2019-1 Database

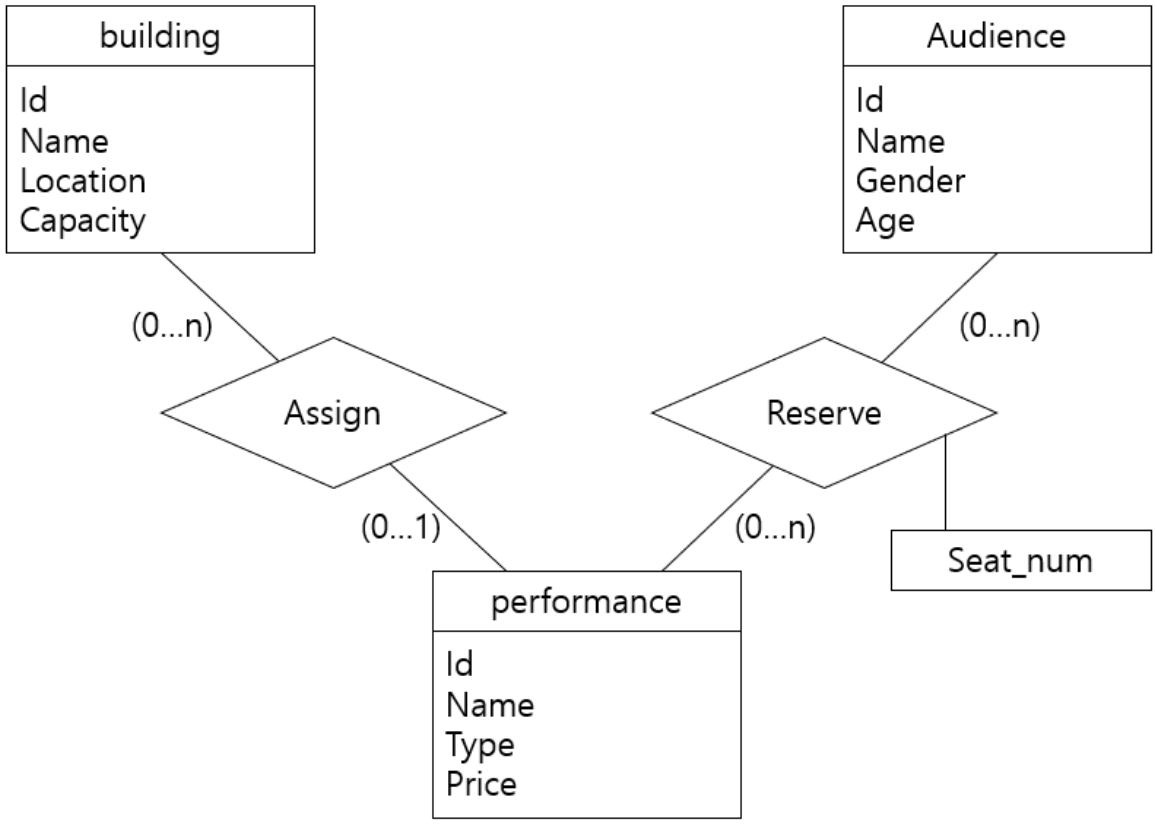
PRJ2

2017-17450

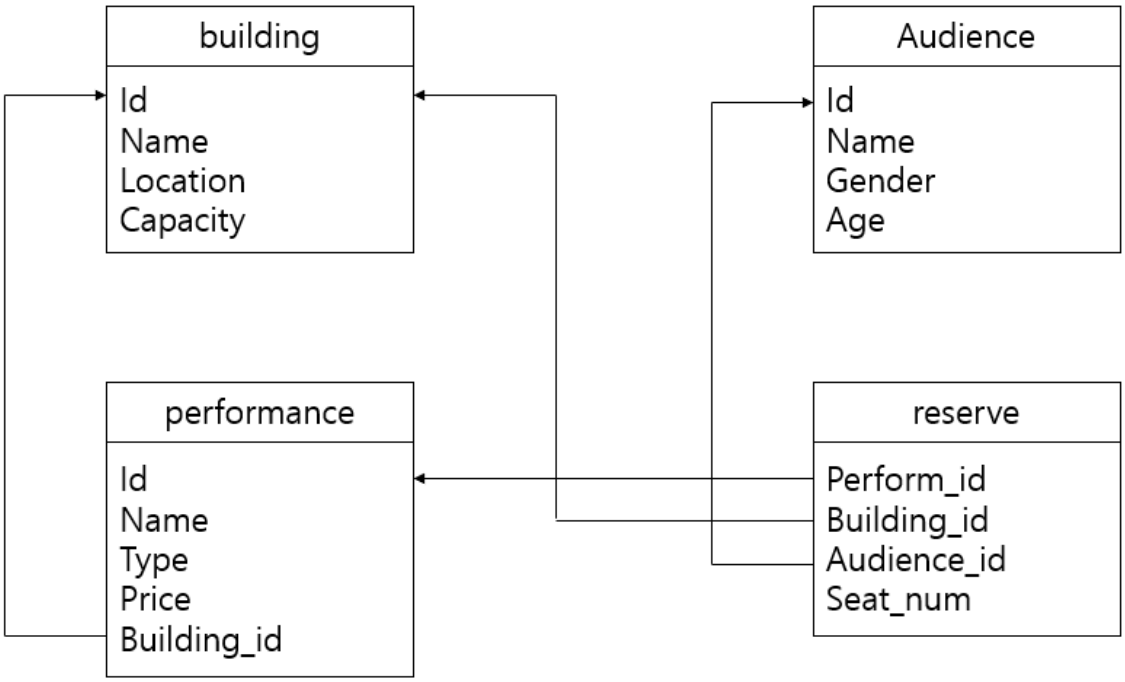
컴퓨터공학부 백근영

1. 핵심 모듈 및 알고리즘

이번 과제는 공연 예약 어플리케이션을 만드는 것이었는데, 제시된 스펙에 따라 구상해 본 ER-diagram은 다음과 같다.



Relational schema diagram을 design할 때, assign은 many to one relation이기 때문에 performance에 building_id라는 foreign key를 둠으로써 표현하기로 하였고, reserve는 many to many relation이므로 이 relation에 해당하는 table을 따로 하나 생성하기로 하였다. 이러한 생각 하에 구상한 relational schema diagram의 형태는 아래와 같다.



전체적으로 크게 복잡한 부분 없이 간단한 relational schema diagram이 완성되었다. Reserve table을 만들면서 고려했던 한 가지 사항은 building_id를 꼭 넣어야 할까 하는 것이었다. Building_id를 넣은 이유는 제시된 요구사항 중 14번 command를 쉽게 처리하기 위함이다. 14번 command를 처리하기 위해서는 공연이 배정된 공연장의 정원 수를 알아야 하는데, reserve 테이블에 building_id를 넣지 않는다면 reserve와 performance 테이블을 join하고 이를 building 테이블과 다시 한 번 join해서 capacity를 가져오는 등의 노력이 필요하다. 반면 building_id가 있으면 한 번의 join으로 정원 수를 받아올 수 있기 때문에 이를 따로 하나의 attribute로 두었다.

위 diagram을 바탕으로 아래와 같은 mysql 커맨드를 이용해 4개의 table을 생성했다.

```
create table building(
    id int not null auto_increment,
    name varchar(200),
    location varchar(200),
    capacity int check (capacity >= 1),
    primary key(id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin auto_increment=1;
```

```
create table performance(
    id int not null auto_increment,
    name varchar(200),
    type varchar(200),
    price int check (price >= 1),
    building_id int,
    primary key(id),
    foreign key(building_id) references building(id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin auto_increment=1;
```

```
create table audience(
    id int not null auto_increment,
    name varchar(200),
    gender char(1) check (gender = 'M' or gender = 'F'),
    age int check (age >= 1),
    primary key(id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin auto_increment=1;
```

```
create table reserve(
    perform_id int not null,
    building_id int not null,
    audience_id int not null,
    seat_num int not null,
    foreign key(perform_id) references performance(id) on delete cascade,
    foreign key(building_id) references building(id) on delete cascade,
    foreign key(audience_id) references audience(id) on delete cascade
)ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

Reserve table의 경우, performance나 audience가 삭제되면 그와 관련된 예매 정보가 함께 삭제되어야 하기 때문에 on delete cascade 옵션을 설정해 주었다. Capacity, age, gender 같이 값에 제약이 있는 것들은 check constraint를 명시해 주었다.

2. 구현 내용

```
while True:
    commandNum = int(input('Select your action: '))

    # command number에 따라 적절한 함수 실행 #
    if commandNum == 1:
        print_all_buildings()
    elif commandNum == 2:
        print_all_performances()
    elif commandNum == 3:
        print_all_audiences()
    elif commandNum == 4:
        insert_new_building()
    elif commandNum == 5:
        remove_building()
    elif commandNum == 6:
        insert_new_performance()
    elif commandNum == 7:
        remove_performance()
    elif commandNum == 8:
        insert_new_audience()
    elif commandNum == 9:
        remove_audience()
    elif commandNum == 10:
        assign_performance()
    elif commandNum == 11:
        book_performance()
    elif commandNum == 12:
        print_assigned_performance()
    elif commandNum == 13:
        print_book_audience()
    elif commandNum == 14:
        print_booking_status()
    elif commandNum == 15:
        connection.close()
```

python file의 작동 방식은 왼쪽과 같다. 먼저 사용자로부터 몇 번 커맨드를 실행할지 입력 받고, 그 숫자에 따라 적절한 함수를 실행한다. 가령 1,2,3번 등 어떤 테이블의 레코드를 출력하는 커맨드는 알맞은 테이블과 조건에 대하여 select query를 실행해 결과를 출력하도록 하였고, 4,6,8번 등 새로운 레코드를 삽입하는 커맨드는 insert query를, 5,7,9번 등 레코드를 제거하는 커맨드는 delete query를 실행해 커맨드가 들어올 때마다 데이터베이스의 내용을 조작하도록 설계하였다. 대부분의 함수들은 query만 적절히 설정해준다면 꽤 간단히 작성을 끝낼 수 있었는데, 관객이 공연을 예매하는 10번 커맨드는 특히 신경 쓸 부분이 많아서 이 커맨드에 대한 구현 방식을 구체적으로 설명하려 한다.

10번 커맨드를 처리하는 과정은 다음과 같다.

1) 해당 공연이 공연장에 배정되어 있는지 검사

   본인이 설계했던 performance table에는 building_id가 foreign key로 존재하기 때문에 쉽게 검사가 가능하다.

2) 공연장의 정원 수를 초과하는 좌석번호로 예약을 시도하는지 검사

   1) 에서 받아온 building_id로 building table에 접근해 capacity 값을 받아온 후, 이 값보다 큰 값이 좌석번호 리스트에 존재하는지 검사한다.

3) 이미 예약된 좌석으로 예약을 시도하는지 검사

   입력받은 performance_id로 reserve table에 접근해 현재 이미 예약되어 있는 좌석들의 리스트를 받아오고, 이를 현재 예약을 시도하고 있는 좌석번호 리스트와 비교한다.

4) 1,2,3의 검사를 차례로 모두 완료했다면, 관객의 나이를 통해 총 가격 계산

   (가격 * (1 – 할인율) * 좌석 수) 를 계산하는데, 소수점 첫째 자리에서 반올림해야 하므로 이 값에 0.5를 더하고 int로 형변환을 해준다.

5) Insert query를 통해 레코드 삽입

```
with connection.cursor() as cursor:
    query = "INSERT INTO `reserve` (perform_id, audience_id, building_id, seat_num) VALUES (%s, %s, %s, %s)"
    for seatNum in seatNumList:
        cursor.execute(query, (performID, audienceID, buildingID, int(seatNum)))
        connection.commit()
```

다른 커맨드들도 비슷한 방식으로 처음에 설계했던 relational schema를 잘 고려해 query를 설정해줌으로써 비교적 어렵지 않게 어플리케이션을 완성할 수 있었다.

3. 구현하지 못한 부분

현재 select query를 통해 레코드들을 출력하는 커맨드의 경우, 테이블의 각 attribute간 간격이 30자 혹은 20자로 고정되어 있는데, 조금 더 완벽하게 하기 위해서는 레코드에 담긴 각각의 value의 길이에 따라 칸의 길이를 유동적으로 할 필요가 있어 보인다. 하지만 그렇게 하기 위해서는 필요 이상의 노력이 필요할 것 같고, 이번 과제를 통해 배워야 할 점들은 충분히 배웠다고 판단하여 이 부분에 대한 수정은 하지 않았다.

4. 가정한 것들

이번 과제는 현실 세계에도 정말 있을 법한 공연 예매에 관한 어플리케이션을 만드는 것이었기 때문에, 현실 세계의 제약들을 잘 반영하기 위해 임의로 가정해야 하는 부분들이 존재했다. 테이블 별로 가정한 사항들에 대해 설명하도록 하겠다.

1) Building table

   Building table은 id, name, location, capacity등 4개의 attribute를 가지는데, 이 중 어떤 것을 super key로 두어야 할 지 고민했다. 기나긴 고민 끝에, (name, location, capacity)를 super key로 두어 이 3가지 attribute의 조합은 unique하게 들어올 것이라고 가정하였다. 사실 location을 얼마나 구체적으로 제시해 주느냐에 따라서 (name, location) 만으로도 super key가 될 수 있을 것이라고 기대하고 있는데, location을 seoul과 같이 광범위하게 제시한다면 capacity까지 있어야 super key로서 적당하다고 판단하였다. 이렇게 가정하긴 했지만 test case가 어떻게 들어올 지 예상할 수 없기 때문에 별다른 constraint를 sql 단계에서 명시적으로 설정하지는 않았다.

2) Performance table

   Performance table은 같은 이름의 공연이 여러 번 열릴 수도 있다고 생각하여 id 말고는 어떠한 단일 attribute 혹은 attribute의 조합도 super key가 될 수 없다고 판단하였다. 그리고 6번 커맨드를 이용해 공연을 새로 삽입하는 경우, 이 순간에는 배정받은 공연장이 없으며 레코드를 삽입함과 동시에 공연을 공연장에 배정할 수는 없도록 어플리케이션을 설계하였다.

3) Reserve table

   Reserve table의 경우 공연장에 배정을 받은 공연만이 이 테이블의 레코드로 등록될 수 있기 때문에 perform_id가 정해진다면 그에 따른 building_id 또한 자동으로 정해지게 될 것이고, 따라서 building_id는 non nullable해야 한다고 생각했다. nullable하게 설정해 주었던 performance table의 building_id와는 차이를 보인다.

5. 컴파일 및 실행 방법

Python main.py 입력

6. 느낀 점

현실 세계에 존재할 법한 주제를 가지고 데이터베이스를 관리하는 어플리케이션을 작성해보면서 실제로 데이터베이스가 어떤 식으로 저장되고 활용되는 지에 대해 이론적인 부분에서 벗어나서까지 생각을 해볼 수 있었던 것 같다. 수업시간에 배웠던 ER schema 및 relational schema diagram을 작성하는 방법을 이 과제에 직접 적용해볼 수 있었고, 데이터베이스 설계를 잘하는 것이 어플리케이션을 운영하는 데 있어서 왜 중요한지 실감할 수 있었다. 본인은 데이터베이스 설계를 처음부터 완벽히 하지는 못해서 어플리케이션을 만드는 도중에 중간중간 데이터베이스 구조를 변경했었는데, 앞으로 더욱 더 복잡한 현실 상황에 따른 데이터베이스 시스템을 관리, 운영하게 된다면 처음부터 설계를 잘하는 것이 정말 중요하겠다는 생각이 들었다.