

2020 MusicApp Assignment

2017029916

양동해

1. 프로그램의 전반적인 개요

데이터베이스 내에는 음원 관리자 및 스트리밍 구독자(사용자), 음원, 앨범, 아티스트, 플레이리스트 등이 저장되어 있다. 첫 화면에서 계정을 생성할 수 있는데, 이때 관리자 코드를 입력하면 관리자로 계정을 생성할 수 있다.

음원 관리자는 유저로도 동작하지만 관리자의 권한을 따로 부여 받아 음원을 등록 및 삭제할 수 있다. 음원은 반드시 하나의 앨범에 속해야 하며, 음원과 앨범을 만든 아티스트는 반드시 존재해야 하므로, 관리자는 아티스트와 앨범도 등록 및 삭제할 수 있다. 관리자는 사용자 또한 관리할 수 있어, 사용자의 신상정보를 조회하거나 삭제하는 것이 가능하다.

사용자는 음원을 찾아보거나 감상할 수 있으며, 자신만의 플레이리스트를 만들고 여기에 음원을 넣어 관리할 수 있다. 플레이리스트에 있는 곡들은 전체 재생이 가능하며, 여기에 음원을 추가 및 삭제하거나 플레이리스트 자체를 삭제하는 것이 가능하다. 이외에도 아티스트 검색, 앨범 검색 등을 제공하며, 사용자들이 재생한 음원의 조회수를 통해 음원의 통계를 제공한다.

이 프로그램 내에서 정해진 constraints로는 음원은 반드시 하나의 앨범에 속해야 하는 것이다. 또한 앨범에는 음원이 존재하지 않는 경우를 허용하며, 아티스트도 음원 및 앨범 발매를 하지 않았을 경우를 허용한다. 그리고 음원이나 앨범이 만들어지기 위해선 아티스트가 필수적으로 존재해야 한다.

2. HW3에서 변경된 사항

- USER

기존에 USER 테이블에, UserIndex를 User_id로 표현했고, attribute로 User_pw도 추가했다. 이는 사용자가 계정을 생성할 수 있게, id와 pw를 명시하기 위함이다. USER의 isAdmin은 boolean 값이 아닌 integer 값으로, 0과 1을 통해 관리자와 유저를 구분했다.

- PLAYLIST

기존 PLAYLIST에 있었던 NumOfMusic을 삭제했는데, 이는 애플리케이션 내에서 ArrayList의 size로 알아낼 수 있기 때문에, 불필요한 정보라고 판단하여 삭제했다.

- 위의 두 테이블을 포함한 DB내의 모든 테이블

이 외에 기능적으로 바뀐 부분은 없으나 전체 테이블을 대상으로, attribute 이름들을 조금씩 수정했다. 이는 '3. DB 제작'에서 확인할 수 있다.

3. DB 제작

위에서 언급했던 변경된 사항 외에는 DB 자체의 변화가 없으며 dump file도 존재하므로, 테이블을 create 했을 때의 DDL로 설명을 대신한다.

- USER

```
CREATE TABLE `USER` (
  `User_id` int(11) NOT NULL,
  `User_pw` varchar(15) NOT NULL,
  `User_name` varchar(30) NOT NULL,
  `Phone` varchar(15) DEFAULT NULL,
  `Ssn` varchar(15) NOT NULL,
  `Address` varchar(30) DEFAULT NULL,
  `Is_admin` int(11) NOT NULL,
  PRIMARY KEY (`User_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- MUSIC

```
CREATE TABLE `MUSIC` (
  `Music_id` int(11) NOT NULL,
  `Music_title` varchar(30) NOT NULL,
  `Lyricist` varchar(30) DEFAULT NULL,
  `Composer` varchar(30) DEFAULT NULL,
  `Arranger` varchar(30) DEFAULT NULL,
  `Hits_num` int(11) DEFAULT 0,
  `Album_idx` int(11) NOT NULL,
  PRIMARY KEY (`Music_id`),
  KEY `Album_idx` (`Album_idx`),
  CONSTRAINT `music_ibfk_1` FOREIGN KEY (`Album_idx`) REFERENCES `ALBUM` (`Album_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- ALBUM

```
CREATE TABLE `ALBUM` (
  `Album_id` int(11) NOT NULL,
  `Album_title` varchar(30) NOT NULL,
  `Album_type` varchar(15) DEFAULT NULL,
  `Agency` varchar(15) DEFAULT NULL,
  `Release_company` varchar(15) DEFAULT NULL,
  `Release_date` date DEFAULT NULL,
  PRIMARY KEY (`Album_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- ARTIST

```
CREATE TABLE `ARTIST` (
  `Artist_id` int(11) NOT NULL,
  `Artist_name` varchar(30) NOT NULL,
  PRIMARY KEY (`Artist_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- PLAYLIST

```
CREATE TABLE `PLAYLIST` (
  `Playlist_id` int(11) NOT NULL,
  `Playlist_title` varchar(30) NOT NULL,
  `Creation_date` date DEFAULT NULL,
  `User_idx` int(11) NOT NULL,
  PRIMARY KEY (`Playlist_id`),
  KEY `User_idx` (`User_idx`),
  CONSTRAINT `playlist_ibfk_1` FOREIGN KEY (`User_idx`) REFERENCES `User` (`User_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- MUSIC_GENRE

```
CREATE TABLE `MUSIC_GENRE` (
  `MIndex` int(11) NOT NULL,
  `MGenre` varchar(15) NOT NULL,
  PRIMARY KEY (`MIndex`,`MGenre`),
  CONSTRAINT `music_genre_ibfk_1` FOREIGN KEY (`MIndex`) REFERENCES `MUSIC` (`Music_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- ALBUM_GENRE

```
CREATE TABLE `ALBUM_GENRE` (
  `AIndex` int(11) NOT NULL,
  `AGenre` varchar(15) NOT NULL,
  PRIMARY KEY (`AIndex`,`AGenre`),
  CONSTRAINT `album_genre_ibfk_1` FOREIGN KEY (`AIndex`) REFERENCES `ALBUM` (`Album_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- MUSIC_MAKE

```
CREATE TABLE `MUSIC_MAKE` (
  `MM_Artist_idx` int(11) NOT NULL,
  `MM_Music_idx` int(11) NOT NULL,
  PRIMARY KEY (`MM_Artist_idx`,`MM_Music_idx`),
  KEY `MM_Music_idx`(`MM_Music_idx`),
  CONSTRAINT `music_make_ibfk_1` FOREIGN KEY (`MM_Music_idx`) REFERENCES `MUSIC` (`Music_id`),
  CONSTRAINT `music_make_ibfk_2` FOREIGN KEY (`MM_Artist_idx`) REFERENCES `ARTIST` (`Artist_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- ALBUM_MAKE

```
CREATE TABLE `ALBUM_MAKE` (
  `AM_Artist_idx` int(11) NOT NULL,
  `AM_Album_idx` int(11) NOT NULL,
  PRIMARY KEY (`AM_Artist_idx`,`AM_Album_idx`),
  KEY `AM_Album_idx`(`AM_Album_idx`),
  CONSTRAINT `album_make_ibfk_1` FOREIGN KEY (`AM_Artist_idx`) REFERENCES `ARTIST` (`Artist_id`),
  CONSTRAINT `album_make_ibfk_2` FOREIGN KEY (`AM_Album_idx`) REFERENCES `ALBUM` (`Album_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- MUSIC_IN_LIST

```
CREATE TABLE `MUSIC_IN_LIST` (
  `MIL_Playlist_idx` int(11) NOT NULL,
  `MIL_Music_idx` int(11) NOT NULL,
  PRIMARY KEY (`MIL_Playlist_idx`,`MIL_Music_idx`),
  KEY `MIL_Music_idx`(`MIL_Music_idx`),
  CONSTRAINT `music_in_list_ibfk_1` FOREIGN KEY (`MIL_Playlist_idx`) REFERENCES `PLAYLIST` (`Playlist_id`),
  CONSTRAINT `music_in_list_ibfk_2` FOREIGN KEY (`MIL_Music_idx`) REFERENCES `Music` (`Music_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

4. 프로그램 코드에 대한 자세한 설명

- User.java, Music.java, Album.java, Artist.java, Playlist.java

각각은 데이터베이스 테이블에 해당하는 클래스로 USER, MUSIC, ALBUM, ARTIST, PLAYLIST를 의미한다. DB의 테이블로는 MUSIC_MAKE, MUSIC_GENRE 등 더 많은 테이블이 존재하지만, 이들은 다른 클래스의 변수들로 표현이 가능하므로 따로 클래스화 하지는 않았다. 각각의 파일마다 특별한 기능은 없고 단지 필요한 변수와 생성자만 존재하므로, 코드사진으로 설명을 대신한다.

```

3 public class User {
4     int userID;
5     String userPW; // 추가함!!
6     String userName;
7     String phone;
8     String ssn;
9     String address;
10    int isAdmin;
11
12    ArrayList<Playlist> playlists = new ArrayList<Playlist>();
13
14    User(){}
15● User(int userID, String userPW, String userName, String phone, String ssn, String address, int isAdmin){
16        this.userID = userID;
17        this.userPW = userPW;
18        this.userName = userName;
19        this.phone = phone;
20        this.ssn = ssn;
21        this.address = address;
22        this.isAdmin = isAdmin;
23    }
24 }

```

[User.java]

```

3 public class Music {
4     int musicID;
5     String musicTitle;
6     String lyricist; // 작사
7     String composer; // 작곡
8     String arranger; // 편곡
9     int hitsNum; // 조회수 (음악 재생 -> 조회수 +1)
10    int albumIdx; // [외래키] 앨범참조
11
12    ArrayList<String> artistName = new ArrayList<String>(); // 아티스트 이름
13    String albumTitle; // 앨범 명
14    ArrayList<String> mGenre = new ArrayList<String>(); // 장르
15
16    Music(){}
17● Music(String musicTitle){
18    this.musicTitle = musicTitle;
19}
20● Music(int musicID, String musicTitle, String albumTitle){
21    this.musicID = musicID;
22    this.musicTitle = musicTitle;
23    this.albumTitle = albumTitle;
24}
25● Music(int musicID, String musicTitle, String lyricist, String composer, String arranger, int hitsNum, int albumIdx){
26    this.musicID = musicID;
27    this.musicTitle = musicTitle;
28    this.lyricist = lyricist;
29    this.composer = composer;
30    this.arranger = arranger;
31    this.hitsNum = hitsNum;
32    this.albumIdx = albumIdx;
33}
34 }

```

[Music.java]

```

4 public class Album {
5     int albumID;
6     String albumTitle;
7     String albumType; // 앨범타입: Single/EP(싱글/EP), Studio album(정규앨범), Album(기타앨범)
8     String agency; // 발매사
9     String releaseCompany; // 기획사
10    Date releaseDate; // 날짜
11
12    ArrayList<String> artistName = new ArrayList<String>(); // 아티스트 이름
13    ArrayList<Integer> artistID = new ArrayList<Integer>(); // 아티스트 id
14    ArrayList<String> musicTitle = new ArrayList<String>(); // 음악 제목
15    ArrayList<Integer> musicID = new ArrayList<Integer>(); // 음악 id
16    ArrayList<String> aGenre = new ArrayList<String>(); // 장르
17
18    Album(){}
19● Album(int albumID){
20    this.albumID = albumID;
21}
22● Album(int albumID, String albumTitle, String albumType, String agency, String releaseCompany, Date releaseDate){
23    this.albumID = albumID;
24    this.albumTitle = albumTitle;
25    this.albumType = albumType;
26    this.agency = agency;
27    this.releaseCompany = releaseCompany;
28    this.releaseDate = releaseDate;
29}
30 }

```

[Album.java]

```

3 public class Artist {
4     int artistID;
5     String artistName;
6
7     ArrayList<Music> musics = new ArrayList<Music>();
8     ArrayList<Album> albums = new ArrayList<Album>();
9
10    Artist(){}
11•    Artist(int artistID, String artistName){
12        this.artistID = artistID;
13        this.artistName = artistName;
14    }
15 }

```

[Artist.java]

```

4 public class Playlist {
5     int playlistID;
6     String playlistTitle;
7     Date creationDate; // 날짜 타입은 다시한번 확인..
8     int userIdx; // [외래키] 유저참조
9
10    // numOfMusic 삭제 -> musics.size()로 구할 수 있기 때문
11    String userName;
12    ArrayList<Music> musics = new ArrayList<Music>();
13
14    Playlist(){}
15•    Playlist(int playlistID, String playlistTitle){
16        this.playlistID = playlistID;
17        this.playlistTitle = playlistTitle;
18    }
19•    Playlist(int playlistID, String playlistTitle, Date creationDate, int userIdx){
20        this.playlistID = playlistID;
21        this.playlistTitle = playlistTitle;
22        this.creationDate = creationDate;
23        this.userIdx = userIdx;
24    }
25 }

```

[Playlist.java]

- Main.java

Main 클래스는 이 프로그램의 핵심적인 기능 및 UI 등 모든 것을 제공하는 클래스이다. 때문에 main 함수를 제외하고도 정말 다양한 함수들이 많이 존재한다. 이 모든 코드를 전부 보고서에 기입하기에는 중복된 내용도 많고 보고서의 양도 많아질 수 있어, 적절히 생략해가며 함수 별로 설명할 예정이다. 그전에 main 함수 부분은 DB를 연결하고 Interface 화면을 실행하는 함수만 존재하므로, 코드 사진으로 설명을 대신한다.

```

public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println();
    System.out.println("=====");
    System.out.println("          ");
    System.out.println("      Welcome to MusicApp!      ");
    System.out.println("          ");
    System.out.println("=====");
    System.out.println();

    try {
        Class.forName("org.mariadb.jdbc.Driver"); // 여기에 path 정보 들어가야함!!
        con = DriverManager.getConnection(
            "jdbc:mariadb://127.0.0.1:3306/music_service",
            "eastsean",
            "east100");

        // login & create account
        User currentUser = new User();
        currentUser.isAdmin = 0;
        int loginMenuCheckNum = 0;
        while(loginMenuCheckNum == 0) loginMenuCheckNum = loginMenuCheck(currentUser);

        // music menu
        if(currentUser.isAdmin == 1) { // 관리자 화면..
            while(endApp != 1) adminInterface(currentUser);
        } else { // 관리자가 아닌 유저의 화면..
            while(endApp != 1) userInterface(currentUser);
        }

        con.close();
    } catch(ClassNotFoundException e) {
        System.out.println("[ERROR] Class not found.");
    } catch(SQLException e) {
        System.out.println("[ERROR] DB connection failed.");
    }
}

```

```

System.out.println();
System.out.println("=====");
System.out.println("          ");
System.out.println("          Good Bye~      ");
System.out.println("=====");
System.out.println();
}

```

[main()]

main()을 제외한 나머지 부분은 크게 계정 부분, 유저 부분, 관리자 부분, 총 3부분으로 구성되어 있지만, 함수들의 기능적인 측면에서 7가지로 재 분류 해보았다. 첫 번째로 계정을 생성 및 접속하는 기능, 두 번째로 메인 인터페이스를 보여주는 기능, 세 번째로 유저가 사용하는 기능, 네 번째로 myPlaylist()를 도와주는 기능, 다섯 번째로 관리자가 음원을 추가하는 기능, 여섯 번째로 관리자가 음원을 삭제하는 기능, 마지막으로 관리자가 유저를 관리하는 기능이다. 전체적으로 정리해보면 구성은 다음과 같다.

- | | |
|--|--|
| I. 계정을 생성 및 접속하는 기능 <ul style="list-style-type: none"> 1. loginMenuCheck() 2. login() 3. createAccount() II. 메인 인터페이스를 보여주는 기능 <ul style="list-style-type: none"> 1. adminInterface() 2. userInterface() III. 유저가 사용하는 기능 <ul style="list-style-type: none"> 1. musicSearch() 2. albumSearch() 3. artistSearch() 4. playlistSearch() 5. createPlaylist() 6. myPlaylist() 7. topMusic() IV. myPlaylist()를 도와주는 기능 <ul style="list-style-type: none"> 1. renamePlaylist() 2. addMusicInPlaylist() 3. deleteMusicInPlaylist() 4. deletePlaylist() | V. 관리자가 음원을 추가하는 기능 <ul style="list-style-type: none"> 1. musicRegistration() 2. albumRegistration() 3. artistRegistration() VI. 관리자가 음원을 삭제하는 기능 <ul style="list-style-type: none"> 1. musicDeletion() 2. albumDeletion() 3. artistDeletion() VII. 관리자가 유저를 관리하는 기능 <ul style="list-style-type: none"> 1. userManagement() 2. reviceUserInfo() 3. deleteUser() |
|--|--|

- 계정을 생성 및 접속하는 기능

loginMenuCheck()는 로그인을 하기 위한 인터페이스 화면이다. 이 화면에선 '1'을 누르면 로그인을 할 수 있는 페이지로 넘어가고(login() 호출), '2'를 누르면 계정을 생성할 수 있는 페이지로 넘어간다(createAccount() 호출). 로그인에 성공했을 시 loginOK 변수에 1이 들어가고 이를 통해 main() 함수에서 로그인을 확인할 수 있다. 여기는 프로그래밍 적으로 특별히 보여줄 부분이 없는 것 같아 사진은 생략했다. (함수 전체에 대한 코드는 Main.java의 21~64줄에 있다.)

`login()`은 `loginMenuCheck()`에서 '1'을 누르면 이동하는 페이지이다. id와 pw를 입력받는데, ID는 반드시 integer 값만 허용한다. 이는 id를 DB의 USER 테이블에서 User_id를 통해 확인하는데, User_id의 값이 integer 형태이기 때문이다. 입력받은 값을 DB USER 테이블에서 찾았다면 해당하는 유저를 반환하고, 입력받은 값이 존재하지 않으면 로그인에 실패되고 0을 return한다. id와 pw가 데이터베이스내에 존재하는지 확인하는 코드는 아래 사진으로 첨부했다. (함수 전체에 대한 코드는 Main.java의 65~119줄에서 확인할 수 있다.)

```

89     String sql = "select * from USER where User_id = ? AND User_pw = ?";
90     PreparedStatement ps = con.prepareStatement(sql);
91     ps.setInt(1, id);
92     ps.setString(2, pw);
93     ResultSet rs = ps.executeQuery();
94
95     while(rs.next()) {
96         int index = 1;
97         currentUser.userID = rs.getInt(index++);
98         currentUser.userPW = rs.getString(index++);
99         currentUser.userName = rs.getString(index++);
100        currentUser.phone = rs.getString(index++);
101        currentUser.ssn = rs.getString(index++);
102        currentUser.address = rs.getString(index++);
103        currentUser.isAdmin = rs.getInt(index++);
104    }
105
106    rs.close();
107    ps.close();

```

[`login()`에서 id와 pw의 일치 여부 확인]

`createAccount()`은 `loginMenuCheck()`에서 '2'를 누르면 이동하는 페이지이다. 계정을 새로 생성하는 부분인데, 여기서 특별한 점은 id 값을 프로그램이 알려준다는 것이다. 즉, 프로그램이 정해준 id를 가지고 계정에 접속해야 한다 (이는 마치 학교에서 신입학생들에게 학번을 부여해주는 것과 유사하다). id는 현재 데이터베이스 내에 존재하는 USER 테이블의 User_id의 최대값 + 1로 반환한다. 이렇게 설계한 이유는, 이 프로그램을 사용할 유저의 수와 integer 값의 범위를 고려했을 때 충분히 수용 가능하다고 판단했기 때문이다. 이와 관련된 자세한 코드는 아래 사진으로 첨부했다.

```

128     int id = 0;
129     /////////////////////////////////
130     String sql = "select MAX(User_id) from USER";
131     PreparedStatement ps = con.prepareStatement(sql);
132     ResultSet rs = ps.executeQuery();
133
134     while(rs.next()) {
135         int index = 1;
136         id = rs.getInt(index++) + 1;
137     }
138
139     rs.close();
140     ps.close();
141     /////////////////////////////////
142
143     System.out.println("Your ID is [" + id + "]");
144     System.out.println("Please don't forget this!");
145     System.out.println("This number is your ID to use MusicApp.");

```

[`createAccount()`에서 id값 할당]

그 뒤 사용자 계정 정보에 대한 입력을 받는데, 모든 입력을 다 받고나면 admin인지를 확인하는 추가적인 입력란이 나온다. 여기서 관리자가 아니라면 '0'을 입력하면 되고 관리자라면 관리자 코드를 입력해야 하는데 이 때 코드는 '2017029916' 이다. 이와 관련된 자세한 코드는 아래 사진으로 첨부했다.

```

196     String adminCode = "0";
197     System.out.println();
198     System.out.println("If you are admin, please enter the AdminCode.");
199     System.out.println("If you are not admin, please enter 0.");
200     System.out.print(">> ");
201     adminCode = sc.nextLine();
202
203     if(adminCode.equals("2017029916")) {
204         System.out.println("You are Admin!!!");
205         isAdmin = 1;
206     } else isAdmin = 0;

```

[`createAccount()`에서 관리자계정으로 생성할지 확인]

그리고 나서 아까 입력받은 정보를 DB에 저장한다. 이와 관련된 자세한 코드는 아래 사진으로 첨부했으며 함수 전체에 대한 코드는 Main.java의 120~228줄에서 확인할 수 있다.

```
209     sql = "insert into USER values (?, ?, ?, ?, ?, ?, ?)";
210     ps = con.prepareStatement(sql);
211     ps.setInt(1, id);
212     ps.setString(2, pw);
213     ps.setString(3, name);
214     ps.setString(4, phone);
215     ps.setString(5, ssn);
216     ps.setString(6, addr);
217     ps.setInt(7, isAdmin);
218     ps.executeUpdate();
219
220     ps.close();
```

[createAccount()에서 입력받은 정보를 DB에 삽입]

- 메인 인터페이스를 보여주는 기능

adminInterface()는 관리자 전용화면으로 관리자 계정으로 접속하면 보여주는 화면이다. 여기서 '0'~'7'까지의 입력은 사용자의 인터페이스와 완전히 동일하고, 추가적으로 '11'~'13'(관리자가 음원을 추가하는 기능), '21'~'23'(관리자가 음원을 삭제하는 기능), '31'(관리자가 사용자를 관리하는 기능)이 있다. 이외에 나머지 기능은 매우 단순하므로, 사진은 생략했다. (함수 전체에 대한 코드는 Main.java의 245~368줄에서 확인할 수 있다.)

userInterface()는 사용자 전용화면으로 관리자가 아닌 일반 유저가 접속하면 보여주는 화면이다. '1'은 음악 검색, '2'는 앨범 검색, '3'은 아티스트 검색, '4'는 플레이리스트 검색 '5'는 플레이리스트 생성, '6'은 마이 플레이리스트, '7'은 음원 통계를 나타내고 '0'을 입력하면 프로그램이 완전히 종료된다. (함수 전체에 대한 코드는 Main.java의 369~440줄에서 확인할 수 있다.)

- 유저가 사용하는 기능

musicSearch()는 찾고자 하는 음원 제목을 검색하면, 그 음원에 대한 정보를 보여주고 음원을 재생할 수 있는 기능을 제공한다. 제목은 대소문자 구분없이 검색할 수 있으며(LOWER 함수 사용) 검색한 내용이 제목에 포함되어 있으면(LIKE로 비교) 모두 보여준다.

```
475     String sql = "select * from MUSIC where LOWER(Music_title) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
476     PreparedStatement ps = con.prepareStatement(sql);
477     title = "%" + title + "%";
478     ps.setString(1, title);
479     ResultSet rs = ps.executeQuery();
480
481     while(rs.next()) {
482         Music tmpMusic = new Music();
483         int index = 1;
484         tmpMusic.musicID = rs.getInt(index++);
485         tmpMusic.musicTitle = rs.getString(index++);
486         tmpMusic.lyricist = rs.getString(index++);
487         tmpMusic.composer = rs.getString(index++);
488         tmpMusic.arranger = rs.getString(index++);
489         tmpMusic.hitsNum = rs.getInt(index++);
490         tmpMusic.albumIdx = rs.getInt(index++);
491         musics.add(tmpMusic);
492     }
493
494     rs.close();
495     ps.close();
```

[musicSearch()에서 제목을 검색하여 음원을 찾는 부분]

제목을 검색하고 나면 그 제목에 해당되는 음원들을 쭉 보여주는데, 이 중 음원 하나를 선택하면 그 음원에 앨범과 아티스트, 장르를 연결하여 'Music info'에서 보여준다.

```
535     // 음악 선택 & 아티스트, 장르, 앨범 연결
536     MUSIC tmpMusic = musics.get(num-1);
537     /////////////////////////////////////////////////
538     // 앨범, 아티스트 연결
539     tmpMusic.artistName.clear(); // 중복 들어간거 제거
540     sql = "select Album_title, Artist_name "
541         + "from MUSIC, MUSIC_MAKE, ARTIST, ALBUM "
542         + "where Music_id = ? AND Music_id = MM_Music_idx AND MM_Artist_idx = Artist_id AND Album_id = ?";
```

```

543     ps = con.prepareStatement(sql);
544     ps.setInt(1, tmpMusic.musicID);
545     ps.setInt(2, tmpMusic.albumIdx);
546     rs = ps.executeQuery();
547
548     while(rs.next()) {
549         int index = 1;
550         tmpMusic.albumTitle = rs.getString(index++);
551         tmpMusic.artistName.add(rs.getString(index++));
552     }
553
554     // 장르 연결
555     tmpMusic.mGenre.clear(); // 중복 들어간거 제거
556     sql = "select MGenre "
557         + "from MUSIC_GENRE "
558         + "where MIndex = ?";
559     ps = con.prepareStatement(sql);
560     ps.setInt(1, tmpMusic.musicID);
561     rs = ps.executeQuery();
562
563     while(rs.next()) {
564         int index = 1;
565         tmpMusic.mGenre.add(rs.getString(index++));
566     }
567
568     rs.close();
569     ps.close();
570

```

[musicSearch()에서 음원에 해당하는 앨범, 아티스트, 장르를 연결하는 부분]

'Music info'에서 '1'을 누르면 음악을 재생할 수 있는데, 음악 재생에 사용되는 코드는 아래 사진으로 첨부했다. 이외에 함수 전체에 대한 코드는 Main.java의 457~636줄에서 확인할 수 있다.

```

609     else if(cmd == 1) { // 음악재생
610         tmpMusic.hitsNum = tmpMusic.hitsNum + 1;
611
612         /////////////////////////////////////////////////
613         sql = "update MUSIC set Hits_num = ? where Music_id = ?";
614         ps = con.prepareStatement(sql);
615         ps.setInt(1, tmpMusic.hitsNum);
616         ps.setInt(2, tmpMusic.musicID);
617         ps.executeUpdate();
618
619         ps.close();
620         /////////////////////////////////////////////////
621
622         System.out.println("                                ");
623         System.out.println(" ~~~ Music play ~~~      ");
624         System.out.println("                                ");
625         System.out.println("Music has been played successfully!");
626         System.out.println("Return to music info.");

```

[musicSearch()에서 음원을 재생하는 부분]

albumSearch()는 찾고자 하는 앨범 제목을 검색하면, 그 앨범에 대한 정보를 보여주고 앨범에 포함된 음원을 보여준다. 제목 검색은 musicSearch()와 구조적으로 동일하므로 사진은 생략했다. 앨범 제목을 검색하고 나면 그 제목에 해당되는 앨범들을 쭉 보여주는데, 이 중 앨범 하나를 선택하면 그 앨범에 음원과 아티스트, 장르를 연결하여 'Album info'에서 보여준다. 이 부분의 코드는 아래 사진으로 첨부했으며, 함수 전체에 대한 코드는 Main.java의 637~812줄에서 확인할 수 있다.

```

713     // 앨범 선택 & 아티스트, 장르, 음악 연결
714     Album tmpAlbum = albums.get(num-1);
715     /////////////////////////////////////////////////
716     // 아티스트 연결
717     tmpAlbum.artistName.clear(); // 중복 들어간거 제거
718     sql = "select Artist_name "
719         + "from ALBUM, ALBUM_MAKE, ARTIST "
720         + "where Album_id = ? AND Album_id = AM_Album_idx AND AM_Artist_idx = Artist_id";
721     ps = con.prepareStatement(sql);
722     ps.setInt(1, tmpAlbum.albumID);
723     rs = ps.executeQuery();
724
725     while(rs.next()) {
726         int index = 1;
727         tmpAlbum.artistName.add(rs.getString(index++));
728     }
729
730     // 음악 연결
731     tmpAlbum.musicTitle.clear(); // 중복 들어간거 제거
732     sql = "select Music_title "
733         + "from MUSIC "
734         + "where Album_idx = ?";

```

```

735     ps = con.prepareStatement(sql);
736     ps.setInt(1, tmpAlbum.albumID);
737     rs = ps.executeQuery();
738
739     while(rs.next()) {
740         int index = 1;
741         tmpAlbum.musicTitle.add(rs.getString(index++));
742     }
743
744     // 장르 연결
745     tmpAlbum.aGenre.clear(); // 중복 들어간거 제거
746     sql = "select AGenre "
747         + "from ALBUM_GENRE "
748         + "where AIndex = ?";
749     ps = con.prepareStatement(sql);
750     ps.setInt(1, tmpAlbum.albumID);
751     rs = ps.executeQuery();
752
753     while(rs.next()) {
754         int index = 1;
755         tmpAlbum.aGenre.add(rs.getString(index++));
756     }
757
758     rs.close();
759     ps.close();

```

[albumSearch()에서 앨범에 해당하는 음원, 아티스트, 장르를 연결하는 부분]

`artistSearch()`는 찾고자 하는 아티스트 이름을 검색하면, 그 아티스트가 발매한 음원을 보여준다. 이름 검색 및 전체적인 구조는 이전 함수들과 동일하므로, 아티스트와 음원을 연결하는 부분만 사진으로 첨부했으며, 함수 전체에 대한 코드는 Main.java의 813~942줄에서 확인할 수 있다.

```

885     Artist tmpArtist = artists.get(num-1);
886     /////////////////////////////////////////////////
887     // 음악, 앨범 연결
888     tmpArtist.musics.clear(); // 중복 들어간거 제거
889     sql = "select Music_title, Album_title "
890         + "from ARTIST, MUSIC_MAKE, MUSIC, ALBUM "
891         + "where Artist_id = ? AND Artist_id = MM_Artist_idx AND MM_Music_idx = Music_id AND Album_idx = Album_id";
892     ps = con.prepareStatement(sql);
893     ps.setInt(1, tmpArtist.artistID);
894     rs = ps.executeQuery();
895
896     while(rs.next()) {
897         int index = 1;
898         tmpArtist.musics.add(new Music(rs.getString(index++)));
899         tmpArtist.musics.get(tmpArtist.musics.size()-1).albumTitle = rs.getString(index++);
900     }
901
902     rs.close();
903     ps.close();

```

[artistSearch()에서 아티스트가 발매한 음원을 연결하는 부분]

`playlistSearch()`는 찾고자 하는 플레이리스트 제목을 검색하면, 그 플레이리스트에 포함된 음원들을 보여준다. 제목 검색 및 전체적인 구조는 이전 함수들과 비슷하지만, 플레이리스트에는 음원 전체 재생 기능이 존재하여 이 부분과 플레이리스트와 음원을 연결하는 부분을 사진으로 첨부했다. (함수 전체에 대한 코드는 Main.java의 943~1135줄에서 확인할 수 있다.)

```

1020 Playlist tmpPlaylist = playlists.get(num-1);
1021 /////////////////////////////////////////////////
1022 // 유저 연결
1023 sql = "select User_name "
1024     + "from PLAYLIST, USER "
1025     + "where Playlist_id = ? AND User_idx = User_id";
1026 ps = con.prepareStatement(sql);
1027 ps.setInt(1, tmpPlaylist.playlistID);
1028 rs = ps.executeQuery();
1029
1030 while(rs.next()) {
1031     int index = 1;
1032     tmpPlaylist.userName = rs.getString(index++);
1033 }
1034
1035 // 음악, 앨범 연결
1036 tmpPlaylist.musics.clear();
1037 sql = "select Music_id, Music_title, Album_title "
1038     + "from PLAYLIST, MUSIC_IN_LIST, MUSIC, ALBUM "
1039     + "where Playlist_id = ? AND MIL_Playlist_idx = Playlist_id AND MIL_Music_idx = Music_id AND Album_idx = Album_id";
1040 ps = con.prepareStatement(sql);
1041 ps.setInt(1, tmpPlaylist.playlistID);
1042 rs = ps.executeQuery();
1043
1044 while(rs.next()) {
1045     int index = 1;
1046     tmpPlaylist.musics.add(new Music(rs.getInt(index++), rs.getString(index++), rs.getString(index++)));
1047 }

```

```

1048 // 음악 -> 아티스트 연결
1049 for(Music tmpMusic: tmpPlaylist.musics) {
1050     tmpMusic.artistName.clear();
1051     sql = "select Artist_name "
1052         + "from MUSIC, MUSIC_MAKE, ARTIST "
1053         + "where Music_id = ? AND MM_Music_idx = Music_id AND MM_Artist_idx = Artist_id";
1054     ps = con.prepareStatement(sql);
1055     ps.setInt(1, tmpMusic.musicID);
1056     rs = ps.executeQuery();
1057
1058     while(rs.next()) {
1059         int index = 1;
1060         tmpMusic.artistName.add(rs.getString(index++));
1061     }
1062 }
1063
1064
1065 rs.close();
1066 ps.close();

```

[playlistSearch()에서 플레이리스트를 만든 유저와 리스트에 포함된 음원을 연결하는 부분]

```

1103             else if(cmd == 1) { // 전체 음악재생
1104                 for(Music tmpMusic: tmpPlaylist.musics) {
1105
1106                     /////////////////////////////////
1107                     sql = "update MUSIC set Hits_num = Hits_num + 1 where Music_id = ?";
1108                     ps = con.prepareStatement(sql);
1109                     ps.setInt(1, tmpMusic.musicID);
1110                     ps.executeUpdate();
1111
1112                     ps.close();
1113                     /////////////////////////////////
1114
1115     }
1116     if(tmpPlaylist.musics.size() == 0) {
1117         System.out.println("          ");
1118         System.out.println("No music in the playlist!");
1119     } else {
1120         System.out.println("          ");
1121         System.out.println("      ~~~ Music play ~~~      ");
1122         System.out.println("          ");
1123         System.out.println("Music has been played successfully!");
1124     }
1125     System.out.println("Return to playlist info.");

```

[playlistSearch()에서 전체 음원을 재생하는 부분]

createPlaylist()는 플레이리스트를 만드는 기능을 제공한다. 제목을 입력받고, 그 플레이리스트의 id를 할당해주는데 이는 createAccount()와 매우 유사하게 동작한다(최대값 + 1). creationDate는 플레이리스트를 만든 현재 날짜가 들어가고, 플레이리스트를 다 만들었을 시, '1'을 누르면 바로 음원을 추가할 수 있다. 이 기능은 addMusicInPlaylist()함수를 호출하여 수행한다. createPlaylist()는 이전 createAccount()와 상당부분 비슷한 동작을 하므로 사진은 생략했고, 함수 전체에 대한 코드는 Main.java의 1136~1220줄에서 확인할 수 있다.

myPlaylist()는 현재 MusicApp에 접속한 사용자가 만든 플레이리스트들을 보여준다. DB의 PLAYLIST 테이블에서 User_idx가 지금 접속한 유저와 맞는 것들을 보여주는 것으로 select를 처리했으며, 나머지 부분은 playlistSearch()와 상당부분 유사하다. 하지만 myPlaylist()는 'Playlist info'에서 차이를 보이는데, 음악을 전체 재생하는 기능 외에도 '2'를 누르면 플레이리스트를 수정하는 기능이 있다. 수정하는 화면에서 '1'을 누르면 플레이리스트 이름을 변경할 수 있고, '2'를 누르면 현재 플레이리스트에 음악을 추가할 수 있고, '3'을 누르면 현재 플레이리스트에 음악을 삭제할 수 있으며, '4'를 누르면 현재 플레이리스트를 삭제할 수 있다. 이에 해당되는 각각의 함수들은 'myPlaylist()'를 도와주는 기능'란에서 자세히 설명할 예정이고, 전체 함수에 대한 코드는 Main.java의 1221~1444줄에서 확인할 수 있다.

```

1393             } else if(cmd == 2) { |
1394                 // Edit!!
1395                 // 플레이리스트 이름 변경, 음악 추가, 음악 제거, 플레이리스트 삭제
1396                 // 1. renamePlaylist
1397                 // 2. addMusicInPlaylist
1398                 // 3. deleteMusicInPlaylist
1399                 // 4. deletePlaylist
1400                 System.out.println();

```

```

1401         System.out.println("What do you want to do?");
1402         System.out.println(" 1. Rename playlist");
1403         System.out.println(" 2. Add music in playlist");
1404         System.out.println(" 3. Delete music in playlist");
1405         System.out.println(" 4. Delete playlist");
1406         System.out.println();
1407         System.out.println(" 0. Return to playlist info");
1408         System.out.print(">> ");
1409
1410         while (!sc.hasNextInt()) {
1411             sc.next();
1412             System.out.print(">> ");
1413         }
1414         int cmd2 = sc.nextInt();
1415         sc.nextLine(); // enter
1416
1417         if(cmd2 == 0) continue;
1418         else if(cmd2 == 1) renamePlaylist(tmpPlaylist);
1419         else if(cmd2 == 2) addMusicInPlaylist(tmpPlaylist);
1420         else if(cmd2 == 3) deleteMusicInPlaylist(tmpPlaylist);
1421         else if(cmd2 == 4) {
1422             if(deletePlaylist(playlists, tmpPlaylist) == 0) continue;
1423         }
1424         else {
1425             System.out.println("[ERROR] It is not allowed cmd!");
1426             System.out.println("Go to menu..");
1427             return;
1428         }
1429
1430         System.out.println("                                         ");
1431         System.out.println("Playlist has been edited successfully!");
1432         System.out.println("Press the enter key to return to the main menu.");
1433         sc.nextLine();
1434     }

```

[myPlaylist()에서 플레이리스트를 수정하는 부분]

topMusic()은 음원들의 조회수를 통해 음원의 순위를 출력해주는 기능을 제공한다. '1'을 누르면 Top 10 음악, '2'를 누르면 Top 100 음악, '3'를 누르면 Top 200 음악을 보여주고 현재 DB에 그만큼의 음악이 존재하지 않을 시 보여줄 수 있는 만큼 보여준다. 음원을 가져오는 부분은 아래 사진으로 첨부했으며, 'Music list' 부분부터는 musicSearch()와 상당부분 유사하다. (전체 함수에 대한 코드는 Main.java의 1445~1640줄에서 확인할 수 있다.)

```

1482     String sql = "select * from MUSIC order by Hits_num desc";
1483     PreparedStatement ps = con.prepareStatement(sql);
1484     ResultSet rs = ps.executeQuery();
1485
1486     while(rs.next()) {
1487         Music tmpMusic = new Music();
1488         int index = 1;
1489         tmpMusic.musicID = rs.getInt(index++);
1490         tmpMusic.musicTitle = rs.getString(index++);
1491         tmpMusic.lyricist = rs.getString(index++);
1492         tmpMusic.composer = rs.getString(index++);
1493         tmpMusic.arranger = rs.getString(index++);
1494         tmpMusic.hitsNum = rs.getInt(index++);
1495         tmpMusic.albumIdx = rs.getInt(index++);
1496         musics.add(tmpMusic);
1497     }
1498
1499     rs.close();
1500     ps.close();

```

[topMusic()에서 음원을 가져오는 부분]

- myPlaylist()를 도와주는 기능

renamePlaylist()은 인자로 받은 플레이리스트의 제목을 변경하는 함수이다. 함수 자체가 워낙 간단하기 때문에 플레이리스트의 제목을 변경하는 SQL 부분만 사진으로 첨부했으며, 전체 함수에 대한 코드는 Main.java의 1657~1694줄에서 확인할 수 있다.

```

1686     String sql = "update PLAYLIST set Playlist_title = ? where Playlist_id = ?";
1687     PreparedStatement ps = con.prepareStatement(sql);
1688     ps.setString(1, tmpPlaylist.playlistTitle);
1689     ps.setInt(2, tmpPlaylist.playlistID);
1690     ps.executeUpdate();
1691
1692     ps.close();

```

[renamePlaylist()에서 플레이리스트 제목을 업데이트 하는 부분]

`addMusicInPlaylist()`는 함수 명에서도 보이듯이 플레이리스트에 음악을 추가하는 함수이다. 음악을 찾는 과정은 `musicSearch()`와 동일하고, 'Music info'부분에서 '1'을 누르면 음악을 추가할 수 있다. 이 또한 이전 함수들과 중복된 부분이 많아, 음악을 추가하는 부분만 사진으로 첨부했으며, 전체 함수에 대한 코드는 `Main.java`의 1695~1879줄에서 확인할 수 있다.

```

1849     sql = "insert into MUSIC_IN_LIST " // 음악 중복 검사
1850         + "select ?,? "
1851         + "from DUAL "
1852         + "where NOT EXISTS ( "
1853         + " select * "
1854         + " from MUSIC_IN_LIST "
1855         + " where MIL_Playlist_idx = ? AND MIL_Music_idx = ?)";
1856     ps = con.prepareStatement(sql);
1857     ps.setInt(1, tmpPlaylist.playlistID);
1858     ps.setInt(2, tmpMusic.musicID);
1859     ps.setInt(3, tmpPlaylist.playlistID);
1860     ps.setInt(4, tmpMusic.musicID);
1861     ps.executeUpdate();
1862
1863     ps.close();

```

[`addMusicInPlaylist()`에서 플레이리스트 안에 음악을 추가하는 부분]

`deleteMusicInPlaylist()`는 인자로 받은 플레이리스트에서 음악을 삭제하는 함수이다. 이 또한 위 함수들과 마찬가지로 단순한 구조이기에 음악을 삭제하는 부분만 사진으로 첨부했다. (전체 함수에 대한 코드는 `Main.java`의 1880~1931줄에서 확인할 수 있다.)

```

1916     String sql = "delete from MUSIC_IN_LIST "
1917         + "where MIL_Playlist_idx = ? AND MIL_Music_idx = ?";
1918     PreparedStatement ps = con.prepareStatement(sql);
1919     ps.setInt(1, tmpPlaylist.playlistID);
1920     ps.setInt(2, tmpMusic.musicID);
1921     ps.executeUpdate();
1922
1923     ps.close();

```

[`deleteMusicInPlaylist()`에서 플레이리스트 안에 음악을 삭제하는 부분]

`deletePlaylist()`는 인자로 받은 플레이리스트를 삭제하는 함수이다. 이 함수는 음악이 들어있다는 정보를 갖고 있는 `MUSIC_IN_LIST`의 내용을 먼저 비운 뒤, `PLAYLIST`에서 삭제해야 하므로 2개의 SQL문으로 구성 되어있다. (전체 함수에 대한 코드는 `Main.java`의 1932~1976줄에서 확인할 수 있다.)

```

1957     // MUSIC_IN_LIST 안에 데이터 삭제
1958     String sql = "delete from MUSIC_IN_LIST where MIL_Playlist_idx = ?";
1959     PreparedStatement ps = con.prepareStatement(sql);
1960     ps.setInt(1, tmpPlaylist.playlistID);
1961     ps.executeUpdate();
1962     ps.close();
1963
1964     // PLAYLIST 삭제
1965     sql = "delete from PLAYLIST where Playlist_id = ?";
1966     ps = con.prepareStatement(sql);
1967     ps.setInt(1, tmpPlaylist.playlistID);
1968     ps.executeUpdate();
1969     ps.close();

```

[`deletePlaylist()`에서 플레이리스트를 삭제하는 부분]

- 관리자가 음원을 추가하는 기능

`musicRegistration()`은 음원을 등록하는 함수인데, 음원을 등록하기전에는 반드시 앨범이 존재해야한다. 앨범을 발매한 아티스트가 음원도 발매한다는 조건이 있으며, 이 함수에선 음원을 등록할 앨범을 먼저 찾고, 그 앨범에 음원을 등록한다. 앨범을 찾는 과정부터 'Album info'를 보여주는 과정까지는 `albumSearch()`와 거의 동일하다. `Music_id`는 다른 `id`를 생성하는 것과 동일한 방식(DB `MUSIC` 테이블의 `Music_id`들 중 최대값+1)으로 진행되고, 음원의 정보를 입력받는 부분은 계정을 생성하는 부분과 비슷한 과정을 반복한다. 마지막으로 음악을 추가하고 그 음악의 장르와 아티스트를 연결하는 부분은 사진으로 첨부했으며, 전체 함수에 대한 코드는 `Main.java`의 1993~2243줄에서 확인할 수 있다.

```

2205 // 음악 추가
2206 sql = "insert into MUSIC values (?, ?, ?, ?, ?, ?, ?)";
2207 ps = con.prepareStatement(sql);
2208 ps.setInt(1, tmpMusic.musicID);
2209 ps.setString(2, tmpMusic.musicTitle);
2210 ps.setString(3, tmpMusic.lyricist);
2211 ps.setString(4, tmpMusic.composer);
2212 ps.setString(5, tmpMusic.arranger);
2213 ps.setInt(6, 0);
2214 ps.setInt(7, tmpAlbum.albumID);
2215 ps.executeUpdate();
2216 ps.close();
2217
2218 // 장르 추가
2219 for(String tmpGenre: tmpMusic.mGenre) {
2220     sql = "insert into MUSIC_GENRE values (?,?)";
2221     ps = con.prepareStatement(sql);
2222     ps.setInt(1, tmpMusic.musicID);
2223     ps.setString(2, tmpGenre);
2224     ps.executeUpdate();
2225     ps.close();
2226 }
2227
2228 // 아티스트 추가
2229 for(Integer tmpArtist: tmpAlbum.artistID) {
2230     sql = "insert into MUSIC_MAKE values (?,?)";
2231     ps = con.prepareStatement(sql);
2232     ps.setInt(1, tmpArtist);
2233     ps.setInt(2, tmpMusic.musicID);
2234     ps.executeUpdate();
2235     ps.close();
2236 }

```

[musicRegistration()에서 음원을 추가하는 부분]

albumRegistration()은 앨범을 등록하는 함수인데, 앨범을 등록하기전에는 반드시 아티스트가 존재해야한다. 아티스트를 찾고 리스트를 보여주는 과정은 artistSearch()와 거의 동일하며, Album_id, 장르, 아티스트 추가도 위의 musicRegistration()에서 했던 방식과 거의 동일하다. 이전 함수와 중복되는 부분이 많아 사진은 생략했으며, 전체 함수에 대한 코드는 Main.java의 2244~2421줄에서 확인할 수 있다.

artistRegistration()은 아티스트를 등록하는 함수이다. 아티스트는 등록하기 위한 선형조건이 따로 없으므로, 이전 함수들보다 과정이 더 단순하며 SQL문 또한 간단하다. 전반적으로도 이전과 중복된 부분이 많아 사진은 생략했고, 전체 함수에 대한 코드는 Main.java의 2422~2486줄에서 확인할 수 있다.

- 관리자가 음원을 삭제하는 기능

musicDeletion()은 음원을 DB에서 완전히 삭제하는 함수이다. 음원을 찾고 'Music info'까지 출력하는 과정은 musicSearch()와 동일하고, info 화면에서 '1'을 누르면 음원을 삭제할 수 있다. 음원을 삭제하기 위해 해당하는 MUSIC_GENRE와 MUSIC_MAKE를 먼저 삭제한 뒤, 모든 플레이리스트와 그 음악과의 연결을 없애고, 마지막으로 MUSIC에서 음원을 삭제한다. 설명은 조금 복잡하지만 삭제하는 과정을 사진으로 보면 매우 단순하다.(전체 함수에 대한 코드는 Main.java의 2503~2698줄에서 확인할 수 있다.)

```

2656 // 장르 삭제
2657 sql = "delete from MUSIC_GENRE where mIndex = ?";
2658 ps = con.prepareStatement(sql);
2659 ps.setInt(1, tmpMusic.musicID);
2660 ps.executeUpdate();
2661 ps.close();
2662
2663 // make 삭제
2664 sql = "delete from MUSIC_MAKE where MM_Music_idx = ?";
2665 ps = con.prepareStatement(sql);
2666 ps.setInt(1, tmpMusic.musicID);
2667 ps.executeUpdate();
2668 ps.close();
2669
2670 // 플레이리스트 인 삭제
2671 sql = "delete from MUSIC_IN_LIST where MIL_Music_idx = ?";
2672 ps = con.prepareStatement(sql);
2673 ps.setInt(1, tmpMusic.musicID);
2674 ps.executeUpdate();
2675 ps.close();

```

```

2676
2677         // 음악 삭제
2678         sql = "delete from MUSIC where Music_id = ?";
2679         ps = con.prepareStatement(sql);
2680         ps.setInt(1, tmpMusic.musicID);
2681         ps.executeUpdate();
2682         ps.close();

```

[musicDeletion()에서 음원을 삭제하는 부분]

albumDeletion()은 앨범을 DB에서 완전히 삭제하는 함수이다. 앨범을 찾고 'Album info'까지 출력하는 과정은 albumSearch()와 동일하고, info 화면에서 '1'을 누르면 앨범을 삭제할 수 있다. 앨범도 음원과 마찬가지로 장르와 MAKE를 먼저 삭제하고, 그 뒤에 ALBUM 테이블에서 앨범 삭제를 진행한다. 앨범이 음원 삭제와 다른 점은 음원은 플레이리스트에 연결된 걸 끊는 작업이 필요하다는 것이지만, 앨범은 앨범 내에 존재하는 음원도 같이 삭제해주어야 한다는 점이 다르다. 앨범 내에 음원 삭제는 musicDeletion()과 동일하므로 따로 사진으로 첨부하지는 않았다. (전체 함수에 대한 코드는 Main.java의 2699~2937줄에서 확인할 수 있다.)

artistDeletion()은 아티스트를 DB에서 완전히 삭제하는 함수이다. 이 함수도 전체적인 구조는 이전 함수들과 동일하지만, 차이점은 아티스트가 발매한 음원과 앨범도 같이 삭제해주어야 한다는 점이다. MUSIC_MAKE와 ALBUM_MAKE에 해당하는 아티스트 인스턴스를 먼저 삭제하고 음원과 앨범 삭제를 진행하는데 이는 위 함수들의 과정과 똑같다. 중복된 내용이므로 사진은 따로 첨부하지 않았지만, 전체 함수에 대한 코드는 Main.java의 2938~3203줄에서 확인할 수 있다.

- 관리자가 유저를 관리하는 기능

userManagement()은 MusicApp에 등록되어 있는 유저들의 정보를 보여주는 함수이다. 유저의 정보를 가져와서 리스트로 출력하고, 'User info'를 제공하는 부분까지는 이전 musicSearch()와 같은 함수들과 구조적으로 큰 차이가 없다. 'User info'에서 '1'을 누르면 user를 edit 할 수 있는데, 수정 화면에서 '1'을 누르면 유저의 신상 정보 및 비밀번호를 변경할 수 있고 가능하고, '2'를 누르면 유저를 삭제할 수 있다. 수정 화면 인터페이스에 대한 사진은 아래에 첨부했으며, 전체 함수에 대한 코드는 Main.java의 3219~3385줄에서 확인할 수 있다.

```

3339         else if(cmd == 1) {
3340             // Edit!!
3341             // 유저 정보변경, 유저삭제
3342             // 1. reviceUserInfo
3343             // 2. deleteUser
3344             System.out.println();
3345             System.out.println("What do you want to do?");
3346             System.out.println(" 1. Revise user info");
3347             System.out.println(" 2. Delete user");
3348             System.out.println();
3349             System.out.println(" 0. Return to user info");
3350             System.out.print("> ");
3351
3352             while (!sc.hasNextInt()) {
3353                 sc.next();
3354                 System.out.print("> ");
3355             }
3356             int cmd2 = sc.nextInt();
3357             sc.nextLine(); // enter
3358
3359             if(cmd2 == 0) continue;
3360             else if(cmd2 == 1) reviceUserInfo(tmpUser);
3361             else if(cmd2 == 2) {
3362                 if(deleteUser(currentUser, tmpUser, users) == 0) continue;
3363             }
3364             else {
3365                 System.out.println("[ERROR] It is not allowed cmd!");
3366                 System.out.println("Go to menu...");
3367                 return;
3368             }
3369
3370             System.out.println("                                         ");
3371             System.out.println("User has been edited successfully!");
3372             System.out.println("Press the enter key to return to the main menu.");
3373             sc.nextLine();
3374             return;

```

[userManagement()에서 유저를 edit하는 인터페이스]

`reviceUserInfo()`는 유저의 신상정보 및 비밀번호를 변경하는 함수이고, `createAccount()`와 비슷하게 정보를 입력받은 뒤, UPDATE SQL 문으로 유저를 업데이트한다. (전체 함수에 대한 코드는 Main.java의 3386~3454줄에서 확인할 수 있다.)

```
3440     String sql = "update USER "
3441         + "set User_pw = ?, User_name = ?, Phone = ?, Ssn = ?, Address = ?, Is_admin = ? where User_id = ?";
3442     PreparedStatement ps = con.prepareStatement(sql);
3443     ps.setString(1, tmpUser.userPW);
3444     ps.setString(2, tmpUser.userName);
3445     ps.setString(3, tmpUser.phone);
3446     ps.setString(4, tmpUser.ssn);
3447     ps.setString(5, tmpUser.address);
3448     ps.setInt(6, tmpUser.isAdmin);
3449     ps.setInt(7, tmpUser.userID);
3450     ps.executeUpdate();
3451
3452     ps.close();
```

[`reviceUserInfo()`에서 UPDATE SQL문]

`deleteUser()`는 인자로 받은 유저를 삭제하는 함수이다. 이전 '관리자가 음원을 삭제하는 부분'에서 본 함수들과 비슷한 구조를 가지고 있고, 유저를 삭제하기전에 MUSIC_IN_LIST, 유저가 만든 PLAYLIST를 순차적으로 삭제한 뒤 마지막으로 유저를 삭제한다. 다른 함수들과 기능적인 측면에서 차이는 분명이 존재하지만 전체적인 흐름과 구조는 큰 차이가 없어 사진은 생략했고, 전체 함수에 대한 코드는 Main.java의 3455~3518줄에서 확인할 수 있다.

5. 수행되는 기능 스크린샷

- 컴파일을 통해 실행하기

자바 버전은 15.0.1을 사용했다. 프로그램을 실행하기 위해선 모든 파일들이 다 같은 디렉토리 내에 존재해야 한다. 처음 git clone을 한 상태에서 디렉토리 목록은 다음과 같다.

```
eastsea@EastSeau-iMac MusicApp_Assignment % ls
Album.java          Main.java          Playlist.java        mariadb-java-client-2.7.0.jar
Artist.java         Music.java         User.java           music_service.sql
```

일단 프로그램을 실행하기 전, dump file을 백업해야 한다. 백업하기 위해 여기서는 mariaDB에 새로 데이터베이스를 만들어 진행했다.

```
eastsea@EastSeau-iMac MusicApp_Assignment % mysql -ueastsea -peast100
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.5.6-MariaDB Homebrew

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database music_service;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| company |
| dreamhome |
| information_schema |
| music_service |
| mysql |
| performance_schema |
| school |
| test |
+-----+
```

mariaDB에 새로 데이터베이스를 만들고 나면 백업을 진행하고, javac 명령어를 이용해 java 컴파일까지 같이 진행한다.

```
eastsea@EastSeau-iMac MusicApp_Assignment % mysql -ueastsea -peast100 music_service < music_service.sql
eastsea@EastSeau-iMac MusicApp_Assignment % javac Album.java Music.java Artist.java Playlist.java Main.java User.java
```

그리고 환경변수를 설정해 java 프로그램을 실행시킨다. 프로그램이 실행에 성공하면 다음과 같은 화면이 뜬다.

```

eastsea@EastSeoui-iMac MusicApp_Assignment % java -cp .:mariadb-java-client-2.7.0.jar Main
=====
Welcome to MusicApp!
=====

-----
      Menu
1. Log in
2. Create account
0. Exit MusicApp
>> 1

```

- 계정 생성 및 로그인 하기

메인 화면에서 '2'를 누르면 계정을 생성할 수 있다.

```

-----
      Menu
1. Log in
2. Create account
0. Exit MusicApp
>> 2
-----

      Create account
-----
Your ID is [20101006]
Please don't forget this!
This number is your ID to use MusicApp.

Add please write this form
1. Password(less than 15 characters) >> test6
2. Name(less than 30 characters) >> Command_test6
3. Phone(less than 15 characters) >> 010-1111-0006
4. SSN(less than 15 characters) >> 101010-0000006
5. Address(less than 30 characters) >> 6, Busan

1. ID      : 20101006
2. Password: test6
3. Name    : Command_test6
4. Phone   : 010-1111-0006
5. SSN    : 101010-0000006
6. Address : 6, Busan
-> Is that true? (Yes:1, No:2) >> 1

```

다음과 같이 정보를 기입하고 '1'을 누르면 관리자 코드를 입력받는 란이 나온다. '2'를 누르게 되면 처음부터 다시 작성할 수 있다. 다음 사진은 '2'를 눌러 정보를 다시 기입하고 '1'을 누른 화면이다.

```

-> Is that true? (Yes:1, No:2) >> 2

Add please write this form
1. Password(less than 15 characters) >> test6
2. Name(less than 30 characters) >> Command_test6
3. Phone(less than 15 characters) >> 010-1111-0006
4. SSN(less than 15 characters) >> 101010-0000006
5. Address(less than 30 characters) >> 6, Busan

1. ID      : 20101006
2. Password: test6
3. Name    : Command_test6
4. Phone   : 010-1111-0006
5. SSN    : 101010-0000006
6. Address : 6, Busan
-> Is that true? (Yes:1, No:2) >> 1

If you are admin, please enter the AdminCode.
If you are not admin, please enter 0.
>> 0

```

관리자 코드를 입력하면 관리자로 계정이 생성되지만 여기선 '0'을 눌러 일반 유저로 계정을 생성했다. 참고로 관리자 코드는 '2017029916'이다.

```

If you are admin, please enter the AdminCode.
If you are not admin, please enter 0.
>> 0

Now, account creation is complete.
Please log in from the menu.

-----
      Menu
1. Log in
2. Create account
0. Exit MusicApp

```

계정 생성이 완료되면 다시 메인 화면이 나온다. 아까 만든 계정으로 로그인을 진행해 보았고 성공적으로 로그인이 완료되면 MusicApp의 메인 메뉴가 나오게 된다.

```
-----  
          Menu  
1. Log in  
2. Create account  
  
0. Exit MusicApp  
-----  
>> 1  
  
-----  
          Log in  
-----  
ID >> 20101006  
Password >> test6  
  
Log in success! Hello~  
  
-----  
          Menu  
1. Music Search  
2. Album Search      (Only search)  
3. Artist Search     (Only search)  
4. Playlist Search  
5. Create Playlist  
6. My Playlist  
7. Top Music  
  
0. Exit MusicApp  
-----  
>> [
```

- 유저 Menu 기능

먼저 '1'을 눌러 음악 검색에 들어갔다. 음악 검색은 LIKE 문으로 비교되기 때문에 문자열을 포함하고 있다면 검색 목록에 출력된다. 여기서는 'WANNABE' 노래를 찾기 위해 'wanna'로 검색해보았다.

```
-----  
          Menu  
1. Music Search  
2. Album Search      (Only search)  
3. Artist Search     (Only search)  
4. Playlist Search  
5. Create Playlist  
6. My Playlist  
7. Top Music  
  
0. Exit MusicApp  
-----  
>> 1  
  
-----  
          Music Search  
-----  
Enter the music title you want to find.  
Just press enter and the all music will be shown.  
>> wanna  
  
-----  
          Music list  
No.           Music Title  
1             WANNABE  
-----  
Enter the 'No.' to see more about the music.  
And you can play the music!  
Enter 0 to go to the menu.  
>> [
```

해당하는 음악을 선택하면 그 음악에 대한 'Music info'를 제공한다.

```
-----  
Enter the 'No.' to see more about the music.  
And you can play the music!  
Enter 0 to go to the menu.  
>> 1  
  
-----  
          Music info  
1. Title   : WANNABE  
2. Artist  : ITZY  
3. Genre   : Dance  
4. Lyricist: 별들의 전쟁  
5. Composer: 별들의 전쟁  
6. Arranger: GALACTIKA  
7. Hits    : 24  
8. in Album: ITz ME  
-----  
Please press the command. (Play to music: 1, Return to music list: 2)  
Enter 0 to go to the menu.  
>> [
```

여기서 '1'을 누르면 음악을 재생할 수 있는데, 음악을 재생하게 되면 음악의 조회수가 올라간다. '2'를 누르면 다시 'Music list'로 복귀한다. '0'을 누르면 메인 화면으로 나갈 수 있다.

```
7. Hits : 24
8. in Album: ITz ME
-----
Please press the command. (Play to music: 1, Return to music list: 2)
Enter 0 to go to the menu.
>> 1

~~~~ Music play ~~~~

Music has been played successfully!
Return to music info.

-----
          Music info
1. Title : WANNABE
2. Artist : ITZY
3. Genre : Dance
4. Lyricist: 별들의전쟁
5. Composer: 별들의전쟁
6. Arranger: GALACTIKA
7. Hits : 25
8. in Album: ITz ME
-----
Please press the command. (Play to music: 1, Return to music list: 2)
Enter 0 to go to the menu.
>> 
```

'0'을 눌러 메인 메뉴로 나간 뒤 '2'를 눌러 이번엔 앨범 검색을 들어갔다.

```
>> 0

-----
          Menu
1. Music Search
2. Album Search (Only search)
3. Artist Search (Only search)
4. Playlist Search
5. Create Playlist
6. My Playlist
7. Top Music
0. Exit MusicApp
-----
>> 2

-----
          Album Search
Enter the album title you want to find.
Just press enter and the all album will be shown.
>> 
```

여기서 그대로 엔터를 치면 전체 앨범 목록이 나온다. 이는 음악 검색을 포함한 거의 대부분의 검색창에서 포함하는 기능이다.

```
-----
          Album Search
Enter the album title you want to find.
Just press enter and the all album will be shown.
>>

-----
          Album list
No.           Album Title
1             Love poem
2             Palette
3             Dynamite
4             ITz ME
5             Boyhood
6             쇼미더머니 9 Episode 1
7             I am
8             YESTERDAY
9             Brother Act.
10            우연히 봄
11            THE ALBUM
-----
Enter the 'No.' to see more about the album.
Enter 0 to go to the menu.
>> 
```

DB에 저장되어 있던 앨범이 전부 나왔다. 보고싶은 앨범 목록을 누르면 'Album info'를 볼 수 있다.

```
>> 5

-----
          Album info
1. Title      : Boyhood
2. Artist     : CHANGMO
3. Genre      : Hip Hop/Rap/Song
4. Type       : Studio album
5. Agency     : Genie Music
6. Release Company: Ambition Musik
```

```
7. Release Date : 2019-11-29
There is 3 music(s) in the album
-> 빌업어
-> METEOR
-> REMEDY
-----
Please press the command. (Return to album list: 1)
Enter 0 to go to the menu.
>>
```

다음으로는 메인 메뉴에서 '3'을 누르면 아티스트 검색으로 이동할 수 있다. 이번엔 알파벳 'b'만 검색하여 나오는 아티스트들을 확인해보았다.

```
>> 3
-----
Artist Search
-----
Enter the artist name you want to find.
Just press enter and the all artist will be shown.
>> b
-----
Artist List
No.          Artist Name
1             BTS
2             yourbeagle
3             BTOB
4             Block B
5             BLACKPICK
-----
Enter the 'No.' to see more about the artist.
Enter 0 to go to the menu.
>>
```

여기서도 해당하는 No.를 눌러 아티스트가 발매한 음원의 목록을 확인할 수 있다.

```
>> 5
-----
Artist info
1. Name: BLACKPICK

There is 2 music(s) in the playlist
Music / Album
-> Pretty Savage / THE ALBUM
-> Lovesick Girls / THE ALBUM

Please press the command. (Return to artist list: 1)
Enter 0 to go to the menu.
>>
```

다시 메인 메뉴에서 '4'를 누르면 플레이리스트 검색으로 이동할 수 있다. 이번에는 'asdf'를 검색해보았다.

```
>> 4
-----
Playlist Search
-----
Enter the playlist title you want to find.
Just press enter and the all playlist will be shown.
>> asdf

I couldn't find any playlist..
Press the enter key to return to the main menu.
>>
```

보이는 것과 같이 해당하는 목록이 없다면 출력해주지 않는다. 이는 음악 검색, 앨범 검색 등에서도 같은 결과를 보여준다. 이번엔 실제 목록에 존재하는 플레이리스트를 검색해서 들어간 모습이다.

```
-----
Playlist info
1. Title      : Show Me The Money
2. Creator    : EastSea_test1
3. Creation Date: 2020-11-25

There is 4 music(s) in the playlist
Music / Album
-> 원해 / 쇼미더머니 9 Episode 1 / Swings, Mckdaddy, Khakii, Layone
-> 원원 / 쇼미더머니 9 Episode 1 / 혀성현, dsel, kaogaii, Untell
-> VVS / 쇼미더머니 9 Episode 1 / 미란이, 면치맨, Khundi Panda, MUSHVENOM
-> Freak / 쇼미더머니 9 Episode 1 / 월보이, 원슈타인, Chillin Homie, 스카이민혁

Please press the command. (Play all music: 1, Return to playlist list: 2)
Enter 0 to go to the menu.
>>
```

'Playlist info'는 다음과 같이 생겼는데 여기서 '1'을 누르면 플레이리스트에 존재하는 모든 음원을 재생하게 된다. 즉, 플레이리스트에 존재하는 모든 음원의 조회수를 1씩 증가시킨다. 다음과 같은 문구가 뜨면 음악 재생에 성공한 것이다.

```
>> 1
~~~~~ Music play ~~~~~
Music has been played successfully!
Return to playlist info.
```

다음으로 메인 메뉴에서 '5'를 누르면 플레이리스트를 생성할 수 있다. 지금 접속한 계정은 방금 만든 계정이므로 마이 플레이리스트가 존재하지 않아 새로 만들어 보았다.

```
Please write the title of the playlist you want to make.
1. Title(less than 30 characters) >> Test6
1. Title : Test6
-> Is that true? (Yes:1, No:2) >> 1

Now, playlist creation is complete.
Do you want to add music in this playlist?
Please press the command. (Yes: 1, No(To menu): 0)
>>
```

플레이리스트 제목을 입력하여 플레이리스트 생성을 하면 바로 음악을 추가할 것인지 물어본다. 이때 '1'을 누르면 addMusicInPlaylist()함수가 실행되어 음악을 넣을 수 있다. 이 기능은 마이 플레이리스트에서도 할 수 있는 기능이라 여기서는 음악을 추가하지 않고 넘어갔다.

메인 화면에서 '6'을 눌러 마이 플레이리스트로 들어오면 아까 만든 플레이리스트가 목록으로 보여진다.

```
>> 6
-----
          My Playlist
  No.      Playlist Title
    1           Test6
-----
Enter the 'No.' to see more about the playlist.
And you can play the music or edit the playlist!
Enter 0 to go to the menu.
>>
```

Test6에 해당하는 No.를 눌러 'Playlist info'를 볼 수 있는데, 여기서 '1'을 누르면 이 플레이리스트 안에 해당하는 음악들을 재생할 수 있고, '2'를 누르면 플레이리스트를 수정할 수 있다. 현재 플레이리스트에 아무 음악도 없기 때문에 먼저 '2'를 눌러 수정을 진행해준다.

```
----- Playlist info
1. Title      : Test6
2. Creator    : Command_test6
3. Creation Date: 2020-12-02

There is 0 music(s) in the playlist

Please press the command. (Play all music: 1, Edit to Playlist: 2, Return to playlist list: 3)
Enter 0 to go to the menu.
>> 2

What do you want to do?
1. Rename playlist
2. Add music in playlist
3. Delete music in playlist
4. Delete playlist
0. Return to playlist info
>>
```

이 화면에선 '1'을 누르면 플레이리스트 이름을 변경할 수 있고, '2'를 누르면 음악을 추가할 수 있으며, '3'을 누르면 음악을 삭제할 수 있고, '4'를 누르면 플레이리스트를 삭제할 수 있다. 여기선 '2'를 눌러 플레이리스트에 음악을 추가해 보았다. Add Music 화면은 Music Search와 거의 동일하기 때문에 음악을 추가하는 부분만 사진으로 넣었다.

```
----- Music info
1. Title      : YESTERDAY
2. Artist     : Block B
3. Genre      : Dance
4. Lyricist: 박 경
5. Composer: 박 경
6. Arranger: 박 경
7. Hits       : 0
8. in Album: YESTERDAY

Please press the command. (Add to music: 1, Return to music list: 2)
Enter 0 to go to the menu.
>> 1

Music successfully added!
Return to music info.
```

음악을 추가하고 나면 다시 Music list(사진에선 music info로 오타가 났다)로 돌아가는데 여기서 음악을 계속 추가할 수 있다. 음악 추가가 끝난 뒤엔 '0'을 눌러 완료할 수 있다.

```
Enter the 'No.' to see more about the music.  
And you can add the music!  
Enter 0 to go to the menu.  
>> 0  
  
Playlist has been edited successfully!  
Press the enter key to return to the main menu.
```

메인 메뉴에서 다시 마이 플레이리스트로 들어가 아까 추가한 음악이 잘 추가되었는지 확인할 수 있다.

```
>> 1  
  
-----  
      Playlist info  
1. Title      : Test6  
2. Creator    : Command_test6  
3. Creation Date: 2020-12-02  
  
There is 3 music(s) in the playlist  
  Music / Album / Artist  
-> 이름 애게 / Palette / IU  
-> YESTERDAY / YESTERDAY / Block B  
-> My Lady / Brother Act. / BTOB  
  
Please press the command. (Play all music: 1, Edit to Playlist: 2, Return to playlist list: 3)  
Enter 0 to go to the menu.  
>> 1
```

다시 '2'를 눌러 수정화면으로 들어간 뒤 '3'을 눌러 음악을 삭제해보았다.

```
>> 2  
  
What do you want to do?  
1. Rename playlist  
2. Add music in playlist  
3. Delete music in playlist  
4. Delete playlist  
  
0. Return to playlist info  
>> 3  
  
-----  
      Music In Playlist  
No.          Music Title  
1            이름 애게  
2            YESTERDAY  
3            My Lady  
  
Enter the 'No.' to delete the music.  
Enter 0 to exit.  
>> 1  
  
Music has been played successfully!
```

음악을 삭제하면 아래 완료 문구가 뜬다.(여기서도 played가 아닌 deleted이다. 실제 프로그램에선 오타를 수정했다.)

다음으로 플레이리스트를 완전히 삭제해보았다.

```
-----  
      Playlist info  
1. Title      : Test6  
2. Creator    : Command_test6  
3. Creation Date: 2020-12-02  
  
There is 2 music(s) in the playlist  
  Music / Album / Artist  
-> YESTERDAY / YESTERDAY / Block B  
-> My Lady / Brother Act. / BTOB  
  
Please press the command. (Play all music: 1, Edit to Playlist: 2, Return to playlist list: 3)  
Enter 0 to go to the menu.  
>> 2  
  
What do you want to do?  
1. Rename playlist  
2. Add music in playlist  
3. Delete music in playlist  
4. Delete playlist  
  
0. Return to playlist info  
>> 4  
  
Are you sure you want to delete the playlist? (Yes: 1, No:2)  
>> 1  
  
Playlist successfully removed!  
  
Playlist has been edited successfully!  
Press the enter key to return to the main menu.
```

삭제가 된 뒤에는 플레이리스트가 없기 때문에 마이 플레이리스트가 비어있다고 나온다.

```

-----
      Menu
1. Music Search
2. Album Search    (Only search)
3. Artist Search   (Only search)
4. Playlist Search
5. Create Playlist
6. My Playlist
7. Top Music

0. Exit MusicApp
-----
>> 6

There is no playlist..
Press the enter key to return to the main menu.

```

마지막으로 메인 메뉴에서 '7'을 누르면 음원 순위를 볼 수 있다. 선택지가 3가지가 있지만 현재 DB내에 데이터가 많이 존재하지 않으므로 보기좋게 Top 10을 선택해보았다.

```

>> 7

-----
      Top Music
1. Top 10
2. Top 100
3. Top 200

Enter 0 to go to the menu.
>> 1

-----
      Music list
No. Music Title / Hits
1 Dynamite / 65
2 VVS / 43
3 METEOR / 36
4 LATATA / 34
5 빌 었 어 / 27
6 MAZE / 25
7 WANNABE / 25
8 우연히 봄 / 23
9 들어줘요 / 22
10 Lovesick Girls / 12

Enter the 'No.' to see more about the music.
And you can play the music!
Enter 0 to go to the menu.
>> 1

```

여기선 음악을 선택하여 그 음악의 'Music info'를 볼 수 있으며 재생도 가능하다. 현재 9위에 있는 '들어줘요'가 '우연히 봄'과 조회수가 1차이 나기 때문에 2번 재생하고 다시 확인해 보았다.

```

>> 1

~~~~ Music play ~~~~

Music has been played successfully!
Return to music info.

-----
      Music info
1. Title : 들어줘요
2. Artist : (G)-IDLE
3. Genre : Dance
4. Lyricist: MosPick
5. Composer: MosPick
6. Arranger: MosPick
7. Hits : 24
8. in Album: I am

Please press the command. (Play to music: 1, Return to music list: 2)
Enter 0 to go to the menu.
>> 2

-----
      Music list
No. Music Title / Hits
1 Dynamite / 65
2 VVS / 43
3 METEOR / 36
4 LATATA / 34
5 빌 었 어 / 27
6 MAZE / 25
7 WANNABE / 25
8 들어줘요 / 24
9 우연히 봄 / 23
10 Lovesick Girls / 12

Enter the 'No.' to see more about the music.
And you can play the music!
Enter 0 to go to the menu.
>> 1

```

음악을 재생한 뒤에 '2'를 눌러 리스트를 확인하면 8위와 9위가 바뀌어 있는걸 확인할 수 있다.

- 관리자 Menu 기능

이번엔 관리자 계정으로 로그인을 진행해보았다. 관리자 계정으로 접속하면 메인 메뉴 화면이 일반 유저와 차이가 있는데, 음원을 등록하고 삭제하는 기능과 유저를 관리하는 기능이 더 추가되어 있다.

```
-----
Log in
-----
ID >> 19980522
Password >> admin

Log in success! Hello~

-----
Admin Menu
1. Music Search      (Only search)
2. Album Search      (Only search)
3. Artist Search     (Only search)
4. Playlist Search
5. Create Playlist
6. My Playlist
7. Top Music

11. Music Registration
12. Album Registration
13. Artist Registration

21. Music Deletion
22. Album Deletion
23. Artist Deletion

31. User Management
0. Exit MusicApp
-----
>> [
```

음원을 등록하기 위해선 먼저 그 음원을 포함하고 있을 앨범이 있어야 하고, 앨범이 있으려면 그 앨범을 발매할 아티스트가 먼저 있어야 한다. 이미 존재하는 아티스트나 앨범에 음원을 추가하고 싶으면 '11', '12'를 눌러 바로 진행해도 되지만, 여기선 '13'을 눌러 새로 아티스트부터 추가해보았다. 아티스트는 이름만 입력하면 추가할 수 있다.

```
-----
Artist Registration
-----

Please write this form
1. Artist Name(less than 30 characters) >> test_artist

1. Artist Name: test_artist
-> Is that true? (Yes:1, No:2) >> 1

Now, artist creation is complete.
Go to menu..
```

이제 메인 메뉴에서 '12'를 눌러 앨범을 추가해보자. 앨범을 추가할 때 먼저 아티스트를 고를 수 있는데 ','를 활용하면 아티스트를 여러 명 선택할 수 있다. 여기서는 방금 만든 아티스트만 선택해서 앨범 만들기를 진행해 보았다. 앨범 타입을 선택하는 부분에선 3가지 종류로만 구분된다.

```
>> 12
-----
Album Registration
-----

Please enter the name of the artist create the album.
Just press enter and the all artist will be shown.
>> te

-----
Artist list
No.          Artist Title
1            Untell
2            test_artist

Please select an artist for the album. (Enter No.)
If there are many people, please separate the numbers by comma. (ex) 1,4,5
>> 2

Please write this form
1. Album Title(less than 30 characters) >> test_album
2. Album Type(Single/EP: 1, Studio album: 2, Album: all but 1,2) >> 1
3. Agency(less than 15 characters) >> tt
4. Release Company(less than 15 characters) >> tt
5. Release Date(ex) 2020-01-01) >> 2020-12-03
6. Genre(Split by slash. ex) Hip Hop/Rap/Ballad) >> Hip Hop

1. Album Title    : test_album
2. Album Type    : Single/EP
3. Agency        : tt
```

```

4. Release Company: tt
5. Release Date : 2020-12-03
5. Genre : Hip Hop
-> Is that true? (Yes:1, No:2) >> 1

Now, album creation is complete.
Go to menu..

```

다음으로 방금 만든 앨범에 노래를 등록해 보았다. 이 부분도 앨범 등록과 비슷한 과정을 거친다.

```

-----  

          Album list  

No.           Album Title  

  1             test_album  

-----  

Please select an album for the song. (Enter No.)  

Enter 0 to go to the menu.  

>> 1  

-----  

          Album info  

1. Title      : test_album  

2. Artist     : test_artist  

3. Genre      : Hip Hop  

4. Type       : Single/EP  

5. Agency     : tt  

6. Release Company: tt  

7. Release Date : 2020-12-03  

There is 0 music(s) in the album  

-----  

Please write this form  

1. Music Title(less than 30 characters) >> test_music  

2. Lyricist(less than 30 characters) >> th  

3. Composer(less than 30 characters) >> th  

4. Arranger(less than 30 characters) >> th  

5. Genre(Split by slash. ex) Hip Hop/Rap/Ballad) >> Rap  

1. Music Title: test_music  

2. Lyricist   : th  

3. Composer   : th  

4. Arranger   : th  

5. Genre      : Rap  

-> Is that true? (Yes:1, No:2) >> 1  

Now, music creation is complete.
Go to menu..

```

음원을 다 만든 뒤에 음악 검색에서 확인해보면 잘 등록되어 있는 것을 확인할 수 있다.

```

>> 1  

-----  

          Music Search  

-----  

Enter the music title you want to find.  

Just press enter and the all music will be shown.  

>> test  

-----  

          Music list  

No.           Music Title  

  1             test_music  

-----  

Enter the 'No.' to see more about the music.  

And you can play the music!  

Enter 0 to go to the menu.  

>> 1  

-----  

          Music info  

1. Title      : test_music  

2. Artist     : test_artist  

3. Genre      : Rap  

4. Lyricist: th  

5. Composer: th  

6. Arranger: th  

7. Hits       : 0  

8. in Album: test_album  

-----  

Please press the command. (Play to music: 1, Return to music list: 2)
Enter 0 to go to the menu.
>> 1

```

다음으로 음원을 삭제해 보았다. 이 부분의 인터페이스는 음악 검색, 앨범 검색, 아티스트 검색과 상당히 유사하다. 이러한 검색들과 다른 점은 info 화면에서 '1'을 누르면 삭제를 하는 기능이 추가되었다는 것이다. 음원 삭제만 하게 되면 앨범과 아티스트는 그대로 남고 음원과 모든 플레이리스트로부터 그 음원 정보가 제거된다. 앨범을 삭제하면 아티스트는 그대로 남지만, 그 앨범부터 앨범에 소속된 음원, 그리고 모든 플레이리스트로부터 그 음원 정보가 사라진다. 아티스트를

삭제하면 아티스트가 발매한 음원부터, 앨범 모두와 모든 플레이리스트로부터 그 음원에 대한 정보가 전부 사라진다. 즉, 아티스트를 삭제하는 것만으로도 방금까지 등록했던 모든 것을 삭제할 수 있어 여기서는 아티스트 삭제만 진행해보았다.

```
>> 23

-----
      Artist Deletion
-----
Please enter the name for the artist you want to delete.
Just press enter and the all artist will be shown.
>> test

-----
      Artist List
  No.          Artist Name
    1            test_artist

Enter the 'No.' to see more about the artist.
And you can delete the artist.
Enter 0 to go to the menu.
>> 1

-----
      Artist info
1. Name: test_artist

There is 1 music(s) in the playlist
  Music / Album
-> test_music / test_album

Please press the command. (Delete to artist: 1, Return to artist list: 2)
Enter 0 to go to the menu.
>> 1

Artist was successfully deleted.
Return to main menu.
```

잘 삭제되었는지 아티스트 검색과 음원 검색을 통해 확인해보았다.

```
>> 3

-----
      Artist Search
-----
Enter the artist name you want to find.
Just press enter and the all artist will be shown.
>> test

I couldn't find any artist..
Press the enter key to return to the main menu.
>> 1

-----
      Music Search
-----
Enter the music title you want to find.
Just press enter and the all music will be shown.
>> test

I couldn't find any songs..
Press the enter key to return to the main menu.
```

마지막으로 유저를 관리하는 기능이다. '31'을 눌러 검색하고 싶은 유저를 찾은 뒤 정보를 수정할 수 있다.

```
>> 31

-----
      User Management
-----
Please enter the user name.
Just press enter and the all user will be shown.
>> tes

-----
      User List
  No.          User Name
    1            EastSea_test1
    2            EastSea_test2
    3            EastSea_test3
    4            Java_in_test4
    5            Java_in_test5
    6            Command_test6

Enter the 'No.' to see more about the user.
Enter 0 to go to the menu.
>> 6

-----
      User info
1. User ID : 20101006
2. User PW : test6
3. User Name: Command_test6
```

```

4. Phone    : 010-1111-0006
5. SSN     : 101010-0000006
6. Address  : 6, Busan
7. Admin    : X

There is 0 playlist(s)
-----
Please press the command. (Edit to user: 1, Return to user list: 2)
Enter 0 to go to the menu.
>> 1

```

다음 화면에서 '1'을 눌러 edit 화면으로 들어온 뒤, 다시 한번 '1'을 눌러 유저의 신상정보를 변경해보았다.

```

>> 1

What do you want to do?
1. Revice user info
2. Delete user

0. Return to user info
>> 1

Please write this form.
1. Password(less than 15 characters) >> test6
2. Name(less than 30 characters) >> Revice_test6
3. Phone(less than 15 characters) >> 010-1111-0006
4. SSN(less than 15 characters) >> 101010-0000006
5. Address(less than 30 characters) >> 6, Seoul
6. isAdmin/Admin: 1, Not admin: all but 1) >> 2

1. ID      : 20101006
2. Password: test6
3. Name    : Revice_test6
4. Phone   : 010-1111-0006
5. SSN    : 101010-0000006
6. Address : 6, Seoul
7. Admin   : X
-> Is that true? (Yes:1, No:2) >> 1

User has been edited successfully!
Press the enter key to return to the main menu.

```

ID과 주소가 잘 바뀌었는지 확인해보았다.

```

>> 31
-----
----- User Management -----
----- Please enter the user name.
Just press enter and the all user will be shown.
>> re

----- User List -----
No.           User Name
1             Revice_test6
----- Enter the 'No.' to see more about the user.
Enter 0 to go to the menu.
>> 1

----- User info -----
1. User ID  : 20101006
2. User PW   : test6
3. User Name: Revice_test6
4. Phone     : 010-1111-0006
5. SSN      : 101010-0000006
6. Address   : 6, Seoul
7. Admin     : X

There is 0 playlist(s)
-----
Please press the command. (Edit to user: 1, Return to user list: 2)
Enter 0 to go to the menu.
>> 1

```

그리고 이 화면에서 다시 edit를 누르고 '2'를 눌러 유저를 삭제해보았다.

```

----- User info -----
1. User ID  : 20101006
2. User PW   : test6
3. User Name: Revice_test6
4. Phone     : 010-1111-0006
5. SSN      : 101010-0000006
6. Address   : 6, Seoul
7. Admin     : X

There is 0 playlist(s)
-----
Please press the command. (Edit to user: 1, Return to user list: 2)
Enter 0 to go to the menu.
>> 1

What do you want to do?
1. Revice user info
2. Delete user

```

```

0. Return to user info
>> 2

Are you sure you want to delete this user? (Yes: 1, No:2)
>> 1

Playlist successfully removed!

User has been edited successfully!
Press the enter key to return to the main menu.

```

삭제한 유저로 로그인을 시도하면 로그인이 되지 않는 것을 확인할 수 있다.

```

----- Menu -----
1. Log in
2. Create account

0. Exit MusicApp
----->> 1
----- Log in -----
ID >> 20101006
Password >> test6
[ERROR] Log in faild!

```

6. 사용되는 SQL 문 명세

생략된 부분도 존재하지만 '4. 프로그램 코드에 대한 자세한 설명'에서 각각의 SQL 문이 어디서 사용되었는지는 어느정도 설명했기 때문에, 여기선 단순히 사용한 SQL 문만 나열해 보았다. CREATE 구문을 이용하여 테이블을 만들었던 부분은 '3. DB 제작'에 명세 되어있기 때문에 여기서는 따로 언급하진 않았고, 중복해서 사용된 SQL 문 또한 생략하였다.

- SELECT

```
89     String sql = "select * from USER where User_id = ? AND User_pw = ?";
```

: USER 테이블에서, 입력받은 id와 pw를 통해 해당하는 유저를 찾아준다.

```
130     String sql = "select MAX(User_id) from USER";
```

```
1177    String sql = "select MAX(Playlist_id) from PLAYLIST";
```

```
2148    sql = "select MAX(Music_id) from MUSIC";
```

```
2318    sql = "select MAX(Album_id) from ALBUM";
```

```
2434    String sql = "select MAX(Artist_id) from ARTIST";
```

: 해당하는 테이블에서, primary key값인 '_id'의 최대값을 출력해준다.

```
475     String sql = "select * from MUSIC where LOWER(Music_title) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
```

```
655     String sql = "select * from ALBUM where LOWER(Album_title) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
```

```
830     String sql = "select * from Artist where LOWER(Artist_name) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
```

```
962     String sql = "select * from PLAYLIST where LOWER(Playlist_title) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
```

```
3235    String sql = "select * from User where LOWER(User_name) LIKE LOWER(?)" // 제목 대소문자 구분없이 검색 및 포함으로 검색
```

: 해당하는 테이블에서, 입력받은 값을 대소문자 구분없이 비교했을 때, 그 값이 '_title'이나 '_name'에 포함되어 있다면 그 음원들을 모두 출력해준다.

```

1052           sql = "select Artist_name "
1053             + "from MUSIC, MUSIC_MAKE, ARTIST "
1054             + "where Music_id = ? AND MM_Music_idx = Music_id AND MM_Artist_idx = Artist_id";
540           sql = "select Album_title, Artist_name "
541             + "from MUSIC, MUSIC_MAKE, ARTIST, ALBUM "
542             + "where Music_id = ? AND Music_id = MM_Music_idx AND MM_Artist_idx = Artist_id AND Album_id = ?";

```

: MUSIC, MUSIC_MAKE, ARTIST의 조인을 통해 음원을 제작한 아티스트를 찾아주고, 아래 SQL 문

은 추가적으로 그 음원에 해당하는 앨범을 연결하였다.

```
718     sql = "select Artist_name "
719         + "from ALBUM, ALBUM_MAKE, ARTIST "
720         + "where Album_id = ? AND Album_id = AM_Album_idx AND AM_Artist_idx = Artist_id";
```

: ALBUM, ALBUM_MAKE, ARTIST의 조인을 통해 앨범을 제작한 아티스트를 찾아준다.

```
1037 l = "select Music_id, Music_title, Album_title "
1038     + "from PLAYLIST, MUSIC_IN_LIST, MUSIC, ALBUM "
1039     + "where Playlist_id = ? AND MIL_Playlist_idx = Playlist_id AND MIL_Music_idx = Music_id AND Album_idx = Album_id";
```

: PLAYLIST, MUSIC_IN_LIST, MUSIC의 조인을 통해 플레이리스트에 들어있는 음악들을 찾아주고, 그 음원에 해당하는 앨범을 연결하였다.

```
889     sql = "select Music_title, Album_title "
890         + "from ARTIST, MUSIC_MAKE, MUSIC, ALBUM "
891         + "where Artist_id = ? AND Artist_id = MM_Artist_idx AND MM_Music_idx = Music_id AND Album_idx = Album_id";
```

: ARTIST, MUSIC_MAKE, MUSIC의 조인을 통해 아티스트가 발매한 음원을 찾아주고, 그 음원에 해당하는 앨범을 연결하였다.

```
556     sql = "select MGenre "
557         + "from MUSIC_GENRE "
558         + "where MIndex = ?";
746     sql = "select AGenre "
747         + "from ALBUM_GENRE "
748         + "where AIndex = ?";
```

: 위의 두 SQL 문은 해당하는 테이블(음원 및 앨범)의 장르를 찾는 구문이다.

```
732     sql = "select Music_title "
733         + "from MUSIC "
734         + "where Album_idx = ?";
```

: 찾고자 하는 앨범에 들어있는 음원의 이름들을 출력해준다.

```
1023     sql = "select User_name "
1024         + "from PLAYLIST, USER "
1025         + "where Playlist_id = ? AND User_idx = User_id";
```

: 플레이리스트를 제작한 유저의 이름을 출력해준다..

```
1230     String sql = "select * from PLAYLIST where User_idx = ?";
```

: 해당하는 유저가 만든 플레이리스트들을 출력해준다.

```
1482     String sql = "select * from MUSIC order by Hits_num desc";
```

: MUSIC 테이블에 있는 모든 값을 출력해주는데 Hits_num의 내림차순으로 출력해준다.

- INSERT

```
209     sql = "insert into USER values (?,?,?,?,?,?)";
```

: 유저에 해당하는 정보를 USER 테이블에 추가해준다.

```
2206     sql = "insert into MUSIC values (?,?,?,?,?,?)";
```

```
2220     sql = "insert into MUSIC_GENRE values (?,?)";
```

```
2230     sql = "insert into MUSIC_MAKE values (?,?)";
```

: 음원에 해당하는 정보를 MUSIC, MUSIC_GENRE, MUSIC_MAKE 테이블에 추가해준다.

```
2385     sql = "insert into ALBUM values (?,?,?,?,?,?)";
```

```
2398     sql = "insert into ALBUM_GENRE values (?,?)";
```

```
2408     sql = "insert into ALBUM_MAKE values (?,?)";
```

: 앨범에 해당하는 정보를 ALBUM, ALBUM_GENRE, ALBUM_MAKE 테이블에 추가해준다.

```
2474     sql = "insert into ARTIST values (?,?)";
```

: 아티스트에 해당하는 정보를 ARTIST 테이블에 추가해준다.

```
1849             sql = "insert into MUSIC_IN_LIST " // 음악 중복 검사
1850                 + "select ?,? "
1851                     + "from DUAL "
1852                     + "where NOT EXISTS ( "
1853                         + " select * "
1854                             + " from MUSIC_IN_LIST "
1855                             + " where MIL_Playlist_idx = ? AND MIL_Music_idx = ?)";
```

: 전체적인 의미는 MUSIC_IN_LIST에 값을 넣는 구문인데, 중복된 값을 넣지 않기 위해 DUAL 테이블과 EXISTS를 이용했다. 추가하고자 하는 음원이 이미 그 플레이리스트에 존재한다면 NOT EXISTS로 나온 결과가 없을 것이고, 따라서 값을 추가해주지 않는다. 추가하고자 하는 음원이 현재 플레이리스트에 존재하지 않으면 NOT EXISTS로 그 값이 나오고, 그 값을 추가해주는 것이다.

- DELETE

```
1916     String sql = "delete from MUSIC_IN_LIST "
1917           + "where MIL_Playlist_idx = ? AND MIL_Music_idx = ?";
```

: 특정한 튜플 하나를 MUSIC_IN_LIST에서 삭제한다.

```
1958     String sql = "delete from MUSIC_IN_LIST where MIL_Playlist_idx = ?";
```

: 특정한 플레이리스트에 속하는 튜플들을 MUSIC_IN_LIST에서 삭제한다.

```
1965     sql = "delete from PLAYLIST where Playlist_id = ?";
```

: 특정한 플레이리스트 하나를 PLAYLIST에서 삭제한다.

```
2657     sql = "delete from MUSIC_GENRE where mIndex = ?";
```

```
2664     sql = "delete from MUSIC_MAKE where MM_Music_idx = ?";
```

```
2671     sql = "delete from MUSIC_IN_LIST where MIL_Music_idx = ?";
```

```
2678     sql = "delete from MUSIC where Music_id = ?";
```

: 특정한 음원 하나에 대해 장르, MAKE, PLATLIST와의 연결을 삭제하고, 마지막으로 음원도 삭제한다.

```
2871     sql = "delete from ALBUM_GENRE where aIndex = ?";
```

```
2878     sql = "delete from ALBUM_MAKE where AM_Album_idx = ?";
```

```
2916     sql = "delete from ALBUM where Album_id = ?";
```

: 특정한 앨범 하나에 대해 장르, MAKE를 삭제하고, 마지막으로 앨범도 삭제한다.

```
3078     sql = "delete from MUSIC_MAKE where MM_Artist_idx = ?";
```

```
3085     sql = "delete from ALBUM_MAKE where AM_Artist_idx = ?";
```

```
3182     sql = "delete from ARTIST where Artist_id = ?";
```

: 특정한 아티스트 하나에 대해 MUSIC_MAKE, ALBUM_MAKE를 삭제하고, 마지막으로 아티스트도 삭제한다.

```
3507     String sql = "delete from User where User_id = ?";
```

: 특정한 유저 하나를 삭제한다.

- UPDATE

```
613     sql = "update MUSIC set Hits_num = ? where Music_id = ?";
```

```
1107     sql = "update MUSIC set Hits_num = Hits_num + 1 where Music_id = ?";
```

: 위의 두 SQL 문은 프로그램 내에서 해당하는 음원의 조회수를 1 증가시키는 역할을 한다.

```
3440     String sql = "update USER "
3441           + "set User_pw = ?, User_name = ?, Phone = ?, Ssn = ?, Address = ?, Is_admin = ? where User_id = ?";
```

: 유저의 속성들을 해당하는 값들로 업데이트한다.

```
1686     String sql = "update PLAYLIST set Playlist_title = ? where Playlist_id = ?";
```

: 업데이트하고자 하는 PLAYLIST에 있는 Playlist_title을 해당하는 값으로 바꾼다.