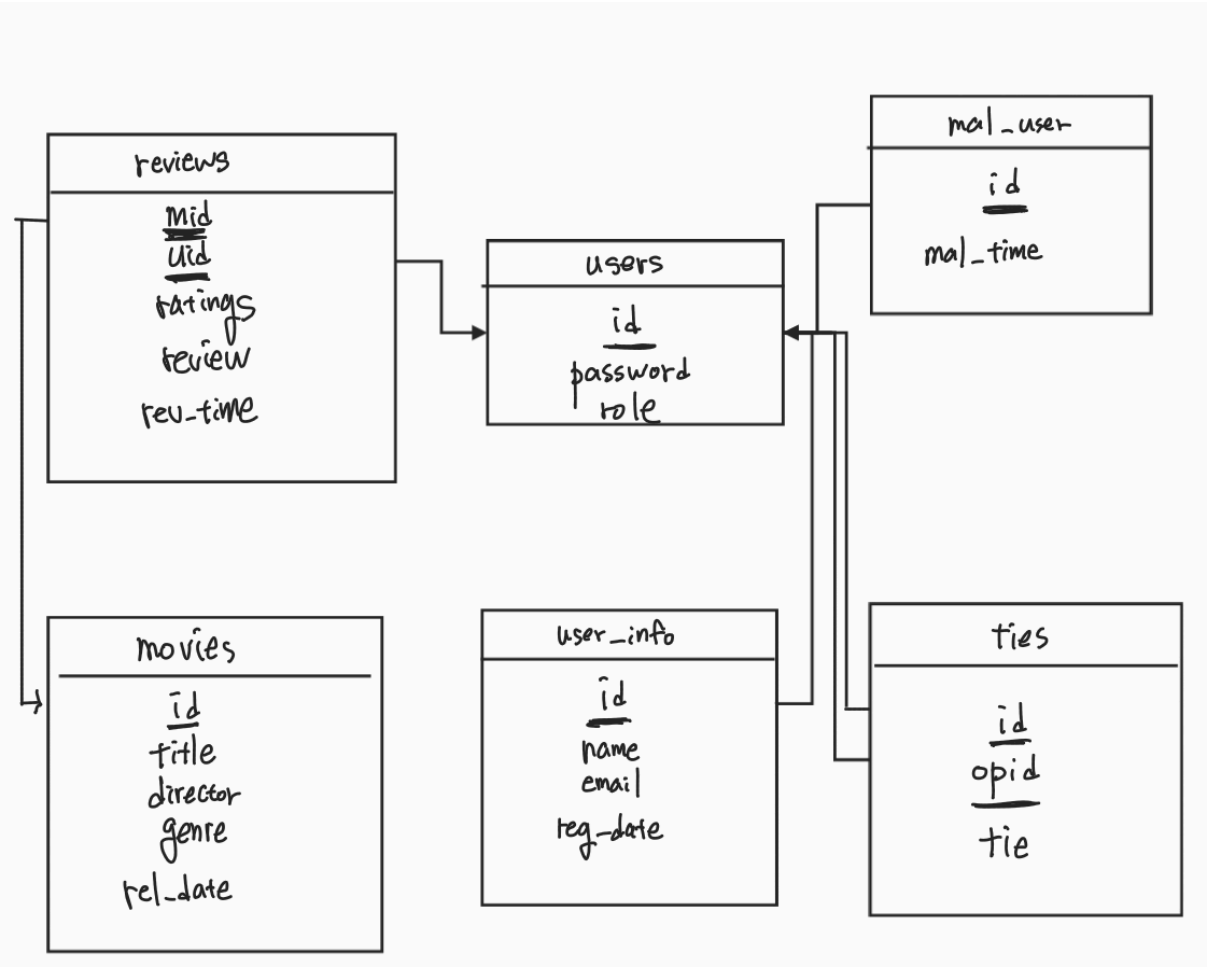
Term Project

✏️ Image of Schema diagram



✏️ Description of file

main/register 함수

`@app.route('/')` 데코레이터는 애플리케이션의 루트 URL (`/`)을 정의한다. `main()` 함수는 루트 URL에 대한 GET요청 처리해, `login.html` 이라는 템플릿을 렌더링하여 반환한다.

`register` 함수는 `/register` 라우트를 정의하며 POST 요청만 허용한다. 그리고 id, password, send에 해당하는 값을 가져와 등록(sign up) 또는 로그인(sign in) 동작을 결정한다.

```
cur.execute("SELECT title, round(avg(ratings),1), director, genre,
 rel_date FROM movies LEFT JOIN reviews ON movies.id = reviews.mid
 GROUP BY title, director, genre, rel_date ORDER BY rel_date DESC;")
movies = cur.fetchall()
```

이 쿼리는 영화 목록과 평균 평점, 감독, 장르, 개봉일을 가져옵니다. 결과는 개봉일을 기준으로 내림차순으로 정렬된다. `LEFT JOIN`은 리뷰가 없는 영화도 모두 포함되도록 하며, `GROUP BY`는 영화 속성을 기준으로 그룹화하여 평균 평점을 계산한다.

```
cur.execute("""
    SELECT ratings, uid, title, review,
    TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS')
    FROM reviews JOIN movies ON movies.id = reviews.mid
    WHERE uid NOT IN (
        SELECT opid FROM ties WHERE id = '{}' and tie = 'mute'
        )
    ORDER BY rev_time DESC;
    """.format(id))
reviews = cur.fetchall()
```

이 쿼리는 평점, 사용자 ID, 영화 제목, 리뷰 텍스트 및 포맷된 리뷰 시간을 포함한 리뷰를 가져옵니다. 현재 사용자가 차단한 사용자의 리뷰는 제외됩니다. `JOIN`은 리뷰와 영화 정보를 결합하고, `WHERE` 절은 차단된 사용자의 리뷰를 제외합니다. 결과는 rev_time 기준으로 내림차순으로 정렬됩니다.

회원가입은 `cur.execute("SELECT id FROM users WHERE id = '{}';".format(id))` 쿼리를 활용한다. users 테이블에 사용자 ID가 이미 존재하는지 확인한다. 사용자 ID가 존재하거나 제공된 ID/비밀번호가 비어 있으면 실패 페이지를 렌더링한다. 사용자 ID가 존재하지 않으면, 새로운 사용자를 users테이블에 삽입하고 로그인 페이지를 렌더링한다.

```
if result or (len(id) < 1 and len(password) < 1 ):
    return render_template("signup_fail.html")
elif not result:
```

```
    cur.execute("INSERT INTO users VALUES('{}', '{}', '{}');".format(id, pass
word, 'user'))
    connect.commit()
    return render_template("login.html")
```

로그인은 제공된 사용자 ID와 비밀번호가 users 테이블의 항목과 일치하는지 확인한다. 일치하는 사용자가 없으면 실패 페이지를 렌더링하고, 일치하는 사용자가 있으면, 영화와 리뷰를 포함한 메인 페이지를 사용자 ID와 함께 렌더링합니다.

```
if not result:
    return render_template("signin_fail.html")
elif result[0] == id and result[1] == password:
    return render_template("main.html", reviews = reviews, movies=movies,
user = id)
```

main.html

로그인 성공 후 main페이지의 html을 렌더링 한다.

`<div id="user-id">ID: {{ user }}</div>` 에서 사용자 ID를 화면의 오른쪽 상단에 표시한다.

```
 <form action="/movie_sort" method="post">
    <input type="hidden" name="id" value={{ user }}>
    <input type="submit" name="send" value="latest">
    <input type="submit" name="send" value="genre">
    <input type="submit" name="send" value="ratings">
</form>

<form action="/review_sort" method="post">
    <input type="hidden" name="id" value={{ user }}>
    <input type="submit" name="send" value="latest">
    <input type="submit" name="send" value="title">
    <input type="submit" name="send" value="followers">
</form>
```

사용자가 선택한 정렬 기준에 따라 영화를 정렬하는 폼 역시 제공한다. 또한 영화 목록, 리뷰 목록을 테이블 형식으로 표시한다. 각 영화 제목은 상세 정보를 확인할 수 있는 폼으로 감싸

져 있다. 영화 테이블과 리뷰 테이블에서 영화 제목을 버튼으로 클릭하면 `/movie_info` URL을 호출하고, 리뷰 테이블에서 역시 리뷰 작성자 이름 버튼을 클릭 하면 `/user_info` 을 호출한다.

```
<form action="/user_info" method="post">
        <input type="hidden" name="id" value={{ user }}>
        <input type="submit" name="target_user_id" value="{{ review[1] }}">
</form>


<form action="/movie_info" method="post">
        <input type="hidden" name="id" value={{ user }}>
        <input type="submit" name="movie" value="{{ review[2] }}">
</form>
```

유저 정보를 클릭하면 현재 로그인한 user에 대한 정보를 id로 넘겨주고, 클릭한 대상을 target_user_id로 넘겨준다. 영화 이름을 클릭하면 현재 로그인한 user 정보를 id로 넘겨주고, 클릭한 영화 이름을 movie로 넘겨준다.

movie_sort 함수

```
cur.execute("""
    SELECT ratings, uid, title, review,
    TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS')
    FROM reviews JOIN movies ON movies.id = reviews.mid
    WHERE uid NOT IN (
        SELECT opid FROM ties WHERE id = '{}' and tie = 'mute'
        )
    ORDER BY rev_time DESC;
    """.format(id))
reviews = cur.fetchall()
```

사용자 ID를 가져오고 제출 버튼의 값을 send로 받아온다. 이후 리뷰 데이터베이스 쿼리를 위와 같이 날려준다. 리뷰 데이터와 영화 정보를 가져오고 현재 사용자가 차단한 사용자의 리뷰를 제외한다.

이후 `send` 값에 따라 영화 데이터를 정렬한다. `latest` 는 개봉일 기준 최신 순, `genre` 는 장르별 정렬, `ratings` 은 평점 기준 내림차순 정렬이다. 각 정렬 기준에 맞는 SQL 쿼리를 실행해 결과를 가져온다. 그리고 마지막으로 main.html을 다시 렌더링한다.

review_sort 함수

로그인한 사용자의 ID를 가져오고, send로 제출 버튼의 값을 가져와 정렬 기준을 정한다. 영화 목록과 평균 평점, 감독, 장르, 개봉일을 가져온다. 결과는 개봉일 기준으로 내림차순 정렬된다.

```
if send == "latest":
        cur.execute("""
            SELECT ratings, uid, title, review,
            TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS')
            FROM reviews JOIN movies ON movies.id = reviews.mid
            WHERE uid NOT IN (
                SELECT opid FROM ties WHERE id = '{}' and tie = 'mute'
                )
            ORDER BY rev_time DESC;
            """.format(id))
        reviews = cur.fetchall()
```

리뷰의 평점, 사용자 ID, 영화 제목, 리뷰 텍스트, 리뷰 시간을 가져온다. 리뷰와 영화 테이블을 결합하고 현재 사용자가 차단한 사용자의 리뷰는 제외한다. 리뷰를 최신순으로 정렬한다.

```
elif send == "title":
        cur.execute("""
            SELECT ratings, uid, title, review,
            TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS')
            FROM reviews JOIN movies ON movies.id = reviews.mid
            WHERE uid NOT IN (
                SELECT opid FROM ties WHERE id = '{}' and tie = 'mute'
                )
            ORDER BY title;
            """.format(id))
        reviews = cur.fetchall()
```

리뷰의 평점, 사용자 ID, 영화 제목, 리뷰 텍스트, 리뷰 시간을 가져온다. 리뷰와 영화 테이블을 결합하여 필요한 정보를 모두 가져오고, 현재 사용자가 차단한 사용자의 리뷰는 제외한다. 리뷰를 영화 제목순으로 정렬한다.

```
elif send == "followers":
        cur.execute("""
            SELECT ratings, uid, title, review,
            TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS'), follower_count
            FROM reviews r JOIN movies m
            ON m.id = r.mid
            LEFT JOIN (
                SELECT opid, count(*) as follower_count
                FROM ties
                WHERE tie = 'follow'
                GROUP BY opid
            ) as t ON r.uid = t.opid
            WHERE uid NOT IN (
                SELECT opid FROM ties WHERE id = '{}' and tie = 'mute'
                )
            ORDER BY follower_count DESC NULLS LAST;
            """.format(id))
        reviews = cur.fetchall()
```

리뷰와 영화 테이블을 결합하여 필요한 정보를 모두 가져오고, 테이블에서 팔로워 수를 계산해 리뷰 작성자와 결합한다. 현재 사용자가 차단한 사용자의 리뷰는 제외한다. 팔로워 수 기준으로 내림차순 정렬하고, 팔로워 수가 없는 항목을 마지막에 배치한다.

movie_info.html

사용자의 ID와 메인 페이지 링크를 오른쪽 위에 표시한다. 이후 영화 제목과 영화 감독, 장르, 개봉일을 표로 보여준다. 리뷰를 표 형태로 표시한다. 각 리뷰는 평점, 사용자 ID, 리뷰 내용, 작성 시간을 포함한다.

```
<form action="/submit_review" method="post">
    <input type="hidden" name="id" value="{{ user }}">
    <input type="hidden" name="movie" value="{{ movie_info[0] }}">
    <input type="hidden" name="movie_name" value="{{ movie_info[1] }}">
    <label for="ratings">My Review:</label><br>
    <select name="ratings" id="ratings">
        {% for i in range(1, 6) %}
            <option value="{{ i }}">{{ i }}</option>
        {% endfor %}
```

```
    </select><br>
    <textarea name="review_text" rows="3" cols="40"></textarea>
    <br>
    <input type="submit" value="Submit">
</form>
```

사용자가 리뷰를 작성할 수 있는 폼이다. 사용자 ID, 영화 ID, 영화 제목을 숨겨진 필드로 포함하며, 평점과 리뷰 내용을 입력할 수 있다. 리뷰를 submit하면 지정된 URL을 호출한다.

movie_info함수

폼 데이터나 URL 파라미터에서 사용자 ID를 가져온다. GET, POST 방식 둘 다 사용이 가능하다. 또한 영화 이름을 가져온다. 우선 주어진 영화 제목으로 영화 정보를 조회한다. `movie_info`는 해당 영화의 모든 정보를 담고 있다.

```
cur.execute("""
    SELECT ratings, uid, review,
    TO_CHAR(rev_time, 'YYYY-MM-DD HH24:MI:SS')
    FROM reviews
    WHERE mid = '{}'
    AND uid NOT IN (SELECT opid FROM ties WHERE id = '{}' AND
    tie = 'mute');
    """.format(movie_info[0], id))
reviews = cur.fetchall()
```

해당 영화에 대한 리뷰를 조회한다. `movie_info[0]`는 영화의 ID를 나타내며, 현재 사용자가 차단한 사용자의 리뷰는 제외한다.

```
cur.execute("SELECT ROUND(AVG(ratings), 1) FROM reviews
WHERE mid = '{}';".format(movie_info[0]))
average_rating = cur.fetchone()[0]
```

해당 영화의 리뷰 평점 평균을 조회한다. 결과는 소수점 첫째 자리까지 반올림한다.

마지막으로 조회된 영화 정보, 리뷰, 평균 평점, 유사한 영화를 `movie_info.html` 템플릿에 전달하여 렌더링한다.

submit_review함수

사용자가 제출한 리뷰를 데이터베이스에 저장한다. `user_id`, `movie_id`, `movie_name`, `ratings`, `review_text`는 폼에서 제출된 데이터를 가져온다.

```
cur.execute("SELECT * FROM reviews WHERE uid = '{}'
AND mid = '{}';".format(user_id, movie_id))
existing_review = cur.fetchone()
```

사용자가 이미 해당 영화를 리뷰했는지 확인한다.

```
if existing_review:
    cur.execute("""
        UPDATE reviews
        SET ratings = '{}', review = '{}',
        rev_time = CURRENT_TIMESTAMP
        WHERE uid = '{}' AND mid = '{}';
    """.format(ratings, review_text, user_id, movie_id))
```

기존 리뷰가 있으면, 평점과 리뷰 텍스트를 업데이트하고 리뷰 시간을 현재 시간으로 갱신한다.

```
else:
    cur.execute("""
        INSERT INTO reviews
        VALUES ('{}', '{}', '{}', '{}', CURRENT_TIMESTAMP);
    """.format(movie_id, user_id, ratings, review_text))
```

기존 리뷰가 없으면, 새로운 리뷰를 삽입합니다. 리뷰 시간은 현재 시간으로 설정한다. commit한 후, `movie_info` 페이지로 리디렉션 한다.

user_info.html

`{% if target_role != 'admin' and user != target_user_id %}`

현재 사용자가 관리자가 아니고 타켓 사용자가 자신이 아닌 경우, 팔로우 및 뮤트 버튼을 제공한다. 이 버튼은 /tie경로를 호출한다.

유저가 작성한 모든 리뷰를 이후에 보여준다. 리뷰 아래에는 followers값으로부터 받아온 유저를 팔로우하는 사용자들을 모두 보여준다. 이 사용자들을 클릭하면 그 사용자의 유저 정보 페이지로 이동할 수 있다.

만약 유저 정보 페이지의 유저가 로그인한 사용자와 같다면, 로그인한 사용자가 팔로우하고, 뮤트하고 있는 사용자 정보 또한 알 수 있다. 이때 `unfollow` 버튼을 누르면 edit_tie경로로 이동해 팔로우를 해제하고, `unmute` 버튼 역시 누르면 뮤트를 해제한다.

```
<form action="/edit_tie" method="post">
  <input type="hidden" name="id" value="{{ user }}">
  <input type="hidden" name="target_user_id" value="{{ followed_user[0] }}"
  <input type="submit" name="send" value="unfollow">
</form>
```

admin으로 로그인한 상태로 admin 유저 페이지를 들어간다면 영화를 추가할 수 있는 폼이 작성되어있다. 영화 제목, 감독을 입력하고, 장르는 select 폼을 사용하여 선택할 수 있다. 마지막으로 Release Date도 선택할 수 있다.

```
<label for="genre">Genre:</label><br>
<select name="genre" required>
    <option value="" disabled selected>Select Genre</option>
    <option value="action">Action</option>
        ....
</select>
<label for="release_date">Release Date:</label><br>
<input type="date" name="release_date" required><br>
```

user_info 함수

POST와 GET 요청을 모두 처리하며, 특정 사용자의 정보를 조회하고 표시한다. `my_id`와 `target_user_id`는 각각 현재 사용자와 타겟 사용자의 ID를 폼 데이터나 URL 파라미터에서 가져온다. reviews에 타겟 사용자가 작성한 리뷰를 조회합니다. 영화와 리뷰를 조인하여 필요한 정보를 모두 가져오며, 최신순으로 정렬한다.

`movies`에는 모든 영화 목록을 조회한 값을 가져온다. `followers`에는 타켓 사용자를 팔로우하는 사용자의 ID를 조회한다. `followed_users`에는 현재 사용자가 팔로우한 사용자의 ID를 조회한다. `muted_users`에는 현재 사용자가 뮤트한 사용자의 ID를 모두 조회한다.

`user_role` 과 `target_role` 은 현재 사용자와 타겟 사용자의 역할을 조회합니다. 예를 들어, 일반 사용자와 관리자 역할을 구분한다.

조회한 결과를 user_info.html파일 렌더링에 함께 보낸다.

add_tie함수

사용자가 다른 사용자를 팔로우하거나 뮤트하는 기능을 가진다. send는 사용자가 선택한 액션(팔로우 또는 뮤트)을 나타낸다.

```
if send in ["follow", "mute"]:
    cur.execute("""
        SELECT * FROM ties
        WHERE id = '{}' AND opid = '{}' AND tie = '{}';
    """.format(id, target_user_id, send))
    exists = cur.fetchone()
```

사용자가 이미 타겟 사용자를 팔로우하거나 뮤트했는지 확인한다.

```
if not exists:
    if send == "follow":
        opposite_send = "mute"
    elif send == "mute":
        opposite_send = "follow"
    cur.execute("""
            DELETE FROM ties
            WHERE id = '{}' AND opid = '{}' AND tie = '{}';
        """.format(id, target_user_id, opposite_send))

    cur.execute("""
            INSERT INTO ties (id, opid, tie)
            VALUES ('{}', '{}', '{}');
        """.format(id, target_user_id, send))
```

사용자가 타겟 사용자를 팔로우하거나 뮤트하지 않은 경우, 반대 관계(팔로우 또는 뮤트)를 삭제하고 새 관계를 추가한다. 만약 이미 팔로우하거나 뮤트한 상태라면 아무일도 발생하지 않는다. follow를 클릭하면 타켓 유저 아이디의 mute상태를 삭제하고, follow로 변경한다. mute 버튼 역시 비슷한 역할을 한다.

edit_tie 함수

이 함수는 로그인한 사용자가 자신의 페이지를 들어갔을 때만 사용할 수 있다. 로그인한 사용자는 followed와 muted된 target_user_id를 알 수 있다. `send` 값이 `unfollow` 또는 `unmute`인 경우, 각각 `follow` 또는 `mute`로 변환한다. 이후 해당 관계를 데이터베이스에서 삭제한다.

add_movie 함수

admin이 자신의 유저 페이지를 들어갔을 때, 영화를 추가할 수 있는 폼이 나온다. 폼 값들을 입력하고 추가를 누르면 이 함수가 실행된다.

```
cur.execute("SELECT MAX(id) FROM movies;")
max_id_result = cur.fetchone()
if max_id_result and max_id_result[0] is not None:
    new_id = int(max_id_result[0]) + 1
else:
    new_id = 1
```

지금 현재 movies 테이블에서 가장 큰 id값을 받아온다. 만약 가장 큰 id 값이 존재한다면 가장 큰 ID값에서 1을 더해 primary key 값을 채울 수 있다. 만약 영화가 없으면 ID는 1이 된다. 폼에서 영화 정보를 가져오고, insert 쿼리를 사용해 새로운 영화를 데이터베이스에 삽입하고, 변경 사항을 커밋한다.

main_page 함수

register함수에서 만들었던 SQL문과 같다. review와 movie에 대한 값들을 SQL문으로 받아와서 main.html을 렌더링한다. 이 함수는 유저 정보 페이지나 영화 정보 페이지에서 main page로 다시 돌아가는 버튼을 누르면 사용된다.

✏️ 추가 기능

movie_info 함수의 내부 SQL 로직 추가

```
<div>
    <h3>Similar Movies</h3>
    <ul>
        {% for similar in similar_movies %}
            <li>{{ similar[0] }}</li>
        {% endfor %}
    </ul>
</div>
```

```
cur.execute("""
SELECT title FROM movies WHERE genre = '{}' AND id != '{}' LIMIT 5;
```

```
""".format(movie_info[3], movie_info[0]))
similar_movies = cur.fetchall()
```

movie_info 페이지에 Similar movies라는 값을 추가한다. 비슷한 영화 값들은 같은 장르의 영화를 알려주는 기능이다. SQL 문은 영화 속성 중, 장르와 id 값을 사용해 장르가 같고, id가 다른 비슷한 영화를 최대 5개 찾아준다. 찾은 값은 similar_movies로 넘겨줘서 html 파일에서 볼 수 있도록 한다.

delete_movie 함수 추가

admin이 자신의 유저 페이지를 들어갔을 때, 영화를 삭제할 수 있는 테이블이 나온다. 테이블에서 delete버튼을 누르면 이 함수가 실행된다.

```
{% if target_role == 'admin' and user == "admin" %}
  <h2>[Delete Movie]</h2>
  <table border="1">
    <thead>
    <th>title</th>
    <th>director</th>
    <th>genre</th>
    <th>rel_date</th>
    <th>delete</th>
    </thead>
    <tbody>
    {% for movie in movies %}
      <tr>
        <td>{{ movie[1] }}</td>
        <td>{{ movie[2] }}</td>
        <td>{{ movie[3] }}</td>
        <td>{{ movie[4] }}</td>
        <td>
          <form action="/delete_movie" method="post">
            <input type="hidden" name="id" value= {{ user }}>
            <input type="hidden" name="movieId" value= {{ movie[0] }}>
            <input type="hidden" name="target_user_id" value="{{ target_u
            <input type="submit" name="send" value="delete">
          </form>
        </td>
      </tr>
```

```
        {% endfor %}
        </tbody>
    </table>
{% endif %}
```

다음과 같은 코드를 user_info.html에 추가한다. 현재 존재하는 모든 영화에 대한 정보를 보여주고, delete 버튼을 누르면 해당하는 영화를 삭제한다.

```
cur.execute("DELETE FROM movies WHERE '{}' = id;".format(movieId))
connect.commit()
```

지정한 영화의 ID를 데이터베이스에서 삭제하고, 변경 사항을 커밋한다. 이후 user_info 페이지로 리디렉션한다.