12171800 심정호

1. 

관리자용과 예약 페이지를 나누기 위해 user table을 만들어주었다.

텍스트, 점수판, 클래퍼보드이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

User table에는 id가 primarykey로 들어간다. Id가 다른 모든 attribute들을 결정하므로 2nf를 만족하고 다른 속성들은 결정자의 역할을 하지 못하므로 3nf도 만족한다.

동명이인과 Role admin이 여러 명인 경우 등을 고려했다.

텍스트, 화면, 스크린샷, 은색이(가) 표시된 사진

자동 생성된 설명

Final\_project database와 연결해준다.



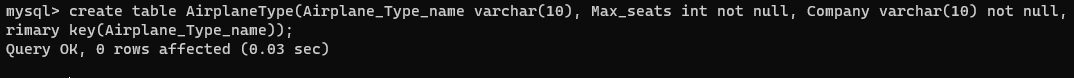
Airport table을 만들어주었다.

Airport\_code = char(6) = attribute가 3가지 이므로 각 attribute에서 두 글자씩 의미를 나타내 airport\_code를 나타내기로 했다. 인천 공항의 경우 한국의 국가 전화번호 +82를 이용하고 인천의 경우 state와 city모두 incheon이므로 82inin 이런 식이다.

Name과 state, city모두 airport가 존재하기 위해서는 필수적인 것이므로 not null 속성을 모두 주었고 airport code를 primary key로 설정했다.

텍스트, 검은색, 점수판, 스크린샷이(가) 표시된 사진

자동 생성된 설명



테이블이(가) 표시된 사진

자동 생성된 설명

Airplane\_Type 테이블을 만들어 주었다. 이 테이블 안에서도 attribute들은 모두 not null 속성을 가진다.

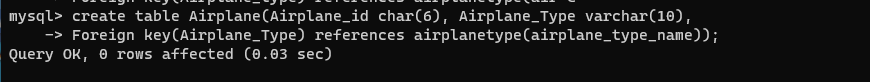
Airplane\_Type이라는 속성이 Max\_seats와 Company를 모두 결정하므로 2NF를 만족하고 이행의 조건을 만족하는 attribute도 없으니 3NF또한 만족한다.



Flight table을 만들어주었다. Primary key는 flight\_number이고 이 테이블의 경우 2nf와 3nf를 모두 만족한다.

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명



Airplane의 테이블을 만들어주었다.

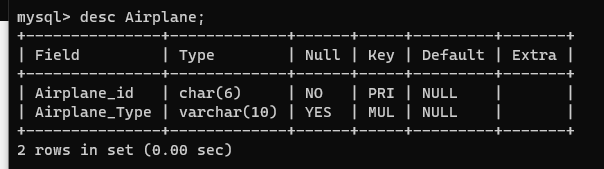
텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

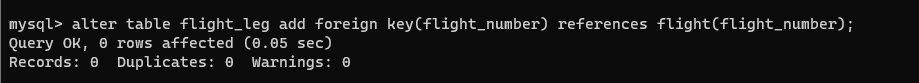
자동 생성된 설명

원래 airplane의 테이블은 다음과 같았으나 airplane\_type이 Total\_number\_of\_seats를 결정하는 이행적 fd가 존재해서 Total\_number\_of\_seats를 없애고 Airplane\_type만 존재하게 납뒀다. Airplane\_type 테이블에 이미 max\_seats라는 속성이 존재하기 때문이다.



텍스트이(가) 표시된 사진

자동 생성된 설명



Flight\_leg 테이블에서는 flight\_number와 leg\_number를 primary key로 가진다. 또한 flight\_number는 flight테이블의 기본 키인 flight\_number를 참조한다. 또한 departure\_airport\_code 와 arrival\_airport\_code또한 airport의 airport\_code 를 참조한다. 위의 테이블은 flight\_number와 leg\_number로 모든 것이 결정되기 때문에 2NF를 만족하고 이행의 조건이 arrival airport code와 departure\_airport\_code, arrival\_time을 알면 departure\_Time이 유도 되어서 나온다는 fd가 있긴 하지만 그냥 입력해주면 되는 문제라 넘기기로 하였다.

텍스트, 점수판, 명판이(가) 표시된 사진

자동 생성된 설명

Leg\_instances table

텍스트이(가) 표시된 사진

자동 생성된 설명

Leg\_instances table을 만들어주었다.

테이블이(가) 표시된 사진

자동 생성된 설명

원래 스키마는 다음과 같은데 flight\_number와 leg\_number가 departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time을 결정해버리는 FD가 존재하므로 2NF를 만족하지 않아 다음 네 속성을 지워주고 Flight\_number, Leg\_number, Date, Number\_of\_available\_seats, Airplane\_id 만 남겨주었다.

그 후 Fare 테이블이 필요 없고 leg\_instance에 종속되도록 추가해주었다.

Flight\_number, leg\_number, Date, Airplane\_id는 항공 구간 하나를 결정한다. Flight은 전체적인 하나의 비행이고 하나의 비행 안에 각각의 leg\_instance가 존재하는데

서울에서 부산가는 버스 안에 경유지가 춘천이 있다면 춘천에서 부산 가는 사람도 같은 버스를 탈 수 있기 때문에 다르게 취급하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

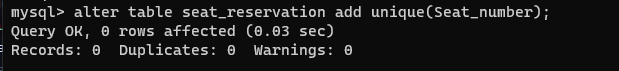
자동 생성된 설명

Fair은 Fare테이블의 값을 의미하는데 결국 leg\_instance에 종속되는 attribute이므로 다른 속성들은 제외하고 leg\_instance table에 column을 추가해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Seat\_reservation 테이블을 만들어주었다.



Seat\_number 는 unique해야하기 때문에 unique속성을 추가해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Desc 명령어를 통해 본 seat\_reservation 테이블이다.

원래 후보키로 (flight\_number, leg\_number, Date, Airplane\_id, Seat\_number)와 (customer\_name, Customer\_phone) 두 가지를 생각했으나 한 사람이 여러 개의 예약을 할 수도 있기 때문에 전자를 선택했다.

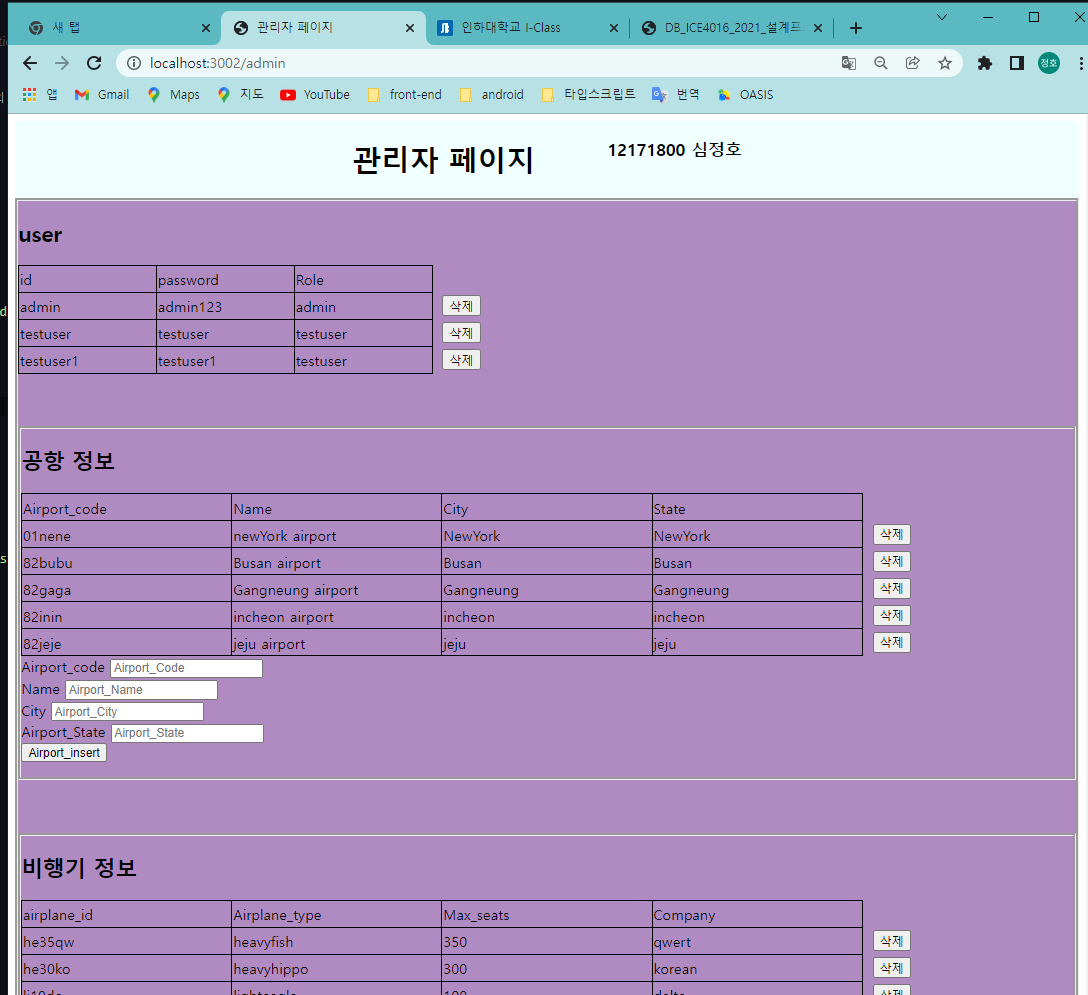
텍스트이(가) 표시된 사진

자동 생성된 설명

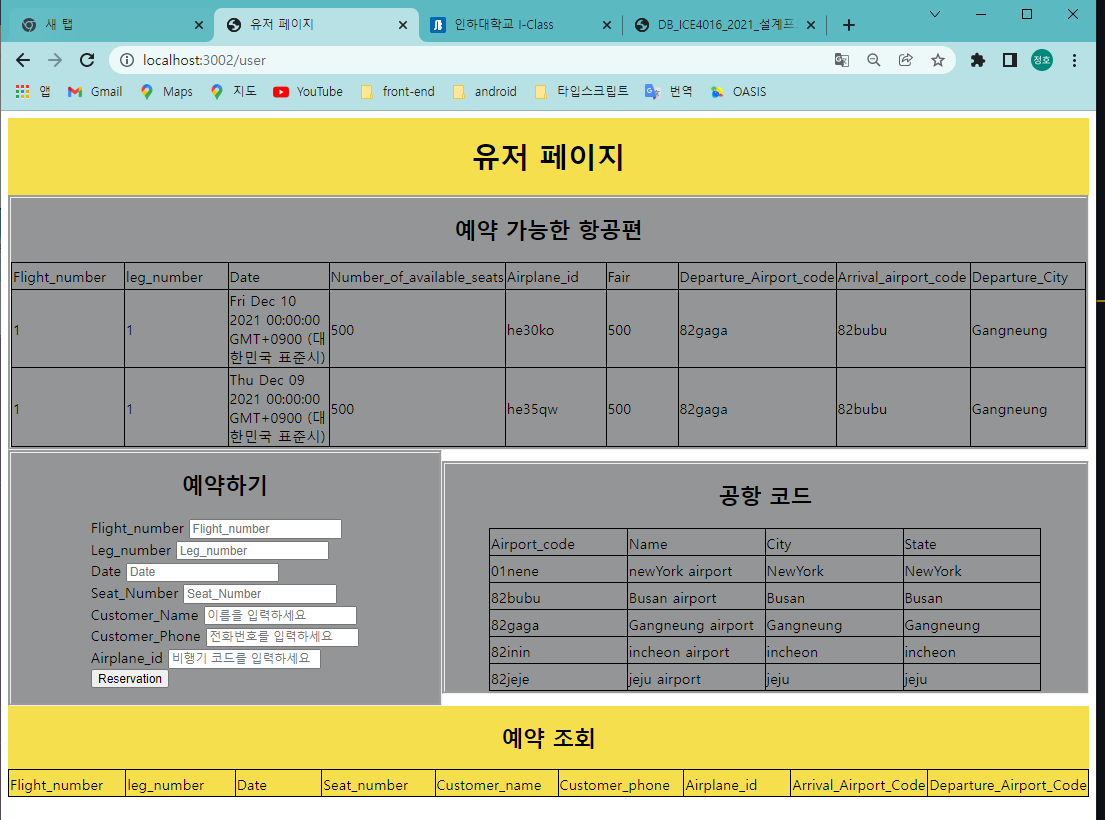
다음과 같이 미리 값을 추가해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명



관리자 페이지의 모습



User 페이지의 모습

텍스트이(가) 표시된 사진

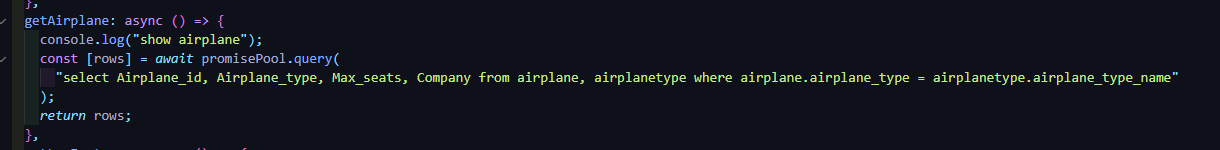
자동 생성된 설명

1. 공항 정보 조회

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

Admin.js에서 getAirplane()함수를 get방식을 통해 불러준다.



getAirplane의 함수는 다음과 같다.

1. 공항 정보 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

우선 admin.hbs에 form태그를 이용해 form 을 만들어준다. Form을 제출할 url은 /insertAirport이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

insertAirport.js의 모습이다. Data를 받아서 sql의 insertAirport함수를 실행시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

insertAirport함수는 다음과 같이 transaction을 수행해주었다.

테이블이(가) 표시된 사진

자동 생성된 설명

다음에서 값을 입력하면

테이블이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

다음과 같이 추가가 된다.

1. 공항 정보 삭제

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 deleteAirport Router를 설정해 준 후 post 요청을 통해 deleteAirport()함수를 실행해준다.

텍스트, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

deleteAirport()함수는 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

만약 저 삭제 버튼을 누르면

테이블이(가) 표시된 사진

자동 생성된 설명

다음과 같이 삭제가 진행된다.

1. 항공기 정보 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 insertAirplaneRouter를 하나 만든 후 post요청을 통해 insert\_Airplane()함수를 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Insert\_Airplane()함수는 다음과 같은 transaction과정을 거친다 먼저 AirplaneType에 입력한 값이 존재하지 않는다면 rollback한다. 존재한다면 insert를 한 후 commit한다.

테이블이(가) 표시된 사진

자동 생성된 설명

다음과 같이 입력해주면

테이블이(가) 표시된 사진

자동 생성된 설명

정상적으로 입력이 된다.

1. 비행기 정보 삭제

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 deleteAirplaneRouter를 만들어 준 후 deleteAirplane함수를 post 방식으로 호출해주었다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

deleteAirplane()함수의 모습은 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

삭제 버튼을 누르면

테이블이(가) 표시된 사진

자동 생성된 설명

정상적으로 삭제가 된다.

1. 항공 구간 정보 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 insertLeg\_InstanceRouter를 만들어 준 후 insertLegInstance()함수를 호출해주고 data를 전달해주었다.

insertLegInstance: async (data) => {

    const sql = `insert into leg\_instance select '${data.Flight\_number}', '${data.leg\_number}', '${data.Date}', '${data.Number\_Of\_Available\_Seats}', '${data.Airplane\_id}', '${data.Fair}' from dual where exists (select Flight\_number, leg\_number, Airplane\_id from flight\_leg a, airplane b where a.flight\_number = '${data.Flight\_number}' and a.leg\_number = '${data.leg\_number}' and b.Airplane\_id = '${data.Airplane\_id}')`;

*await* promisePool.query(sql);

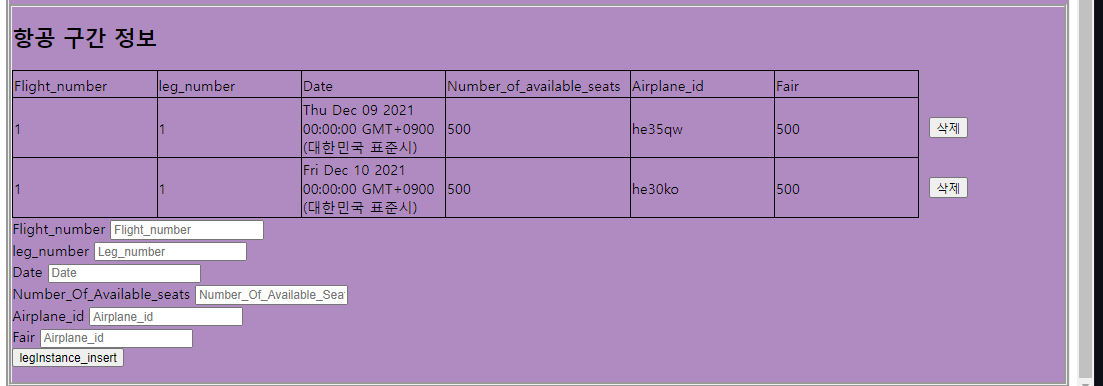
  },

Insert LegInstance의 함수는 다음과 같다. Insert into 와 select를 합친 sql 구문으로 별도의 확인 없이 바로 넣어주었다. 사전의 체크는 이미 foreign key와 primary key로 이루어진 제약조건들로 인해 체크가 되므로 바로 삽입을 해도 이상이 없다.

텍스트, 장치, 안테나, 게이지이(가) 표시된 사진

자동 생성된 설명

다음과 같이 정보를 입력해주면



정상적으로 입력이 된다.

1. 항공 구간 정보 삭제

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 deleteLegInstanceRouter를 하나 만들어 deleteLegInstance()함수를 호출해준다.

deleteLegInstance: async (data) => {

    console.log("deletesql.Airplane\_id: ", data.Airplane\_id);

    console.log("deleteleginstance flight\_number: ", data.Flight\_number);

    console.log("deleteLegInstance leg number : ", data.Leg\_number);

    console.log("나 leg instance deletor야 ");

    const sql = `delete from leg\_instance where Airplane\_id ='${data.Airplane\_id}' and Flight\_number = ${data.Flight\_number} and Leg\_number = ${data.Leg\_number}`; *//*

*await* promisePool.query(sql);

  },

deleteLegInstance의 모슴은 다음과 같다. 여기서도 sql문 한 번에 처리가 되기 때문에 딱히 트랜잭션 처리는 안해주었다.

테이블이(가) 표시된 사진

자동 생성된 설명

유저 페이지에서는 다음과 같이 예약 가능한 항공편이 뜬다. 유저는 이 테이블에서 예약 가능한 항공편을 확인하고 자신이 예약하고자 하는 항공편을 예약하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

User는 render될 때 다음과 같은 정보를 받아 오는데 이 때 예약 가능한 항공편에 해당하는 정보는 Av변수에 담긴 getAvailableInstance()함수에서 불러온다.

다음 함수를 확인해보면 다음과 같다.

getAvailableInstance: async () => {

    console.log("getAvailabeInstance!!");

    const [rows] = *await* promisePool.query(

      `select a.Flight\_number, a.leg\_number, a.Date, a.Number\_of\_available\_seats, a.Airplane\_id, a.Fair, b.departure\_airport\_code, b.arrival\_airport\_code, c.city as departureCity from leg\_instance a, flight\_leg b, Airport c where a.Number\_of\_available\_seats > 0 and c.Airport\_code = b.departure\_airport\_code;`

    );

*return* rows;

  },

다음 함수 도한 sql문 하나로 해결이 되기 때문에 트랜잭션 처리는 하지 않았다.

1. 예약하기

텍스트이(가) 표시된 사진

자동 생성된 설명

예약하기는 다음과 같은 form에서 이루어진다.

테이블이(가) 표시된 사진

자동 생성된 설명

사용자가 공항코드를 잘 모를 수 있으므로 공항 코드에 대한 테이블 정보를 db에서 불러와 옆에 출력하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

ReservationRouter의 모습은 다음과 같다. 해당 요청에서 Reservation함수를 호출해준다.

 Reservation: async (data) => {

    console.log("나Reservation!이야!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

    console.log("Resetvationdata입니다. ", data);

*try* {

      console.log("try 문입니다.");

      const sql0 = `set autocommit = 0`;

*await* promisePool.query(sql0);

      const sql1 = `start transaction`;

*await* promisePool.query(sql1);

      console.log("select 전입니더");

      const sql2 = `select Flight\_number, Leg\_number, Airplane\_id from leg\_instance where flight\_number = ${data.Flight\_number} and Leg\_number = ${data.Leg\_number} and Airplane\_id = '${data.Airplane\_id}'`;

      console.log("select까지는 됩니더");

      const sql3 = `insert into seat\_reservation values ( ${data.Flight\_number}, ${data.Leg\_number}, '${data.Date}', ${data.Seat\_Number}, '${data.Customer\_Name}', '${data.Customer\_Phone}', '${data.Airplane\_id}')`;

*//await promisePool.query(sql2);*

      const updatesql = `update leg\_instance set Number\_of\_available\_seats = Number\_of\_available\_seats - 1 where Flight\_number = ${data.Flight\_number} and leg\_number = ${data.Leg\_number} and Airplane\_id = '${data.Airplane\_id}'`;

      let Result1 = *await* promisePool.query(sql2);

      console.log("select 됩니다.");

      console.log("Result1 결과 !", Result1[0]);

*if* (Result1[0].length < 1) {

        console.log("Reservation Failed");

        const sql4 = `rollback`;

*await* promisePool.query(sql4);

*throw* new Error(`there has no leg\_instance`);

      } *else* {

        console.log("Reservation Success");

        const sql5 = `commit`;

*await* promisePool.query(sql3);

*await* promisePool.query(updatesql); *// number of available seats 1 줄이기*

*await* promisePool.query(sql5);

        console.log("insert 됩니다. ");

      }

    } *catch* (err) {

      const sql6 = `rollback`;

      console.log("Reservation에서 error 발생1");

*await* promisePool.query(sql6);

    } *finally* {

      const sql7 = `set autocommit = 1`;

*await* promisePool.query(sql7);

    }

  },

Reservation함수는 다음과 같이 트랜잭션 처리를 해주었다. 우선 autocommit을 바꾸고 start transaction을 해준 후 입력 받은 정보를 이용해 seat\_reservation테이블에 insert를 해주고 만약 insert가 되었다면 leg\_instance테이블의 number\_of\_available\_seats속성의 값을 -1 시켜준다. 그 후 정상적으로 처리가 되었다면 commit error가 발생했다면 rollback을 시켜준다.

테이블이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

다음과 같이 해당 항공 편의 number\_of\_available\_seats가 줄어들면서 예약이 추가되는 것을 확인할 수 있다.

1. 예약 취소

텍스트이(가) 표시된 사진

자동 생성된 설명

예약 취소는 다음과 같이 deleteReservationRouter를 만들어 준 후 deleteSeatReservation함수를 호출해준다.

정보는 모두 버튼 안에 들어있어야 하므로 한 번에 넣어준 후 javascript의 substring함수를 이용해서 추출해주었다.

deleteSeatReservation: async (data) => {

    console.log("deleteReservation");

    console.log("나 DeleteReservation이야\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

    console.log("DeleteReservation입니다. ", data);

*//console.log("data.Flight\_number", data.Flight\_number);*

*try* {

      console.log("try문입니더 delete");

      const sql0 = `set autocommit = 0`;

*await* promisePool.query(sql0);

      const transactionsql = `start transaction`;

*await* promisePool.query(transactionsql);

      console.log("트랜잭션 까지 됩니더 delete");

      const sql2 = `delete from seat\_reservation where flight\_number = ${data.Flight\_number} and leg\_number = ${data.Leg\_number} and Seat\_number = ${data.Seat\_Number} and Airplane\_id = '${data.Airplane\_id}'`;

*await* promisePool.query(sql2);

      console.log("delete가 안됩니더");

      const updatesql = `update leg\_instance set Number\_of\_available\_seats = Number\_of\_available\_seats + 1 where Flight\_number = ${data.Flight\_number} and leg\_number = ${data.Leg\_number} and Airplane\_id = '${data.Airplane\_id}'`;

*await* promisePool.query(updatesql);

      console.log("delete update입니더");

      const commit = `commit`;

*await* promisePool.query(commit);

    } *catch* (err) {

      const Rollback = `rollback`;

*await* promisePool.query(Rollback);

    } *finally* {

      const Autocommit = `set autocommit = 1`;

*await* promisePool.query(Autocommit);

    }

  },

DeleteSeatReservation()함수는 다음과 같이 트랜잭션을 처리해준다. 우선 autocommit을 꺼주고 start transaction을 해준다. 그 후 seat\_reservation 테이블에서 입력받은 정보를 이용해 delete를 해주고 취소가 되었다면 leg\_instance테이블의 number\_of\_available\_seats 속성의 값을 1+ 해준다. 그 후 commit해준다 만약 에러가 중간에 발생하였다면 rollback한다.



예약 취소 버튼을 누르게 되면

테이블이(가) 표시된 사진

자동 생성된 설명

다음과 같이 number\_of\_available\_Seats의 숫자가 늘어나면서 예약이 취소되는 것을 확인할 수 있다.

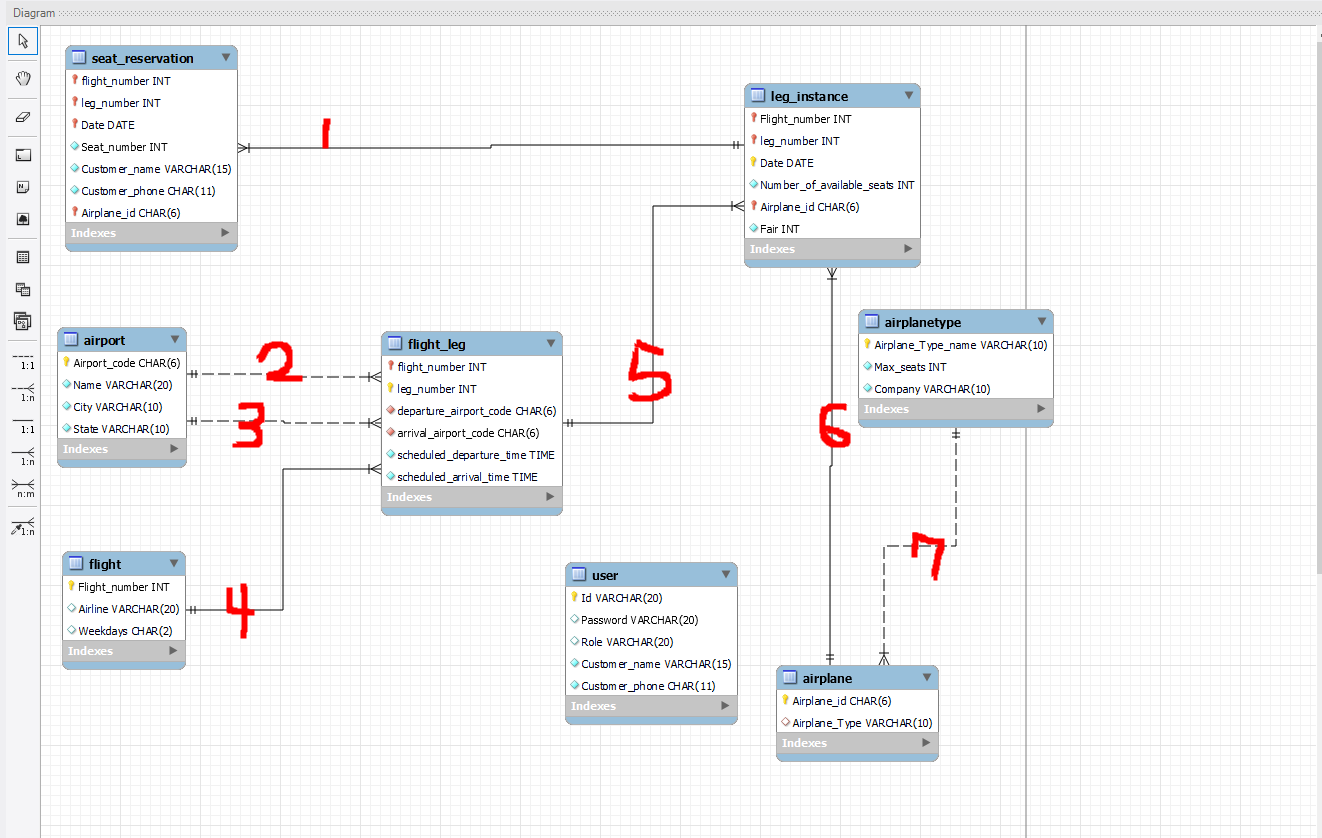
1. ERD

나는 cmd창에서 script를 직접 입력하여 데이터베이스를 만들었기 때문에 erd를 그리기 위해 mysql workbench를 이용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Reverse Engineering을 이용하면 존재하던 데이터 베이스를 erd로 변환할 수 있다.

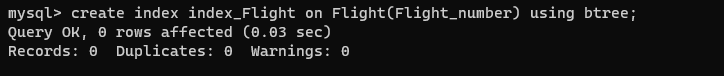


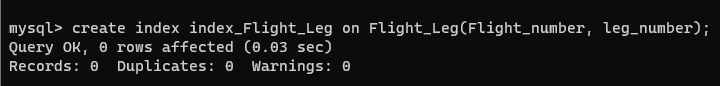
생성된 erd는 다음과 같다.

1. 의 제약 조건은 leg\_instance와 seat\_reservation의 관계이다. 이 때 seat\_reservation은 leg\_instance를 예약하는 것이므로 무조건 식별 관계를 가지고 하나의 leg\_instance에는 여러 개의 예약이 있을 수 있으므로 1:N의 관계를 갖는다.
2. 의 제약 조건은 Flight\_leg가 Airport를 참조하는 비식별관계이다. 이 때 Flight\_leg에는 도착 공항과 출발 공항 attribute가 존재하는데 위 관계는 airport의 primary키를 flight\_leg에서는 primary key로 사용하고 있지 않기 때문에 비 식별관계이다. 또한 하나의 airport는 여러 flight\_leg의 도착 혹은 출발 공항이 될 수 있으므로 1:N관계를 갖는다.
3. 2번과 동일하다.
4. 의 제약 조건은 flight과 flight\_leg의 제약조건이다. 이 때 flight\_leg는 flight과 식별 적인 관계로 flight이 존재해야지만 그 flight에 해당하는 flight\_leg가 존재할 수 있다. 따라서 식별 관계이고 하나의 flight에는 여러 개의 flight\_leg가 존재할 수 있기 때문에 1:N의 관계를 갖는다.
5. 의 제약 조건은 flight\_leg와 leg\_instance의 관계로 leg\_instance는 실제로 존재하는 flight\_leg를 의미한다. 따라서 leg\_instance는 flight\_leg가 존재해야지만 존재할 수 있으므로 식별 관계에 해당하고 하나의 flight\_leg당 여러 개의 실제 leg\_instance들이 존재할 수 있다. 날짜가 다르던지 운행하는 비행기가 다르던지 leg\_instance의 primary key에 따라 달라질 수 있으므로 여러 개의 instance를 가질 수 있다. 따라서 1:N관계이다.
6. 의 제약 조건은 airplane과 leg\_instance의 관계이다. 이 때 실제로 행해지는 leg\_instance는 실제로 존재하는 비행기가 있어야 행해질 수 있기 때문에 식별관계에 해당하고 한 대의 비행기는 여러 번의 실제 비행을 할 수 있기 때문에 1:N 관계에 해당한다.
7. 의 제약 조건은 airplanetype과 airplane의 제약조건이다. 하나의 airplaneType은 여러 개의 비행기 들이 참조할 수 있기 때문에 1:N의 관계를 갖는다. 아직 type을 정의하지 않은 상태로 비행기가 추가될 수 있기 때문에 airplanetype을 primary key로 설정해주지 않았고 따라서 비 식별관계에 해당한다.

Index 설정







텍스트이(가) 표시된 사진

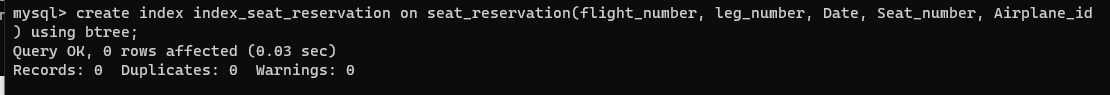
자동 생성된 설명

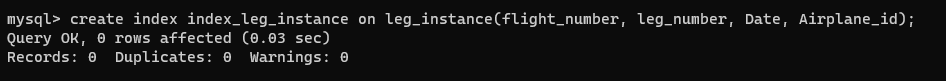
텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명





다음과 같이 각 테이블의 primary key를 이용해서 index를 만들어 주었다. 나의 데이터베이스 엔진은 inno database이기 때문에 btree를 사용해서 index를 만들어주었다.

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema final\_project

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema final\_project

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `final\_project` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `final\_project` ;

-- -----------------------------------------------------

-- Table `final\_project`.`airplanetype`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`airplanetype` (

`Airplane\_Type\_name` VARCHAR(10) NOT NULL,

`Max\_seats` INT NOT NULL,

`Company` VARCHAR(10) NOT NULL,

PRIMARY KEY (`Airplane\_Type\_name`),

INDEX `index\_AirplaneType` (`Airplane\_Type\_name` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`airplane`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`airplane` (

`Airplane\_id` CHAR(6) NOT NULL,

`Airplane\_Type` VARCHAR(10) NULL DEFAULT NULL,

PRIMARY KEY (`Airplane\_id`),

INDEX `Airplane\_Type` (`Airplane\_Type` ASC) VISIBLE,

INDEX `index\_airplane` USING BTREE (`Airplane\_id`) VISIBLE,

CONSTRAINT `airplane\_ibfk\_1`

FOREIGN KEY (`Airplane\_Type`)

REFERENCES `final\_project`.`airplanetype` (`Airplane\_Type\_name`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`airport`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`airport` (

`Airport\_code` CHAR(6) NOT NULL,

`Name` VARCHAR(20) NOT NULL,

`City` VARCHAR(10) NOT NULL,

`State` VARCHAR(10) NOT NULL,

PRIMARY KEY (`Airport\_code`),

INDEX `index\_Airport` (`Airport\_code` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`flight`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`flight` (

`Flight\_number` INT NOT NULL,

`Airline` VARCHAR(20) NULL DEFAULT NULL,

`Weekdays` CHAR(2) NULL DEFAULT NULL,

PRIMARY KEY (`Flight\_number`),

INDEX `index\_Flight` USING BTREE (`Flight\_number`) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`flight\_leg`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`flight\_leg` (

`flight\_number` INT NOT NULL,

`leg\_number` INT NOT NULL,

`departure\_airport\_code` CHAR(6) NOT NULL,

`arrival\_airport\_code` CHAR(6) NOT NULL,

`scheduled\_departure\_time` TIME NOT NULL,

`scheduled\_arrival\_time` TIME NOT NULL,

PRIMARY KEY (`flight\_number`, `leg\_number`),

INDEX `departure\_airport\_code` (`departure\_airport\_code` ASC) VISIBLE,

INDEX `arrival\_airport\_code` (`arrival\_airport\_code` ASC) VISIBLE,

INDEX `index\_Flight\_Leg` (`flight\_number` ASC, `leg\_number` ASC) VISIBLE,

CONSTRAINT `flight\_leg\_ibfk\_1`

FOREIGN KEY (`departure\_airport\_code`)

REFERENCES `final\_project`.`airport` (`Airport\_code`),

CONSTRAINT `flight\_leg\_ibfk\_2`

FOREIGN KEY (`arrival\_airport\_code`)

REFERENCES `final\_project`.`airport` (`Airport\_code`),

CONSTRAINT `flight\_leg\_ibfk\_3`

FOREIGN KEY (`flight\_number`)

REFERENCES `final\_project`.`flight` (`Flight\_number`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`fare`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`fare` (

`Flight\_number` INT NOT NULL,

`leg\_number` INT NOT NULL,

`Fare\_code` INT NOT NULL,

`amount` INT NOT NULL,

`restrictions` TINYINT(1) NULL DEFAULT NULL,

PRIMARY KEY (`Flight\_number`, `leg\_number`, `Fare\_code`),

CONSTRAINT `fare\_ibfk\_1`

FOREIGN KEY (`Flight\_number`)

REFERENCES `final\_project`.`flight\_leg` (`flight\_number`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`leg\_instance`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`leg\_instance` (

`Flight\_number` INT NOT NULL,

`leg\_number` INT NOT NULL,

`Date` DATE NOT NULL,

`Number\_of\_available\_seats` INT NOT NULL,

`Airplane\_id` CHAR(6) NOT NULL,

`Fair` INT NOT NULL,

PRIMARY KEY (`Flight\_number`, `leg\_number`, `Date`, `Airplane\_id`),

INDEX `Airplane\_id` (`Airplane\_id` ASC) VISIBLE,

INDEX `index\_leg\_instance` (`Flight\_number` ASC, `leg\_number` ASC, `Date` ASC, `Airplane\_id` ASC) VISIBLE,

CONSTRAINT `leg\_instance\_ibfk\_1`

FOREIGN KEY (`Flight\_number` , `leg\_number`)

REFERENCES `final\_project`.`flight\_leg` (`flight\_number` , `leg\_number`),

CONSTRAINT `leg\_instance\_ibfk\_2`

FOREIGN KEY (`Airplane\_id`)

REFERENCES `final\_project`.`airplane` (`Airplane\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`seat\_reservation`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`seat\_reservation` (

`flight\_number` INT NOT NULL,

`leg\_number` INT NOT NULL,

`Date` DATE NOT NULL,

`Seat\_number` INT NOT NULL,

`Customer\_name` VARCHAR(15) NOT NULL,

`Customer\_phone` CHAR(11) NOT NULL,

`Airplane\_id` CHAR(6) NOT NULL,

PRIMARY KEY (`flight\_number`, `leg\_number`, `Date`, `Airplane\_id`),

UNIQUE INDEX `Seat\_number` (`Seat\_number` ASC) VISIBLE,

INDEX `index\_seat\_reservation` USING BTREE (`flight\_number`, `leg\_number`, `Date`, `Seat\_number`, `Airplane\_id`) VISIBLE,

CONSTRAINT `seat\_reservation\_ibfk\_1`

FOREIGN KEY (`flight\_number` , `leg\_number` , `Date` , `Airplane\_id`)

REFERENCES `final\_project`.`leg\_instance` (`Flight\_number` , `leg\_number` , `Date` , `Airplane\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `final\_project`.`user`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `final\_project`.`user` (

`Id` VARCHAR(20) NOT NULL,

`Password` VARCHAR(20) NULL DEFAULT NULL,

`Role` VARCHAR(20) NULL DEFAULT NULL,

`Customer\_name` VARCHAR(15) NOT NULL,

`Customer\_phone` CHAR(11) NOT NULL,

PRIMARY KEY (`Id`),

INDEX `index\_user` USING BTREE (`Id`) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

다음은 erd를 forword engineering해서 만든 script문이다.

시간이 좀 더 많았다면 jwt와 같은 것을 사용해서 로그인 기능도 제대로 구현하고 React와 같은 front library를 사용해 axios라이브러리로 통신하는 것도 해보고 싶었지만 많은 시험들과 플젝에 하지 못하였다. 이번 과제를 제출하고 난 후에 더 업그레이드를 시켜볼 생각이다. 하면서 express에 대해 많이 배운 것 같고 정말 유익한 시험이었다. Sql에 대해 흥미가 생겨 dba자격증 이나 sqld와 같이 database와 관련된 자격증 또한 준비해 볼 생각이다.