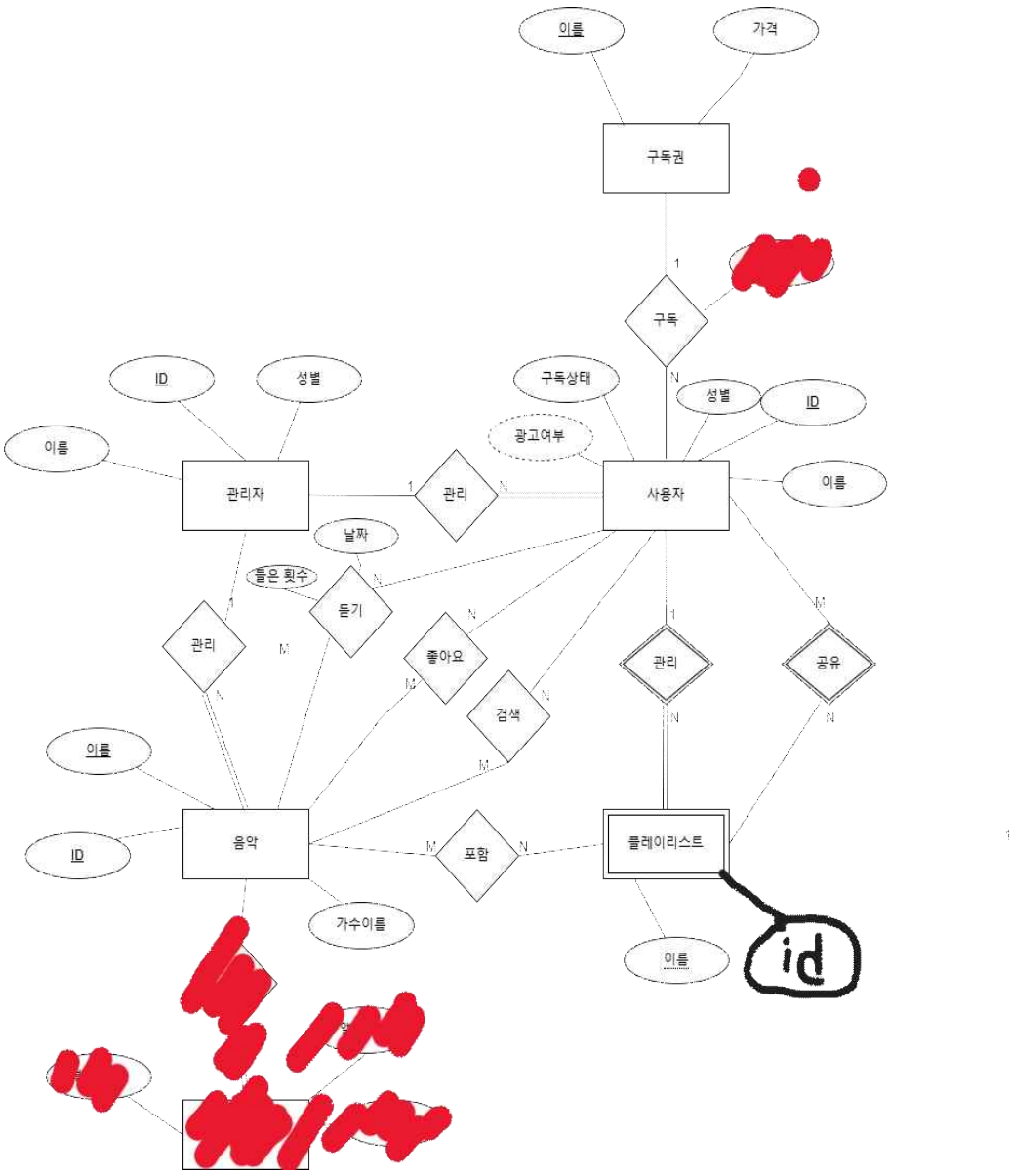
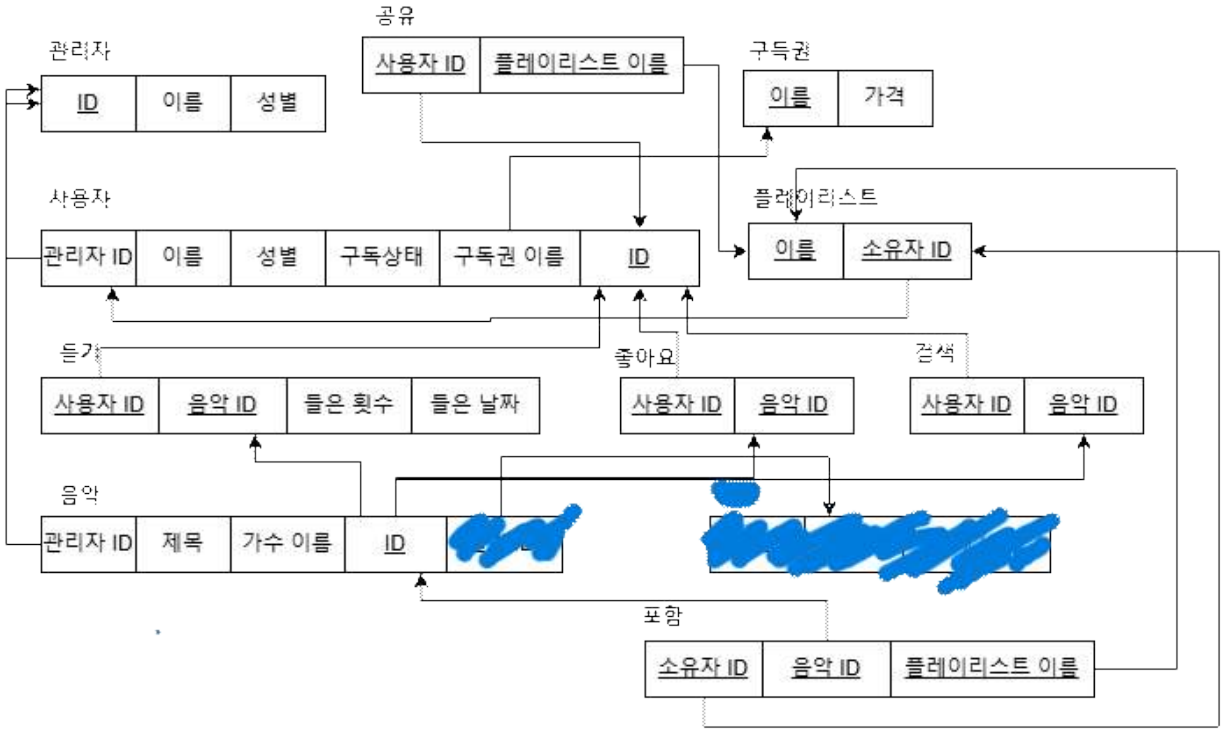
Final 프로젝트: DBMS 프로그램 개발
2022031994 김준수

수정사항(노트북을 바꾸면서 E-R diagram과 relational model 파일들이 날아가서 전에 제출했던 리포트의 사진을 이용한 점 양해 부탁드립니다!):
- 플레이리스트 table의 primary key를 name으로 설정했었는데 ID라는 attribute를 추가하고 이 ID를 primary key로 설정한다. 다른 사용자마다 같은 플레이리스트 이름이 존재할 수 있기 때문이다.
- 기능을 구현하는데 필요없다고 생각되어 앨범 table을 없앴다. 그리고 music table의 albumID attribute도 없애주었다.




공유
사용자 ID
플레이리스트 이름
관리자
ID
이름
성별
구독권
이름
가격
사용자
관리자 ID
이름
성별
구독상태
구독권 이름
ID
플레이리스트
이름
소유자 ID
사용자 ID
음악 ID
들은 횟수
들은 날짜
좋아요
사용자 ID
음악 ID
검색
사용자 ID
음악 ID
음악
관리자 ID
제목
가수 이름
ID
포함
소유자 ID
음악 ID
플레이리스트 이름

- adminmenu()

관리자가 1명 존재한다고 가정했고 DB에 관리자 ID와 이름, 성별이 들어가 있는 상태이다. 관리자 table인 Adminstrator에서 ID를 가져와 Managemusic과 ManageUser에 인자로 넣어주었다.

```
def adminmenu():
    sql="SELECT ID FROM administrator"
    cursor.execute(sql)
    ID=cursor.fetchall()
    connection.commit()
    while True:
        print()
        print("Administrator mode!!")
        print('0. Return to previous menu')
        print('1. Manage music')
        print('2. Manage user')
        x=int(input("Select option: "))
        if x==0:
            break
        elif x==1:
            Managemusic(ID)
        elif x==2:
            ManageUser(ID)
        else:
            print("Error: input correct menu")
```

ID	name	sex
1	master	M

```
Administrator mode!!
0. Return to previous menu
1. Manage music
2. Manage user
Select option:
```

음악과 사용자를 관리할 수 있는 관리자 메뉴이다.

- Managemusic()

```
0. Return to previous menu
1. Enroll music
2. Delete music
3. Show music list
Select option:
```

음악을 관리하는 메뉴이다. 음악을 등록하거나 삭제할 수 있고 등록한 음악들의 list를 볼 수 있다.

- EnrollMusic(id)

```
def EnrollMusic(id):
    #처음 음악 등록할 때는 들은 횟수와 들은 날짜는 없으므로 default값과 null
    값으로 설정해준다.
    sql='INSERT INTO music(adminID, MID, Mname, singer) VALUES (%s, %s,
    %s, %s)'
    a=input('ID: ')
    b=input('title: ')
    c=input('singer: ')
    cursor.execute(sql,(id,a,b,c))
    connection.commit()
    sql='SELECT Mname FROM music WHERE MID= %s'
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "enrolled!")
    print()
```

관리자 id를 인자로 받고 음악을 등록해준다. 인자로 받은 관리자 ID(foreign key)와 입력받은 음악 ID, 음악 제목, 가수를 music table에 삽입해준다.

```
0. Return to previous menu
1. Enroll music
2. Delete music
3. Show music list
Select option: 1
ID: 1
title: blue
singer: bigbang
blue enrolled!
```

- DeleteMusic()

```
def DeleteMusic():
    sql='SELECT Mname FROM music WHERE MID=%s'
    a=input("music ID to delete: ")
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "deleted!")
    connection.commit()

    #heard table에서 삭제
    sql = 'DELETE FROM heard WHERE MID=%s'
    cursor.execute(sql, a)

    #including 테이블에서 삭제
    sql = 'DELETE FROM including WHERE MID=%s'
    cursor.execute(sql, a)

    #music 테이블에서 삭제
    sql = 'DELETE FROM music WHERE MID=%s'
    cursor.execute(sql, a)

    #favorite 테이블에서 삭제
    sql = 'DELETE FROM favorite WHERE MID=%s'
    cursor.execute(sql, a)

    #search table에서 삭제
    sql='DELETE FROM search WHERE MID=%s'
    cursor.execute(sql,a)

    connection.commit()
```

DELETE 구문을 이용해 입력받은 ID인 음악을 테이블에서 삭제해준다. music 테이블에서 삭제해주고 music ID를 foreign key 값으로 가지고 있는 heard(music-user), including(music-playlist), favorite(music-user), search(music-user) 테이블에서도 삭제시켜준다.

```
0. Return to previous menu
1. Enroll music
2. Delete music
3. Show music list
Select option: 2
music ID to delete: 1
blue deleted!
```

- ShowmusicList()

```
sql="Select * from music"      #music table 출력
cursor.execute(sql)
result=cursor.fetchall()
headers=['admin ID', 'Music ID', 'Music name', 'Singer']
print(tabulate(result,headers,tablefmt='grid'))
connection.commit()
```

SELECT * 구문을 써서 music 테이블에 있는 모든 값들을 출력하였다. 파이썬의 tabulate 모듈을 사용하였다.

```
0. Return to previous menu
1. Enroll music
2. Delete music
3. Show music list
Select option: 3
+------------+------------+--------------+-----------+
|   admin ID |   Music ID | Music name   | Singer    |
+============+============+==============+===========+
|          1 |          1 | blue         | bigbang   |
+------------+------------+--------------+-----------+
|          1 |          2 | apt          | rose      |
+------------+------------+--------------+-----------+
|          1 |          3 | love dive    | izone     |
+------------+------------+--------------+-----------+
```

- ManageUser()

```
0. Return to previous menu
1. Register User
2. Delete User
3. Show user list
Select option:
```

User들을 관리하는 메뉴이다. user를 등록하거나 삭제할 수 있고 등록된 user들의 list를 볼 수 있다.

- RegisterUser()

```python
def RegisterUser(id):
    #adminID는 관리자 한 명이므로 그 ID를 받아와서 넣어줌, 유저 처음 등록할때는 구독 안된 상
    sql="INSERT INTO user(adminID, UID,password,name,sex) VALUES(%s, %s, %s, %s, %s)"
    a=input('ID: ')
    b=input('password: ')
    c=input('Name: ')
    d=input('Sex: ')
    # d=input('subStatus: ')
    # e=input('subName: ')

    cursor.execute(sql,(id,a,b,c,d))
    connection.commit()
    sql="SELECT name FROM user WHERE UID=%s"
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "register done")

    connection.commit()
```

INSERT INTO user(adminID, UID, password,name,sex) VALUES (%s,%s,%s,%s,%s) 구문으로 user 테이블에 입력받은 값들을 삽입해주었다. (foreign key인 관리자 ID)
ID와 비밀번호, 이름, 성별 등을 입력받는다. 처음 등록할 때는 구독을 안 한 상태이므로 default값으로 구독권 상태는 'N'(No)으로, 구독권 이름은 null로 설정해주었다.

```
0. Return to previous menu
1. Register User
2. Delete User
3. Show user list
Select option: 1
ID: 1
password: 1
Name: john
Sex: f
john register done
```

- DeleteUser()

```
def DeleteUser(id):

    sql='SELECT name FROM user WHERE UID=%s'
    a=input("user ID to delete: ")
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "deleted!")
    connection.commit()

    #user table에서 삭제
    sql='DELETE FROM user WHERE UID=%s'
    cursor.execute(sql,a)

    #heard table에서 삭제
    sql='DELETE FROM heard WHERE UID=%s'
    cursor.execute(sql,a)

    #including table에서 삭제
    sql='DELETE FROM including WHERE UID=%s'
    cursor.execute(sql,a)

    # #like table에서 삭제
    sql='DELETE FROM favorite WHERE UID=%s'
    cursor.execute(sql,a)

    #playlist table에서 삭제

    sql='DELETE FROM playlist WHERE MgrID=%s'
    cursor.execute(sql,a)

    #search table에서 삭제
    sql='DELETE FROM search WHERE UID=%s'
    cursor.execute(sql,a)
```

DELETE FROM 구문을 이용해 입력받은 user ID에 해당하는 user를 삭제해주었다. user테이블과 relation을 가진 다른 테이블에서도 모두 삭제를 해주었다. (주석에 있는 like table이 favorite table이다!)

```
0. Return to previous menu
1. Register User
2. Delete User
3. Show user list
Select option: 2
user ID to delete: 1
john deleted!
```

- ShowUser()

```
sql="Select * from user"
cursor.execute(sql)
result=cursor.fetchall()
headers=['admin ID', 'User ID', 'User password','User name', 'Sex', 'subStatus', 'subName']

print(tabulate(result,headers,tablefmt='grid'))
print()
connection.commit()
```

SELECT * FROM 구문을 이용해 user에 있는 모든 값들을 출력하였다.

```
0. Return to previous menu
1. Register User
2. Delete User
3. Show user list
Select option: 3
+------------+-----------+-----------------+-------------+-------+-------------+-----------+
|   admin ID |   User ID |   User password | User name   | Sex   | subStatus   | subName   |
+============+===========+=================+=============+=======+=============+===========+
|          1 |         1 |               1 | john        | m     | N           |           |
+------------+-----------+-----------------+-------------+-------+-------------+-----------+
|          1 |         2 |               2 | bob         | f     | N           |           |
+------------+-----------+-----------------+-------------+-------+-------------+-----------+
|          1 |         3 |               3 | ava         | f     | N           |           |
+------------+-----------+-----------------+-------------+-------+-------------+-----------+
```

- usermenu()

```
User mode!!
0. Return to previous menu
1. Sign up
2. Log in
Select option:
```

usermenu로 들어가면 회원가입 옵션과 로그인 옵션이 있다.

- Signup()

```
def Signup():
    sql="SELECT ID FROM administrator"
    cursor.execute(sql)
    ID=cursor.fetchall()
    connection.commit()
    sql="INSERT INTO user(adminID, UID,password, name,sex) VALUES(%s, %s, %s, %s, %s)"
    a=input('ID: ')
    b=input('Password: ')
    c=input('Name: ')
    d=input('Sex: ')
    # d=input('subStatus: ')
    # e=input('subName: ')

    cursor.execute(sql,(ID,a,b,c,d))
    connection.commit()
    sql="SELECT name FROM user WHERE UID=%s"
    cursor.execute(sql,a)
    rows = cursor.fetchall()

    for row in rows:
        print(row[0], "sign up completed!")

    connection.commit()
    print()
```

INSERT INTO user 구문을 이용해 입력받은 ID와 비밀번호 이름, 성별을 user 테이블에 삽입해준다. (관리자가 user를 등록해줄 때와 같음)

```
User mode!!
0. Return to previous menu
1. Sign up
2. Log in
Select option: 1
ID: 4
Password: 4
Name: emily
Sex: f
emily sign up completed!
```

- Login()

```
def Login():
    print("Login")

    sql='SELECT name,UID FROM user WHERE UID=%s and password=%s'
    a=input("ID: ")
    b=input("password: ")

    cursor.execute(sql,(a,b))
    rows = cursor.fetchall()

    for row in rows:
        name=row[0]
        id=row[1]
        print("log in completed!")
```

SELECT name, UID FROM user WHERE UID=%s, password=%s 구문을 이용해 ID와 비밀번호를 입력받고 user 테이블에서 입력받은 name과 UID 값을 가져왔다. 만약 회원가입이 되지 않은 ID와 비밀번호를 입력해서 login을 시도하는 경우 다시 입력받도록 하였다.

-Login menu()

```
User mode!!
0. Return to previous menu
1. Sign up
2. Log in
Select option: 2
Login
ID: 1
password: 1
log in completed!

john nice to meet you!
0. log out
1. Manage playlist
2. Show music list
3. Play Music
4. Subscribe
5. Show recently listen music list
6. Show mostly heard music
7. Search Music
8. Like music
9. Show Liked music list
select option:
```

login 해서 들어가면 선택할 수 있는 옵션들이 많이 있다.

- ManagePlaylist()

```
0. Return to previous menu
1. Create playlist
2. Delete playlist
3. Show my playlists
4. Show all playlists
5. Add music to playlist
6. Delete music from playlist
7. Show Music in Playlist
select option:
```

1번 option인 ManagePlaylist를 선택해서 플레이리스트들을 관리할 수 있다.

- CreatePlaylist()

```
def CreatePlaylist(id):

    sql='INSERT INTO playlist (Pname, MgrID, PID) VALUES (%s, %s, %s)'
    a=input("Input playlist name: ")
    b=input('Input playlist ID: ')
    cursor.execute(sql,(a,id,b))
    connection.commit()
    sql='SELECT Pname FROM playlist WHERE PID= %s'
    cursor.execute(sql,b)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "created!")
    print()
```

INSERT INTO playlist 구문을 이용하여 입력받은 playlist 이름과 playlist ID를 playlist 테이블에 삽입해주었다. foreign key로서 playlist를 만든 사람의 ID를 삽입해주었다.

```
select option: 1
Input playlist name: summer
Input playlist ID: 1
summer created!
```

- DeletePlaylist()

```
def DeletePlaylist(id):

    sql='SELECT Pname FROM playlist WHERE PID=%s'
    a=input("Playlist ID to delete: ")
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "deleted!")
    connection.commit()

    sql='DELETE FROM playlist WHERE PID=%s'
    cursor.execute(sql,a)
    connection.commit()

    sql='DELETE FROM including WHERE PID=%s'
    cursor.execute(sql,a)
    connection.commit()
```

DELTE FROM playlist 구문을 이용하여 삭제할 playlist ID를 입력받아 playlist에서 삭제해주고 playlist에 담긴 음악을 관리하는 including 테이블(playlist가 음악을 포함하고 있다고 생각하여 'including'이라고 이름 지었다.)에서도 삭제해주었다.

```
select option: 2
Playlist ID to delete: 1
summer deleted!
```

- ShowMyPlaylist()

SELECT Pname FROM playlist WHERE MgrID=%s 구문을 이용하여 현재 로그인 한 user가 만든 playlist들만 출력한다.

```
def ShowMyPlayslist(id):

    sql="Select Pname from playlist WHERE MgrID=%s"
    cursor.execute(sql,id)
    result=cursor.fetchall()
    headers=['Playlistname']
    print(tabulate(result,headers,tablefmt='grid'))
    connection.commit()
```

```
select option: 3
+----------------+
| Playlistname   |
+================+
| summer         |
+----------------+
| shower         |
+----------------+
| night          |
+----------------+
```

- ShowAllPlaylist()

```
#플레이리스트 공유
def ShowAllPlaylist(id):
    sql="Select playlist.Pname, user.name,playlist.PID from user,playlist WHERE user.UID=playlist.MgrID"
        #music table 출력
    cursor.execute(sql)
    result=cursor.fetchall()
    headers=['Playlistname','Owner name','Playlist ID']
    print(tabulate(result,headers,tablefmt='grid'))
    connection.commit()
```

SELECT playlist.Pname, user.name, playlist.PID FROM user,playlist WHERE user.UID=playlist.MgrId 구문을 이용해 user의 primary key와 이 primary key를 foreign key로서 가지고 있는 playlist의 MgrID가 같을 때의 특정 값들을 출력하였다. 각각의 user가 만든 플레이리스트들 공유하여 모두가 볼 수 있다.

```
select option: 4
+----------------+--------------+---------------+
| Playlistname   | Owner name   |   Playlist ID |
+================+==============+===============+
| summer         | john         |             1 |
+----------------+--------------+---------------+
| shower         | john         |             2 |
+----------------+--------------+---------------+
| night          | john         |             3 |
+----------------+--------------+---------------+
| summer         | bob          |             4 |
+----------------+--------------+---------------+
| sad            | bob          |             5 |
+----------------+--------------+---------------+
| fitness        | ava          |             6 |
+----------------+--------------+---------------+
```

- AddMusicToPlaylist()

```
def AddMusicToPlaylist(id):
    sql='INSERT INTO including VALUES (%s, %s, %s)'
    a=input('Music ID to add to playlist: ')
    b=input('Playlist Id: ')
    cursor.execute(sql,(id,a,b))
    connection.commit()
    sql='SELECT Mname FROM music WHERE MID= %s'
    cursor.execute(sql,a)
    rows = cursor.fetchall()
    for row in rows:
        print(row[0], "inserted!")
    print()
```

INSERT INTO including VALUES (%s, %s, %s) 구문을 이용하여 입력받은 music ID를 입력받은 playlist ID에 해당하는 playlist에 삽입해주었다.

```
select option: 5
Music ID to add to playlist: 1
Playlist Id: 1
blue inserted!
```

- ShowMusicInPlaylist()

```
def ShowMusicInPlaylist(id):
    sql='''SELECT music.Mname from music,including
    WHERE including.UID=%s and including.PID=%s and music.MID=including.MID'''
    a=input('Input Playlist ID to show music: ')
    cursor.execute(sql,(id,a))
    result=cursor.fetchall()
    headers=['Music name']
    print(tabulate(result,headers,tablefmt='grid'))
    connection.commit()
```

playlist 안에 있는 음악들을 보여주는 함수이다. 보고 싶은 playlist의 ID를 입력받는다. WHERE UID=%s and including.PID and music.MID=including.MID를 통해 현재 로그인 한 사용자가 입력받은 playlistID에 해당하는 playlist에 있는 음악들을 보여준다.

```
select option: 7
Input Playlist ID to show music: 1
+--------------+
| Music name   |
+==============+
| blue         |
+--------------+
```

- DeleteMusicFromPlaylist()

```
def DeleteMusicToPlaylist(id):
    sql='SELECT Mname FROM music WHERE MID=%s'
    a=input("Music ID to delete: ")
    b=input('Playlist ID to delete: ')
    cursor.execute(sql,a)
    rows = cursor.fetchall()

    for row in rows:
        print(row[0], "deleted!")
    connection.commit()

    #현재 로그인 한 user가 만든 플레이리스트에 한해서만 음악 삭제
    sql='DELETE FROM including WHERE UID=%s and MID=%s and PID=%s'
    cursor.execute(sql,(id,a,b))
    connection.commit()
```

현재 로그인 한 user가 만든 playlist에 한해서만 음악을 삭제한다. 삭제할 music ID와 그 음악이 들어있는 playlist ID를 입력받아 해당하는 음악을 삭제한다.

```
select option: 6
Music ID to delete: 1
Playlist ID to delete: 1
blue deleted!
```

- ShowMusicList()

```
sql="Select * from music"     #music table 출력
cursor.execute(sql)
result=cursor.fetchall()

headers=['admin ID', 'Music ID', 'Music name', 'Singer']
print(tabulate(result,headers,tablefmt='grid'))
connection.commit()
```

SELECT * 구문을 통해 music table에서 모든 tuple들을 가져온다.

```
select option: 2
+------------+------------+--------------+----------+
|   admin ID |   Music ID | Music name   | Singer   |
+============+============+==============+==========+
|          1 |          1 | blue         | bigbang  |
+------------+------------+--------------+----------+
|          1 |          2 | apt          | rose     |
+------------+------------+--------------+----------+
|          1 |          3 | love dive    | izone    |
+------------+------------+--------------+----------+
```

-PlayMusic()

```
def PlayMusic(id):
    global count

    while True:
        sql='SELECT MID,Mname from music WHERE MID=%s'
        a=input('Input music ID: ')
        cursor.execute(sql,a)
        connection.commit()
        rows=cursor.fetchall()

        if rows:
            #primary key가 중복일때(이미 들은 음악일 때), 들은 횟수와 들은 날짜
            sql='''INSERT INTO heard VALUES (%s,%s,%s,%s)
            ON DUPLICATE KEY UPDATE heardNum=heardNum+1, heardDate=%s'''

            cursor.execute(sql,(id,a,count,datetime.now(),datetime.now()))
            MID = cursor.fetchall()
            for MID in rows:
                print("Playing ",MID[1])
            connection.commit()
            return
        else:
            print("There is no music!")
```

SELECT로 music에서 music ID를 select해서 입력받은 ID에 해당하는 음악이 있는지 확인한다. 만약 음악이 있다면 heard(music과 user의 relation) table에 현재 로그인한 user의 ID와 입력받은 음악 ID를 삽입해준다. 음악을 들을 때마다 들은 횟수와 들은 날짜를 update 해준다. 여기서 heard table에서는 User ID와 music ID가 primary key인데 만약 같은 음악을 여러 번 듣는다면 중복이 발생한다. 따라서 ON DUPLICATE KEY UPDATE를 이용하여 중복된 값이 나오면 update를 해주는 구문을 사용하였다.

```
select option: 3
Input music ID: 1
Playing  blue
```

- Subscription()

```
#구독을 시작하면 user table의 구독상태와 구독권의 이름이 update 됨
def Subscription(id):
    sql='SELECT * from membership'
    cursor.execute(sql)
    result=cursor.fetchall()
    sname=result[0][0]
    connection.commit()
    sql='UPDATE user SET subStatus=%s, subName=%s WHERE UID=%s'
    cursor.execute(sql,('Y',sname,id))
    print("Subcription complete of",sname,"memebership")
    connection.commit()
```

구독을 하는 함수로 사용자는 구독을 하여 membership에 가입할 수 있다. 처음 회원가입했을 때는 default값으로 N이었던 구독상태를 나타내는 subStatus와 default값이 null이었던 구독권 이름을 나타내는 subName이 update 된다. 현재 membership은 1개만 있다고 가정한다. membership table에 있는 membership 이름을 select하여 user table에 update 해준다. 'UPDATE user SET subStatus=%s, subName=%s WHERE UID=%s'

Mname	price
premium	10,000

```
select option: 4
Subcription complete of premium memebership
```

- ShowRecentlyListenMusic()

```
def ShowRecentlyListenMusic(id):
    sql='''SELECT music.Mname,music.singer,heard.heardDate from heard,music
     WHERE UID=%s and music.MID=heard.MID ORDER BY heardDate DESC LIMIT 3'''
    cursor.execute(sql,id)
    result=cursor.fetchall()
    headers=['Music name', 'Singer','Heard Date']
    print(tabulate(result,headers,tablefmt='grid'))

    connection.commit()
```

최근에 본 music list를 보여주는 함수이다. 현재 로그인 한 사용자가 들은 음악들만 보여준다. 또한 music의 primary key인 MID는 heard의 foreign key이므로 서로 같을 때인 MID를 select해준다. 최근 들은 날짜 순으로 정렬(ORDER BY)해서 최대 3개(LIMIT 3)까지만 보여주도록 하였다. 'SELECT music.Mname, music.singer, heard.heardDate FROM heard, music WHERE UID=%s and music.MID=heard.MID ORDER BY heardDate DESC LIMIT 3'

```
select option: 5
+--------------+----------+----------------------+
| Music name   | Singer   | Heard Date           |
+==============+==========+======================+
| apt          | rose     | 2024-12-07 10:57:15  |
+--------------+----------+----------------------+
| blue         | bigbang  | 2024-12-07 10:52:00  |
+--------------+----------+----------------------+
| love dive    | izone    | 2024-12-06 07:35:16  |
+--------------+----------+----------------------+
```

- SearchMusic()

```python
def SearchMusic(id):
    sql='SELECT MID,Mname,singer FROM music WHERE Mname=%s'
    print('Search music by name')
    name=input('Input Music name: ')
    cursor.execute(sql,name)
    result=cursor.fetchall()
    tmp=result[0][0]
    headers=['Music ID', 'Music name', 'Singer']
    print(tabulate(result,headers,tablefmt='grid'))
    sql='INSERT INTO search VALUES (%s, %s)'
    cursor.execute(sql,(id,tmp))

    connection.commit()
```

음악을 검색하는 함수이다. 입력받은 음악 이름을 사용하여 검색한다. 입력받은 이름에 해당하는 음악을 music table에서 찾아서 id와 이름, 가수 등을 보여준다.
그리고 INESRT INTO 구문을 이용해 search table에 음악을 검색한 사용자 ID와 검색한 음악 ID를 삽입해주었다.

```
select option: 7
Search music by name
Input Music name: blue
+------------+--------------+----------+
|   Music ID | Music name   | Singer   |
+============+==============+==========+
|          1 | blue         | bigbang  |
+------------+--------------+----------+
```

- LikeMusic()

```
#좋아요 표시를 한 음악 모아두는 table
def LikeMusic(id):
    sql_check_music='SELECT Mname FROM music WHERE MID=%s'
    a=input('Input Music ID to like: ')
    cursor.execute(sql_check_music,a)

    music=cursor.fetchone()
    if not music:
        print("Invalid music ID. Enter a valid ID")
        return

    sql_check_like = 'SELECT * FROM favorite WHERE UID = %s AND MID = %s'
    cursor.execute(sql_check_like, (id, a))
    already_liked = cursor.fetchone()

    if already_liked:
        print("You already liked the music:",music[0])
        return

    sql='INSERT INTO favorite (UID,MID) VALUES (%s, %s)'
    cursor.execute(sql,(id,a))
    connection.commit()
    print("Successfully liked the music:",music[0])

    connection.commit()
```

음악에 좋아요 표시를 해주는 함수이다. 좋아요 표시를 할 음악 ID를 입력받는다. INSERT INTO 문을 이용해 좋아하는 음악들을 모아둔 favorite table에 해당 음악을 삽입하여준다.

```
select option: 8
Input Music ID to like: 2
Successfully liked the music: apt
```

- ShowLikedMusic()

```
def ShowLikedMusic(id):
    sql='''SELECT music.Mname,music.singer from favorite,music
    WHERE UID=%s and music.MID=favorite.MID'''
    cursor.execute(sql,id)
    result=cursor.fetchall()
    headers=['Music name', 'Singer']
    print(tabulate(result,headers,tablefmt='grid'))

    connection.commit()
```

좋아요 표시를 해둔 음악들의 list를 출력해주는 함수이다. 현재 로그인 한 user가 좋아요 표시를 한 음악들만 보여주고 music table의 primary key인 MID는 favorite의 foreign key이므로 둘이 같을 때 music의 이름과 가수들을 select해준다.

```
9. Show Liked music list
select option: 9
+--------------+----------+
| Music name   | Singer   |
+==============+==========+
| apt          | rose     |
+--------------+----------+
```

-ShowMostHeardMusic()

```
def ShowMostHeardMusic(id):
    sql='SELECT music.MID,music.Mname,music.singer,heard.heardNum FROM
    heard,music WHERE heardNum=(SELECT MAX(heardNum) FROM heard WHERE
    UID=%s) and music.MID=heard.MID and UID=%s'
    cursor.execute(sql,(id,id))
    result=cursor.fetchall()
    headers=['Music ID','Music name', 'Singer','Heard number']
    print(tabulate(result,headers,tablefmt='grid'))


    connection.commit()
```

가장 많이 들은 음악을 보여주는 함수이다. 중첩문을 사용해서 먼저 현재 로그인한 user가 가장 많이 들은 음악의 횟수만 선택한다. 그리고 music의 primary key는 heard의 foreign key이므로 둘이 같을 때 music의 이름과 music Id, music singer, 들은 횟수를 선택해준다. 사용자마다 각자가 가장 많이 들은 음악을 볼 수 있다.

```
select option: 6
+------------+--------------+----------+----------------+
|   Music ID | Music name   | Singer   |   Heard number |
+============+==============+==========+================+
|          1 | blue         | bigbang  |              3 |
+------------+--------------+----------+----------------+
```