**Database System**

**<음악 플레이 어플리케이션 프로그램>**

Project # 4 : DBMS 프로그램 개발

김상욱 교수님

2015004284 강태희

2017 / 12/ 5

1. *Class 설명*

**<Model>**// 각 table의 객체화를 위한 java class

* Music – num, genre, artist, album, trnum, likenum
* User – num, isadmin, id, pw, addr, name, plist
* Artist – num, name
* Album – num, name, artist
* Playlist – num, user
* MusicInPlaylist – musicnum, plistnum

**<Controller>**

// 각 테이블 객체에 대한 insert, delete, selet, update 쿼리문 실행을 위한 클래스

* BaseController
* open(), close() : 쿼리문 실행시에 자주 쓰이는 connection의 open, close
* initialized() : 프로그램 처음 실행시에 DBConstants 클래스에 저장해두었던 쿼리문들을 순차적으로 실행하는 함수. 테이블을 만들고 테스트케이스들을 insert한다.
* MusicController

Music 객체를 이용하여 mysql DB의 music table에 insert, delete, select, update하는 함수들의 클래스이다.

질의문이 포함된 함수를 미리 구현해 놓음으로써 메인함수에서 코드의 중복으로 코드가 길어지는 것을 방지하고 의미파악이 쉽도록 하였다.

<다양한 select 함수의 구현>

* select(String name, String artist) : 음악 이름, 아티스트 이름으로 음악 select
* selectByNum(**int** num): Music 테이블의 primary key인 num으로 음악 select
* <T> ArrayList<Music> selectBy(String attr, T value) : 특정 attribute가 value 값을 가지는 음악들을 Arraylist로 반환
* ArrayList<Music> selectLike(String str): 음악의 attribute 값중에 str이 포함되어 있는 음악을 SELECT LIKE 쿼리로 구현
* ArrayList<Music> selectAll() : 전체 음악을 ArrayList로 반환
* UserController

User 객체를 이용하여 mysql DB의 user table에 insert, delete, select, update하는 함수들의 클래스이다.

MusicDB와 마찬가지로 insert, delete, update그리고 다양한 select함수가 구현되어 있다.

* ArtistController

Artist 객체를 이용하여 mysql DB의 artist table에 insert, delete, select, update하는 함수들의 클래스이다.

MusicDB와 마찬가지로 insert, delete, update그리고 다양한 select함수가 구현되어 있다.

* AlbumController

Album 객체를 이용하여 mysql DB의 album table에 insert, delete, select, update하는 함수들의 클래스이다.

MusicDB와 마찬가지로 insert, delete, update그리고 다양한 select함수가 구현되어 있다.

* PlaylistController

Playlist 객체를 이용하여 mysql DB의 Playlist table에 insert, delete, select, update하는 함수들의 클래스이다.

MusicDB와 마찬가지로 insert, delete, update그리고 다양한 select함수가 구현되어 있다.

* MusicInPlaylistController

MusicInPlaylist 객체를 이용하여 mysql DB의 MusicInPlaylist table에 insert, delete, select, 하는 함수들의 클래스이다. 이 테이블의 경우 튜플의 update하는 상황이 필요하지 않기 때문에 update함수는 따로 만들지 않았다.

Select함수의 경우 MusicInPlaylist 튜플의 음악이나 소유주로 검색할수 있어야 하므로 attribute에서 value 값을 찾는

<T> ArrayList<MusicInPlaylist> select(String attr, T value) 함수를 구현하였다.

**<기타Class>**

* MJMain  
  : 처음 연결정보를 입력하고 연결되면 MUSICJAVA 메인화면이 출력된다. 메인화면에서 로그인을 하여 유저 대시보드, 어드민 대시보드로 연결된다. 회원가입도 가능하다.
* AdminDashboard  
  메인화면에서 관리자모드로 로그인 하면 이 클래스의 printDashboard함수로 어드민 페이지 메뉴가 출력된다. 이곳에서 음악과 유저의 Insert, Update, Delete, Select가 가능하다.
* UserDashboard  
  메인화면에서 유저모드로 로그인 하면 이 클래스의 printDashboard함수로 유저 메인페이지 메뉴가 출력된다. 이곳에서 유저는 플레이리스트에 음악을 추가하고 플레이리스트 보기, 음악, 아티스트, 앨범, 장르별로 음악목록 검색, 내 정보 보기 의 기능을 수행할수 있다.
* DBConstants   
  : Mysql database 연결 정보, 구성할 테이블, 테이블들의 애트리뷰트들, 프로그램 처음에 실행될 테이블 create 쿼리문들, 그리고 초기 테스트케이스 insert 쿼리문들이 static final String 로 저장되어있는 클래스.
* Genre  
  : 장르 리스트가 String 배열로 저장되어있다.

1. *Query*  
   ->MUSIC테이블 쿼리문을 기반으로 다른 테이블의 쿼리문을 작성하였기 때문에 MUSIC table에 사용한 쿼리문을 설명하겠습니다.

|  |
| --- |
| String sql = **"INSERT INTO "** + DBConstants.Table.***MUSIC*** + **" VALUES (NULL,?,?,?,?,?,0)"**; **if** (!open()) **return false**; **try** {  **pstmt** = **conn**.prepareStatement(sql);  **pstmt**.setString(1, music.**name**);  **pstmt**.setString(2, music.**genre**);  **pstmt**.setString(3, music.**album**);  **pstmt**.setString(4, music.**artist**);  **pstmt**.setInt(5, music.**trnum**);  **pstmt**.executeUpdate();  close();  **return true**; } |

**1)insert**

Insert문으로 Music table에 value를 넣는다. 첫번째 인자는 MUSICNUM인데 primary key이고 auto increment로 설정해놓았기 때문에 null로 넣어주었고 맨마지막 playlist num은 playlist생성후에 넣어주는 것으로 처리해놓았기 때문에 우선 0으로 넣어주었다.

|  |
| --- |
| String sql = String.*format*(**"delete from %s where %s = '%s' and %s = '%s'"**,  DBConstants.Table.***MUSIC***,  DBConstants.Column.***MUSIC\_NAME***, music.**name**,  DBConstants.Column.***MUSIC\_ARTIST***, music.**artist**); **if** (!open()) **return false**; **try** {  **stmt** = **conn**.createStatement();  **stmt**.executeUpdate(sql);  close();  **return true**; } |

**2) delete**

delete문으로 musicName과 musicArtist가 같은 music을 찾아 Music table에서 삭제한다.

|  |
| --- |
| String sql = String.*format*(**"update %s set %s = '%s' where %s = '%d'"**,  DBConstants.Table.***MUSIC***,  attr, String.*valueOf*(value),  DBConstants.Column.***MUSIC\_NUM***, music.**num**);  **if** (!open()) **return false**;  **try** {  **stmt** = **conn**.createStatement();  **stmt**.executeUpdate(sql);  close();  **return true**; } |

**3) update**

Update문으로 primarykey로 튜플을 찾아내어 attr(애트리뷰트)를 value값으로 변경한다.  
**4) select**

|  |
| --- |
| **1.select**  String sql = String.*format*(**"select \* from %s where %s ='%s' and %s = '%s'"**,  DBConstants.Table.***MUSIC***,  DBConstants.Column.***MUSIC\_NAME***, name,  DBConstants.Column.***MUSIC\_ARTIST***, artist);  **2.selectByNum**  String sql = String.*format*(**"select \* from %s where %s ='%d'"**,  DBConstants.Table.***MUSIC***,  DBConstants.Column.***MUSIC\_NUM***, num);  **3.selectBy**  String sql = String.*format*(**"select \* from %s where %s ='%s'"**,  DBConstants.Table.***MUSIC***,  attr, String.*valueOf*(value));  **4.selectLike**  String sql = String.*format*(**"select \* from %s where %s LIKE '%%%s%%' or %s LIKE '%%%s%%' or %s LIKE '%%%s%%'"**,  DBConstants.Table.***MUSIC***,  DBConstants.Column.***MUSIC\_NAME***, str,  DBConstants.Column.***MUSIC\_ARTIST***, str,  DBConstants.Column.***MUSIC\_ALBUM***, str);  **5.selectAll**  String sql = String.*format*(**"select \* from %s"**,  DBConstants.Table.***MUSIC***); |

Selecta문으로 5가지 종류의 select를 만들었다.

1번 셀렉트는 musicname과 musicartist를 비교하여 찾는 셀렉트 문이다.

2번 셀렉트는 music의 primary key인 music num으로 찾는 셀렉트 문이다.

3번 셀렉트는 비교하고자 하는 애트리뷰트와 그 값을 비교하여 찾는 셀렉트 문이다.

4번 셀렉트는 애트리뷰트값들 중에 해당 스트링이 포함되는 값을 가지는 튜플들을 모두 찾는 셀렉트 문이다.

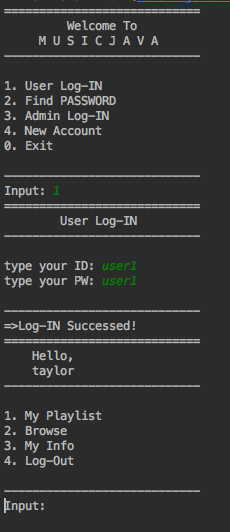
5번 셀렉트는 모든 튜플들을 셀렉트하는 쿼리이다.

|  |
| --- |
| **public static final** String ***CREATE\_TABLE\_MUSIC*** =  **"create table "** + Table.***MUSIC*** + **" ( "** + Column.***MUSIC\_NUM*** + **" int(11) primary key auto\_increment, "** + Column.***MUSIC\_NAME*** + **" varchar(32) not null, "** + Column.***MUSIC\_GENRE*** + **" varchar(16) not null, "** + Column.***MUSIC\_ALBUM*** + **" varchar(32) not null, "** + Column.***MUSIC\_ARTIST*** + **" varchar(32) not null, "** + Column.***MUSIC\_TRACK*** + **" int(11) not null, "** + Column.***MUSIC\_LIKENUM*** + **" int(11) not null )"**; |

**5)create**

create문으로 7가지 애트리뷰트를 가지는 music table을 만드는 쿼리이다.

1. 실행 화면  
   *<User 모드>*

**application의 처음 메인화면입니다.**

메뉴는 1.User모드로 로그인

2.패스워드 찾기

3.Admin모드로 로그인

4.회원가입

5. 프로그램 종료 가 있습니다.

User모드로 로그인하기 위해서 ID와 PW를 입력하고 성공시에

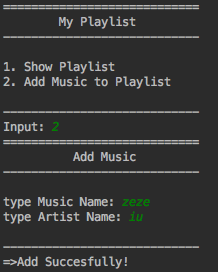
**User의 Name이 출력되면서 UserDashboard가 열립니다.**

메뉴는 1.나의 플레이리스트관리

2.음악목록 검색

3.나의 정보보기

4.로그아웃 이 있습니다.

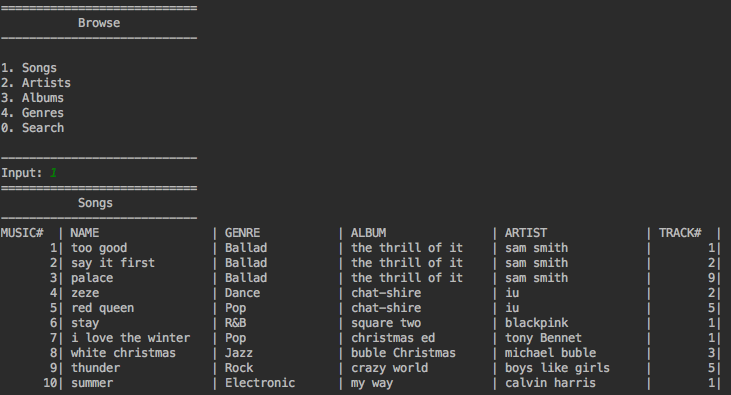
**My Playlist 메뉴로 들어간 화면입니다.**

메뉴는 1.현재 나의 플레이리스트에 담겨있는 곡 보기

2.나의 플레이리스트에 음악 추가

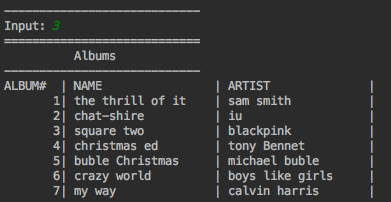
음악 추가 메뉴를 선택하면

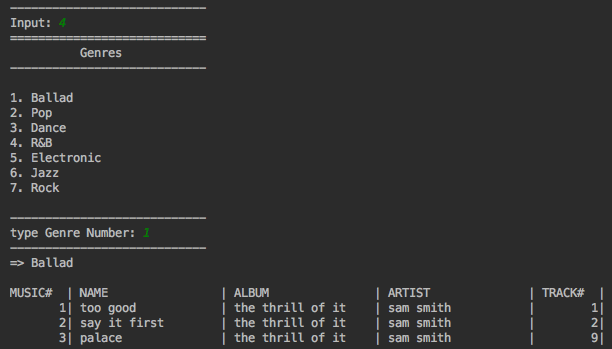
음악 이름과 artist이름을 검색하여 추가 할 수 있습니다.

 아래는 1번 메뉴를 선택하여 플레이리스트에 담긴 **음악들의 리스트를 보여주는 화면** 입니다.

Browse 메뉴로 들어간 화면입니다.

Songs 음악 리스트, Artists 아티스트 리스트, Albums 앨범 리스트 를 선택하여 볼 수 있습니다.



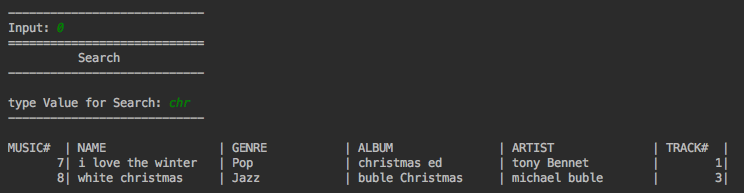


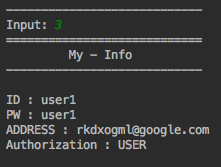
장르 메뉴를 선택하면

장르 리스트가 나오고

장르 번호를 선택하면

해당 장르에 속하는 음악리스트를 볼 수 있습니다.

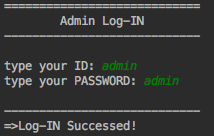
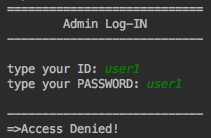
**0번 search메뉴**에서는 모든 음악 목록에서 서치키워드를 검색합니다.

UseDashboard에서 **3번 My-Info 메뉴**를 선택하면

회원가입시에 적었던 나의 정보를 볼 수 있습니다.

Admin 권한을 가진 계정의 경우

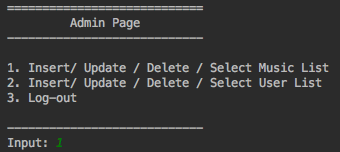
Authorization 이 admin로 설정되어 있습니다.

*******<Admin 모드>*

**Admin Log-In**에서는

admin권한을 가진 User만이

로그인 할 수 있습니다.

로그인에 성공하면

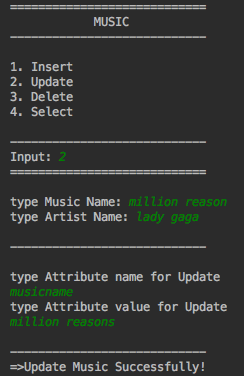
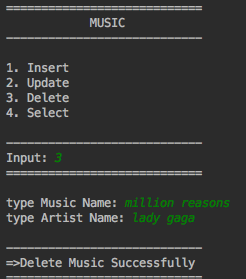
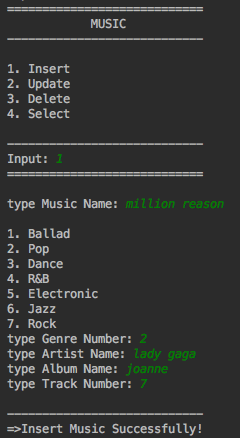
**AdminDashboard** 가 뜨고

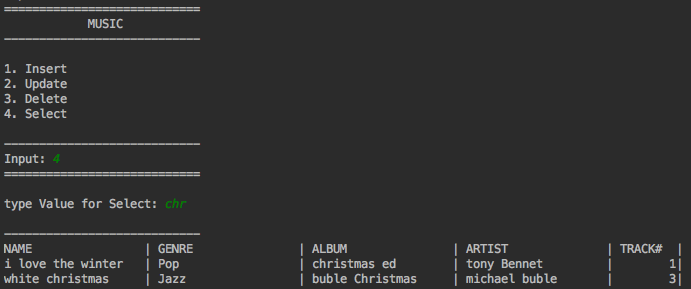
메뉴는

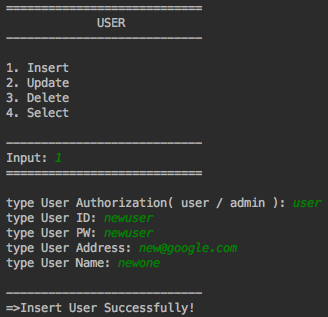
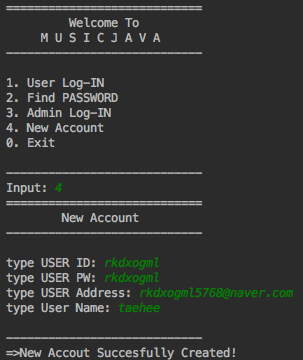
1.음악리스트의 Insert/Update/Delete/Select

2.유저리스트의 Insert/Update/Delete/Select

**1번메뉴의 실행 화면입니다**. (유저리스트 관리도 비슷한 모습입니다.)







**회원가입하는 화면입니다**.