实习一：数据库应用案例设计

成员：张成谦 马千里 杨仕博 张钧天

```
%load_ext sql
```

```
import pymysql
pymysql.install_as_MySQLdb()
%sql mysql://stu2100013111:stu2100013111@162.105.146.37:43306
```

```
%sql use stu2100013111;
```

本次实习的目标是设计网易云音乐数据库，包括列举业务需求、设计ER图、将ER图转换为关系表、用SQL语句实现业务功能。

一、业务需求

网易云音乐提供了多种功能，我们主要关注播放音乐、私信互动、云村动态这三个功能。

- 播放音乐：网易云音乐用户通过手机号/qq号/邮箱来登录，创建或收藏歌单，播放歌曲或MV，在音乐下发表评论、喜欢该音乐，还可以查看歌手、专辑等更详细的信息。
- 私信互动：两个网易云音乐用户之间可以互相收发私信。
- 云村动态：用户可以发云村动态，每个动态都可以设置分享范围。所有能看见帖子的用户都能点赞/评论/转发。

二、ER图设计

根据上述业务需求，我们需要确定实体，以及实体之间的联系。

1. 实体

我们共设立12种实体，并确定它们各自的属性（主码用下划线来标识）：

考虑到登录方式可以为手机/微信/邮箱，因此登录方式为多值属性，每个账号用唯一的网易云ID区分

- 用户（网易云ID，登录方式，会员等级，昵称）
- 私信（私信ID，私信时间，私信内容）
- 动态（动态ID，动态内容，分享范围）

为了使表述更精确，采用“特化/概化”的方法，将“音乐”细分为歌曲和MV，二者都能在播放、评论等过程与用户发生类似的联系过程，又需要在归属划分等方面加以区别
它们是“Is a”的“父类-子类”关系，歌曲与MV继承了音乐的ID属性：

- 音乐（音乐ID）
- 歌曲（音乐ID，曲名）
- MV（音乐ID，视频标题）

以及，歌手是用户的一种特化，继承了用户的网易云ID等属性（登录方式，会员等级，昵称这三种属性为简化考虑，不列入歌手属性）

- 歌手（网易云ID，姓名，地区，风格）
- 专辑（专辑ID，专辑名称）
- 歌单（歌单ID，歌单标题，排序方式）

以上9种都是强实体，而对于评论/点赞/转发，其ID都是针对同一个动态而言的（例如动态0和动态1都会有ID为0的评论
/点赞/转发），因此我们将其视作弱实体，依赖于动态而存在。主码应该是动态ID+分辨符。

- 评论（评论ID，动态ID，评论时间，评论内容）
- 点赞（点赞ID，动态ID）
- 转发（转发ID，动态ID，转发时间，转发内容）

2. 联系

实体之间存在着若干种联系，我们将一一列举。

- 关注：用户与用户之间，一对多。该联系具有“关注时间”这一属性
- 发送：用户与私信之间，一对多
- 收到：用户与私信之间，一对多
- 发布动态：用户与动态之间，一对多。该联系具有“发布动态时间”这一属性
- 创建：用户与歌单之间，一对多
- 收藏：用户与歌单之间，多对多
- 播放：音乐与用户之间，一对多。该联系具有“播放状态”、“播放顺序”这两个属性
- 评论：音乐与用户之间，多对多。该联系具有“评论内容”这一属性
- 制作：歌手与音乐之间，多对多

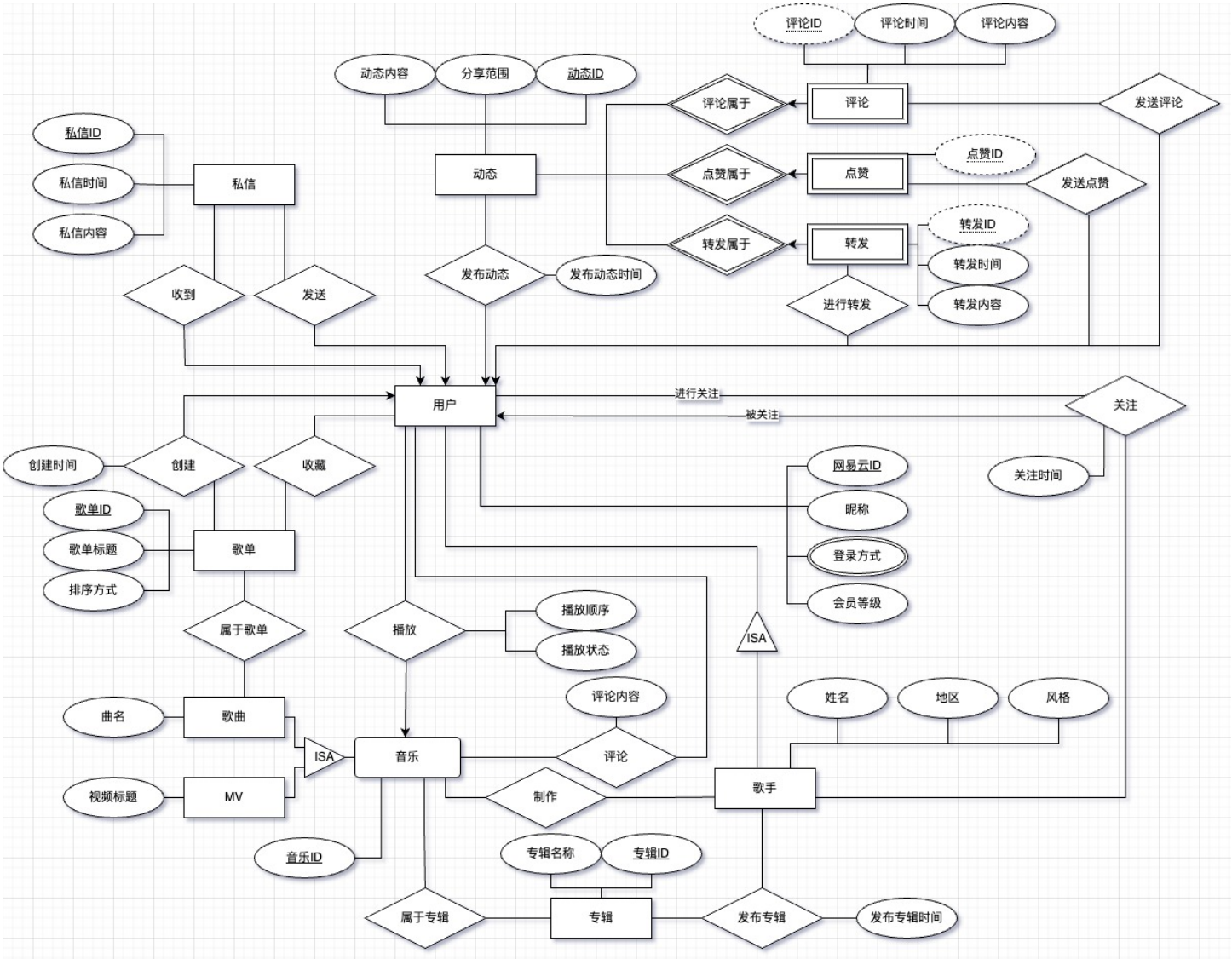
- 发布专辑：歌手与专辑之间，多对多。该联系具有“发布专辑时间”这一属性
- 属于专辑：专辑与音乐之间，多对多
- 属于歌单：歌曲与歌单之间，多对多

对于评论、点赞、转发这三个弱实体，它们与动态有标识性联系，表示隶属关系。它们也与用户之间存在“发送”联系。
即：

- 评论属于：动态与评论之间，一对多
- 点赞属于：动态与点赞之间，一对多
- 转发属于：动态与转发之间，一对多
- 发送评论：用户与评论之间，一对多
- 发送点赞：用户与点赞之间，一对多
- 进行转发：用户与转发之间，一对多

3. ER图

根据列出的实体和联系，我们可以绘制出ER图。



三、关系表创建

完成ER图的设计后，要将其转换成关系表。

9个实体各自对应一张表，要注意的是，3个弱实体的表中应有强实体的主码，代表对强实体的依附。

7个较为重要的联系需单独创建表，表的主码是联系双方的主码。

共有19张表，下面我们一一进行创建，注释中包含着一些说明。

```
%%sql

SET @@foreign_key_checks=0;

-- 1. 创建用户表，并检查qq，邮箱，电话之一是否有填写，并检查填写是否合规。
DROP TABLE IF EXISTS user_tb;

CREATE TABLE user_tb
(
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    level_of_membership TINYINT NULL DEFAULT 1,
```

```
    user_phone VARCHAR(20) NULL,
    qq BIGINT NULL,
    mail VARCHAR(40) NULL,
    nickname VARCHAR(40) NOT NULL,
    CHECK(LENGTH(user_phone)=11 OR user_phone IS NULL),
    CHECK(
        mail LIKE '%@%'
        OR mail IS NULL
    ),
    CHECK(
        user_phone IS NOT NULL OR
        mail IS NOT NULL OR
        qq IS NOT NULL
    )
);

SET @@foreign_key_checks=1;

-- 2. 创建歌手表，包括用户ID（歌手继承用户，以用户id(music_id)为主键），歌手姓名，所在地区和风格

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS singer_tb;
CREATE TABLE singer_tb
(
    singer_id BIGINT NOT NULL PRIMARY KEY,
    singer_name VARCHAR(50) NOT NULL,
    singer_district VARCHAR(50) NOT NULL,
    singer_style VARCHAR(50) NOT NULL,
    constraint singer_fk_user foreign key(singer_id) references user_tb(id)
);
SET @@foreign_key_checks=1;

-- 3. 创建私信表，私信和用户间的“收到”和“发送”关系是一对多关系，需要在表中填写对应的用户id

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS private_message_tb;
CREATE TABLE private_message_tb
(
        id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    time_of_private_message TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    content VARCHAR(255) NOT NULL,
    id_of_the_msg_accepted_user BIGINT NOT NULL,
    id_of_the_msg_sent_user BIGINT NOT NULL,
    CONSTRAINT private_msg_fk_accept FOREIGN KEY(id_of_the_msg_accepted_user)
REFERENCES user_tb(id),
    CONSTRAINT private_msg_fk_send FOREIGN KEY(id_of_the_msg_sent_user) REFERENCES
user_tb(id)
);
SET @@foreign_key_checks=1;

-- 4. 创建专辑表，包括专辑ID和专辑名称

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS album_tb;
CREATE TABLE album_tb
```

```
(
    album_id BIGINT NOT NULL PRIMARY KEY,
    -- 增加专辑名称
    album_name VARCHAR(50) NOT NULL

);
SET @@foreign_key_checks=1;

-- 5. 创建音乐表，包括音乐id、音乐名称和所属专辑（如果有）（因专辑和音乐一对多的联系而在此添加）

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS music_tb;
CREATE TABLE music_tb
(
    id BIGINT NOT NULL PRIMARY KEY,
    music_name VARCHAR(50) NOT NULL,
    belong_album_id BIGINT,
    constraint music_fk_album foreign key(belong_album_id) references
album_tb(album_id)
);
SET @@foreign_key_checks=1;

-- 6. 创建音乐制作表，记录歌手制作音乐的状况

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS produce_tb;
CREATE TABLE produce_tb
(
    producer_id BIGINT NOT NULL,
    produced_music BIGINT NOT NULL,
    primary key(producer_id,produced_music),
    constraint produce_fk_user foreign key(producer_id) references
singer_tb(singer_id),
    constraint produce_fk_music foreign key(produced_music) references music_tb(id)
);
SET @@foreign_key_checks=1;

-- 7. 创建专辑制作表，记录歌手发布专辑的状况

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS release_tb;
CREATE TABLE release_tb
(
    releaser_id BIGINT NOT NULL,
    released_album BIGINT NOT NULL,
    release_time DATE,
    primary key(releaser_id,released_album),
    constraint release_fk_user foreign key(releaser_id) references user_tb(id),
    constraint release_fk_album foreign key(released_album) references
album_tb(album_id)
);
SET @@foreign_key_checks=1;


-- 8. 创建播放表，记录不同用户对不同音乐的播放状态
```

```
SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS play_tb;
CREATE TABLE play_tb
(
    player_id BIGINT NOT NULL,
    played_music BIGINT NOT NULL,
    play_state ENUM('Pause','Start') NOT NULL,
    play_order ENUM('Order','Random') NOT NULL,
    primary key(player_id,played_music),
    constraint play_fk_user foreign key(player_id) references user_tb(id),
    constraint play_fk_music foreign key(played_music) references music_tb(id)
);
SET @@foreign_key_checks=1;

-- 9. 创建评论表，记录不同用户对不同音乐的评论

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS comment_music_tb;
CREATE TABLE comment_music_tb
(
    commentator_id BIGINT NOT NULL,
    commented_music BIGINT NOT NULL,
    comment_content VARCHAR(500) NOT NULL,
    primary key(commentator_id,commented_music),
    constraint comment_fk_user foreign key(commentator_id) references user_tb(id),
    constraint comment_fk_music foreign key(commented_music) references music_tb(id)
);
SET @@foreign_key_checks=1;

-- 10.创建关注表，记录用户对用户的关注
-- # 关注（关注用户ID，被关注用户ID，关注时间）

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS follow_tb;
CREATE TABLE follow_tb
(   subscriber_id BIGINT NOT NULL,
    publisher_id BIGINT NOT NULL,
    follow_time DATE,
    primary key(subscriber_id,publisher_id),
    constraint sub_fk_user foreign key(subscriber_id) references user_tb(id),
    constraint pub_fk_user foreign key(publisher_id) references user_tb(id)
);
SET @@foreign_key_checks=1;

-- 11. 创建歌曲表，歌曲继承自音乐，需要以音乐的id(music_id)为主键。

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS song_tb;
CREATE TABLE song_tb
(
    song_music_id BIGINT NOT NULL PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    CONSTRAINT song_fk_music_id FOREIGN KEY(song_music_id) REFERENCES music_tb(id)
);
```

```
SET @@foreign_key_checks=1;

-- 12. 创建MV表，MV继承自音乐，需要以音乐的id(music_id)为主键。

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS mv_tb;
CREATE TABLE mv_tb
(
    mv_music_id BIGINT NOT NULL PRIMARY KEY,
    video_title VARCHAR(20) NOT NULL,
    CONSTRAINT mv_fk_music_id FOREIGN KEY(mv_music_id) REFERENCES music_tb(id)
);
SET @@foreign_key_checks=1;

-- 13. 创建歌单表。歌单与用户的创建关系为一对多关系，因而需要指明创建的用户的id
-- (id_of_the_creating_user)和创造的时间(cerating_time)

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS song_list_tb;
CREATE TABLE song_list_tb
(
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(20) NOT NULL,
    sortord VARCHAR(20) NULL DEFAULT '默认排序',
    id_of_the_list_creating_user BIGINT NOT NULL,
    creating_time TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    CHECK(sortord IN ('默认排序', '歌曲名', '歌手', '专辑')),
    CONSTRAINT list_fk_create_list FOREIGN KEY(id_of_the_list_creating_user)
REFERENCES user_tb(id)
);

SET @@foreign_key_checks=1;

-- 14. 创建收藏表。收藏关系是用户和歌单的多对多关系，主键为用户id(user_id)和歌单
id(song_list_id)。

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS collect_tb;
CREATE TABLE collect_tb
(
    collect_user_id BIGINT NOT NULL,
    collect_song_list_id BIGINT NOT NULL,
    PRIMARY KEY(collect_user_id, collect_song_list_id),
    CONSTRAINT collect_fk_user FOREIGN KEY(collect_user_id) REFERENCES user_tb(id),
    CONSTRAINT collect_fk_list FOREIGN KEY(collect_song_list_id) REFERENCES
song_list_tb(id)
);
SET @@foreign_key_checks=1;

-- 15. 创建歌曲歌单表标识歌曲和歌单之间的属于关系。这个关系是多对多关系，需要以歌曲id(music_id)和
歌单id(song_list_id)为主键

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS song_belongs_to_song_list_tb;
CREATE TABLE song_belongs_to_song_list_tb
```

```
(
    belong_music_id BIGINT NOT NULL,
    belong_song_list_id BIGINT NOT NULL,
    PRIMARY KEY(belong_music_id, belong_song_list_id),
    CONSTRAINT song_fk_music FOREIGN KEY(belong_music_id) REFERENCES
song_tb(song_music_id),
    CONSTRAINT song_fk_list FOREIGN KEY(belong_song_list_id) REFERENCES
song_list_tb(id)
);
SET @@foreign_key_checks=1;

-- 16. 创建动态表。动态与用户的创建关系为一对多关系，因而需要指明创建的用户的id
-- (id_of_the_moment_creating_user)和发布动态的时间（creating_time）

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS moment_tb;
CREATE TABLE moment_tb
(
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    content VARCHAR(255) NOT NULL,
    range_of_share VARCHAR(20) NULL DEFAULT '所有人可见',
    id_of_the_moment_creating_user BIGINT NOT NULL,
    creating_time TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    CHECK(range_of_share IN ('所有人可见', '粉丝可见', '自己可见')),
    CONSTRAINT moment_fk_create_list FOREIGN KEY(id_of_the_moment_creating_user)
REFERENCES user_tb(id)
);
SET @@foreign_key_checks=1;

-- 17. 创建评论表。id是分辨符，belong_moment_id是强实体的主码。
-- “发送评论”是一对多联系，id_of_the_comment_user是对应的外码

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS comment_tb;
CREATE TABLE comment_tb
(
    id BIGINT NOT NULL AUTO_INCREMENT,
    time_of_comment TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    content VARCHAR(255) NOT NULL,
    belong_moment_id BIGINT NOT NULL,
    id_of_the_comment_user BIGINT NOT NULL,
    PRIMARY KEY(id, belong_moment_id),
    CONSTRAINT comment_belong_fk_moment_id FOREIGN KEY(belong_moment_id) REFERENCES
moment_tb(id),
    CONSTRAINT comment_fk_user_id FOREIGN KEY(id_of_the_comment_user) REFERENCES
user_tb(id)
);
SET @@foreign_key_checks=1;

-- 18. 创建点赞表。id是分辨符，belong_moment_id是强实体的主码。
-- “发送点赞”是一对多联系，id_of_the_liked_user是对应的外码

SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS liked_tb;
CREATE TABLE liked_tb
```

```
(
    id BIGINT NOT NULL AUTO_INCREMENT,
    belong_moment_id BIGINT NOT NULL,
    id_of_the_liked_user BIGINT NOT NULL,
    PRIMARY KEY(id, belong_moment_id),
    CONSTRAINT liked_belong_fk_moment_id FOREIGN KEY(belong_moment_id) REFERENCES
moment_tb(id),
    CONSTRAINT liked_fk_user_id FOREIGN KEY(id_of_the_liked_user) REFERENCES
user_tb(id)
);
SET @@foreign_key_checks=1;

-- 19. 创建转发表。id是分辨符，belong_moment_id是强实体的主码。
-- “进行转发”是一对多联系，id_of_the_trans_user是对应的外码
SET @@foreign_key_checks=0;
DROP TABLE IF EXISTS trans_tb;
CREATE TABLE trans_tb
(
    id BIGINT NOT NULL AUTO_INCREMENT,
    time_of_trans TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
    content VARCHAR(255) NOT NULL,
    belong_moment_id BIGINT NOT NULL,
    id_of_the_trans_user BIGINT NOT NULL,
    PRIMARY KEY(id, belong_moment_id),
    CONSTRAINT trans_belong_fk_moment_id FOREIGN KEY(belong_moment_id) REFERENCES
moment_tb(id),
    CONSTRAINT trans_bk_user_id FOREIGN KEY(id_of_the_trans_user) REFERENCES
user_tb(id)
);
SET @@foreign_key_checks=1;

-- 12+7=19
```

```
%%sql

/* 1. 插入用户表数据。*/

INSERT INTO user_tb (
    level_of_membership,
    user_phone,
    nickname
) VALUES (
    1,
    '18712345678',
    '张三'
);

INSERT INTO user_tb (
    level_of_membership,
    qq,
    nickname
) VALUES (
```

```
    4,
    976344301,
    '李四'
);

INSERT INTO user_tb (
    level_of_membership,
    mail,
    nickname
) VALUES (
    4,
    'testing@163.com',
    '王五'
);

INSERT INTO user_tb (
    level_of_membership,
    user_phone,
    nickname
) VALUES (
    5,
    '15530616876',
    '小北'
);

-- 2. 创建歌手表

INSERT INTO singer_tb(
    singer_id,
    singer_name,
    singer_district,
    singer_style
) VALUES (
    3,
    'Mili',
    '日本',
    '电子音乐'
);

INSERT INTO singer_tb(
    singer_id,
    singer_name,
    singer_district,
    singer_style
) VALUES (
    4,
    '北大学子',
    '北京',
    '经典'
);

-- 3. 创建私信表。

INSERT INTO private_message_tb (
    content,
```

```
    id_of_the_msg_accepted_user,
    id_of_the_msg_sent_user
) VALUES (
    'hello!',
    2,
    1
);

INSERT INTO private_message_tb (
    content,
    id_of_the_msg_accepted_user,
    id_of_the_msg_sent_user
) VALUES (
    'hi!',
    1,
    2
);

INSERT INTO private_message_tb (
    content,
    id_of_the_msg_accepted_user,
    id_of_the_msg_sent_user
) VALUES (
    'how do you do',
    3,
    1
);

-- 4. 创建专辑表。

INSERT INTO album_tb(
    album_id,
    album_name
) VALUES (
    1,
    '百大金曲'
);

INSERT INTO album_tb(
    album_id,
    album_name
) VALUES (
    2,
    'Miracle Milk'
);

-- 5. 创建音乐表。

INSERT INTO music_tb (
    id,music_name,
    belong_album_id
) VALUES (
    1,
    'world.execute(me);',
    2
```

```
);

INSERT INTO music_tb (
    id,
    music_name
) VALUES (
    2,
    'Creeper?'
);

INSERT INTO music_tb (
    id,
    music_name,
    belong_album_id
) VALUES (
    3,
    '清华大学校歌',
    1
);

INSERT INTO music_tb (
    id,
    music_name
) VALUES (
    4,
    'heal the world'
);

INSERT INTO music_tb (
    id,
    music_name,
    belong_album_id
) VALUES (
    5,
    '只因你太美',
    1
);

-- 6. 歌手制作音乐

INSERT INTO produce_tb(
    producer_id,
    produced_music
) VALUES (
    3,
    1
);

INSERT INTO produce_tb(
    producer_id,
    produced_music
) VALUES (
    3,
    2
);
```

```
INSERT INTO produce_tb(
    producer_id,
    produced_music
) VALUES (
    4,
    3
);

INSERT INTO produce_tb(
    producer_id,
    produced_music
) VALUES (
    3,
    4
);

INSERT INTO produce_tb(
    producer_id,
    produced_music
) VALUES (
    4,
    5
);

-- 7. 歌手发布专辑

INSERT INTO release_tb(
    releaser_id,
    released_album,
    release_time
) VALUES (
    4,
    1,
    '2023-04-01'
);

INSERT INTO release_tb(
    releaser_id,
    released_album,
    release_time
) VALUES (
    3,
    2,
    '2016-10-12'
);

-- 8. 创建播放表

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
```

```
    1,
    1,
    'Pause',
    'Order'
);

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    1,
    2,
    'Start',
    'Order'
);

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    2,
    1,
    'Start',
    'Random'
);

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    2,
    5,
    'Pause',
    'Order'
);

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    3,
    3,
    'Pause',
    'Random'
);

INSERT INTO play_tb(
```

```
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    3,
    4,
    'Start',
    'Order'
);

INSERT INTO play_tb(
    player_id,
    played_music,
    play_state,
    play_order
) VALUES (
    4,
    3,
    'Start',
    'Order'
);

-- 9. 创建评论表

INSERT INTO comment_music_tb(
    commentator_id,
    commented_music,
    comment_content
) VALUES (
    2,
    1,
    '这首歌的主题是计算机编程和模拟,通过使用编程术语和比喻来描述关系和情感。歌词中提到了对象创建、初始化和模拟等编程术语,同时也包含了对情感和关系的描绘,表达了在虚拟世界中寻求满足和幸福的渴望。这句话是ChatGPT回答的。'
);

INSERT INTO comment_music_tb(
    commentator_id,
    commented_music,
    comment_content
) VALUES (
    4,
    3,
    '还是我大PKU的燕园情好听。'
);

INSERT INTO comment_music_tb(
    commentator_id,
    commented_music,
    comment_content
) VALUES (
    3,
    5,
    '巅峰产生虚伪的拥护,黄昏见证虔诚的信徒!'
```

```
);

-- 11. 创建歌曲表。

INSERT INTO song_tb (
    song_music_id,
    name
) VALUES (
    1,
    'world.execute(me);'
);

INSERT INTO song_tb (
    song_music_id,
    name
) VALUES (
    2,
    'Creeper?'
);

INSERT INTO song_tb (
    song_music_id,
    name
) VALUES (
    3,
    '清华大学校歌'
);

INSERT INTO song_tb (
    song_music_id,
    name
) VALUES (
    5,
    '只因你太美'
);

-- 12. 创建MV表。

INSERT INTO mv_tb (
    mv_music_id,
    video_title
) VALUES (
    4,
    'heal the world'
);

-- 13. 创建歌单表。

INSERT INTO song_list_tb (
    title,
    id_of_the_list_creating_user
) VALUES (
    '自建歌单1',
    1
);
```

```
INSERT INTO song_list_tb (
    title,
    id_of_the_list_creating_user
) VALUES (
    '自建歌单2',
    1
);

INSERT INTO song_list_tb (
    title,
    id_of_the_list_creating_user
) VALUES (
    '自建歌单1',
    3
);

-- 14. 创建收藏表。

INSERT INTO collect_tb (
    collect_user_id,
    collect_song_list_id
) VALUES(
    1,
    1
);

INSERT INTO collect_tb (
    collect_user_id,
    collect_song_list_id
) VALUES(
    2,
    1
);

INSERT INTO collect_tb (
    collect_user_id,
    collect_song_list_id
) VALUES(
    2,
    2
);

INSERT INTO collect_tb (
    collect_user_id,
    collect_song_list_id
) VALUES(
    3,
    3
);

-- 15. 创建标识歌单属于关系的表。

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
```

```
    belong_song_list_id
) VALUES (
    1,
    1
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    2,
    1
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    2,
    2
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    3,
    2
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    1,
    3
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    2,
    3
);

INSERT INTO song_belongs_to_song_list_tb (
    belong_music_id,
    belong_song_list_id
) VALUES  (
    3,
    3
);

-- 16. 创建动态表。
```

```
INSERT INTO moment_tb (
    content,
    range_of_share,
    id_of_the_moment_creating_user
) VALUES (
    '今天去家园食堂吃了鸡腿饭',
    '粉丝可见',
    1
);

INSERT INTO moment_tb (
    content,
    range_of_share,
    id_of_the_moment_creating_user
) VALUES (
    '2023考研冲冲冲',
    '自己可见',
    1
);

INSERT INTO moment_tb (
    content,
    id_of_the_moment_creating_user
) VALUES (
    '许嵩又出新专辑',
    3
);

-- 17. 创建评论表。

INSERT INTO comment_tb (
    content,
    belong_moment_id,
    id_of_the_comment_user
) VALUES (
    '家园的鸡排饭也很不错',
    1,
    1
);

INSERT INTO comment_tb (
    content,
    belong_moment_id,
    id_of_the_comment_user
) VALUES (
    '考研加油',
    2,
    2
);

INSERT INTO comment_tb (
    content,
    belong_moment_id,
    id_of_the_comment_user
```

```
) VALUES (
    '许嵩粉丝报道',
    3,
    3
);

-- 18. 创建点赞表。

INSERT INTO liked_tb (
    belong_moment_id,
    id_of_the_liked_user
) VALUES (
    1,
    2
);

INSERT INTO liked_tb (
    belong_moment_id,
    id_of_the_liked_user
) VALUES (
    2,
    3
);

INSERT INTO liked_tb (
    belong_moment_id,
    id_of_the_liked_user
) VALUES (
    3,
    1
);

INSERT INTO liked_tb (
    belong_moment_id,
    id_of_the_liked_user
) VALUES (
    2,
    2
);

-- 19. 创建转发表。

INSERT INTO trans_tb (
    content,
    belong_moment_id,
    id_of_the_trans_user
) VALUES (
    '我们也去家园吃饭吧',
    1,
    2
);

INSERT INTO trans_tb (
    content,
    belong_moment_id,
```

```
    id_of_the_trans_user
) VALUES (
    '看看当代大学生内卷的现状吧',
    2,
    3
);

INSERT INTO trans_tb (
    content,
    belong_moment_id,
    id_of_the_trans_user
) VALUES (
    '姐妹们我买好票了',
    3,
    1
);
```

四、业务功能实现（SQL语句）

我们已经完成了关系表的创建，并插入了数据，现在我们借助PyMySQL，使用SQL语句来实现增删改查操作。
以下共列举了12种操作的方法，用数据查询来验证结果。

```
import pymysql
db = pymysql.connect(host='162.105.146.37', user='stu2100013111',
passwd='stu2100013111', port=43306, db = 'stu2100013111')
cursor = db.cursor()
```

```
# 1. 一个用户给另一个用户发送信息(更新发送表和收到信息的表)
user_id_1 = 1
user_id_2 = 2
message = 'Creeper!'
sql = (
    "INSERT INTO private_message_tb "
    "(content, id_of_the_msg_accepted_user, "
    "id_of_the_msg_sent_user) "
    "VALUES "
    "(%s, %s, %s)"
)
try:
    cursor.execute(sql, (message, user_id_1, user_id_2))
    print("send_message", cursor.fetchall())
    db.commit()
except:
    db.rollback()

# 收信人查看更新后的信息表
sql = (
    "SELECT * FROM private_message_tb "
```

```
    "WHERE id_of_the_msg_accepted_user = %s "
)
try:
    cursor.execute(sql, (user_id_1, ))
    print("the_recipient_views_the_updated_information", cursor.fetchall())
    db.commit()
except:
    db.rollback()

# 发信人查看更新后的信息表
sql = (
    "SELECT * FROM private_message_tb "
    "WHERE id_of_the_msg_sent_user = %s "
)
try:
    cursor.execute(sql, (user_id_2, ))
    print("the_sender_views_the_updated_information", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

send_message ()
the_recipient_views_the_updated_information ((2, datetime.datetime(2023, 4, 22, 15, 43, 48), 'hi!', 1, 2), (4, datetime.datetime(2023, 4, 22, 15, 44), 'Creeper!', 1, 2))
the_sender_views_the_updated_information ((2, datetime.datetime(2023, 4, 22, 15, 43, 48), 'hi!', 1, 2), (4, datetime.datetime(2023, 4, 22, 15, 44), 'Creeper!', 1, 2))

```
# 2. 用户创建歌单，并把对应的歌曲放进歌单
# 用户创建歌单
user_id = 3
title = '想考清华？听完这张歌单再去学习！'
sql = (
    "INSERT INTO song_list_tb "
    "(title, id_of_the_list_creating_user) "
    "VALUES (%s, %s)"
)
try:
    cursor.execute(sql, (title, user_id))
    print("creating_a_playlist", cursor.fetchall())
    db.commit()
except:
    db.rollback()

# 用户插入歌曲
song_list_id = 3
music_ids = [2, 3]
sql = (
    "INSERT INTO song_belongs_to_song_list_tb "
    "(belong_music_id, belong_song_list_id) "
    "VALUES (%s, %s)"
)
for music_id in music_ids:
```

```
    try:
        cursor.execute(sql, (music_id, song_list_id))
        print("Insert_songs", cursor.fetchall())
        db.commit()
    except:
        db.rollback()
        break

# 查看新建完成的歌单
song_list_id = 3
sql = (
    "SELECT song_tb.name FROM "
    "song_belongs_to_song_list_tb AS list "
    "JOIN song_tb ON list.belong_music_id = "
    "song_tb.song_music_id AND list.belong_song_list_id = %s"
)
try:
    cursor.execute(sql, (song_list_id, ))
    print("view_the_new_playlist", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

creating_a_playlist ()
view_the_new_playlist (('world.execute(me);',), ('Creeper?',), ('清华大学校歌',))

```
# 3. 按名字降序播放歌单里的歌曲
song_list_id = 3
sql = (
    "SELECT song_tb.name FROM "
    "song_belongs_to_song_list_tb AS list "
    "JOIN song_tb ON list.belong_music_id = "
    "song_tb.song_music_id AND list.belong_song_list_id = %s "
    "ORDER BY song_tb.name DESC"
)
try:
    cursor.execute(sql, (song_list_id, ))
    print("sort_the_songs_in_descending_order", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

sort_the_songs_in_descending_order (('清华大学校歌',), ('world.execute(me);',), ('Creeper?',))

```
# 4. 按昵称查找用户
sql = "SELECT * FROM user_tb WHERE nickname LIKE %s"

try:
    cursor.execute(sql, ("李四", ))
    print('Results:', cursor.fetchall())
```

```
except:
    print('Error')
```

Results: ((2, 4, None, 976344301, None, '李四'),)

```
# 5. 歌手注册
login='user_phone'
level_of_membership=5
login_key='15530616876'
nickname='小北'
sql=(
    "INSERT INTO user_tb "
    "(level_of_membership, %s, nickname) "
    "VALUES (%d, \'%s\', \'%s\')"
    % (login,level_of_membership,login_key,nickname)
)
try:
    cursor.execute(sql)
    print("user_registered", cursor.fetchall())
    db.commit()
except:
    db.rollback()

sql = "SELECT MAX(id) FROM user_tb"
now_id = 0
try:
    cursor.execute(sql)
    res = cursor.fetchone()
    now_id = res[0]
    db.commit()
except:
    db.rollback()

singer_name='北大学子'
singer_district='北京'
singer_style='经典'
sql=(
    "INSERT INTO singer_tb "
    "(singer_id, singer_name, singer_district, singer_style) "
    "VALUES (%d,\'%s\',\'%s\',\'%s\')"
    % (now_id,singer_name,singer_district,singer_style)
)
try:
    cursor.execute(sql)
    print("singer_registered", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

user_registered ()
singer_registered ()

```
# 6.歌手发布专辑
album_id=1
album_name='百大金曲'
belong_album_id=1
producer_id=3

# 发布专辑
release_time='2023-04-01'
sql=(
    "INSERT INTO release_tb "
    "(releaser_id,released_album,release_time) "
    "VALUES (%d,%d,\'%s\') "
    % (producer_id,album_id,release_time)
)
try:
    cursor.execute(sql)
    print("album_release", cursor.fetchall())
    db.commit()
except:
    db.rollback()

sql = "SELECT * FROM singer_tb"
cursor.execute(sql)
print(cursor.fetchall())

# 添加音乐3music3
music_id=3
sql=(
    "INSERT INTO produce_tb "
    "(producer_id,produced_music) "
    "VALUES (%d,%d)"
    % (producer_id,music_id)
)
print(sql)
try:
    cursor.execute(sql)
    print("produce_insert", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

album_release ()
((3, 'Mili', '日本', '电子音乐'), (4, '北大学子', '北京', '经典'), (5, '北大学子', '北京', '经典'))
INSERT INTO produce_tb (producer_id,produced_music) VALUES (3,3)
produce_insert ()

singer_id	singer_name	singer_district	singer_style
3	Mili	日本	电子音乐
4	北大学子	北京	经典

```
# 7.用户播放并评论音乐，可选择关注歌手

# 播放音乐
player_id=1
played_music=5
play_state='Start'
play_order='Order'
sql=(
    "INSERT INTO play_tb "
    "(player_id,played_music,play_state,play_order) "
    "VALUES (%d,%d,\'%s\',\'%s\')"
    % (player_id,played_music,play_state,play_order)
)
try:
    cursor.execute(sql)
    print("music_play", cursor.fetchall())
    db.commit()
except:
    db.rollback()

# 评论音乐
comment_content='立燕解清!'
sql=(
    "INSERT INTO comment_music_tb"
    "(commentator_id,commented_music,comment_content) "
    "VALUES (%d,%d,\'%s\')"
    % (player_id,played_music,comment_content)
)
try:
    cursor.execute(sql)
    print("music_comment", cursor.fetchall())
    db.commit()
except:
    db.rollback()

# 关注歌手
follow_time='2023-04-05'

producer_ids = []
# 查找创作本音乐的所有歌手
sql =(
    "select producer_id "
    "FROM produce_tb "
    "where produced_music=%d"
    % played_music
)
try:
    cursor.execute(sql)
    producer_ids = cursor.fetchall()
    db.commit()
except:
    db.rollback()

# 添加当前用户与所有歌手的关注关系
```

```
for producer_id in producer_ids:
    sql=(
        "INSERT INTO follow_tb"
        "(subscriber_id,publisher_id,follow_time)"
        "VALUES(%d,%d,\'%s\')"
        % (player_id,producer_id[0],follow_time)
    )
    try:
        cursor.execute(sql)
        print("singer_follow", cursor.fetchall())
        db.commit()
    except:
        db.rollback()
```

music_play ()
music_comment ()
singer_follow ()

```
# 8.查询查看过某个专辑音乐的所有用户
album_id=1

sql=(
    "select P.player_id "
    "from play_tb as P "
    "where exists ("
    "select * from music_tb as M "
    "where M.id=P.played_music "
    "and M.belong_album_id=%d) "
    % (album_id)
)
try:
    cursor.execute(sql)
    print("album_player",cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

album_player ((3,), (4,), (1,), (2,))

```
#9.用户增加一条动态，并查看更新前后用户的所有动态

#显示用户1所有动态
user_id = 1
sql = (
    "SELECT * FROM moment_tb "
    "WHERE id_of_the_moment_creating_user = %s "
)
try:
    cursor.execute(sql, (user_id, ))
    print("All moment", cursor.fetchall())
    db.commit()
```

```
except:
    db.rollback()

#用户1增加一条动态
new_content = "湖人总冠军"
new_range_of_share = "粉丝可见"
new_id_of_the_moment_creating_user = 1
sql=(
    "INSERT INTO moment_tb "
    "(content, range_of_share, id_of_the_moment_creating_user) "
    "VALUES (%s, %s, %s)"
)
try:
    cursor.execute(sql, (new_content, new_range_of_share,
new_id_of_the_moment_creating_user))
    print("Add new moment", cursor.fetchall())
    db.commit()
except:
    db.rollback()

#显示增加动态之后用户1的所有动态
sql = (
    "SELECT * FROM moment_tb "
    "WHERE id_of_the_moment_creating_user = %s "
)
try:
    cursor.execute(sql, (user_id, ))
    print("All moment", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

All moment ((1, '今天去家园食堂吃了鸡腿饭', '粉丝可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (2, '2023考研冲冲冲', '自己可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)))
Add new moment ()
All moment ((1, '今天去家园食堂吃了鸡腿饭', '粉丝可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (2, '2023考研冲冲冲', '自己可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (4, '湖人总冠军', '粉丝可见', 1, datetime.datetime(2023, 4, 22, 15, 45, 3)))

```
#10.查询用户1所有可见范围为"粉丝可见"的动态
user_id = 1
target_range_of_share = "粉丝可见"
sql = (
    "SELECT * FROM moment_tb "
    "WHERE id_of_the_moment_creating_user = %s and range_of_share = %s "
)
try:
    cursor.execute(sql, (user_id, target_range_of_share))
    print("Moment whose range of share is 粉丝可见:", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

Moment whose range of share is 粉丝可见: ((1, '今天去家园食堂吃了鸡腿饭', '粉丝可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (4, '湖人总冠军', '粉丝可见', 1, datetime.datetime(2023, 4, 22, 15, 45, 3)))

```
#11.将用户所有可见范围为"粉丝可见"的动态修改为"所有人可见"
new_range_of_share = "所有人可见"
old_range_of_share = "粉丝可见"
sql = (
    "UPDATE moment_tb "
    "SET range_of_share = %s "
    "WHERE range_of_share = %s "
)
try:
    cursor.execute(sql, (new_range_of_share, old_range_of_share))
    print("Update moment_tb:", cursor.fetchall())
    db.commit()
except:
    db.rollback()

#显示所有可见范围为“所有人可见”的动态
target_range_of_share = "所有人可见"
sql = (
    "SELECT * FROM moment_tb "
    "WHERE range_of_share = %s "
)
try:
    cursor.execute(sql, (target_range_of_share))
    print("Moment whose range of share is 所有人可见:", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

Update moment_tb: ()
Moment whose range of share is 所有人可见: ((1, '今天去家园食堂吃了鸡腿饭', '所有人可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (3, '许嵩又出新专辑', '所有人可见', 3, datetime.datetime(2023, 4, 22, 15, 43, 48)), (4, '湖人总冠军', '所有人可见', 1, datetime.datetime(2023, 4, 22, 15, 45, 3)))

```
#12.删除可见范围为"所有人可见"的动态的所有评论

#显示所有评论
sql = (
    "SELECT * FROM comment_tb "
)
try:
    cursor.execute(sql)
    print("All comments:", cursor.fetchall())
    db.commit()
except:
```

```
    db.rollback()

sql = (
    "SELECT * FROM moment_tb "
)
try:
    cursor.execute(sql)
    print("All moments:", cursor.fetchall())
    db.commit()
except:
    db.rollback()

#删除可见范围为“所有人可见”的动态的所有评论
delete_range_of_share = "所有人可见"
sql = (
    "DELETE  "
    "FROM comment_tb "
    "WHERE belong_moment_id in (SELECT id FROM moment_tb WHERE range_of_share = %s
)"
)
try:
    cursor.execute(sql, (delete_range_of_share, ))
    print("Delete the moments whose range of share is 粉丝可见:", cursor.fetchall())
    db.commit()
except:
    db.rollback()

#显示删除之后的所有评论
sql = (
    "SELECT * FROM comment_tb "
)
try:
    cursor.execute(sql)
    print("All comments:", cursor.fetchall())
    db.commit()
except:
    db.rollback()
```

All comments: ((1, datetime.datetime(2023, 4, 22, 15, 43, 48), '家园的鸡排饭也很不错', 1, 1), (2, datetime.datetime(2023, 4, 22, 15, 43, 48), '考研加油', 2, 2), (3, datetime.datetime(2023, 4, 22, 15, 43, 48), '许嵩粉丝报道', 3, 3))
All moments: ((1, '今天去家园食堂吃了鸡腿饭', '所有人可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (2, '2023考研冲冲冲', '自己可见', 1, datetime.datetime(2023, 4, 22, 15, 43, 48)), (3, '许嵩又出新专辑', '所有人可见', 3, datetime.datetime(2023, 4, 22, 15, 43, 48)), (4, '湖人总冠军', '所有人可见', 1, datetime.datetime(2023, 4, 22, 15, 45, 3)))
Delete the moments whose range of share is 粉丝可见: ()
All comments: ((2, datetime.datetime(2023, 4, 22, 15, 43, 48), '考研加油', 2, 2),)