Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with

게시글 추가 기능 수정

```
router.post('/write', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      id: req.session.userId,
      title: req.body.title,
      content: req.body.content,
    };
    boardDB.writeArticle(newArticle, (data) => {
      if (data.affectedRows >= 1) {
        res.status(200);
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

```
  writeArticle: (newArticle, cb) => {
    boardDB.query(
      `INSERT INTO mydb1.board (USERID, TITLE, CONTENT) VALUES ('${newArticle.id}',
'${newArticle.title}', '${newArticle.content}')`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      },
    );
  },
```

게시글 수정 기능

수정

게시글 수정 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/modify/:id', isLogin, (req, res) => {
  boardDB.getArticle(req.params.id, (data) => {
    if (data.length > 0) {
      res.render('dbBoard_modify', { selectedArticle: data[0] });
    }
  });
});
```

```
router.post('/modify/:id', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    db.modifyArticle(req.params.id, req.body, (data) => {
      console.log(data);
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 수정 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

게시글 삭제 기능 수정

게시글 삭제 기능

- 로그인이 안되어 있으면 게시글 삭제 요청이 안되도록 설정 그 외의 부분은 동일하므로 건들 필요가 없음!

```
router.delete('/delete/:id', isLogin, (req, res) => {
  db.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.affectedRows >= 1) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

쿠키를 사용한

자동 로그인 구현

```
router.post('/', (req, res) => {
  db.userCheck(req.body.id, (data) => {
    if (data.length > 0) {
      if (data[0].PASSWORD === req.body.password) {
        req.session.login = true;
        req.session.userId = req.body.id;
        // 쿠키 발행
        res.cookie('user', req.body.id, {
          maxAge: 1000 * 10,
          httpOnly: true,
          signed: true,
        });
        res.redirect('/dbBoard');
      } else {
        res.status(400);
        res.send(
          '비밀번호가 다릅니다.<br><a href="/login">로그인으로 이동</a>',
        );
      }
    } else {
      res.status(400);
      res.send(
        '회원 ID를 찾을 수 없습니다.<br><a href="/login">로그인으로 이동</a>',
      );
    }
  });
});
```

쿠키를 사용한 자동 로그인

- isLogin 함수에 쿠키에 의한 로그인 처리 기능 추가

```
const isLogin = (req, res, next) => {
  if (req.session.login || req.signedCookies.user) {
    next();
  } else {
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
};
```

```
// 로그 아웃 처리
router.get('/logout', (req, res) => {
  req.session.destroy((err) => {
    if (err) throw err;
    res.clearCookie('user');
    res.redirect('/');
  });
});
```

DOTENV

.ENV

DOTENV, 중요 정보를 관리하는 모듈

- DOTENV 는 중요한 정보(서버 접속 정보 등등)를 외부 코드에서 확인이 불가능 하도록 도와주는 모듈입니다!
- 일단 설치 합시다
- Npm i dotenv
- 모듈 호출하기

```
require('dotenv').config();
```

DOTENV, 중요 정보를 관리하는 모듈

- .env 파일을 최상단 폴더에 만들기
- 중요한 정보를 .env 파일에 저장

```
PORT = 4000
DB_USER = root
DB_PASSWORD = dlrladk
DB_DATABASE = mydb
```

- 해당 정보가 필요한 곳에서 process.env.저장명 으로 사용

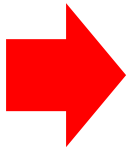
```
const PORT = process.env.PORT;
```

구조분해 할당 문법

Destructuring Assignment

배열 구조 분해 할당

```
const arr = [1, 2, 3];

const one = arr[0];
const two = arr[1];
const three = arr[2];

console.log(one, two, three);
```



```
const arr = [1, 2, 3];

const [one, two, three] = arr;

console.log(one, two, three); // 1 2 3
```

- 배열의 각 요소를 추출하여 바로 변수로 할당
- 추출되는 기준은 배열의 순서에 따라서 할당 된다

객체 구조 분해 할당

```
const obj = { firstName: "효석", lastName: "이" };

const firstName = obj.firstName;
const lastName = obj.lastName;

console.log(firstName, lastName); // 효석 이
```

```
const obj = { firstName: "효석", lastName: "이" };

const { lastName, firstName } = obj;

console.log(firstName, lastName); // 효석 이
```

전개 구문

Spread Syntax(...)

전개 구문 – 객체 합치기

```
const tetzData = {
  name: '이효석',
  gender: 'M',
};

const tetzInfo = {
  nickName: 'gotetz',
  email: 'xenosign@gmail.com',
};

const tetz = {
  ...tetzData,
  ...tetzInfo,
};

console.log(tetz);
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/KDT/__수업 자료/정규 수업/29/backend
$ node test.js
{
  name: '이효석',
  gender: 'M',
  nickName: 'gotetz',
  email: 'xenosign@gmail.com'
}
```

전개 구문 – 배열 합치기

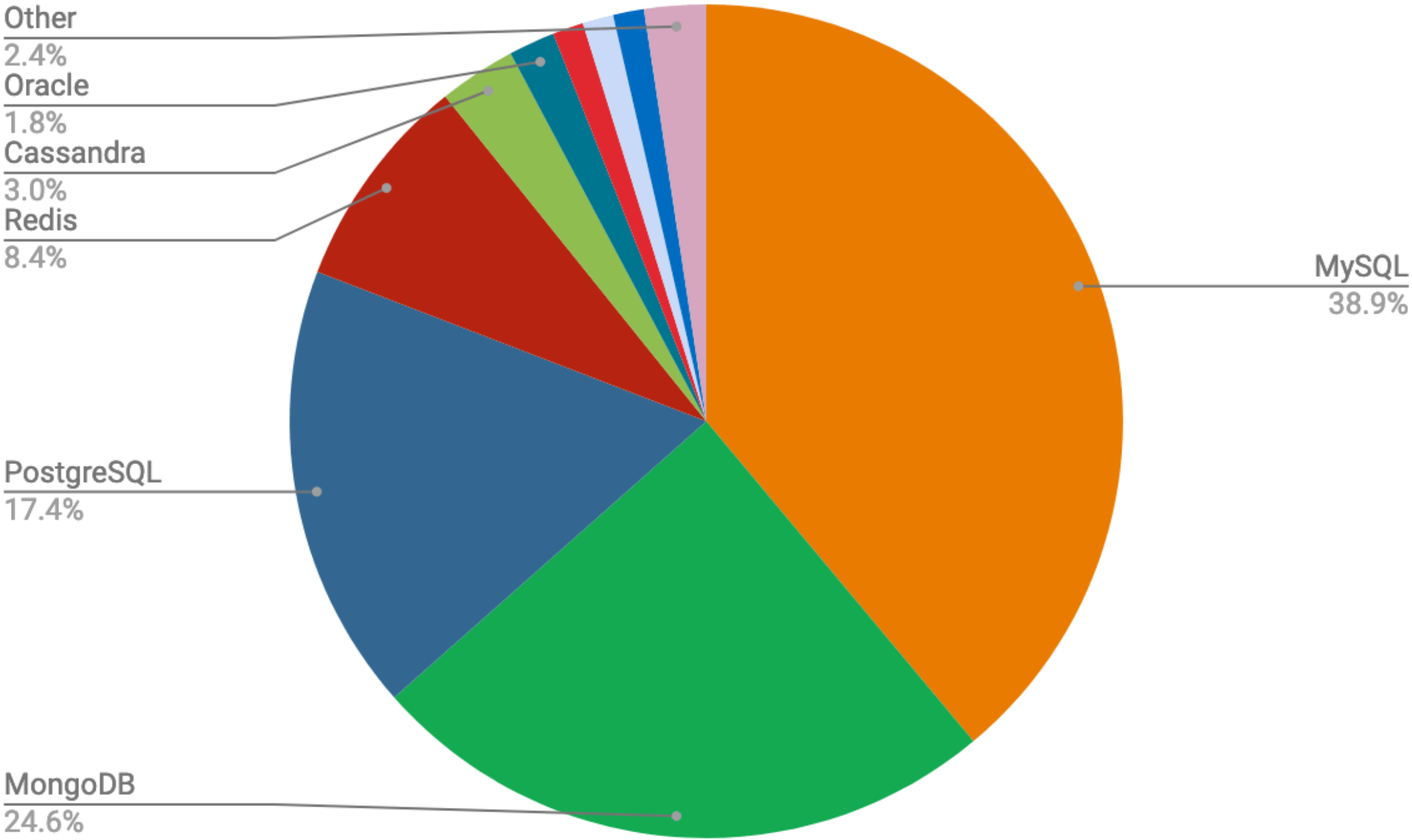
```
const arr1 = [1, 2, 3, 4, 5];
const arr2 = ['6', '7', '8'];

const merge = [...arr1, ...arr2];

console.log(merge);
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/KDT/__수업 자료/정규 수업/29/backend
$ node test.js
[
  1, 2,   3,   4,
  5, '6', '7', '8'
]
```

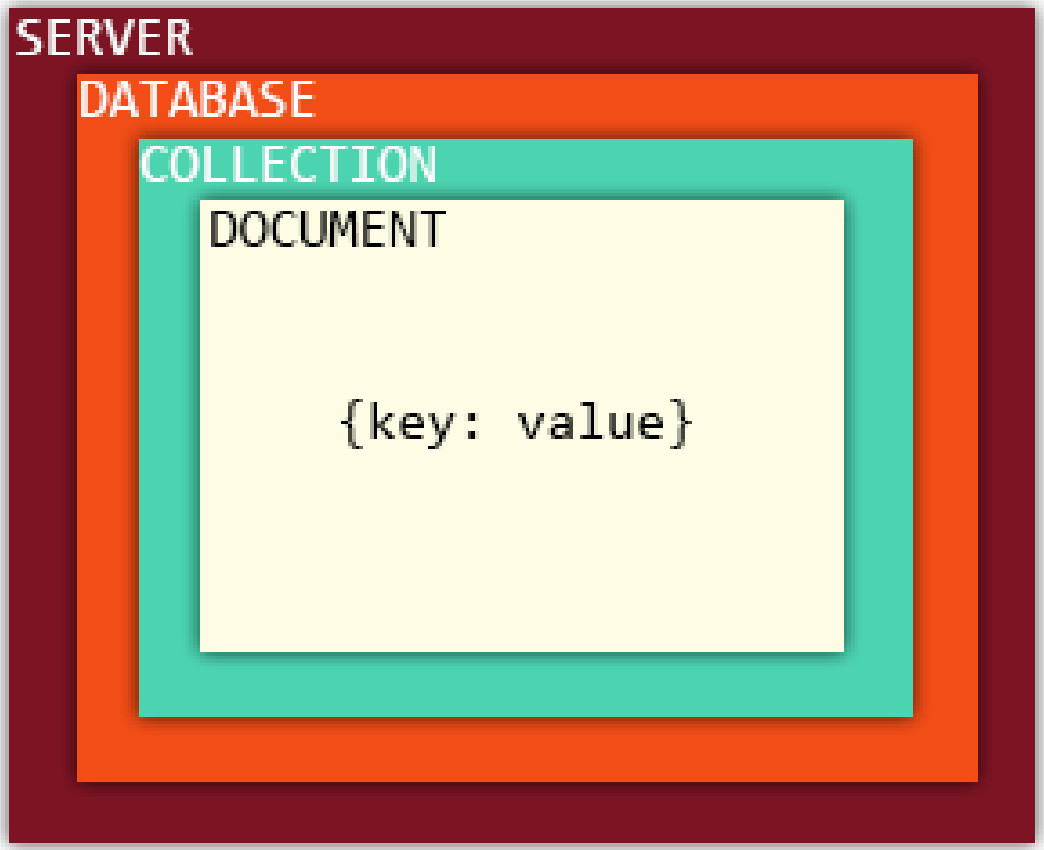
NoSQL

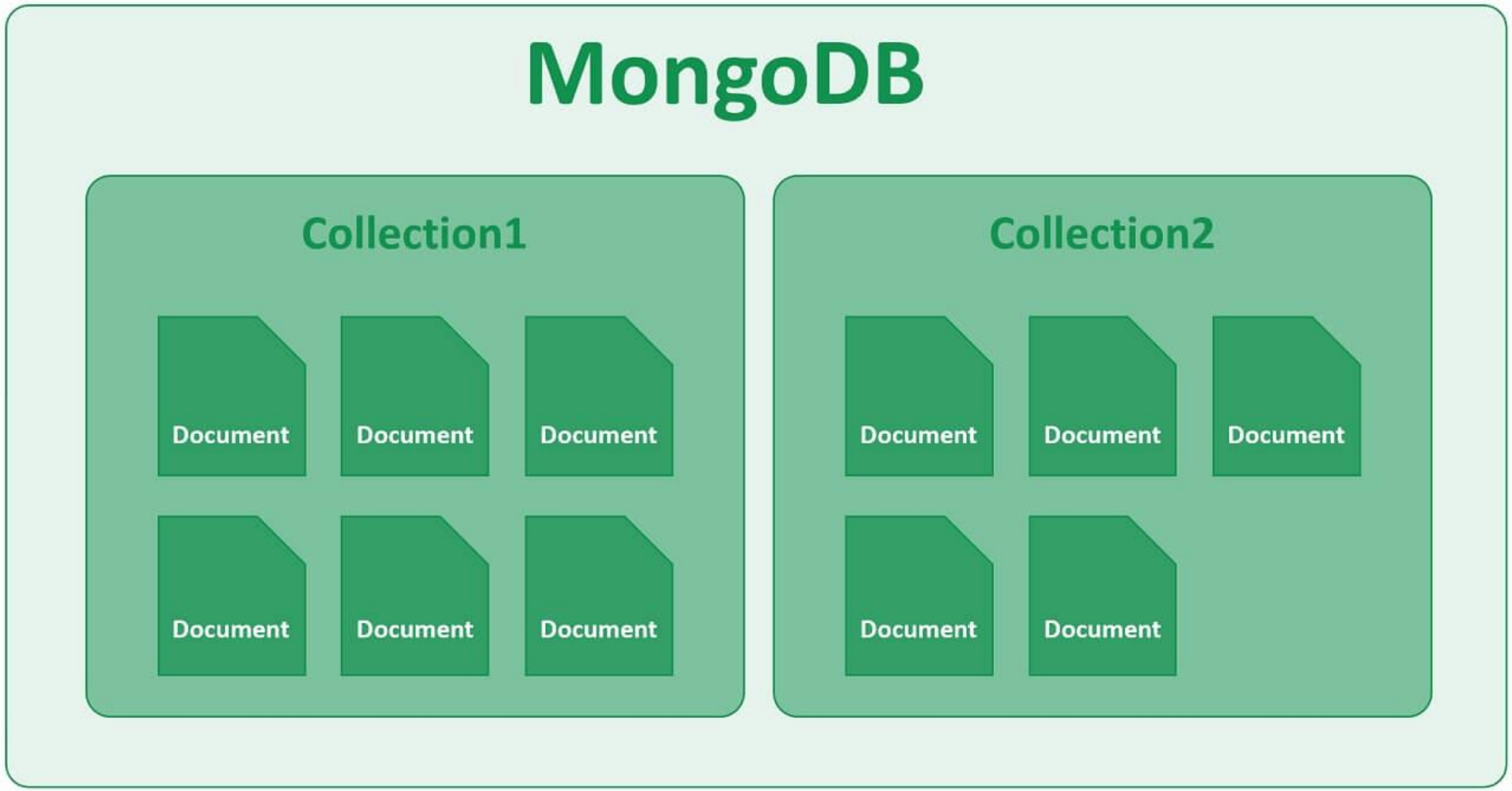
mongoDB®

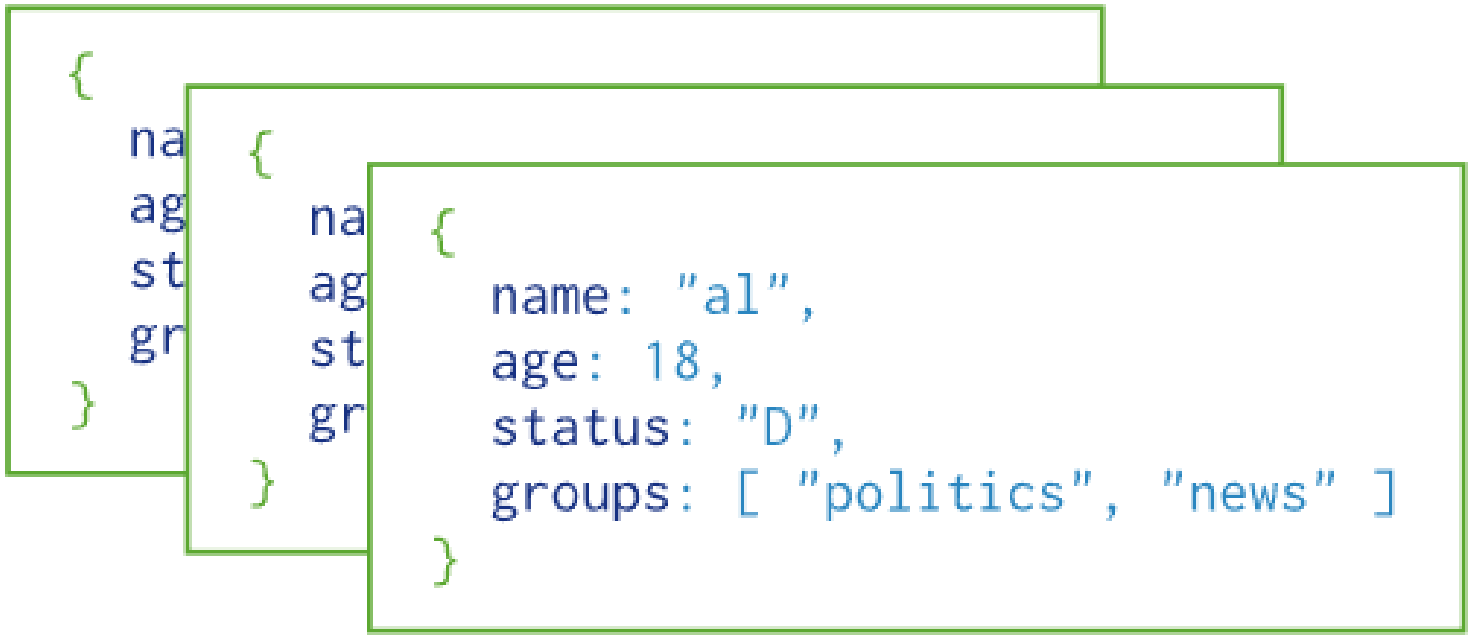
2019년 “최고 인기” 데이터베이스


Other
2.4%
Oracle
1.8%
Cassandra
3.0%
Redis
8.4%
MySQL
38.9%
PostgreSQL
17.4%
MongoDB
24.6%

MongoDB 의 구조


SERVER
DATABASE
COLLECTION
DOCUMENT
{key: value}


MongoDB
Collection1
Document
Document
Document
Document
Document
Document
Collection2
Document
Document
Document
Document
Document



Collection

click to enlarge

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

const uri =
  "mongodb+srv://tetz:qwer1234@cluster0.sdiakr0.mongodb.net/?retryWrites=true&w=majority";

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  console.log(test);
  client.close();
});
```

MongoDB 첫 데이터 베이스 생성하기!

- 이제 슬슬 MongoDB 의 명령어를 외우셔야 합니다!

```
const users = client.db('kdt5').collection('users');
```

- MongoDB 의 구조에 따라 먼저 DB 명을 쓰고, collection 명을 써줍니다
- 이렇게만해도 Schema 역할을 하는 DB 와 table 역할을 하는 collection 생성이 끝났습니다! MySQL 보다 편하죠!?
- RDMBS에서는 먼저 스키마를 통해 DB 이름과, 테이블 이름, 구조를 전부다 만들어 줘야만 뭔가를 시작 할 수 있습니다
- MongoDB 는 그냥 이렇게 간단하게 선언해서 사용할 수 있습니다.

```
client.connect((err, db) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (err) => {
    test.insertOne(
      {
        name: 'tetz',
        nickName: 'chickenHead',
      },
      (err, result) => {
          console.log(result);
      },
    );
  });
});
```

```
{
  acknowledged: true,
  insertedId: new ObjectId("6417508486656b1e86edc660f")
}
```

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (err) => {
    test.insertOne(
      {
        name: "tetz",
        nickName: "chickenHead",
      },
      (err, result) => {
        if (result.acknowledged) {
          const findData = test.find({});
          findData.toArray((err, data) => {
            console.log(data);
          });
        }
      }
    );
  });
});
```


지금 시작합니다

MongoDB

Query 배우기

삽입

insertOne

- 하나의 도큐먼트를 삽입합니다

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertOne(
      {
        name: "pororo",
        age: 5,
      },
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        console.log(insertResult);
      }
    );
  });
});
```

```
{
  acknowledged: true,
  insertedId: new ObjectId("64185fa319e4137e9b9ccc6e")
}
```

```
_id: ObjectId('641860a7ce5bf85071433f3c')
name: "pororo"
age: 5
```

insertMany

- 여러 도큐먼트를 한번에 삽입 합니다
- 삽입할 도큐먼트는 배열에 담긴 객체 형태로 전달 되어야 합니다

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        console.log(insertResult);
      }
    );
  });
});
```

```
{
  acknowledged: true,
  insertedCount: 3,
  insertedIds: {
    '0': new ObjectId("641861305e5008f364de434e"),
    '1': new ObjectId("641861305e5008f364de434f"),
    '2': new ObjectId("641861305e5008f364de4350")
  }
}
```

```
_id: ObjectId('641861305e5008f364de434e')
name: "pororo"
age: 5
```

```
_id: ObjectId('641861305e5008f364de434f')
name: "loopy"
age: 6
```

```
_id: ObjectId('641861305e5008f364de4350')
name: "crong"
age: 4
```

삭제

deleteOne

- 조건을 만족하는 가장 처음의 도큐먼트 하나를 삭제합니다
- 조건은 객체 형태로서 deleteOne 의 첫번째 인자로 전달하면 됩니다!

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.deleteOne({ name: "crong" },
        (deleteOneErr, deleteOneResult) => {
          console.log(deleteOneResult);
        });
      }
    );
  });
});
```

```
[nodemon] starting `node mongo.js`
{ acknowledged: true, deletedCount: 1 }
```

```
_id: ObjectId('641861e28815cb7f3d67f769')
name: "pororo"
age: 5
```

```
_id: ObjectId('641861e28815cb7f3d67f76a')
name: "loopy"
age: 6
```

deleteMany

- 조건을 만족하는 모든 도큐먼트를 삭제 합니다
- 조건은 객체 형태로서 deleteOne 의 첫번째 인자로 전달하면 됩니다!

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.deleteMany(
          { age: { $gte: 5 } },
          (deleteOneErr, deleteOneResult) => {
            console.log(deleteOneResult);
          }
        );
      }
    );
  });
});
```

```
{ acknowledged: true, deletedCount: 2 }
```

```
_id: ObjectId('6418623ed0a4aeec08e85f84')
name: "crong"
age: 4
```

수정

updateOne

- 조건을 만족하는 가정 처음의 도큐먼트 하나를 수정합니다
- 조건은 첫번째 인자로 전달하고, 변경점은 두번째 인자로 전달 합니다~!
- 변경 부분은 $set 을 사용합니다~!

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
         test.updateOne(
          { name: "loopy" },
          { $set: { name: "루피" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});});
```

```
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

```
_id: ObjectId('641863106d515e6c5c088460')
name: "pororo"
age: 5
```

```
_id: ObjectId('641863106d515e6c5c088461')
name: "루피"
age: 6
```

```
_id: ObjectId('641863106d515e6c5c088462')
name: "crong"
age: 4
```

updateMany

- 조건을 만족하는 모든 도큐먼트를 수정합니다
- 조건은 첫번째 인자로 전달하고, 변경점은 두번째 인자로 전달 합니다~!
- 변경 부분은 $set 을 사용합니다~!

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.updateMany(
          { age: { $gte: 5 } },
          { $set: { name: "5살 이상인 친구들" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});
```

```
{
  acknowledged: true,
  modifiedCount: 2,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 2
}
```

```
_id: ObjectId('641863c99b2b873f95d1911e')
name: "5살 이상인 친구들"
age: 5
```

```
_id: ObjectId('641863c99b2b873f95d1911f')
name: "5살 이상인 친구들"
age: 6
```

```
_id: ObjectId('641863c99b2b873f95d19120')
name: "crong"
age: 4
```

검색

findOne

- 검색 조건을 만족하는 최초의 도큐먼트 하나를 찾아 줍니다

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.findOne({ name: "loopy" }, (findErr, findData) => {
          console.log(findData);
        });
      }
    );
  });
});
```

```
{
  _id: new ObjectId("64186cce2f4c66eaea8e7716"),
  name: 'loopy',
  age: 6
}
```

find

- 조건에 맞는 도큐먼트를 전부 찾아 줍니다.
- 단, find 는 독특한 특성을 가집니다
- find 로 찾은 값은 리턴이 되긴 하지만 MongoDB 의 정보를 가진 Cursor 객체로 저장이 되며, 시간이 필요한 작업이 아닙니다!
- 대신 찾은 DB의 정보를 데이터화 할 때에는 시간이 필요합니다! (콜백, 프로미스, Await 사용 필요!)

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        const findCursor = test.find({ name: "loopy" });
        console.log(findCursor);
      }
    );
  });
});
```

```
  raw: false
},
[Symbol(filter)]: { name: 'loopy' },
[Symbol(builtOptions)]: {
  raw: false,
  promoteLongs: true,
  promoteValues: true,
  promoteBuffers: false,
  ignoreUndefined: false,
  bsonRegExp: false,
  serializeFunctions: false,
  fieldsAsRaw: {},
  writeConcern: WriteConcern { w: 'majority' },
  readPreference: ReadPreference {
    mode: 'primary',
    tags: undefined,
```

Find 쿼리는 기존 쿼리와는 달리
콜백을 사용하지 않는
시간이 걸리지 않는 쿼리!

대신 원하는 데이터를 찾아 주는 것이
아니라 해당 데이터가 있는 위치 정보
를 가르키는 cursor 를 리턴

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        const findCursor = test.find({ name: "loopy" });
        console.log(findCursor);
        findCursor.toArray((toArrErr, arrData) => console.log(arrData));

      }
    );
  });
});
```

단 Cursor 에서 데이터를 뽑을 때에는 시간이 필요하여 콜백을 사용!

$set

$set: {}

- MongoDB 의 도큐먼트를 수정할 때 사용합니다.
- 수정 Query 에서 도큐먼트를 수정 할 때 $set: { 수정할 내용 } 으로 수정을 해야 합니다.

```
users.updateOne(
    {
      name: 'loopy',
    },
    {
      $set: {
        name: '루피',
      },
    }
```

조건 → name 프로퍼티가 'loopy'인 도큐먼트 찾기

찾은 도큐먼트의 name 프로퍼티를 해당 값으로 변경!

비교식

쿼리	설명
$eq	일치하는 값을 찾는다.
$gt	지정된 값보다 큰 값을 찾는다.
$gte	크거나 같은 값을 찾는다.
$lt	지정된 값보다 작은 값을 찾는다.
$lte	작거나 같은 값을 찾는다.
$ne	일치하지 않는 모든 값을 찾는다.($eq의 부정)
$in	배열에 지정된 값 중 하나와 일치한 값을 찾는다.
$nin	배열에 지정된 값과 일치하지 않는 값을 찾는다.

```
users.updateMany(
    {
      age: { $gte: 5 },
    },
    {
      $set: {
        name: '5살 이상',
      },
    }
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규
$ node routes/mongo.js
{
  _id: new ObjectId("63177bda8532c56fcbdd55c5"),
  name: '5살 이상',
  age: 5
}
{
  _id: new ObjectId("63177bda8532c56fcbdd55c6"),
  name: '5살 이상',
  age: 6
}
{
  _id: new ObjectId("63177bda8532c56fcbdd55c7"),
  name: 'crong',
  age: 4
}
```

```
users.updateMany(
    {
      name: { $ne: 'loopy' },
    },
    {
      $set: {
        name: '루피 아님',
      },
    },
```

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/4th_backend (main)
$ node controllers/mongoConnect.js
[
  {
    _id: new ObjectId("6388646f2ed89be042075980"),
    name: '루피 아님',
    age: 5
  },
  {
    _id: new ObjectId("6388646f2ed89be042075981"),
    name: 'loopy',
    age: 6
  },
  {
    _id: new ObjectId("6388646f2ed89be042075982"),
    name: '루피 아님',
    age: 4
  }
]
```

논리식

- 여러 조건을 걸 때에는 조건들을 배열에 담긴 객체 형태로 전달 합니다!

```
db.컬렉션명.find({쿼리: [{조건1}, {조건2}, ...]})
```

쿼리	설명
$or	조건들 중 하나라도 true면 반환 (true: 조건과 일치, false: 조건과 불일치)
$and	조건들이 모두 true일 때 반환
$not	조건이 false일 때 반환
$nor	조건들이 모두 false일 때 반환

```
const cusor = users.find({
    $and: [{ age: { $gte: 5 } }, { name: 'loopy' }],
  });

cursor.toArray((err, data) => {
   console.log(data);
});
```

```
$ node routes/mongo.js
{
  _id: new ObjectId("631782529e9a0d941eda7ebd"),
  name: 'loopy',
  age: 6
}
```

실습, 데이터 삽입 – 수정 – 삭제 – 검색 하기

- Kdt5 데이터 베이스, member 컬렉션을 만들어 주세요
- 자신과 같은 줄에 앉은 사람의 이름과 나이 정보를 insertMany 쿼리를 이용하여 컬렉션에 추가해 주세요 (Ex. { name: '이효석', age: 39 })
- 자신의 바로 앞 or 뒤에 앉은 사람의 이름과 나이 정보를 insertOne 쿼리를 이용하여 컬렉션에 추가해 주세요
- 자신의 바로 옆에 앉은 사람의 도큐먼트를 삭제해 주세요
- 자신의 바로 앞에 앉은 사람의 이름과 나이 정보를 옆에 앉은 사람의 정보로 변경해 주세요!

실습, 데이터 삽입 – 수정 – 삭제 – 검색 하기

- Member 컬렉션 중에서 나이가 25살 이상인 사람들을 전부 찾아서 console.log 로 출력해 주세요~!

콜백 지옥을

Async / Await 로!

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.updateMany(
          { age: { $gte: 5 } },
          { $set: { name: "5살 이상인 친구들" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});
```

콜백 지옥의 구원자 Async/Await

- 이러한 콜백 지옥에서 벗어나고자 JS는 ES6 에서 부터 Promise 와 Async/Await 를 지원하게 되었습니다!
- 다만 Promise 의 경우도 Promise Chain 이 발생하기 때문에 이전에 작성하셨던 JS 와는 또 느낌이 다르게 됩니다!

```
fetchUser(id: "123")
  .map      { try JSONDecoder().decode(User.self, from: $0) }
  .map      { try isOverEighteen(user: $0) }
  .flatMap  { saveUserInDb(user: $0) }
  .map      { "👍 Welcome \(user.name)" }
  .catch    { error in "👎 Sorry, something went wrong." }
  .map      { updateUi(text: $0) }
```

콜백 지옥의 구원자 Async/Await

- 그래서 이제 Async / Await 를 쓰시면 됩니다!

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

const uri = '';

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

client.connect((err, db) => {
  console.log(db);
});
```

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

const uri = '';

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

async function main() {
  const db = await client.connect();
  console.log(db);
  client.close();
}

main();
```

삽입

insertOne

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertOne(
      {
        name: "pororo",
        age: 5,
      },
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        console.log(insertResult);
      }
    );
  });
});
```

```
async function main() {
  await client.connect();
  const test = client.db("kdt5").collection("test");

  const deleteResult = await test.deleteMany({});
  if (!deleteResult.acknowledged) return "삭제 에러 발생";

  const insertResult = await test.insertOne({
    name: "pororo",
    age: 5,
  });
  if (!insertResult.acknowledged) return "삽입 에러 발생";

  console.log(insertResult);

  client.close();
}

main();
```

삭제

deleteOne

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
        test.deleteOne({ name: "crong" },
        (deleteOneErr, deleteOneResult) => {
          console.log(deleteOneResult);
        });
      }
    );
  });
});
```

```
async function main() {
  await client.connect();
  const test = client.db("kdt5").collection("test");

  const deleteAllResult = await test.deleteMany({});
  if (!deleteAllResult.acknowledged) return "삭제 에러 발생";

  const insertResult = await test.insertMany([
    { name: "pororo", age: 5 },
    { name: "loopy", age: 6 },
    { name: "crong", age: 4 },
  ]);
  if (!insertResult.acknowledged) return "삽입 에러 발생";

  const deleteOneResult = await test.deleteOne({ name: "crong" });
  if (!deleteOneResult.acknowledged) return "삭제 에러 발생";

  console.log(deleteOneResult);

  client.close();
}

main();
```

수정

updateOne

```
client.connect((err) => {
  const test = client.db("kdt5").collection("test");
  test.deleteMany({}, (deleteErr, deleteResult) => {
    if (deleteErr) throw deleteErr;
    test.insertMany(
      [
        { name: "pororo", age: 5 },
        { name: "loopy", age: 6 },
        { name: "crong", age: 4 },
      ],
      (insertErr, insertResult) => {
        if (insertErr) throw insertErr;
         test.updateOne(
          { name: "loopy" },
          { $set: { name: "루피" } },
          (updateErr, updateResult) => {
            if (updateErr) throw updateErr;
            console.log(updateResult);
          }
        );
      }
    );
  });});
```

```
async function main() {
  await client.connect();
  const test = client.db("kdt5").collection("test");

  const deleteAllResult = await test.deleteMany({});
  if (!deleteAllResult.acknowledged) return "삭제 에러 발생";

  const insertResult = await test.insertMany([
    { name: "pororo", age: 5 },
    { name: "loopy", age: 6 },
    { name: "crong", age: 4 },
  ]);
  if (!insertResult.acknowledged) return "삽입 에러 발생";

  const updateOneResult = await test.updateOne(
    { name: "loopy" },
    { $set: { name: "루피" } }
  );

  if (!updateOneResult.acknowledged) return "수정 에러 발생";
  console.log(updateOneResult);

  client.close();
}

main();
```

실습, Async / Await 로 변경

- 이전 실습 코드를 전부 Async / Await 로 변경해 주세요!

MongoDB로 기능 구현

MySQL로 구현 했던 기능을 MongoDB로

- 이제는 MongoDB를 사용해서 기능을 구현해 봅시다~!

MongoDB 접속 클라이언트를 모듈화!

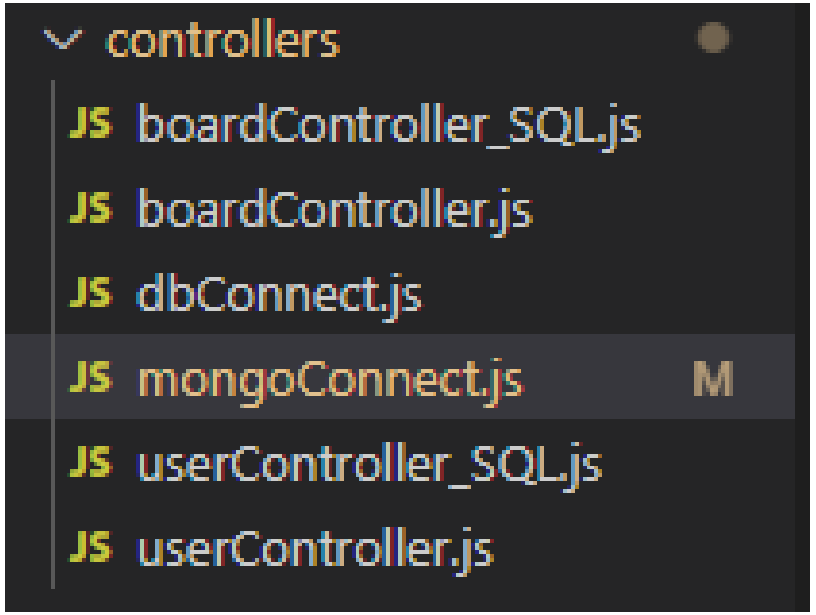
- Controllers 폴더에 mongoConnect.js 파일을 만들고 몽고 디비 접속 클라이언트를 모듈화 시켜 줍시다!

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

const uri =
  'mongodb+srv://xenosign1:qwer1234@cluster0.8sphltr.mongodb.net/?retryWrites=true&w=majority';

const client = new MongoClient(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverApi: ServerApiVersion.v1,
});

module.exports = client;
```

MongoDB 용 컨트롤러 생성

MongoDB 용 컨트롤러 만들기!

- 기존 MySQL로 구현한 컨트롤러는 각각의 파일명 뒤에 SQL 를 붙여서 따로 보관하고 이제 MongoDB 용으로 변경해 봅시다!



MongoDB 용

회원가입 컨트롤러

MongoDB 용 컨트롤러 코드 작성!

- 먼저 회원 가입 및 로그인 기능부터 구현을 해야 하므로 userController.js 부터 작업을 합시다!
- 몽고 디비 접속용 클라이언트 모듈 불러오기

```
const mongoClient = require('./mongoConnect');
```

MongoDB 용 컨트롤러 코드 작성!

- 기존 컨트롤러에서 필요한 기능은 2가지 였습니다!
- 중복 회원 체크 기능 / 회원 가입 기능
- 중복 회원 체크 기능부터 몽고 디비로 변경해 봅시다!

기존 중복 회원 체크 기능!

```
userCheck: (userId, cb) => {
    connection.query(
      `SELECT * FROM mydb1.user WHERE USERID = '${userId}';`,
      (err, data) => {
        if (err) throw err;
        console.log(data);
        cb(data);
      },
    );
  },
```

- 콜백을 Async / Await 으로 변경
- MySQL을 몽고 디비로 변경!

새로운 중복 회원 체크 기능!

```
const mongoClient = require('./mongoConnect');

const userDB = {
  userCheck: async (userId) => {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');
    const findUser = await user.findOne({ id: userId });
    if (!findUser) return false;
    return findUser;
  },
};

module.exports = userDB;
```

기존 회원 가입 기능!

```
registerUser: (newUser, cb) => {
    connection.query(
      `INSERT INTO mydb1.user (USERID, PASSWORD) VALUES ('${newUser.id}', '${newUser.password}');`,
      (err, data) => {
        if (err) throw err;
        console.log(data);
        cb(data);
      },
    );
  },
```

- 콜백을 Async / Await 으로 변경
- MySQL을 몽고 디비로 변경!

새로운 회원 가입 기능!

```
registerUser: async (newUser) => {
    const client = await mongoClient.connect();
    const user = client.db('kdt5').collection('user');

    const insertResult = await user.insertOne(newUser);
    if (!insertResult.acknowledged) throw new Error('회원 등록 실패');
    return true;
  },
```

회원가입

아이디

11

비밀번호

••

회원가입

QUERY RESULTS: 1-1 OF 1

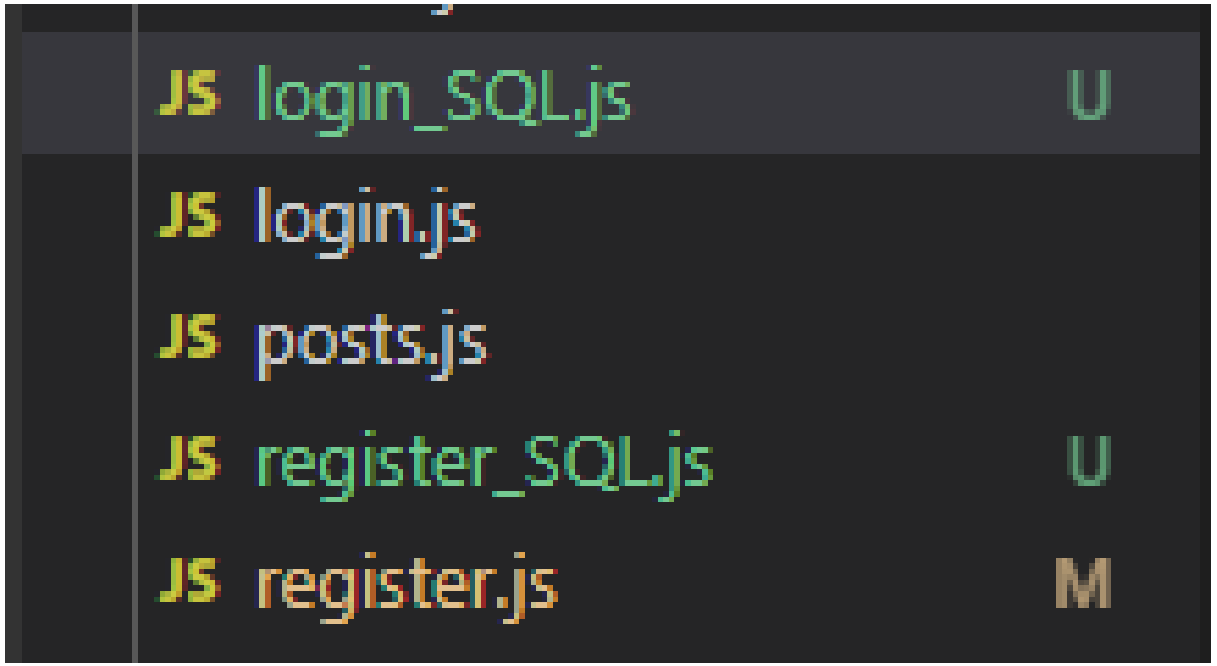
```
_id: ObjectId('638a3f5cb0ff62ab587c85ef')
id: "11"
password: "11"
```

회원 가입 라우터

코드 수정!

MongoDB 용 라우터 만들기!

- 기존 MySQL로 구현한 라우터는 각각의 파일명 뒤에 SQL 를 붙여서 따로 보관하고 이제 MongoDB 용으로 변경해 봅시다!



MongoDB 용 라우터 코드 작성!

- 몽고 디비 용 클라이언트 불러오기!

```
const userDB = require('../controllers/userController');
```

기존 회원 가입 라우터 코드

```
router.post('/', (req, res) => {
  userDB.userCheck(req.body.id, (data) => {
    if (data.length === 0) {
      userDB.registerUser(req.body, (result) => {
        if (result.affectedRows >= 1) {
          res.status(200);
          res.send('회원 가입 성공!<br><a href="/login">로그인으로 이동</a>');
        } else {
          res.status(500);
          res.send(
            '회원 가입 실패! 알 수 없는 문제 발생<br><a href="/register">회원 가입으로 이동</a>',
          );
        }
      });
    } else {
      res.status(400);
      res.send(
        '동일한 ID를 가진 회원이 존재 합니다!<br><a href="/register">회원 가입으로 이동</a>',
      );
    }
  });
});
```

새로운 회원 가입 라우터 코드

```
router.post('/', async (req, res) => {
  const duplicatedUser = await userDB.userCheck(req.body.id);
  if (!duplicatedUser) {
    const registerResult = await userDB.registerUser(req.body);
    if (registerResult) {
      res.send('회원 가입 성공!<br><a href="/login">로그인 페이지로 이동</a>');
    } else {
      res.status(404);
      res.send(
        '회원 가입 문제 발생.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  } else {
    res.send(
      '중복된 id 가 존재합니다.<br><a href="/register">회원가입 페이지로 이동</a>',
    );
  }
});
```

로그인 라우터

코드 수정!

실습, 로그인 라우터 코드 변경하기!

- 기존 MySQL로 구현한 로그인 라우터를 MongoDB 버전으로 변경 하시면 됩니다!
- 어렵겠지만 한번 직접 도전해 보시고 설명을 듣는 편이 훨씬 더 도움이 되실 겁니다!
- 일단 콜백으로 구현 되었던 코드를 Async / Await 으로 구현해 주세요!
- 컨트롤러에서 어떤 값을 return 하는지를 잘 생각하셔서 결과에 대한 처리를 해주세요!

실습, 로그인 라우터 코드 변경하기!

- 컨트롤러 사용 코드를 제외하고는 나머지는 코드는 전부 동일 합니다!

로그인

아이디

11

비밀번호

••

로그인

Tetz Board

현재 등록 글 : 2 글쓰기 로그아웃

작성자 : tetz

제목1

테스트 컨텐츠 입니다!

작성자 : tetz

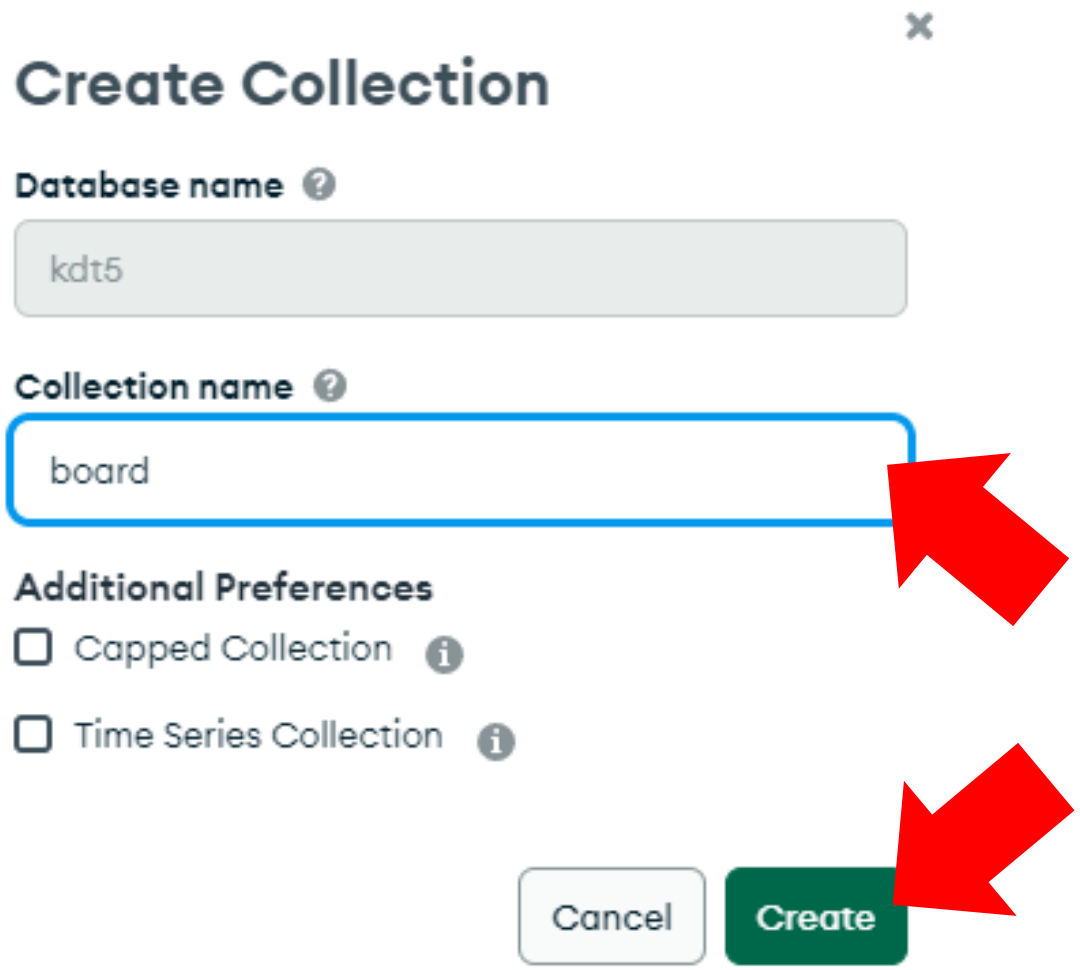
제목2

테스트 컨텐츠 입니다!

게시판도

MongoDB로!

게시판용 콜렉션과 도큐먼트 생성하기!



+ Create Database

Search Namespaces

- kdt-test
- kdt1
- kdt4
- kdt5
  - board
  - test
  - users
- login

kdt5.board

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' }

Apply Reset

QUERY RESULTS: 0

Insert to Collection board

VIEW { } ☰

```
1   _id: 638a4b32545fa5ecb5be50cb                ObjectId
2   USERID: "11  //"                             String
3   TITLE: "제목1//"                             String
4   CONTENT: "내용1//"                           String
```

Cancel Insert

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
USERID: "11"
TITLE: "제목1"
CONTENT: "내용1"
```

게시판용

컨트롤러 만들기!

MongoDB 용 컨트롤러 코드 작성!

- 몽고 디비 접속용 클라이언트 모듈 불러오기

```
const mongoClient = require('./mongoConnect');
```

전체 게시글 가져오기 기능

기존 전체 게시글 가져오기 컨트롤러 코드

```
getAllArticles: (cb) => {
    connection.query('SELECT * FROM mydb1.board;', (err, data) => {
      if (err) throw err;
      cb(data);
    });
  },
```

새로운 전체 게시글 가져오기 컨트롤러 코드

```
getAllArticles: async () => {
    const client = await mongoClient.connect();
    const boardDB = client.db('kdt5').collection('board');

    const allArticlesCursor = boardDB.find({});
    const allArticles = await allArticlesCursor.toArray();
    return allArticles;
},
```

기존 전체 게시글 보여주기 라우터 코드 수정

- 전체 게시글을 MongoDB에서 받아오는 컨트롤러를 만들었으니, 해당 기능을 이용 기존 라우터를 변경해 봅시다!
- 게시판 라우터도 SQL 버전을 하나 만들어서 백업!

기존 게시판 페이지 라우터

```
router.get('/', isLogin, (req, res) => {
  db.getAllArticles((data) => {
    const ARTICLE = data;
    const articleCounts = ARTICLE.length;
    res.render('dbBoard', {
      ARTICLE,
      articleCounts,
      userId: req.session.userId,
    });
  });
});
```

새로운 게시판 페이지 라우터

```
router.get('/', isLogin, async (req, res) => {
  const ARTICLE = await db.getAllArticles();
  const articleCounts = ARTICLE.length;
  res.render('dbBoard', {
    ARTICLE,
    articleCounts,
    userId: req.session.userId,
  });
});
```

Tetz Board

현재 등록 글 : 1 글쓰기 로그아웃

작성자 : 11

제목1

내용1

수정 삭제

글 쓰기 기능!

기존 글 쓰기 컨트롤러 코드

```
connection.query(
      `INSERT INTO mydb1.board (USERID, TITLE, CONTENT) VALUES ('${newArticle.id}',
'${newArticle.title}', '${newArticle.content}')`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      },
    );
```

새로운 글 쓰기 컨트롤러 코드

```
writeArticle: async (newArticle) => {
    const client = await mongoClient.connect();
    const boardDB = client.db('kdt5').collection('board');

    const writeResult = await boardDB.insertOne(newArticle);
    if (!writeResult.acknowledged) throw new Error('게시글 추가 실패');
    return true;
  },
```

기존 글 쓰기 라우터 코드

```
router.post('/write', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      userId: req.session.userId,
      title: req.body.title,
      content: req.body.content,
    };
    boardDB.writeArticle(newArticle, (data) => {
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        err.statusCode = 500;
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    err.statusCode = 400;
    throw err;
  }
});
```

새로운 글 쓰기 라우터 코드

```
router.post('/write', isLogin, async (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      USERID: req.session.userId,
      TITLE: req.body.title,
      CONTENT: req.body.content,
    };

    const writeResult = await boardDB.writeArticle(newArticle);
    if (writeResult) {
      res.redirect('/dbBoard');
    } else {
      const err = new Error('글 쓰기 실패');
      throw err;
    }
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

글 쓰기

제목

테스트

내용

테스트

글 작성하기

Tetz Board

현재 등록 글 : 3 글쓰기 로그아웃

작성자 : 11

제목1

내용1

수정 삭제

작성자 : 11

테스트

테스트

수정 삭제

글 수정하기

코드 수정!

엇? 그런데!?

- 기존 MySQL 에서는 ID_PK 라는 컬럼을 이용해서 게시글을 특정 할 수 있었습니다!
- 그런데 MongoDB에는 ID_PK 값이 없네요!?!?
- 이럴 땐, objectID 인 _id 값을 사용하면 됩니다!

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
USERID: "11"
TITLE: "제목1"
CONTENT: "내용1"
```

수정을 위해 ejs 파일 코드 수정

- 기존의 ID_PK 값을 전달 하던 것을, _id 값을 전달 하도록 수정해 줍시다!

```
<div class="foot">
    <% if (ARTICLE[i].USERID === userId) { %>
    <a class="btn orange" href="dbBoard/modify/<%= ARTICLE[i]._id %>">수정</a>
    <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].ID_PK %>')">삭제</a>
    <% } %>
</div>
```

기존 게시글 찾기 컨트롤러 코드

```
getArticle: (id, cb) => {
    connection.query(
      `SELECT * FROM mydb1.board WHERE ID_PK = ${id};`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      },
    );
  },
```

새로운 게시글 찾기 컨트롤러 코드

- ObjectID 를 사용하려면 mongdb 모듈의 ObjectID 클래스를 가져와야만 합니다!₩

```
_id: ObjectId('638a4b32545fa5ecb5be50cb')
```

- _id 는 단순한 문자열로 보이지만 해당 문자열은 특정 의미를 가지고 있으며 해당 의미는 ObjectID 클래스로 해독이 가능하기 때문이죠!

```
// @ts-check
const { ObjectId } = require('mongodb');
const mongoClient = require('./mongoConnect');
```

MongoDB의 ObjectID

ObjectID는 12byte 크기의 문자와 숫자로 구성된 값입니다. ObjectID()의 값을 반환하면 12byte의 hexadecimal 값으로 결과를 반환합니다. 그리고 이 값들은 각각의 의미를 가지고 있습니다.

- 첫 4byte는 timestamp 값을 의미합니다. 이 값은 Unix시대부터 초단위로 측정된 값을 의미합니다.
- 다음 5byte는 랜덤으로 생성된 값입니다.
- 다음 3byte는 증가하는 count이며, 최초값은 랜덤으로 생성됩니다.

새로운 게시글 찾기 컨트롤러 코드

```
getArticle: async (id) => {
    const client = await mongoClient.connect();
    const boardDB = client.db('kdt5').collection('board');

    const findArticle = await boardDB.findOne({ _id: ObjectId(id) });
    if (!findArticle) return false;
    return findArticle;
  },
```

기존 수정 모드로 이동 라우터 코드

```
router.get('/modify/:id', isLogin, (req, res) => {
  boardDB.getArticle(req.params.id, (data) => {
    if (data.length > 0) {
      res.render('dbBoard_modify', { selectedArticle: data[0] });
    }
  });
});
```

새로운 수정 모드로 이동 라우터 코드

```
router.get('/modify/:id', isLogin, async (req, res) => {
  const findArticle = await db.getArticle(req.params.id);
  if (findArticle) {
    res.render('dbBoard_modify', { selectedArticle: findArticle });
  }
});
```

작성자 : 11

테스트

테스트

수정 삭제

Write Mode

제목

테스트

내용

테스트

글 수정하기

실습, 게시글 수정 기능 완성하기!

- 게시글 수정 모드로 이동 까지는 완성 했습니다!
- 그럼 이제 글 수정하기 버튼을 클릭하면 해당 글이 수정 되도록 코드를 수정 해주시면 됩니다!
- 먼저 글을 수정하는 ejs 파일에 가서 ID_PK 가 아닌 _id 값을 전달 하도록 수정 → 글을 수정해서 DB에 Update 하는 modifyArticle 컨트롤러를 수정 → 수정 요청을 수행하는 라우터 코드 수정하기!
- 역시 어렵겠지만, 어려운 만큼 고민을 하시면서 실력이 상승 하실 겁니다!!

글 삭제하기

코드 수정!

ejs 파일 코드 수정

- 수정 때와 마찬 가지로 기존의 ID_PK 값을 전달 하던 것을, _id 값을 전달 하도록 수정해 줍시다!

```
<div class="foot">
    <% if (ARTICLE[i].USERID === userId) { %>
    <a class="btn orange" href="dbBoard/modify/<%= ARTICLE[i]._id %>">수정</a>
    <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i]._id %>')">삭제</a>
    <% } %>
</div>
```

기존 삭제 컨트롤러 코드

```
deleteArticle: (id, cb) => {
    connection.query(
      `DELETE FROM mydb1.board WHERE ID_PK = ${id};`,
      (err, data) => {
        if (err) throw err;
        cb(data);
      },
    );
  },
```

새로운 삭제 컨트롤러 코드

```
deleteArticle: async (id) => {
    const client = await mongoClient.connect();
    const boardDB = client.db('kdt5').collection('board');

    const deleteResult = await boardDB.deleteOne({ _id: ObjectId(id) });

    if (!deleteResult.acknowledged) throw new Error('삭제 실패');
    return true;
  },
```

기존 삭제 라우터 코드

```
router.delete('/delete/:id', isLogin, (req, res) => {
  db.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.protocol41) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

새로운 삭제 라우터 코드

```
router.delete('/delete/:id', isLogin, async (req, res) => {
  const deleteResult = await boardDB.deleteArticle(req.params.id);
  if (deleteResult) {
    res.send('삭제 완료!');
  } else {
    const err = new Error('글 삭제 실패');
    throw err;
  }
});
```


수고하셨습니다!