1. **Section 1 ER-**다이어그램 구축 및 계정 생성

1-1 정규화 연습 **(3**점**)**

- **1** 정규형

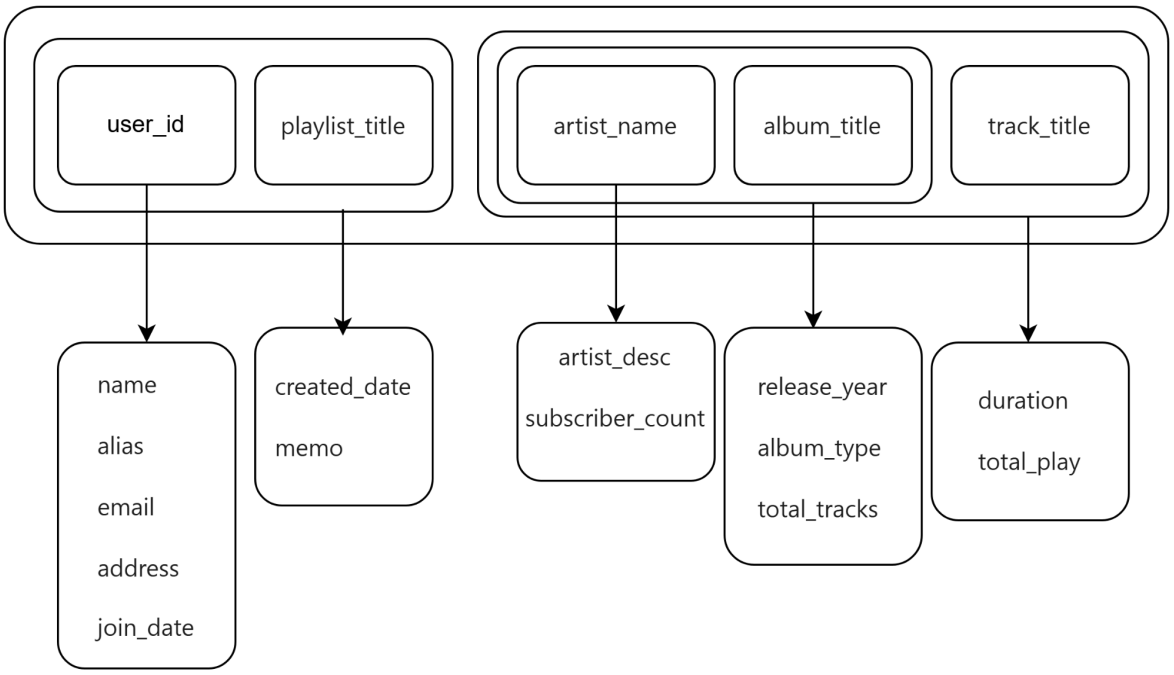
name	alias	email	address	join_date	playlist_title	created_date	memo
김철수	cheol777	cs.kim@hallym.ac.kr	강원도 춘천시 퇴계동 A 아파트	2021-01-10	2024 트렌딩	2024-12-03	최신곡들 모음
김철수	cheol777	cs.kim@hallym.ac.kr	강원도 춘천시 퇴계동 A 아파트	2021-01-10	2024 트렌딩	2024-12-03	최신곡들 모음
이영희	lyh004	yh.lee@hallym.ac.kr	강원도 춘천시 소양동 B빌라	2023-02-15	노동요	2023-10-09	출근해서 듣기
이영희	lyh004	yh.lee@hallym.ac.kr	강원도 춘천시 소양동 B빌라	2023-02-15	노동요	2023-10-09	출근해서 듣기
박민수	soosoo	ms.park@hallym.ac.kr	강원도 춘천시 우두동 C원룸	2022-03-20	국내 겨울 음악	2024-11-29	포근한 노래
최수정	crystal_choi	sj.choi@hallym.ac.kr	강원도 춘천시 퇴계동 D아파트	2024-04-25	Kpop Hits 2024	2024-12-01	글로벌 케이팝
홍길동	west_south8	gd.hong@hallym.ac.kr	강원도 춘천시 우두동 E빌리지	2023-05-30	Pop Hits	2024-12-01	pop certified

artist_name	artist_desc	subscriber_count	album_title	release_year	album_type	total_tracks	track_title	duration
사브리나 카펜터	미국의 가수이자 배우	10100000	Short n Sweet	2024	Album	12	Espresso	0:02:56
로제	걸그룹 BLACKPINK의 멤버	13400000	rosie	2024	Album	12	APT.	0:02:50
브루노 마스	미국의 싱어송라이터	39600000	24K Magic	2016	Album	36	24K Magic	0:03:46
브루노 마스	미국의 싱어송라이터	39600000	24K Magic	2016	Album	36	Finesse	0:03:11
로제	걸그룹 BLACKPINK의 멤버	13400000	rosie	2024	Album	12	number one girl	0:03:37
지드래곤	대한민국의 가수	3300000	POWER	2024	Single	1	POWER	0:02:24
사브리나 카펜터	미국의 가수이자 배우	10100000	Short n Sweet	2024	Album	12	Please, please, please	0:03:07

시나리오02.xlsx 파일의 데이터는 모든 속성값이 원자성을 만족하므로, 1 정규형의 조건을 이미 충족합니다.

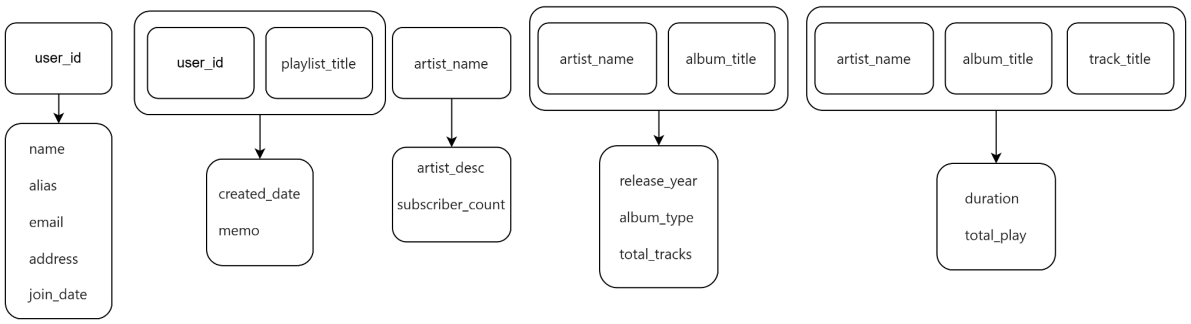
- **2** 정규형

주어진 테이블의 복합 키들과 속성들의 관계를 나타내면 다음과 같습니다.



-

2 정규형은 부분 함수 종속을 제거하는 것이므로 복합 키의 일부 기본키에만 의존하는 테이블을 분리해서 부분 함수 종속을 제거함으로써 2 정규화를 합니다, 2 정규화를 통해 데이터 중복을 줄이고 데이터 무결성을 유지할 수 있습니다.



위 다이어그램을 바탕으로 시나리오02.xlsx 데이터의 테이블을 분리하면 다음과 같다.

[users 테이블]

name	alias	email	address	join_date
김철수	cheol777	cs.kim@hallym.ac.kr	강원도 춘천시 퇴계동 A 아파트	2021-01-10
이영희	lyh004	yh.lee@hallym.ac.kr	강원도 춘천시 소양동 B빌라	2023-02-15
박민수	soosoo	ms.park@hallym.ac.kr	강원도 춘천시 우두동 C원룸	2022-03-20
최수정	crystal_choi	sj.choi@hallym.ac.kr	강원도 춘천시 퇴계동 D아파트	2024-04-25
홍길동	west_south8	gd.hong@hallym.ac.kr	강원도 춘천시 우두동 E빌리지	2023-05-30

[playlists 테이블]

name	playlist_title	created_date	memo
김철수	2024 트렌딩	2024-12-03	최신곡들 모음
이영희	노동요	2023-10-09	출근해서 듣기
박민수	국내 겨울 음악	2024-11-29	포근한 노래
최수정	Kpop Hits 2024	2024-12-01	글로벌 케이팝
홍길동	Pop Hits	2024-12-01	pop certified

[artists 테이블]

artist_name	artist_desc	subscriber_count
사브리나 카펜터	미국의 가수이자 배우	10100000
로제	걸그룹 BLACKPINK의 멤버	13400000
브루노 마스	미국의 싱어송라이터	39600000
지드래곤	대한민국의 가수	3300000

[albums 테이블]

artist_name	album_title	release_year	album_type	total_tracks
사브리나 카펜터	Short n Sweet	2024	Album	12
로제	rosie	2024	Album	12
브루노 마스	24K Magic	2016	Album	36
지드래곤	POWER	2024	Single	1

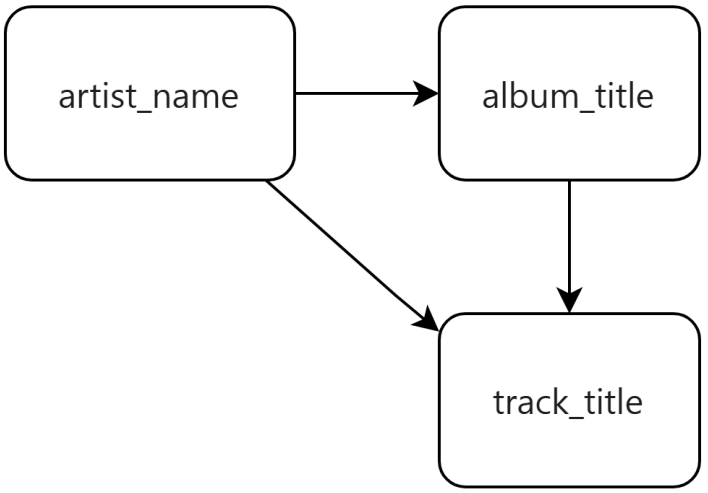
[tracks 테이블]

artist_name	album_title	track_title	duration	total_play
사브리나 카펜터	Short n Sweet	Espresso	0:02:56	620000000
로제	rosie	APT.	0:02:50	500000000
브루노 마스	24K Magic	24K Magic	0:03:46	1900000000
브루노 마스	24K Magic	Finesse	0:03:11	476700000
로제	rosie	number one girl	0:03:37	57310000
지드래곤	POWER	POWER	0:02:24	70850000
사브리나 카펜터	Short n Sweet	Please, please, please	0:03:07	370000000

- **3** 정규형

2정규형 과정을 통해 부분 함수 종속은 제거되었으나, tracks 테이블에서 이행적 함수 종속이 발생하고 있습니다. artist_name이 album_title을 결정하고, album_title이 track_title을 결정하므로, artist_name과 track_title 간에 이행적 함수 종속이 존재합니다. 따라서, tracks 테이블에서 artist_name을 분리해야 합니다.

[이행적 종속 관계]

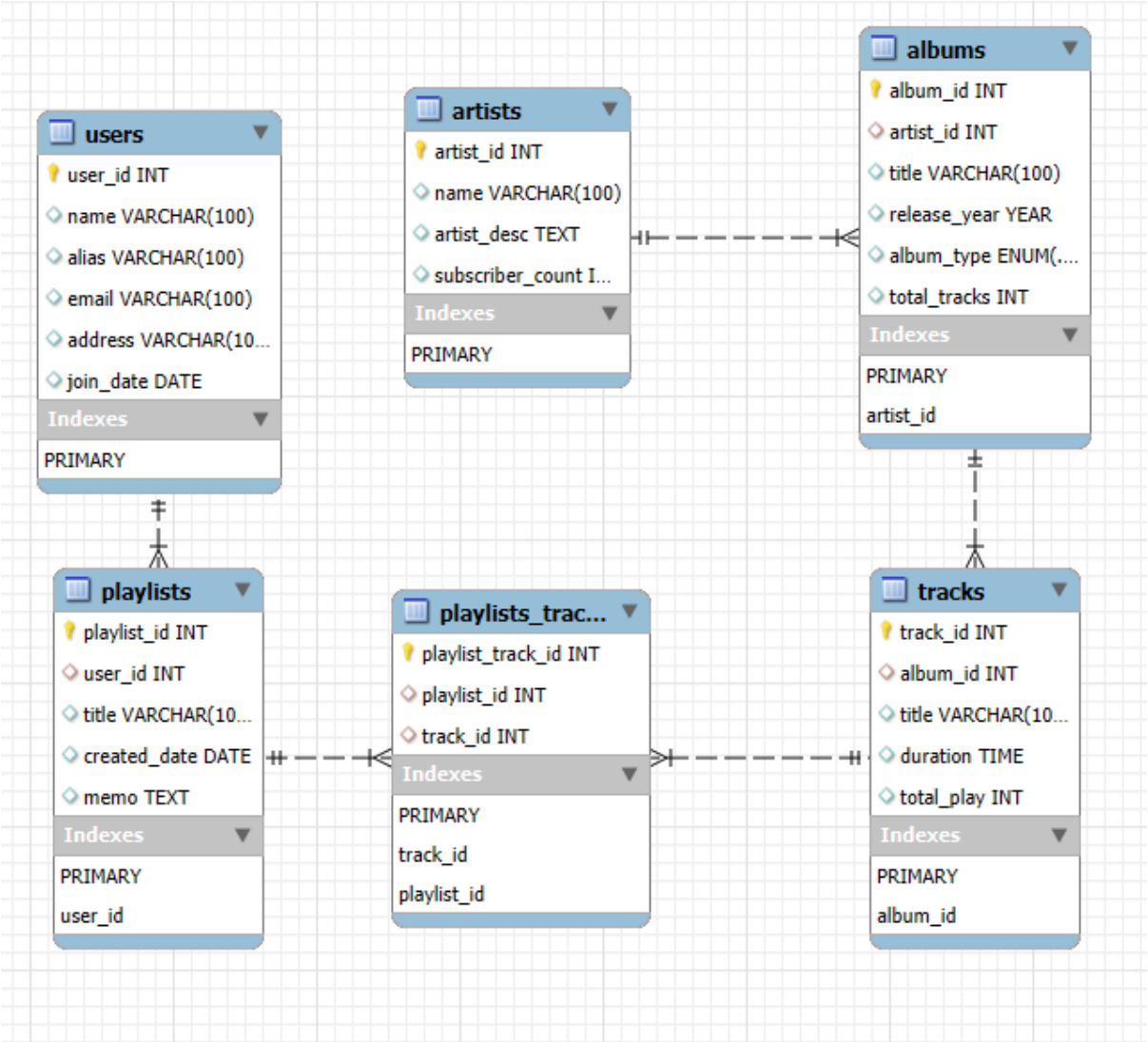


[artist_name을 분리한 tracks 테이블]

album_title	track_title	duration	total_play
Short n Sweet	Espresso	0:02:56	620000000
rosie	APT.	0:02:50	500000000
24K Magic	24K Magic	0:03:46	1900000000
24K Magic	Finesse	0:03:11	476700000
rosie	number one girl	0:03:37	57310000
POWER	POWER	0:02:24	70850000
Short n Sweet	Please, please, please	0:03:07	370000000

1-2 ER-다이어그램 구축 **(2**점**)**

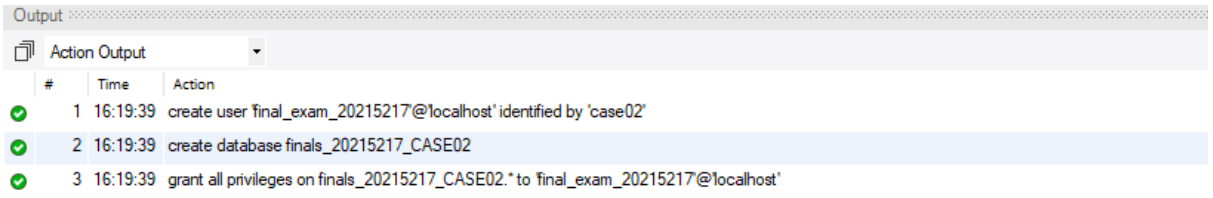
<시나리오2> 의 요구사항을 충족하는 <한림뮤직>의 ER-다이어그램이다.



1-2 데이터베이스 및 계정 셋팅 **(setup_db.sql) (2**점**)**

[final_20215217_CASE02 데이터베이스와 데이터베이스의 권한을 부여받을 finals_exam_20215217 로컬호스트 계정을 생성하고 권한을 부여하는 SQL 코드이다.]

```
-- 새로운 계정 final_exam_[학번] 생성, 비밀번호 case02
create user 'final_exam_20215217'@'localhost' identified by 'case02';
-- 데이터베이스 finals_[학번]_CASE02 구축
create database finals_20215217_CASE02;
-- 권한 부여
grant all privileges on finals_20215217_CASE02.* to 'final_exam_20215217'@'localhost';
```



2. **Section 2** 테이블 구축 **(3**점**)**

2-1 다음 태스크에 적합한 **build_tables.sql**를 작성하시오**.**

[users 테이블을 생성하는 SQL 코드이다. users 테이블의 id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다.
AI가 키를 1부터 증가시키도록 설정하였다.]

```
-- 생성한 데이터베이스를 사용해서 테이블을 구축한다.
use finals_20215217_CASE02;

-- 1. 사용자(users)
-- 고유한 사용자번호( user_id )로 식별하며, 이름( name ), 별칭( alias ),
-- 이메일( email ), 주소( address ), 가입일( join_date ) 정보를 가진다.
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,  -- 자동 증가하는 기본 키
    name varchar(100),
    alias varchar(100),
    email varchar(100),
    address varchar(100),
    join_date date
)AUTO_INCREMENT = 1; -- 기본 키가 1부터 증가하도록 설정
```

[playlists 테이블을 생성하는 SQL 코드이다. id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다. AI 속성의 키를 1부터 증가시키도록 설정하였다, users테이블의 user_id를 외래키로 가진다.]

```sql
-- 2. 플레이리스트(playlists)
-- 고유한 플레이리스트번호( playlist_id )로 식별하며, 사용자번호( user_id )를 통해
-- 이 플레이리스트를 만든 사용자를 식별할 수 있으며, 이는 users 테이블과 연계되어 있다.
-- 플레이리스트 제목( title ), 생성일자( created_date ), 메모( memo ) 정보를 가진다.
-- playlists_tracks 테이블을 통해 특정 플레이리스트에 포함된 트랙 정보를 연결할 수 있다.
create table playlists (
    playlist_id INT AUTO_INCREMENT PRIMARY KEY,  -- 자동 증가하는 기본 키
    user_id int,
    title varchar(100),
    created_date date,
    memo text,
    FOREIGN KEY (user_id) REFERENCES users(user_id)  -- 외래 키 설정
)AUTO_INCREMENT = 1;  -- 기본 키가 1부터 증가하도록 설정
```

[artists 테이블을 생성하는 SQL 코드이다. id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다. AI 속성의 키를 1부터 증가시키도록 설정하였다.]

```sql
-- 3. 아티스트(artists)
-- 고유한 아티스트번호( artist_id )로 식별하며, 아티스트 이름( name ),
-- 아티스트 설명( artist_desc ), 구독자 수( subscriber_count ) 정보를 가진다.
-- 이 테이블은 albums 테이블과의 관계를 통해 특정 앨범이 어떤 아티스트에 속하는지 식별할 수 있다.
create table artists(
    artist_id INT AUTO_INCREMENT PRIMARY KEY,  -- 자동 증가하는 기본 키
    name varchar(100),
    artist_desc text,
    subscriber_count int
)AUTO_INCREMENT = 1;  -- 기본 키가 1부터 증가하도록 설정
```

[albums 테이블을 생성하는 SQL 코드이다. id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다. AI 속성의 키를 1부터 증가시키도록 설정하였다, artists테이블의 artist_id를 외래키로 가진다. 외래키 제약조건이 ON DELETE CASCADE로 설정해놓아서 artist테이블의 데이터가 삭제되면 삭제된 아티스트의 앨범도 삭제되도록 해놓았다.]

```
-- 4. 앨범(albums)
-- 고유한 앨범번호( album_id )로 식별한다.
-- 아티스트번호( artist_id )를 통해 이 앨범을 낸 아티스트를 식별하며,
-- 아티스트 삭제 시 연관된 앨범도 자동으로 삭제되도록 설정(ON DELETE CASCADE)되어 있다.
CREATE TABLE albums (
    album_id INT AUTO_INCREMENT PRIMARY KEY,  -- 자동 증가하는 기본 키
    artist_id INT,
    title VARCHAR(100),
    release_year YEAR,  -- 연도만 저장
    album_type ENUM('Single', 'Album'),  -- 앨범 유형
    total_tracks INT,
    FOREIGN KEY (artist_id) REFERENCES artists(artist_id)  -- 외래 키 설정
    ON DELETE CASCADE  -- artist가 삭제되면 앨범도 자동 삭제
)AUTO_INCREMENT = 1;  -- 기본 키가 1부터 증가하도록 설정
```

[tracks 테이블을 생성하는 SQL 코드이다. id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다. AI 속성의 키를 1부터 증가시키도록 설정하였다, albums테이블의 album_id를 외래키로 가진다.]

```
-- 5. 트랙(tracks)
-- 고유한 트랙번호( track_id )로 식별한다.
-- 앨범번호( album_id )를 통해 각 트랙이 속한 앨범을 식별할 수 있다.
-- 트랙 제목( title ), 재생시간( duration ), 총 재생수( total_play ) 정보를 가진다.
-- 플레이리스트와 트랙의 연결을 표현하는 playlists_tracks 테이블을 통해,
-- 특정 플레이리스트에 어떤 곡이 포함되는지 알 수 있다.

create table tracks(
    track_id int auto_increment primary key,
    album_id int,
    title varchar(100),
    duration time,
    total_play int,
    foreign key(album_id) references albums(album_id)
)AUTO_INCREMENT = 1;  -- 기본 키가 1부터 증가하도록 설정
```

[playlists_tracks 테이블을 생성하는 SQL 코드이다. id 컬럼은 Auto Increment(AI) 속성을 사용하여, 새로운 레코드가 추가될 때 자동으로 고유한 번호를 생성한다. AI 속성의 키를 1부터 증가시키도록 설정하였다, playlists테이블의 playlist_id와 tracks테이블의 track_id를 외래키로 가진다.]

```sql
-- 6. 플레이리스트-트랙 관계(playlists_tracks)
-- 고유한 식별번호( id )로 식별하며, 플레이리스트번호( playlist_id )와
-- 트랙번호( track_id )를 통해 특정 플레이리스트에 속한 트랙들을 연결한다.
-- 이를 통해 하나의 플레이리스트가 여러 트랙을 가질 수 있고, 하나의 트랙도
-- 여러 플레이리스트에 속할 수 있는 N:M 관계를 형성한다

create table playlists_tracks(
    playlist_track_id int auto_increment primary key,
    playlist_id int,
    track_id int,
    foreign key(track_id) references tracks(track_id),
    foreign key(playlist_id) references playlists(playlist_id)
)AUTO_INCREMENT = 1;  -- 기본 키가 1부터 증가하도록 설정
```

요구사항을 만족하는 build_tables.sql 스크립트를 작성하여 다음과 같은 결과를 얻었다.

Output

Action Output

#	Time	Action
1	16:21:46	use finals_20215217_CASE02
2	16:22:35	CREATE TABLE users (user_id INT AUTO_INCREMENT PRIMARY KEY, -- 자동 증가하는 기본 키 name varchar(100), alias varchar(10...
3	16:22:35	create table playlists (playlist_id INT AUTO_INCREMENT PRIMARY KEY, -- 자동 증가하는 기본 키 user_id int, title varchar(100), created...
4	16:22:35	create table artists(artist_id INT AUTO_INCREMENT PRIMARY KEY, -- 자동 증가하는 기본 키 name varchar(100), artist_desc text, subscriber_c...
5	16:22:35	CREATE TABLE albums (album_id INT AUTO_INCREMENT PRIMARY KEY, -- 자동 증가하는 기본 키 artist_id INT, title VARCHAR(100)...
6	16:22:35	create table tracks(track_id int auto_increment primary key, album_id int, title varchar(100), duration time, total_play int, foreign key(alb...
7	16:22:35	create table playlists_tracks(playlist_track_id int auto_increment primary key, playlist_id int, track_id int, foreign key(track_id) references tra...

3. **Section 3** 데이터 입력 **(2**점**)**

3-1. 다음 태스크에 적합한 **insert_data.sql**을 작성하시오**.**

- 데이터 입력 구문 작성

[아래는 시나리오02의 데이터를 각각 알맞는 테이블에 삽입하는 SQL 코드이다. 마지막 부분에는 데이터가 삽입되었는지 확인할 수 있도록 테이블들을 모두 출력하는 코드가 있다.]

```
-- 시나리오02.xlsx 데이터를 그대로 users 테이블에 삽입한다.
insert into users (name, alias, email, address, join_date) values
("김철수", "cheol777", "cs.kim@hallym.ac.kr", "강원도 춘천시 퇴계동 A 아파트", "2021-01-10"),
("이영희", "lyh004", "yh.lee@hallym.ac.kr", "강원도 춘천시 소양동 B빌라", "2023-02-15"),
("박민수", "soosoo", "ms.park@hallym.ac.kr", "강원도 춘천시 우두동 C원룸", "2022-03-20"),
("최수정", "crystal_choi", "sj.choi@hallym.ac.kr", "강원도 춘천시 퇴계동 D아파트", "2024-04-25"),
("홍길동", "west_south8", "gd.hong@hallym.ac.kr", "강원도 춘천시 우두동 E빌리지", "2023-05-30");

-- 시나리오02.xlsx 데이터를 그대로 playlists 테이블에 삽입한다.
insert into playlists (user_id, title, created_date, memo) values
(1, "2024 트렌딩", "2024-12-03", "최신곡들 모음"),
(2, "노동요", "2023-10-09", "출근해서 듣기"),
(3, "국내 겨울 음악", "2024-11-29", "포근한 노래"),
(4, "Kpop Hits 2024", "2024-12-01", "글로벌 케이팝"),
(5, "Pop Hits", "2024-12-01", "pop certified");

-- 시나리오02.xlsx 데이터를 그대로 artists 테이블에 삽입한다.
insert into artists (name, artist_desc, subscriber_count) values
("사브리나 카펜터", "미국의 가수이자 배우", 10100000),
("로제", "걸그룹 BLACKPINK의 멤버", 13400000),
("브루노 마스", "미국의 싱어송라이터", 39600000),
("지드래곤", "대한민국의 가수", 3300000);

-- 시나리오02.xlsx 데이터를 그대로 albums 테이블에 삽입한다.
insert into albums (artist_id, title, release_year, album_type, total_tracks) values
(1, "Short n Sweet", "2024", "Album", 12),
(2, "rosie", "2024", "Album", 12),
(3, "24K Magic", "2016", "Album", 36),
(4, "POWER", "2024", "Single", 1);
```

```sql
-- 시나리오02.xlsx 데이터를 그대로 tracks 테이블에 삽입한다.
insert into tracks(album_id, title, duration, total_play) values
(1, "Espresso", "0:02:56",  620000000),
(1, "Please, please, please", "0:03:07", 370000000),
(2, "APT.", "0:02:50", 500000000),
(2, "number one girl", "0:03:37", 57310000),
(3, "24K Magic", "0:03:46", 1900000000),
(3, "Finesse", "0:03:11", 476700000),
(4, "POWER", "0:02:24", 70850000);

-- 시나리오02.xlsx 데이터를 그대로 playlists_tracks 테이블에 삽입한다.
insert into playlists_tracks (playlist_id, track_id) values
(1, 1),
(1, 3),
(2, 5),
(2, 6),
(3, 4),
(4, 7),
(5, 2);

-- 데이터가 삽입이 되었는지 출력해서 확인한다.
select * from users;
select * from playlists;
select * from artists;
select * from albums;
select * from tracks;
select * from playlists_tracks;
```

요구사항을 만족하는 insert_data.sql 스크립트를 작성하여 다음과 같은 결과를 얻었다.

[데이터 삽입 후 **users** 테이블 출력 결과]

user_id	name	alias	email	address	join_date
1	김철수	cheol777	cs.kim@hallym.ac.kr	강원도 춘천시 퇴계동 A 아파트	2021-01-10
2	이영희	lyh004	yh.lee@hallym.ac.kr	강원도 춘천시 소양동 B빌라	2023-02-15
3	박민수	soosoo	ms.park@hallym.ac.kr	강원도 춘천시 우두동 C원룸	2022-03-20
4	최수정	crystal_choi	sj.choi@hallym.ac.kr	강원도 춘천시 퇴계동 D아파트	2024-04-25
5	홍길동	west_south8	gd.hong@hallym.ac.kr	강원도 춘천시 우두동 E빌리지	2023-05-30
NULL	NULL	NULL	NULL	NULL	NULL

[데이터 삽입 후 **playlists** 테이블 출력 결과]

playlist_id	user_id	title	created_date	memo
1	1	2024 트렌딩	2024-12-03	최신곡들 모음
2	2	노동요	2023-10-09	출근해서 듣기
3	3	국내 겨울 음악	2024-11-29	포근한 노래
4	4	Kpop Hits 2024	2024-12-01	글로벌 케이팝
5	5	Pop Hits	2024-12-01	pop certified
NULL	NULL	NULL	NULL	NULL

[데이터 삽입 후 **artists** 테이블 출력 결과]

artist_id	name	artist_desc	subscriber_count
1	사브리나 카펜터	미국의 가수이자 배우	10100000
2	로제	걸그룹 BLACKPINK의 멤버	13400000
3	브루노 마스	미국의 싱어송라이터	39600000
4	지드래곤	대한민국의 가수	3300000
NULL	NULL	NULL	NULL

[데이터 삽입 후 **albums** 테이블 출력 결과]

album_id	artist_id	title	release_year	album_type	total_tracks
1	1	Short n Sweet	2024	Album	12
2	2	rosie	2024	Album	12
3	3	24K Magic	2016	Album	36
4	4	POWER	2024	Single	1

[데이터 삽입 후 **tracks** 테이블 출력 결과]

track_id	album_id	title	duration	total_play
1	1	Espresso	00:02:56	620000000
2	1	Please, please, please	00:03:07	370000000
3	2	APT.	00:02:50	500000000
4	2	number one girl	00:03:37	57310000
5	3	24K Magic	00:03:46	1900000000
6	3	Finesse	00:03:11	476700000
7	4	POWER	00:02:24	70850000

[데이터 삽입 후 playlists_tracks 테이블 출력 결과]

playlist_track_id	playlist_id	track_id
1	1	1
2	1	3
3	2	5
4	2	6
5	3	4
6	4	7
7	5	2

4. **Section 4** 데이터 변경 및 삭제 **(2점)**

4-1. 다음 태스크에 적합한 **alter_data.sql**을 작성하시오**.**

[아래는 메모에 '노동요'라고 저장된 플레이리스트의 메모를 '출근할 때 듣는 신나는 노래들'로 변경하고 출력하는 SQL 코드이다.]

```
-- ① 데이터 변경 1: 플레이리스트의 메모 수정: '노동요' 플레이리스트의
-- 메모를 '출근할 때 듣는 신나는 노래들'로 변경
update playlists set memo = "출근할 때 듣는 신나는 노래들" where title = "노동요";
select * from playlists;
```

[변경된 playlists 테이블 출력]

playlist_id	user_id	title	created_date	memo
1	1	2024 트렌딩	2024-12-03	최신곡들 모음
2	2	노동요	2023-10-09	출근할 때 듣는 신나는 노래
3	3	국내 겨울 음악	2024-11-29	포근한 노래
4	4	Kpop Hits 2024	2024-12-01	글로벌 케이팝
5	5	Pop Hits	2024-12-01	pop certified
NULL	NULL	NULL	NULL	NULL

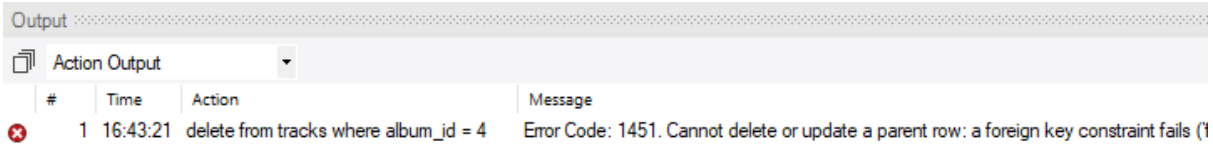
- 삭제 구문 작성 / 발생가능한 오류 - 해결 방법 제시 및 설명

tracks 테이블에서 album_id가 4인 앨범의 기록을 삭제하려고 할 때, track_id를 외래키로 가지는 playlists_tracks 테이블의 존재로 인해 데이터 무결성 제약에 의해 tracks 테이블의 데이터를 삭제할 수 없습니다. 그러나 playlists_tracks 테이블의 외래키 제약 조건을 ON DELETE CASCADE로 변경하면, 해당 앨범의 트랙이 삭제될 때 관련된 playlists_tracks 테이블의 레코드도 자동으로 삭제되어, tracks 테이블에서 데이터를 삭제할 수 있습니다. 기존의 외래키를 삭제, 외래키 제약 조건이 ON DELETE CASCADE인 외래키를 추가합니다.

[tracks테이블에서 album_id가 4인 데이터를 삭제하고 tracks테이블을 출력하는 SQL 코드이다.]

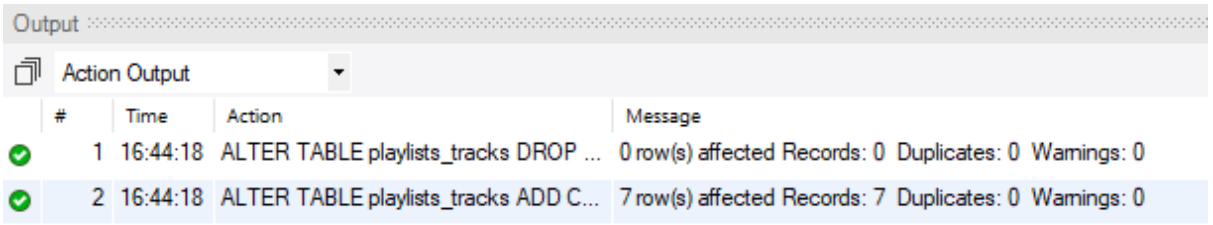
```
-- 삭제 구문 작성
-- ② 데이터 삭제 1: tracks 테이블에서 album_id가 4인 앨범 기록을 삭제
delete from tracks where album_id = 4;
```

[playlists_tracks 테이블이 track_id키를 참조하고 있어서 데이터 무결성 원칙으로 인해 삭제가 불가능하다.]



[playlists_tracks 테이블의 track_id 기존 외래키를 삭제하고 ON DELETE CASCADE옵션을 가진 외래키로 새로 만드는 SQL코드이다.]

```
-- playlists_tracks의 외래키 제약조건으로 삭제 불가능
-- 기존의 외래키를 삭제하고 ON CASCASE DELETE 제약 조건을 가진 외래키를 생성
ALTER TABLE playlists_tracks
DROP FOREIGN KEY playlists_tracks_ibfk_1;

ALTER TABLE playlists_tracks
ADD CONSTRAINT playlists_tracks_ibfk_1
FOREIGN KEY (track_id) REFERENCES tracks(track_id)
ON DELETE CASCADE;
```



[tracks테이블에서 album_id가 4인 데이터를 삭제하고 tracks테이블을 출력하는 SQL 코드이다.]

- 삭제한 테이블 확인

```sql
-- 삭제한 테이블 확인
delete from tracks where album_id = 4;
select * from tracks;
```

Result Grid | Filter Rows: | Edit:

track_id	album_id	title	duration	total_play
1	1	Espresso	00:02:56	620000000
2	1	Please, please, please	00:03:07	370000000
3	2	APT.	00:02:50	500000000
4	2	number one girl	00:03:37	57310000
5	3	24K Magic	00:03:46	1900000000
6	3	Finesse	00:03:11	476700000
NULL	NULL	NULL	NULL	NULL

5. **Section 5** 데이터 검색 **(4점)**

5-1 다음 태스크에 적합한 **query_view.sql**을 작성하시오**.**

- 데이터 검색 쿼리 작성
- 필요시 뷰로 생성

[부속질의를 사용하여 평균보다 많은 플레이 수(total_play)를 가진 아티스트 이름을 조회하는 SQL 코드이다.]

```sql
-- <시나리오2>
-- ※ 주의사항: tracks 테이블에 artist_id 외래 키가 없으므로, albums 테이블을 통해 artists 테이블과 조인해야 함!
-- ① 부속질의를 사용하여 평균보다 많은 플레이 수(total_play)를 가진 아티스트 이름을 조회
SELECT a.name, t.total_play
FROM artists a
JOIN albums al ON a.artist_id = al.artist_id
JOIN tracks t ON al.album_id = t.album_id
WHERE t.total_play > (SELECT AVG(total_play) FROM tracks);
```

name	total_play
브루노 마스	1900000000

[각 아티스트가 가진 트랙 수를 계산하는 뷰(artist_track_count)를 생성하고, 이를 이용해 트랙 수가 2개 이상인 아티스트를 조회하는 SQL코드이다.]

```sql
-- ⑦ 각 아티스트가 가진 트랙 수를 계산하는 뷰(artist_track_count)를 생성하고, 이를 이용해 트랙 수가 2개 이상인 아티스트를 조회
CREATE VIEW artist_track_count AS
SELECT a.name, COUNT(t.track_id) AS track_count
FROM artists a
JOIN albums al ON a.artist_id = al.artist_id
JOIN tracks t ON al.album_id = t.album_id
GROUP BY a.name;

SELECT name, track_count
FROM artist_track_count
WHERE track_count >= 2;
```

name	track_count
사브리나 카펜터	2
로제	2
브루노 마스	2

[playlists_tracks 테이블과 tracks, albums, artists 테이블을 조인하여 트랙의 제목과 아티스트 이름을 조회, 이를 뷰(playlist_tracks_view)로 생성하는 SQL코드이다.]

```sql
-- ⑧ playlists_tracks 테이블과 tracks, albums, artists 테이블을 조인하여
-- 트랙의 제목과 아티스트 이름을 조회, 이를 뷰(playlist_tracks_view)로 생성
-- 뷰 조회: SELECT * FROM playlist_tracks_view;
-- ⑨ 위의 뷰를 사용하여 플레이리스트 ID가 1인 플레이리스트의 트랙 1건
-- 조회 (LIMIT 사용)
CREATE VIEW playlist_tracks_view AS
SELECT pt.playlist_id, t.title, a.name
FROM playlists_tracks pt
JOIN tracks t ON pt.track_id = t.track_id
JOIN albums al ON t.album_id = al.album_id
JOIN artists a ON al.artist_id = a.artist_id;

SELECT * FROM playlist_tracks_view;
```

playlist_id	title	name
1	Espresso	사브리나 카펜터
5	Please, please, please	사브리나 카펜터
1	APT.	로제
3	number one girl	로제
2	24K Magic	브루노 마스
2	Finesse	브루노 마스

[위의 뷰를 사용하여 플레이리스트 ID가 1인 플레이리스트의 트랙 1건 조회 (LIMIT 사용)하는 SQL코드이다.]

```sql
-- ⑧ 위의 뷰를 사용하여 플레이리스트 ID가 1인 플레이리스트의 트랙 1건 조회 (LIMIT 사용)
SELECT *
FROM playlist_tracks_view
WHERE playlist_id = 1
LIMIT 1;
```

	playlist_id	title	name
▶	1	Espresso	사브리나 카펜터

6. **Section 6** 웹 프로그래밍 **(15점)**

- 각 화면 설계에 적합한 **Streamlit UI** 구현
- **MySQL** 데이터베이스와 연결
- 로그인 사이드바 추가
- 데이터 **CRUD** 작업 기능**(**읽기**)**
- 데이터 **CRUD** 작업 기능**(**쓰기**)**

[MySQL 데이터베이스와 연결 / secrets.toml 파일 및 st.connection으로 연결]

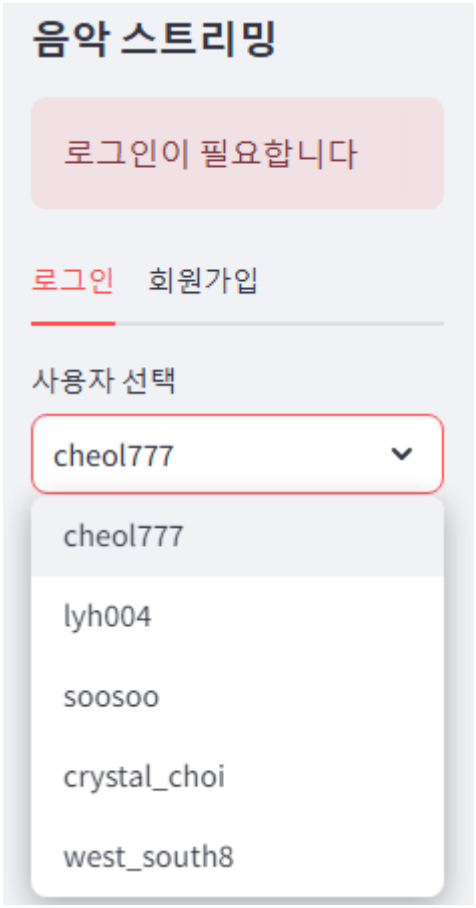
secrets.toml

```
[connections.mysql]
type = "sql"
dialect = "mysql"  # 데이터베이스의 dialect 추가
port = 3306
host = "localhost"
database = "finals_20215217_CASE02"
username = "final_exam_20215217"  # '@localhost' 포함 가능
password = "case02"
```

```python
import streamlit as st
from sqlalchemy import text
from datetime import datetime
import pandas as pd
import re

# MySQL 데이터베이스 연결
conn = st.connection('mysql', type='sql')
```

[로그인 사이드바 / 로그인 상태 메시지 출력]

[로그인 사이드바 / 사용자 선택 드롭다운]



[로그인 사이드바 / 스트림릿 코드]

```python
# 로그인 세션 상태 초기화
if 'logged_in' not in st.session_state:
    st.session_state.logged_in = False
if 'user_alias' not in st.session_state:
    st.session_state.user_alias = ""

# 사이드바 UI
with st.sidebar:
    st.header("음악 스트리밍")

    # 로그인 상태 메시지
    if not st.session_state.logged_in:
        st.error("로그인이 필요합니다")
    else:
        st.success(f"🟢{st.session_state.user_alias}님으로 로그인됨")

    # 로그인 탭, 회원가입 탭
    tab1, tab2 = st.tabs(['로그인', '회원가입'])

    # 로그인 탭
    with tab1:
        # 사용자 선택 select box, DB에서 alias를 드롭다운으로 읽어옴
        alias = conn.session.execute(text('SELECT alias FROM users'))
        selected_alias = st.selectbox("사용자 선택", alias)

        # 로그인 버튼
        if st.button('로그인', key="login_button"):
            if selected_alias:
                # 로그인 상태로 변경
                st.session_state.logged_in = True
                st.session_state.user_alias = selected_alias
                # 최상단 메시지 업데이트를 위해 rerun 호출
                st.rerun()
```

위 코드는 Streamlit UI에서 음악 스트리밍 사이드바를 생성합니다.
사이드바에는 로그인 탭과 회원가입 탭이 있으며, 로그인 탭에서는 데이터베이스에서 가져온 alias 목록을 드롭다운으로 표시해 사용자가 선택할 수 있습니다.
사용자가 alias를 선택하고 로그인 버튼을 클릭하면 session_state가 로그인 상태로 변경되고, 선택한 alias가 user_alias로 저장됩니다. 이후 st.rerun으로 페이지를 새로고침하며 사이드바 최상단에 선택한 alias로 로그인을 하였다는 성공 메시지가 사이드바의 최상단에 표시됩니다.

[회원가입 사이드바 / 회원가입 성공 메시지 출력]

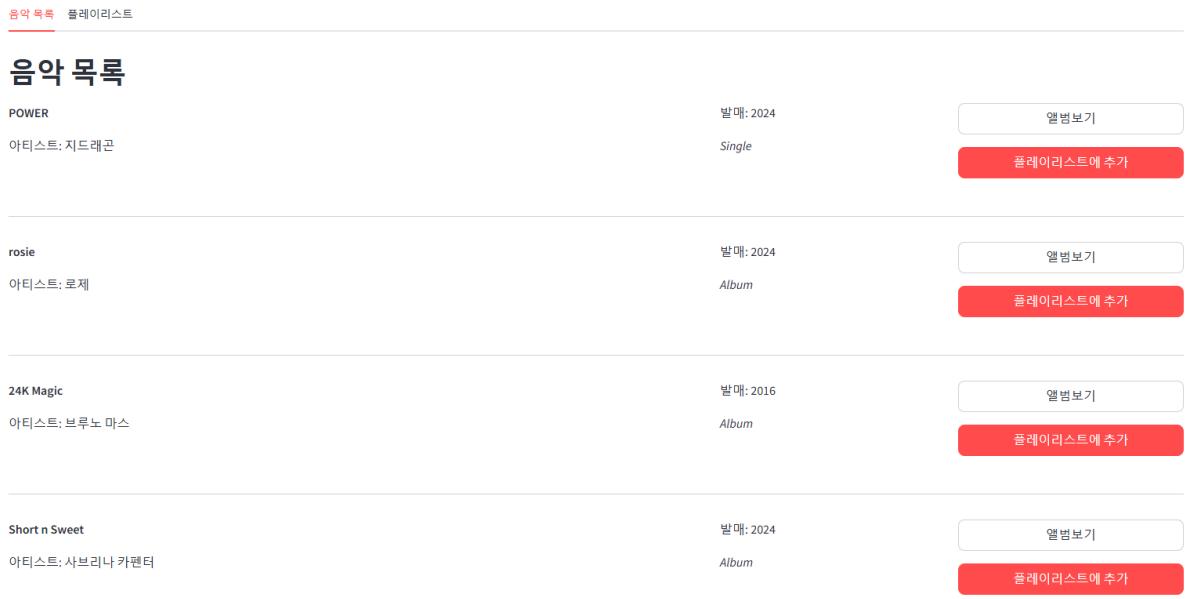
[회원가입 사이드바 / 스트림릿 코드]

```python
# 회원가입 탭
with tab2:
    name = st.text_input("이름")
    alias = st.text_input("별칭")
    email = st.text_input("이메일")
    address = st.text_input("주소")
    created_at = st.date_input("날짜를 선택하세요", value=datetime.today())

    if st.button('가입하기', key="sign_button"):
        # 입력된 데이터를 바인딩하여 쿼리 실행
        query = text('''INSERT INTO users (name, alias, email, address, join_date)
                        VALUES (:name, :alias, :email, :address, :join_date)''')

        # 쿼리 실행
        conn.session.execute(query, {
            "name": name,
            "alias": alias,
            "email": email,
            "address": address,
            "join_date": created_at
        })

        # 커밋해서 데이터베이스에 반영
        conn.session.commit()

        st.success("회원가입이 완료되었습니다!")
```

위 코드는 회원가입 탭의 UI와 동작을 구현합니다.

회원가입 탭에서 이름, 별칭, 이메일, 주소, 가입 날짜를 입력할 수 있는 입력 필드를 제공합니다. 사용자가 모든 정보를 입력한 뒤 가입하기 버튼을 클릭하면 입력된 데이터를 데이터베이스에 저장하는 쿼리가 실행됩니다. 쿼리 실행 후 데이터 변경 사항을 데이터베이스에 반영하기 위해 커밋을 수행하며, 회원가입 성공 메시지가 회원가입 버튼 아래의 화면에 표시됩니다.

[데이터 CRUD 작업 읽기 - 음악 목록 출력]

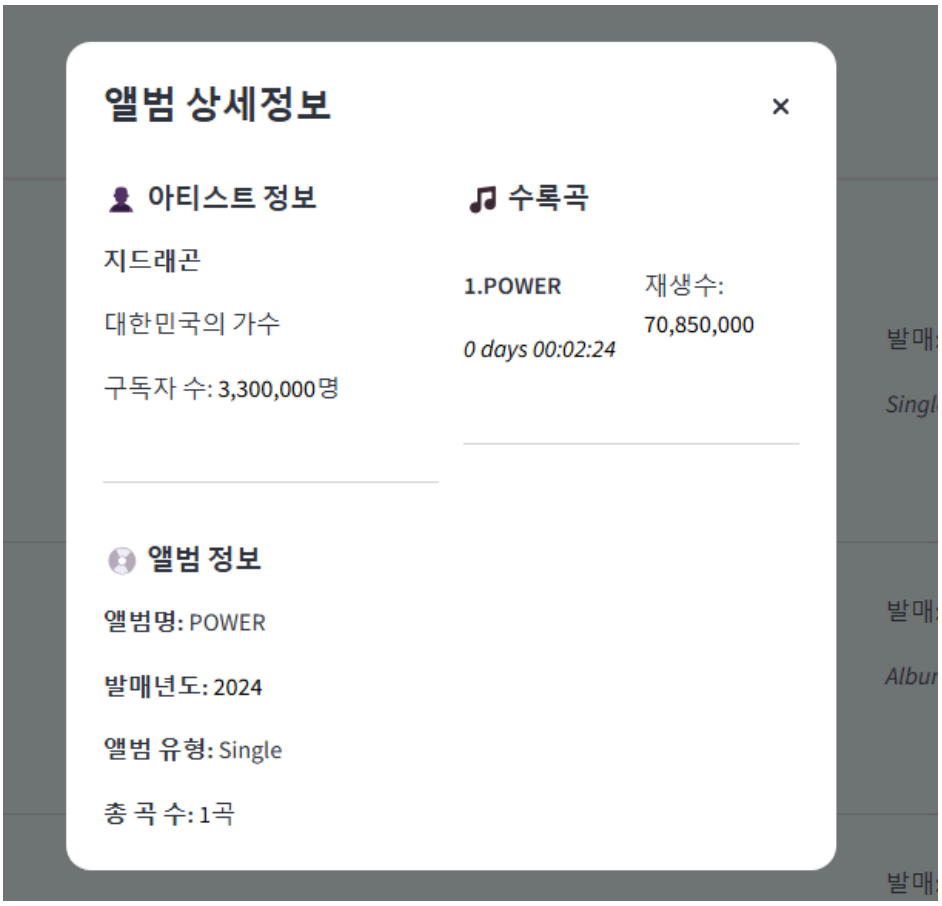


[데이터 CRUD 작업 읽기 - 음악 목록 / 스트림릿 코드]

```python
# 메인 화면 UI
# 음악 목록, 플레이리스트 탭
tab1, tab2 = st.tabs(['음악 목록', '플레이리스트'])

with tab1:
    st.header("음악 목록")

    # 음악 목록 불러오기
    # f-string을 사용하여 날짜 변수를 쿼리 내에 삽입
    query = f"""
        SELECT
            ar.name,
            al.title,
            al.release_year,
            al.album_type
        FROM artists ar
        JOIN albums al ON ar.artist_id = al.artist_id
        ORDER BY al.album_type = 'single' DESC, ar.name ASC;
    """
    df = conn.query(query, ttl=0)

    # 가수 목록 출력
    for row in df.itertuples():
        with st.container():
            col1, col2, col3, col4 = st.columns([2, 4, 2, 2])
            with col1:
                st.markdown(f"**{row.title}**") # bold체
                st.write(f"아티스트: {row.name}")
            # 두 컬럼 사이에 간격을 두고 출력하고 싶다.
            with col2:
                pass
            with col3:
                st.write(f"발매: {row.release_year}")
                st.markdown(f"*{row.album_type}*") # italic체
            with col4:
                if st.button(f"앨범보기", key=f"view_album_button_{row.Index}"):

                    show_album(row.name, row.title)
                if st.button("플레이리스트에 추가", type="primary", key=f"add_playlist_button_{row.Index}"):
                    add_playlist(row.title)
            st.divider()
```

위 코드는 메인 화면 UI에서 음악 목록과 플레이리스트 탭을 구현한 내용입니다. 음악 목록 탭에서는 데이터베이스에서 가수와 앨범 정보를 조회하여 화면에 표시합니다. 데이터는 가수 이름, 앨범 제목, 발매 연도, 앨범 유형을 포함하며, 앨범 유형이 싱글인 항목을 우선으로 정렬해 출력합니다. 각 행은 가수 이름, 앨범 정보, 발매 연도, 앨범 유형, 그리고 앨범보기, 플레이리스트 추가 버튼으로 구성된 여러 열로 나뉘어 출력됩니다. 사용자가 앨범보기 버튼을 클릭하면 해당 앨범의 상세 정보를 표시하는 함수가 실행됩니다.
플레이리스트에 추가 버튼을 클릭하면 선택한 앨범이 플레이리스트에 추가됩니다.
각 데이터 행 사이에는 st.divider로 구분선을 사용해 항목들을 시각적으로 구분합니다.

[데이터 CRUD 작업 읽기 - 앨범보기 버튼클릭해서 앨범 상세정보 팝업창 출력]



[데이터 CRUD 작업 읽기 - 앨범 상세보기 코드 작동 방식]

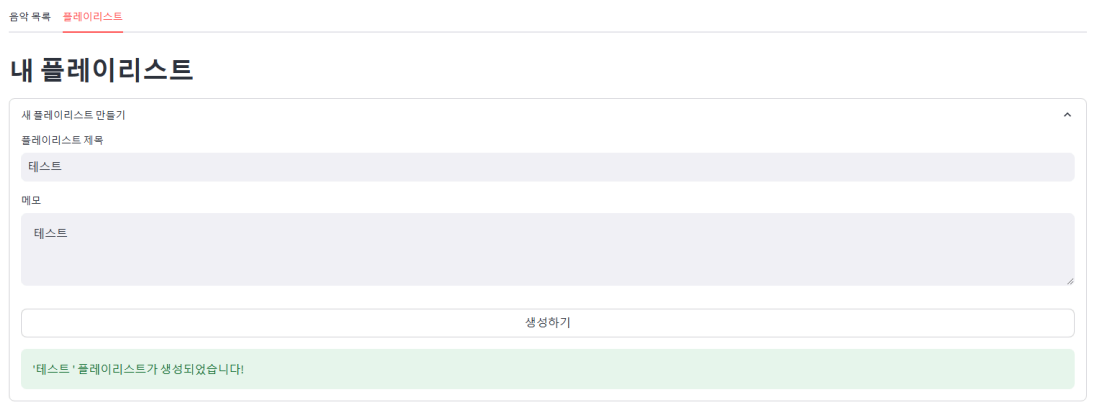
1. 앨범보기 버튼을 누르면 show_album()함수를 호출합니다.
2. show_album()함수는 인자로 입력받은 아티스트의 이름과 앨범 타이틀로 데이터베이스에서 아티스트와 해당 앨범 및 수록곡의 정보를 조회합니다.
3. 아티스트의 이름으로 앨범에 수록된 곡들의 타이틀, 재생 시간, 재생수 조회합니다.
4. 앨범 타이틀로 트랙 타이틀, 재생시간, 재생수를 조회합니다.
5. 조회된 두 결과를 각각 pandas DataFrame으로 변환합니다.
6. 화면을 두 열로 나누어 첫 번째 열에는 아티스트 정보와 앨범 정보를 출력합니다.
7. 아티스트 이름, 설명, 구독자 수와 앨범명, 발매 연도, 앨범 유형, 총 곡 수를 구분자로 나누어서 표시합니다.
8. 두 번째 열에는 수록곡 정보를 출력하며, 각 곡은 번호, 제목, 재생 시간, 재생 횟수로 구성됩니다.
9. 수록곡 간에는 구분선을 추가하여 목록의 가독성을 높입니다.

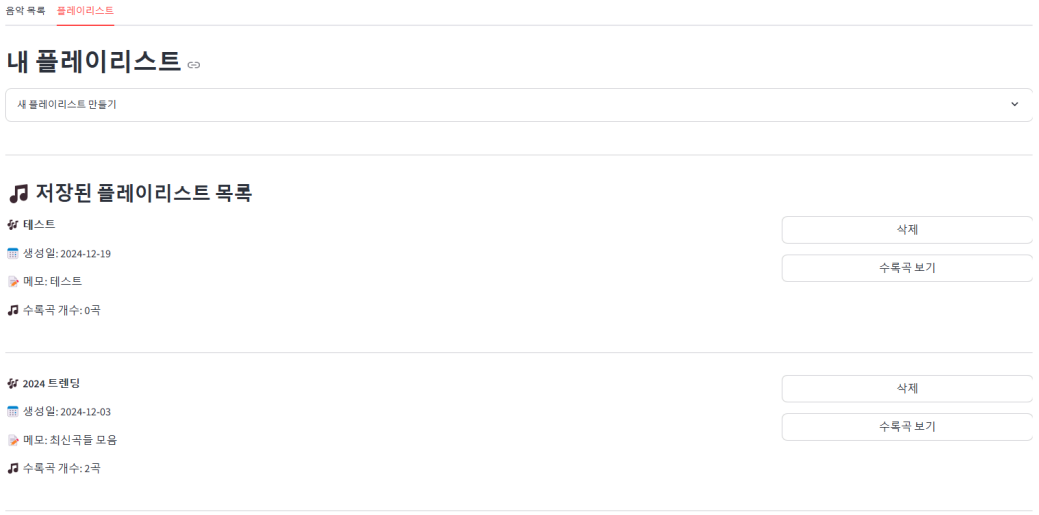
[데이터 CRUD 작업 읽기 - show_album() 스트림릿 코드]

```python
# 앨범 상세정보를 띄우는 다이얼로그
@st.dialog("앨범 상세정보")
def show_album(name, title):
    with conn.session as s:
        query = text("""
            SELECT
                ar.name,
                ar.artist_desc,
                ar.subscriber_count,
                al.title,
                al.release_year,
                al.album_type,
                al.total_tracks
            FROM artists ar
            JOIN albums al ON ar.artist_id = al.artist_id
            WHERE ar.name = :name
        """)
        query_2 = text("""
            SELECT
                a.title,
                t.title as track_title,
                t.duration,
                t.total_play
            FROM tracks t
            JOIN albums a ON t.album_id = a.album_id
            WHERE a.title = :title
        """)
        result_a = conn.session.execute(query, {"name": name})
        result_t = conn.session.execute(query_2, {"title": title})
        # 결과를 pandas DataFrame으로 변환
        df_a = pd.DataFrame(result_a.fetchall(), columns=result_a.keys())
        df_t = pd.DataFrame(result_t.fetchall(), columns=result_t.keys())
```

```python
with st.container():
    col1, col2 = st.columns([1, 1])
    with col1:
        for row in df_a.itertuples():
            st.subheader("**👤 아티스트 정보**")
            st.markdown(f"**{row.name}**") # bold체
            st.write(row.artist_desc)
            st.markdown(f"구독자 수: <span style='color:black;'>{row.subscriber_count:,}</span>명", unsafe_allow_html=True)
            st.divider()
            st.subheader("**💿 앨범 정보**")
            st.write("**앨범명:**", row.title) # bold체
            st.markdown(f"**발매년도:** <span style='color:black;'>{ row.release_year}</span>", unsafe_allow_html=True)
            st.write("**앨범 유형:**", row.album_type) # bold체
            st.markdown(f"**총 곡 수:** <span style='color:black;'>{ row.total_tracks}</span>곡", unsafe_allow_html=True)
    with col2:
        st.subheader("**🎵 수록곡**")
        col1, col2 = st.columns([1, 1])
        for idx, row in enumerate(df_t.itertuples(index=True), start=1):
            col3, col4 = st.columns([1, 1])
            with col3:
                st.write(f"**{idx}.{row.track_title}**") # 1.title 형식으로 출력되게
                st.markdown(f"*<span style='color:black;'>{row.duration}</span>*", unsafe_allow_html=True)
            with col4:
                st.markdown(f"재생수: <span style='color:black;'>{row.total_play:,}</span>", unsafe_allow_html=True)
            st.divider()
```

[데이터 CRUD 작업 쓰기 - 새 플레이리스트 만들기 및 저장된 플레이리스트 출력 화면]





[데이터 CRUD 작업 쓰기 - 새로운 플레이리스트 만들기 코드 작동 방식]

1. "새 플레이리스트 만들기" 확장 메뉴를 사용하여 사용자로부터 플레이리스트 제목과 메모를 입력받습니다.
2. 로그인한 상태에서 로그인 세션의 alias에 해당하는 user_id를 조회해서 가지고옵니다.
3. 제목 입력창과 메모 입력창이 제공되며, 텍스트 입력창 크기에 맞춘 버튼 스타일을 적용합니다.
4. "생성하기" 버튼을 클릭하면, 제목이 입력되지 않으면 경고 메시지가 표시됩니다.
5. 제목이 입력되면 데이터베이스에 새 플레이리스트를 생성하는 쿼리가 실행됩니다.
6. 생성된 플레이리스트의 제목과 메모는 데이터베이스에 저장되고, 성공 메시지가 사용자에게 표시됩니다.
7. 오류가 발생하면 오류 메시지를 출력합니다.

8. 화면 상단에 구분선을 추가하여 저장된 플레이리스트 목록과의 구분을 제공합니다.

[데이터 CRUD 작업 쓰기 - 새로운 플레이리스트 작성 코드]

```
with tab2:
    st.header("내 플레이리스트")

    # "새 플레이리스트 만들기" 확장 메뉴
    with st.expander("새 플레이리스트 만들기"):
        playlist_input = st.text_input("플레이리스트 제목", placeholder="플레이리스트 제목을 입력하세요")
        memo_input = st.text_area("메모", placeholder="플레이리스트에 대한 메모를 입력하세요")

        # 텍스트 입력창 크기에 맞는 버튼 스타일 정의
        st.markdown(
            """
            <style>
            .stButton > button {
                width: 100%;
                height: 40px;
                font-size: 16px;
            }
            </style>
            """,
            unsafe_allow_html=True
        )
        # 로그인한 사용자의 user_id를 가져오는 쿼리
        user_id = None
        if st.session_state.logged_in:
            user_alias = st.session_state.user_alias
            with conn.session as session:
                result = session.execute(
                    text("""SELECT user_id FROM users WHERE alias = :alias"""),
                    {"alias": user_alias}
                )
                user_id = result.scalar()  # 첫 번째 컬럼값(user_id)만 가져옴
```

```
        # '생성하기' 버튼 클릭 시 실행
        if st.button("생성하기"):
            if not playlist_input.strip():
                st.warning("플레이리스트 제목을 입력하세요!")
            else:
                with conn.session as session:
                    try:
                        session.execute(
                            text("""
                            INSERT INTO playlists (user_id, title, memo, created_date)
                            VALUES (:title, :memo, NOW())
                            """),
                            {"user_id": user_id, "title": playlist_input.strip(), "memo": memo_input.strip()}
                        )
                        session.commit()
                        st.success(f"'{playlist_input}' 플레이리스트가 생성되었습니다!")
                    except Exception as e:
                        st.error(f"플레이리스트 생성 중 오류가 발생하였습니다: {str(e)}")

    st.divider()  # 시각적 구분선
```

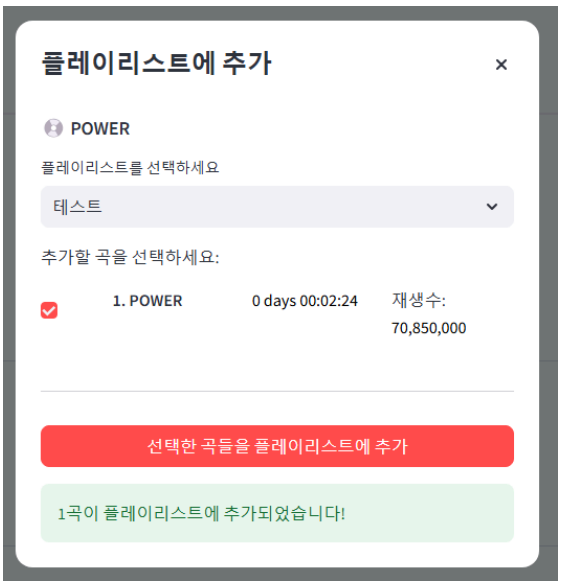
[데이터 CRUD 작업 읽기 - 저장된 플레이리스트 출력 코드 작동방식]

1. 플레이리스트 탭으로 전환하면 데이터베이스에서 플레이리스트 목록과 각 플레이리스트에 포함된 수록곡 개수를 조회합니다.
2. 조회된 결과가 없으면 저장된 플레이리스트가 없다는 메시지를 출력합니다.
3. 결과가 있으면 각 플레이리스트의 제목, 생성일, 메모, 수록곡 개수를 화면에 출력합니다.
4. 삭제 버튼을 누르면 해당 플레이리스트를 데이터베이스에서 삭제하고 변경 사항을 저장합니다.
5. 수록곡 보기 버튼을 누르면 해당 플레이리스트의 수록곡 상세 정보를 표시하는 다이얼로그가 실행됩니다.
6. 각 플레이리스트 항목 사이에는 구분선을 추가하여 항목 간 경계를 명확히 합니다.
7. 오류가 발생하면 사용자에게 에러 메시지를 출력합니다.

[데이터 CRUD 작업 읽기 - 플레이리스트 보기 스트림릿 코드]

```python
# 플레이리스트 목록 및 수록곡 개수 조회
st.subheader("🎵 저장된 플레이리스트 목록")
with conn.session as session:
    try:
        # 플레이리스트와 수록곡 개수 조회
        result = session.execute(
            text("""
            SELECT
                p.title AS playlist_title,
                p.created_date,
                p.memo,
                COUNT(pt.track_id) AS track_count
            FROM playlists p
            LEFT JOIN playlists_tracks pt ON p.playlist_id = pt.playlist_id
            GROUP BY p.playlist_id
            ORDER BY p.created_date DESC
            """)
        )
        playlists = result.fetchall()

    if not playlists:
        st.info("저장된 플레이리스트가 없습니다. 새 플레이리스트를 만들어 보세요!")
    else:
        # 플레이리스트 출력
        for playlist in playlists:
            with st.container():
                col1, col2 = st.columns([3, 1])
                with col1:
                    st.markdown(f"**🎶 {playlist.playlist_title}**")
                    st.markdown(f"📅 생성일: {playlist.created_date.strftime('%Y-%m-%d')}")
                    if playlist.memo:
                        st.markdown(f"📝 메모: {playlist.memo}")
                    st.markdown(f"🎵 수록곡 개수: {playlist.track_count}곡")
                with col2:
                    if st.button("삭제", key=f"delete_{playlist.playlist_title}"):
                        try:
                            # 플레이리스트 삭제
                            session.execute(
                                text("""
                                DELETE FROM playlists WHERE title = :title
                                """),
                                {"title": playlist.playlist_title}
                            )
                            session.commit()
                            st.success(f"'{playlist.playlist_title}'이 삭제되었습니다!")
                        except Exception as e:
                            st.error(f"삭제 중 오류가 발생하였습니다: {str(e)}")

                    if st.button("수록곡 보기", key=f"view_{playlist.playlist_title}"):
                        show_dialog(playlist.playlist_title)

            st.divider()  # 플레이리스트 간 구분선

except Exception as e:
    st.error(f"플레이리스트 조회 중 오류가 발생하였습니다: {str(e)}")
```

[데이터 CRUD 작업 쓰기 - 플레이리스트에 추가버튼 클릭화면]



[데이터 CRUD 작업 쓰기 - 플레이리스트에 추가 코드 작동 방식]

1. 음악 목록의 플레이리스트 추가 버튼을 클릭하면 add_playlist 함수가 호출됩니다.
2. add_playlist 함수는 플레이리스트에 트랙을 추가하는 다이얼로그로, 함수를 호출한 앨범의 트랙 목록을 불러와서 플레이리스트에 추가할 수 있도록 합니다.
3. 사용자가 앨범 제목을 기반으로 트랙을 조회하고, 플레이리스트를 선택하는 드롭다운 목록을 표시합니다.
4. 트랙은 체크박스로 선택할 수 있으며, 선택된 트랙들의 ID는 리스트에 저장됩니다.
5. "선택한 곡들을 플레이리스트에 추가" 버튼을 클릭하면 add_playlist_query 함수가 호출되고 선택된 트랙들이 지정된 플레이리스트에 추가됩니다.
6. add_playlist_query 함수는 선택된 트랙들을 플레이리스트에 추가하는 역할을 하며, 트랙 ID를 사용해 데이터베이스에 playlists_tracks 테이블에 여러 곡을 한 번에 추가합니다.
7. 트랙을 성공적으로 추가하면 추가된 곡 수를 알려주는 메시지가 출력되며, 트랙이 선택되지 않은 경우 경고 메시지가 표시됩니다.
8. 데이터베이스에서 트랙과 플레이리스트 정보를 가져오는 과정에서 오류가 발생하면 오류 메시지가 출력됩니다.

[데이터 CRUD 작업 - 플레이리스트 추가 add_playlist_query 함수 코드]

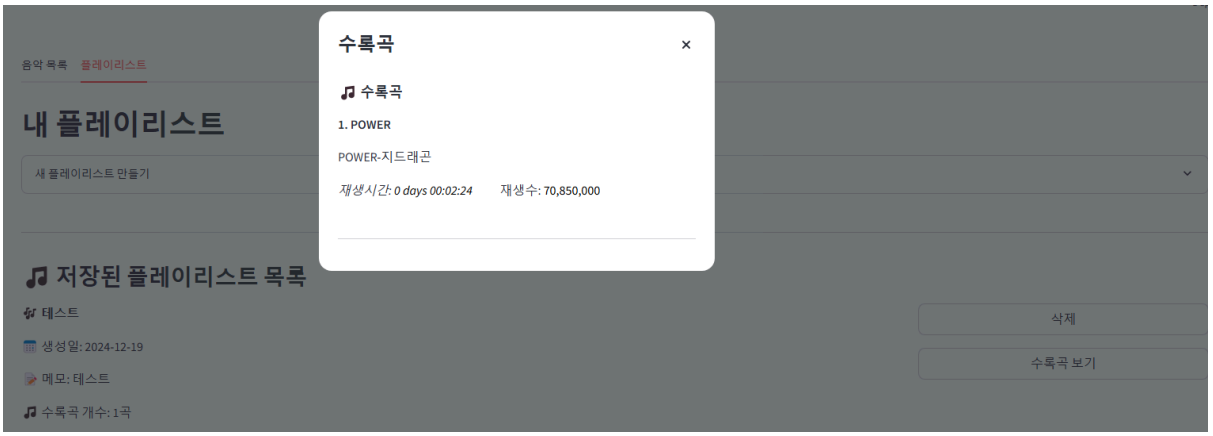
```python
# 플레이리스트에 선택한 트랙을 추가하는 함수
def add_playlist_query(playlist_id, track_ids):
    with conn.session as session:
        try:
            # 여러 track_id를 한 번에 추가하는 쿼리 작성
            session.execute(
                text("""
                INSERT INTO playlists_tracks(playlist_id, track_id)
                VALUES (:playlist_id, :track_id)
                """),
                [{"playlist_id": playlist_id, "track_id": track_id} for track_id in track_ids]
            )
            session.commit()
        except Exception as e:
            raise Exception(f"저장 중 오류가 발생하였습니다: {str(e)}")
```

[데이터 CRUD 작업 - 플레이리스트 추가 add_playlist 함수 코드]

```python
# 플레이리스트에 트랙을 추가하는 팝업창을 띄우는 다이얼로그
@st.dialog("플레이리스트에 추가")
def add_playlist(title):
    with conn.session as s:
        # 트랙 조회 쿼리
        result_t = s.execute(
            text("""
            SELECT
                t.track_id,
                t.title AS track_title,
                t.duration,
                t.total_play
            FROM tracks t
            JOIN albums a ON t.album_id = a.album_id
            WHERE a.title = :title
            """),
            {"title": title}
        )
        # 플레이리스트 조회 쿼리
        result_p = s.execute(
            text("""
            SELECT
                playlist_id,
                title
            FROM playlists
            """)
        )
        playlists = list(result_p.fetchall())
        options = [re.sub(r"['\"]", '', item[1]) for item in playlists]  # playlists의 title을 options로 사용
        playlist_ids = [item[0] for item in playlists]  # playlist_id 값을 별도로 저장

        # 결과를 pandas DataFrame으로 변환
        df_t = pd.DataFrame(result_t.fetchall(), columns=result_t.keys())
```

```python
with st.container():
    st.subheader(f"💿 {title}")  # 앨범 제목 출력
    selected_option = st.selectbox("플레이리스트를 선택하세요", options)  # 플레이리스트 선택
    selected_playlist_id = playlist_ids[options.index(selected_option)]  # 선택된 playlist_id 가져오기

    st.write("추가할 곡을 선택하세요:")
    track_ids = []  # 체크된 track_id들을 저장할 리스트
    for idx, row in enumerate(df_t.itertuples(index=True), start=1):
        col1, col2, col3, col4 = st.columns([0.5, 1, 1, 1])
        with col1:
            # 체크박스의 값으로 track_id를 사용
            checked = st.checkbox(label=' ', key=f"track_{row.track_id}")  # 고유 키 생성
            if checked:
                track_ids.append(row.track_id)  # 체크된 track_id 저장
        with col2:
            st.write(f"**{idx}. {row.track_title}**")  # 1. title 형식으로 출력
        with col3:
            st.markdown(f"<span style='color:black;'>{row.duration}</span>", unsafe_allow_html=True)
        with col4:
            st.markdown(f"재생수: <span style='color:black;'>{row.total_play:,}</span>", unsafe_allow_html=True)

st.divider()

# '선택한 곡들을 플레이리스트에 추가' 버튼 클릭 시
if st.button("선택한 곡들을 플레이리스트에 추가", type="primary"):
    if track_ids:  # 선택된 곡이 있는 경우
        try:
            # 플레이리스트에 선택한 곡 추가
            add_playlist_query(selected_playlist_id, track_ids)

            # 성공 메시지 출력
            st.success(f"{len(track_ids)}곡이 플레이리스트에 추가되었습니다!")
        except Exception as e:
            # 오류 발생 시 에러 메시지 출력
            st.error(str(e))
    else:
        # 선택된 곡이 없을 경우 경고 메시지 출력
        st.warning("추가할 곡을 선택하세요!")
```

[데이터 CRUD 작업 읽기 - 플레이리스트 수록곡 보기 화면]



[데이터 CRUD 작업 읽기 - 수록곡 보기 버튼 클릭 코드 작동 방식]

1. 수록곡 보기 버튼을 클릭하면 주어진 플레이리스트 제목을 기준으로 데이터베이스에서 해당 플레이리스트에 포함된 수록곡 정보를 조회합니다.
2. 각 곡에 대한 트랙 제목, 재생 시간, 재생 횟수, 아티스트 이름을 가져옵니다.
3. 조회된 데이터를 pandas DataFrame으로 변환합니다.
4. 화면에 수록곡 제목과 아티스트 이름을 출력하며, 트랙 번호를 함께 표시합니다.
5. 트랙의 재생 시간과 재생 횟수는 가로로 나누어 두 열에 표시합니다.
6. 각 트랙 간에는 구분선을 추가하여 항목을 명확히 구분합니다.
7. 트랙 목록은 가독성을 높이기 위해 세로와 가로 배치로 구성됩니다.

[데이터 CRUD 작업 읽기 - 수록곡 보기 스트림릿 코드]

```python
#플레이리스트의 수록곡 정보
@st.dialog("수록곡")
def show_playlist_tracks(playlist_title):
    with conn.session as s:
        query_2 = text("""
            SELECT
                t.title AS track_title,
                t.duration,
                t.total_play,
                ar.name AS artist_name
            FROM tracks t
            LEFT JOIN playlists_tracks pt ON pt.track_id = t.track_id
            LEFT JOIN playlists p ON p.playlist_id = pt.playlist_id
            LEFT JOIN albums a ON a.album_id = t.album_id
            LEFT JOIN artists ar ON ar.artist_id = a.artist_id
            WHERE p.title = :playlist_title;
        """)
        result_t = conn.session.execute(query_2, {"playlist_title": playlist_title})
        # 결과를 pandas DataFrame으로 변환
        df_t = pd.DataFrame(result_t.fetchall(), columns=result_t.keys())

        with st.container():
            st.subheader("** 🎵 수록곡**")
            for idx, row in enumerate(df_t.itertuples(index=True), start=1):
                col1, col2 = st.columns([4, 1])  # 첫 번째 컬럼은 좁게, 두 번째 컬럼은 넓게

                with col1:
                    # 트랙 제목과 아티스트 이름을 세로로 표시
                    st.write(f"**{idx}. {row.track_title}**")  # 트랙 제목
                    st.write(f"{row.track_title}-{row.artist_name}")        # 아티스트 이름
                    col3, col4 = st.columns([5, 4])
                    with col3:
                    # 재생시간과 재생수를 가로로 표시
                        st.markdown(f"*재생시간: <span style='color:black;'>{row.duration}</span>*", unsafe_allow_html=True)
                    with col4:
                        st.markdown(f"재생수: <span style='color:black;'>{row.total_play:,}</span>", unsafe_allow_html=True)

                with col2:
                    pass

                st.divider()  # 각 트랙마다 구분선 추가
```

7. Section 7 에러 및 해결 과정 (2점)

테스트 플레이리스트 삭제 시 오류발생

원인 : playlists_tracks 테이블에서 playlist_id가 playlists 테이블의 playlist_id를 참조하고 있기 때문에, playlists 테이블에서 해당 playlist_id를 삭제하려면 먼저 playlists_tracks 테이블에서 그 참조를 제거해야 합니다. 즉, 외래 키 제약에 의해 부모 테이블인 playlists에서 삭제를 시도할 때, 자식 테이블인 playlists_tracks에 연관된 데이터가 남아 있으면 삭제가 불가능합니다.

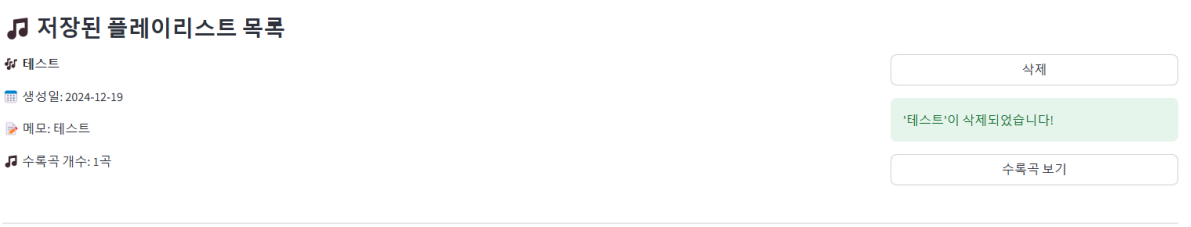
해결방법 : playlists_tracks 테이블의 playlist_id 기존 외래키 삭제 후 제약조건이 on delete cascade 옵션을 가진 playlist_id 외래키를 새로 생성

```
1  USE finals_20215217_CASE02;
2
3  ALTER TABLE playlists_tracks
4  DROP FOREIGN KEY playlists_tracks_ibfk_2;
5
6  ALTER TABLE playlists_tracks
7  ADD CONSTRAINT playlists_tracks_ibfk_2
8  FOREIGN KEY (playlist_id) REFERENCES playlists(playlist_id)
9  ON DELETE CASCADE;
```

Output

Action Output

#	Time	Action	Message
1	18:53:33	USE finals_20215217_CASE02	0 row(s) affected
2	18:53:33	ALTER TABLE playlists_tracks DROP FOREIGN KEY playlists_tracks_ibfk_2	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3	18:53:33	ALTER TABLE playlists_tracks ADD CONSTRAINT playlists_tracks_ibfk_2 FOREIGN KEY (playlist_id) REFERENCES playlists(playlist_id) ON DEL...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0



gpt 사용 내역

1. **st.dialog** 사용방법

dialog를 통해 팝업창을 띄우는 것을 몰랐는데 **gpt**를 통해 **dialog**로 팝업창을 띄우는 방식을 알게되어 앨범 상세정보 창을 제작하는데 도움이 되었다**.**

2. 화면비율 조정

set_page_config을 조작하면 화면의 레이아웃 비율을 조절할 수 있음을 알게되었음

3. **tab** 선택화면

tab으로 선택 화면을 제공하는 방법을 알게되었음

4. 버튼 비율조정

markdown을 통해서 **css**로 버튼비율을 원하는 크기로 조절하는 방법을 알게되었음