**Tue/Fri 6 Period**

Database Final Report

- User-Personalized Music Recommendation System -

Team C

21000048 김강산

21000494 이성경

21000759 최철웅

21200021 강진영

21300356 범가연

I.N.D.E.X

[1. Program description 3](#_Toc454456881)

[2. Database System Design (E/R Diagram and Relational Schema) 4](#_Toc454456882)

[3. Program Functions and Query 8](#_Toc454456883)

[4. Constraints (Foreign Keys) 20](#_Toc454456884)

[5. Trigger 22](#_Toc454456885)

1. Program description
2. Description

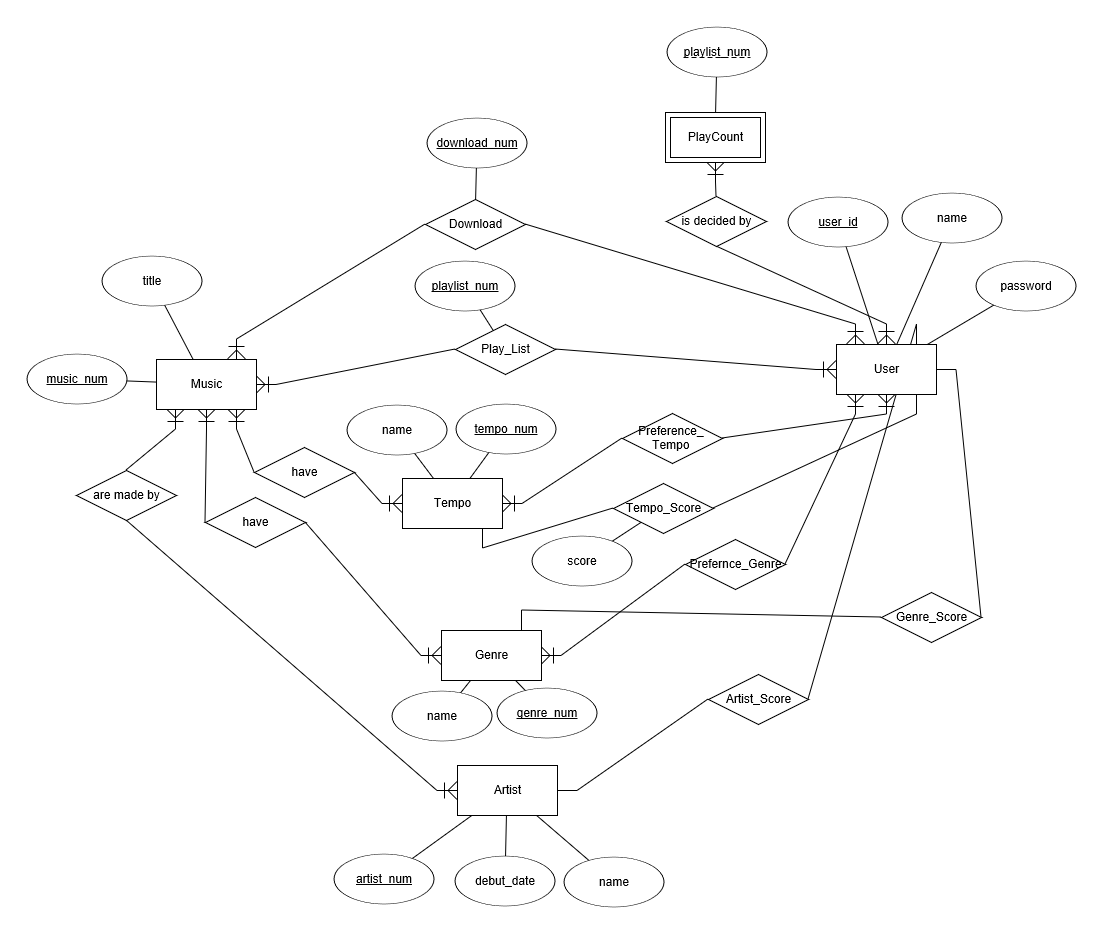
: User-personalized music recommendation system.

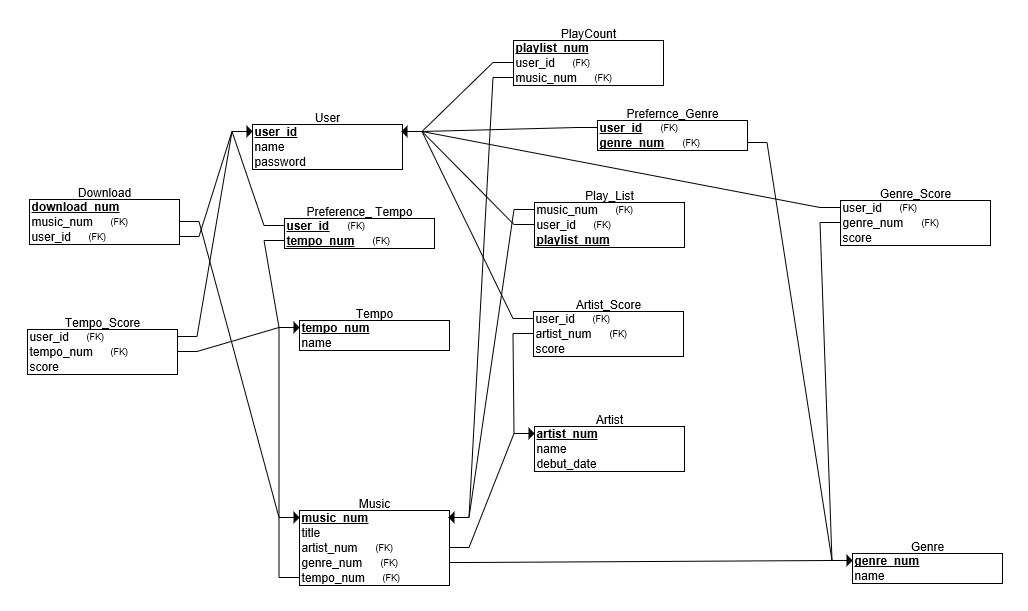
Program analyzes user’s preference based on user’s selections. When user play music, download music, add music to play list and select genre and tempo preference, program scores user’s preference item by item; artist, genre, tempo. On the basis of these score, program recommends music.

1. User instruction

User can start the program by login or register. After login, user can use search and add music to playlist or download music which user want by enter the music index number that console shows. User also can manage play list by play music and remove music. In recommendation page, user selects genre and tempo preference. Based on user’s whole selections, program gives music recommendation list. User can add music to play list or download music, like search page. Also user can check one’s own preference.

1. Database System Design (E/R Diagram and Relational Schema)



[Figure ] E/R Diagram

[Figure ] Relational Schema

Our system consists of 13 Tables.

1. Artist

It has three attributes.

“Artist\_num” is an index attribute.

“Name” is an attribute, which indicates Artist’s name.

“Debut\_data” is an attribute, which shows Artist’s Debut date.

1. Artist\_Score

It has three attributes.

“User\_id” is Foreign key, which references [User] table and indicates user’s id.

“Artist\_num” is Foreign key, which references [Artist] table.

“Score” is attribute to save each Artist’s score

1. Download

It has three attributes.

“Download\_num” is an index attribute.

“Music\_num” is Foreign key, which references [Music] table and indicates music’s index.

“User\_id” is Foreign key, which references [User] table and indicates user’s id.

1. Genre

It has two attributes.

“Genre\_num” is an index attribute.

“Name” is an attribute, which indicates Genre’s type.

1. Genre\_Score

It has three attributes.

“User\_id” is Foreign key, which references [User] table and indicates User’s id.

“Genre\_num” is Foreign key, which references [Genre] table and indicates Genre’s index.

“Score” is attribute to save each Genre’s Score.

1. Music

It has five attributes.

“Music\_num” is index attribute.

“Title” is attribute to indicate music title.

“Genre\_num” is Foreign key, which references [Genre] table and indicates Genre’s index.

“Tempo\_num” is Foreign key, which references [Tempo] table and indicates Tempo’s index.

“Artist\_num” is Foreign key, which references [Artist] table and indicates Artist’s index.

1. Play\_Count

It has three attributes.

“Music\_num” is Foreign key, which references [Music] table and indicates Music’s index.

“User\_id” is Foreign key, which references [User] table and indicates User’s id.

“Play\_Count” is attribute to count how the music plays.

1. Play\_List

It has three attributes.

“Playlist\_num” is index attribute.

“User id” is Foreign key, which references [User] table and indicates User’s id.

“Music\_num” is Foreign key, which references [Music] table and indicates Music’s index.

1. Preference\_Genre

It has two attributes.

“Genre\_num” is Foreign key, which references [Genre] table and indicates Genre’s index.

“User\_id” is Foreign key, which references [User] table and indicates User’s id.

1. Preference\_Tempo

It has two attributes.

“Tempo\_num” is Foreign key, which references [Tempo] table and indicates Tempo’s index.

“User\_id” is Foreign key, which references [User] table and indicates User’s id.

1. Tempo

It has two attributes.

“Tempo” is index attribute.

“Name” is attribute, which indicates Tempo’s type.

1. Tempo\_Score

It has three attributes.

“User\_id” is Foreign key, which references [User] table and indicates User’s id.

“Tempo\_num” is Foreign key, which references [Tempo] table and indicates Tempo’s index.

“Score” is attribute to save each Tempo’s score.

1. User

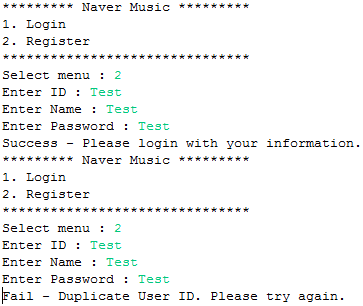
It has three attributes.

“User\_id” is attribute to indicate User’s id.

“Name” is attribute to show User’s name.

“Password” is attribute to indicate User’s password.

1. Program Functions and Query
2. **Register**



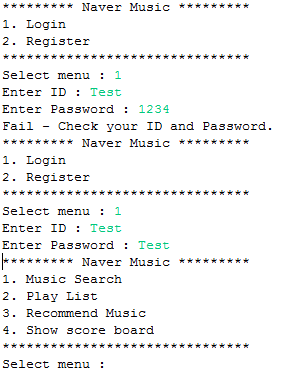
When user select menu number 2, program require some information ( ID, Name, Password ). If there is duplicate user id, then print error message.

**Query**

String query = "INSERT INTO User (user\_id, name, password) VALUES ('"+id+"', '"+name+"', '"+password+"')";

Insert information to User table.

1. **Login**



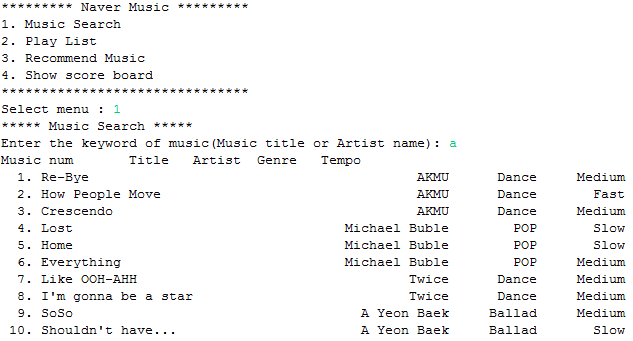
When user select menu number 1, program require information user id and password. If there is no matching with Database, then print error message. But if there is matching with Database, then print success message and print Main menu.

**Query**

String query = "SELECT \* FROM User where user\_id='" + id + "' and password='" + password + "' LIMIT 1";

Compare user input information with User table.

1. **Music Search**



When user select menu number 1, program require some keyword of music(title or artist). If user input keyword, then program print all music list that including keyword. And print next step menu.

**Query**

String query = "select \* from "

+ "(select m.music\_num, m.title, a.name as Aname, t.name as Tname, g.name as Gname "

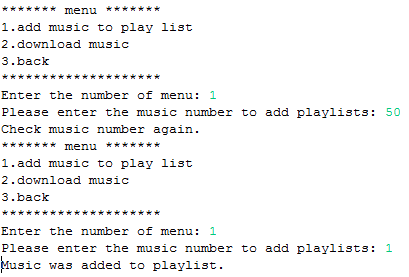
+ "from Music as m , Artist as a, Tempo as t, Genre as g where m.artist\_num=a.artist\_num "

+ "and m.tempo\_num=t.tempo\_num and g.genre\_num=m.genre\_num) p " + "where title like '%" + keyword

+ "%' or Aname like '%" + keyword + "%' order by music\_num asc;";

Select all music list including keyword.

1. **Add music to play list**

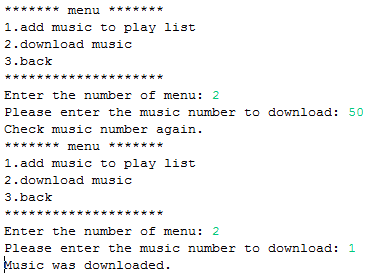


When user select menu number 1, program require music number. If user input number of music, then add music to play list table. If user input another character, print error message.

**Query**

query = "INSERT INTO Play\_List (user\_id, music\_num) VALUES ('" + user\_id + "', " + music\_num + ");";

Insert information to Play\_List table.



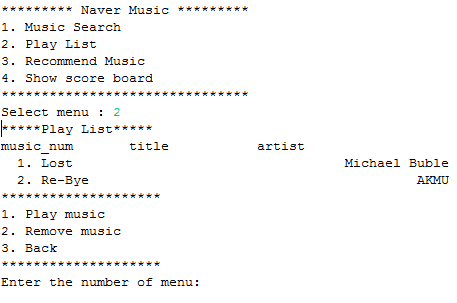
When user select menu number 2, program require music number. If user input number of music, then add music to download table. If user input another character, print error message.

**Query**

query = "INSERT INTO Download (music\_num, user\_id) VALUES ('" + music\_num + "', '" + user\_id + "');";

Insert information to Download table.

1. **Play List**



When user select menu number 2, program print all music in play list. And print sub menu.

**Query**

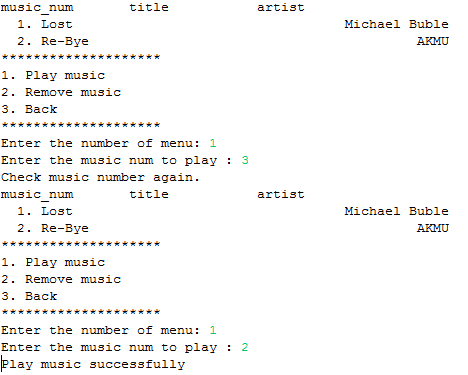
query = "SELECT \* FROM (SELECT pl.playlist\_num, m.\*, a.name as Aname FROM Play\_List as pl, Music as m, Artist as a "

+ "WHERE user\_id = '" + user\_id

+ "' and pl.music\_num = m.music\_num and m.artist\_num = a.artist\_num) p ;";

Select all music list in Play\_List table.

1. **Play music**



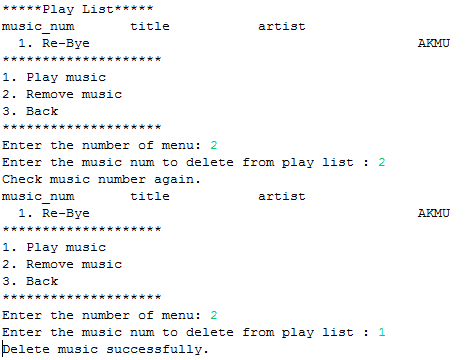
When user select menu number 1, program require number of music. If user input number, then update +1 play\_count field in Play\_Count table. If user input another character, print error message.

**Query**

query = "UPDATE Play\_Count SET play\_count = play\_count + 1 WHERE user\_id = '" + user\_id + "' AND music\_num='" + music\_num + "';";

Update play\_count +1 in Play\_Count table.

1. **Remove music**



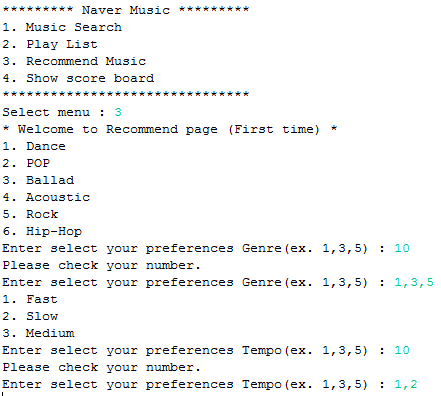
When user select menu number 1, program require number of music. If user input number, then delete from Play\_Count table. If user input another character, print error message.

**Query**

query = "DELETE FROM Play\_List WHERE playlist\_num='" + playlist\_num + "';";

Delete music in Play\_Count table.

1. **Recommend music(First time)**



When user select menu number 3, if user visit this page at first, program require some information about user’s preference of music. If user input wrong numbers, print error message. If user input right numbers, print next step.

**Query**

query = "SELECT \* FROM Genre";

Select all genre names.

query = "START TRANSACTION;";

query = "DELETE FROM Preference\_Genre WHERE user\_id='" + user\_id + "'";

query = "INSERT INTO Preference\_Genre (genre\_num, user\_id) VALUES ( '" + genre + "', '" + user\_id + "' )";

query = "ROLLBACK";

query = "COMMIT";

Use transaction for user input something wrong character or not exist number. If user right numbers, then execute commit query, else execute rollback query.

query = "SELECT \* FROM Tempo";

Select all tempo names.

query = "START TRANSACTION;";

query = "DELETE FROM Preference\_Tempo WHERE user\_id='" + user\_id + "'";

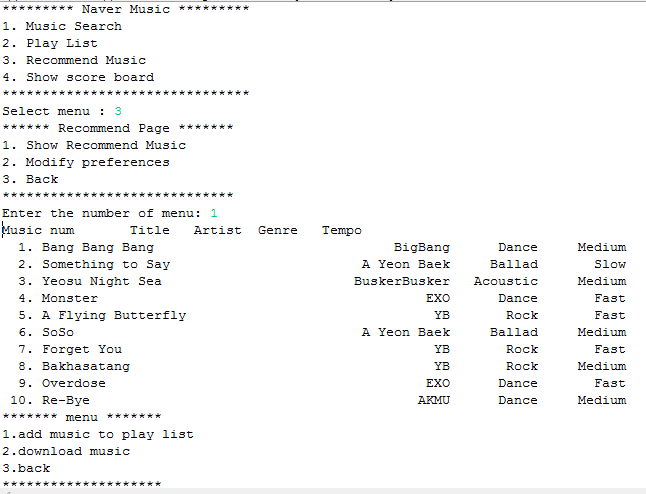
query = "INSERT INTO Preference\_Tempo (tempo\_num, user\_id) VALUES ( '" + tempo + "', '" + user\_id + "' )";

query = "ROLLBACK";

query = "COMMIT";

Use transaction for user input something wrong character or not exist number. If user right numbers, then execute commit query, else execute rollback query.

1. **Recommend Music (after) – Show recommend music**



When user select menu number 1, program print 10 music randomized. Because of assumption that the recommend algorithm using score board is already implemented.

**Query**

query = "SELECT p.\* FROM "

+ "(SELECT m.music\_num, m.title, a.name as Aname, t.name as Tname, g.name as Gname "

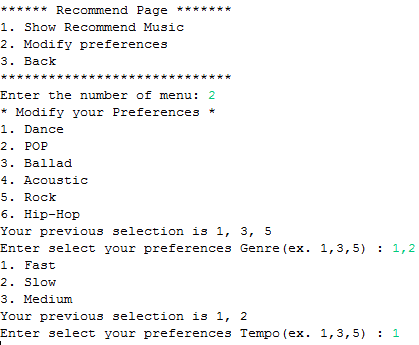
+ "FROM Music as m , Artist as a, Tempo as t, Genre as g where m.artist\_num=a.artist\_num "

+ "AND m.tempo\_num=t.tempo\_num AND g.genre\_num=m.genre\_num) p "

+ "ORDER BY RAND() LIMIT 10;";

Select 10 music randomized.

1. **Recommend Music (after) – Modify preferences**



When user select menu number 2, program print preference list with previous selected. Another implementation are same with visit at first time.

**Query**

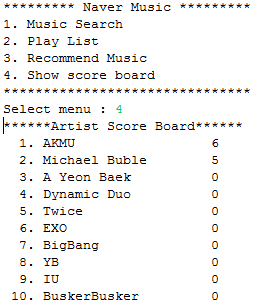
query = "SELECT \* FROM Preference\_Genre as pg WHERE pg.user\_id='" + user\_id + "'";

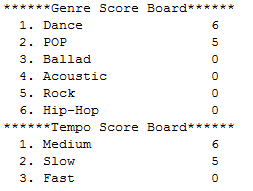
Select user’s previous selection in Preference\_Genre table.

query = "SELECT \* FROM Preference\_Tempo as pt WHERE pt.user\_id='" + user\_id + "'";

Select user’s previous selection in Preference\_Tempo table.

1. **Show score board**





When user select menu number 4, program show score board about artist, genre, tempo that user interested.

**Query**

query = "SELECT a.name, IF(ars.score IS NULL, 0, ars.score) as score FROM Artist as a "

+ "LEFT OUTER JOIN Artist\_Score as ars on ars.artist\_num=a.artist\_num AND ars.user\_id='" + user\_id + "' ORDER BY ars.score DESC";

Select artist score

query = "SELECT g.name, IF(gs.score IS NULL, 0, gs.score) as score FROM Genre as g "

+ "LEFT OUTER JOIN Genre\_Score as gs on gs.genre\_num=g.genre\_num AND gs.user\_id='" + user\_id + "' ORDER BY gs.score DESC";

Select genre score

query = "SELECT t.name, IF(ts.score IS NULL, 0, ts.score) as score FROM Tempo as t "

+ "LEFT OUTER JOIN Tempo\_Score as ts on ts.tempo\_num=t.tempo\_num AND ts.user\_id='" + user\_id + "' ORDER BY ts.score DESC";

Select tempo score.

1. Constraints (Foreign Keys)
2. **Artist\_Score**

Use foreign keys to refer the information of Artist and User.

CONSTRAINT ‘fk\_Artist\_Score\_artist\_num’ FOREIGN KEY (‘artist\_num’) REFERENCES ‘Artist’ (‘artist\_num’)

CONSTRAINT ‘fk\_Artist\_Score\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

1. **Genre\_Score**

Use foreign keys to refer the information of Genre and User.

CONSTRAINT ‘fk\_Genre\_Score\_genre\_num’ FOREIGN KEY (‘genre\_num’) REFERENCES ‘Genre’ (‘genre\_num’)

CONSTRAINT ‘fk\_Genre\_Score\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

1. **Tempo\_Score**

Use foreign keys to refer the information of Tempo and User.

CONSTRAINT ‘fk\_Tempo\_Score\_genre\_num’ FOREIGN KEY (‘tempo\_num’’) REFERENCES ‘Tempo’ (‘tempo\_num’)

CONSTRAINT ‘fk\_Tempo\_Score\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

1. **Download**

Use foreign keys to refer the information of User and Music.

CONSTRAINT ‘fk\_download\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

CONSTRAINT ‘fk\_download\_music\_num’ FOREIGN KEY (‘music\_num’) REFERENCES ‘Music’ (‘music\_num’)

1. **Music**

Use foreign keys to refer the information of Artist, Tempo, and Genre.

CONSTRAINT ‘fk\_music\_aritst\_num’ FOREIGN KEY (‘aritst\_num’) REFERENCES ‘Artist’ (‘aritst\_num’)

CONSTRAINT ‘fk\_music\_tempo\_num’ FOREIGN KEY (‘tempo\_num’) REFERENCES ‘Tempo’ (‘tempo\_num’)

CONSTRAINT ‘fk\_music\_genre\_num’ FOREIGN KEY (‘genre\_num’) REFERENCES ‘Genre’ (‘genre\_num’)

1. **Play\_Count**

Use foreign keys to refer the information of User and Music.

CONSTRAINT ‘fk\_play\_count\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

CONSTRAINT ‘fk\_play\_count\_music\_num’ FOREIGN KEY (‘music\_num’) REFERENCES ‘Music’ (‘music\_num’)

1. **Play\_List**

Use foreign keys to refer the information of User and Music.

CONSTRAINT ‘fk\_Play\_List\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

CONSTRAINT ‘fk\_Play\_List\_music\_num’ FOREIGN KEY (‘music\_num’) REFERENCES ‘Music’ (‘music\_num’)

1. **Preference\_Genre**

Use foreign keys to refer the information of User and Genre.

CONSTRAINT ‘fk\_preference\_genre\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

CONSTRAINT ‘fk\_preference\_genre\_genre\_num’ FOREIGN KEY (‘genre\_num’) REFERENCES ‘Genre’ (‘genre\_num’)

1. **Preference\_Tempo**

Use foreign keys to refer the information of User and Tempo.

CONSTRAINT ‘fk\_preference\_tempo\_User\_id’ FOREIGN KEY (‘User\_id’) REFERENCES ‘User’ (‘User\_id’)

CONSTRAINT ‘fk\_preference\_tempo\_num\_genre\_num’ FOREIGN KEY (‘tempo\_num’) REFERENCES ‘Tempo’ (‘tempo\_num’)

1. Trigger
2. **Add music to play list**

Scores of genre, tempo and artist related to the music increase 1 point.

When music is added to playlist, the new tuple is inserted to Play\_List table. At this time, the trigger makes score of the table increase. If a related score tuple is already made, increase the score. Otherwise, insert a tuple that has 1 point.

Trigger: PlayList\_To\_Score

Event: INSERT

Table: Play\_List

Statement: BEGIN

#In Genre\_Score, if there is a tuple that has same genre\_num with now added music and user\_id

IF (SELECT EXISTS( SELECT \*

FROM Genre\_Score, Music

WHERE Genre\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Genre\_Score.genre\_num = Music.genre\_num) )

#If exist, update the value of score with set “score = score+1”

THEN

UPDATE Genre\_Score SET Genre\_Score.score = Genre\_Score.score + 1

WHERE Genre\_Score.user\_id = new.user\_id

AND Genre\_Score.genre\_num

IN ( SELECTMusic.genre\_num FROM Music WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, genre\_num, 1”

ELSE

INSERT INTO Genre\_Score( user\_id, genre\_num )

SELECT new.user\_id as user\_id, genre\_num

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

#In Artist\_Score, if there is a tuple that has same artist\_num with now added music and user\_id

IF (SELECT EXISTS( SELECT \*

FROM Artist\_Score, Music

WHERE Artist\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Artist\_Score.artist\_num = Music.artist\_num) )

#If exist, update the value of score with set “score = score+1”

THEN

UPDATE Artist\_Score SET Artist\_Score.score = Artist\_Score.score + 1

WHERE Artist\_Score.user\_id = new.user\_id

AND Artist\_Score.artist\_num

IN ( SELECTMusic.artist\_num FROM Music WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, artist\_num, 1”

ELSE

INSERT INTO Artist\_Score( user\_id, artist\_num )

SELECT new.user\_id as user\_id, artist\_num

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

#In Tempo\_Score, if there is a tuple that has same tempo\_num with now added music and user\_id

IF (SELECT EXISTS( SELECT \*

FROM Tempo\_Score, Music

WHERE Tempo\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Tempo\_Score.tempo\_num = Music.tempo\_num) )

#If exist, update the value of score with set “score = score+1”

THEN

UPDATE Tempo\_Score SET Tempo\_Score.score = Tempo\_Score.score + 1

WHERE Tempo\_Score.user\_id = new.user\_id

AND Tempo\_Score.tempo\_num

IN ( SELECTMusic.tempo\_num FROM Music WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, tempo\_num, 1”

ELSE

INSERT INTO Tempo\_Score( user\_id, tempo\_num )

SELECT new.user\_id as user\_id, tempo\_num

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

END

1. **Play music**

Scores of genre, tempo and artist related to the music increase 1 point.

When music playing, the value play\_count is updated. At this time, the trigger makes score of the table increase. If a related score tuple is already made, increase the score. Otherwise, insert a tuple that has 1 point.

Trigger: Play\_To\_Score

Event: UPDATE

Table: Play\_Count

Statement: BEGIN

#In Artist\_Score, if there is a tuple that has same artist\_num with now played music and user\_id

IF EXISTS( SELECT \*

FROM Artist\_Score, Music

WHERE Artist\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Artist\_Score.artist\_num = Music.artist\_num)

THEN

#If exist, update the value of score with set “score = score+1”

UPDATE Artist\_Score SET score = score + 1

WHERE Artist\_Score.user\_id = NEW.user\_id

AND Artist\_Score.artist\_num

IN (SELECT artist\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, artist\_num, 1”

ELSE

INSERT INTO Artist\_Score( user\_id, artist\_num, score )

SELECT new.user\_id, artist\_num, '1'

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

#In Genre\_Score, if there is a tuple that has same genre\_num with now played music and user\_id

IF EXISTS( SELECT \*

FROM Genre\_Score, Music

WHERE Genre\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Genre\_Score.genre\_num = Music.genre\_num)

#If exist, update the value of score with set “score = score+1”

THEN UPDATE Genre\_Score SET score = score + 1

WHERE Genre\_Score.user\_id = NEW.user\_id

AND Genre\_Score.genre\_num

IN (SELECT Music.genre\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, genre\_num, 1”

ELSE

INSERT INTO Genre\_Score( user\_id, genre\_num, score )

SELECT new.user\_id, genre\_num, '1'

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

#In Tempo\_Score, if there is a tuple that has same tempo\_num with now played music and user\_id

IF EXISTS( SELECT \*

FROM Tempo\_Score, Music

WHERE Tempo\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Tempo\_Score.tempo\_num = Music.tempo\_num)

#If exist, update the value of score with set “score = score+1”

THEN UPDATE Tempo\_Score SET score = score + 1

WHERE Tempo\_Score.user\_id = NEW.user\_id

AND Tempo\_Score.tempo\_num

IN (SELECT Music.tempo\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, tempo\_num, 1”

ELSE

INSERT INTO Tempo\_Score( user\_id, tempo\_num, score )

SELECT new.user\_id, tempo\_num, '1'

FROM Music

WHERE music\_num = NEW.music\_num;

END IF;

END

1. **Download music**

Scores of genre, tempo and artist related to the music increase 5 point.

When music downloaded, the new tuple is inserted to the Download table. At this time, the trigger makes score of the table increase. If a related score tuple is already made, increase the score. Otherwise, insert a tuple that has 5 point.

Trigger: Download\_To\_Score

Event: INSERT

Table: Download

Statement: BEGIN

#In Artist\_Score, if there is a tuple that has same artist\_num with now downloaded music and user\_id

IF EXISTS( SELECT \*

FROM Artist\_Score, Music

WHERE Artist\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Artist\_Score.artist\_num = Music.artist\_num )

#If exist, update the value of score with set “score = score+5”

THEN

UPDATE Artist\_Score SET score = score + 5

WHERE Artist\_Score.user\_id = NEW.user\_id

AND Artist\_Score.artist\_num

IN (SELECT artist\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, artist\_num, 5”

ELSE

INSERT INTO Artist\_Score( user\_id, artist\_num, score )

SELECT new.user\_id, artist\_num,'5'

FROM Music

WHERE music\_num=NEW.music\_num;

END IF;

#In Genre\_Score, if there is a tuple that has same genre\_num with now downloaded music and user\_id

IF EXISTS( SELECT \*

FROM Genre\_Score, Music

WHERE Genre\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Genre\_Score.genre\_num = Music.genre\_num )

#If exist, update the value of score with set “score = score+5”

THEN

UPDATE Genre\_Score SET score = score + 5

WHERE Genre\_Score.user\_id = NEW.user\_id

AND Genre\_Score.genre\_num

IN (SELECT Music.genre\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

#otherwise, insert the tuple with the value of “user\_id, genre\_num, 5”

ELSE

INSERT INTO Genre\_Score( user\_id, genre\_num, score )

SELECT new.user\_id, genre\_num,'5'

FROM Music

WHERE music\_num=NEW.music\_num;

END IF;

#In Tempo\_Score, if there is a tuple that has same tempo\_num with now downloaded music and user\_id

IF EXISTS( SELECT \*

FROM Tempo\_Score, Music

WHERE Tempo\_Score.user\_id = NEW.user\_id

AND NEW.music\_num = Music.music\_num

AND Tempo\_Score.tempo\_num = Music.tempo\_num )

#If exist, update the value of score with set “score = score+5”

THEN

UPDATE Tempo\_Score SET score = score + 5

WHERE Tempo\_Score.user\_id = NEW.user\_id

AND Tempo\_Score.tempo\_num

IN (SELECT Music.tempo\_num

FROM Music

WHERE NEW.music\_num = Music.music\_num );

ELSE

#otherwise, insert the tuple with the value of “user\_id, tempo\_num, 1”

INSERT INTO Tempo\_Score( user\_id, tempo\_num, score )

SELECT new.user\_id, tempo\_num,'5'

FROM Music

WHERE music\_num=NEW.music\_num;

END IF;

END

1. **Select genre preference**

Scores of the genre increase 10 point.

Once user selects genre preference, the tuple about user’s genre preference is deleted and re-inserted in Genre\_Preference table. When delete, trigger decrease the score 10 and when increase the trigger gives 10 point related tuple in Genre\_Score table. If the score tuple is already exist, increase the score. Otherwise, insert a tuple that has 10 point.

Trigger: Preference\_Genre\_Del\_Trigger

Event: DELETE

Table: Preference\_Genre

Statement: BEGIN

UPDATE Genre\_Score SET Genre\_Score.score = Genre\_Score.score - 10 WHERE Genre\_Sc

ore.user\_id = OLD.user\_id and Genre\_Score.genre\_num=OLD.genre\_num;

END

Trigger: Preference\_Genre\_Ins\_Trigger

Event: INSERT

Table: Preference\_Genre

Statement: BEGIN

#If there is the tuple that has same information of genre and user

IF EXISTS( SELECT \* FROM Genre\_Score WHERE Genre\_Score.user\_id = NEW.user\_id and

Genre\_Score.genre\_num = NEW.genre\_num)

#then update the value of the score with score + 10

THEN

UPDATE Genre\_Score SET Genre\_Score.score = Genre\_Score.score + 10 WHERE Genre\_Sc

ore.user\_id = new.user\_id and Genre\_Score.genre\_num=NEW.genre\_num;

#otherwise, insert new tuple with score 10

ELSE

INSERT INTO Genre\_Score( user\_id, genre\_num, score) SELECT new.user\_id, new.genr

e\_num, '10';

END IF;

END

1. **Select tempo preference**

Scores of the tempo increase 10 point.

Once user selects tempo preference, the tuple about user’s tempo preference is deleted and re-inserted in Tempo\_Preference table. When delete, trigger decrease the score 10 and when increase the trigger gives 10 point related tuple in Tempo\_Score table. If the score tuple is already exist, increase the score. Otherwise, insert a tuple that has 10 point.

Trigger: Preference\_Tempo\_Del\_Trigger

Event: DELETE

Table: Preference\_Tempo

Statement: BEGIN

UPDATE Tempo\_Score SET Tempo\_Score.score = Tempo\_Score.score - 10 WHERE Tempo\_Score.user\_id = OLD.user\_id and Tempo\_Score.tempo\_num=OLD.tempo\_num;

END

Trigger: Preference\_Tempo\_Ins\_Trigger

Event: INSERT

Table: Preference\_Tempo

Statement: BEGIN

#If there is the tuple that has same information of tempo and user

IF EXISTS( SELECT \* FROM Tempo\_Score WHERE Tempo\_Score.user\_id = NEW.user\_id and

Tempo\_Score.tempo\_num = NEW.tempo\_num)

#then update the value of the score with score + 10

THEN

UPDATE Tempo\_Score SET Tempo\_Score.score = Tempo\_Score.score + 10 WHERE Tempo\_Sc

ore.user\_id = NEW.user\_id and Tempo\_Score.tempo\_num=NEW.tempo\_num;

ELSE

#otherwise, insert new tuple with score 10

INSERT INTO Tempo\_Score( user\_id, tempo\_num, score) SELECT new.user\_id, new.temp

o\_num, '10';

END IF;

END