PROJECT#4 보고서

2016024957 컴퓨터소프트웨어학부 이원석

구현한 기능

1. 유저 OR 관리자 인터페이스 외부에서 지원하는 기능

A. 로그인 기능 – DB에 회원 정보를 저장하여 ID, Password를 확인 후 로그인 시 인터페이스로 이동.

B. 회원가입 기능 – 유저에 한하여, 계정을 만들 수 있음. 이때 비밀번호는 틀리면 안되므로 1차, 2차 입력으로 총 2번 입력받아야함.

2. 유저 인터페이스

A. 곡 검색기능 : 이후 검색한 곡을 바로 플레이할 수 있음.

B. 앨범 검색기능

C. 가수 검색기능 : 이후 가수의 곡 목록을 보여주고 추가적으로 가수의 이력 확인가능.

D. 모든 곡 보여주기 : DB안의 곡을 모두 보여줌

E. 플레이리스트 :

로그인한 유저의 플레이리스트 목록을 보여주고, 목록 중 원하는 플레이리스트를 선택하면 해당 리스트 안의 곡들을 자세히 보여줌. 리스트로 들어간 뒤에 해당 리스트에 곡을 추가하거나, 제거할 수 있음.

F. 인기순위 보여주기 : 모든 유저의 스트리밍 정보를 취합하여 재생순서가 많은

순서대로 차례로 곡을 보여줌. (재생횟수 0인 곡은 리스트에 제외)

G. 작곡가 검색기능 -> 이후 작곡가의 곡 목록을 보여주고 추가적으로 이력 확인가능.

H. 종료

***유저에게 지원하는 기능에는, 상대적으로 입력이 쉽도록, 특정 query에 대한 입력은 id number를 직접 입력하는 것이 아니라, 주어진 표에서 index를 이용하여 입력할 수 있도록 했습니다.**

3. 관리자 인터페이스

A. 유저생성 : 유저 회원가입 시 사용한 create_user() 함수를 재 사용했습니다.

B. 유저삭제 : 유저와 관련된 플레이리스트 및 스트리밍 기록 삭제 후 유저삭제

C. 음원등록 (새로운 앨범이 필요한 경우, 등록하는 과정에서 앨범을 등록할 수 있음)

D. 음원삭제

(해당 음원의 스트리밍 기록 및 해당 음원을 가지고있는 플레이리스트에서도 음원을 삭제)

E. 모든 유저 목록을 출력합니다.

F. 유저 정보 변경 : 계정의 고유 번호인 ID를 제외한 정보를 변경할 수 있습니다.

G. 종료

H.

강조사항

1. **유저 인터페이스와 관리자 인터페이스 간 가장 큰 차이는**, 관리자 기능은 특정 관리자의 기록을 따로 남기진 않는데 반해, 유저가 이용할 수 있는 기능들은, 해당 기능을 구현해주는 함수의 인자로서, 로그인 시 자동으로 입력받은 유저의 ID 번호를 이용한다는 것 입니다. 이를 통해 곡 재생, 리스트 생성과 같은 UPDATE나 INSERT DELETE 기능을 사용하는 경우, 추가적인 명령 없이 DB에 기록할 수 있게끔 하는 것입니다.

2. ID 넘버링이 필요한 테이블의 경우, 생성시 따로 ID번호를 할당하는 것이 아닌, IDENTITY 명령으로 생성시에 자동적으로 (기존의 ID값 +1) 으로 값이 할당되도록 했습니다.

3. 등록날짜 column의 경우, 파이썬의 datetime 모듈을 이용하여 현재 날짜,시간을 반영하였습니다.

4. 목록에서 음원이나 아티스트 및 앨범을 검색하는 경우는, ('SONG')_Exist 와 같은 임시변수를 이용하여, 1차적으로 SQL문으로 값의 유무를 확인 후, 없는 값일 경우 에러메시지 출력 후, 재입력을 받도록 하였습니다.

5. 기존에 설계단계에서도, 같은 곡을 서로 다른 앨범에 쓰일 수 있다는 것을 고려하여 설계를 했는데, 실제로 DB에 곡과 앨범을 집어넣을 때 에도, 이런 경우가 있었습니다. 해당 경우, M:N relation으로 처리하였으며, 서로 다른 앨범에 있더라도 해당 곡이 같은 곡임을 알 수 있습니다.

6. 제 DB에는 없지만, 곡의 이름이 중복될 경우를 대비하여, 곡을 플레이리스트에 넣거나, 재생하기 전, 검색 시 해당 이름을 가진 곡을 모두 보여주어, 이 index를 이용해 편하게 query를 보낼 수 있도록 하였습니다.

변동사항 (RELATIONAL SCHEMA에서의 변동사항)

1. 기존 Relational model에서 플레이리스트에 FOREIGN KEY 화살표를 누락했는데, 구현 시 FOREIGN KEY 적용을 하였습니다.
2. 앨범의 TITLE곡의 화살표도 누락된 것을 확인 하였습니다.

3. 사용자에 유저담당자 COLUMN의 경우, 불필요한 부분이라 생각되어 제거했습니다. 음원의 등록담당자 또한 제거했습니다.
4. 작곡가의 경우, DB에는 데이터가 있으나, 따로 작곡가를 검색하는 경우를 제외하면, 필요성이 크게 느껴지지 않는데다 음원 정보에 작곡가 정보까지 넣으면 너무 COLUMN이 많았습니다. 이에 활용하지 않기로 결정하여, 작곡가는 음원의 COLUMN에 넣지 않고, 단순 검색만을 구현했습니다.

아래는 실제 코드를 바탕으로, 실행결과와, 함수에 대한 자세한 설명입니다.

1. 구현한 함수 목록

```
def admin_menu(): #관리자 인터페이스…
def manage_user():#유저 정보 변경…
def Show_userlist(): # 모든 유저정보를 출력…
def Delete_user(): # 특정 유저를 제거…
def Register_music(): # 새로운 음원 등록…
def Delete_music(): # 음원 제거…
def search_artist_admin(): # 아티스트 검색 (관리자 전용)…
def user_menu(user_id): # 유저 인터페이스…
def search_song(user_id): #곡 검색 이후, 곡의 id 값을 반환하는 함수…
def search_and_play_song(user_id): # for viewing list and playing music…
def search_album(user_id): # 앨범 검색…
def search_artist(user_id): # 아티스트 검색…
def search_composer(user_id): # 작곡가 검색…
def play_music(user_id,song_id,song_name): # 음원 재생…
def create_playlist(user_id): # 새로운 플레이리스트 생성…
def view_playlist(user_id): # 플레이리스트 목록을 보여줌…
def view_detail(user_id,list_id): # 특정 플레이리스트의 소장 곡들을 모두 자세히 보여줌…
def delete_playlist(user_id,list_id): # 플레이리스트를 제거함…
def add_music(user_id,playlist_id): # 플레이리스트에 음원 추가…
def delete_from_list(user_id,list_id,song_id): # 플레이리스트에서 곡 삭제…
def show_all(user_id): # DB 내 모든 음원을 보여줌…
def show_ranking(user_id): # 각 음원들을 플레이 횟수순으로 나열 (인기차트)…
def create_user(): # USER 회원가입…
def start(): # 프로그램 실행 시 제일 처음 실행되는 함수…
```

함수의 대략적인 역할은, 주석에 적혀있습니다.!!

2. 시작창

```
def start():
    user_type=int(input("Choose usertype\n1. User \n2. Admin\n3. Exit : "))
    print("\n")
    if user_type==1:
        start_command=int(input("1. Create a new Acccount \n2. Login : "))
        if start_command==1:
            create_user()
            start()
        else:
            user_id=input('User ID : ')
            user_password=input('Password: ')
            sql='select knickname, user_id,user_password from user'
            cursor.execute(sql)
            resultset=cursor.fetchall()
            correct=0
            id_number=0
            for row in resultset:
              #  print(row[0],row[1])
                if(user_id==row[0] and user_password==row[2]):
                    correct=1
                    id_number=row[1]
                    break
            if correct==1:
                user_menu(id_number)
            else:
                print('User ID or Password is not correct.\nPlease check again\n\n')
                start()

    elif user_type==2:
        correct=0
        admin_id=int(input('Admin ID : '))
        admin_password=input('Password: ')
        sql='select admin_id, admin_password, admin_name from admin'
        cursor.execute(sql)
        resultset=cursor.fetchall()
        for row in resultset:
            if(admin_id==row[0] and admin_password==row[1]):
                correct=1
                name=row[2]
        if correct==1:
            print("Welcome %s" %name)
            admin_menu()
        else:
            print('Admin ID or Password is not correct.\nPlease check again\n\n')
            start()
    else:
        print('Bye')
        return
```

- 시작 시에 유저 타입을 선택합니다. 1번 2번 각 일반 유저와 관리자이며, 일반 유저의 경우, 계정이 없을경우 회원가입을 할 수 있으며 이는 아래에 있습니다.
- 로그인 후에는 일반 유저의 경우, 유저 전용 인터페이스로, 관리자는 관리자 전용 인터페이스로 이동합니다.

3. 유저생성 / 회원가입 함수

```
def create_user():
    ID=input("input ID : ")
    password1=input("Input Password : ")
    password2=input("Input Password again : ")
    if password1!=password2:
        print("Oops!! Password not Match..\n")
        create_user()
    else:
        name=input("Input name : ")
        Sex=input("Input sex : ")
        phone=input("Input phone number : ")
        birth = input("Input birth date : ")
        sql='insert into user(knickname, user_password, user_name, sex, phone_number, birth_date) values (%s,%s,%s,%s,%s,%s)'
        cursor.execute(sql,(ID,password1, name, Sex, phone, birth))
        conn.commit()
        print("Account has successfully created.\nLogin now!!")
```

회원가입 시 입력하는ID는 계정의 ID number가 아닌, 일반적으로 쓰이는 ID와 같습니다. Ex)뽀v대v도적

비밀번호를 확인하는데, 일치하지 않을 시, 다시 회원가입 시작 창으로 넘어갑니다.

개인정보를 입력받은 후, 각 변수에 저장해 둔뒤 SQL문을 활용해 INSERT INTO UNSER에 인자로 전달합니다.

```
PS C:\Users\이원석\python work> & C:/Users/이원석
Choose usertype
1. User
2. Admin
3. Exit : 1


1. Create a new Acccount
2. Login : 1
input ID : TEST_ID
Input Password : 1234
Input Password again : 1234
Input name : TEST_NAME
Input sex : M
Input phone number : 010-1234-5678
Input birth date : 2020-12-07
Account has successfully created.
Login now!!
Choose usertype
1. User
2. Admin
3. Exit : 1


1. Create a new Acccount
2. Login : 2
User ID : TEST_ID
Password: 1234
1. Search Song
2. Search Album
3. Search Artist
4. Show all song list
5. Go to my Playlist
6. Show ranking chart
Any other key to terminate :
```

1~3 과정의 실행 결과입니다.

4. 위 결과를 반영한 DB를 확인 가능.

Knickname	user_id	user_password	user_na
pupafafa	50,006	1234	kim
TEST_ID	50,007	1234	TEST_N

5. 곡 검색부분

```
def search_and_play_song(user_id): # for viewing list and playing music
    name=input('input name of Song to search : ')
    sql='select song_name from song where %s=song_name'
    cursor.execute(sql,(name))
    find_song_album=cursor.fetchall()
    song_album_exist=0
    song_id=[]
    for row in find_song_album:
        if(name==row[0]):
            song_album_exist=1
            #UPPER(컬럼명) LIKE UPPER(검색명)
            sql="select ar.artist_name,s.song_name, s.play_time, al.album_name, s.song_id from song as s,
            cursor.execute(sql,(name))
            resultset=cursor.fetchall()
            for row in resultset:
                song_id.append(row[4])
            Result=DataFrame(resultset,columns=['Artist','Music','Playtime','Album','ID'])
            print(Result)

            break

    if song_album_exist==0:
        print("Not founed..  \nCheck the capital again \'%s\'"%name)
        command=input("Press 1 to goback menu, Press 0 to terminate : ")
        if command=='1':
            user_menu(user_id)
        else:
            return

    elif song_album_exist==1:
        command=input("Press 1 to play, 2 to go menu, any other keys to terminate : ")
        if command=='1':
            play_song=int(input("input index to play: "))
            play_music(user_id,song_id[play_song],name)
        elif command=='2':
            user_menu(user_id)
        else:
            return
```

곡의 이름을 먼저 받은뒤에, db에 접근하여 해당 이름을 가진 곡을 모두 보여줍니다.

잘려서 보이지 않는 sql문은

```
            sql="select ar.artist_name,s.song_name, s.play_time, al.album_name, s.song_id from song as s, album
 as al, album_song as r, artist as ar where %s = s.song_name and s.song_id=r.song and r.album=al.album_id and s
.artist=ar.artist_id "
```

입니다.

이때, 곡의 ID NUMBER를 song_id TUPLE에 저장해두어, 곡을 PLAY 할때, 인자로 저장할 수 있도록 합니다.

6. 곡 재생

```
def play_music(user_id,song_id,song_name): # user 정보를 저장해놓아야함
    #streamin : sonng user play time
    sql1='select user, song  from streaming where %s=user and %s= song'
    cursor.execute(sql1,(user_id,song_id))
    streaming_history=cursor.fetchall()
    streaming_exist=0
    for streaming in streaming_history:
        print(streaming[0],streaming[1])
        if(user_id==streaming[0] and song_id==streaming[1]): # 이미 기록이 있음
            streaming_exist=1
            sql='UPDATE streaming set play_time=play_time+1 where user=%s and song=%s'
            cursor.execute(sql,(user_id,song_id))
            conn.commit()
            print('Now Playing \'%s\' ..... ' %song_name)
            break
    if streaming_exist==0: # 처음 플레이한 곡
        sql='insert into streaming(song,user,play_time) values (%s,%s,1)'
        cursor.execute(sql,(song_id,user_id))
        conn.commit()
        print('Now Playing \'%s\' ..... \n' %song_name)

    user_menu(user_id)
```

곡 검색 함수에서 곡의 id를 인자로 전달 받아, 해당 곡을 플레이 하는 함수 입니다.

이때, 유저가 해당 곡을 플레이한 기록이 있으면, UPDATE을 통해 해당 TUPLE에 PLAY_TIME+=1을 해주고, 처음 듣는 곡의 경우에는 새로 streaming 기록을 만들어 재생횟수를 1로 설정해줍니다.

```
1. Search Song
2. Search Album
3. Search Artist
4. Show all song list
5. Go to my Playlist
6. Show ranking chart
Any other key to terminate :  1
input name of Song to search : Someone like you
  Artist            Music  Playtime Album     ID
0  Adele  Someone like you  00:04:46    21  20005
1  Adele  Someone like you  00:04:46    25  20005
Press 1 to play, 2 to go menu, any other keys to terminate : 1
input index to play: 0
Now Playing 'Someone like you' .....

1. Search Song
2. Search Album
3. Search Artist
4. Show all song list
5. Go to my Playlist
6. Show ranking chart
Any other key to terminate :  1
input name of Song to search : no_song
Not founed..
Check the capital again 'no_song'
Press 1 to goback menu, Press 0 to terminate :
```

6~7의 실행결과로, 곡이 없는 경우에는 Not found 메시지를 출력합니다.

7. 앨범 검색기능

```
def search_album(user_id):
    name=input('input name of Album to search : ')
    sql='select album.album_name from  album where %s=album_name'
    cursor.execute(sql,(name))
    find_song_album=cursor.fetchall()
    song_album_exist=0
    for row in find_song_album:
        if(name==row[0]):
            song_album_exist=1
            sql='select al.album_name, r.track_num, s.song_name,ar.artist_name,s.play_time from song as s,
            cursor.execute(sql,(name))
            resultset=cursor.fetchall()
            Result=DataFrame(resultset,columns=['Album','Track num','Music','Artist', 'Playtime'])
            print(Result)
            break

    if song_album_exist==0:
        print("Not founed... \nCheck the capital again \'%s\'"%name)
        command=input("Press 1 to goback menu, Press 0 to terminate : ")
        if command=='1':
            user_menu(user_id)
        else:
            return
    elif song_album_exist==1:
        command=input("Press 1 to goback menu, Press 0 to terminate : ")
        if command=='1':
            user_menu(user_id)
        else:
            return
```

앨범 검색기능은, 앨범의 이름을 받아, 해당 앨범을 DB에서 찾습니다. 곡과 마찬가지로 앨범이 있는지 없는지 확

인합니다. 앨범을 찾은 뒤, 앨범과, 음원, 아티스트의 수록정보를 모두 JOIN하여 앨범의 수록곡, 아티스트를 자세하게 보여줍니다.

잘려서 보이지 않는 sql문은

```
sql='select al.album_name, r.track_num, s.song_name,ar.artist_name, s.play_time
from song as s, album as al, album_song as r, artist as ar
where %s=al.album_name and al.album_id=r.album and r.song=s.song_id
 and s.artist=ar.artist_id'
```

입니다.

```
input name of Album to search : Be
  Album  Track num           Music Artist  Playtime
0    Be          1    Life goes on    BTS  00:03:28
1    Be          2  Fly to My Room    BTS  00:03:43
2    Be          3     Blue & Grey    BTS  00:04:15
3    Be          4       Telepathy    BTS  00:03:23
4    Be          5         Disease    BTS  00:04:00
```

앨범 검색 결과를 확인 할 수 있습니다.

8. 아티스트 검색 및 이력 확인

```
def search_artist(user_id):
    name=input('Search artist : ')
    sql1='select artist.artist_name from artist where %s=artist_name'
    cursor.execute(sql1,(name))
    find_artist=cursor.fetchall()
    artist_exist=0
    for row in find_artist:
        print(row[0],name)
        if(name == row[0]):
            artist_exist=1
            sql2='select artist.artist_name,song.song_name from artist,song where %s=artist.artist_name and artist.artist_id=song.artist'
            cursor.execute(sql2,(name))
            resultset2=cursor.fetchall()
            Result=DataFrame(resultset2,columns=['Artist','Music'])
            print(Result)

            command2=input('If you want to view history of \'%s\'  type history/History : '%name)
            if command2=='history' or command2=='History':
                sql3='select artist_name, content from artist,artist_history where %s=artist_name and artist_id=artist'
                cursor.execute(sql3,(name))
                resultset3=cursor.fetchall()
                Result2=DataFrame(resultset3,columns=['Artist','History'])
                print(Result2)
                break

        if artist_exist==0:
            print("No Artist founed...   Check the capital")

    user_menu(user_id)
```

아티스트의 이름을 입력받아, SELECT로 DB에 값을 전달해, 존재하는 아티스트인지 확인을 먼저합니다.

이후 ARTIST와 SONG TABLE을 JOIN하여 해당 아티스트의 곡을 SELECT를 통하여 전부 보여줍니다.

추가적으로 해당 아티스트의 이력을 보고 싶으면 history를 입력하면 이력을 볼 수 있습니다.

이때의 SQL문은 ARTIST와 ARTIST_HISTORY를 단순 JOIN하여 생성했습니다.

```
  Artist              Music
0  Adele    Someone like you
1  Adele            Lovesong
2  Adele      Turning Tables
3  Adele  Don't you remember
4  Adele               Hello
5  Adele        Send My Love
6  Adele              Remedy
7  Adele           All I Ask
If you want to view history of 'Adele'  type history/History : history
  Artist                          History
0  Adele                       2013 Debut
1  Adele  Win 2016 Grammy 'Top Female Vocal'
2  Adele  Win 2020 Grammy 'Top Female Vocal'
```

실행 부분입니다.

작곡가 검색과 작곡가 이력확인은 위와 거의 동일합니다.

9. 모든 곡 보여주기 기능

```
def show_all(user_id):
    sql='select song_name,play_time,artist_name from song,artist where song.artist=artist_id order by artist_name'
    cursor.execute(sql)
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=['Music','Play time','Artist'])
    print(Result)
    user_menu(user_id)
```

]

모든 곡을 보여주는데, 아티스트와 join하여 곡 정보와 아티스트 정보를 같이 보여줍니다.

```
                 Music Play time   Artist
0     Someone like you  00:04:46    Adele
1             Lovesong  00:05:17    Adele
2       Turning Tables  00:04:11    Adele
3   Don't you remember  00:04:04    Adele
4                Hello  00:04:56    Adele
..                 ...       ...      ...
96         A+ Let's Go  00:01:25  Villain
97   Coding is My Life  00:04:44  Villain
98          Final exam  00:03:22  Villain
99             No Copy  00:03:17  Villain
100      I Will Get A+  00:09:19  WonSeok
```

결과 입니다.

10. 플레이리스트

```
def view_playlist(user_id):
    sql='select list_name,create_date,list_id from playlist where %s =list_owner'
    cursor.execute(sql,(user_id))
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=['Name','Create Date','List ID'])
    print(Result)
    print('\n')
    save_list = []
    for value in (resultset):
       save_list.append(value[2])

    command=int(input('Index of playlist. See datail \n119. Create another Playlist \n112. Delete Playlist  \n999. Goback menu  : '))
    if command==119:
        create_playlist(user_id)
    elif command==112:
        delete_list=int(input('Type the number of playlist to delete : '))
        delete_playlist(user_id,save_list[delete_list])
    elif command==999:
        user_menu(user_id)
    else:
        view_detail(user_id,save_list[command])
```

유저가 소장하고 있는 플레이리스트를 모두 보여줍니다. 이때는 간단히, 리스트이름과 생성날짜만을 보여줍니다.

여기서 원하는 리스트의 정보를 index를 간단하게 입력해주어 자세히 볼 수 있습니다.

해당 부분은 view_detail 함수로 구현했습니다.

```
def view_detail(user_id,list_id):
    sql='select list_name,song_name,artist_name,play_time,song_id from playlist,list_song,artist,song
    cursor.execute(sql,(user_id,list_id,list_id))
    resultset=cursor.fetchall()
    Result = DataFrame(resultset,columns=['List name','Music','Artist','Playtime','Song id num'])
    print(Result)
    save_id=[]
    save_name=[]
    for value in (resultset):
       save_id.append(value[4])
       save_name.append(value[1])

    command=int(input('Index number of music. Play/Delete\n119. add music on this list : '))
    if command==119:
        add_music(user_id,list_id)

    else:
        command2=int(input("111. Delete\n112. Play : "))
        if command2==111:
            delete_from_list(user_id,list_id,save_id[command])
        elif command2==112:
            play_music(user_id,save_id[command],save_name[command])

    user_menu(user_id)
```

PLAYLIST와, 수록정보, SONG,ARTIST 4개 TABLE을 JOIN하여, SELECT를 통하여 리스트 내부의 곡을 Detail하게 볼 수 있습니다.

이때 해당 플레이리스트를 들어가면, 리스트 내부의 원하는 곡을 PLAY 할 수 있고, 리스트에 곡을 추가하거나 제거할 수 있습니다.

```
def add_music(user_id,playlist_id):
    result = search_song(user_id)
    index = int(input('Index of the song to add : '))
    song_id=result[index]
    check_sql='select count(*) from list_song where %s=list and %s = song '
    cursor.execute(check_sql,(playlist_id,song_id))
    check_result=cursor.fetchall()
    print(check_result[0][0])
    if check_result[0][0]>0:
        print("It is already in your playlist")
        return
    else:
        sql='insert into list_song(list,song) values(%s,%s)'
        cursor.execute(sql,(playlist_id,song_id))
        conn.commit()
        print('List Updated...')
        return
```

SQL문의 COUNT를 활용하여, 해당 곡이 이미 리스트에 있으면 count가 1이므로, 이미 곡이 등록되어있다는 문구를 출력합니다.

이때, 곡을 추가할때는 serach_song 함수의 일부를 다시 활용하여 곡의 id를 자동으로 반환받아 등록이 가능합니다.

```
def search_song(user_id): #only for gett song_id
    name=input('input name of Song to search : ')
    sql='select song_name from song where %s=song_name'
    cursor.execute(sql,(name))
    find_song_album=cursor.fetchall()
    song_album_exist=0
    song_id=99999
    id_list=[]
    for row in find_song_album:
        if(name==row[0]):
            song_album_exist=1
            #UPPER(컬럼명) LIKE UPPER(검색명)
            sql="select ar.artist_name,s.song_name, s.play_time, al.album_name, s.song_id fr
            cursor.execute(sql,(name))
            resultset=cursor.fetchall()
            for row in resultset:
                id_list.append(row[4])
            Result=DataFrame(resultset,columns=['Artist','Music','Playtime','Album','ID'])
            print(Result)

            break

    if song_album_exist==0:
        print("Not founed..  \nCheck the capital again \'%s\'"%name)
        command=input("Press 1 to goback menu, Press 0 to terminate : ")
        if command=='1':
            user_menu(user_id)
        else:
            return
    return id_list
```

위 함수는 search_and_play_song() 함수와는 곡의 ID 번호를 반환한다는 점을 제외하면 전부 동일합니다.

```
def delete_from_list(user_id,list_id,song_id):
    sql='delete from list_song where list=%s and song=%s'
    cursor.execute(sql,(list_id,song_id))
    conn.commit()
    print("Delete completed.")
```

리스트에서 곡을 제거하는 함수입니다.

DELETE 명령을 이용하여 해당 리스트에 곡을 제거해버립니다.

다른 TABLE의 TUPLE은 건들필요없이 리스트와 음원의 수록관계 TABLE에서만 DELETE처리해줍니다.

10-1 리스트 자세히 보기 -> 곡 추가

```
    List name            Music        Artist  Playtime  Song id num
0  myplaylist  Someone like you         Adele  00:04:46        20005
1  myplaylist         Bellyache  Billie Eilish  00:02:59        20019
2  myplaylist         This Love        Maroon5  00:03:27        20026
Index number of music. Play/Delete
119. add music on this list : 119
input name of Song to search : TT
  Artist Music  Playtime             Album     ID
0  TWICE    TT  00:03:32  TWICEcoaster PART2  20077
Index of the song to add : 0
0
List Updated...
```

10-2 리스트 자세히 보기 -> 곡 제거

```
    List name            Music        Artist  Playtime  Song id num
0  myplaylist  Someone like you         Adele  00:04:46        20005
1  myplaylist         Bellyache  Billie Eilish  00:02:59        20019
2  myplaylist         This Love        Maroon5  00:03:27        20026
3  myplaylist                TT         TWICE  00:03:32        20077
Index number of music. Play/Delete
119. add music on this list : 3
111. Delete
112. Play : 111
Delete completed.
```

11.인기 순위 보여주기

```
def show_ranking(user_id):
    sql='select song_name,artist_name,s.play_time,sum(st.play_time) from streaming as st, sor
    cursor.execute(sql)
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=['Music','Artist','Play length','Play_time'])
    print(Result)
    print('\n')
    user_menu(user_id)
```

유저의 스트리밍 기록을 통합하여, **SUM(PLAY TIME)** 을 이용하여, 곡을 재생순서의 내림차순, 즉 인기순대로 보여줍니다.

잘려서 보이지 않는 SQL 문 입니다.

```
sql='select song_name,artist_name,s.play_time,sum(st.play_time)
From streaming as st, song as s, user as u,artist as a
where st.song=s.song_id and st.user=u.user_id and a.artist_id=s.art
ist group by song_name order by sum(st.play_time) desc'
```

실행결과입니다.

```
              Music        Artist Play length Play_time
0       Giant Rabbit      NanYeong    00:03:02        41
1           Believed          Lauv    00:02:50        15
2       Send My Love         Adele    00:03:44        13
3  It was always you       Maroon5    00:04:00        11
4                 TT         TWICE    00:03:32         2
5   Someone like you         Adele    00:04:46         2
6             My Boy  Billie Eilish    00:02:51         1
```

12.유저 삭제

관리자 ONLY 기능으로, **FOREIGN KEY CONSTRAINT를 유지하기 위하여**

유저를 삭제하기 전 유저의 스트리밍 정보, PLAYLIST, SONG-PLAYLIST TUPLE을 모두 제거해줍니다.

```
def Delete_music(): # 수록정보-소장정보-스트리밍정보 먼저 제거
    song_name=input("Input song name to delete : ")
    sql1='select song_id,song_name,artist_name from song, artist where song_name=%s and artist=artist_id'
    cursor.execute(sql1,(song_name))
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=['song id','song name','artist'])
    print(Result)
    print('\n')
    song_id=input('Input song id to delete : ')
    sql2='delete from album_song where song=%s'
    cursor.execute(sql2,(song_id))
    conn.commit()
    sql3='delete from list_song where song=%s'
    cursor.execute(sql3,(song_id))
    conn.commit()
    sql4='delete from streaming where song=%s'
    cursor.execute(sql4,(song_id))
    conn.commit()
    sql5='delete from song where song_id=%s'
    cursor.execute(sql5,(song_id))
    conn.commit()
    print('Delete Completed....')

    admin_menu()
```

위의 3가지에 해당하는 TUPLE을 모두 제거한 뒤에 최종적으로 유저를 제거합니다.

```
Admin ID : 90000
Password: 1234
Welcome admin1
1. Register new user
2. Delete user
3. Register music/album
4. Delete music
5. Show User list
6. Manage user info-
7. Terminate : 2
Input user_id to delete : 50007
Delete Completed....
```

정상적으로 삭제가 수행이 된 것을 확인할 수 있습니다.

13.음원 등록

```
def Register_music():
    input1=input("Is it a song by existing artist in db??  y/n : ")
    if input1=='y' or input1 == 'Y': # 이미 artist 있음 -> 단순 등록
        artist_id=search_artist_admin()
        song_name=input("Song name : ")
        print(type(artist_id))
        paly_time=input("Play time : ")
        Registration_date=datetime.today().strftime("%Y/%m/%d %H:%M:%S")
        sql1='insert into song(song_name,play_time,artist,Registration_date) VALUES(%s,%s,%s,%s)'
        cursor.execute(sql1,(song_name,paly_time,artist_id,Registration_date))

        conn.commit()
        print("Music Registration Completed....")
        admin_menu()
    elif input1=='n' or input1=='N':
        print("Register Artist first....")
        artist_name=input("Input artist name to register : ")
        sql1='insert into artist(artist_name) values(%s)'
        cursor.execute(sql1,(artist_name))
        conn.commit()
        Register_music()
```

음원을 등록할때는, 해당 음원의 아티스트가 DB에 있는지 확인한 뒤, 새로운 아티스트면 바로 만들어줍니다.

기존 아티스트의 새로운 음원의 경우, 바로 곡의 정보를 받은 뒤 INSERT INTO SONG 명령으로 입력해주면 됩니다.

등록날짜는 파이썬의 DATE모듈을 이용하여 현재 날짜와 시간을 그대로 가져다 썼습니다.

```
1. Register new user
2. Delete user
3. Register music/album
4. Delete music
5. Show User list
6. Manage user info-
7. Terminate : 3
Is it a song by existing artist in db??  y/n : N
Register Artist first....
Input artist name to register : test_artist
Is it a song by existing artist in db??  y/n : y
Input artist name : test_artist
15
Song name : testsong
<class 'int'>
Play time : 0000
Music Registration Completed....
```

새로운 아티스트를 등록하고, 곡을 넣어준 결과입니다.

14.음원 삭제

```
def Delete_music(): # 수록정보-소장정보-스트리밍정보 먼저 제거
    song_name=input("Input song name to delete : ")
    sql1='select song_id,song_name,artist_name from song, artist where song_name=%s and artist=artist_id'
    cursor.execute(sql1,(song_name))
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=['song id','song name','artist'])
    print(Result)
    print('\n')
    song_id=input('Input song id to delete : ')
    sql2='delete from album_song where song=%s'
    cursor.execute(sql2,(song_id))
    conn.commit()
    sql3='delete from list_song where song=%s'
    cursor.execute(sql3,(song_id))
    conn.commit()
    sql4='delete from streaming where song=%s'
    cursor.execute(sql4,(song_id))
    conn.commit()
    sql5='delete from song where song_id=%s'
    cursor.execute(sql5,(song_id))
    conn.commit()
    print('Delete Completed....')

    admin_menu()
```

음원 삭제도 유저 삭제와 마찬가지로, **FOREIGNKEY CONSTRAINT 를 유지하기 위해 음원 수록정보, 플레이리스트 소장정보, 스트리밍 정보를을 모두 제거 후에 최종적으로 음원을 DELETE 명령으로 제거합니다.**

```
1. Register new user
2. Delete user
3. Register music/album
4. Delete music
5. Show User list
6. Manage user info-
7. Terminate : 4
Input song name to delete : testsong
   song id song name       artist
0    30004  testsong  test_artist


Input song id to delete : 30004
Delete Completed....
```

삭제가 잘 되었습니다.

15.모든 유저정보 확인

```
def Show_userlist():
    sql='select * from user'
    cursor.execute(sql)
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=["Knickname","ID number",'Password','Name','Sex','Phone number','Birth date'])
    print(Result)
    admin_menu()
```

단순히 SELECT * FROM USER 명령으로 모든 유저의 정보를 그대로 보여주도록 했습니다.

```
  Knickname  ID number Password Name Sex Phone number Birth date
0  pupafafa      50006     1234  kim   f            0          0
```

아까 유저를 삭제했기에 계정이 한 개 밖에 없습니다.

16.유저 정보 변경

```
def manage_user():
    sql='select * from user'
    cursor.execute(sql)
    resultset=cursor.fetchall()
    Result=DataFrame(resultset,columns=["Knickname","ID number",'Password','Name','Sex','Phone number','Birth date'])
    print(Result)

    user_id=input("Input user id to manage : ")
    knickname=input("Knickname : ")
    password=input("Password : ")
    Name= input ("Name : ")
    Sex=input("Sex : ")
    phone_number=input("Phone number : ")
    Birth=input("Birth date : ")
    sql = 'update user set knickname=%s,user_password=%s,user_Name=%s,Sex=%s,phone_number=%s,birth_date=%s where user_id = %s'
    cursor.execute(sql,(knickname,password,Name,Sex,phone_number,Birth,user_id))
    conn.commit()

    admin_menu()
```

UPDATE 명령으로, 고유 번호인 ID 를 제외하고 유저의 정보를 모두 변경할 수 있습니다.