose");
2
3   // Mongoose가 MongoDB에 연결하도록 지시하는 함수
4   mongoose.connect("mongodb://127.0.0.1:27017/testdb");
5   //   .then(() => console.log("몽고DB 연결 성공"))
6   //   .catch((err) => console.error("연결 실패:", err));
7
8   // 스키마 정의
9   const studentSchema = new mongoose.Schema({
10    name: String,
11    age: Number,
12    grade: Number,
13  });
```

MongoDB에 'testdb'라는 데이터베이스를 연결한 뒤, students 컬렉션에 학생(진달래) 도큐먼트를 저장

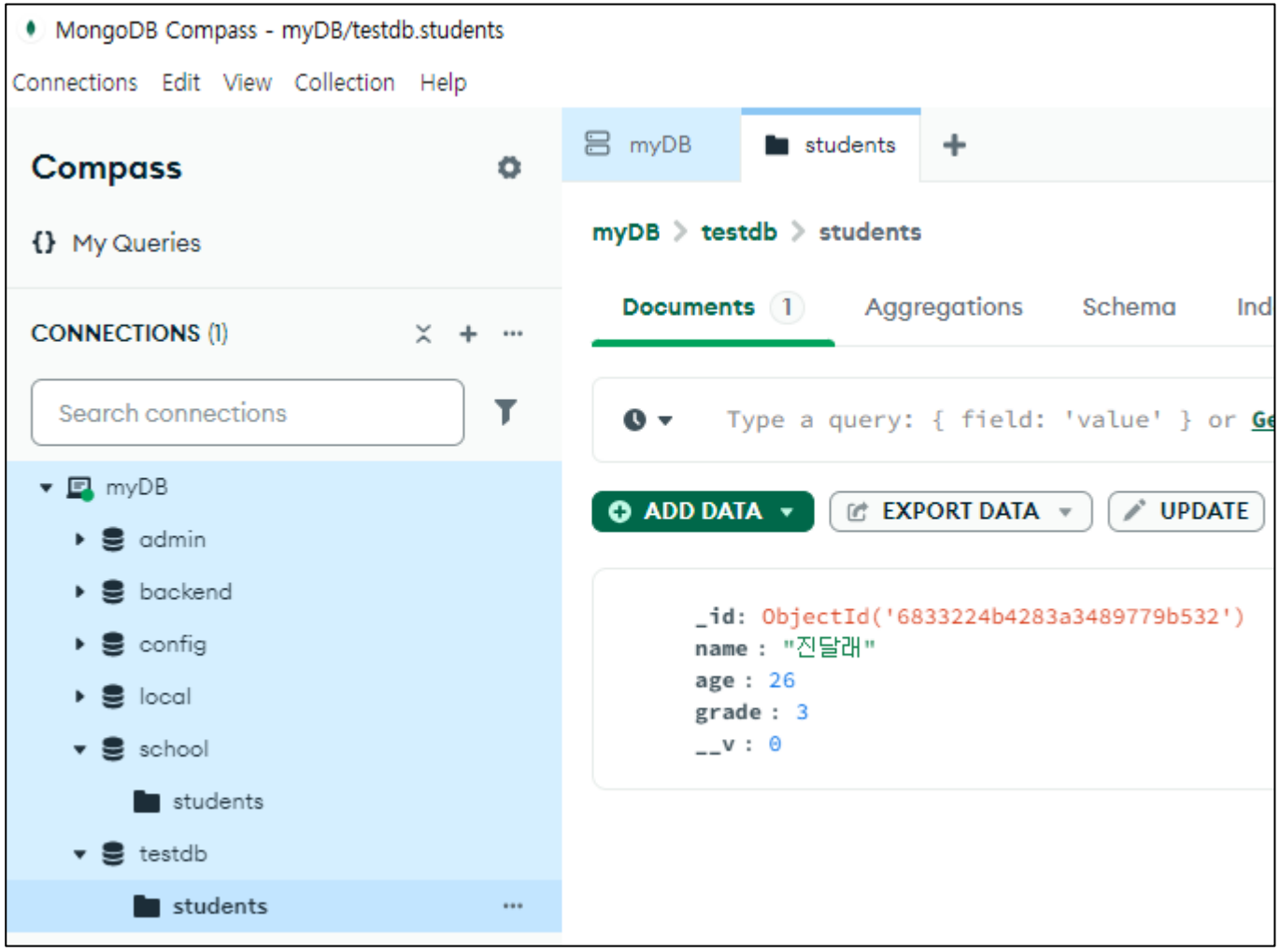
MongoDB에서는 이 모델을 통해 students 컬렉션에 접근함. (자동 복수형 처리)

```
15  // 모델 생성(Student:모델 이름, studentSchema:스키마 객체)
16  const Student = mongoose.model("Student", studentSchema);
17
18  // 데이터 저장
19  const newStudent = new Student({ name: "진달래", age: 26, grade: 3 });
20  newStudent.save()
21    .then((savedStudent) => {
22      //savedStudent:MongoDB에 저장된 후 반환된 문서(document)
23      console.log("저장 완료:", savedStudent);
24    })
25    .catch((err) => console.error("에러 발생:", err));
```

〈실행 결과〉

```
저장 완료: {
  name: '진달래',
  age: 26,
  grade: 3,
  _id: new ObjectId('68331bc34435a7510d1f973f'),
  __v: 0
}
```

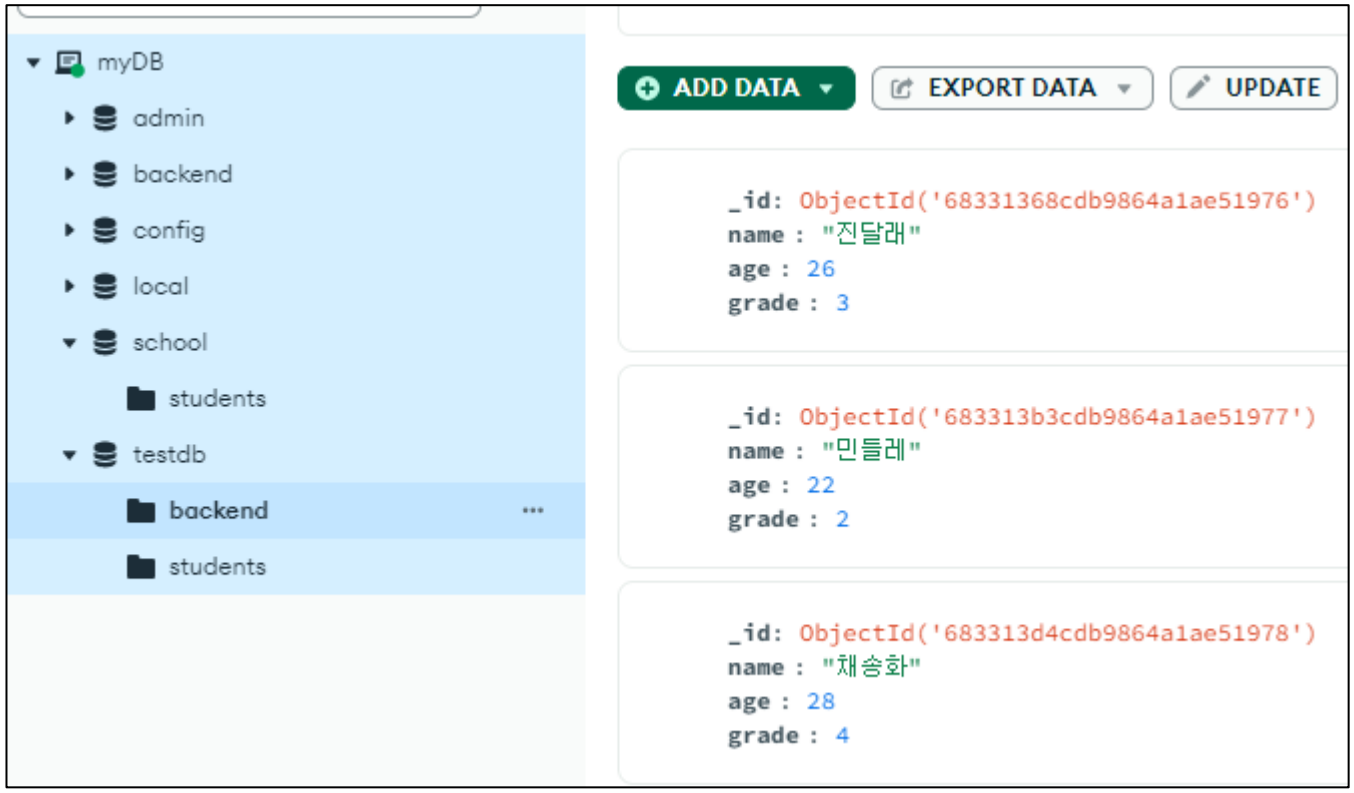
MongoDB에서 저장된 문서를 출력할 때 _id, __v 같은 시스템 필드도 같이 출력



Ctrl+R (새로고침) or

```
// 데이터 저장
const newStudent = new Student({ name: "진달래", age: 26, grade: 3 });
newStudent.save()
  .then((savedStudent) => {
    const plainStudent = savedStudent.toObject(); // 문서를 일반 객체로 변환
    delete plainStudent._id;
    delete plainStudent.__v;
    console.log("저장 완료:", plainStudent);
  })
  .catch((err) => console.error("에러 발생:", err));
```

```
[nodemon] starting `node mongoose_2.js`
저장 완료: { name: '진달래', age: 26, grade: 3 }
```


myDB
admin
backend
config
local
school
students
testdb
backend
students
ADD DATA
EXPORT DATA
UPDATE
_id: ObjectId('68331368cdb9864a1ae51976')
name : "진달래"
age : 26
grade : 3
_id: ObjectId('683313b3cdb9864a1ae51977')
name : "민들레"
age : 22
grade : 2
_id: ObjectId('683313d4cdb9864a1ae51978')
name : "채송화"
age : 28
grade : 4

실습)MongoDB 데이터베이스에서 'backend' 컬렉션에 저장된 문서(도큐먼트) 조회해서 출력하는 코드

```
1   const mongoose = require("mongoose");
2
3   mongoose
4     .connect("mongodb://127.0.0.1:27017/testdb")
5     .then(() => {
6       console.log("몽고DB 연결 성공");
7
8       // 스키마 및 모델 정의
9       const backendSchema = new mongoose.Schema({
10        name: String,
11        age: Number,
12        grade: Number,
13      });
```

```
    // 정확히 'backend' 컬렉션을 지정
    const Backend = mongoose.model("Backend", backendSchema, "backend");

    // 도큐먼트 조회(.find().lean() – Mongoose 전용)
    return Backend.find().lean();
  })
  .then((backend) => {
    const cleanStudents = backend.map(({ _id, __v, ...rest }) => rest);
    console.log("저장된 학생 목록:", cleanStudents);
  })
  .catch((err) => {
    console.error("에러 발생:", err);
  })
  .finally(() => {
    mongoose.disconnect();
  });
```

.map() : 배열의 각 요소를 원하는 방식으로 변형해서 새로운 배열을 만드는 메서드

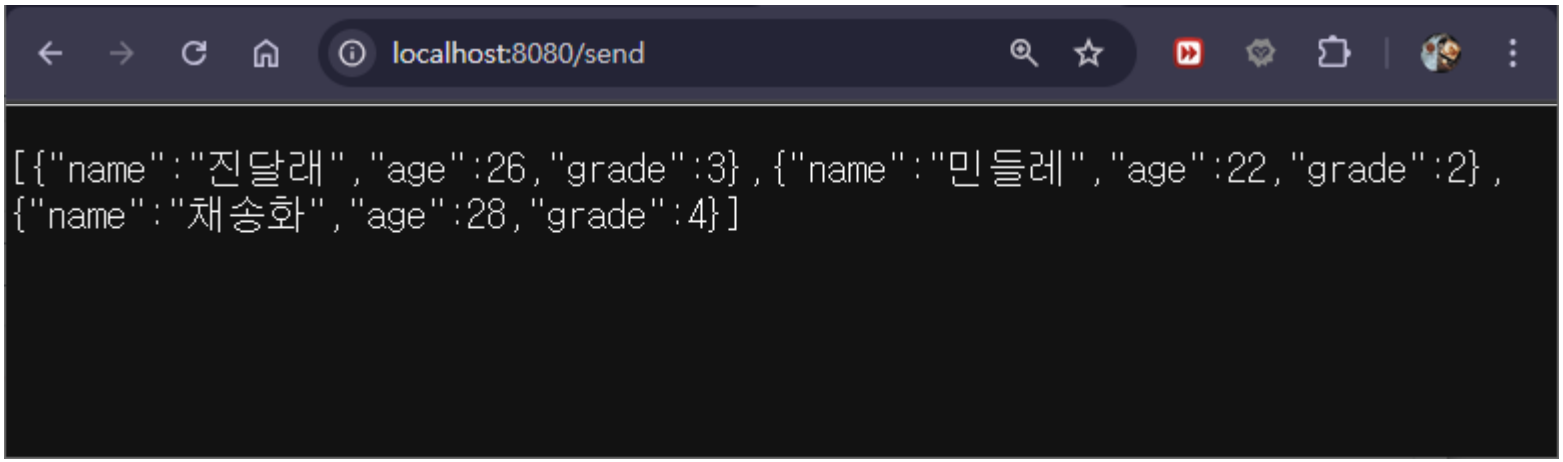
실습) MongoDB에 저장된 컬렉션(backend)의 데이터를 브라우저에서 확인할 수 있는 API 서버를 만드는 코드

클라이언트가 /send 경로로 요청 → 서버는 MongoDB에서 데이터 꺼내어 → JSON으로 보내줌.

```
1   const express = require("express");
2   const mongoose = require("mongoose");
3
4   const app = express();
5
6   // 스키마 및 모델 정의
7   const backendSchema = new mongoose.Schema({
8     name: String,
9     age: Number,
10    grade: Number,
11  });
12  const Backend = mongoose.model("Backend", backendSchema, "backend");
```

```
14  // 1.MongoDB 연결 2.서버 실행
15  mongoose.connect("mongodb://127.0.0.1:27017/testdb")
16    .then(() => {
17      console.log("몽고DB 연결 성공");
18
19      // 서버 실행
20      app.listen(8080, () => {
21        console.log("서버 실행 중...");
22      });
23    })
24    .catch((err) => {
25      console.error("몽고DB 연결 실패:", err);
26    });
27
28  // /send 경로에서 도큐먼트 응답
29  app.get("/send", (req, res) => {
30    Backend.find().lean()
31      .then((documents) => {
32        const cleanDocs = documents.map(({ _id, __v, ...rest }) => rest);
33        res.json(cleanDocs);
34      })
35      .catch((error) => {
36        res.status(500).send("데이터 조회 중 에러 발생");
37      });
38  });
```

```
몽고DB 연결 성공
서버 실행 중...
```



```
// MongoDB 연결 및 서버 실행 (async/await 사용)
async function startServer() {
  try {
    await mongoose.connect("mongodb://127.0.0.1:27017/testdb");
    console.log("몽고DB 연결 성공");

    app.listen(8080, () => {
      console.log("서버 실행 중...");
    });
  } catch (err) {
    console.error("몽고DB 연결 실패:", err);
  }
}
startServer();

// /send 경로에서 도큐먼트 응답
app.get("/send", async (req, res) => {
  try {
    const documents = await Backend.find().lean();
    const cleanDocs = documents.map(({ _id, __v, ...rest }) => rest);
    res.json(cleanDocs);
  } catch (error) {
    res.status(500).send("데이터 조회 중 에러 발생");
  }
});
```

07

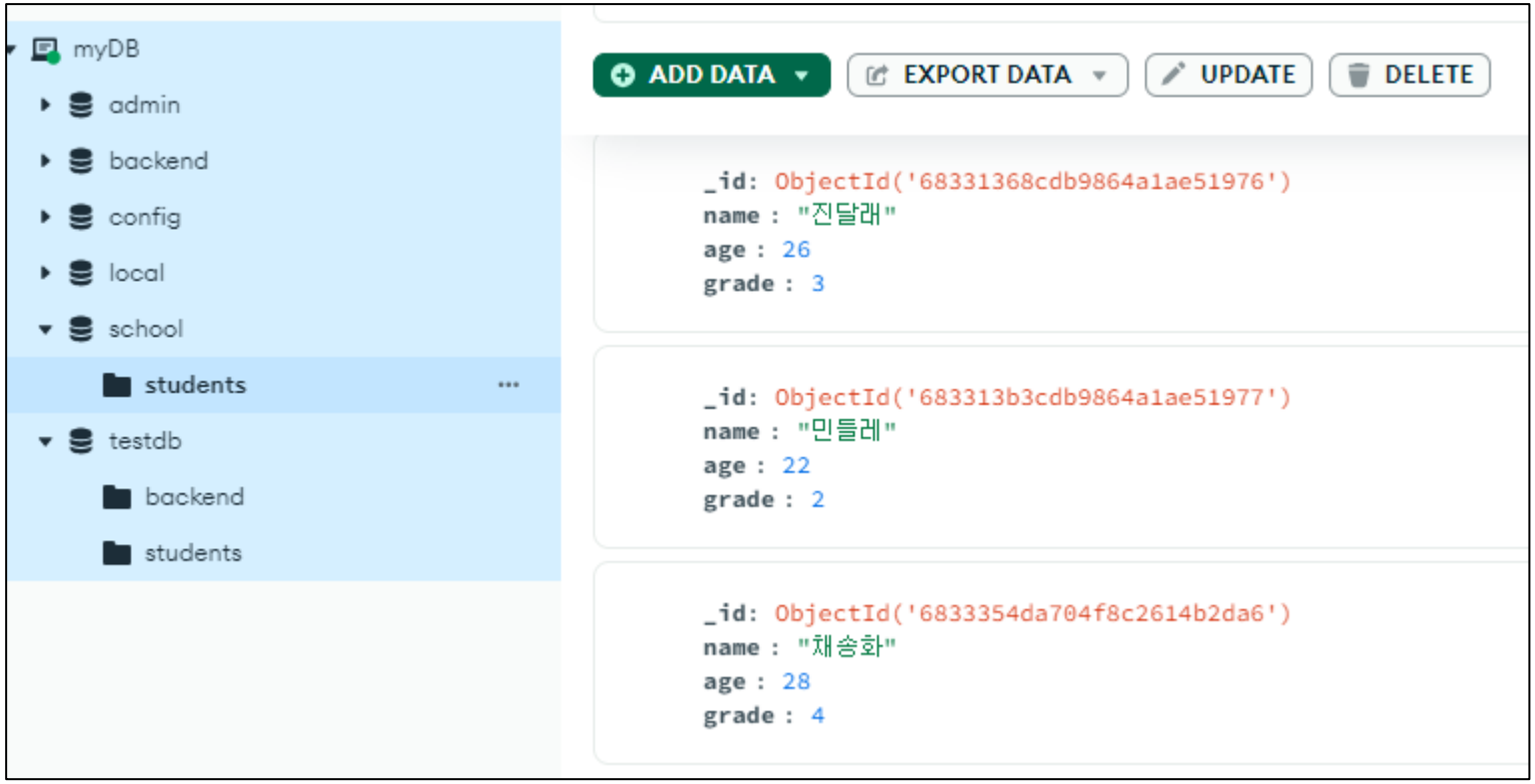
CRUD

7. CRUD

CRUD란?

- CRUD는 Create(생성), Read(읽기), Update(갱신), Delete(삭제)를 묶어서 일컫는 용어
- 일반적인 애플리케이션에서 데이터를 다룰 때 갖추어야 할 기본 기능

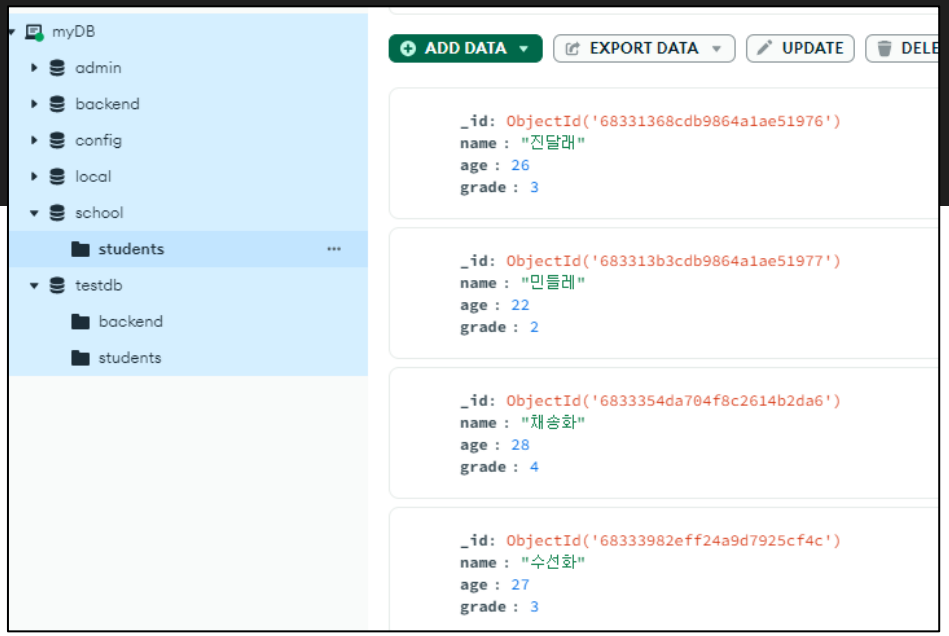
이름	기능
Create	추가
Read	조회
Update	갱신
Delete	삭제


myDB
admin
backend
config
local
school
students
testdb
backend
students
ADD DATA
EXPORT DATA
UPDATE
DELETE
_id: ObjectId('68331368cdb9864a1ae51976')
name : "진달래"
age : 26
grade : 3
_id: ObjectId('683313b3cdb9864a1ae51977')
name : "민들레"
age : 22
grade : 2
_id: ObjectId('6833354da704f8c2614b2da6')
name : "채송화"
age : 28
grade : 4

실습) 데이터 추가

```
const { MongoClient } = require("mongodb");

const url = "mongodb://127.0.0.1:27017";
const dbName = "school";
const collectionName = "students";

const client = new MongoClient(url);
let db;
let collection;
async function run() {
  try {
    await client.connect();
    console.log("MongoDB 연결 성공");

    db = client.db(dbName);
    collection = db.collection(collectionName);

    console.log("1.-------------------------------------");
    // 1. Create (문서 추가)
    const insertResult = await collection.insertOne({
      name: "수선화",
      age: 27,
      grade: 3,
    });
```

```
26      // 삽입된 문서 전체 출력
27      const insertedDoc = await collection.findOne({
28        _id: insertResult.insertedId,
29      });
30      console.log("삽입된 문서 내용:", insertedDoc);
31    } catch (err) {
32      console.error("에러 발생:", err);
33    } finally {
34      await client.close();
35      console.log("MongoDB 연결 종료");
36    }
37  }
38
39  run();
```



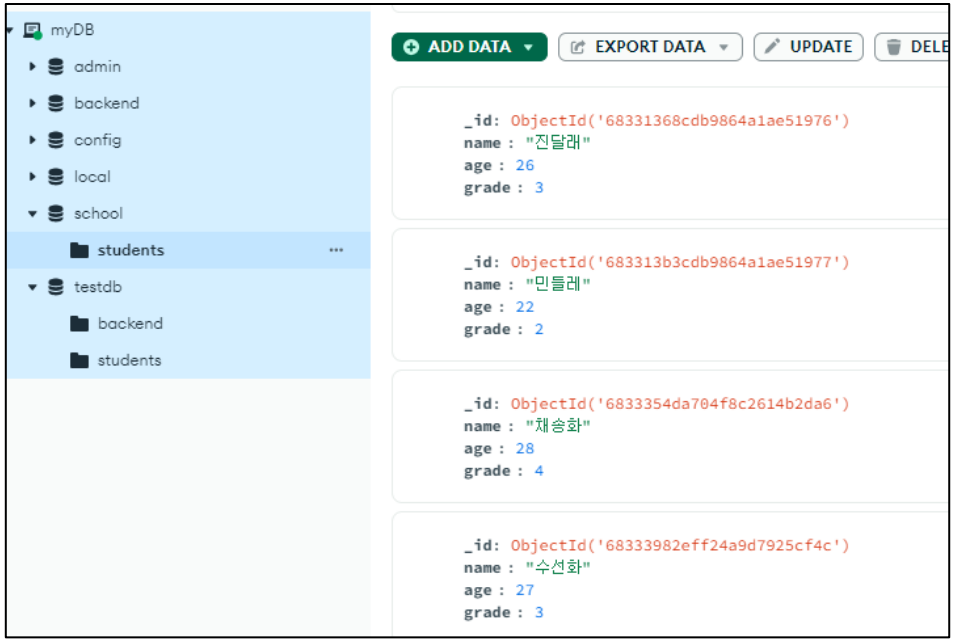
실습) 데이터 조회

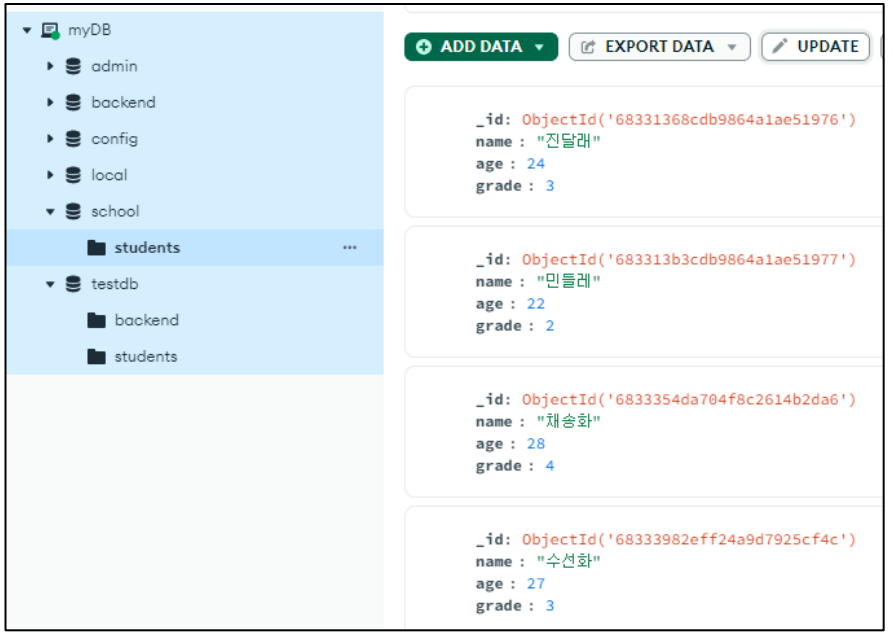
```
console.log("2.----------------------------------------");
// 2. Read (문서 조회)
const students = await collection.find().toArray();
console.log("전체 학생 목록:", students);
```

```
2.----------------------------------------
전체 학생 목록: [
  {
    _id: new ObjectId('68331368cdb9864a1ae51976'),
    name: '진달래',
    age: 26,
    grade: 3
  },
  {
    _id: new ObjectId('683313b3cdb9864a1ae51977'),
    name: '민들레',
    age: 22,
    grade: 2
  },
  {
    _id: new ObjectId('6833354da704f8c2614b2da6'),
    name: '채송화',
    age: 28,
    grade: 4
  },
  {
    _id: new ObjectId('68333e6d46fa759ae9e717b2'),
    name: '수선화',
    age: 27,
    grade: 3
  }
]
```

실습) 데이터 수정

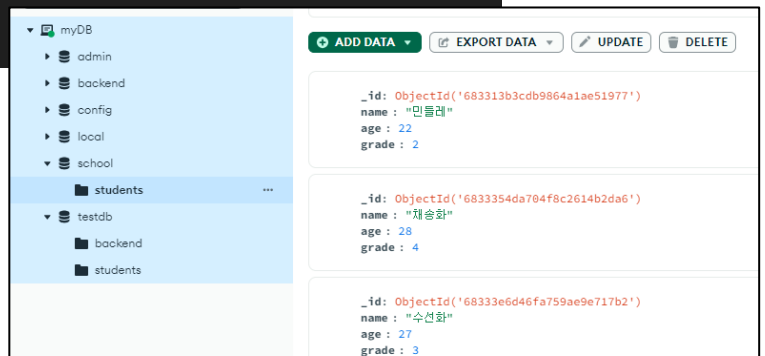
```
console.log("2. 문서 수정 ----------------------------------");
// 2. Update (문서 수정)
const updateResult = await collection.updateOne(
  { name: "진달래" }, //필터
  { $set: { age: 24 } } // 수정 내용
);
const updatedDoc = await collection.findOne({ name: "진달래" });
console.log("수정된 문서:", updatedDoc);
```





실습) 데이터 삭제

```
console.log("4.----------------------------------------");
// 4. Delete (문서 삭제)
// 삭제할 문서를 먼저 찾기
const docToDelete = await collection.findOne({
  name: "진달래",
});

if (docToDelete) {
  // 문서가 존재하면 삭제
  const deleteResult = await collection.deleteOne({
    name: "진달래",
  });
  console.log("삭제된 문서:", docToDelete);
  console.log("삭제된 문서 수:", deleteResult.deletedCount);
} else {
  console.log("삭제할 문서를 찾을 수 없습니다.");
}
```



수고했습니다.