

Python, MYSQL을 이용한 어플리케이션 제작

사용 언어

Pyhon, Mysql(Python에서는 **Pymysql** 라이브러리를 이용하여 mysql 활용)

성명

2016032897 산업공학과 김기범

어플리케이션 기능의 설명

가상의 스트리밍 구독자에게 음원 제공 서비스를 제공하는 어플리케이션을 만든다. 스트리밍 구독자는 여러 개의 **플레이리스트**를 만들 수 있고, 음원을 넣을 수 있다. 플레이리스트를 삭제 하거나 추가할 수 있다. 플레이리스트에 음원의 목록을 보고 선택 할 수 있다. 음원을 선택하게 된다면 이를 실제에서는 **스트리밍**한다고 생각하여 스트리밍 횟수가 증가한다. 증가된 횟수는 곧 실시간 차트에도 반영되고, 통계치에도 반영된다. 스트리밍 될 때는 개인의 결제날짜를 반영하여 '1분 미리듣기', '전체 듣기'의 기능이 따로 주어진다. 그리고 가입시에 가입한 주민등록번호 앞자리로 나이를 계산하여 나이대 별 선호하는 노래 통계치에 영향을 준다.

하지만 이런 기능들을 이용하기 위해서는 회원가입을 하고 **로그인** 절차를 밟아야 한다. 스트리밍 구독자는 로그인하게 되면 실시간 차트로 10개의 음원을 볼 수 있다. 플레이리스트 뿐만 아니라 음원 검색을 할 수 있다. 음원을 찾고 싶을 때는 **음원명,가수명(그룹 포함),작사/작곡가** 이름을 음원을 찾을 수 있다. 음원을 찾고 나면 **퍼센트** 통계치를 제공하여 어느 나이대가 선호하는 노래인지의 정보를 제공한다.

음원을 스트리밍하게 되면 나이대에 맞는 스트리밍 횟수가 증가하게 된다. 이 통계치를 활용하여 10대 이하,10대~40대, 50대 이상 선호하는 노래를 상위에서 원하는 갯수만큼 볼 수 있다. 나이대별로 노래 순위를 제공하면 그 즉시 스트리밍할 수 있으며 플레이리스트에 추가할 수 있는 기능을 제공한다.

스트리밍 구독자는 개인 정보를 변경하거나 결제하기의 기능을 수행할 수 있다. 결제하기 기능을 이용하면, 현재 날짜로 결제날짜가 바뀌며 30일간 "전체듣기"가 가능해진다.

관리인은 위의 스트리밍 구독자의 기능을 전부 이용할 수 있을 뿐만 아니라, 음원, 아티스트, 작사/작곡가 정보를 추가하거나 삭제할 수 있다. 그리고 전체 정보 열람도 가능하다. **관리인**은 음원 뿐만 아니라 스트리밍 구독자 관리도 가능하다. 삭제하고 싶은 구독자 계정을 삭제할 수 있으며, 의도적으로 결제날짜를 연장할 수 있다. 또한, 전체 스트리밍 구독자의 정보를 열람이 가능하다.

데이터베이스 구조의 변경

- **View**를 2개 생성했다. View를 통해 쿼리 코드를 단순화시키는데 목적을 두었다. **soundtrack_artist_2** View는 soundtrack(음원정보의 table),artist(아티스트=가수의 table), sing(**M:N** 관계를 유지시키는 table) table를 join하여 음원 정보와 그 음원을 발매한 가수의 정보를 같이 가져온 table이다. 즉, 노래의 제목 등의 정보와 가수의 그룹 여부, 데뷔날짜 등의 attribute를 모아놓은 table이다. 음원과 가수의 정보를 가져올 때 일일이 Join문을 입력해야하기 때문에 코드 단순화를 위해 생성했다.
다른 하나의 View는 **soundtrack_compose_write** view다. 이 View는 음원과 작곡/작사가의 정보를 Join한 결과를 나타내는 Table이다. 작사/작곡가의 이름, 음원명 등의 Attribute를 갖는다.
View를 사용함으로써 음원과 아티스트, 음원과 작곡/작사의 정보를 가져올 때마다의 table를 **Join**해야 하는데 간결하게 표현했다.
- 과제 3에서는 음원과 Composer/Writer를 1:N관계로 형성했었다. 하지만, 한 작곡가는 여러 음악을 작곡하고 한 음원은 **여러** 작곡가에 의해 작곡되거나 작사되어질 수 있다고 판단했다. 따라서 M:N관계로 형성하기 위해 Composer/Writer table과 soundtrack(음원) 사이에 **compose_write** table을 만들었다. 그리고

편곡되어지는 경우는 작곡,작사 되는 경우보다 현저히 적고 스트리밍될 때 편곡자의 정보는 중요하지 않아 제외시켰다. 그리고 작곡/작사 table에서 별도의 Index PK를 생성하지 않고, 작곡/작사가의 이름과 생년월일을 합쳐서 **Primary Key**로 만들었다. 이렇게 하여 composer/writer의 table에 저장용량을 낮췄다.

- 음원관리자의 table을 제거시키고 스트리밍 구독자 중 한명으로 포함시켰다. 음원관리자도 음악을 스트리밍 할 수 있고, 유저의 기능을 이용할 수 있게 하기 위해 하나의 table로 합쳤다. 단지 차이점은 관리인이 아닌 스트리밍 구독자에게 하나의 Foreign Key(Super_Id)를 생성시켜 스트리밍 구독자 테이블에서 관리인과 관계를 맺도록 했다. 이 방법으로 음원관리자와 관리인을 구별 할 수 있게 되었고, '음원관리자'라는 별도의 테이블을 만들지 않아 저장 낭비를 줄였다. 따라서, 스트리밍 구독자 table에는 **M_SSN**이 사라지고 **Super_Id**가 형성되었다. 음원관리자 table이 없어지면서 음원 table에서의 **M_SSN** Foreign key를 없앴다. 음원과 음원관리자는 별도의 관계가 필요없다고 판단하여 **Relation**을 안했다.
 - Soundtrack(음원)의 경우에는 별도의 PK를 위해 Index를 생성했었지만, Title(제목)과 Sing_time(노래 시간)을 합쳐 PK로 만들었다. 제목은 동일 할 수 있지만, 노래 시간까지 같은 음원은 존재하지 않아 PK로 잡았다. 이렇게 하여 하나의 Attribute에 저장되는 값을 줄여 효율성을 높였다. 음원의 PK가 2개 되면서 Playlist,compose/write,재생횟수(**play_count**),Sing에 FK를 2개씩 형성시켰다. 추가적으로 음원에서 **Yesterday_Streaming** Attribute는 구현 과정 중에 큰 역할을 하지 않아 제거했다.
- 과제 3 음원 안에 앨범 관련 Attribute가 있었다. 하지만 어플리케이션 구현 과정 중에 앨범 관련 정보가 활용되지 않아 제거했다. 또한, 앨범 정보는 스트리밍 구독자에게 큰 유용한 정보를 제공하지 않은 것으로 판단하여 앨범 관련 Attribute(n집,앨범이름,타이틀곡,수록곡)은 제외시켰다. 그리고 가사를 Attribute에 **varchar**로 넣기에는 너무 큰 용량을 차지하여 제외시켰다. 효율적인 저장 공간 활용을 위해 제외시켰다.

Soundtrack(음원) table에서 음원이 그룹으로 발매된 것인지 솔로로 발매된 것인지 구분이 필요했다. 그 구분을 위해 **is_Group** Attribute를 생성하여 해당 음원이 그룹가수에게 발매된 것인지 알게 했다.

- Artist table의 경우에도 인위적인 Index를 제외시키고 아티스트 **이름과 데뷔날짜**를 합쳐 **PK**로 이용했다. Storage에서 인위적인 Index를 제거하고 존재하던 Attribute로 PK를 형성하여 저장공간 활용도를 높이기 위해서다. 그리고 국적의 정보가 스트리밍 구독자에게 큰 유용한 정보를 제공하지 않아 제외시켰다.
- Artist table에서 솔로/그룹 Attribute에 그룹이면 그룹명의 값을 들어가게 하고 솔로면 'solo_'값을 넣어 해당 가수가 그룹에 속한지, 솔로인지 구분지었다. 따로 Attribute를 안 만들고 기존 Plan에서의 Attribute를 활용했다.
- Playlist table에서 플레이리스트 내 유저가 많이 듣는 음원을 추출하기 위해 **listen_count** Attribute를 생성시켰다. 추가적인 기능을 위해 Attribute를 만들었다.
 - 스트리밍 구독자가 플레이리스트를 만들었을 때 어떠한 음원이 안 담겨있다. 만약 과제 3처럼 별도의 playlist 이름 목록(table)을 안 만들면 playlist table에서 플레이리스트에 음원이 존재하지 않아 음원의 PK를 가리키지 못하여 table 내의 PK를 형성하지 못하게 된다. 따라서, 스트리밍 구독자에게 별도의 플레이리스트 이름이 담긴 table이 필요하다고 생각하여 playlist_name의 table을 형성했다. 그리고 Playlist table에서 같은 유저의 같은 이름의 플레이리스트에서 같은 음원은 존재하지 않는다. 따라서, 인위적으로 생성한 Index는 제거해도 무방하다고 생각하여 제외시켰다.

프로그램 코드 설명 및 화면 캡처

SQL_Query.py

이 파일은 어플리케이션 실행과 관련된 모든 Query를 포함하고 있다. 그리고 아래는 각 쿼리의 설명이다.

login_query는 데이터베이스에 Id와 Password가 존재하는지 확인하는 쿼리다. Id와 Password를 **Select**를 통해 있는지 확인한다.

-> select 사용

realtime_get_query는 음원 데이터베이스에서 **하루 들은 횟수**를 기준으로 **ORDER BY**하여 상위 10개의 음원만 보여준다. 이는 **실시간 음원 차트**를 구현하기 위해서 데이터베이스에서 불러오는 쿼리다. 음원 제목을

Group by하여 같은 음원을 부른 가수(듀엣 등)가 그룹으로 묶여지게 된다. 이를 가수명을 **group_concat**을 통해 같은 음원을 부른 가수가 여러명 존재하면 하나의 튜플로 압축한다. 위에서 언급한 **View**를 사용해도 되지만, 이번에는 sing table, soundtrack table, artist table를 직접 **join**하여 하나의 table로 생성시켰다.

-> select, join, group_concat, groupby, order by desc 사용

singup_query는 회원가입시 사용되는 쿼리다. 회원정보를 모두 기입하면 Insert를 통해 table에 **tuple**을 넣는다.

-> insert 사용

is_id_exist는 Id가 다른 사용자와 중복되어 만들어지지 않기 위해 Id **중복 검사**하는 쿼리다. **Select**를 통해 존재하는지 확인한다. -> select 사용

show_playlist_name은 한 사용자의 **플레이리스트 이름 목록**을 가져오는 쿼리로, 유저의 플레이리스트가 어떤 것이 존재하는지 확인 할 때 사용하는 쿼리다. 스트리밍 유저의 Id를 통해 플레이리스트 목록을 **Select**를 통해 가져온다.

-> select 사용

is_playlist_name_exist는 플레이리스트를 만들기 전에 이미 같은 이름의 플레이리스트 이름이 있는지 확인하는 쿼리다. **중복검사**의 기능을 한다. 유저의 ID와 플레이리스트 이름으로 **Select**를 통해 확인한다.

-> select 사용

show_playlist_music은 유저의 플레이리스트에서 **음원 목록**을 가져오는 쿼리다. **soundtrack_artist_2**와 **playlist** table를 join하여 유저의 플레이리스트에 있는 음원의 정보를 가져오는 쿼리다. **soundtrack_artist_2**는 위에서 언급한 View에 해당한다.

-> select, join 사용

show_music_detail은 **soundtrack_artist_2** table에서 원하는 Primary Key인 음원제목과 음원 시간으로 가져오는 쿼리다. **soundtrack_artist_2**는 view로 *데이터베이스 구조의 변경*을 통해 확인 할 수 있다.

-> select 사용

plus_own_soundtrack은 노래를 클릭했을 때, 들은 것으로 간주하여 자신의 playlist에서 **들은 횟수**를 **Update**를 통해 증가시킨다. 이 쿼리는 나중에 유저의 한 플레이리스트에서 많이 들었던 곡 순위를 가져오는데 영향을 미친다.

-> update 사용

plus_today_soundtrack은 바로 위의 쿼리와 다르게 한 음원이 하루동안 들은 횟수를 **Update**를 통해 증가시킨다. 이 음원이 하루동안 들은 횟수를 증가시켜 나중에 **실시간 차트**를 불러오는데 영향을 끼치는 쿼리다.

-> update 사용

get_ranking_top10은 **플레이리스트에서 어떤 노래를 많이 듣는지**를 불러오는 쿼리다. 해당 유저의 플레이리스트에서만 **스트리밍된 횟수**를 고려하여 **ORDER BY**문을 사용하여 상위 10개의 음원 정보를 가져오는 쿼리다. 그리고 **desc Limit**을 통해 내림차순으로 갯수 제한을 두었다. 그리고 playlist와 soundtrack table(음원)을 **join**하여 그 테이블에서의 튜플을 가져왔다.

-> select, join, order by desc limit 사용

plus_play_count_0~50은 유저의 나이를 고려하여 음원이 10대 이하,10대,20대,30대,40대,50대 이상에게서 스트리밍되는지 파악 후, **나이대**에 맞는 **스트리밍 횟수**를 **Update**를 통해 증가시킨다. 나이대별로 정리하여 이후에 나이대 별로 가장 선호되는 음원의 순위를 가져오는데 영향을 끼치는 쿼리다.

-> update 사용

insert_playlist_name은 한 유저의 플레이리스트를 **Insert**를 통해 추가하는 쿼리다.

-> Insert 사용

search_music_by_name은 **음원의 제목**을 통해 음원 제목, 장르, 가수 등의 정보를 불러오는 쿼리다.

-> select 사용

search_music_by_composer_name은 **작곡가/작사가**의 이름을 통해 음원의 정보, 가수 정보를 가져오는 쿼리다. 작곡가와 음원의 정보를 Join한 View인 **soundtrack_artist_compose**와 sing table, artist table를 join하고 음원의 제목으로 **group by**를 한다. 그러면 듀엣 같은 경우 음원 별로 가수들이 **group by**된다. 이를

group_concat을 통해 여러 가수가 같은 음원을 부른 경우 하나의 튜플로 압축시킨다.

-> select, join, group_concat(distinct), group by 사용

search_music_by_artist_name은 **가수(아티스트)**의 이름을 통해 음원, 제목, 가수 등의 정보를 불러오는 쿼리다. 여기에서도 **soundtrack_artist View**를 사용하지 않고 직접 soundtrack, sing, artist를 **join**하여 음원과 아티스트의 정보를 묶었다.

-> select, join, group by 사용

insert_playlist_music은 한 유저의 플레이리스트에 **음원을 추가**하기 위한 쿼리다.

-> Insert into 사용

get_user_paymentday는 한 유저의 **결제 날짜**를 가져오기 위한 쿼리다. 이 쿼리를 통해 음악을 1분 미리듣기 하는지, 전체 듣기가 가능한지 **Select**를 통해 파악이 가능하다.

-> select 사용

search_n_ranking_0~50은 0~50대에게 어떤 음원이 인기가 많은지 보여질 때 사용되는 쿼리다. **ORDER BY**를 통해 어떤 노래가 많이 스트리밍됐는지 나이대별로 알 수 있게 하는 쿼리다. **Desc limit**을 통해 상위 n개의 tuple를 가져온다. 그리고 나이대별로 스트리밍 횟수를 모아놓은 playcount table과 soundtrack table를 **join**하여 음원의 정보를 가져온다.

-> select, join, order by desc limit 사용

show_statistics_ratio는 **한 음원**의 10대이하~50대이상 나이대의 모든 스트리밍된 횟수를 가져와 퍼센트를 구하기 위한 쿼리다.

-> select 사용

update_user_subscriber~name은 유저의 개인정보를 바꾸기 위한 쿼리다. **UPDATE**를 통해 개인 정보를 변경할 수 있다.

-> update 사용

delete_id는 유저 **탈퇴** 기능 구현을 위한 쿼리다.

-> delete 사용

get_artist_index는 가수의 이름을 가지고 **전체 정보**를 가져오기 위한 쿼리다. 전체 정보를 가져오면 **Primary Key**도 같이 가져오기 때문이다.

-> select 사용

delete_artist는 관리인이 아티스트의 정보를 삭제하기 위한 쿼리다.

-> delete 사용

insert_artist는 관리인이 아티스트의 정보를 추가하기 위한 쿼리다.

-> insert 사용

delete_soundtrack은 관리인이 음원을 제거하기 위한 쿼리다.

-> delete 사용

insert_soundtrack은 관리인이 음원을 추가하기 위한 쿼리다.

-> insert 사용

delete_composer는 관리인이 작곡/작사가의 정보를 제거하기 위한 쿼리다.

-> delete 사용

insert_composer은 관리인이 작곡/작사가의 정보를 추가하기 위한 쿼리다.

-> insert 사용

insert_into_sing은 관리인이 음원과 아티스트의 M:N관계를 위해 형성된 **sing** 테이블에 음원과 아티스트의 정보를 넣어 음원과 아티스트의 **관계**를 연결시켜주는 쿼리다.

-> insert 사용

insert_compose_write은 관리인이 음원과 작곡/작사가의 M:N관계를 위해 형성된 **compose_write**테이블에 음원과 작곡/작사가의 정보를 넣어 음원과 작사/작곡가의 **관계**를 연결시켜주는 쿼리다.

-> insert 사용

show_all_streaming은 관리인이 스트리밍 유저들의 모든 정보를 가져오기 위한 쿼리다. **Select**를 통해

streaming_subscriber table의 정보를 가져온다.

-> select 사용

delete_user는 관리인이 유저를 삭제시키기 위한 쿼리다.

-> delete 사용

extend_payment는 관리인의 권한으로 한 유저의 구독기간을 늘려주는 쿼리다.

-> Update 사용

insert_play_count_table은 한 음원이 생성될 때 나이대별로 스트리밍횟수를 관리하기 위한 table(**play_count**)에 해당 음원의 튜플을 생성하는 쿼리다.

-> insert 사용

show_all_artist는 관리인이 모든 가수(아티스트)의 정보를 불러오기 위한 쿼리다.

-> select 사용

show_all_soundtrack은 관리인이 모든 음원의 정보를 가져오기 위한 쿼리다.

-> select 사용

get_group_name은 가수 중에서 그룹멤버의 이름을 통해 모든 가수들의 정보를 불러오는 쿼리다.

-> select 사용

is_already_put_music은 스트리밍 구독자가 이미 플레이리스트에 동일한 음원을 넣었는지 확인하기 위한 쿼리다.

-> select, join 사용

delete_playlist는 유저가 플레이리스트를 삭제 할 때 플레이리스트에 있는 음원을 삭제하는 쿼리다.(playlist table에서 튜플 제거)

-> delete

delete_playlist_name유저가 플레이리스트를 삭제 할 때 유저가 갖고 있는 플레이리스트 목록에서 제거한다. 즉, **playlist_name** table에서 유저의 삭제하려는 플레이리스트 이름에 맞는 튜플을 삭제한다.

-> delete

Database_Main.py

이 파일은 처음 어플리케이션을 켜올 때 실행되는 파일이다. 시작시 **pymysql** 라이브러리를 통해 데이터베이스 채널을 연결한다.

```
connection=pms.connect(host="127.0.0.1",port=3306,user="root",password="martin!1107",db="soundtrack")
cursor=connection.cursor()
```

다음 두 줄을 통해 채널을 만들고 **커서**를 만든다.

처음 실행시에는 **While**문이 실행되며 **로그인** 절차를 따르게 된다. id와 password를 입력하면

show_login_window() 함수가 실행되어 데이터베이스에 id와 Password가 존재하는지 확인한다. 만약 커서를 통해 반환되는 결과가 없더라면

로그인이 되지 않게하고 계속해서 질문하게 한다. 만약 **N** 입력시 자동으로 회원가입창으로 넘어가지게 했으며, 프로그램을 종료하고 싶으면 **Q**입력을 통해 종료시킨다. 처음 실행시 아래와 같은 화면이 구현된다.

```
C:\Users\rlqja>python DataBase_Main.py
-----데이터베이스 음원 스트리밍에 오신 것을 환영합니다-----
로그인이 필요합니다.

아이디가 존재하나요?(Y/N)
"N" 입력시 회원가입란이 나옵니다.
종료하려면 Q를 입력해주세요
Input : _
```

show_login_window()

데이터베이스에 ID와 Password를 보내고 반환되는지 확인하는 함수다.

show_signup()

'N'입력시 **회원가입** 창으로 넘어가지며 이 함수를 호출한다. 스트리밍 구독자의 데이터베이스에서 Id가 **Primary Key**이므로 중복검사가 필수적이다. `is_id_exist` 쿼리를 통해 있는지 새로 생성하려는 ID가 DB에 있는지 확인하고 다음 질문으로 넘어가게 한다.

모든 정보를 입력한 후, `signup_query`를 통해 스트리밍 구독자의 데이터베이스에 넣는다.

만약 Id, Password가 DB에 존재한다면 Login이 되게 한다. 로그인 했을 때의 기능들은 **User_Main.py**를 통해 진행한다. 이 함수가 실행 될시 아래와 같은 화면이 나온다.

```
아이디가 존재하나요?(Y/N)
"N" 입력시 회원가입란이 나옵니다.
종료하실려면 Q를 입력해주세요
Input : n

회원가입란
ID : hanyang

종료하려면 Q를 입력해주세요
ID를 다시 입력해주세요 : database
ID 사용가능
Password : 1234
Phone : 01022222222
Address : Seoul
Social Security Number(앞자리) : 981101
Name : Hanyang
Email : hanyang.ac.kr

회원가입이 완료되었습니다.
```

User_Main.py

이 파일은 유저/관리자가 여러 기능들을 수행할 때 사용되는 파일이다. e.x)유저 메뉴 화면 출력, 관리인의 DB관리 등

show_user_main()

Login시 처음으로 실행되는 함수다. 이 때 매개변수로 `is_supervisor`가 넘어오게 된다. 이 변수는 로그인한 유저가 관리인인지 확인하는 변수다. `is_supervisor`가 None(=Null)이면 관리인을 의미한다. 관리인이 아닐시에는 데이터베이스에서 Super_ID가 'root'로 관리인을 가리키고 있기 때문에 **None**이 될 수 없다. 스트리밍 구독자의 DB(`streaming_subscriber`)에는 SSN이 들어가게 되는데 이 때, 주민등록번호 앞자리가 들어가게 된다. 그러면 나이를 추측할 수 있게 되는데, 나이로 변환하는 함수가 `age_convert()` 함수를 통해서 주민등록번호 앞자리로 인해 나이로 변경되어진다.

로그인이 되는 동시에 실시간 차트를 `show_realtime_chart()` 함수를 통해 보여주게 된다. 실시간 차트를 보여준 후, `show_menu()` 함수를 통해 유저가 사용할 수 있는 기능들을 나열하게 된다. 아래 사진은 이 함수가 실행되어 유저화면이 켜졌을 때의 사진이다. 관리인으로 로그인했을 때는 맨 아래 **6.관리인모드**가 생겨난다. 관리인은 Id가 'root'이고 비밀번호는 1234로 관리할 수 있게 했다. 관리인도 노래를 들을 수 있고, 유저의 기능들도 **같이** 이용할 수 있게 했다. 관리인도 유저와 똑같이 플레이리스트를 만들 수 있고, 음원 검색 기능을 이용할 수

있다.

```

환영합니다! root님
-----실시간 차트 순위-----
1. 주문(19) - 동방신기
2. 몸매(19) - 박재범
3. 사계 - 태연
4. I Believe - 신승훈
5. 내 사람 - SG워너비
6. 하루 끝 - 아이유
7. 보여줄게 - 에일리
8. All For You - 서인국,정은지
9. Bad Boy - 레드벨벳

1. 플레이리스트를 보기
2. 음원 검색
3. 각 나이별 노래 순위
4. 개인정보 변경
5. 뒤로가기
6. 관리자 모드
Input :

```

show_menu()

is_supervisor변수를 통해 관리인인지 확인을 통해 기본 메뉴 중에 **관리인 모드**를 추가하여 다른 기능들을 수행할 수 있게 한다. 음원 추가,삭제 스트리밍 구독자 관리할 수 있는 메뉴를 하나 더 만드는 함수다.

switch_user 클래스에서 통해 메뉴 선택을 할 수 있게 된다. 'switch-user'의 자세한 내용은 아래와 같다.

switch_user()

이 함수에서는 Java의 **switch-case**기능을 구현시켰다. **Class**에 여러 함수를 구현하고 사용자가 선택시 알맞는 함수가 구현되는 방식이다. 이는 파이썬의 **getattr()** 기본 함수를 통해 구현했다. 번호마다의 기능은 위 사진을 참고하면 된다. 1번을 눌렀을 때는 **Playlist.py**파일이 실행되며 플레이리스트 관련 기능을 수행한다. 2번 눌렀을 때는 **Search_Main.py**파일을 통해 **음원 검색**기능을 이용할 수 있다. 이 때, 작곡/작사가, 가수를 통해서도 음원 검색하여 플레이리스트에 추가가 가능하게 했다. 3번을 눌렀을 때는 각 **나이대** 별로 선호하는 노래를 스트리밍 횟수를 통해 알 수 있는데, 이는 **Statistics_Show.py**파일에서 구현했다. 4번을 누르면 개인 정보를 변경할 수 있게 하는 **Change_Information.py** 파일이 실행된다. 6번을 눌렀을 때의 관리모드는 **User_Main.py**에서 구현했다.

supervise_mode()

관리인이 6번을 누르면 이 함수가 먼저 실행된다. 아래와 사진 같은 화면이 나온다. 1번을 누르면 음원,아티스트,작사/작곡가를 추가하거나 삭제할 수 있는 화면이 나온다. 이는 **manage_overall()** 함수를 호출하여 기능을 수행한다. 2번을 누르면 스트리밍구독자를 계정 강제삭제시킬 수도 있고 결제날짜를 변경시켜 **전체 듣기 기능**을 부여할 수 있다. 이는 **manage_streaming()** 함수를 호출하여 기능을 수행한다. 이 함수는 관리인의 메뉴판 역할을 한다.

```

6. 관리자 모드
Input : 6
-----관리자 모드-----
1. 음원, 아티스트, 작사가 관리
2. 스트리밍 구독자 관리
3. 뒤로가기
Input :

```

manage_overall()

supervise_mode()에서 1번을 눌렀을 때 호출되는 함수다. 이 때, 어떤 것을 삭제 또는 추가시킬지 고르는 메뉴판 기능이다. 아티스트, 음원, 작사/작곡의 쿼리를 Directory구조를 이용하여 저장했고, 코드를 간결화시켰다.

뒤로가기를 제외한 1,2,3을 누르면 `insert_delete_update()` 함수가 호출된다.

```
-----관리자 모드-----
1. 음원, 아티스트, 작사가 관리
2. 스트리밍 구독자 관리
3. 뒤로가기
Input : 1

추가/삭제 가능
1. 아티스트 관리
2. 음원 관리
3. 작사가 관리
4. 뒤로가기
Input :
```

`insert_delete_overall()`

작사,음원, 아티스트에서 뒤로가기를 제외한 숫자를 누르면 이 함수가 호출되어 아래와 같은 화면이 나온다. 삭제할지, 추가할지 모두 볼지 고르게 된다. 2개의 함수(`delete_database()`, `add_database()`)를 Directory를 이용하여 저장시켰고 코드를 간결화시켰다. 만약 1번을 누르게 되면 `delete_database()` 함수가 호출된다. 2번(추가)을 누르면 `add_database()`가 호출되고 3번(모두 보기)를 누르면 `show_all_inform()`을 통해 모두 볼 수 있는 기능을 수행한다. 이 함수도 역시 메뉴판과 같은 역할을 수행한다.

```
추가/삭제 가능
1. 아티스트 관리
2. 음원 관리
3. 작사가 관리
4. 뒤로가기
Input : 1
1. 삭제
2. 추가
3. 모두 보기
4. 뒤로가기
Input :
```

`delete_database()`

이 함수에서는 위에서 고른 아티스트, 음원, 작곡/작사가의 이름을 입력 후 **제거**하는 기능을 수행한다. 이 함수가 실행되어 모든 기능을 수행하면 아래와 같은 사진이 나온다. 위의 `insert_delete_overall()`에서 1번(**아티스트**)을 눌렀을 때, 2번(**음원**)을 눌렀을 때, 3번(**작곡/작사가**)를 눌렀을 때 if문을 통해 구분하게 했다. 아래 사진과 같이 1번(삭제)을 누르면 이 함수가 호출된다. `insert_delete_overall()`에서 **1.아티스트 관리**를 눌렀다고 하자. 그리고 삭제할 이름을 검색했을 때 `request_inform()` 함수가 호출되어 DB에 삭제할 이름의 **tuple**이 존재하는지 확인한다. 만약 없을 경우에는 재귀함수 형태로 다시 삭제할 이름을 입력받게 한다.

위의 `supervise_mode()` 함수를 보면 쿼리를 directory구조로 만들었다고 했다. 그 Directory구조를 활용하여 `request_inform()`에서 알맞는 쿼리를 이용하여 DB에 튜플이 존재하는지 확인한다.

만약 DB에 존재한다면 목록을 보여준다. **for**문을 통해 중복되는 이름의 아티스트가 존재할 수도 있기 때문에 검색한 이름의 모든 튜플을 보여준다. 그리고 원하는 정보를 삭제하게끔 번호를 입력받도록 한다. 이 때 가져온 튜플을 index에 맞게 `index_list`변수의 List에 저장하여 번호를 눌렀을 때 그 정보(**Primary Key**)를 활용하여 삭제하도록 한다. 삭제 할 때는 `SQL_Query.py`에서의 `delete_artist`쿼리문을 활용하여 삭제했다.

음원은 삭제시에 M:N 관계 형성을 위해 생성된 `compose_write`와 `sing table`에서도 관계된 튜플이 삭제된다. 이는 **on delete cascade**를 통해 구현했다.

여기서 주의점은 음원이 삭제되기 전에는 아티스트와 작곡/작사가를 삭제 못하게 했다. 음원이 존재 할 때는 그 작곡/작사가와 가수가 있어야 음원이 발매되기 때문이다. 따라서 가수와 작곡/작사가를 제거해야 될 때는 관련된 음원을 제거해야 한다. 따라서 **on delete cascade**로 설정하지 않고 **restrict**로 설정했다.


```

Input : 1
1. 삭제
2. 추가
3. 모두 보기
4. 뒤로가기
Input : 1

삭제 이름(명)
뒤로가기 : Q
Input : 조이
1. 이름 : 조이   성별 : 여자   데뷔 날짜 : 2014-08-01

번호 입력 : 1

성공

```

나머지 음원과 작사/작곡가 삭제시에도 동일한 과정을 거친다. `request_inform()`을 통해 DB에 검색한 튜플이 존재하는지 확인 한 후, 검색한 이름의 모든 튜플을 보여주고 선택하여 삭제하도록 한다. 번호에 맞게 목록을 보여주는 화면은 아래의 사진과 같다. 위의 아티스트와 다른 점은 음원의 경우에는 `delete_soundtrack`의 쿼리를 사용했고, 작사가의 경우에는 `delete_composer`의 쿼리를 이용했다는 점이다. 지금부터 언급하는 쿼리문은 모두 `SQL_Query`에서의 변수명이다.

```

삭제 이름(명)
뒤로가기 : Q
Input : 웬디
1. 이름 : 웬디   성별 : 여자   데뷔 날짜 : 2014-08-01

번호 입력 : 1

성공

```

add_database()

이 함수는 `insert_delete_overall()`에서 2번(추가)를 눌렀을 때 호출되는 함수다. 이 함수에서도 if문을 통해 아티스트,음원,작곡/작사가를 구분하여 DB에 추가한다. **choose**변수가 1이라는 것은 위에서 아티스트를 선택했다는 의미다. 아티스트 선택시 정보들을 `input()`을 통해 입력받고 `insert_soundtrack`의 쿼리문을 통해 DB에 **Insert**한다. 여기서 가수는 그룹이 있을 수도 있고 없을 수도 있는데 그룹이 없을 때는 아티스트 table의 그룹명 **Attribute**에 'solo_'를 넣는다. 'solo_'를 통해 가수가 그룹이 있는지 없는지 확인한다.

음원의 경우에는 살짝 복잡하다. 음원을 내는 가수가 그룹인지 솔로인지 확인이 필요하다. 이 때 그룹이 존재한다면 그룹명 **Attribute**에 그룹명을 넣고 **group_check**변수의 flag를 통해 그룹인지 아닌지 판단한다. 그룹일 경우에는 1로 셋팅되고 솔로일 경우에는 0인 값을 갖는다. 이 값은 음원 DB에 들어가서 아티스트가 솔로로 발매한 것인지, 그룹으로 발매한 것인지 확인할 수 있게 한다.

또한, 그룹이 아니더라도 듀엣으로 발매하는 경우가 존재한다. 예를 들어, 'All For You'는 서인국과 정은지가 발매한 아티스트가 2명이상 존재할 수 있다. 따라서, 음원을 발매한 아티스트를 모두 입력받게 한다. 이때도, 같은 이름의 아티스트가 존재할 수 있어, 확인이 필요하다. 확인하는 절차는 아래의 사진과 같다. 아래의 사진과 같이 DB에 존재하는 이름의 아티스트를 모두 불러들여 관리인이 선택할 수 있게 한다.

이는 작곡가/작사가도 마찬가지다. 작곡/작사가의 확인 과정은 `search_compose()`를 통해 같은 이름의 작곡가가 있으면 선택하여 지정할 수 있게 한다. 아래의 사진에서 "윤일상" 선택하는 과정이 이 함수에 해당한다. 선택한 후, 바로 `search_compose()`에서 `insert_compose_write` 쿼리를 통해 table에 넣는다. 이 과정을 통해 음원에 넣기 때문에 작곡/작사가와 가수의 데이터가 DB에 저장되어있어야 한다.

가수(아티스트)와 음원, 그리고 음원과 작곡/작사가 그리고 아티스트의 관계는 각각 M:N으로 관계 유지를 위해 별도의 Table인 **compose_write,sing**이 존재한다. 관계 유지를 위해 두 테이블에도 넣어주어야 한다. 아티스트가 그룹인 경우에는 그룹명에 해당하는 아티스트 한 tuple을 DB에서 가져온다. 그리고 그 가수의 Primary Key와 음원의 PK를 관계 유지를 위한 **sing** table에 넣는다. 이 때, 음원 table에 아티스트가 그룹인지를 check하는 Attribute가 존재하기 때문에, 나중에 음원을 발매한 아티스트를 불러오더라도 한 가수의 이름을 가져오는 것이 아니라 가수가 속한 그룹명을 가져오게 된다.

마지막으로는 한 음원의 나이대별로 통계치 관리를 위한 **play_count** table(1:1)에도 음원의 PK와 같이 `insert_play_count_table` 쿼리를 통해 튜플을 넣어준다.

3번(작곡/작사가)을 눌렀을 때는 `insert_compose()` 함수가 호출되어 작곡/작사가의 정보를 넣게 한다. 이 때, `get_composer_index_2` 쿼리를 통해 동일한 작곡/작사가의 정보를 넣으려는지 확인 한 후, Insert한다. 작곡/작사가의 정보를 넣을 때는 `insert_composer` 쿼리를 통해 넣는다.

아티스트와 음원에 데이터를 넣으려고 할 때, 이미 존재할 수 있다. 따라서 `try-except`를 통해서 판별한다. `except`로 실행되는 경우에는 이미 Table에 존재하기 때문에 더 이상 실행시키지 않는다.

```

1. 삭제
2. 추가
3. 모두 보기
4. 뒤로가기
Input : 2
1. 음원명 : 끝사랑
2. 19세이상(Y/N) : n
3. 노래시간(MM:SS) : 03:35
4. 장르 : 발론
5-1. 그룹으로 불렀는가?(Y/N) : n
5-2. 아티스트 전체 인원(숫자만) : 1
5-3. 아티스트명 : 김범수
1. 김범수 데뷔날짜 : 1999-10-05
해당 아티스트 번호 입력 : 1
6. 작곡(생략가능) : 윤일상
7. 작사(생략가능) :
저장하시겠습니까?(Y/N)
Input : y

작곡 : 윤일상 선택
1. 이름 : 윤일상 Type : 작곡 Birth : 740221

번호 입력 : 1
DB 저장중 ....

```

show_all_inform()

이 함수는 `insert_delete_overall()`에서 3번(모두보기)를 눌렀을 때 호출되는 함수다. 이 함수는 DB에 저장되어 있는 음원, 가수, 또는 작곡/작사가의 정보를 모두 불러올 때 사용한다. 아티스트의 경우에는 DB에서 'Solo_Group' Attribute에 솔로인지 그룹인지 확인하는 Attribute가 존재한다. 'solo_'인 경우에는 솔로를 의미하고 나머지는 그룹에 속한다고 판단한다. 따라서 정보를 불러와서 출력시킬 때는 이 값을 확인 한 후, 그룹명도 같이 출력하게 한다. 이 때는 `show_all_artist` 쿼리를 이용하여 모든 정보를 불러온다.

음원의 경우에는 노래가 19세인지 아닌지 확인하는 'Over_19'의 Attribute가 존재한다. 'Over_19' Attribute가 1인 경우에는 19세 이상을 의미하므로 19세를 노래명에 표시한다. 그리고 'is_Group' Attribute를 통해 이 음원이 그룹으로 발매된 것인지 확인한다. 값이 1이면 아티스트의 이름이 아닌 그룹명을 가져오도록 한다. 만약 값이 0이면 솔로 또는 다수의 가수(듀엣 등)가 모여 발매한 것이다. 이 때는 이 음원과 관계되어 있는 모든 아티스트를 불러오고 `GROUP_CONCAT(아티스트 이름 Attribute)`으로 하나의 튜플로 아티스트의 이름을 압축한다. 아래의 사진에서 All For You 음원의 경우 2개의 아티스트 이름 Attribute를 통합시킨 것이다.

```

1. 삭제
2. 추가
3. 모두 보기
4. 뒤로가기
Input : 3
-----전체 음원 목록-----
1. All For You - (세인국,정은지) Time : 03:50
2. Bad Boy - (레드벨벳) Time : 03:13
3. I Believe - (신승훈) Time : 03:30
4. 끝사랑 - (김범수) Time : 03:35
5. 내 사람 - (SG워너비) Time : 04:13
6. 몸매(19) - (박재범) Time : 03:15
7. 보여줄게 - (에일리) Time : 04:13
8. 사계 - (태연) Time : 03:13
9. 주문(19) - (동방신기) Time : 03:35
10. 하루 끝 - (아이유) Time : 03:55

```

manage_streaming()

이 함수는 `manage_overall()`에서 2번(스트리밍 구독자 관리)를 눌렀을 때 실행되는 함수다. 처음에 `show_all_streaming()`을 통해 모든 스트리밍 구독자의 정보를 화면에 출력한다. 이 때는 `show_all_streaming`의 쿼리를 통해 모든 튜플을 가져온다. 그리고 아래의 화면과 같이 3개의 선택사항이 있다. 만약 1(삭제)를 입력하면 Id를 입력받은 후, `delete_user` 쿼리를 통해 삭제시킨다. 2(결제기간 연장)을 누르면 원하는 유저의 결제기간을 현재로 설정하여 전체듣기기능을 이용할 수 있다. 결제기간 연장을 하면 아래의 사진과 같이 구독기간도 같이 늘어난다. 구독기간을 통해 어떤 유저가 단골인지 확인할 수 있다. 구독기간을 늘릴 때는 `get_payment_day` 쿼리를 통해 유저의 결제 날짜를 가져 온 후, 연장 개월수 *30일만큼 결제날짜를 연장시킨다. 이 때, `extend_payment` 쿼리를 통해 구독 전체 기간도 늘어나게 한다.

```
-----전체 스트리밍 Id-----
1. Id: database Name: Hanyang Payment_day : 2015-10-10구독 전체기간 : 0
2. Id: Hanyang Name: 한양 Payment_day : 2015-10-10구독 전체기간 : 0
3. Id: riqja Name: 김기범 Payment_day : 2016-02-07구독 전체기간 : 2
4. Id: riqja1107 Name: 홍길동 Payment_day : 2020-12-01구독 전체기간 : 1

1. 삭제
2. 결제기간 연장
3. 뒤로가기
Input : 2
뒤로가기 : Q
스트리밍 ID 입력 : database
연장 개월 수 : 3
연장 완료
1. 삭제
2. 결제기간 연장
3. 뒤로가기
Input : 3

-----관리자 모드-----
1. 음원, 아티스트, 작사가 관리
2. 스트리밍 구독자 관리
3. 뒤로가기
Input :2

-----전체 스트리밍 Id-----
1. Id: database Name: Hanyang Payment_day : 2016-01-08구독 전체기간 : 3
2. Id: Hanyang Name: 한양 Payment_day : 2015-10-10구독 전체기간 : 0
3. Id: riqja Name: 김기범 Payment_day : 2016-02-07구독 전체기간 : 2
4. Id: riqja1107 Name: 홍길동 Payment_day : 2020-12-01구독 전체기간 : 1
```

show_realtime_chart()

이 함수는 처음 실행시 실시간 차트를 보여주기 위해 실행되는 함수다. `realtime_get_query`를 통해 음원 table의 `Today_streaming` Attribute를 Order by하여 상위 10개의 음원을 가져오는 함수다. 화면에 보여질 때는 table에 저장된 `Over_19`(19세 이상의 음원 판단), `is_Group`(그룹으로 발매했는지 판단)을 통해 아티스트명으로 그룹명을 보여줄 것인지, 19세 이상의 음악인지 보여주게 된다.

Search_Main.py

이 파일은 음원 검색 기능이 구현되어 있는 파일이다.

search_main()

`User_Main.py`에서 2번(음원검색)을 눌렀을 때 처음으로 실행되는 함수다. 이 함수가 실행되면 처음에 아래와 같은 화면이 구현된다. 원하는 번호를 입력하게 되면 `switch_search_music` class의 자바의 `switch-case` 기능과 유사한 기능이 실행하게 된다. 4번(뒤로가기)을 누르기 전에는 계속 음원검색창을 보여주기 위해 재귀함수를 이용했다.

```
1. 플레이리스트를 보기
2. 음원 검색
3. 각 나이별 노래 순위
4. 개인정보 변경
5. 뒤로가기
6. 관리자 모드
Input : 2

-----음원 검색-----
1. 음원명으로 검색
2. 작곡가, 편곡가, 작사가로 검색
3. 아티스트로 검색
4. 뒤로가기 :
Input:
```

switch_search_music()

이 함수는 java에서의 **switch-case**문을 class를 통해 구현했다. 1번을 누르면 `search_by_music_name()`, 2번 누르면 `search_by_composer()`, 3번을 누르면 `search_by_artist()`를 실행시켰다. 생성자에서 `getattr()`을 통해 함수 이름을 가지고 객체로 변형하여 실행시키게 했다. 다른 번호를 1~6번 이외의 숫자를 입력하게 되면 `case_6()`이 호출되어 `is_continue=True`로 while문을 계속 실행하게 했다.

search_by_music_name()

이 함수는 1번(음원명으로 검색)을 눌렀을 때 실행되는 함수다. 찾을 음원명을 **input**으로 받은 후, `search_music_by_name` 쿼리를 통해 음원 table에 튜플이 존재하는지 확인한다. 없을 경우 재귀함수로 다시 음원명을 입력받도록 한다.

만약 음원명을 검색하면 `convert_to_music_title()` 함수가 호출되어 찾는 이름의 음원을 검색하여 보여주도록 한다. 검색이 다 끝나면 재귀함수를 통해 다시 음원명을 입력받도록 한다.

```
1. 음원명으로 검색
2. 작곡가, 편곡가, 작사가로 검색
3. 아티스트로 검색
4. 뒤로가기 :
Input: 1

음원명을 검색해주세요
뒤로가기 : Q
Input : 
```

convert_to_music_title()

다른 함수에서 DB에서 불러온 튜플들을 화면에 음원명과 가수명을 출력하는 함수다. `search_music_by_name` 쿼리에서 검색 결과가 존재하면 정보들을 아래와 같은 화면이 출력된다. 이 때도 역시, 위에서 설명한 것과 같이 'is_Group', 'Over_19' Attribute를 이용한다.

이 함수가 종료되면 `user_listen_music()` 함수를 통해 어떤 음원을 선택할지 고르게 된다.

```
음원명을 검색해주세요
뒤로가기 : Q
Input : 내 사람

1. 내 사람 - (SG워너비) - Time : 04:13

음원 듣기(번호 선택)
뒤로가기 : Q
Input : 
```

user_listen_music()

이 함수는 어떤 음원을 들을 것인지 Input을 받고 처리하는 함수다. 'Q'를 입력하여 뒤로가기를 할 것인지, 범위 안의 숫자를 입력했는지 처리하게 된다. 만약 숫자 범위의 음원을 Input으로 입력하게 되면 `show_music_detail()`이 호출되어 음원과 관련된 자세한 정보들을 화면에 출력하게 된다.

show_music_detail()

이 함수는 음원의 PK인 **음원 제목**과 **노래시간**을 매개변수로 받아 자세한 정보들을 출력하는 함수다. 음원의 정보 뿐만 아니라, 어떤 가수가 불렀는지, 작곡가는 누구인지의 정보를 출력하게 된다. 작곡/작사가가 없을 때는

DB에 'X'가 저장되게 된다. 따라서 작곡/작사가의 이름이 'X'로 되어있으면 '정보없음'을 출력한다. 이 함수가 호출될 때는 실제로 음원이 스트리밍된다고 가정했다. **show_music_detail** 쿼리를 호출하여 음원, 아티스트 정보를 가져온 후, **show_music_compose** 쿼리를 통해 음원의 정보와 관련된 작곡/작사가의 정보를 가져와 출력한다. 정보를 출력하기 전에 **compare_day()** 함수가 호출되어 사용자가 **결제** 날짜를 보고 **전체** 듣기가 가능한지 판단한다. **compare_day()**에서 결제했던 날짜로부터 30일 이내면 전체듣기가 가능하고 아니면 1분 미리듣기만 가능하게 했다. 날짜 비교는 **datetime** 라이브러리를 활용했다.

음원의 자세한 정보를 출력한 후, 이 음원의 통계치를 **show_statistics_ratio()**를 통해 보여준다. 이 함수에서는 음원 table과 1:1 연결되어 있는 **play_count** table로부터 나이대별 streaming 횟수를 가져온 후, 퍼센트로 통계치를 보여준다. 아래 사진의 마지막 line과 같이 어느 나이대가 많이 듣고 있는지 보여준다.

이 함수가 호출되면 음원이 스트리밍된다고 가정했다. 따라서, 음원과 관련된 스트리밍 횟수를 증가시켜야 하는데 이는 **increase_streaming()**을 통해 구현했다. **increase_streaming()**에서는 User의 나이를 이용하여 어느 나이대인지 판단한 후, **plus_play_count_0~50** 쿼리를 이용하여 나이대에 맞는 횟수를 증가시킨다. 뿐만 아니라, 음원 자체의 오늘 스트리밍된 횟수를 **plus_today_soundtrack** 쿼리를 통해 증가시켜준다.

마지막에는 이 음악을 플레이리스트에 추가할 것인지 확인한다. playlist에 넣는다고 'Y'를 출력하면

put_playlist_or_not() 함수 호출을 통해 playlist에 넣는 작업을 수행한다. 'N'을 입력하게 되면 이 함수를 종료하게 한다.

```

음원 듣기(번호 선택)
뒤로가기 : Q
Input : 1

>>>노래 전체 듣기가능<<<<

음원명 : 내 사람
아티스트 : SG워너비(그룹)
노래 시간 : 04:13
장르 : 발라드
No type : 정보 없음
10대 이하: 26.9% 10대: 9.6% 20대: 30.9% 30대: 0.3% 40대: 12.0% 50대 이상: 20.2
플레이리스트에 넣으시겠습니까?(Y/N)
Input :

10대 이하: 26.9% 10대: 9.6% 20대: 30.9% 30대: 0.3%
플레이리스트에 넣으시겠습니까?(Y/N)
Input : y
-----Playlist 목록-----
1. DATABASE
2. HERO
3. root

어느 Playlist에 넣으시겠습니까?(위의 번호 입력)
뒤로가기 : -1
Input : 1

DATABASE 에 담았습니다.

음원명을 검색해 주세요
뒤로가기 : Q
Input :
  
```

put_playlist_or_music()

이 함수에서는 플레이리스트가 존재한다면 목록을 보여주고 선택하게 하고, 존재하지 않으면 플레이리스트를 만들도록 유도한다. **show_playlist_name** 쿼리를 통해 유저의 **playlist**가 있는지 확인한 후, 플레이리스트 이름을 가져온다. 없을 경우에는 **Playlist.py**의 **make_playlist()** 함수를 통해 플레이리스트를 만들도록 하고 재귀 함수를 통해 플레이리스트에 넣을 것인지 확인한다.

존재한다면 플레이리스트 목록을 for문을 통해 보여주고, 번호로 선택받게 한다. 플레이리스트에 넣으려고 할 때 먼저 플레이리스트에 해당 노래가 담겨있는지 확인해야 한다. 확인하는 쿼리가 **is_already_put_music** 쿼리를 통해 진행된다. 존재한다면 함수를 종료시키고, 없다면 **insert_playlist_music** 쿼리를 통해 플레이리스트에 넣는다. 플레이리스트에 넣었을 때 위 사진의 2번째 사진과 같이 나오게 된다.

search_by_artist()

`search_main()`에서 3번(아티스트명으로 찾기)를 눌렀을 때 실행되는 함수다. 2가지 케이스가 존재한다. **그룹명**으로 검색할 것인지, **가수명**으로 검색할 것인지다. 그룹명으로 검색시, `search_music_by_artist_group` 쿼리를 통해 그룹명에 해당하는 노래를 가져온다. 존재하지 않을 경우 재귀함수를 통해 다시 input을 받도록 한다. 만약 존재한다면 for문으로 정보를 출력한 후, `user_listen_music()`을 통해 들을 것인지 확인한다.

`user_listen_music()`의 설명은 위에서 설명했다.

가수명으로 검색시에는 `search_music_by_artist_name` 쿼리를 이용하여 가수명에 해당하는 음원을 가져온다. 이 때는 맨 처음의 View에서 설명한 `soundtrack_artist_2`에 쿼리를 던져서 가져오게 된다. 이 때도 존재할 경우, `user_listen_music()`을 통해 스트리밍 할 것인지 물어본다.

아래 사진은 음원을 `user_listen_music()`에 있는 함수 `show_music_detail()`을 통해 스트리밍 된후, 플레이리스트에 넣을 것인지 확인하는 화면이다.

```
Input : y
그룹명 : 레드벨벳
1. Bad Boy - 레드벨벳 Time : 03:13
음원 듣기(번호 선택)
뒤로가기 : Q
Input : 1
>>>>노래 전체 듣기가능<<<<
음원명 : Bad Boy
아티스트 : 레드벨벳(그룹)
노래 시간 : 03:13
장르 : 가요
No type : 정보 없음
10대 이하: 12.5% 10대: 20.1% 20대: 25.1% 30대: 20.1% 40대: 10.0% 50대 이상: 12.2%
플레이리스트에 넣으시겠습니까?(Y/N)
```

search_by_composer()

이 함수는 `search_main()`에서 2번(작곡/작사가로 검색)을 입력했을 때 호출되는 함수다. 이 함수는 간단하게 `search_music_by_composer_name` 쿼리를 통해 음원, 아티스트, 작곡/작사의 정보를 가져온다. 이 쿼리는 View에서 설명한 `soundtrack_compose_write`와 `artist(가수)` table를 join하여 정보를 가져온다. 가져온 결과가 있더라면 `convert_to_music_title()`를 통해 자세한 음원,가수 등의 정보들을 출력한다.

`convert_to_music_title()`의 자세한 설명은 위에서 설명했다. 자세한 정보를 출력하여 원하는 음원을 고르게 한다. 아래와 사진같이 고르면 스트리밍된다고 가정한다.

```
Input: 2
작곡가 또는 작사가의 이름을 입력해주세요.
뒤로가기 : Q
Input : 윤일상
1. Bad Boy - (레드벨벳) - Time : 03:13
2. 끝사랑 - (김범수) - Time : 03:35
3. 주문(19) - (동방신기) - Time : 03:35
음원 듣기(번호 선택)
뒤로가기 : Q
Input : 2
>>>>노래 전체 듣기가능<<<<
음원명 : 끝사랑
아티스트 : 김범수
노래 시간 : 03:35
장르 : 발라드
작곡 : 김도훈
통계치를 내는데 현재 정보가 부족합니다.
```

Playlist.py

이 파일은 플레이리스트 관리에 관한 파일이다.

show_playlist()

Database_Main에서 1번(플레이리스트 보기)를 입력하면 가장 먼저 실행되어 유저의 플레이리스트를 보여주는 함수다. 이 함수가 실행되면 아래와 같은 사진이 출력된다. 플레이리스트를 가져오는 데는 **show_playlist_name** 쿼리를 통해 정보를 가져온다. 번호를 입력하면 **switch_playlist()**에서 맞는 함수가 호출된다. **is_next** 변수를 통해 뒤로가기를 통해 **is_next** 변수가 3 또는 2이외의 값으로 변경되면 함수를 종료한다.

```

-----플레이리스트 목록-----
1. DATABASE
2. HERO
3. root
0. 플레이리스트 추가
Q. 뒤로가기
해당 번호, 플레이리스트 목록 보기
Input : 1

```

switch_playlist(class)

이 class는 java에서 switch-case문과 동일한 기능을 한다. 만약 0번을 눌러 플레이리스트를 더 만들고자 한다면 **make_playlist()** 함수가 호출되어 기능을 수행한다. 플레이리스트 목록이 존재할 경우 해당 번호를 누르면 **show_detail()** 함수가 호출되어 음원 목록을 보여준다. 이 함수는 **Search_Music.py**에서의 **convert_to_music_title()**과 동일한 기능을 수행한다. 뒤로가기를 제외한 나머지 기능을 수행하면 **is_next** 변수 설정을 통해 종료시키지 않고 뒤로가기를 입력했을 때만 **is_next** 변수 값을 변경시켜 playlist 기능을 종료시킨다.

show_detail()

Search_Music.py에서의 **convert_to_music_title()**과의 차이는 여기서서는 한 노래만을 자세히 보여주는 것이 아니라 해당 플레이리스트에 있는 노래를 보여주는 것이다. 이는 **show_playlist_music** 쿼리를 통해 정보를 가져오며 노래가 없을 경우에는 이 함수를 종료시킨다. 여기서도 **Search_Music.py**와 마찬가지로 음원 스트리밍 여부를 물어본다.

추가적인 기능으로 플레이리스트에서 가장 많이 듣는 노래 10개를 가져온다. 이는 **show_ranking_top10()** 함수를 통해 수행된다. 노래 번호에 맞는 숫자를 입력하면 **show_music_detail_()**이 수행되는데 이는

Search_Music.py에서 **show_music_detail()**과 동일한 기능을 수행한다. 노래 횟수 증가와 자세한 노래 정보 출력, 통계치 출력을 수행한다.

```

-----DATABASE의 플레이리스트 음원 목록-----
1. All For You - 서인국, 정은지
2. 내 사람 - SG워너비(그룹)
3. 주문(19) - 동방신기(그룹)

음원 선택(번호)
뒤로가기 : Q
가장 많이 듣는 10개 list 보기 : 0
Input : 0

```

show_ranking_top10()

이 함수는 Playlist중 가장 많이 듣는 10개 노래를 출력하는 함수다. **get_ranking_top10** 쿼리를 통해 Playlist table에서 **listen_count** 변수를 확인하여 상위 10개만을 가져오는 것이다. 가져온 후, 마찬가지로 스트리밍할지 여부를 while문을 통해 수행한다. 음원 번호에 맞는 숫자를 입력하면 **show_music_detail_()**이 수행된다. **show_music_detail_()**은 **Search_Music**에서 **show_music_detail()**와 같은 기능을 구현하므로 참고하면

된다.

```
음원 선택(번호)
뒤로가기 : Q
가장 많이 듣는 10개 list 보기 : 0
Input : 0

-----Playlist 중 가장 많이 듣는 Top 10-----
1. All For You - 서인국,정은지
2. 내 사람 - SG워너비(그룹)
3. 주문(19) - 동방신기(그룹)

음원 듣기(번호)
뒤로가기 :Q
Input : _
```

make_playlist()

이 함수는 사용자가 플레이리스트를 만들 때 호출하는 함수다. 즉, 위에서 0번(플레이리스트 추가)을 눌렀을 때 이 함수가 호출된다. 플레이리스트 이름을 입력 한 후, DB에 유저의 같은 이름 플레이리스트가 존재하는지 확인한다. 있다면 함수를 종료하고 없다면 **insert_playlist_name** 쿼리를 통해 DB에 넣는다.

```
0. 플레이리스트 추가
Q. 뒤로가기
해당 번호. 플레이리스트 목록 보기
Input : 0

Playlist 이름을 만들어주세요.
종료 : Q
Input : Good
성공적으로 GOOD Playlist를 생성했습니다.
-----플레이리스트 목록-----
1. DATABASE
2. GOOD
3. HERO
4. root

0. 플레이리스트 추가
Q. 뒤로가기
해당 번호. 플레이리스트 목록 보기
Input : 2
```

Statistics_Show.py

만약 **User_Main.py**의 메인 화면에서 3번(각 나이별 노래 순위)를 입력하면 수행되는 파일이다.

statistctics_main()

이 함수는 **User_Main.py**에서 이 파일로 함수를 수행하기 위해 넘어올 때 가장 먼저 수행되는 함수다. 먼저 어느 나이대의 선호하는 음원을 보고 싶은지 메뉴를 출력한다. 숫자를 입력하지 않으면 **재귀함수**를 통해 다시 Input하도록 한다. 만약 범위 안의 숫자를 입력하면 **each_age_show_statistics()**가 수행되어 원하는 나이대의 음원을 상위 n개 보여준다. **each_age_show_statistics()**에서는 **search_n_ranking_0~50** 쿼리를 **play_count** table에 보내서 원하는 나이대의 상위 n개 정보를 가져오도록 한다. **play_count** table에서 원하는 나이대의 스트리밍 횟수를 order by하여 상위 n개의 음원을 Join을 통해 가져온다. 가져오고 나서 **Search_Music.py**에서 **convert_to_music_title()**를 수행하여 음원 목록을 보여주고 playlist에 추가할 것인지, 스트리밍 할 것인지 기능을 수행한다. 아래 사진은 원하는 나이대의 번호를 입력하고 원하는 갯수의 음원 정보를 입력하면 나오는 결과창이다. 맨 아래와 같이 해당 음원을 스트리밍할 것인지를 **Search_Music.py**의

****convert_to_music_title()****를 통해 물어본다.

```

1. 플레이리스트를 보기
2. 음원 검색
3. 각 나이별 노래 순위
4. 개인정보 변경
5. 뒤로가기
6. 관리자 모드
Input : 3

0. 10대이하 선호 노래
1. 10대 선호 노래
2. 20대 선호 노래
3. 30대 선호 노래
4. 40대 이상 선호 노래
5. 50대이상 선호 노래 : 50
뒤로가기 :-1
Input : 2

20대가 선호하는 노래 상위 n개 보기
Input : 4

1. 보여줄게 - (에일리) - Time : 04:13
2. 내 사람 - (SG워너비) - Time : 04:13
3. 주문(19) - (동방신기) - Time : 03:35
4. 사계 - (태연) - Time : 03:13

음원 듣기(번호 선택)
뒤로가기 : Q
Input :

```

Change_Information.py

이 파일은 개인정보 변경에 관한 파일이다. id를 제외한 모든 **정보 변경**이 가능하고, **결제**를 통해 전체 듣기 기능을 수행 할 수 있다.

change_information_main()

이 함수는 **User_Main.py**에서 4번(개인정보 변경)을 입력했을 때 처음으로 호출되는 함수다. 호출되었을 때, **get_user_paymentday**를 통해 유저의 결제날짜를 가져온다. 결제 날짜로부터 30일동안 전체 듣기 기능 수행 기간을 출력하여 유저에게 정보를 제공한다. 아래 사진과 같이 어떤 기능을 수행할 것인지 유저로부터 Input을 받으면 **switch_change** class에서 알맞는 함수를 매칭하여 호출하게 된다. **switch_change**는 자바에서의 switch-case의 기능을 수행한다. ci_id의 전역 변수의 경우 **계정삭제**하게 되면 전역변수의 값이 **None**된다. None이 되었다면 False를 return하여 **User_Main.py**에서도 바로 함수가 종료되어 로그인창으로 넘어가도록 한다. 개인정보 변경 기능을 계속 이용할 것인지의 여부는 **is_off**변수를 통해 확인한다. 뒤로가기를 눌렀을 때만 **is_off**가 False가 되어 이 함수가 종료된다.

```

전곡 듣기 가능 날짜 : 2020 - 12 - 1 ~ 2020 - 12 - 31

1. 비밀번호 변경
2. email 변경
3. 핸드폰 번호 변경
4. 주소 변경
5. 이름 변경
6. 결제하기
7. 계정삭제
8. 뒤로가기
Input :

```

change_Inform()

switch_change에서는 코드 간결화를 위해 비밀번호, 이메일,핸드폰,주소,이름 변경의 쿼리들을 Directory 구조로 되어, 알맞는 쿼리를 이 함수에서 수행하게 된다. 1~5번까지의 기능이 수행될 때 이 함수가 호출되는데 알맞

는 쿼리를 호출하여 정보를 변경하도록 한다.

```

1. 비밀번호 변경
2. email 변경
3. 핸드폰 번호 변경
4. 주소 변경
5. 이름 변경
6. 결제하기
7. 계정삭제
8. 뒤로가기
9 Input : 1

```

```

변경할 password 입력
Input : 12345

```

payment_now()

이 함수는 6번(결제하기)를 입력 받았을 때 수행하게 된다. 가상으로 결제하게 된다면 **update_user_payment** 쿼리를 통해 결제 날짜를 현재로 변경하고 **update_user_subscriber** 쿼리를 통해 구독기간은 1 증가시킨다.

```

Input : 6
결제하시겠습니까?(Y/N)
뒤로가기 : Q
Input : y

결제가 완료되었습니다.
전곡 듣기 가능 날짜 : 2020 - 12 - 2 ~ 2021 - 1 - 1

```

delete_user()

이 함수는 7번(계정 삭제)를 입력 받았을 때 수행되는 함수다. **delete_id** 쿼리를 통해 DB에서 계정을 삭제한다. 스트리밍 구독자의 table에서 해당 튜플이 삭제 되면 나머지 플레이리스트 정보 등 관련된 정보는 cascade를 통해서 삭제가 된다. 삭제가 된다면 **ci_id** 변수를 False로 수행하게 하여 **Login** 화면으로 넘어가게 한다.

```

1. 비밀번호 변경
2. email 변경
3. 핸드폰 번호 변경
4. 주소 변경
5. 이름 변경
6. 결제하기
7. 계정삭제
8. 뒤로가기
9 Input : 7

정말로 삭제하시겠습니까?(Y/N)
Input : y

삭제 되었습니다.

아이디가 존재하나요?(Y/N)
"N" 입력시 회원가입란이 나옵니다.
종료하실려면 0를 입력해주세요
Input :

```