

Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with





# 백엔트 쿠키 기능들!



# 쿠키의 최대 나이? 정하기!

- 쿠키의 생존은 expires 로 정해도 되지만 maxAge 로도 설정이 가능합니다!
- maxAge 는 쿠키의 생성 시간을 기준으로 밀리 세컨드 단위로 생존 시간을 결정합니다!

```
router.get('/cookie', (req, res) => {  
  res.cookie('cookie', true, {  
    maxAge: 1000 * 60,  
    httpOnly: false,  
  });  
  res.send('쿠키 굿기 성공!');  
});
```



# 쿠키 삭제하기!

- 쿠키를 삭제하는 방법은 `res.clearCookie('쿠키 이름')` 을 사용하면 됩니다!

```
router.get('/cookie', (req, res) => {  
  res.cookie('cookie', true, {  
    maxAge: 1000 * 60,  
    httpOnly: false,  
  });  
  res.clearCookie('cookie');  
  res.send('쿠키 굿기 성공!');  
});
```



# 쿠키 활용하기



ID

아이디를 입력해주세요.

PW

비밀번호를 입력해주세요.

로그인

☐ 아이디 저장

[아이디 찾기](#)

[비밀번호 찾기](#)

안녕하세요 늑대털쓴양입니다.

팝업 하루동안  
안뜨게 하기 강좌 바로가기

☐ 오늘 하루동안 보지 않기 CLOSE

프로필

양

좌

플래시킹

주인  
자인

<http://blog.naver.com/hyove>



# 쿠키를 활용하는 cookie.ejs 페이지 작성

- 쿠키가 없으면 alert 창을 띄우는 cookie.ejs 페이지를 만들어 봅시다!

```
<body>
  <h1>쿠키 팔아요!</h1>
  <input type="checkbox" name="check" id="check" />
  <label>ALERT 하루동안 보지 않기</label>

  <script>
    if (!document.cookie) alert('쿠키 팔아요!');
  </script>
</body>
```



```
router.get('/cook', (req, res) => {  
  res.cookie('alert', true, {  
    maxAge: 1000 * 5,  
    httpOnly: false,  
  });  
  res.status(200).json('쿠키 발급 성공!');  
});
```

```
checkbox.addEventListener('click', async () => {  
  if (checkbox.checked) {  
    const res = await fetch(`http://localhost:4000/cookie/cook`, {  
      method: 'GET',  
      headers: {  
        'Content-type': 'application/json',  
      },  
    });  
    if (res.status === 200) {  
      const msg = await res.json();  
      console.log(msg);  
    } else {  
      alert('쿠키 발행 실패');  
    }  
  }  
})
```







회원테이블  
만들기!



```
2 • CREATE TABLE user (  
3     `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
4     `USERID` VARCHAR(20) NOT NULL UNIQUE,  
5     `PASSWORD` VARCHAR(20) NOT NULL,  
6     `REGISTER_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP,  
7     `UPDATE_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
8 );
```

```
10 • SELECT * FROM user;
```

```
11 • INSERT INTO user (USERID, PASSWORD) VALUES ('tetz', '11');
```

```
12 • INSERT INTO user (USERID, PASSWORD) VALUES ('pororo', '11');
```

	ID_PK	USERID	PASSWORD	REGISTER_TIME	UPDATE_TIME
▶	1	tetz	11	2022-11-26 09:28:54	2022-11-26 09:28:54
	3	pororo	11	2022-11-26 09:29:54	2022-11-26 09:29:54
●	NULL	NULL	NULL	NULL	NULL



# Session

# 모듈 추가하기



# 세션 모듈 추가하기

- 먼저 express-session 모듈 부터 설치 합시다
  - npm i express-session
- 모듈 추가 및 미들웨어 연결

```
const session = require('express-session');
const app = express();
app.use(
  session({
    secret: 'tetz',
    resave: false,
    saveUninitialized: true,
    cookie: {
      maxAge: 1000 * 60 * 60,
    },
  })
);
```



# resister.js

# 구현하기



# 회원 가입 용 라우터 구현

- register.js 를 만들어서 회원가입 기능을 모듈화

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

const router = express.Router();

router.get('/', (req, res) => {
  res.render('register');
});

module.exports = router;
```

- 서버에 라우터 등록

```
const registerRouter = require('./routes/register');
app.use('/register', registerRouter);
```

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







# 회원 가입 컨트롤러 작성

- 중복이 없다면 이제 회원 가입을 하는 컨트롤러를 만들어 주시면 됩니다!

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



```
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">로그인 페이지로 이동</a>',
          );
        } else {
          res.status(500);
          res.send(
            '회원 가입 문제 발생.<br><a href="/register">회원가입 페이지로 이동</a>',
          );
        }
      });
    } else {
      res.status(400);
      res.send(
        '중복된 id 가 존재합니다.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  });
});
```



# login.js

# 구현하기



# 로그인 용 라우터 구현

- login.js 를 만들어서 로그인 기능을 모듈화

```
const express = require('express');
const router = express.Router();

router.get('/', async (req, res) => {
  res.render('login');
});

module.exports = router;
```

- 서버에 라우터 등록

```
const loginRouter = require('./routes/login');
app.use('/login', loginRouter);
```



# 로그인 구현

```
router.post('/', (req, res) => {
  userDB.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;
        res.status(200);
        res.redirect('/dbBoard');
      } else {
        res.status(400);
        res.send(
          '비밀번호가 다릅니다.<br><a href="/login">로그인 페이지로 이동</a>',
        );
      }
    } else {
      res.status(400);
      res.send(
        '해당 id 가 존재하지 않습니다.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  });
});
```



# 로그 아웃 구현



# 로그아웃 버튼 만들기

- dbBoard.ejs 파일에 로그아웃 버튼 추가
- GET 방식 /login/logout 주소로 로그아웃 요청

```
<div class="board_write">  
  <span>현재 등록 글 : &nbsp; <%= articleCounts %></span>  
  <a class="btn red" href="/board/write">글쓰기</a>  
  <a class="btn orange" href="/login/logout">로그아웃</a>  
</div>
```





# 로그 아웃 처리

- 로그 아웃 요청이 들어오면 생성 된 req.session 을 삭제 처리 → 최초 화면으로 이동

```
router.get('/logout', async (req, res) => {  
  req.session.destroy((err) => {  
    if (err) throw err;  
    res.redirect('/');  
  });  
});
```



# 로그인 여부에 따른 게시판 서비스 변경

```
router.get('/', (req, res) => {
  if (req.session.login) {
    db.getAllArticles((data) => {
      const ARTICLE = data;
      const articleCounts = ARTICLE.length;
      res.render('dbBoard', {
        ARTICLE,
        articleCounts,
        userId: req.session.userId,
      });
    });
  } else {
    res.status(404);
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
});
```





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

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



# 게시글에 작성자 id 정보 추가



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ID_PK	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
TITLE	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
CONTENT	VARCHAR(300)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
REGISTER_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP
UPDATE_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP ON...
USERID	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



<						
Result Grid						
Filter Rows:						
Edit:						
Export/Import:						
	ID_PK	TITLE	CONTENT	REGISTER_DATE	UPDATE_DATE	USERID
	1	제목1	테스트 컨텐츠 입니다!	2022-11-24 14:27:39	2022-11-26 11:25:24	tetz
	2	제목2	테스트 컨텐츠 입니다!	2022-11-24 14:27:41	2022-11-26 11:25:24	tetz
▶*	NULL	NULL	NULL	NULL	NULL	NULL





# Board.ejs

## 파일 수정



# 작성자 표시

- 제목 위에 작성자의 id 를 보여주는 부분 추가

```
<li>
  <div class="author">
    작성자 : <%= ARTICLE[i].USERID %>
  </div>
  <div class="title">
    <%= ARTICLE[i].TITLE %>
  </div>
```





# 자신이 작성한 글에만 수정 삭제 버튼 표시

- 게시글의 작성자 id 와 로그인한 유저의 id 를 비교해서 수정 및 삭제 버튼 표시

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



**지금 시작합니다**



# 게시글 추가 기능

## 수정



# 게시글 추가 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/write', isLogin, (req, res) => {  
  res.render('board_write');  
});
```



# 게시글 추가 기능

- 글을 추가 할 때에도 로그인 여부 판별
- 새로운 게시글을 추가할 때, title, content 이외에 id 값으로 로그인한 유저의 id 값을 받아서 글을 추가



```
router.post('/write', isLoggedIn, (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] });  
    }  
  });  
});
```



# 게시글 수정 기능

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



```
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

- 로그인 안되어 있으면 게시물 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



# 쿠키를 사용한 자동 로그인

- 실습에서 하루동안 팝업을 뜨지 않게 한 것처럼, 로그 아웃 후 60초 동안은 세션이 없어도 자동 로그인이 되도록 구현해 봅시다
- 왜 쿠키를 쓸까요?
- 세션은 브라우저를 종료하면 사라지지만 쿠키는 만료일 까지 남아 있게 됩니다! 따라서, 쿠키를 이용해서 구현을 합니다.



# 쿠키를 사용한 자동 로그인

- 먼저 로그인을 하면 쿠키를 발행
  - 사용자 id 정보
  - 10초 의 expires 설정
  - httpOnly 옵션 켜기
  - Signed 옵션 켜기(사용자 ID 가 저장 되므로) → 서버의 Cookie-parser 에 암호화 키 설정 필요

```
app.use(cookieParser('tetz'));
```





```
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>');  
  }  
};
```



# 쿠키가 정상 작동 하는지 테스트!

- 로그인 후, 브라우저를 종료 하고 다시 켜 다음 게시판 서비스로 접근하기!
- 특정 시간 이후 접근이 안되는지 확인!



# 로그아웃을 하면 쿠키도...

- 로그아웃은 사용자가 로그아웃 하겠다는 의사를 밝힌 것이므로 쿠키도 같이 삭제가 되어야 합니다!
- 로그아웃을 한 다음에 다른 사용자가 와서 접근했을 때 로그인 처리가 되면 안되니까요!

```
// 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 = d1rladk  
DB_DATABASE = mydb
```

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

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





# DOTENV, 중요 정보를 관리하는 모듈

- 정말 중요한 정보만 저장이 되는 파일이므로 github 에 올리면 안되겠죠?
- .gitignore 에 추가해 줍니다!

```
node_modules/  
.env
```

- 따라서 해당 파일은 직접 업로드 하면서 사용하시면 됩니다!



# .gitignore 가 안되나요?

- Git 도 캐시를 사용하기 때문에 로컬에 해당 내용이 남아 있어서 그렇게 됩니다 → 깃 캐시를 삭제하고, 다시 푸쉬해 주시면 됩니다!
- `git rm -r --cached .`
- `git add .`
- `git commit -m "clear git cache"`
- `git push --all`



# 구조분해 할당 문법

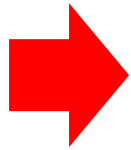
## Destructuring

## 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

# 배열 구조 분해 할당



```
const today = new Date();  
console.log(today); // 2023-03-19T15:57:58.774Z  
  
const formattedDate = today.toISOString().substring(0, 10);  
console.log(formattedDate); // "2023-03-19"  
  
const [year, month, day] = formattedDate.split("-");  
console.log(year, month, day); // 2023, 03, 19
```



# 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
```

# 객체 구조 분해 할당



```
const person = {  
  name: "Lee",  
  address: {  
    zipCode: "03068",  
    city: "Seoul",  
  },  
};  
  
const {  
  address: { city, zipCode },  
} = person;  
  
console.log(city); // 'Seoul'  
console.log(zipCode); // 03068
```

- 객체의 키를 기준으로 변수에 할당
- 순서는 상관이 없고, 객체의 키와 이름이 같아야만 할당이 된다~!



# 전개 구문

# Spread Syntax(...)





# 전개 구문

- 2015년에 추가 된 문법입니다!
- 백 엔드에서 큰 데이터를 다룰 때 자주 사용하므로 간단하게 배우고 넘어갈게요!
- 병합, 구조 분해 할당 등에 다양하게 사용이 가능합니다.
- 말그대로 배열 또는 객체의 값을 하나하나 따로 분리해서 흩뿌리는 역할을 해줍니다.
- 원하는 변수 앞에 ... 을 써주면 됩니다

```
const arr = [1, 2, 3, 4, 5, 6];  
  
console.log(arr);  
console.log(...arr);
```

```
tetz@DESKTOP-P7Q4OLL MINGW64  
$ node test.js  
[ 1, 2, 3, 4, 5, 6 ]  
1 2 3 4 5 6
```



# 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-P7Q40LL MINGW64 ~/Desktop/KDT/__수업 자료/정규 수업/29/backend  
$ node test.js  
{  
  tetzData: { name: '이효석', gender: 'M' },  
  tetzInfo: { nickName: 'gotetz', email: 'xenosign@gmail.com' }  
}
```



# 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-P7Q40LL 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'  
]
```



# 전개 구문 - 나머지 연산자, 객체

```
const tetzData = {  
  name: '이효석',  
  gender: 'M',  
  nickName: 'gotetz',  
  email: 'xenosign@gmail.com',  
};  
  
const { name, ...tetzInfo } = tetzData;  
console.log(name, tetzInfo);
```

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



# 전개 구문 - 나머지 연산자, 배열

```
const [first, ...rest] = [1, 2, 3, 4, 5];  
  
console.log(first);  
console.log(rest);
```

```
tetz@DESKTOP-P7Q40LL  
$ node test.js  
1  
[ 2, 3, 4, 5 ]
```

# 전개 구문 - 매개변수



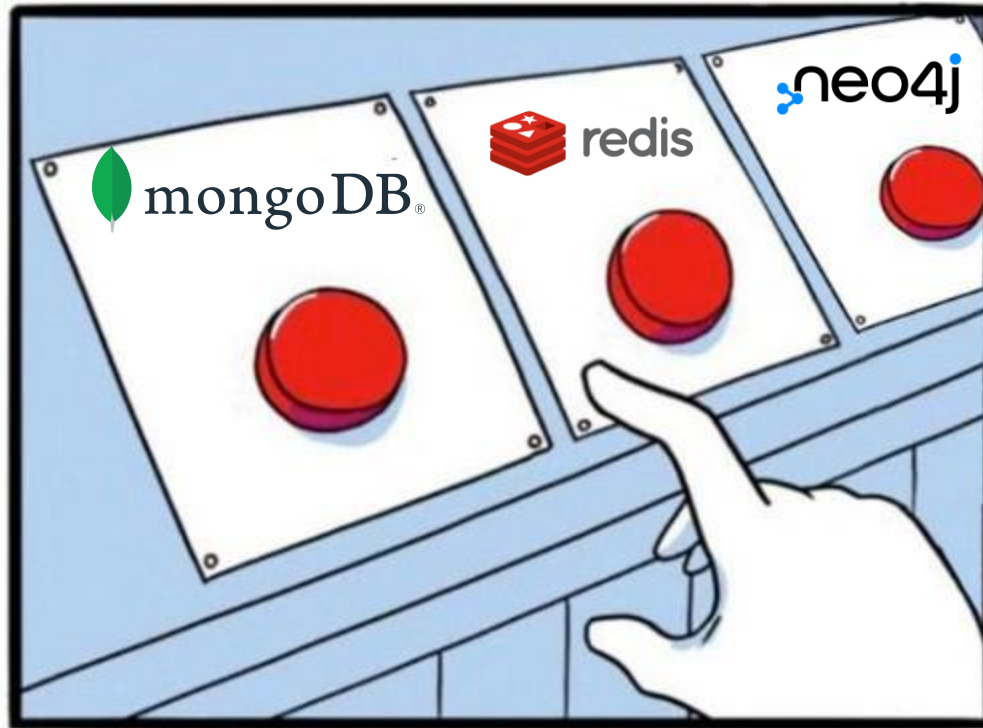
```
function spread(fisrt, ...rest) {  
  console.log(fisrt);  
  console.log(rest);  
}
```

```
spread(1, 2, 3, 4, 5, 6);
```

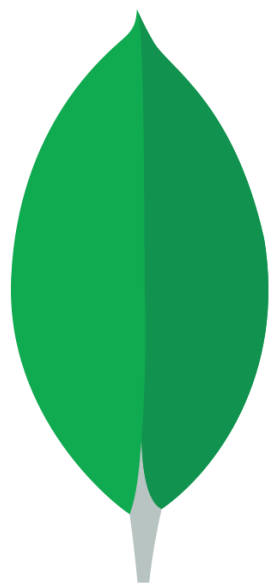
```
tetz@DESKTOP-P7Q40LL  
$ node test.js  
1  
[ 2, 3, 4, 5, 6 ]
```



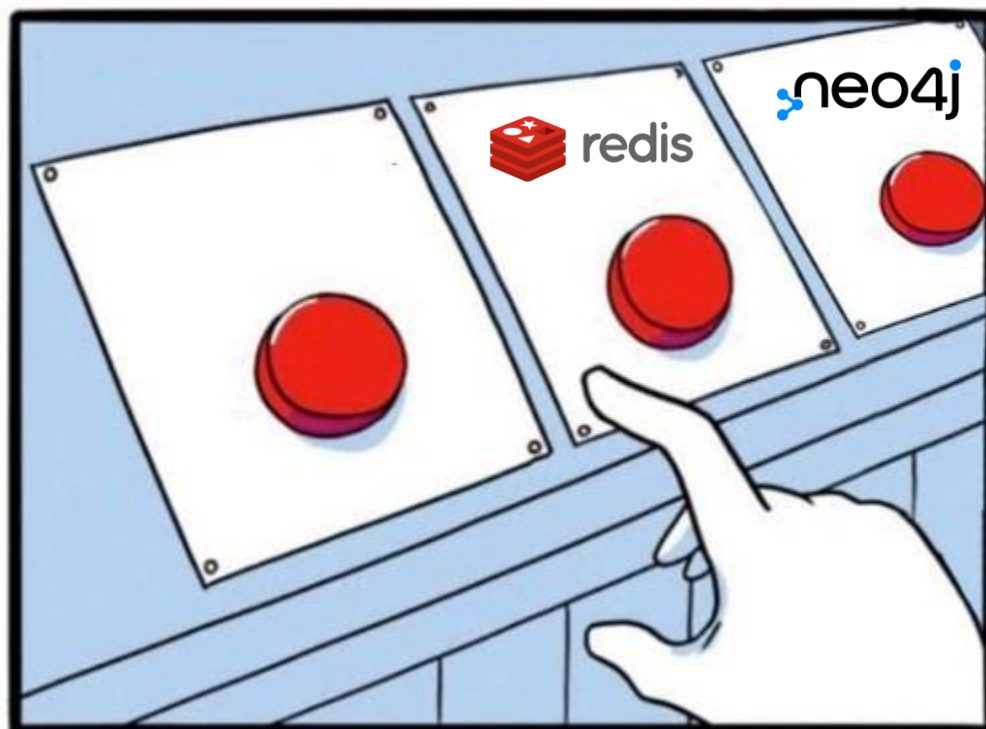
# NoSQL

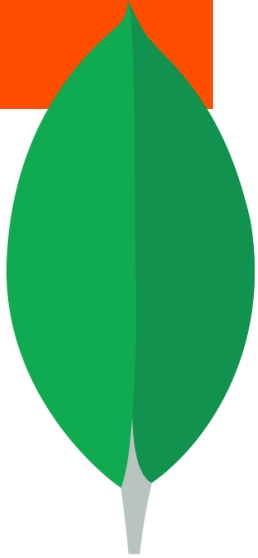






# mongoDB®





mongoDB®



user document

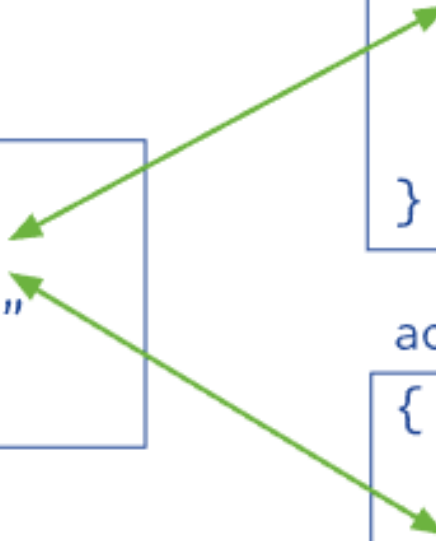
```
{  
  _id: <ObjectId1>,  
  username: "123xyz"  
}
```

contact document

```
{  
  _id: <ObjectId2>,  
  user_id: <ObjectId1>,  
  phone: "123-456-7890",  
  email: "xyz@example.com"  
}
```

access document

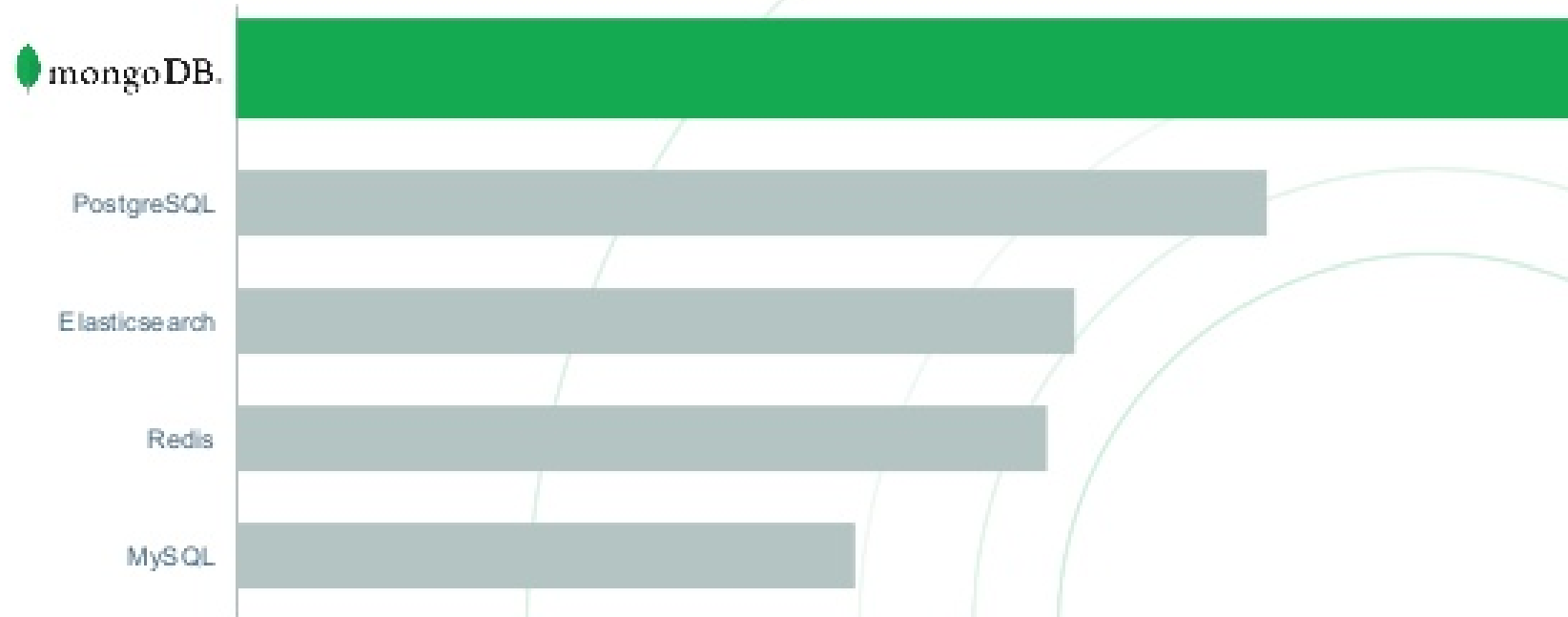
```
{  
  _id: <ObjectId3>,  
  user_id: <ObjectId1>,  
  level: 5,  
  group: "dev"  
}
```

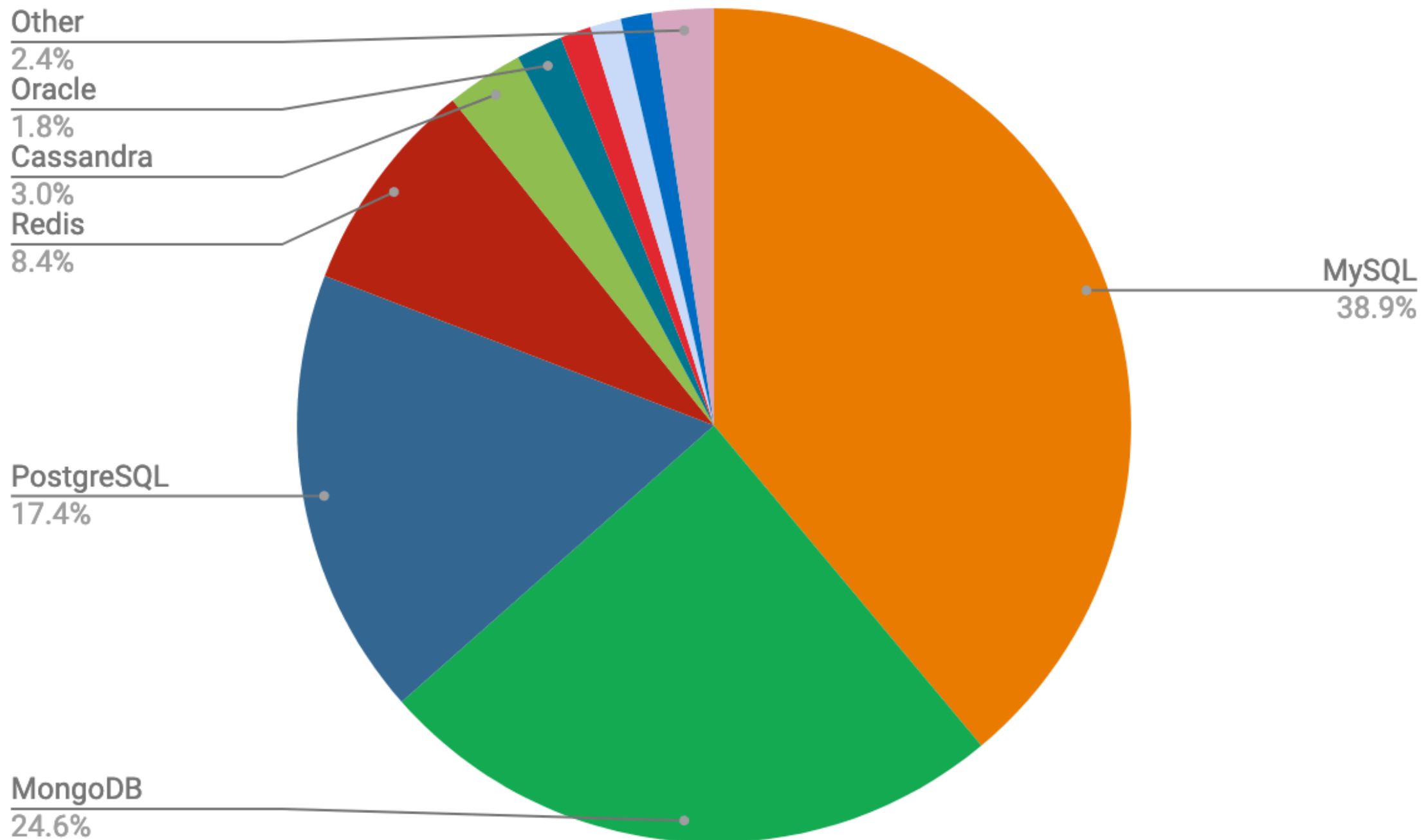






# 2019년 “최고 인기” 데이터베이스







# MongoDB(Humongous DB)

- 장점

- Data 를 익숙한 JSON 형태로 처리 → 빠르게 JSON 전환 가능
- DB 구조의 변경이 용이
- 제약이 없음 → 높은 수평 확장성, 스키마 설계의 유연성

- 단점

- 표준이 없어요(= 제약이 없음)!
- 데이터가 구조화 되어 있지 않음
  - 단순한 구조의 쿼리만 사용 가능
  - 데이터의 일관성 및 안정성을 DB가 아닌 APP 레벨에서 관리해줘야 함 → 버그 발생 확률 높음



<https://www.mongodb.com/ko-kr/cloud/atlas/efficiency>









### High Availability

Each self-healing Atlas cluster is distributed by design across the availability zones within your cloud region.



### Built-in Security

MongoDB Atlas comes with preconfigured security features for authentication, authorization, encryption, and more.



### Automated Backups

MongoDB Atlas offers fully managed backup options, including incremental data recovery and cloud provider snapshots.

시작하기



☐ I agree to the **Terms of Service** and **Privacy Policy**.

Create your Atlas account

or

 Sign up with Google



## Accept Privacy Policy & Terms of Service

Please acknowledge the following terms and conditions to finish creating your account.

☒ I accept the [Privacy Policy](#) and the [Terms of Service](#)

Cancel Signup

Submit





## What is your goal today?

Your answer will help us guide you to successfully getting started with MongoDB Atlas.

- ☒ Learn MongoDB
- ☐ Build a new application
- ☐ Explore what I can build
- ☐ Migrate an existing application

---

## What type of application are you building?

Application Modernization ▼

---

## What is your preferred language?

We'll use this to customize code samples and content we share with you. You can always change this later.

JS JavaScript ▼

Finish



MONGODB ATLAS

# Deploy a cloud database

Experience the best of MongoDB on AWS, Azure, and Google Cloud. Choose a deployment option to get started.

NEW

## Serverless

For application development and testing, or workloads with variable traffic. Minimal configuration required.

- ✓ Pay only for the operations you run
- ✓ Resources scale seamlessly to meet your workload
- ✓ Always-on security and backups

Create

Starting at  
**\$0.10/1M reads**

ADVANCED

## Dedicated

For production applications with sophisticated workload requirements. Advanced configuration controls.

- ✓ Network isolation and fine-grained access controls
- ✓ On-demand performance advice
- ✓ Multi-region and multi-cloud options available

Create

Starting at  
**\$0.08/hr\***  
\*estimated cost \$56.94/month

FREE

## Shared

For learning and exploring MongoDB in a cloud environment. Basic configuration options.

- ✓ No credit card required to start
- ✓ Explore with sample datasets
- ✓ Upgrade to dedicated clusters for full functionality

Create

Starting at  
**FREE**



[I'll do this later](#)

[Advanced Configuration Options](#)



## Cloud Provider & Region

AWS, Seoul (ap-northeast-2) ▾




aws

Google Cloud



Azure

★ Recommended region ⓘ  Dedicated tier region ⓘ

### NORTH AMERICA

 **Oregon** (us-west-2) ★

 **N. Virginia** (us-east-1) ★

 **Ohio** (us-east-2) ★ 


 **N. California** (us-west-1) 

 **Montreal** (ca-central-1) ★ 

### SOUTH AMERICA


 **Sao Paulo** (sa-east-1) ★

### EUROPE

 **Paris** (eu-west-3) ★

 **Frankfurt** (eu-central-1)


 **Stockholm** (eu-north-1) ★

 **Ireland** (eu-west-1) ★

 **London** (eu-west-2) ★ 

 **Milan** (eu-south-1) ★ 

### MIDDLE EAST

 **Bahrain** (me-south-1) ★


### AFRICA

 **Cape Town** (af-south-1) ★


### AUSTRALIA


 **Sydney** (ap-southeast-2) ★


### ASIA

 **Singapore** (ap-southeast-1) ★

 **Hong Kong** (ap-east-1) ★

 **Tokyo** (ap-northeast-1) ★

 **Mumbai** (ap-south-1) ★

 **Seoul** (ap-northeast-2) ★

 **Jakarta** (ap-southeast-3) ★ 

 **Osaka** (ap-northeast-3) ★ 





**Cluster Tier**

M0 Sandbox (Shared RAM, 512 MB Storage)  
Encrypted



**Additional Settings**

MongoDB 5.0, No Backup



**Cluster Name**

Cluster0



**FREE**

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

[Back](#)

**Create Cluster**






Create a database user using a username and password. Users will be given the *read and write to any database* [privilege](#) by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

**Username**

**Password** 

 Autogenerate Secure Password


 Copy

Create User




✓ Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.



### My Local Environment

Use this to add network IP addresses to the IP Access List. This can be modified at any time.



### Cloud Environment

Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

ADVANCED

Add entries to your IP Access List

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#).

IP Address

Description

Enter IP Address

Enter description

Add Entry

Add My Current IP Address

IP Access List

Description

115.136.110.37/32

My IP Address

 REMOVE



## Congratulations on setting up access rules!

You will now be able to connect to your deployments. You can continue to add and update access rules in [Database Access](#) and [Network Access](#).

- ☒ Hide Quickstart guide in the navigation. You can visit [Project Settings](#) to access it in the future.

Go to Databases





효석's Org - ...



Access Manager ▾

Billing



Project 0 ▾



Atlas

App Services

Charts

DEPLOYMENT

Database

Data Lake

PREVIEW

DATA SERVICES

Triggers

Data API

Data Federation

Atlas Search

SECURITY

Database Access

Network Access

Advanced

New On Atlas

4

효석'S ORG - 2022-09-06 > PROJECT 0

## Database Deployments



Find a database deployment...

Cluster0

Connect

View Monitoring

Browse Collections



VERSION

REGION

CLUSTER TIER

TYPE

5.0.11

AWS / Seoul (ap-northeast-2)

M0 Sandbox (General)

Replica Set - 3 nodes



## Connect to Cluster0

✓ Setup connection security

Choose a connection method

Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.



**Connect with the MongoDB Shell**

Interact with your cluster using MongoDB's interactive Javascript interface



**Connect your application**

Connect your application to your cluster using MongoDB's native drivers



**Connect using MongoDB Compass**

Explore, modify, and visualize your data with MongoDB's GUI



**Connect using VS Code**

Connect to a MongoDB host in Visual Studio Code



Go Back

Close

# Connect to Cluster0



✓ Setup connection security

✓ Choose a connection method

Connect

## 1 Select your driver and version

DRIVER

Node.js

VERSION

4.1 or later

## 2 Add your connection string into your application code

☒ Include full driver code example

```
const { MongoClient, ServerApiVersion } = require('mongodb');
const uri = "mongodb+srv://tetz:<password>@cluster0.sdiakr0.mongodb.net/?
retryWrites=true&w=majority";
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology:
true, serverApi: ServerApiVersion.v1 });
client.connect(err => {
  const collection = client.db("test").collection("devices");
  // perform actions on the collection object
  client.close();
});
```

Replace **<password>** with the password for the **tetz** user. Ensure any option params are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

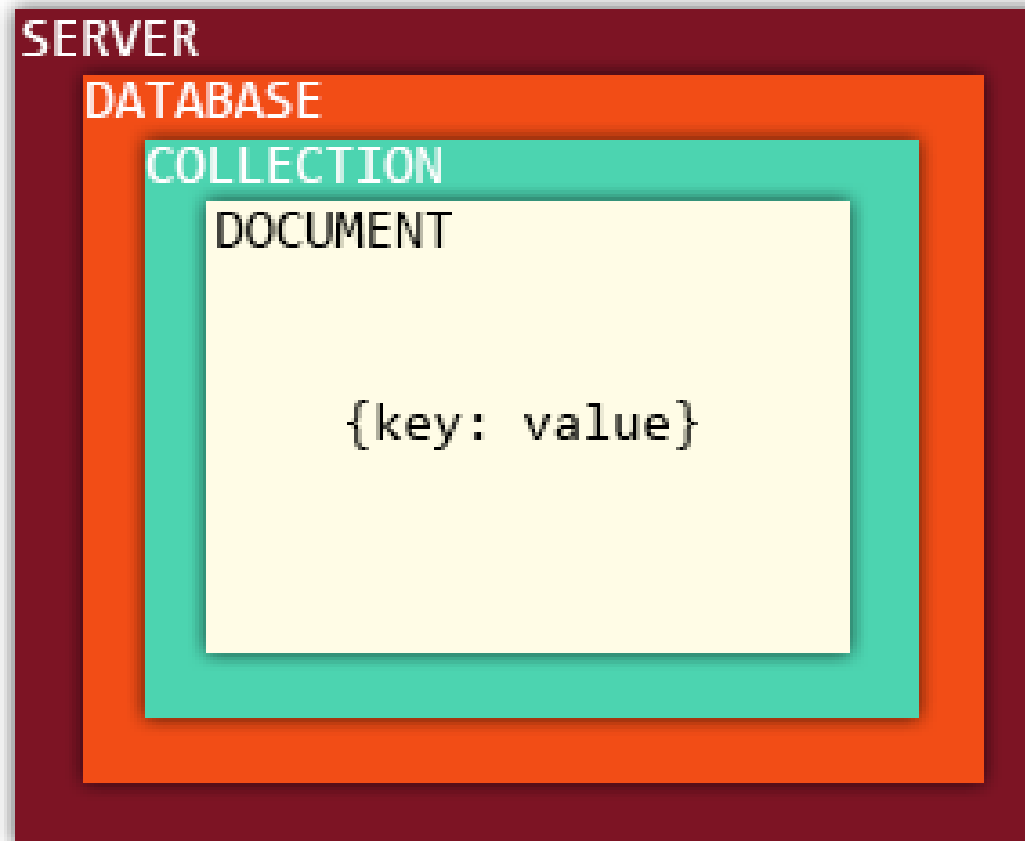
Go Back

Close



# MongoDB 의 구조







# MongoDB

## Collection1

Document

Document

Document

Document

Document

Document

## Collection2

Document

Document

Document

Document

Document



```
{
```

```
  name: "a1",  
  age: 18,  
  status: "D",  
  groups: [ "politics", "news" ]
```

```
}
```

```
{
```

```
  name: "a1",  
  age: 18,  
  status: "D",  
  groups: [ "politics", "news" ]
```

```
}
```

```
{
```

```
  name: "a1",  
  age: 18,  
  status: "D",  
  groups: [ "politics", "news" ]
```

```
}
```

## Collection

[click to enlarge](#)



# MongoDB

# 패키지 설치



# MongoDB 용 js 파일 생성

- mongo.js 파일 생성하기
- 복사한 코드 붙여 넣기!
- 비밀번호는 직접 입력 하셔야 합니다!

```
const { MongoClient, ServerApiVersion } = require('mongodb');  
const uri =  
  'mongodb+srv://tetz:qwer1234@cluster0.sdiakr0.mongodb.net/?retryWrites=  
true&w=majority';
```



# MongoDB 접속용 함수를 생성

- MongoDB 가 5.0 이상 버전 부터는 콜백을 지원 안해주네요.
- 전부 Async / Await 를 쓰도록 변경 되었습니다!
- 하지만 우리는 콜백도 연습하고, 콜백을 Async / Await 변경 하면서 연습을 해나갈 예정입니다!
- 따라서 4.0.0 버전을 설치 해서 콜백으로 먼저 연습하고, 콜백 코드를 Async / Await 로 변경해 봅시다!



# MongoDB 패키지 설치

- Npm i mongodb@4.0.0 -S

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/kdt-5th (main)
$ npm i mongodb@4.0.0 -S

added 18 packages, and audited 35 packages in 9s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```



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





```
[nodemon] starting `node BE/mongo.js`
```

```
Collection {  
  s: {  
    db: Db { s: [Object] },  
    options: {  
      raw: false,  
      promoteLongs: true,  
      promoteValues: true,  
      promoteBuffers: false,  
      ignoreUndefined: false,  
      bsonRegExp: false,  
      serializeFunctions: false,  
      fieldsAsRaw: {},  
      writeConcern: [WriteConcern],  
      readPreference: [ReadPreference]  
    },  
    namespace: MongoDBNamespace { db: 'kdt5', collection: 'test' },  
    pkFactory: { createPk: [Function: createPk] },  
  },  
}
```



# 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.



# MongoDB 첫 데이터 베이스 생성하기!

- 사실 하다보면 이걸 뭐 거의 JS 의 변수를 엮어서 저장해 주는 기능이 아닌가 싶을 정도로 JS의 객체, JSON 과 유사합니다!
- 그래서 편리하죠! ;)



# 첫 데이터 베이스에 데이터 추가하기!

- `Users.insertMany([{}])` 는 많은 데이터(Document)를 한꺼번에 삽입
- 그리고 해당 쿼리의 성공 여부는 리턴 된 객체의 `acknowledged` 값을 통해 확인 할 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("641750848656b1e86edc660f")
}
```



# 첫 데이터 베이스에 데이터 확인하기!

- `Users.find([조건들])` 는 조건들에 부합하는 데이터(Document)를 가져다 줍니다.
- 단, 바로 데이터로 저장해 주는 것이 아니라 해당 데이터의 위치를 알려주는 `cursor` 형태로 가져 옵니다!

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





```
[nodemon] starting `node BE/mongo.js`  
[  
  {  
    _id: new ObjectId("6417512e31da1d77be24c229"),  
    name: 'tetz',  
    nickName: 'chickenHead'  
  }  
]  
□
```

#### QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('638862525ab02be72d628047')  
name: "tetz"  
nickName: "chickenHead"
```





# MongoDB

# Query 배우기



삼입



# insertOne

- 하나의 도큐먼트를 삽입합니다

```
users.insertOne(  
  {  
    name: 'pororo',  
    age: 5,  
  },  
)
```

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/4th_backend (main)  
$ node controllers/mongoConnect.js  
{  
  _id: new ObjectId("638863399a09a39e76ebae31"),  
  name: 'pororo',  
  age: 5  
}
```



# insertMany

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

```
users.insertMany(  
  [  
    {  
      name: 'pororo',  
      age: 5,  
    },  
    {  
      name: 'loopy',  
      age: 6,  
    },  
    {  
      name: 'crong',  
      age: 4,  
    },  
  ],  
)
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규  
$ node routes/mongo.js  
{  
  _id: new ObjectId("6317765527f861681cf45f79"),  
  name: 'pororo',  
  age: 5  
}  
{  
  _id: new ObjectId("6317765527f861681cf45f7a"),  
  name: 'loopy',  
  age: 6  
}  
{  
  _id: new ObjectId("6317765527f861681cf45f7b"),  
  name: 'crong',  
  age: 4  
}
```



# 삭제



# deleteOne

- 조건을 만족하는 가장 처음의 documento 하나를 삭제합니다

```
users.deleteOne(  
  {  
    name: 'crong',  
  },  
)
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규  
$ node routes/mongo.js  
{  
  _id: new ObjectId("6317765527f861681cf45f79"),  
  name: 'pororo',  
  age: 5  
}  
{  
  _id: new ObjectId("6317765527f861681cf45f7a"),  
  name: 'loopy',  
  age: 6  
}
```



# deleteMany

- 조건을 만족하는 모든 도큐먼트를 삭제 합니다

```
users.deleteMany(  
  {  
    age: { $gte: 5 },  
  },  
)
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정  
$ node routes/mongo.js  
{  
  _id: new ObjectId("6317791f9825e69f79e4b160"),  
  name: 'crong',  
  age: 4  
}
```



수정





# updateOne

- 조건을 만족하는 가장 처음의 도큐먼트 하나를 수정합니다

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

```
$ node routes/mongo.js  
{  
  _id: new ObjectId("63177b7fe26abfdaf397586"),  
  name: 'pororo',  
  age: 5  
}  
{ _id: new ObjectId("63177b7fe26abfdaf397587"), name: '루피', age: 6 }  
{  
  _id: new ObjectId("63177b7fe26abfdaf397588"),  
  name: 'crong',  
  age: 4  
}
```



# updateMany

- 조건을 만족하는 모든 도큐먼트를 수정합니다

```
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  
}
```



# 검색



# findOne

- 검색 조건을 만족하는 최초의 도큐먼트를 찾아 줍니다
- 찾아서 값을 반환 하므로 변수에 넣어서 처리, 또는 바로 출력해 줘야 합니다

```
const data = users.findOne({ name: 'loopy' }, (err, findData) => {  
  console.log(findData);  
});
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규 수업/25/node_set  
$ node routes/mongo.js  
{ _id: new ObjectId("6317755f1c7057b4a9be2b37"), id: 'pororo', age: 5 }
```



# findOne

- 검색 조건을 만족하는 최초의 도큐먼트를 찾아 줍니다
- 찾아서 값을 반환 하므로 변수에 넣어서 처리, 또는 바로 출력해 줘야 합니다

```
const data = users.findOne({ name: 'loopy' }, (err, findData) => {  
  console.log(findData);  
});
```

```
const data = await users.findOne({ name: 'loopy' });  
console.log(data);
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규 수업/25/node_set  
$ node routes/mongo.js  
{ _id: new ObjectId("6317755f1c7057b4a9be2b37"), id: 'pororo', age: 5 }
```

# 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 저장됩니다(즉, 특정 DB를 가르키고 있다는 의미).
  - 리턴이 될 때, Await 을 사용하지 않습니다.
  - Find 로 찾은 값을 출력하려면 forEach() 라는 메소드를 사용해야만 합니다. 단, 이 때는 await 붙여 줘야만 출력이 가능합니다.
  - 데이터로 변경하려면 toArray() 메소드를 사용합니다. 역시 await 사용!



```
const cursor = users.find({
  name: 'loopy',
});

console.log(cursor);

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

```
const userCursor = users.find({
  name: "loopy",
});
const data = await userCursor.toArray();
console.log(data);
```

```
bsonRegExp: false,
serializeFunctions: false,
fieldsAsRaw: {},
enableUtf8Validation: true,
writeConcern: WriteConcern { w: 'majority' },
readPreference: ReadPreference {
  mode: 'primary',
  tags: undefined,
  hedge: undefined,
  maxStalenessSeconds: undefined,
  minWireVersion: undefined
}
}
[
  {
    _id: new ObjectId("638863f3982e74b721653945"),
    name: 'loopy',
    age: 6
  }
]
```



# \$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',  
    },  
  }  
)
```



# 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



```
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 cursor = 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 user = client.db('kdt5').collection('user');

  user.deleteMany({}, (err, deleteResult) => {
    if (deleteResult?.acknowledged) {
      user.insertOne(
        {
          name: 'pororo',
          age: 5,
        },
        (err, insertResult) => {
          if (insertResult?.acknowledged) {
            user.updateMany(
              {
                age: { $gte: 5 },
              },
              {
                $set: {
                  name: '5살 이상',
                },
              },
              (err, deleteResult2) => {
                if (deleteResult2?.acknowledged) {
                  const findDataCursor = user.find({});
                  findDataCursor.toArray((err, data) => {
                    console.log(data);
                    client.close();
                  });
                }
              },
            );
          }
        },
      );
    }
  });
});
```



# 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 작성 하셨던 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();
```

# 첫 데이터 베이스에 데이터 추가하기!





```

user.deleteMany({}, (err, deleteResult) => {
  if (deleteResult?.acknowledged) {
    user.insertMany(
      [
        {
          name: 'pororo',
          age: 5,
        },
        {
          name: 'loopy',
          age: 6,
        },
        {
          name: 'crong',
          age: 4,
        },
      ],
      (err, insertResult) => {
        if (insertResult?.acknowledged) {
          const findDataCursor = user.find({});
          findDataCursor.toArray((err, data) =>
            {
              console.log(data);
              client.close();
            }
          );
        }
      }
    );
  }
});

```

```

async function main() {
  await client.connect();

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

  await users.deleteMany({});
  await users.insertMany([
    {
      name: 'pororo',
      age: 5,
    },
    {
      name: 'loopy',
      age: 6,
    },
    {
      name: 'crong',
      age: 4,
    },
  ]);

  const data = users.find({});
  await data.forEach(console.log);

  await client.close();
}

```



삼입

# insertOne





```
client.connect((err) => {
  const user = client.db('kdt5').collection('user');

  user.deleteMany({}, (err, deleteResult) => {
    if (deleteResult?.acknowledged) {
      user.insertOne(
        {
          name: 'pororo',
          age: 5,
        },
        (err, insertResult) => {
          if (insertResult?.acknowledged) {
            const findDataCursor = user.find({});
            findDataCursor.toArray((err, data) => {
              console.log(data);
              client.close();
            });
          }
        },
      );
    }
  });
});
```

```
async function main() {
  await client.connect();
  const users =
    client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertOne({
    name: 'pororo',
    age: 5,
  });

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}

main();
```

# insertMany



```

user.deleteMany({}, (err, deleteResult) => {
  if (deleteResult?.acknowledged) {
    user.insertMany(
      [
        {
          name: 'pororo',
          age: 5,
        },
        {
          name: 'loopy',
          age: 6,
        },
        {
          name: 'crong',
          age: 4,
        },
      ],
      (err, insertResult) => {
        if (insertResult?.acknowledged) {
          const findDataCursor = user.find({});
          findDataCursor.toArray((err, data) => {
            console.log(data);
            client.close();
          });
        }
      },
    );
  }
});

```

```

async function main() {
  await client.connect();
  const users = client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertMany([
    {
      name: 'pororo',
      age: 5,
    },
    {
      name: 'loopy',
      age: 6,
    },
    {
      name: 'crong',
      age: 4,
    },
  ]);

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}

main();

```



# 삭제

# deleteOne





```

user.deleteMany({}, (err, deleteResult) => {
  if (deleteResult?.acknowledged) {
    user.insertMany(
      [
        {
          name: 'pororo',
          age: 5,
        },
        {
          name: 'loopy',
          age: 6,
        },
        {
          name: 'crong',
          age: 4,
        },
      ],
      (err, insertResult) => {
        if (insertResult?.acknowledged) {
          user.deleteOne(
            {
              name: 'crong',
            },
            (err, deleteResult2) => {
              if (deleteResult2?.acknowledged) {
                const findDataCursor = user.find({});
                findDataCursor.toArray((err, data) => {
                  console.log(data);
                  client.close();
                });
              }
            }
          );
        }
      }
    );
  }
});

```



```

async function main() {
  await client.connect();
  const users =
client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertMany([
    {
      name: 'pororo',
      age: 5,
    },
    {
      name: 'loopy',
      age: 6,
    },
    {
      name: 'crong',
      age: 4,
    },
  ]);

  await users.deleteOne({ name: 'crong' });

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}
main();

```

# deleteMany



```

async function main() {
  await client.connect();
  const users =
client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertMany([
    {
      name: 'pororo',
      age: 5,
    },
    {
      name: 'loopy',
      age: 6,
    },
    {
      name: 'crong',
      age: 4,
    },
  ]);

  await users.deleteMany({ age: { $gte: 5 } });

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}
main();

```

```

client.connect((err) => {
  const user = client.db('kdt5').collection('users');

  user.deleteMany({}, (err, deleteResult) => {
    if (deleteResult?.acknowledged) {
      user.insertMany(
        [
          {
            name: 'pororo',
            age: 5,
          },
          {
            name: 'loopy',
            age: 6,
          },
          {
            name: 'crong',
            age: 4,
          },
        ],
        (err, insertResult) => {
          if (insertResult?.acknowledged) {
            user.deleteMany({ age: { $gte: 5 } },
              (err, deleteResult2) => {
                if (deleteResult2?.acknowledged) {
                  const findDataCursor = user.find({});
                  findDataCursor.toArray((err, data) => {
                    console.log(data);
                    client.close();
                  });
                }
              });
          }
        }
      );
    }
  });
}

```





수정

# updateOne





```
user.deleteMany({}, (err, deleteResult) => {
  if (deleteResult?.acknowledged) {
    user.insertMany(
      [],
      (err, insertResult) => {
        if (insertResult?.acknowledged) {
          user.updateOne(
            {
              name: 'loopy',
            },
            {
              $set: {
                name: '루피',
              },
            },
            (err, deleteResult2) => {
              if (deleteResult2?.acknowledged) {
                const findDataCursor = user.find({});
                findDataCursor.toArray((err, data) => {
                  console.log(data);
                  client.close();
                });
              }
            }
          );
        }
      }
    );
  }
});
```

```
async function main() {
  await client.connect();
  const users = client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertMany([
  ]);

  await users.updateOne(
    {
      name: 'loopy',
    },
    {
      $set: {
        name: '루피',
      },
    },
  );

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}
main();
```

# updateMany



```

user.deleteMany({}, (err, deleteResult) => {
  if (deleteResult?.acknowledged) {
    user.insertMany([],
      (err, insertResult) => {
        if (insertResult?.acknowledged) {
          user.updateMany(
            {
              age: { $gte: 5 },
            },
            {
              $set: {
                name: '5살 이상',
              },
            },
            (err, deleteResult2) => {
              if (deleteResult2?.acknowledged) {
                const findDataCursor = user.find({});
                findDataCursor.toArray((err, data) => {
                  console.log(data);
                  client.close();
                });
              }
            },
          ),
        },
      ),
    },
  );
});

```

```

async function main() {
  await client.connect();
  const users = client.db('kdt5').collection('users');
  await users.deleteMany({});

  await users.insertMany([]);

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

  const data = users.find({});
  await data.forEach(console.log);
  await client.close();
}
main();

```





# 실습, Async / Await 로 변경

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

