



SOPT  
20<sup>th</sup> server part  
5차 세미나

Database와 Node.js



1. Workbench 사용법 (실습)
2. Node.js 에서 mysql DB 연동하기 (이론, 실습)
  - sql injection 방어하기
  - transaction 구현하기
3. 정규화 (이론)
4. cascade (실습)
5. DB 백업하기 (실습)



# 1.Workbench 사용법 (실습)

# 1. Workbench 사용법



- DB에서 데이터를 처리하기 위한 4가지 기능 (CRUD) -

**저장(Create)** : insert

**조회(Read or Retrieve)** : select

**수정(Update)** : update

**삭제>Delete)** : delete

(사용자 인터페이스(UI)가 갖추어야 할 4가지 기능이기도 합니다.)

# 1. Workbench 사용법



Set) mysql command line client로 DB를 관리한다.

- 100개의 서로 다른 값을 가지는 레코드를 insert 하고 싶다면?  
→ `insert into table values(value1, value2, ..) X 100`
  - 스키마를 생성하고 여러 개의 테이블을 생성해야 한다면?  
→ `create schema,,, create table(col1, col2, col3,,,)...`  
(귀찮고 sql 쿼리 문법 오류나면 다시 쳐야 함.)
- ∴ GUI 형태의 데이터베이스 관리 툴인 workbench 사용



# 1. Workbench 사용법

- 테이블 생성 시 column 설정

PK	NN	UQ	B	UN	ZF	AI	G
<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"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

PK(Primary Key) : 기본키

NN(Not Null) : null 값 허용 안 함

UQ(Unique) : 테이블 내 중복되는 column값 허용 안 함

AI(Auto Increment) : 레코드가 insert되면 자동으로 값 증가

B(Binary) : 이진 값

ZF(Zero Fill) : 정수 값 들어오면 앞에 0으로 채움 ex) 1 → 00001

UN(Unsigned) : 음이 아닌 값만 허용



## 2. Node.js 에서 mysql DB 연동하기(실습)



## 2. Node.js와 DB 연동

- node.js에서 Database에 데이터를 조회, 저장, 수정, 삭제
- `npm install mysql`
- DB 서버에 접속하기 위한 config 정보가 있어야 한다.
- config 정보를 가지고 mysql의 connection 객체 생성.



## 2. Node.js와 DB 연동



```
[ RowDataPacket { id: 1, name: '김다은', part: '서버' } ]
```

### 주의할 점!

**select 문으로 가져온 레코드는 JSON 객체배열 형태**

예) data :

```
{  
  id: 1,  
  name: '김다은',  
  part: '서버'  
}
```

]

**∴ data[i].id 로 select한 레코드의 id값 접근**

## 2. Node.js와 DB 연동



### 주의할 점! - `connection.release()` (connection 반납)

- `connection.query(...){...}`에서 작업이 끝나면 반드시 connection 반납.
  - query 에러가 발생해도 꼭 connection을 반납해야 한다.
  - pool 내부의 connection 개수는 유한하므로 제때 반납하지 않으면 connection leak 발생, 다른 쿼리 작업들이 지연된다.
  - connection을 이용한 query, release 메소드도 비동기 방식.
- ∴ `connection.query(...){...}` 내부에서 `connection.release()`를 호출해야 한다.



## 2. Node.js와 DB 연동

```
let query = 'select password, id from trainer where email = ?';
connection.query(query, req.body.email, function(err, result){
  if(err) {
    connection.release();
    callback("first query error"+err, null);
  }
  else if(result.length===0){ // 입력한 이메일주소의 정보가 디비에
    res.status(403).send('존재하지 않는 계정입니다. ');
    connection.release();
    callback(null, null);
  }
  else callback(null, result, connection);
}
```

query 에러가 발생했을 때 에러처리하고 connection 반납.

요청에 대한 응답 처리하고 나서도 connection 반납.

다음 태스크에서 query를 해야 할 때만 connection을 넘겨준다.

## 2. Node.js와 DB 연동



### connection 반납의 나쁜 예

```
let query = 'select password, id from trainer where email = ?';
connection.query(query, req.body.email, function(err, result){
  if(err) {
    callback("first query error"+err, null);
  }
  else if(result.length===0){ // 입력한 이메일주소의 정보가 디비에
    res.status(403).send('존재하지 않는 계정입니다.');
```

```
    callback(null, null);
  }
  else callback(null, result, connection);
});
connection.release();
```

connection.query() 보다 connection.release()가 먼저 실행되어서(∴ 비동기)  
DB에 query를 날리기도 전에 DB와 연결이 해제됨.

## 2-1. SQL injection attack



**SQL Injection**

서버개발자가 생각치 못한 SQL문을 실행되게 함으로써 데이터베이스를 비정상적으로 조작하는 코드 삽입공격 방법.

## 2-1. SQL injection attack



### 학생 점수 테이블 in DB

student_id	name	score
3349	하태경	20
3928	김다운	70
9323	김주영	89

[www.sopt.ac.kr/student/score](http://www.sopt.ac.kr/student/score)

### 점수조회

학번 : 3349

점수조회



### 결과(자기 점수만 확인)

```
[{"student_id": "3349", "name": "하태경", "score": 20}]
```

## 2-1. SQL injection attack



### 학생 점수 테이블 in DB

student_id	name	score
3349	하태경	20
3928	김다은	70
9323	김준영	89

www.sopt.ac.kr/student/score

### 점수조회

학번 :

점수조회



### 결과(모든 학생 점수를 조회)

```
[{"student_id": "3349", "name": "하태경", "score": 20},  
{"student_id": "3928", "name": "김다은", "score": 70},  
{"student_id": "9323", "name": "김준영", "score": 89}]
```

## 2-1. SQL injection attack



```
let query = 'select * from student where student_id = ';  
connection.query(query + req.body.student_id, function(err, data){  
  if(err) console.log("query error : " + err);  
  else if(data.length===0) res.status(403).send("학생정보가 없습니다.");  
  else res.status(201).send(data);  
});
```

쿼리 : select \* from student **where student\_id = 1234 or 1=1**

‘1=1’ 은 true이므로 where절 전체가 true로 바뀌면서 모든 레코드를 가져옴  
(본인 점수만 확인하도록 개발자는 의도했으나 다른 학생 점수도 볼 수 있게 함.)

위와 같은 공격방법을 ‘SQL injection attack’  
(sql 주입공격)이라고 한다.



## 2-1. SQL injection attack



### SQL injection attack을 방지하려면

- ① 사용자가 입력한 데이터를 그대로 가지고 SQL query를 만들면 안 된다.

`query + req.body.student_id` (X)

- ② `escape`메소드(sql구문에 맞는 형변환 + sql주입공격 방어역할) 사용

`connection.query(query, mysql.escape(req.body.student_id)...`

- ③ SQL query문에 placeholder(물음표) 사용

`query = 'select * from student where student_id = ?';`

`connection.query(query, req.body.student_id, function(...))`

## 2-2. transaction



### transaction

- 여러 개의 query들로 이뤄진 하나의 논리 작업.
- 여러 개의 query들이 동일한 connection 객체를 공유한다.
- 작업이 100% 완료되도록 보장하거나, 100% 완료되지 않으면 작업 전 상태로 돌아가도록 작업이 아예 취소되는 것을 보장한다.

## 2-2. transaction



**= 1 transaction**

예) 상품 구매 작업

사용자가 포인트로 상품 구매를 완료해서 사용자 포인트잔액이 차감된 후에,

사용자의 구매이력에 상품이 기록되려고 하는데 서버가 shutdown 될 경우

→ 포인트는 차감이 됐는데 구매이력에 구매한 상품이 없음. (큰일남)

→ shutdown을 포함한 여러 에러가 발생하면 지금까지 해왔던 query들이 모두 취소(rollback)되어야 한다. (100% or 0% 작업완료가 보장되어야.)

```
pool.getConnection(function(err1, connection){
  if(err1) { console.log('getConnection error: ',err1); return; }
  else {
    connection.beginTransaction(function(err2){
      if(err2){ console.log('beginning transaction error: ',err2); return; }
      else {
        let query1 = '....';
        connection.query(query1, function(err3, data1){
          if(err3){
            console.log('first trasaction query error:', err3);
            connection.rollback();
            // connection.release();
          }
          else {
            let query2 = '....';
            connection.query(query2, function(err4, data2){
              if(err4){
                console.log('first trasaction query error:', err4);
                connection.rollback();
                // connection.release();
              }
              else connection.commit();
            });
          }
        });
      }
    });
  }
}
```



### 3. 정규화



### 3. 정규화

- RDMBS에서는 테이블 간의 중복되는 정보가 없어야 한다.
- 동일한 정보가 여러 테이블에 저장되어 있으면 정보 수정 시 부담이 있고, 무결성 제약 유지가 쉽지 않다.
- 무결성 제약 : 테이블에 원치 않는(null 또는 중복)값이 들어가지 않도록 하는 제약.  
예) PK, UQ, FK NN, 옵션들은 RDBMS에서 무결성을 유지하게 한다.
- 테이블 간의 중복데이터가 없게 하려면 외래키(FK)로 다른 테이블의 데이터를 참조해야.  
→ 참조 당하는 테이블에는 반드시 데이터가 있어야 한다. ( = 참조 무결성 제약 )  
→ 중복성 제거 후 필요한 정보는 join연산으로 가져온다.
- DB에서 무결성을 유지하려면 PK, UQ, FK NN 말고도 정규화 작업이 필요함.

### 3. 정규화



정규화 : 데이터베이스 내에 중복되는 정보들을 최소화 하는 것.

#### pokemonster

trainer_id	email	trainer	password	pokemon_id	pokemon	character
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	1	피카츄	전기
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	2	이상해씨	씨앗
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	1	피카츄	전기
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	3	꼬부기	물
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	3	꼬부기	물

### 3. 정규화



**정규화 : 데이터베이스 내에 중복되는 정보들을 최소화 하는 것.**

**pokemonster**

trainer_id	email	trainer	password	pokemon_id	pokemon	character
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	1	피카츄	전기
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	2	이상해씨	씨앗
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	1	피카츄	전기
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	3	꼬부기	물
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	3	꼬부기	물

**문제 :**

- 한 테이블 내에서 trainer와 pokemon의 정보가 중복된다.
- 하나의 정보가 수정되면 그 정보가 포함된 모든 레코드를 수정해야 한다.
- 트레이너 한 명에게 포켓몬을 추가할 때마다 트레이너 정보를 또 입력해야 한다.



### 3. 정규화



#### trainer

id	email	name	password	pokemon_id
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	1
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	2
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	1
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	3
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	3

#### pokemon

id	name	character
1	피카츄	전기
2	이상해씨	씨앗
3	꼬부기	물

### 3. 정규화



#### trainer

id	email	name	password	pokemon_id
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	1
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	2
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	1
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517	3
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234	3

#### pokemon

id	name	character
1	피카츄	전기
2	이상해씨	씨앗
3	꼬부기	물

문제 :

trainer 정보 중복됨

(앞서 언급된 문제들 아직까지 미해결)

### 3. 정규화



**trainer**

id(PK)	email	name	password
1036	<a href="mailto:kde6260@gmail.com">kde6260@gmail.com</a>	김다은	1234
2457	<a href="mailto:0909@gmail.com">0909@gmail.com</a>	이영규	4517

**owns**

trainer_id (refers trainer.id)	pokemon_id (refers pokemon.id)
1036	1
1036	2
1036	3
2457	1
2457	3

**pokemon**

id(PK)	name	character
1	피카츄	전기
2	이상해씨	씨앗
3	꼬부기	물

- 세 테이블 모두 중복되는 레코드 없음.
- 필요한 레코드는 join연산으로 가져옴.



## 4. cascade

## 4. cascade



- 정규화를 하다 보면 여기저기 참조하고 참조당하는 키들이 많아진다.
- 참조 당하는 column에서 수정 또는 삭제가 발생할 때 그 column을 참조하는 다른 테이블에서도 수정 또는 삭제가 되게 하고싶다면?
- cascade 옵션 쓴다. (cascade : 종속)

add constraint 'trainer\_id'

foreign key ('trainer\_id') references trainer.id

on delete cascade

← trainer테이블의 id가 삭제되면 id를 참조하는 레코드도 삭제

on update cascade;

← trainer테이블의 id가 수정되면 id를 참조하는 외래키도 수정

## 4. cascade



on delete **restrict** | **cascade** | **set null** | **no action**

아래의 옵션들은 참조당하는 키 값이 삭제 되었을 때 (수정되었을 때는 on update)  
외래키가 어떻게 동작할 것인가에 대한 옵션을 나타낸다.

**restrict** - 참조 당하는 키 값이 삭제되어도 지우지 말고 버텨라. (에러 발생시킴)

**cascade** - 참조 당하는 키 값이 삭제되면 참조하는 외래키의 레코드도 삭제해라.

**set null** - 참조 당하는 키 값이 삭제되면 참조하는 외래키를 null로 변경해라.

**no action** - 참조 당하는 키 값이 삭제되면 그냥 무시해라. (에러 발생시킴)



## 5. DB 백업 및 복구

## 5. DB 백업 및 복구



백업 : 데이터베이스, 테이블을 파일에 저장 할 수 있다. (=덤프를 떠놓는다.)

CLI : cmd 창 열어서 현재 작업 디렉토리를

C:\Program Files\MySQL\MySQL server 5.x\bin 으로.

특정 DB만 백업하려면

➤ `mysqldump -u아이디 -p -databases [DB이름] > 덤프파일이름.sql`

특정 DB의 특정 테이블만 백업하려면

➤ `mysqldump -u아이디 -p [DB이름] [table이름] > 덤프파일이름.sql`

데이터 없이 스키마만 백업하려면

➤ `mysqldump -u아이디 -p --no-data ... > 덤프파일이름.sql`

스키마 없이 데이터만 백업하려면

➤ `mysqldump -u아이디 -p --no-create-info ... > 덤프파일이름.sql`



## 5. DB 백업 및 복구



복구 : 파일에 저장된 데이터베이스, 스키마, 테이블을 불러올 수 있다.

CLI : cmd 창 열어서 현재 작업 디렉토리를

C:\Program Files\MySQL\MySQL server 5.x\bin 으로.

빈 DB스키마에 DB를 복구하려면

➤ `mysql -uroot -p [복구 받을 DB이름] < 덤프파일이름.sql`

빈 테이블스키마에 테이블을 복구하려면

➤ `mysql -uroot -p [DB이름] [복구 받을 table이름] < 덤프파일이름.sql`



## 회원가입, 로그인, 소유포켓몬 조회하기

- 스키마

trainer(id, email, name, password)

pokemon(id, name, character)

owns(trainer\_id, pokemon\_id) → trainer\_id, pokemon\_id는 외래키

- 트레이너와 포켓몬은 N:N 관계입니다.
- owns 테이블에는 workbench에서 직접 레코드를 입력하세요.  
(포켓몬 추가하는 기능 넣으셔도 됩니다.)
- 회원가입, 로그인 기능에는 bcrypt모듈을 응용해주세요.
- next 호출 없이 하나의 미들웨어 내부에서 모든 처리가 이뤄지도록 흐름제어(async or promise)를 해주세요.



## 회원가입, 로그인, 소유포켓몬 조회하기

- 이미 가입된 이메일주소로는 재가입을 할 수 없습니다.(시연영상 참고)
- 로그인에 성공하면 로그인한 트레이너가 소유하고 있는 포켓몬 리스트가 출력됩니다.
- 로그인에 성공하면 '/pokemon/트레이너 아이디'로 redirect하는 점에 유의해주세요. (res.redirect(), req.params 참고)
- 위 과제가 어려우신 분들은 회원가입 기능만 하셔도 좋습니다.
- 5/20까지 node\_modules를 제외한 express 프로젝트를 압축해서 제출해주세요.