

스타트업 개발자와 함께 공부하는 Node.js

07. 데이터베이스연동

강의 내용은 강사가 별도로 명시하지 않는 한 비공개로 간주합니다.
녹음이나 사진 촬영을 허락하지 않으며 콘텐츠를 블로그, SNS 등에 게시하거나 공개적으로 공유하지 마세요.

콘텐츠 공유 가능 여부에 대해 궁금한 점이 있는 경우 강사에게 문의하시기 바랍니다.



목차

1. 데이터베이스 소개
2. SQLite3 설치 및 사용
3. 비동기 처리
4. 페이징 게시판 만들기

데이터베이스



데이터베이스

개요

- ❑ 컴퓨터의 저장능력을 활용하여 자료를 저장, 가공, 활용
- ❑ 자료를 유용하게 활용하기 위해 합산, 집계, 통계 등의 알고리즘을 적용하여 정보로 가공



데이터베이스

DBMS(DataBase Management System)

- ❑ 정보의 저장과 관리를 전담하는 소프트웨어
- ❑ 프로그램과 데이터의 완벽한 분리
- ❑ 다수의 응용프로그램이 데이터를 공동으로 이용



데이터베이스

데이터베이스 역사

- ❑ 가장 고전적인 저장 방법은 종이
- ❑ 파일시스템: 메모장에 텍스트 또는 엑셀에 기록
- ❑ SAM 및 ISAM 파일 : 순차적으로 파일에 기록
- ❑ 1969년 관계형 데이터베이스 탄생

데이터베이스

DBMS 의 종류

- ❑ 오라클
- ❑ SQL서버
- ❑ MySQL
- ❑ DB2
- ❑ PostgreSQL
- ❑ Sqlite3

ORACLE



데이터베이스

DBMS 표준어

- ❑ 초창기에는 데이터베이스 제품에 따라 구조가 독특해 관리 방법 제각각
- ❑ 제품을 바꿀 때 마다 데이터베이스 언어를 다시 배워야 함
- ❑ 공통적인 표준 언어가 바로 SQL
- ❑ SQL은 고유의 문법 체계를 가지고 있음

```
SELECT * FROM user WHERE age > 30
```

데이터베이스

SQL 특징

- ❑ SQL은 대화식 언어
- ❑ 단순 명령을 조합하여 복잡한 명령 처리
- ❑ 제어문이 빈약해 프로그래밍 언어와 함께 사용

데이터베이스

테이블

- 관계형 데이터베이스는 정보를 표형태로 저장
- 익숙하고 직관적이다. 정보를 표형태로 표현한 것이 테이블(table) 이라고 부름
- 엔티티(Entity) : 테이블이 표현하는 대상
- 레코드(Record) : 테이블에 저장된 정보
- 필드(Field) : 레코드를 구성하는 각각의 세부 속성

데이터베이스

테이블

- ❑ 필드 여러 개가 모여서 레코드 하나가 되고
- ❑ 레코드 여러 개가 모여서 테이블이 됨

아이 디	이름	성별	나이	주소	MBTI
1	하니	여	19	서울	ENTP
2	헤린	여	20	서울	INTP
3	민지	여	19	서울	ISTJ

데이터베이스

오브젝트

- 테이블
- 인덱스
- 제약조건
- 뷰
- 프로시저

데이터베이스

명명규칙

- ❑ 같은 범위 내에서 이름이 중복되면 안됨
- ❑ 대소문자는 구분하지 않지만 일관성 유지
- ❑ 최대 길이 128 자
- ❑ SQL 예약어는 쓸 수 없음
- ❑ 가능하면 특수문자 사용 않음
- ❑ 간결한 이름이 중요
- ❑ 규칙을 일관되게 지키는 것이 중요

데이터베이스

데이터타입

- ❑ 데이터의 크기와 형태를 규정하는 것이 데이터타입
- ❑ 필요한 정보의 크기에 맞게 메모리를 사용해야 함
- ❑ DBMS가 타입을 미리 알고 있으면, 최적화된 방법으로 정보를 빠르게 읽을 수 있음
- ❑ 보통 5개의 데이터 타입을 지원(현실 세계 98%이상 표현)

타입	설명
INT	정수
DECIMAL	실수
CHAR	고정문자열
VARCHAR	가변문자열
DATE	날짜

SQLite

개요

- ❑ 소형 DBMS 엔진
- ❑ Self – Container
- ❑ Serverless
- ❑ Zero Configuration
- ❑ Transaction
- ❑ Cross – Platform
- ❑ Open Source

SQLite

설치

- ❑ 다운로드 페이지
- ❑ <https://www.sqlite.org/download.html>
- ❑ Precompiled Binaries for Window

[C:\SQLite] 디렉토리에 압축 풀기 →



Home About Documentation Download License Support Purchase

SQLite Download Page

Source Code

- [sqlite-amalgamation-3460000.zip](#) (2.64 MiB) C source code as an [amalgamation](#), version 3.46.0.
(SHA3-256: 1221eed70de626871912bfca144c00411f0c30d3c2b7935cff3963b63370ef7c)
- [sqlite-autoconf-3460000.tar.gz](#) (3.11 MiB) C source code as an [amalgamation](#). Also includes a "configure" script and [TEA](#) r
(SHA3-256: 83d2acf79453deb7d6520338b1f4585f12e39b27cd370fb08593afa198f471fc)

Documentation

- [sqlite-doc-3460000.zip](#) (10.34 MiB) Documentation as a bundle of static HTML files.
(SHA3-256: 54f612d5260e5a54fbf0824c83afbeb80f6b4b01148b89615d48af83e405dfc9)

Precompiled Binaries for Android

- [sqlite-android-3460000.aar](#) (3.44 MiB) A precompiled Android library [containing the core SQLite together with appropri](#);
(SHA3-256: c325e15e7dbe369eb1b2823d66bb9bf0fd516c241ac8dc9ea4a7f848ccdfb640)

Precompiled Binaries for Linux

- [sqlite-tools-linux-x64-3460000.zip](#) (10.54 MiB) A bundle of command-line tools for managing SQLite database files, including th
(SHA3-256: 7ba6ea0d94e7b945b22e98cc306220e35156a34fe6e7a370beb88580569a4caf)

Precompiled Binaries for Mac OS X (x86)

- [sqlite-tools-osx-x64-3460000.zip](#) (3.55 MiB) A bundle of command-line tools for managing SQLite database files, including th
(SHA3-256: 99e2b1014211151e94d6ce0c91de03494bc8d3b749a739af120f5387effe5de8)

Precompiled Binaries for Windows

- [sqlite-dll-win-x86-3460000.zip](#) (1.01 MiB) 32-bit DLL (x86) for SQLite version 3.46.0.
(SHA3-256: 41eafc690909cc4244166f46f86c5f9704e4e6508e6e4a7202f98511fbef221)
- [sqlite-dll-win-x64-3460000.zip](#) (1.26 MiB) 64-bit DLL (x64) for SQLite version 3.46.0.
(SHA3-256: f4824402a8a08af1d05f22d77e487b238d9ff7ebb28942abad08c40c7ae50e91)
- [sqlite-tools-win-x64-3460000.zip](#) (4.80 MiB) A bundle of command-line tools for managing SQLite database files, including th
(SHA3-256: a0cf6a21509210d931f1f174fe68cbfaa1979d555158efdc059a5171ce108e1a)

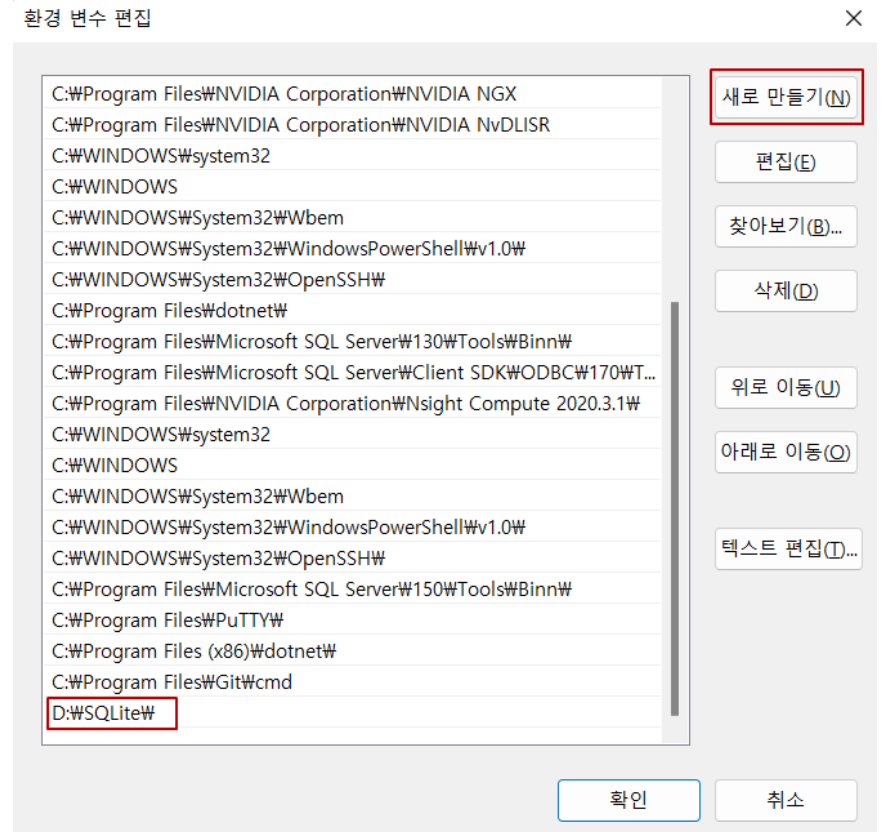
Precompiled Binaries for .NET

- [System.Data.SQLite](#) Visit the [System.Data.SQLite.org](#) website and especially the [download page](#) for :

SQLite

환경 변수 설정

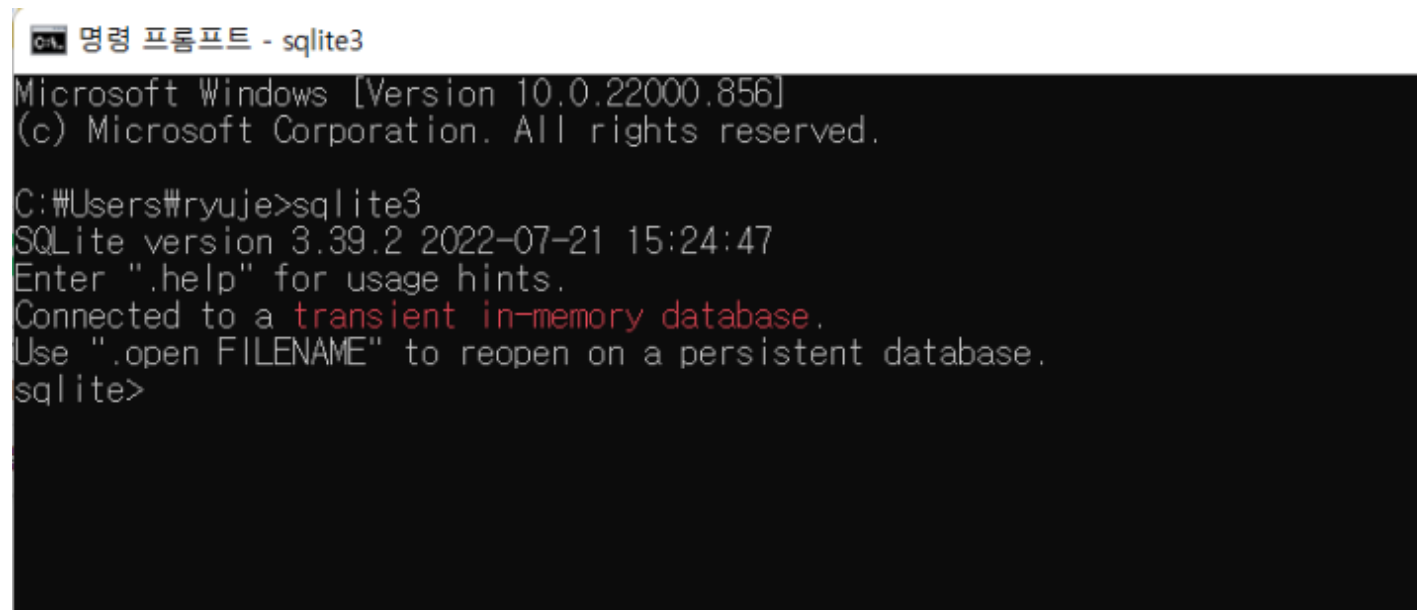
□ 환경 변수 편집 화면을 열고 **[C:\SQLite\]** 를 추가한다.



SQLite

Command Line Interface

- ❑ [윈도우 + R] 버튼을 누른 뒤 cmd 명령 입력 [Enter]
- ❑ 명령 프롬프트가 뜨면 sqlite3 입력 [Enter]



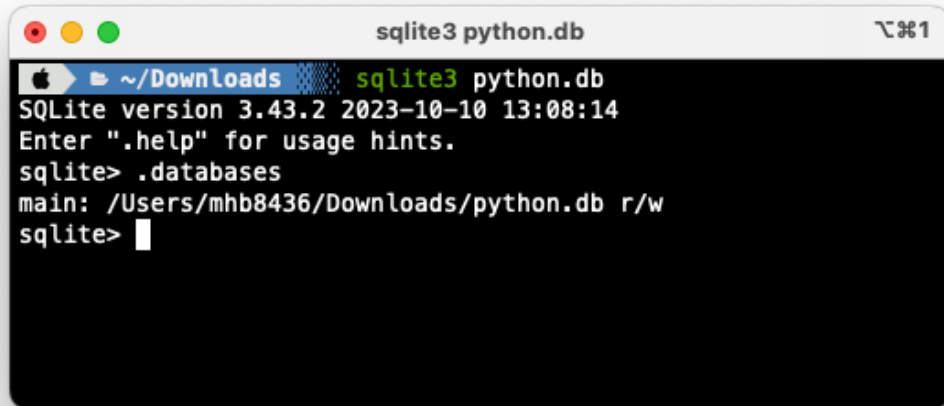
```
명령 프롬프트 - sqlite3
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ryuje>sqlite3
SQLite version 3.39.2 2022-07-21 15:24:47
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

SQLite

데이터베이스 생성

❑ 데이터베이스 생성 및 조회



```
sqlite3 python.db
SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
sqlite> .databases
main: /Users/mhb8436/Downloads/python.db r/w
sqlite>
```

sqlite3 [데이터베이스 파일 이름].db

sqlite> .databases

SQLite

테이블

□ 테이블 생성 및 조회

```
create table [tablename] (  
    field1_name [field_type] [option],  
    field2_name [field_type]  
)
```

SQLite

테이블 생성 및 조회

```
// 07/ch07_01.sql

1  create table users (
2      user_id integer primary key autoincrement,
3      name text not null,
4      email text,
5      phone text,
6      create_dt date default (datetime('now', 'localtime'))
7  )
8
9
10
```

SQLite

테이블 생성 옵션

- ❑ primary key
- ❑ autoincrement
- ❑ not null
- ❑ default
- ❑ datetime('now', 'localtime')

SQLite

레코드 추가

- 테이블에 레코드 추가

```
insert into [tablename]([fieldname],..)
values([field value], ..)
```

SQLite

레코드 추가

```
// 07/ch07_02.sql
```

```
1  insert into users(name, email, phone) values ('김민준', 'kim01@naver.com', '010-2221-3433');
2  insert into users(name, email, phone) values ('이서연', 'lee01@naver.com', '010-2222-3533');
3  insert into users(name, email, phone) values ('이정재', 'lee02@naver.com', '010-2223-3633');
4  insert into users(name, email, phone) values ('박지훈', 'park1@naver.com', '010-2224-3733');
5  insert into users(name, email, phone) values ('최소진', 'choi1@naver.com', '010-2225-3383');
6  insert into users(name, email, phone) values ('이나영', 'lee03@naver.com', '010-2226-3393');
7  insert into users(name, email, phone) values ('이현지', 'lee04@naver.com', '010-2227-3303');
8
9
10
```

SQLite

테이블 조회

□ SELECT 쿼리문을 사용

```
select [field name], .. from [table name] where [condition]
```

```
select * from [table name] where [condition]
```

SQLite

테이블 조회

```
// 07/ch07_03.sql  
  
1  select * from users;  
2  
3  select user_id, name from users;  
4  
5  select user_id, name, email, phone from users;  
6  
7  select user_id, name, email, phone, create_dt from users;  
8  
-
```

SQLite

테이블 조회

□ 특정 조건 데이터 조회

`select [field name], .. from [table name] where name = '김민준'`

`=`
`like`
`<>`

The diagram illustrates the modification of a SQL query. The original query is `select [field name], .. from [table name] where name = '김민준'`. The equals sign (`=`) is circled in red. An arrow points from this circle to a set of angle brackets (`<>`). Above the angle brackets, the word `like` is written, and above `like`, another equals sign (`=`) is shown, indicating the replacement of the original equals sign with `like` and `<>`.

SQLite

특정 조건 데이터 조회

```
// 07/ch07_04.sql
```

```
1  select * from users where name = '김민준';
2
3  select * from users where user_id > 2;
4
5  select * from users where email like '%naver%';
6
7  select * from users where create_dt between '2024-07-08' and '2024-07-09';
8
9  select * from users where user_id between 3 and 5;
10
11
```

SQLite

데이터 수정

- 특정 필드의 데이터를 수정할 때에는 UPDATE 쿼리문 사용

`update [table name] set [field name] = [value] where name = '김민준'`

`=`
`like`
`<>`
↑
`=`

SQLite

특정 레코드 필드 수정

```
// 07/ch07_05.sql
```

```
1  update users set phone = '010-7777-8888' where name = '김민준';
2
3  update users set create_dt = '2024-12-01' where user_id > 3;
4
5  update users set email = 'abcd@gmail.com' where email = 'lee02@naver.com';
6
7  select * from users;
8
```


SQLite

데이터 삭제

- 특정 필드의 데이터를 삭제할 때에는 DELETE 쿼리문 사용

`delete from [table name] where name = '김민준'`

Diagram illustrating the comparison operator in the SQL query:

- The equals sign (`=`) in the query is circled in red.
- An arrow points from the circled equals sign to the text `<>`.
- The text `like` is positioned above `<>`.
- The text `=` is positioned above `like`.

SQLite

특정 레코드 필드 삭제

```
// 07/ch07_06.sql
```

```
1  delete from users where name = '김민준';  
2  
3  delete from users where user_id > 7;  
4  
5  delete from users where email like '%gmail.com%';  
6  
7  delete from users where create_dt > '2024-11-01';  
8  
9  select * from users;  
10  
11
```

SQLite

확인문제

```
// 07/ch07_sol_01.sql
// 1. 다음과 같이 데이터베이스와 테이블을 만들어보세요
// 데이터베이스 : board.db
// 테이블 명 : board
// 컬럼 1 : id int형, pk, 자동증가
// 컬럼 2 : title varchar 형
// 컬럼 3 : content text 형
// 컬럼 4 : writer text 형
// 컬럼 5 : write_date text 형

// 2. 테이블에 10개의 데이터를 넣어보세요
```

비동기 처리



비동기 처리

소개

□ 동기 코드

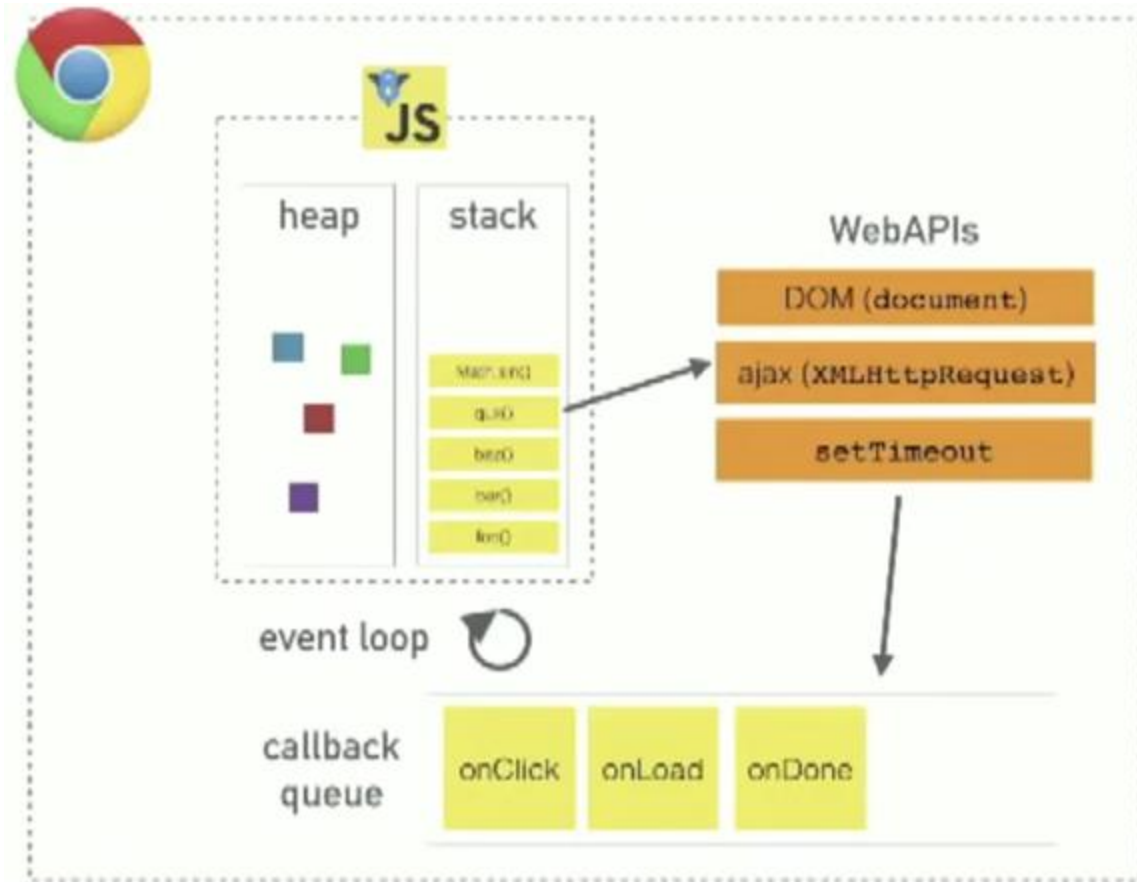
- 대부분의 프로그램 코드는 동기식
- 코드가 한 줄 씩 실행
- 이전 줄이 완료될 때 까지 대기
- 시간이 오래 걸리는 코드는 블록
- 본질적 블로킹

□ 비동기 코드

- 백그라운드에서 코드 실행
- 본질적 논-블로킹
- 이전 작업이 완료될 때 까지 기다리지 않음

비동기 처리

자바스크립트 엔진 구조



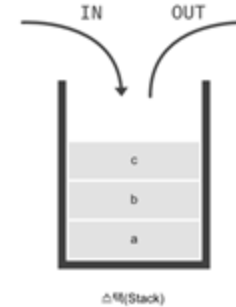
Overview of major components in a browser

- ☐ Heap
- ☐ Stack
- ☐ Web API
- ☐ Callback Queue
- ☐ Event Loop

비동기 처리

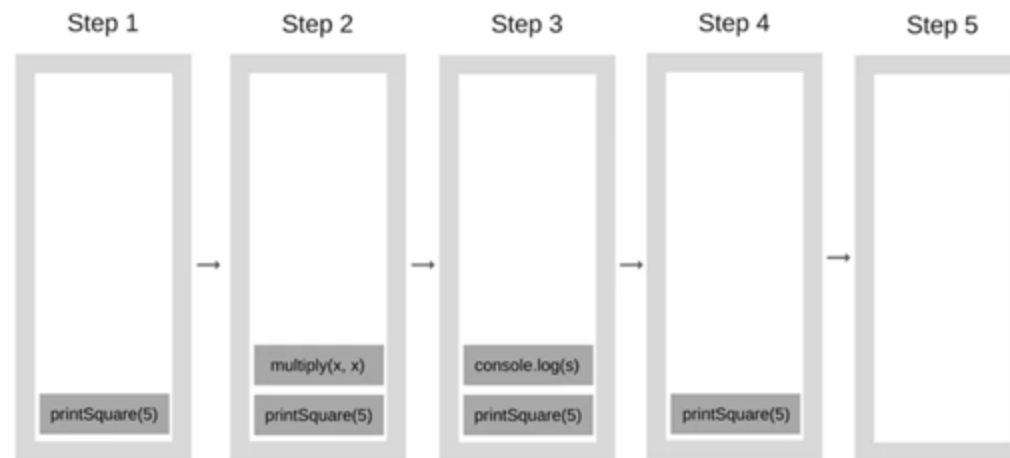
싱글 스레드 및 콜백

- ❑ Stack 자료 구조는 후입 선출
- ❑ Callback은 프로그램 상 위치를 기록하는 자료 구조
- ❑ 호출 함수가 가장 상단엔 위치



```
function multiply(x, y) {  
  return x * y;  
}  
  
function printSquare(x) {  
  var s = multiply(x, x);  
  console.log(s);  
}  
  
printSquare(5);
```

Call Stack



콜백

일반 코드

```
// 07/ch07_01.js

1 console.log(`----- normal job-----`)
2 function getDB(){
3   let data
4   data = 100;
5   return data;
6 }
7
8 function list(){
9   let value = getDB();
10  value += 2;
11  console.log(`list value : ${value}`);
12 }
13
14 list();
```

오래 걸리는 코드

```
// 07/ch07_02.js

1 console.log(`----- long time job-----`)
2 function getDB(){
3   let data
4   setTimeout(()=>{
5     data = 100;
6   }, 1000);
7   return data;
8 }
9
10 function list(){
11   let value = getDB();
12   value += 2;
13   console.log(`list value : ${value}`);
14 }
15
16 list();
```


콜백

callback

// 07/ch07_03.js

```

1 console.log(`----- callbak -----`)
2 function getDB(callback){
3     let data
4     setTimeout(()=>{
5         data = 100;
6         callback(data);
7     }, 1000);
8     return data;
9 }
10
11 function list(){
12     getDB((value)=> {
13         value += 2;
14         console.log(`list value : ${value}`);
15     });
16 }
17
18 list();
19

```

- ❑ 콜 백(call back) 함수는 함수가 실행 된 뒤에 호출 되는 함수를 의미
- ❑ 단점 : 콜 백 지옥(Callback Hell)

```

step1(function (value1) {
  step2(function (value2) {
    step3(function (value3) {
      step4(function (value4) {
        step5(function (value5) {
          step6(function (value6) {
            // Do something with value6
          });
        });
      });
    });
  });
});

```

콜백

callback

// 07/ch07_04.js

```

1 console.log(`---- promise ----`)
2 function getDB(){
3     let data;
4     return new Promise((resolve, reject) => {
5         setTimeout(()=> {
6             data = 100;
7             resolve(data);
8         }, 1000);
9     });
10 }
11
12 function list(){
13     getDB().then((value)=> {
14         let data = value + 2;
15         console.log(`list value : ${data}`);
16     })
17     .catch((error)=> {
18         console.error(error);
19     });
20 }
21
22 list();

```

- ❑ 프로미스 상태 종류 : 이행, 대기, 거절
- ❑ 생성 : 대기
- ❑ resolve 호출 : 이행
- ❑ reject 호출 : 거절
- ❑ 단점 : Promise Hell

```

wakeUp()
    .then(data => {
        console.log(data)

        haveMeal()
            .then(data => {
                console.log(data)

                drinkSoju()
                    .then(data => {
                        console.log(data)

                        sleep()
                            .then(data => {
                                console.log(data)

                                })
                            })
                        })
                    })
                })
            })
        })
    })

```

콜백

async / await

```
// 07/ch07_05.js
1 console.log(`---- async/await ----`)
2 function getDB(){
3     let data;
4     return new Promise((resolve, reject) => {
5         setTimeout(()=> {
6             data = 100;
7             resolve(data);
8         }, 1000);
9     });
10 }
11
12 async function list(){
13     let data = await getDB();
14     data += 2;
15     console.log(`list value : ${data}`);
16 }
17
18 list();
```

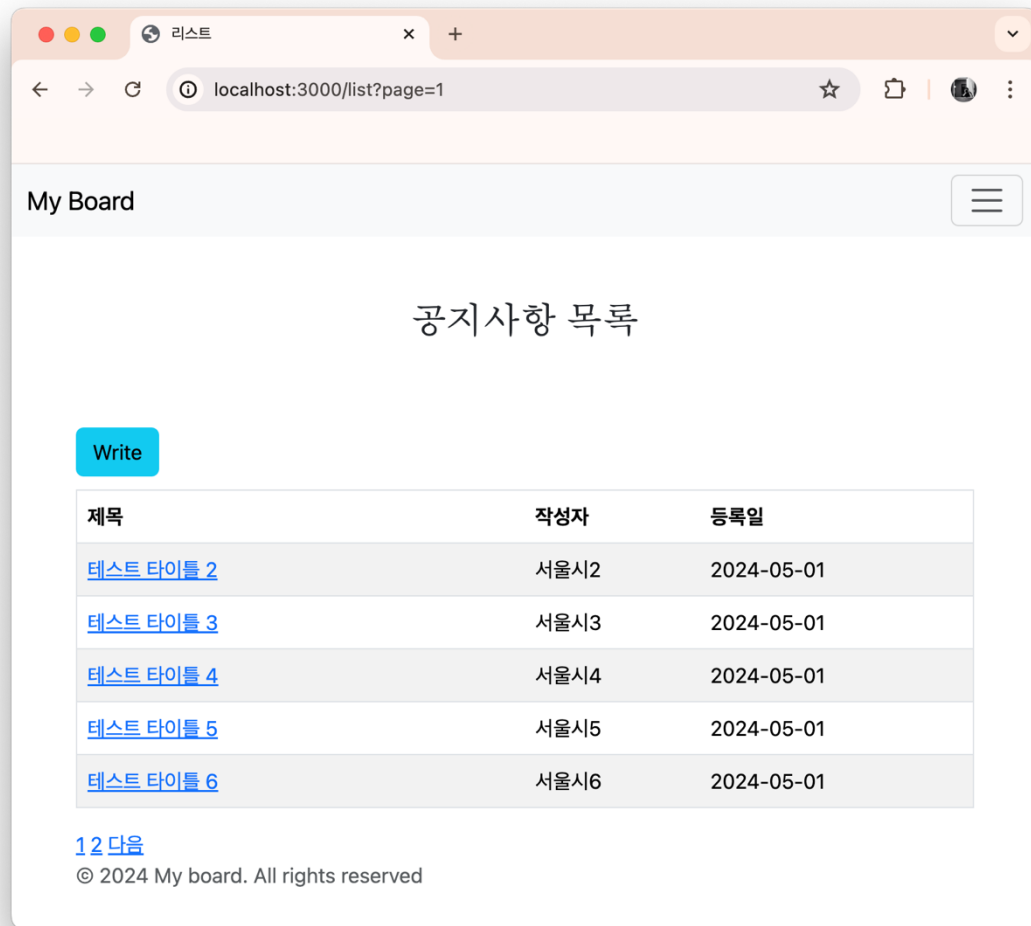
- ❑ Promise 기반
- ❑ async와 await 사용

페이징 게시판

```
function todoitem(data) ;  
    var self = this  
    data = data || {}  
    / / Son - persisted properties  
    <html> <errorMessage = text - '200px'>to , observable() ;  
    <style="color:orange;">HTML font code is here  
    / / Son - persisted properties  
    <html> <errorMessage = text - '200px'>to , observable() ;  
    <body style="background-color:yellowgreen">  
    <div - '200px'> <todolistid = data.todolistid  
    <div - '200px'> <errorMessage = ko , observable() ;
```

페이징 게시판

소개



- 한 페이지에 5개 게시물
- 하단에 페이지 번호
- 페이지 번호 클릭 시 해당 페이지로 이동

페이징 게시판

페이징을 위한 기술

1. SQL 문 LIMIT, OFFSET
2. 페이징을 위한 변수

```
select * from [table name]  
where [condition]  
order by date desc limit [value] offset [value]
```

```
select * from board limit 20 offset 5
```

페이징 게시판

페이징을 위한 기술

1. SQL 문 LIMIT, OFFSET
2. 페이징을 위한 변수

page : 현재 페이지

limit : 한 페이지에 나오는 글의 수

offset : 시작 게시글 번호 $\Rightarrow (page - 1) * limit$

totalPage : 전체 게시글 개수

페이징 게시판

프로젝트 생성

```
npm init -y  
npm install express nodemon sqlite3  
moment
```

- ❑ [C]-[nodejs]-[project]-[07]-[ch07_06]
- ❑ 왼쪽 커맨드로 프로젝트 생성
- ❑ 의존성 설치

페이징 게시판

리스트

```
29 app.get('/list', (req, res) => {
30   let p = req.query.page;
31
32   const page = req.query.page ? parseInt(req.query.page) : 1
33   const limit = 5;
34   const offset = (page - 1) * limit;
35
36   let sql = `select id, title, content, writer, write_date
37             from board ORDER BY write_date DESC LIMIT ? OFFSET ? `;
38
39   db.all(sql, [limit, offset], (err, rows) => {
40     if (err) {
41       console.error(err.message);
42       res.status(500).send("Internal Server Error");
43     } else {
44       db.get(`SELECT COUNT(*) as count FROM board`, (err, row) => {
45         if (err) {
46           console.error(err);
47           res.status(500).send("Internal Server Error");
48         } else {
49           const total = row.count;
50           const totalPages = Math.ceil(total / limit);
51           res.render('pages/list', { items: rows, currentPage: page, totalPages });
52         }
53       });
54     }
55   });
56 });
57 });
```

페이징 게시판

글 상세

```
60 app.get('/detail/:id', (req, res) => {
61   const id = req.params.id;
62
63   let sql = `select id, title, content, writer, write_date from board where id = ${id}`;
64   console.log(`id => ${id}, sql => ${sql}`);
65   let detail = {};
66   db.all(sql, [], (err, rows) => { // 6. run query
67     if (err) {
68       console.error(err.message);
69     }
70     // console.log(rows);
71     rows.forEach((row) => {
72       detail = row;
73     });
74     console.log(detail);
75     res.render('pages/detail', {detail: detail}); // 8. render page with data
76   });
77 });
```

페이징 게시판

글 쓰기

```
80 app.get('/write', (req, res) => {
81   |   res.render('pages/write');
82 });
83
84 app.use(express.urlencoded({ extended: true }));
85
86 app.post('/write', (req, res) => {
87   |   console.log('/write post', req.body);
88   |   const write_date = moment().format('YYYY-MM-DD');
89   |   let sql = `insert into board(title, content, writer, write_date)
90   |       values('${req.body.title}', '${req.body.content}', 'tester', '${write_date}')`
91   |   db.run(sql, (err) => {
92   |       |   if(err) {
93   |       |       |   console.error(err);
94   |       |   }
95   |       |   console.log(`A row has been inserted with rowid ${this.lastID}`)
96   |       |   res.redirect('/list');
97   |   });
98 });
```

페이징 게시판

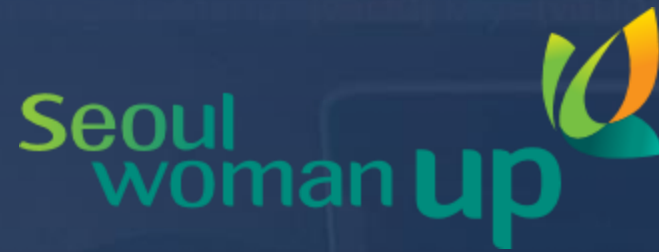
글 수정

```
101 app.get('/update/:id', (req, res) => {
102   const id = req.params.id;
103
104   let sql = `select id, title, content, writer, write_date from board where id = ${id}`;
105   console.log(sql);
106   db.all(sql, [], (err, rows) => { // 6. run query
107     if (err) {
108       console.error(err.message);
109     }
110     let detail = {};
111     rows.forEach((row) => {
112       detail = row;
113     });
114     res.render('pages/update', {detail: detail}); // 8. render page with data
115   });
116 });
117
118 app.post('/update/:id', (req, res) => {
119   const id = req.params.id;
120
121   let sql = `update board set title = '${req.body.title}', content = '${req.body.content}' where id = ${id}`;
122   db.run(sql, (err) => {
123     if(err) {
124       console.error(err);
125     }
126     console.log(`A row has been updated with rowid ${this.lastID}`)
127     res.redirect('/list');
128   });
129 });
```

페이징 게시판

글 삭제

```
131 app.get('/delete/:id', (req, res) => {  
132     const id = req.params.id;  
133  
134     let sql = `delete from board where id = ${id}`  
135     db.run(sql, (err) => {  
136         if(err) {  
137             console.error(err);  
138         }  
139         console.log(`A row has been deleted with rowid ${this.lastID}`)  
140         res.redirect('/list');  
141     });  
142 });
```



백문이 불여일타