

Final Project: DMBS 프로그램 개발

컴퓨터소프트웨어학부

2022085069 손주은

목차

1. ER diagram / Relational diagram

- 1) ER diagram 수정사항
- 2) Relational diagram 수정사항

2. 코드 설명

- 1) 디렉토리 구조 설명
- 2) User 코드 설명 – User_controllers, User Views
- 3) Admin 코드 설명 – Admin_controllers, Admin Views
- 4) 그 외 코드 설명 – main.py, database.py

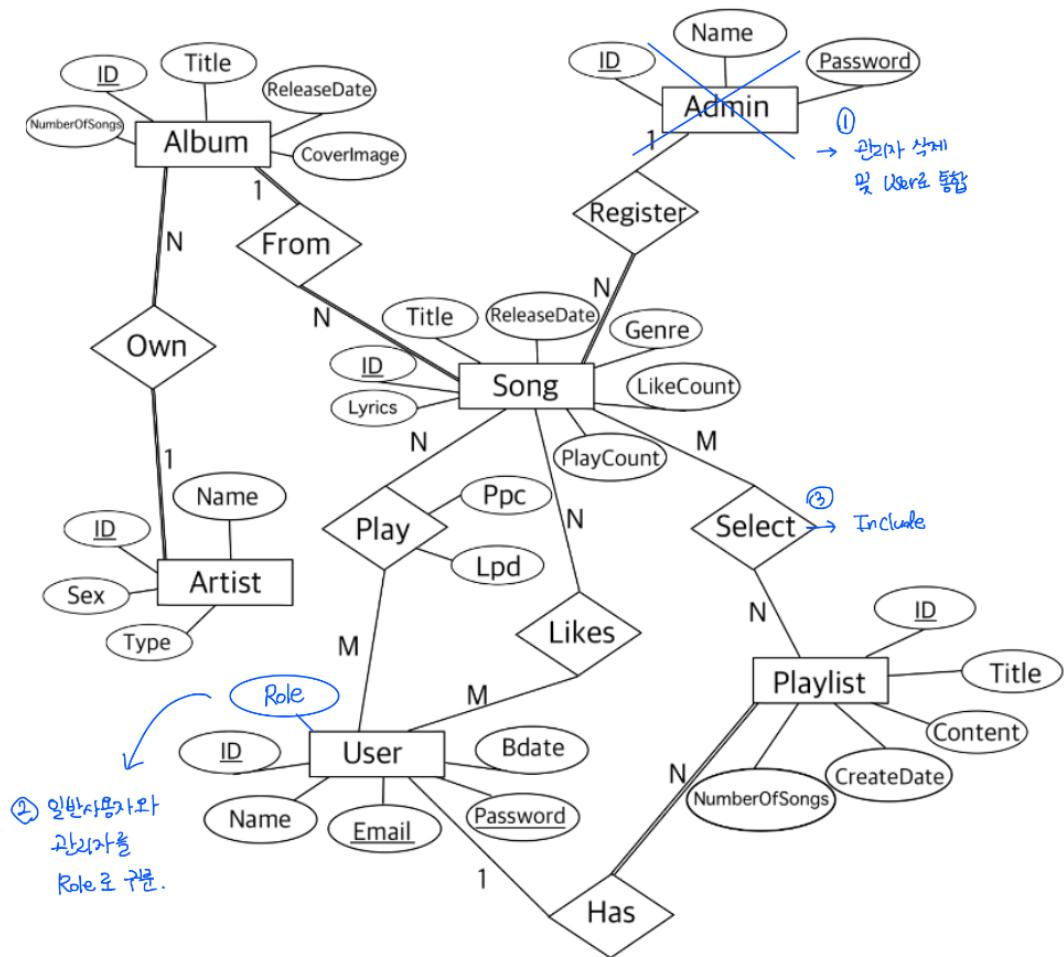
3. 수행 결과 (스크린샷)

- 1) User 입장
- 2) Admin 입장

4. SQL 문 명세

1. ER diagram / Relational diagram

1) ER diagram 수정사항



① 관리자 삭제 및 User로 통합

- 관리자는 1명만 두는 것으로 정하였고, 따라서 관리자 테이블이 별도로 필요하지 않다고 생각해서 삭제하였다.

② User table에 Role attribute 추가

- 관리자를 User table에 통합함에 따라 역할 구분이 필요해져서 Role attribute를 추가하였다.
- 일반 사용자는 'User', 관리자는 'Admin'으로 구분한다.

③ Select → Include

- Select가 예약어라서 Include로 테이블명을 수정하였다.

2) Relational diagram 수정사항



③ Select → Include로 수정

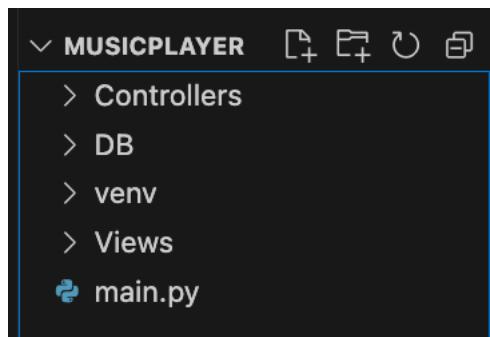
- 앞서 ER model에서 수정한 바와 동일하게 relational model에서도 수정하였다.

④ From → Album으로 수정

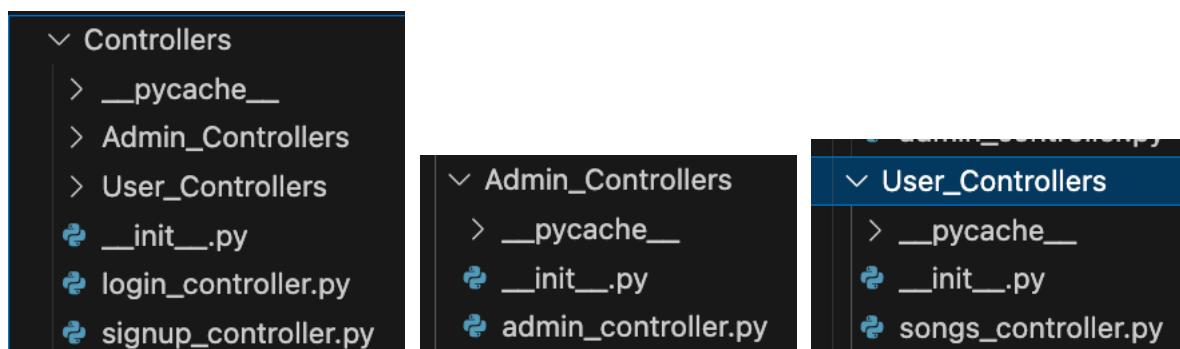
- 수정사항 ③과 마찬가지로 from이 예약어라서 Album으로 수정하였다.

2. 코드 설명

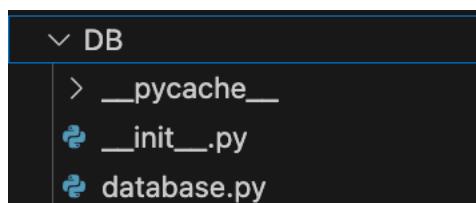
1) 디렉토리 구조 설명



- 전체 구조는 위와 같다.
- 크게 Controllers, DB, Views 3개의 폴더와 main.py로 구성된다.
- Venv 폴더는 python과 pip이 설치된 가상환경과 관련된 폴더이다.



- Controllers 폴더는 위와 같이 구성된다.
- 크게 Admin_Controllers, User_Controllers 2개의 폴더와 login_controller.py, signup_controller.py로 구성된다.
- Admin_Controllers 내부에는 admin_controller.py 파일이 존재한다.
- User_Controllers 내부에는 songs_controller.py 파일이 존재한다.
- Controllers 코드는 DB와 프로그램 사이에서 수행되는 작업을 담당한다.



- DB 폴더는 위와 같이 database.py 파일이 존재한다.
- __pycache__ 파일은 자동으로 생성된 파일이다.
- __init__.py 파일은 디렉토리 구조를 명확히 하기 위해 추가한 파일로, 비어있다.
- Database.py는 DB와 프로그램을 연결하는 코드를 담고 있다.

<pre> <--> Admin > __pycache__ <--> __init__.py <--> admin_main.py <--> edit_albums_view.py <--> edit_artists_view.py <--> edit_songs_view.py <--> manage_albums.py <--> manage_artists.py <--> manage_songs.py </pre>	<pre> <--> User > __pycache__ <--> __init__.py <--> liked_songs_view.py <--> manage_playlists.py <--> search_song.py <--> songs_view.py <--> user_main.py </pre>
<pre> <--> Views > __pycache__ > Admin > User <--> __init__.py <--> login.py <--> signup.py </pre>	

- Views 폴더는 위와 같이 구성된다.
- Views 폴더는 프로그램을 구성하는 화면에 대한 코드를 담고있다.
- 크게 Admin, User 2개의 폴더와, login.py, signup.py 파일로 구성된다.
- Admin 폴더는 songs, artists, albums를 manage하거나 edit하는 부분과 main 화면으로 구분된다.
- User 폴더는 main 화면과 곡 보기, 찾기, 플레이리스트 관리하기, 좋아요 표시한 곡들 모아 보기 화면으로 구분된다.

2) User 코드 설명

Views/User/user_main.py

```

3   class UserMain:
4       def __init__(self, logged_in_user):
5           self.logged_in_user = logged_in_user
6
7       def display(self):
8           while True:
9               print("\n메인 화면")
10              print("0. 로그아웃")
11              print("1. 곡 보기")
12              print("2. 곡 찾기")
13              print("3. 플레이리스트 관리하기")
14              print("4. 좋아요 표시한 곡 보기")
15
    
```

- 일반 사용자가 보게 되는 메인 화면의 구성이다.

```

16     try:
17         choice = int(input("원하는 작업을 선택하세요: "))
18         if choice == 0:
19             self.go_to_login_page()
20             break
21
22         elif choice == 1:
23             self.view_songs()
24             break
25
26         elif choice == 2:
27             self.search_song()
28             break
29
30         elif choice == 3:
31             self.manage_playlists()
32             break
33
34         elif choice == 4:
35             self.view_liked_songs()
36             break
37
38         else:
39             print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
40
41     except ValueError:
42         print("숫자를 입력하세요.")
43
44     def go_to_login_page(self):
45         print("Logging out...")
46         from Views.login import LoginView
47         login_view = LoginView()
48         login_view.display()
49
50
51     def view_songs(self):
52         from Views.User.songs_view import SongsView
53         songs_view = SongsView(self.logged_in_user)
54         songs_view.display()
55
56     def search_song(self):
57         from Views.User.search_song import SearchSong
58         search_view = SearchSong(self.logged_in_user)
59         search_view.display()
60
61     def manage_playlists(self):
62         from Views.User.manage_playlists import ManagePlaylists
63         manage_playlists_view = ManagePlaylists(self.logged_in_user)
64         manage_playlists_view.display()
65
66     def view_liked_songs(self):
67         from Views.User.liked_songs_view import LikedSongsView
68         liked_songs_view = LikedSongsView(self.logged_in_user)
69         liked_songs_view.display()
70
71

```

- 사용자가 선택한 번호에 따라 수행하는 작업이 달라진다.
- 각 옵션은 각각의 뷰로 정의되어 수행되는 흐름이다.

Views/User/songs_view.py

```
1   from Controllers.User.Controllers.songs_controller import SongsController
```

- Controller에 있는 코드가 필요하기 때문에 import 해주었다.

```

2   class Songsview:
3       def __init__(self, logged_in_user):
4           self.controller = SongsController()
5           self.logged_in_user = logged_in_user
6
7       def display(self):
8           while True:
9               songs = self.controller.get_all_songs()
10              print("\n==== 곡 목록 ===")
11              print(f"{'ID':<5}{'제목':<20}{'아티스트':<20}")
12              print("-" * 50)
13
14              for song in songs:
15                  print(f"{song['Song_id']:<5}{song['Title']:<20}{song['AtName']:<20}")
16                  print("-" * 50)
17
18              print("0. 이전 메뉴로")
19              print("1. 재생하기")
20              print("2. 곡 정보 보기")
21              print("3. 좋아요 표시 / 취소")
22              print("4. 플레이리스트에 곡 추가하기") ## 수정하기
23
24              choice = int(input("원하는 작업을 선택하세요: ").strip())
25              if choice == 0:
26                  self.go_to_main_page()
27                  break
28                  #return
29              elif choice == 1:
30                  self.play_song(songs)
31              elif choice == 2:
32                  self.show_song_info(songs)
33              elif choice == 3:
34                  self.toggle_like(songs)
35              elif choice == 4:
36                  self.add_song_to_playlist(songs)
37              else:
38                  print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")

```

- 메인 화면에서 1번을 눌렀을 때 이동하는 화면이다.
- 이 화면에서는 곡 재생하기/ 곡 정보보기/ 좋아요 표시 및 취소하기/ 플레이리스트에 추가하기 기능을 제공한다.

```

47     def play_song(self, songs):
48         song_id = input("재생할 곡의 ID를 입력하세요: ").strip()
49         song = next((song for song in songs if str(song['Song_id']) == song_id), None)
50         if song:
51             user_id = self.logged_in_user['User_id']
52             self.controller.increment_play_count(user_id, song_id)
53             print(f"\n▶ {song['Title']} by {song['AtName']} 재생 중 ... \n")
54         else:
55             print("유효하지 않은 ID입니다. 다시 시도하세요.\n")

```

- 곡 재생하기를 선택한 경우, 재생할 곡의 ID를 입력 받고 해당 곡을 재생한다.

```

85     def show_song_info(self, songs):
86         song_id = input("정보를 확인할 곡의 ID를 입력하세요: ").strip()
87         song = next((song for song in songs if str(song['Song_id']) == song_id), None)
88         if song:
89             print(f"\n==== 곡 정보 ====")
90             print(f"ID: {song['Song_id']}")
91             print(f"제목: {song['Title']}")
92             print(f"아티스트: {song['AtName']}")
93             print(f"발매일: {song['ReleaseDate']}")
94             print(f"장르: {song['Genre']}")
95             print(f"앨범명: {song['AbTitle']}")
96             #발매일,
97             print("=". * 30 + "\n")
98         else:
99             print("\n유효하지 않은 ID입니다. 다시 시도하세요.\n")

```

- 곡 정보보기를 선택한 경우, ID를 입력 받고 해당 곡의 정보를 화면에 제시한다.

```

57     def toggle_like(self, songs):
58         # 좋아요 토클 기능
59         song_id = input("좋아요를 변경할 곡의 ID를 입력하세요: ").strip()
60         song = next((song for song in songs if str(song['Song_id']) == song_id), None)
61         if song:
62             user_id = self.logged_in_user['User_id']
63
64             if self.controller.check_like_status(user_id, song_id):
65                 # 이미 좋아요를 누른 경우 -> 좋아요 취소
66                 confirm = input("이미 좋아요를 누른 곡입니다. 좋아요를 취소하시겠습니까? (y/n): ").strip().lower()
67                 if confirm == 'y':
68                     self.controller.remove_like(user_id, song_id)
69                     self.controller.decrement_like_count(song_id)
70                     print("좋아요가 취소되었습니다.")
71                 else:
72                     print("좋아요를 유지합니다.")
73             else:
74                 # 좋아요를 누르지 않은 경우 -> 좋아요 추가
75                 confirm = input("이 곡을 좋아요 하시겠습니까? (y/n): ").strip().lower()
76                 if confirm == 'y':
77                     self.controller.add_like(user_id, song_id)
78                     self.controller.increment_like_count(song_id)
79                     print("좋아요가 추가되었습니다.")
80                 else:
81                     print("좋아요를 취소하였습니다.")
82         else:
83             print("유효하지 않은 ID입니다. 다시 시도하세요.\n")

```

- 좋아요 표시 및 취소를 선택한 경우, ID를 입력 받고 해당 곡의 현재 좋아요 값에 따라 toggle시킨다.

```

103  def add_song_to_playlist(self, songs):
104      try:
105          song_id = int(input("플레이리스트에 추가할 곡의 ID를 입력하세요: ").strip())
106          selected_song = next((song for song in songs if song['Song_id'] == song_id), None)
107          if not selected_song:
108              print("유효한 곡 ID를 선택하세요.")
109              return
110      except ValueError:
111          print("유효한 숫자를 입력하세요.")
112          return
113
114      # 사용자 플레이리스트 목록 가져오기
115      playlists = self.controller.get_user_playlists(self.logged_in_user["User_id"])
116      if not playlists:
117          print("플레이리스트가 없습니다.")
118          return
119
120      # 플레이리스트 선택
121      print("\n==== 내 플레이리스트 목록 ===")
122      for playlist in playlists:
123          print(f"ID: {playlist['Playlist_id']}, 이름: {playlist['PlTitle']}, 곡 수: {playlist['PnumberOfSongs']} ")
124      try:
125          playlist_id = int(input("곡을 추가할 플레이리스트 ID를 입력하세요: ").strip())
126          selected_playlist = next((pl for pl in playlists if pl['Playlist_id'] == playlist_id), None)
127          if not selected_playlist:
128              print("유효한 플레이리스트 ID를 선택하세요.")
129              return
130      except ValueError:
131          print("유효한 숫자를 입력하세요.")
132          return
133
134      # 곡 추가
135      self.controller.add_song_to_playlist(playlist_id, song_id)

```

- 플레이리스트에 곡 추가하기를 선택한 경우, 플레이리스트와 곡 ID를 순서대로 입력 받고 해당 곡을 입력 받은 플레이리스트에 넣는다.

Views/User/search_song.py

```

9   def display(self):
10      # 검색 화면 출력 및 사용자 입력 처리
11      print("\n==== 곡 검색 ===")
12      query = input("검색할 곡 제목이나 아티스트 이름을 입력하세요: ").strip()
13      if not query:
14          print("검색어를 입력해주세요.")
15          return
16
17      #검색 실행
18      results = self.controller.search_songs(query)
19
20      #검색 결과 출력
21      if results:
22          print("\n==== 검색 결과 ===")
23          print(f"{'ID':<5}{{'제목':<20}{['아티스트']:<20}}")
24          print("-*50")
25          for song in results:
26              print(f"{song['Song_id']:<5}{song['Title']:<20}{song['AtName']:<20}")
27              print("-*50")
28
29      else:
30          print("검색 결과가 없습니다.")
31
32
33      self.handle_results_menu(results)

```

- 메인 화면에서 사용자가 2번을 눌렀을 때 이동하는 화면이다.
- 곡 제목이나 아티스트 이름을 입력 받아서 검색한다.

```

35     def handle_results_menu(self, results):
36         while True:
37             print("0. 이전 메뉴로")
38             print("1. 재생하기")
39             print("2. 곡 정보 보기")
40             try:
41                 choice = int(input("원하는 작업을 선택하세요: ").strip())
42                 if choice == 0:
43                     self.songs_view.go_to_main_page()
44                     break
45                 elif choice == 1:
46                     self.songs_view.play_song(results)
47                 elif choice == 2:
48                     self.songs_view.show_song_info(results)
49                 else:
50                     print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
51             except ValueError:
52                 print("숫자를 입력하세요.")

```

- 검색된 곡이 있을 때 나타나는 메뉴 화면이다. 사용자 입력에 따라 작업을 수행한다. songs_view.py 의 코드를 이용한다.

Views/User/manage_playlist.py

```

9     def display(self):
10        while True:
11            #현재 사용자의 플레이리스트 목록 가져오기
12            print("\n--- 플레이리스트 ---")
13            playlists = self.songs_controller.get_user_playlists(self.user_data["User_id"])
14            if playlists:
15                for playlist in playlists:
16                    print(f"ID: {playlist['Playlist_id']}, 이름: {playlist['PlTitle']}, 곡 수: {playlist['PnumberOfSongs']} ")
17            else:
18                print("플레이리스트가 없습니다.")
19
20            print("0. 이전 메뉴로")
21            print("1. 새 플레이리스트 생성")
22            print("2. 플레이리스트 삭제")
23            print("3. 플레이리스트 관리")
24
25            try:
26                choice = int(input("원하는 작업을 선택하세요: ").strip())
27                if choice == 0:
28                    self.songs_view.go_to_main_page()
29                    break
30                elif choice == 1:
31                    self.create_playlist()
32                elif choice == 2:
33                    self.delete_playlist()
34                elif choice == 3:
35                    self.edit_playlist()
36                else:
37                    print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
38            except ValueError:
39                print("숫자를 입력하세요.")

```

- 메인 화면에서 사용자가 3번을 눌렀을 때 이동하는 화면이다.
- 플레이리스트 목록을 보여주고, 메뉴를 제시해서 사용자의 입력을 받는다.

```

41     def create_playlist(self):
42
43         playlist_name = input("새 플레이리스트 이름을 입력하세요: ").strip()
44         if not playlist_name:
45             print("플레이리스트 이름을 입력해야 합니다.")
46             return
47         self.songs_controller.create_playlist(self.user_data["User_id"], playlist_name)

```

- 위의 화면에서 1번을 입력했을 때, 플레이리스트 이름을 입력 받고 생성한다.

```

122     def delete_playlist(self):
123         # 사용자의 플레이리스트 목록 가져오기
124         playlists = self.songs_controller.get_user_playlists(self.user_data["User_id"])
125         if not playlists:
126             print("삭제할 플레이리스트가 없습니다.")
127             return
128         print("\n==== 플레이리스트 목록 ===")
129         for playlist in playlists:
130             print(f"ID: {playlist['Playlist_id']}, 이름: {playlist['PlTitle']}, 곡 수: {playlist['PnumberOfSongs']}")
131         try:
132             # 삭제할 플레이리스트 선택
133             playlist_id = int(input("삭제할 플레이리스트 ID를 입력하세요: ").strip())
134             selected_playlist = next((pl for pl in playlists if pl['Playlist_id'] == playlist_id), None)
135             if not selected_playlist:
136                 print("유효한 플레이리스트 ID를 선택하세요.")
137                 return
138             # 삭제 확인
139             confirm = input(f"\n{selected_playlist['PlTitle']} 플레이리스트를 삭제하시겠습니까? (Y/N): ").strip().lower()
140             if confirm == 'y':
141                 self.songs_controller.delete_playlist(playlist_id)
142                 print(f"플레이리스트 '{selected_playlist['PlTitle']}' 가 삭제되었습니다.")
143             else:
144                 print("삭제가 취소되었습니다.")
145         except ValueError:
146             print("유효한 숫자를 입력하세요.")

```

- 위의 화면에서 2번을 입력했을 때, 삭제할 플레이리스트 ID를 입력 받아 삭제한다.
- 삭제하기 전에는 진짜 삭제할 것인지 확인하는 절차를 거친다.

```

49     def edit_playlist(self):
50         try:
51             print("\n==== 플레이리스트 ===")
52             playlists = self.songs_controller.get_user_playlists(self.user_data["User_id"])
53             if playlists:
54                 for playlist in playlists:
55                     print(f"ID: {playlist['Playlist_id']}, 이름: {playlist['PlTitle']}, 곡 수: {playlist['PnumberOfSongs']}")
56             else:
57                 print("플레이리스트가 없습니다.")
58
59             playlist_id = int(input("수정할 플레이리스트 ID를 입력하세요: ").strip())
60             # 사용자의 플레이리스트 가져오기
61             playlists = self.songs_controller.get_user_playlists(self.user_data["User_id"])
62             selected_playlist = next((pl for pl in playlists if pl['Playlist_id'] == playlist_id), None)
63             if not selected_playlist:
64                 print("유효한 플레이리스트 ID를 선택하세요.")
65                 return
66
67             # 선택한 플레이리스트의 곡 목록
68             songs_in_playlist = self.songs_controller.get_songs_in_playlist(selected_playlist['Playlist_id'])
69             print(f"\n==== 플레이리스트 '{selected_playlist['PlTitle']}' 의 곡 목록 ===")
70             if not songs_in_playlist:
71                 print("플레이리스트에 곡이 없습니다.")
72             else:
73                 for song in songs_in_playlist:
74                     print(f"ID: {song['Song_id']}, 제목: {song['Title']}, 아티스트: {song['Artist']}")

```

- 위의 화면에서 3번을 입력했을 때, ID를 입력 받아 수정 작업을 수행한다.

```

76     #수정 작업 선택
77     print("\n0. 이전메뉴로")
78     print("1. 곡 삭제하기")
79     choice = int(input("원하는 작업을 선택하세요: ").strip())
80     if choice == 0:
81         # 이전 메뉴로
82         return
83     elif choice == 1:
84         self.remove_song_from_playlist[playlist_id]
85     else:
86         print("잘못된 입력입니다.")
87     except ValueError:
88         print("유효한 숫자를 입력하세요.")

```

- 수정 작업의 종류는 이전 메뉴로 가기와 곡 삭제하기로 구성된다.

```

91     def remove_song_from_playlist(self, playlist_id):
92         # 선택한 플레이리스트 내 곡 목록 가져오기
93         songs_in_playlist = self.songs_controller.get_songs_in_playlist(playlist_id)
94         if not songs_in_playlist:
95             print("플레이리스트에 곡이 없습니다.")
96             return
97
98         print("\n==== 플레이리스트 내 곡 목록 ===")
99         for song in songs_in_playlist:
100             print(f"ID: {song['Song_id']}, 제목: {song['Title']}, 아티스트: {song['Artist']}")  

101
102     try:
103         # 삭제할 곡 선택
104         song_id = int(input("삭제할 곡 ID를 입력하세요: ").strip())
105         selected_song = next((song for song in songs_in_playlist if song['Song_id'] == song_id), None)
106         if not selected_song:
107             print("유효한 곡 ID를 선택하세요.")
108             return
109
110         # 곡 삭제
111         self.songs_controller.remove_song_from_playlist(playlist_id, song_id)
112         print(f"곡 '{selected_song['Title']}'가 플레이리스트에서 삭제되었습니다.")
113     except ValueError:
114         print("유효한 숫자를 입력하세요..")

```

- 플레이리스트에서 곡을 삭제하기를 원하는 경우, 곡 ID를 입력 받아 해당 곡을 삭제한다.

```

116     def delete_playlist(self):
117         # 사용자의 플레이리스트 목록 가져오기
118         playlists = self.songs_controller.get_user_playlists(self.user_data["User_id"])
119         if not playlists:
120             print("삭제할 플레이리스트가 없습니다.")
121             return
122         print("\n==== 플레이리스트 목록 ===")
123         for playlist in playlists:
124             print(f"ID: {playlist['Playlist_id']}, 이름: {playlist['PlTitle']}, 곡 수: {playlist['PnumberOfSongs']}")  

125     try:
126         # 삭제할 플레이리스트 선택
127         playlist_id = int(input("삭제할 플레이리스트 ID를 입력하세요: ").strip())
128         selected_playlist = next((pl for pl in playlists if pl['Playlist_id'] == playlist_id), None)
129         if not selected_playlist:
130             print("유효한 플레이리스트 ID를 선택하세요.")
131             return
132         # 삭제 확인
133         confirm = input(f"'{selected_playlist['PlTitle']}' 플레이리스트를 삭제하시겠습니까? (Y/N): ").strip().lower()
134         if confirm == 'y':
135             self.songs_controller.delete_playlist(playlist_id)
136             print(f"플레이리스트 '{selected_playlist['PlTitle']}'가 삭제되었습니다.")
137         else:
138             print("삭제가 취소되었습니다.")
139     except ValueError:
140         print("유효한 숫자를 입력하세요..")

```

- 이 함수는 플레이리스트 자체를 삭제하는 함수이다.

Views/User/liked_songs_view.py

```
9     def display(self):
10
11         # 좋아요 누른 곡 가져오기
12         liked_songs = self.controller.get_liked_songs(self.logged_in_user['User_id'])
13         print("\n==== 좋아요 누른 곡 목록 ===")
14         if liked_songs:
15             print(f"{'ID':<5}{['제목':<20}{['아티스트':<20}''")
16             print("-"*50)
17             for song in liked_songs:
18                 print(f"{song['Song_id']:<5}{song['Title']:<20}{song['AtName']:<20}")
19                 print("-"*50)
20         else:
21             print("좋아요를 누른 곡이 없습니다.")
22         return
23     self.handle_results_menu(liked_songs)
```

- 사용자가 좋아요 표시를 한 곡들만 모아보는 기능이다.

```
26     def handle_results_menu(self, liked_songs):
27         while True:
28             print("0. 이전 메뉴로")
29             if liked_songs:
30                 print("1. 곡 재생하기")
31                 print("2. 곡 정보 보기")
32                 print("3. 좋아요 표시/수정하기\n")
33             try:
34                 choice = int(input("원하는 작업을 선택하세요: ").strip())
35                 if choice == 0:
36                     self.songs_view.go_to_main_page()
37                     break
38                 elif choice == 1:
39                     self.songs_view.play_song(liked_songs)
40                 elif choice == 2:
41                     self.songs_view.show_song_info(liked_songs)
42                 elif choice == 3:
43                     self.songs_view.toggle_like(liked_songs)
44                 else:
45                     print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
46             except ValueError:
47                 print("숫자를 입력하세요.")
```

- 좋아요 표시를 한 곡들을 모아볼 때도 곡을 재생하거나 곡 정보를 확인하거나 혹은 좋아요를 취소할 수 있도록 메뉴를 제공한다.

Controllers/User.Controllers/songs_controller.py

```
7     def get_all_songs(self):
8         query = """
9             SELECT
10                 SONG.Song_id,
11                 SONG.Title,
12                 SONG.ReleaseDate,
13                 SONG.Genre,
14                 Album.AbTitle,
15                 ARTIST.AtName
16             FROM
17                 SONG
18             JOIN
19                 ALBUM
20             ON
21                 SONG.Album = ALBUM.Album_id
22             JOIN
23                 ARTIST
24             ON ALBUM.Artist = ARTIST.Artist_id
25         """
26
27     try:
28         return self.db.execute_query(query)
29     except Exception as e:
30         print(f"DB에서 곡 목록을 불러오는 중 오류가 발생했습니다.: {e}")
31         return []
32
33     def get_liked_songs(self, user_id):
34         query = """
35             SELECT
36                 SONG.Song_id,
37                 SONG.Title,
38                 SONG.ReleaseDate,
39                 SONG.Genre,
40                 Album.AbTitle,
41                 ARTIST.AtName
42             FROM
43                 LIKES
44             JOIN
45                 SONG
46             ON
47                 LIKES.LSong = SONG.Song_id
48             JOIN
49                 ALBUM
50             ON
51                 SONG.Album = ALBUM.Album_id
52             JOIN
53                 ARTIST
54             ON
55                 ALBUM.Artist = ARTIST.Artist_id
56             WHERE
57                 LIKES.Who = %s
58         """
59
60     try:
61         return self.db.execute_query(query, (user_id))
62     except Exception as e:
63         print(f"좋아요 누른 곡을 불러오는 중 오류가 발생했습니다.: {e}")
64         return []
```

- 곡 리스트를 DB에서 가져오는 함수들이다.

```
65     def increment_play_count(self, user_id, song_id):
66         try:
67             # 전체 재생 횟수 증가
68             update_song_query = "UPDATE SONG SET PlayCount = PlayCount + 1 WHERE Song_id = %s"
69             self.db.execute_query(update_song_query, (song_id))
70             # 개인 재생 횟수 업데이트
71             check_play_query = "SELECT Ppc FROM PLAY WHERE Who = %s AND PSong = %s"
72             play_record = self.db.execute_query(check_play_query, (user_id, song_id))
73             if play_record:
74                 # 이미 재생 기록이 있으면
75                 update_play_query = "UPDATE PLAY SET Ppc = Ppc + 1, Lpd = NOW() WHERE Who = %s AND PSong = %s"
76                 self.db.execute_query(update_play_query, (user_id, song_id))
77             else:
78                 # 재생 기록이 없는 경우
79                 insert_play_query = "INSERT INTO PLAY (Who, PSong, Ppc, Lpd) VALUES (%s, %s, 1, NOW())"
80                 self.db.execute_query(insert_play_query, (user_id, song_id))
81                 default_playlist_id = self.get_or_create_default_playlist(user_id)
82                 self.add_song_to_playlist(default_playlist_id, song_id)
83                 print(f"재생 횟수가 업데이트되었습니다.: 사용자 {user_id}, 곡 {song_id}")
84         except Exception as e:
85             print("재생 횟수 업데이트 중 오류 발생: {e}")
```

- 곡을 재생한 경우, 전체 재생 횟수와 개인 재생 횟수를 1씩 증가시킨다.
- 재생 기록이 없었던 경우, 재생 기록을 남기고 플레이리스트에 추가한다.
- 사용자가 최초로 곡을 재생한 경우라면, default playlist도 이때 만들어지게 된다.

```

87     def search_songs(self, query):
88         search_query = """
89             SELECT
90                 SONG.Song_id,
91                 SONG.Title,
92                 SONG.ReleaseDate,
93                 SONG.Genre,
94                 ARTIST.AtName,
95                 ALBUM.AbTitle
96             FROM
97                 SONG
98             JOIN
99                 ALBUM
100            ON
101                SONG.Album = ALBUM.Album_id
102            JOIN
103                ARTIST
104            ON
105                ALBUM.Artist = ARTIST.Artist_id
106            WHERE
107                SONG.Title Like %s OR ARTIST.AtName Like %
108            """
109        try:
110            return self.db.execute_query(search_query, (f"%{query}%", f"%{query}%"))
111        except Exception as e:
112            print(f"곡 검색 중 오류 발생: {e}")
113            return []

```

- 곡 제목과 아티스트 이름으로 DB에서 일치하는 곡을 찾는 함수이다.

```

115     def check_like_status(self, user_id, song_id):
116         # 좋아요 상태 확인
117         query = "SELECT 1 FROM LIKES WHERE Who = %s AND LSong = %s"
118         result = self.db.execute_query(query, (user_id, song_id))
119         return bool(result)
120     def add_like(self, user_id, song_id):
121         query = "INSERT INTO LIKES (Who, LSong) VALUES (%s, %s)"
122         self.db.execute_query(query, (user_id, song_id))
123
124     def remove_like(self, user_id, song_id):
125         query = "DELETE FROM LIKES WHERE Who = %s AND LSong = %s"
126         self.db.execute_query(query, (user_id, song_id))
127

```

- 좋아요와 관련된 함수들이다.
- 좋아요 상태를 확인하고, 좋아요 기록이 없었다면 추가하고, 있었다면 삭제할 수 있다.

```

128     def increment_like_count(self, song_id):
129         query = """
130             UPDATE SONG SET LikeCount = LikeCount + 1
131             WHERE Song_id = %s
132             """
133
134         try:
135             self.db.execute_query(query, (song_id))
136         except Exception as e:
137             print(f"LikeCount 증가 실패: {e}")
138
139     def decrement_like_count(self, song_id):
140         query = "UPDATE SONG SET LikeCount = GREATEST(0, LikeCount - 1) WHERE Song_id = %s"
141         self.db.execute_query(query, (song_id))
142

```

- 곡의 전체 좋아요 수를 증가시키거나 감소시키는 함수이다.
- 이 변수를 이용해서 인기곡을 파악할 수 있다.

```

144     def create_playlist(self, user_id, playlist_name, content = None):
145         query_check = """
146             SELECT Playlist_id
147             FROM PLAYLIST
148             WHERE Owner = %s AND PlTitle = %s
149             """
150
151         existing_playlist = self.db.execute_query(query_check, (user_id, playlist_name))
152         if existing_playlist:
153             print("이미 동일한 이름의 플레이리스트가 존재합니다.")
154             return
155         query_add = """
156             INSERT INTO PLAYLIST (PlTitle, Content, CreateDate, Owner, PnumberOfSongs)
157             VALUES (%s, %s, NOW(), %s, 0)
158             """
159
160         self.db.execute_query(query_add, (playlist_name, content, user_id))
161         #방금 추가한 플레이리스트 ID 가져오기
162         new_playlist_id = self.db.get_last_insert_id()
163         print(f"새로운 플레이리스트가 생성되었습니다. ID: {new_playlist_id}")
164         return new_playlist_id

```

- 플레이리스트를 생성하는 함수다.
- 누가 어떤 플레이리스트를 생성했는지를 저장하며, 동일한 플레이리스트는 허용하지 않는다.

```

164     def get_or_create_default_playlist(self, user_id):
165         # 기본 플레이리스트 확인
166         query_check= """
167             SELECT Playlist_id
168             FROM PLAYLIST
169             WHERE Owner = %s AND PlTitle = '나의 플레이리스트'
170             """
171
172         default_playlist = self.db.execute_query(query_check, (user_id,))
173         if default_playlist:
174             return default_playlist[0]['Playlist_id']
175         query_add = """
176             INSERT INTO PLAYLIST (PlTitle, Content, CreateDate, Owner, PnumberOfSongs)
177             VALUES ('나의 플레이리스트', 'default playlist', NOW(), %s, 0)
178             """
179         self.db.execute_query(query_add, (user_id,))
180         return self.db.get_last_insert_id()

```

- Default playlist를 가져오거나 없다면 생성하는 함수이다.
- 곡을 재생했을 때 플레이리스트에 해당 곡을 추가하는 상황에서 사용된다.

```

182     def add_song_to_playlist(self, playlist_id, song_id):
183         query_check = """
184             SELECT * FROM INCLUDE
185             WHERE WhatPl = %s AND WhichSong = %s
186             """
187
188         existing_entry = self.db.execute_query(query_check, (playlist_id, song_id))
189         if existing_entry:
190             print("이 곡은 이미 플레이리스트에 추가되어 있습니다.")
191             return
192         #곡 추가
193         query_add = """
194             INSERT INTO INCLUDE (WhatPl, WhichSong)
195             VALUES (%s, %s)
196             """
197
198         self.db.execute_query(query_add, (playlist_id, song_id))
199         # PnumberOfSongs 업데이트
200         query_update = """
201             UPDATE PLAYLIST
202             SET PnumberOfSongs = PnumberOfSongs + 1
203             WHERE Playlist_id = %s
204             """
205
206         self.db.execute_query(query_update, (playlist_id,))
207         print("곡이 플레이리스트에 추가되었습니다.")

```

- 플레이리스트에 곡을 추가하는 함수이다.
- 중복을 체크하고, 곡을 추가했다면 PnumberOfSongs를 업데이트 한다.

```

225     def remove_song_from_playlist(self, playlist_id, song_id):
226         # INCLUDE 테이블에서 곡 삭제
227         query_delete = "DELETE FROM INCLUDE WHERE WhatPl = %s AND WhichSong = %s"
228         self.db.execute_query(query_delete, (playlist_id, song_id))
229         # PLAYLIST 테이블의 곡 수 업데이트
230         query_update = """
231             UPDATE PLAYLIST
232             SET PnumberOfSongs = PnumberOfSongs - 1
233             WHERE Playlist_id = %s
234             """
235
236         self.db.execute_query(query_update, (playlist_id,))
237         print("곡이 성공적으로 삭제되었습니다.")
238
239     def delete_playlist(self, playlist_id):
240         # INCLUDE 테이블에서 플레이리스트에 포함된 곡 삭제
241         query_delete_includes = "DELETE FROM INCLUDE WHERE WhatPl = %s"
242         self.db.execute_query(query_delete_includes, (playlist_id,))
243         # PLAYLIST 테이블에서 플레이리스트 삭제
244         query_delete_playlist = "DELETE FROM PLAYLIST WHERE Playlist_id = %s"
245         self.db.execute_query(query_delete_playlist, (playlist_id,))
246         print(f"플레이리스트 ID {playlist_id}가 성공적으로 삭제되었습니다.")

```

- remove_song_from_playlist는 플레이리스트에서 곡을 삭제하는 함수이다.
- 역시 PnumberOfSongs를 업데이트 한다.
- delete_playlist는 플레이리스트 자체를 삭제하는 함수로, 플레이리스트에 있는 곡을 DB에서 다 삭제한 후에 플레이리스트를 삭제한다.

```

206     def get_user_playlists(self, user_id):
207         query = """
208             SELECT Playlist_id, PlTitle, PnumberOfSongs
209             FROM PLAYLIST
210             WHERE Owner = %s
211             """
212         return self.db.execute_query(query, (user_id,))
213
214     def get_songs_in_playlist(self, playlist_id):
215         query = """
216             SELECT S.Song_id, S.Title, A.AtName AS Artist, Alb.AbTitle AS Album
217             FROM INCLUDE I
218             JOIN SONG S ON I.WhichSong = S.Song_id
219             JOIN ALBUM Alb ON S.Album = Alb.Album_id
220             JOIN ARTIST A ON Alb.Artist = A.Artist_id
221             WHERE I.WhatPl = %s
222             """
223         return self.db.execute_query(query, (playlist_id,))

```

- 모든 playlist와 playlist에 속하는 곡들을 불러오는 함수이다.

3) Admin 코드 설명

Views/Admin/admin_main.py

```

2 class AdminMain:
3     def __init__(self, user_data):
4         self.user_data = user_data
5     def display(self):
6         while True:
7             print("\n==== 관리자 메인 화면 ===")
8             print("0. 로그아웃")
9             print("1. 곡 관리하기")
10            print("2. 아티스트 관리하기")
11            print("3. 앨범 관리하기")
12            try:
13                choice = int(input("원하는 작업을 선택하세요: "))
14                if choice == 0:
15                    # 로그인 페이지로
16                    self.go_to_login_page()
17                    break
18                elif choice == 1:
19                    self.manage_songs()
20                    break
21                elif choice == 2:
22                    self.manage_artists()
23                    break
24                elif choice == 3:
25                    self.manage_albums()
26                else:
27                    print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
28            except ValueError:
29                print("숫자를 입력하세요.")
34
35     def go_to_login_page(self):
36         from Views.login import LoginView
37         login_view = LoginView()
38         login_view.display()
39
40     def manage_songs(self):
41         from Views.Admin.manage_songs import ManageSongs
42         manage_songs_view = ManageSongs(self.user_data)
43         manage_songs_view.display()
44
45     def manage_artists(self):
46         from Views.Admin.manage_artists import ManageArtists
47         manage_artists_view = ManageArtists(self.user_data)
48         manage_artists_view.display()
49
50     def manage_albums(self):
51         from Views.Admin.manage_albums import ManageAlbums
52         manage_albums_view = ManageAlbums(self.user_data)
53         manage_albums_view.display()

```

- 관리자 메인 화면은 일반 사용자 화면과 구분되며, 관리자가 보게 되는 화면이다.

Views/Admin/manage_songs.py

```
11     def display(self):
12         while True:
13             print("\n==== 곡 관리하기 ====")
14             print("0. 이전 메뉴로")
15             print("1. 추가하기")
16             print("2. 수정 및 삭제하기")
17             try:
18                 choice = int(input("원하는 작업을 선택하세요: "))
19                 if choice == 0:
20                     #이전 메뉴로
21                     self.go_to_main_page()
22                     break
23                 elif choice == 1:
24                     self.add_song()
25                 elif choice == 2:
26                     self.edit_songs_view.display()
27                 else:
28                     print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
29             except ValueError:
30                 print("숫자를 입력하세요.")
```

- 관리자의 메인 화면에서 1번을 눌렀을 때 이동하는 화면이다.
- 곡 관리하기 메뉴에서는 이전메뉴로/ 추가하기 / 수정 및 삭제하기 기능을 제공한다.
- 즉 관리자가 곡을 추가하고 수정 및 삭제할 수 있는 기능이다.

```
37     def add_song(self):
38         print("\n==== 곡 추가하기 ====")
39         print("==== 1. 아티스트 정보 입력 및 확인 ====")
40         artist_name = input("아티스트 이름을 입력하세요: ").strip()
41         # 이름이 중복되면 화면 보여주고 관리자가 선택하게 하기
42         artist_id = self.controller.get_or_add_artist(artist_name)
43
44         print("==== 2. 앨범 정보 입력 및 확인 ====")
45         album_title = input("앨범명을 입력하세요: ").strip()
46         album_release_date = input("발매일을 입력하세요(yyyy-dd-mm): ").strip()
47         album_id = self.controller.get_or_add_album(album_title, album_release_date, artist_id)
48
49         print("==== 3. 곡 정보 입력 및 확인 ====")
50         song_title = input("곡 제목을 입력하세요: ").strip()
51         genre = input("장르를 입력하세요: ").strip()
52         lyrics = input("가사를 입력하세요: ").strip()
53
54         if not song_title:
55             print("곡 제목을 입력하세요.")
56             return
57         # db에 곡 등록
58         self.controller.add_song_to_db(song_title, album_release_date, genre, album_id, lyrics)
```

- 곡을 추가하기 위해서는 앨범 정보가 필요한데, 앨범 정보에서는 아티스트 정보가 필요하다. 따라서 아티스트 → 앨범 → 곡 순서대로 정보를 확인 및 입력 받는다.

Views/Admin/edit_songs_view.py

- 관리자의 메인 화면에서 2번을 눌렀을 때 이동하게 되는 화면으로, 뷰를 분리했다.

```
51     def handle_song_edit_or_delete(self, selected_song):
52         print("\n0. 취소하기")
53         print("1. 곡 수정하기")
54         print("2. 곡 삭제하기")
55     try:
56         choice = int(input("원하는 작업을 선택하세요: ").strip())
57         if choice == 0:
58             print("작업이 취소되었습니다.")
59             return
60         elif choice == 1:
61             self.edit_song(selected_song)
62         elif choice == 2:
63             self.delete_song(selected_song)
64         else:
65             print("잘못된 입력이빈다. 유효한 옵션을 선택하세요.")
66     except ValueError:
67         print("숫자를 입력하세요.")
```

- 곡 목록에서 곡을 선택한 후, 수정할지 삭제할지를 입력하는 화면이 제공된다.

```
69     def edit_song(self, selected_song):
70         print("\n== 곡 수정하기 ==")
71         new_title = input(f"새 제목 (현재: {selected_song['Title']}): ").strip() or selected_song['Title']
72         new_release_date = input(f"새 발매일 (현재: {selected_song.get('ReleaseDate', 'N/A')})": "").strip() or selected_song.get('ReleaseDate', 'N/A')
73         new_genre = input(f"새 장르 (현재: {selected_song.get('Genre', 'N/A')})": "").strip() or selected_song.get('Genre', 'N/A')
74         new_lyrics = input(f"새 가사 (현재: {selected_song.get('Lyrics', 'N/A')})": "").strip() or selected_song.get('Lyrics', 'N/A')
75         self.admin_controller.update_songs(selected_song['Song_id'], new_title, new_release_date, new_genre, new_lyrics)
76         print("곡 정보가 성공적으로 수정되었습니다.")
77
78     def delete_song(self, selected_song):
79         print("\n== 곡 삭제하기 ==")
80         confirm = input(f"정말로 '{selected_song['Title']}' 곡을 삭제하시겠습니까? (Y/N): ").strip().lower()
81         if confirm == 'y':
82             self.admin_controller.delete_songs(selected_song['Song_id'])
83             print("곡이 성공적으로 삭제되었습니다.")
84         else:
85             print("삭제가 취소되었습니다.")
```

- 사용자의 입력에 따라 수정하거나 삭제하게 된다.

Views/Admin/manage_artists.py, Views/Admin/manage_albums.py 와

Views/Admin/edit_artists_view.py, Views/Admin/edit_albums.py 역시 비슷한 흐름이다.

4) 그 외 코드 설명

main.py

```
5  def main():
6      while True:
7          print("\n==== 메인 메뉴 ====")
8          print("0. 프로그램 종료")
9          print("1. 로그인")
10         print("2. 회원가입")
11
12     try:
13         choice = int(input("원하는 작업을 선택하세요: ").strip())
14         if choice == 0:
15             print("프로그램을 종료합니다.")
16             break
17         elif choice == 1:
18             login_view = LoginView()
19             result = login_view.display()
20             if result:
21                 print("로그인 성공 ! 다음단계로 이동합니다.")
22             elif result == "exit":
23                 print("프로그램을 종료합니다.")
24         elif choice == 2:
25             signup_view = SignupView()
26             signup_view.display()
27         else:
28             print("잘못된 입력입니다. 유효한 옵션을 선택하세요.")
29     except ValueError:
30         print("유효한 숫자를 입력하세요.")
```

- 전체 프로그램의 메인 화면으로 종료, 로그인, 회원가입 메뉴를 제공한다.

Views/User/signup.py

```
7  def display(self):
8      while True:
9          print("==== 회원가입 화면 ====")
10         username = input("이름을 입력하세요: ").strip()
11         email = input("이메일을 입력하세요: ").strip()
12         password = input("비밀번호를 입력하세요: ").strip()
13         birthdate = input("생년월일을 입력하세요(ex, yyyy-mm-dd): ")
14
15         if not email or not password or not username or not birthdate:
16             print("입력되지 않은 항목이 있습니다.\n")
17             continue
18
19         if self.controller.register_user(username, email, password, birthdate):
20             print("회원가입 성공!\n")
21             return "success" # 성공하면 다음 단계로 이동
22         else:
23             print("회원가입 실패, 다시 시도하세요.\n")
24             retry = input("다시 시도하려면 Enter, 종료하려면 'q'를 입력하세요.").strip().lower()
25             if retry == 'q':
26                 return "exit" # 프로그램 종료
```

- 회원가입 화면이다.

Views/User/login.py

```
 9     def display(self):
10         while True:
11             print("==== 로그인 화면 ===")
12             email = input("이메일을 입력하세요: ").strip()
13             password = input("비밀번호를 입력하세요: ").strip()
14             if not email or not password:
15                 print("이메일과 비밀번호를 모두 입력해주세요.\n")
16                 continue
17             user_data = self.controller.authenticate(email, password)
18             if user_data:
19                 print("로그인 성공!\n")
20                 # 로그인한 사용자 정보 저장
21                 self.logged_in_user = user_data
22                 # Role에 따라 메인 화면 호출
23                 if self.logged_in_user['Role'] == 'Admin':
24                     from Views.Admin.admin_main import AdminMain
25                     admin_main = AdminMain(self.logged_in_user)
26                     admin_main.display()
27                 else:
28                     #UserMain에 사용자 정보 전달
29                     from Views.User.user_main import UserMain
30                     user_main = UserMain(self.logged_in_user)
31                     user_main.display()
32                 break
33             else:
34                 print("로그인 실패, 다시 시도하세요.\n")
35                 retry = input("다시 시도하려면 Enter, 종료하려면 'q'를 입력하세요.").strip().lower()
36                 if retry == 'q':
37                     print("프로그램을 종료합니다.")
38                     import sys
39                     sys.exit()
40             return "exit"
```

- 로그인 화면이며, Role에 따라 메인 화면이 달라진다.

Controllers/signup_controller.py

```
 1  import bcrypt
 2  from DB.database import Database
 3
 4  class SignupController:
 5      def __init__(self):
 6          self.db = Database()
 7
 8      def register_user(self, username, email, password, birth_date):
 9          # 비밀번호 해시 처리
10          hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')
11          query = """
12              INSERT INTO USER (Uname, Email, Password, Bdate)
13              VALUES (%s, %s, %s, %s)
14              """
15          values = (username, email, hashed_password, birth_date)
16          try:
17              # 데이터베이스에 사용자 추가
18              self.db.execute_query(query, values)
19              print("회원 가입이 성공적으로 완료되었습니다.")
20              return True
21          except Exception as e:
22              print(f"회원 가입 실패: {e}")
23              return False
```

- 회원가입에 필요한 DB 접근 컨트롤러이며, 암호는 해시값으로 관리한다.

Controllers/login_controller.py

```
1 import bcrypt
2 from DB.database import Database
3
4 class LoginController:
5     def __init__(self):
6         self.db = Database()
7     def authenticate(self, email, password):
8         # db에서 사용자 정보 확인
9         query = "SELECT * FROM USER WHERE email = %s"
10        result = self.db.execute_query(query, (email, ))
11        if result:
12            hashed_password = result[0]['Password']
13
14            if bcrypt.checkpw(password.encode('utf-8'), hashed_password.encode('utf-8')):
15                return result[0]
16        return None
```

- 로그인에 필요한 DB 접근 컨트롤러이며, 역시 암호는 해시값으로 관리한다.
- 암호를 비교할 때는 checkpw 함수를 이용해서 decoding한 후 비교할 수 있다.

DB/database.py

```
3 class Database:
4     _instance = None
5
6     def __new__(cls, *args, **kargs):
7         # 이미 생성된 인스턴스가 있으면 해당 인스턴스를 반환
8         if cls._instance is None:
9             cls._instance = super().__new__(cls)
10        return cls._instance
11
12    def __init__(self, host='localhost', user='root', password='602003', database='Music', port=3306):
13
14        if hasattr(self, "connection"):
15            return
16
17        #Mysql 연결 설정
18        try:
19            self.connection = pymysql.connect(
20                host = host,
21                user = user,
22                password = password,
23                database = database,
24                port = port,
25                charset = "utf8mb4",
26                cursorclass= pymysql.cursors.DictCursor
27
28            )
29            print("데이터베이스 연결 성공")
30        except pymysql.MySQLError as e:
31            print("데이터베이스 연결 실패: {e}")
32            self.connection = None
```

- 프로그램을 DB와 연결 시키는 부분으로, 인스턴스를 하나만 생성하도록 관리한다.
- 이미 생성된 인스턴스가 있으면 해당 인스턴스를 반환하며, 없으면 생성한다.

```
34     def execute_query(self, query, params=()):  
35         with self.connection.cursor() as cursor:  
36             cursor.execute(query, params)  
37             self.connection.commit()  
38         return cursor.fetchall() #query result
```

- query와 파라미터를 인자로 받아서 일관된 방식으로 query를 처리할 수 있다.

```
40     def close(self):  
41         if self.connection:  
42             self.connection.close()  
43
```

- 연결을 닫는 함수이다.

```
44     def get_last_insert_id(self):  
45         # 최근에 삽입된 auto_increment 값 반환  
46         with self.connection.cursor() as cursor:  
47             cursor.execute("SELECT LAST_INSERT_ID();")  
48             result = cursor.fetchone()  
49         return result['LAST_INSERT_ID()'] # 삽입이 없으면 0을 반환
```

- 가장 최근에 DB에 삽입한 객체의 ID를 반환하는 함수이다.

3. 수행 스크린샷

1) 로그인

```
==== 메인 메뉴 ====  
0. 프로그램 종료  
1. 로그인  
2. 회원가입  
원하는 작업을 선택하세요 :
```

```
원하는 작업을 선택하세요 : 1  
데이터베이스 연결 성공  
==== 로그인 화면 ====  
이메일을 입력하세요 : sonje515@hanyang.ac.kr  
비밀번호를 입력하세요 : 030206  
로그인 성공 !
```

```
메인 화면  
0. 로그아웃  
1. 곡보기  
2. 곡찾기  
3. 플레이리스트 관리하기  
4. 좋아요 표시한 곡보기  
원하는 작업을 선택하세요 :
```

로그인 후 메인 화면

1-1) 곡 보기

```
원하는 작업을 선택하세요 : 1  
== 곡 목록 ==  
ID 제목 아티스트  
-----  
1 HOME SWEET HOME G-DRAGON  
4 Supernova aespa  
5 Armageddon aespa  
-----: 곡 목록
```

1-1-1) 곡 재생하기

```
0. 이전 메뉴로  
1. 재생하기  
2. 곡 정보 보기  
3. 좋아요 표시 / 취소  
4. 플레이리스트에 곡 추가하기  
원하는 작업을 선택하세요 : 1
```

```
원하는 작업을 선택하세요 : 1  
재생할 곡의 ID를 입력하세요 : 4  
재생 횟수가 업데이트되었습니다.: 사용자 7, 곡 4
```

♪ 'Supernova' by aespa 재생 중 ...

1-1-2) 곡 정보보기

```
원하는 작업을 선택하세요 : 2  
정보를 확인할 곡의 ID를 입력하세요 : 4  
== 곡 정보 ==  
ID: 4  
제목 : Supernova  
아티스트 : aespa  
발매일 : 2024-05-27  
장르 : 댄스 팝  
앨범명 : Armageddon  
=====
```

1-1-3) 좋아요 표시 / 취소

```
원하는 작업을 선택하세요 : 3  
좋아요를 변경할 곡의 ID를 입력하세요 : 5  
이미 좋아요를 누른 곡입니다. 좋아요를 취소하시겠습니까? (y/n): y  
좋아요가 취소되었습니다.
```

: 좋아요 취소

```
원하는 작업을 선택하세요 : 3  
좋아요를 변경할 곡의 ID를 입력하세요 : 5  
이 곡을 좋아요 하시겠습니까? (y/n): y  
좋아요가 추가되었습니다.
```

: 좋아요 표시

1-1-4) 플레이리스트에 곡 추가하기

원하는 작업을 선택하세요 : 4
플레이리스트에 추가할 곡의 ID를 입력하세요 : 1

--- 내 플레이리스트 목록 ---
ID: 2, 이름: 나의 플레이리스트, 곡 수: 1
곡을 추가할 플레이리스트 ID를 입력하세요: 2
이 곡은 이미 플레이리스트에 추가되어 있습니다.

]: 이미 추가된 경우

원하는 작업을 선택하세요 : 4
플레이리스트에 추가할 곡의 ID를 입력하세요 : 1

== 내 플레이리스트 목록 ==
ID: 2, 이름: 나의 플레이리스트, 곡 수: 1
곡을 추가할 플레이리스트 ID를 입력하세요: 2
이곡은 이미 플레이리스트에 추가되어 있습니다.

: 새롭게 추가한 경우

1-2) 곡 찾기

원하는 작업을 선택하세요 : 2

==== 곡 검색 ====
검색할 곡 제목이나 아티스트 이름을 입력하세요 : HOME

==== 검색 결과 ====
ID 제목 아티스트

1 HOME SWEET HOME G-DRAGON

: 제목으로 검색

원하는 작업을 선택하세요 : 2

==== 곡 검색 =====
검색 할 곡 제목 이나 아티스트 이름을 입력하세요 : G-DRAGON

==== 감 삭 결 과 ====

1 HOME SWEET HOME G-DRAGON

: 아티스트로 검색

1-3) 플레이리스트 관리하기

원하는 작업을 선택하세요 : 3

```
== 플레이리스트 ==
ID: 2, 이름: 나의 플레이리스트, 곡 수: 2
0. 이전 메뉴로
1. 새 플레이리스트 생성
2. 플레이리스트 삭제
3. 플레이리스트 관리
원하는 작업을 선택하세요: [ ]
```

▶ : 플레이리스트 관리 메뉴

1-3-1) 새 플레이리스트 생성

원하는 작업을 선택하세요 : 1
새 플레이리스트 이름을 입력하세요 : 크리스마스 캐롤 모음
새로운 플레이리스트가 생성되었습니다. ID: 4

```
==== 플레이리스트 ====  
ID: 2, 이름: 나의 플레이리스트, 곡 수: 2  
ID: 4, 이름: 크리스 캐롤 모음, 곡 수: 0
```

1-3-2) 플레이리스트 삭제

```
원하는 작업을 선택하세요 : 2

==== 플레이리스트 목록 ====
ID: 2, 이름: 나의 플레이리스트, 곡 수: 2
ID: 4, 이름: 크리스 캐롤 모음, 곡 수: 0
삭제할 플레이리스트 ID를 입력하세요 : 4
'크리스 캐롤 모음' 플레이리스트를 삭제하시겠습니까? (Y/N): y
플레이리스트 ID 4가 성공적으로 삭제되었습니다.

==== 플레이리스트 ====
ID: 2, 이름: 나의 플레이리스트, 곡 수: 2
```

1-3-3) 플레이리스트 관리

```
원하는 작업을 선택하세요 : 3

==== 플레이리스트 ====
ID: 2, 이름: 나의 플레이리스트, 곡 수: 2
수정할 플레이리스트 ID를 입력하세요 : 2

==== 플레이리스트 '나의 플레이리스트'의 곡 목록 ====
ID: 1, 제목: HOME SWEET HOME, 아티스트: G-DRAGON
ID: 4, 제목: Supernova, 아티스트: aespa

0. 이전 메뉴로
1. 곡 삭제하기
```

1-3-3-1) 플레이리스트에서 곡 삭제하기

```
원하는 작업을 선택하세요 : 1

==== 플레이리스트 내 곡 목록 ====
ID: 1, 제목: HOME SWEET HOME, 아티스트: G-DRAGON
ID: 4, 제목: Supernova, 아티스트: aespa
삭제할 곡 ID를 입력하세요 : 4
곡이 성공적으로 삭제되었습니다.
곡 'Supernova'가 플레이리스트에서 삭제되었습니다.

==== 플레이리스트 ====
ID: 2, 이름: 나의 플레이리스트, 곡 수: 1
```

1-4) 좋아요 표시한 곡 보기

```
원하는 작업을 선택하세요 : 4

==== 좋아요 누른 곡 목록 ====
ID    제목          아티스트
-----+
1    HOME SWEET HOME      G-DRAGON
5    Armageddon            aespa
-----+
0. 이전 메뉴로
1. 곡 재생하기
2. 곡 정보 보기
3. 좋아요 표시/수정하기
```

2) 회원가입

```
원하는 작업을 선택하세요 : 2
데이터베이스 연결 성공
==== 회원가입 화면 ===-
```

```
이름을 입력하세요 : 이효정  
이메일을 입력하세요 : hyojung@hanyang.ac.kr  
비밀번호를 입력하세요 : 040102  
생년월일을 입력하세요 (ex, yyyy-mm-dd) : 2004-01-02  
회원가입이 성공적으로 완료되었습니다.  
회원가입 성공!
```

3) 관리자 로그인

```
==== 로그인 화면 ====  
이메일을 입력하세요 : admin@hanyang.ac.kr  
비밀번호를 입력하세요 : password001  
로그인 성공!
```

```
==== 관리자 메인 화면 ====  
0. 로그아웃  
1. 곡 관리하기  
2. 아티스트 관리하기  
3. 앨범 관리하기  
원하는 작업을 선택하세요 :
```

: 관리자 메인 화면

3-1) 곡 관리하기

```
원하는 작업을 선택하세요 : 1  
  
==== 곡 관리하기 ====  
0. 이전 메뉴로  
1. 추가하기  
2. 수정 및 삭제하기  
원하는 작업을 선택하세요 :
```

3-1-1) 추가하기

```
원하는 작업을 선택하세요 : 1  
  
==== 곡 추가하기 ====  
==== 1. 아티스트 정보 입력 및 확인 ====  
아티스트 이름을 입력하세요 : 재쓰비  
새로운 아티스트를 추가합니다.  
성별을 입력하세요 (남성/여성/혼성) : 혼성  
아티스트 유형을 입력하세요 (솔로/듀오/그룹) : 그룹  
아티스트 '재쓰비'가 추가되었습니다. ID: 8  
==== 2. 앨범 정보 입력 및 확인 ====  
앨범명을 입력하세요 : 너와의 모든 지금  
발매일을 입력하세요 (yyyy-dd-mm) : 2024-11-11  
새로운 앨범을 추가합니다.  
앨범 '너와의 모든 지금'가 추가되었습니다. ID: 9  
==== 3. 곡 정보 입력 및 확인 ====  
곡 제목을 입력하세요 : 너와의 모든 지금  
장르를 입력하세요 : 댄스팝  
가사를 입력하세요 :  
곡 '너와의 모든 지금'가 성공적으로 추가되었습니다. ID: 6
```

3-1-2) 수정 및 삭제하기

원하는 작업을 선택하세요 : 2

==== 곡 수정 및 삭제 ====

수정 또는 삭제할 곡의 제목을 입력하세요 (또는 '목록' 입력하여 전체 곡 보기) : 너와의 모든 지금

==== 검색 결과 ====
ID 제목 아티스트
6 너와의 모든 지금 재쓰비

수정 또는 삭제할 곡의 ID를 입력하세요 : 6

0. 취소하기
1. 곡 수정하기
2. 곡 삭제하기

원하는 작업을 선택하세요 : ■

3-1-2-1) 곡 수정하기

원하는 작업을 선택하세요 : 1

==== 곡 수정하기 ====
새 제목 (현재 : 너와의 모든 지금) : 너와의 모든 지금
새 발매일 (현재 : 2024-11-11) :
새 장르 (현재 : 댄스 팝) :
새 가사 (현재 : N/A) :
곡 ID 6의 정보가 성공적으로 수정되었습니다.
곡 정보가 성공적으로 수정되었습니다.

3-1-2-2) 곡 삭제하기

원하는 작업을 선택하세요 : 2

==== 곡 수정 및 삭제 ====
수정 또는 삭제할 곡의 제목을 입력하세요 (또는 '목록' 입력하여 전체 곡 보기) : 목록

==== 곡 목록 ====
ID 제목 아티스트
1 HOME SWEET HOME G-DRAGON
4 Supernova aespa
5 Armageddon aespa
6 모든 지금 재쓰비

수정 또는 삭제할 곡의 ID를 입력하세요 : 6

0. 취소하기
1. 곡 수정하기
2. 곡 삭제하기

원하는 작업을 선택하세요 : 2

==== 곡 삭제하기 ====
정말로 '모든 지금' 곡을 삭제하시겠습니까? (Y/N) : y
곡 ID 6가 성공적으로 삭제되었습니다.
곡이 성공적으로 삭제되었습니다.

3-2) 아티스트 관리하기

원하는 작업을 선택하세요 : 2

==== 아티스트 관리 ====
0. 이전 메뉴로
1. 아티스트 추가하기
2. 아티스트 수정 및 삭제하기

원하는 작업을 선택하세요 : ■

3-2-1) 아티스트 추가하기

```
원하는 작업을 선택하세요 : 1

==== 아티스트 추가 ====
아티스트 이름을 입력하세요 : DAY6
새로운 아티스트를 추가합니다.
성별을 입력하세요 (남성/여성/혼성) : 남성
아티스트 유형을 입력하세요 (솔로/듀오/그룹) : 그룹
아티스트 'DAY6'가 추가되었습니다. ID: 9
```

3-2-2) 아티스트 수정 및 삭제하기

```
원하는 작업을 선택하세요 : 2

==== 아티스트 수정 및 삭제 ====
수정 또는 삭제할 아티스트의 이름을 입력하세요 (또는 '목록'을 입력하여 전체 아티스트 보기) : day6

==== 검색 결과 ====
ID 이름           성별       유형
-----  
9   DAY6          M          Group
-----  
수정 또는 삭제할 아티스트의 ID를 입력하세요 : ■
```

3-2-2-1) 아티스트 수정하기

```
원하는 작업을 선택하세요 : 1

==== 아티스트 수정하기 ====
새 이름 (현재 : DAY6) : 데 이식스
새 성별 (현재 : M, 남성/여성/혼성) :
새 유형 (현재 : Group, 솔로/듀오/그룹) :
아티스트 정보가 성공적으로 수정되었습니다.
```

3-2-2-2) 아티스트 삭제하기

```
==== 아티스트 목록 ====
ID 이름           성별       유형
-----  
1   G-DRAGON        M          Solo
5   로제 (ROSE)      F          Solo
7   aespa            F          Group
8   재쓰비          Other      Group
9   데 이식스       M          Group
-----  
수정 또는 삭제할 아티스트의 ID를 입력하세요 : 9

0. 취소하기
1. 아티스트 수정하기
2. 아티스트 삭제하기
원하는 작업을 선택하세요 : 2

==== 아티스트 삭제하기 ====
정말로 '데 이식스' 아티스트를 삭제하시겠습니까? (Y/N) : y
아티스트 ID 9가 성공적으로 삭제되었습니다.
아티스트가 성공적으로 삭제되었습니다.
```

3-3) 앨범 관리하기

```
원하는 작업을 선택하세요 : 3

==== 앨범 관리 ====
0. 이전 메뉴로
1. 앨범 추가하기
2. 앨범 수정 및 삭제하기
원하는 작업을 선택하세요 : ■
```

3-3-1) 앨범 추가하기

```
원하는 작업을 선택하세요 : 1

==== 앨범 추가 ====
앨범 제목을 입력하세요 : Fourever
앨범 발매일자를 입력하세요 (yyyy-mm-dd) : 2024-05-28
앨범의 아티스트 이름을 입력하세요 : DAY6
새로운 아티스트를 추가합니다.
성별을 입력하세요 (남성 / 여성 / 혼성) : 남성
아티스트 유형을 입력하세요 (솔로 / 듀오 / 그룹) : 그룹
아티스트 'DAY6'가 추가되었습니다. ID: 10
새로운 앨범을 추가합니다.
앨범 'Fourever'가 추가되었습니다. ID: 10
```

3-3-2) 앨범 수정 및 삭제하기

```
원하는 작업을 선택하세요 : 2

==== 앨범 수정 및 삭제 ====
수정 또는 삭제할 앨범의 제목을 입력하세요 (또는 '목록'을 입력하여 전체 앨범 보기) : fourever

==== 검색 결과 ====
ID 제목           발매일           아티스트
-----  
10 Fourever       <1510
```

수정 또는 삭제할 앨범의 ID를 입력하세요 : ■

3-3-2-1) 앨범 수정하기

```
수정 또는 삭제할 앨범의 ID를 입력하세요 : 10

0. 취소하기
1. 앨범 수정하기
2. 앨범 삭제하기
원하는 작업을 선택하세요 : 1

==== 앨범 수정하기 ====
새제목 (현재 : Fourever) :
새발매일 (현재 : 2024-05-28) : 2024-03-18
앨범 정보가 성공적으로 수정되었습니다.
```

3-3-2-2) 앨범 삭제하기

```
2. 결제 삭제하기
원하는 작업을 선택하세요 : 2

==== 앨범 삭제하기 ====
정말로 'Fourever' 앨범을 삭제하시겠습니까? (Y/N) : y
앨범이 성공적으로 삭제되었습니다.
```

4. SQL문 명세

1) SELECT

```
query = "SELECT * FROM USER WHERE email = %s" [login_controller.py]
```

- 로그인 시, 사용자의 정보가 저장되어 있는지 확인한다.
- 확인하는 조건은 email이다. 따라서 email은 UNIQUE constraint를 갖는다.

```
query = """
    SELECT
        SONG.Song_id,
        SONG.Title,
        SONG.ReleaseDate,
        SONG.Genre,
        Album.AbTitle,
        ARTIST.AtName
    FROM
        SONG
    JOIN
        ALBUM
    ON
        SONG.Album = ALBUM.Album_id
    JOIN
        ARTIST
    ON ALBUM.Artist = ARTIST.Artist_id
"""
query = """
    SELECT
        SONG.Song_id,
        SONG.Title,
        SONG.ReleaseDate,
        SONG.Genre,
        Album.AbTitle,
        ARTIST.AtName
    FROM
        LIKES
    JOIN
        SONG
    ON
        LIKES.LSong = SONG.Song_id
    JOIN
        ALBUM
    ON
        SONG.Album = ALBUM.Album_id
    JOIN
        ARTIST
    ON
        ALBUM.Artist = ARTIST.Artist_id
    WHERE
        LIKES.Who = %s
"""
[songs_controller.py]
```

- 둘 다 곡을 가져오는 query문이다.
- 왼쪽 query문은 DB에 등록된 모든 곡을 가져오는 query문으로, 곡의 id, 제목, 발매일, 장르와 앨범명, 아티스트 이름을 가져온다. 단, SONG table에 앨범명과 아티스트 이름이 저장되는 것이 아니라 해당 곡이 속한 앨범의 id와 아티스트의 id가 foreign key로 저장되는 것에 불과하기 때문에, JOIN을 통해서 가져올 수 있다.
- 오른쪽 query문은 좋아요를 표시한 곡들만 가져오는 query문으로, 가져오는 항목은 동일하다. 단, 좋아요를 표시한 곡을 가져와야 하므로 SONG table에서 가져오는 것이 아니라 LIKES 테이블에 속한 곡을 가져오며, 필요한 정보를 JOIN을 통해 가져올 수 있다. 사용자별로 LIKES 테이블의 인스턴스가 다르기 때문에 WHERE절도 필요하다.

```
check_play_query = "SELECT Ppc FROM PLAY WHERE Who = %s AND PSong = %s"
```

- 곡을 재생할 때 사용되는 query문으로, 개인 재생 기록이 있는지를 확인하는 구문이다.

```

search_query = """
    SELECT
        SONG.Song_id,
        SONG.Title,
        SONG.ReleaseDate,
        SONG.Genre,
        ARTIST.AtName,
        ALBUM.AbTitle
    FROM
        SONG
    JOIN
        ALBUM
    ON
        SONG.Album = ALBUM.Album_id
    JOIN
        ARTIST
    ON
        ALBUM.Artist = ARTIST.Artist_id
    WHERE
        SONG.Title Like %s OR ARTIST.AtName Like %s
"""

```

- 곡을 찾는 query문으로, 곡 제목이나 아티스트를 입력 받아 검색한다.
- 정확히 일치하는 제목이나 아티스트로 찾는 게 아니라 그 단어를 포함하고 있으면 다 검색된다.

```
query = "SELECT 1 FROM LIKES WHERE Who = %s AND LSong = %s"
```

- 좋아요 상태를 확인하는 query문으로, 조건을 만족하는 행이 있는지 확인한다. 있으면 1 반환.

```

query_check = """
    SELECT Playlist_id
    FROM PLAYLIST
    WHERE Owner = %s AND PlTitle = %s
"""

```

```

query_check= """
    SELECT Playlist_id
    FROM PLAYLIST
    WHERE Owner = %s AND PlTitle = '나의 플레이리스트'
"""

```

- 플레이리스트를 검색하는 query문으로, default인지 그 외의 플레이리스트인지만 다르다.

```

query_check = """
    SELECT * FROM INCLUDE
    WHERE WhatPl = %s AND WhichSong = %s
"""

```

- 플레이리스트에 곡이 포함되어 있는지 확인할 때 사용되는 query문이다.

```

query = """
    SELECT Playlist_id, PlTitle, PnumberOfSongs
    FROM PLAYLIST
    WHERE Owner = %s
"""

```

- 사용자의 플레이리스트를 검색하는 query문이다.

```

query = """
SELECT S.Song_id, S.Title, A.AtName AS Artist, Alb.AbTitle AS Album
FROM INCLUDE I
JOIN SONG S ON I.WhichSong = S.Song_id
JOIN ALBUM Alb ON S.Album = Alb.Album_id
JOIN ARTIST A ON Alb.Artist = A.Artist_id
WHERE I.WhatPl = %s
"""

```

- 플레이리스트에 속한 곡들을 검색하며, 곡의 ID, 제목, 아티스트 이름, 앨범 제목을 반환한다.
- 플레이리스트와 곡을 연결하는 INCLUDE 테이블에서 가져오되, 곡 제목, 앨범명, 아티스트 이름은 JOIN을 통해서 가져올 수 있다.

```

query_check = "SELECT Artist_id, AtName FROM Artist WHERE AtName LIKE %s"

```

- 아티스트 이름을 입력 받고, 입력값을 포함하는 아티스트를 모두 찾아서 id, 이름을 반환한다.

2) INSERT

컬럼명	# Data Type	Not Null	Auto Increment	Key	디폴트	Extra
123 Song_id	1 int	[v]	[v]	PRI		auto_increment
A-Z Title	2 varchar(100)	[v]	[]			
ReleaseDate	3 date	[]	[]			
A-Z Genre	4 varchar(50)	[]	[]			
123 LikeCount	5 int	[]	[]		0	
123 PlayCount	6 int	[]	[]		0	
123 Album	7 int	[]	[]	MUL		
A-Z Lyrics	8 text	[]	[]			

- 모든 테이블에는 id값이 존재하는데, 위의 사진에서 확인할 수 있는 것처럼 auto_increment 조건을 추가해서 실제로 instance를 추가할 때 id값을 따로 지정/조정하지 않는다.

```

query = """
INSERT INTO USER (Uname, Email, Password, Bdate)
VALUES (%s, %s, %s, %s)
"""

```

- 회원가입 시, 새로운 유저를 테이블에 등록하는 구문이다.
- 이름, 이메일, 패스워드, 생년월일을 입력 받아 추가한다.

USER Enter a SQL expression to filter results (use Ctrl+Space)						
그리드	User_id	Uname	Email	Password	Bdate	Role
1	1	Admin	admin@hanyang.ac.kr	\$2b\$12\$UD9mU1zfSJ2cuJvbN27zgOwge4pQBITXlq3dXUjydFlUVUWoLZdgG	1990-01-01	Admin
2	7	손주은	sonje515@hanyang.ac.kr	\$2b\$12\$fjDLYa6f0xaRfdNEIT3fm.R/I.BWDRK186VtUD4UkbhE4y5j19qYu	2003-02-06	User
3	8	김소정	cindy@hanyang.ac.kr	\$2b\$12\$Y8irg1qykJwSagmVd6/uMfvpxXOK.ogQetahn6eJA8xTxG7/Bee	2003-03-25	User
4	9	이효정	hyojung@hanyang.ac.kr	\$2b\$12\$SZ9gA5Y18yecUZMz8E/aJu68zsZ9ZM.Tr0mzgsTKxLBnOulZ6XXWu	2004-01-02	User

- Role attribute은 따로 입력 받지 않으며, 입력하지 않는 경우에는 default로 user가 들어간다.

```

insert_play_query = "INSERT INTO PLAY (Who, PSong, Ppc, Lpd) VALUES (%s, %s, 1, NOW())"

```

- 누가 어떤 곡을 재생했는지를 기록하는 query 문이다. 재생 기록이 없다가 추가하는 부분이므로, 개인 재생 횟수는 1이되고, last play date은 NOW()가 된다.

```
query = "INSERT INTO LIKES (Who, LSong) VALUES (%s, %s)"
```

- 누가 어떤 곡에 좋아요 표시를 했는지 기록하는 구문이다.

```
query_add = """
INSERT INTO PLAYLIST (PlTitle, Content, CreateDate, Owner, PnumberOfSongs)
VALUES (%s, %s, NOW(), %s, 0)
....
```

```
query_add = """
INSERT INTO PLAYLIST (PlTitle, Content, CreateDate, Owner, PnumberOfSongs)
VALUES ('나의 플레이리스트', 'default playlist', NOW(), %s, 0)
....
```

- 플레이리스트를 추가하는 query문이다. 새롭게 추가되므로 생성일은 Now()이고, PnumberOfSongs는 0으로 세팅된다. default인지, 그 외에 추가되는 플레이리스트인지만 다르다.

```
query_add = """
INSERT INTO INCLUDE (WhatPl, WhichSong)
VALUES (%s, %s)
....
```

- 플레이리스트에 곡을 추가할 때, 어떤 플레이리스트에 어떤 곡이 포함되는지를 기록한다.

```
query_add = "INSERT INTO Artist (AtName, Sex, Type) VALUES (%s, %s ,%s)"
```

- 아티스트를 새롭게 추가한다.

3) UPDATE

```
update_song_query = "UPDATE SONG SET PlayCount = PlayCount + 1 WHERE Song_id = %s"
update_play_query = "UPDATE PLAY SET Ppc = Ppc + 1, Lpd = NOW() WHERE Who = %s AND PSong = %s"
```

- 전체 재생횟수, 개인 재생횟수를 업데이트하는 query문이다.

```
query = """
    UPDATE SONG SET LikeCount = LikeCount + 1 |
    WHERE Song_id = %s
....
```

- 곡의 전체 좋아요 개수를 기록한다. 인기곡을 선별할 때 좋아요 개수를 이용할 수 있다.

```
query = "UPDATE SONG SET LikeCount = GREATEST(0, LikeCount - 1) WHERE Song_id = %s"
```

- 사용자가 좋아요를 취소하면 전체 좋아요 개수에서도 1이 감소해야 일관성을 유지할 수 있다.

```
query_update = """
UPDATE PLAYLIST
SET PnumberOfSongs = PnumberOfSongs + 1
WHERE Playlist_id = %s
.....
```

```
query_update = """
UPDATE PLAYLIST
SET PnumberOfSongs = PnumberOfSongs - 1
WHERE Playlist_id = %s
.....
```

- 플레이리스트에 곡을 추가/삭제함에 따라 곡 수를 증감시킨다.

4) DELETE

```
query = "DELETE FROM LIKES WHERE Who = %s AND LSong = %s"
```

- 사용자가 좋아요 표시를 취소하기를 원할 때 수행되는 query문이다.

```
query_delete_includes = "DELETE FROM INCLUDE WHERE WhatPl = %s"
```

```
query_delete_playlist = "DELETE FROM PLAYLIST WHERE Playlist_id = %s"
```

- 플레이리스트를 삭제할 때, 해당 플레이리스트에 속하는 곡들을 먼저 삭제한 후, 플레이리스트 자체를 삭제한다.

```
query_check_songs = "SELECT COUNT(*) AS SongCount FROM SONG WHERE Album IN (SELECT Album_id FROM Album WHERE Artist = %s)"
query_check_albums = "SELECT COUNT(*) AS AlbumCount FROM Album WHERE Artist = %s"
```

- 첫번째 query는 특정 아티스트가 가지는 모든 곡의 개수를 반환하는 query다. 서브쿼리가 먼저 실행되어, 특정 아티스트의 모든 앨범을 반환하며 그 앨범들에 속하는 곡의 개수를 계산한다.
- 이 쿼리문은 아티스트를 삭제할 때 사용되는 query문이기 때문에 특정 앨범에 속하는 곡의 개수를 계산하는 게 아니라 아티스트가 가지는 모든 곡을 탐색한다.
- 두번째 query는 특정 아티스트의 총 앨범 개수를 반환하는 query다.

이외에도 사용된 query문들이 더 있지만 구조는 동일한데 변수만 다른 구문들이 많아서 분량상 생략하였다.