评分系统数据库项目报告

汪之立　2200012966

陈麟丰　2200012872

马天麟　2200012907

一、 业务需求

我们主要实现的是类似虎扑评分的一个评分系统，保留了其核心业务功能，具体罗列如下：

1. **主题系统**
   - 用户可在某一主题下发帖（如 NBA、CBA、影视、数码）。
   - 每个帖子可关联一个或多个主题。
2. **帖子**
   - 帖子标题。
   - 帖子可被用户收藏。
   - 帖子可关联多个被评分对象（如球员、电影、手机等）。
3. **用户系统**
   - 用户拥有唯一身份，可发帖、评分、评论。
   - 用户行为会被记录：发帖、评论、打分、点赞、收藏等。
4. **评分系统**
   - 用户可以对评分对象打分（0-10 分）。
   - 支持查看平均分、打分人数、分布图等统计信息。
5. **评分对象**
   - 可以是球员、球队、电影、数码产品等。
   - 每种对象类型属性：名称，简介。
6. **评论系统**
   - 评论可以关联一个评分。
   - 评论可嵌套（楼中楼结构）。
   - 评论可点赞。

二、 ER 图设计

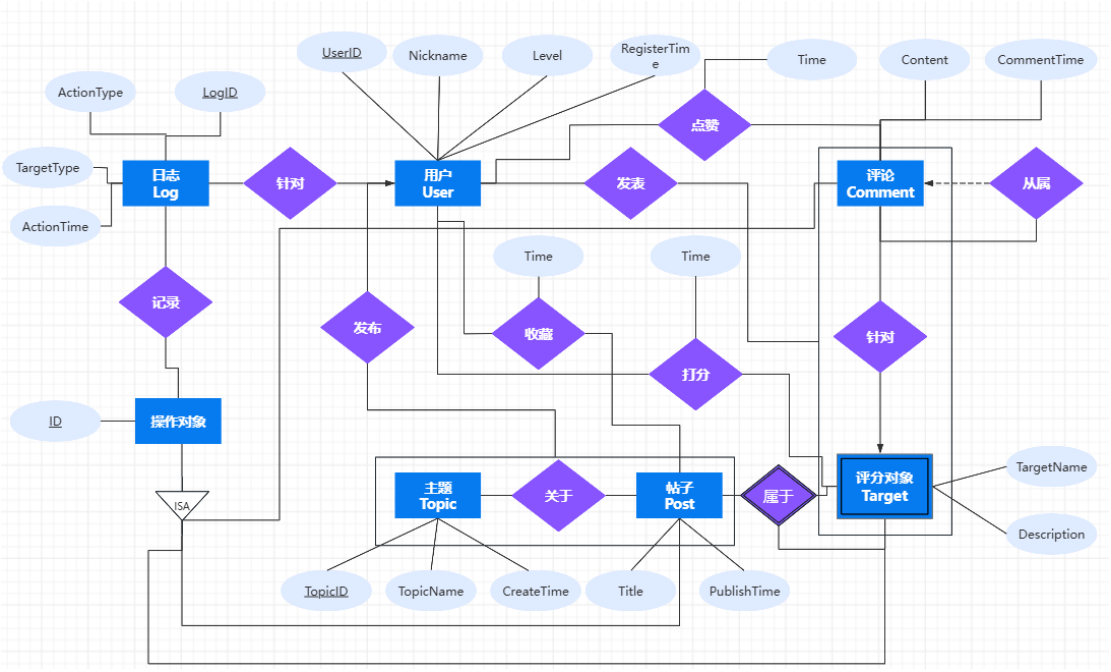
首先提炼出一些关键的实体和联系：

实体/联系	描述	主要属性
User	用户	id, 昵称, 等级, 注册时间
Post	帖子	id, user_id, 标题, 发布时间
Topic	主题/分类	id, 名称
Comment	评论	id, post_id, user_id, content, parent_comment_id, 时间戳, target_id
ScoreTarget	被评分对象	id, 名称, 简介, post_id
ScoreRecord	用户评分记录	user_id, target_id, post_id ,score, 时间戳
UserActionLog	行为记录	user_id, 动作类型, 目标 id, 时间戳
FavoritePost	收藏记录	user_id, post_id
Like	点赞	user_id, comment_id

2. 核心联系

联系类型	参与实体	特性
发帖	用户 → 帖子	1:N（一个用户可发多帖）
主题关联	帖子 ↔ 主题	M:N（多对多，通过PostTopic表实现）
评分对象绑定	帖子 → 评分对象	1:N（每个评分对象仅属于一个帖子）
评分记录	用户 ↔ 评分对象	M:N（通过ScoreRecord表实现，含评分值）
评论隶属	帖子 → 评论	1:N（评论必须属于一个帖子）
评论层级	评论 → 评论	1:N（楼中楼结构）
行为关联	用户 → 日志	1:N（用户行为记录）

合并整理后绘制图像



三、 SQL 定义和业务功能实现

In [230…

```
import sqlite3

DB_CONNECTION = None
```

In [231…

```
def createDB(path = ':memory:'):
    conn = sqlite3.connect(path)
    conn.execute("PRAGMA foreign_keys = 1")
    cursor = conn.cursor()
    cursor.executescript('''
    -- 用户表：存储平台用户核心信息
    CREATE TABLE User (
    UserID          INTEGER PRIMARY KEY AUTOINCREMENT,  -- 自增主键
    Nickname        VARCHAR(50) NOT NULL UNIQUE,        -- 昵称唯一约束
    Level           INTEGER DEFAULT 1 CHECK(Level > 0),-- 等级最小为1
    RegisterTime    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
    );

    -- 主题分类表：内容分类（如NBA/影视）
    CREATE TABLE Topic (
    TopicID         INTEGER PRIMARY KEY AUTOINCREMENT,
    TopicName       VARCHAR(30) NOT NULL UNIQUE,        -- 主题名唯一
    CreateTime      DATETIME DEFAULT CURRENT_TIMESTAMP -- 新增创建时间字段
    );

    -- 帖子表：用户发布的主题帖
    CREATE TABLE Post (
    PostID          INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID          INTEGER NOT NULL,
    Title           VARCHAR(200) NOT NULL CHECK(LENGTH(Title) >= 5), -- 标题长度限制
    PublishTime     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
    );

    -- 帖子-主题关联表（多对多关系）
    CREATE TABLE PostTopic (
    PostID          INTEGER NOT NULL,
    TopicID         INTEGER NOT NULL,
    PRIMARY KEY (PostID, TopicID),
    FOREIGN KEY (PostID) REFERENCES Post(PostID) ON DELETE CASCADE,
    FOREIGN KEY (TopicID) REFERENCES Topic(TopicID) ON DELETE RESTRICT
    );

    -- 评分对象表：需与帖子严格绑定
    CREATE TABLE ScoreTarget (
    TargetID        INTEGER PRIMARY KEY AUTOINCREMENT,
    PostID          INTEGER NOT NULL,                   -- 强制绑定到一个帖子
    TargetName      VARCHAR(100) NOT NULL CHECK(LENGTH(TargetName) >= 2),
    Description     TEXT,
    FOREIGN KEY (PostID) REFERENCES Post(PostID) ON DELETE CASCADE
    );

    -- 评分记录表：用户对对象的打分
    CREATE TABLE ScoreRecord (
    RecordID        INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID          INTEGER NOT NULL,
    TargetID        INTEGER NOT NULL,
    PostID          INTEGER NOT NULL,                   -- 冗余存储便于查询
    Score           DECIMAL(3,1) NOT NULL CHECK(Score BETWEEN 0 AND 10),
    RecordTime      DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (TargetID) REFERENCES ScoreTarget(TargetID) ON DELETE CASCADE,
    FOREIGN KEY (PostID) REFERENCES Post(PostID) ON DELETE CASCADE,
```

```
    UNIQUE (UserID, TargetID)                          -- 防止重复评分
    );

    -- 评论表：支持楼中楼结构
    CREATE TABLE Comment (
    CommentID       INTEGER PRIMARY KEY AUTOINCREMENT,
    PostID          INTEGER NOT NULL,
    UserID          INTEGER NOT NULL,
    Content         TEXT NOT NULL CHECK(LENGTH(Content) >= 5),
    ParentID        INTEGER,                           -- 父评论ID实现嵌套
    TargetID        INTEGER,                           -- 可选关联评分对象
    CommentTime     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Post(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (ParentID) REFERENCES Comment(CommentID) ON DELETE CASCADE,
    FOREIGN KEY (TargetID) REFERENCES ScoreTarget(TargetID) ON DELETE SET NULL
    );

    -- 用户行为日志表（审计用）
    CREATE TABLE UserActionLog (
    LogID           INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID          INTEGER NOT NULL,
    ActionType      VARCHAR(20) NOT NULL CHECK(ActionType IN (
        'CREATE_POST','COMMENT','SCORE','LIKE','FAVORITE')), -- 枚举约束
    TargetType      VARCHAR(20) NOT NULL CHECK(TargetType IN (
        'post','comment','target')),
    TargetID        INTEGER NOT NULL,
    ActionTime      DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
    );

    -- 帖子收藏表
    CREATE TABLE FavoritePost (
    UserID          INTEGER NOT NULL,
    PostID          INTEGER NOT NULL,
    FavoriteTime    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (UserID, PostID),
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (PostID) REFERENCES Post(PostID) ON DELETE CASCADE
    );

    -- 评论点赞表
    CREATE TABLE Like (
    UserID          INTEGER NOT NULL,
    CommentID       INTEGER NOT NULL,
    LikeTime        DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (UserID, CommentID),
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (CommentID) REFERENCES Comment(CommentID) ON DELETE CASCADE
    );
    ''')

    # 提交事务
    conn.commit()

    return conn
```

In [232…

```
def initDB(path =  ':memory:'):
    global DB_CONNECTION
    assert DB_CONNECTION is None
    DB_CONNECTION = createDB(path)
```

In [233…

```
def shutdownDB():
    global DB_CONNECTION
    assert DB_CONNECTION is not None
    DB_CONNECTION.close()
    DB_CONNECTION = None
```

In [234…

```
def getDB():
    global DB_CONNECTION
    assert DB_CONNECTION is not None
    return DB_CONNECTION
```

In [235…

```
def init_sample_data():
    conn = getDB()
    cursor = conn.cursor()

    # 清空现有数据（测试用）
    cursor.execute("DELETE FROM Like")
    cursor.execute("DELETE FROM FavoritePost")
    cursor.execute("DELETE FROM UserActionLog")
    cursor.execute("DELETE FROM Comment")
    cursor.execute("DELETE FROM ScoreRecord")
    cursor.execute("DELETE FROM ScoreTarget")
    cursor.execute("DELETE FROM PostTopic")
    cursor.execute("DELETE FROM Post")
    cursor.execute("DELETE FROM Topic")
    cursor.execute("DELETE FROM User")

    # 1. 用户数据
    users = [
        ('虎扑JR123', 5),
        ('湖人总冠军', 3),
        ('数码评测君', 8),
        ('电影爱好者', 2),
        ('CBA观察员', 4)
    ]
    cursor.executemany('''
    INSERT INTO User (Nickname, Level)
    VALUES (?, ?)
    ''', users)

    # 2. 主题数据
    topics = [
        ('NBA', '2023-01-01 00:00:00'),
        ('CBA', '2023-01-02 00:00:00'),
        ('影视', '2023-01-03 00:00:00'),
        ('数码', '2023-01-04 00:00:00')
    ]
    cursor.executemany('''
    INSERT INTO Topic (TopicName, CreateTime)
    VALUES (?, ?)
    ''', topics)

    # 3. 帖子数据（每个用户至少2个帖子）
    posts = [
        (1, '湖人vs勇士全场集锦！詹姆斯关键三分', '2023-05-01 19:30:00'),
        (1, '约基奇最新赛季数据分析', '2023-05-02 14:00:00'),
        (2, 'iPhone15上手实测报告', '2023-05-03 10:15:00'),
        (3, '《流浪地球2》深度影评', '2023-05-04 20:45:00'),
        (4, 'CBA总决赛辽宁vs浙江前瞻', '2023-05-05 09:00:00'),
        (5, '小米13 Ultra相机评测', '2023-05-06 16:20:00')
    ]
    cursor.executemany('''
    INSERT INTO Post (UserID, Title, PublishTime)
```

```
    VALUES (?, ?, ?)
    ''', posts)

    # 4. 帖子-主题关联
    post_topics = [
        (1, 1),  # NBA
        (2, 1),  # NBA
        (3, 4),  # 数码
        (4, 3),  # 影视
        (5, 2),  # CBA
        (6, 4)   # 数码
    ]
    cursor.executemany('''
    INSERT INTO PostTopic (PostID, TopicID)
    VALUES (?, ?)
    ''', post_topics)

    # 5. 评分对象（每个帖子2个对象）
    score_targets = [
        (1, '勒布朗·詹姆斯', '湖人队核心球员'),
        (1, '斯蒂芬·库里', '勇士队当家球星'),
        (3, 'iPhone15 Pro', '苹果最新旗舰手机'),
        (3, '三星S23 Ultra', '安卓机皇'),
        (4, '《流浪地球2》', '中国科幻大片'),
        (5, '郭艾伦', '辽宁队后卫'),
        (6, '小米13 Ultra', '徕卡影像旗舰')
    ]
    cursor.executemany('''
    INSERT INTO ScoreTarget (PostID, TargetName, Description)
    VALUES (?, ?, ?)
    ''', score_targets)

    # 6. 评分记录（每个对象3-5个评分）
    score_records = [
        (1, 1, 1, 9.5),
        (2, 1, 1, 8.8),
        (3, 1, 2, 9.0),
        (1, 3, 3, 9.2),
        (2, 3, 3, 8.5),
        (4, 3, 3, 7.9),
        (3, 4, 4, 9.7),
        (5, 6, 6, 9.4)
    ]
    cursor.executemany('''
    INSERT INTO ScoreRecord (UserID, TargetID, PostID, Score)
    VALUES (?, ?, ?, ?)
    ''', score_records)

    # 7. 评论数据（含楼中楼）
    comments = [
        (1, 1, '詹姆斯今天太神了！', None, 1),
        (2, 1, '库里三分还是稳', None, 2),
        (3, 1, '裁判有几个判罚有问题', 1, 1),
        (4, 3, 'iPhone的录像功能确实强', None, 3),
        (6, 5, '小米这次影像进步很大', None, 7)
    ]
    cursor.executemany('''
    INSERT INTO Comment (PostID, UserID, Content, ParentID, TargetID)
    VALUES (?, ?, ?, ?, ?)
    ''', comments)

    # 8. 收藏记录
    favorites = [
        (2, 1),
```

```
        (3, 3),
        (4, 4),
        (5, 6)
    ]
    cursor.executemany('''
    INSERT INTO FavoritePost (UserID, PostID)
    VALUES (?, ?)
    ''', favorites)

    # 9. 点赞记录
    likes = [
        (2, 1),
        (3, 4),
        (4, 5)
    ]
    cursor.executemany('''
    INSERT INTO Like (UserID, CommentID)
    VALUES (?, ?)
    ''', likes)

    # 10. 行为日志（自动生成）
    actions = [
        (1, 'CREATE_POST', 'post', 1),
        (2, 'COMMENT', 'comment', 1),
        (3, 'SCORE', 'target', 1),
        (4, 'FAVORITE', 'post', 3),
        (5, 'LIKE', 'comment', 5)
    ]
    cursor.executemany('''
    INSERT INTO UserActionLog (UserID, ActionType, TargetType, TargetID)
    VALUES (?, ?, ?, ?)
    ''', actions)

    conn.commit()
```

In [236…

```
def displayDB(conn, callback):
    import pandas as pd
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' ORDER BY name;
    table_name = cursor.fetchall()
    for table in table_name:
        df = pd.read_sql_query(f"SELECT * FROM {table[0]}", con=conn)
        callback(df)
```

In [237…

```
def check_value(param, valid_values):
    def _check_value(param, valid_values):
        if param not in valid_values:
            raise ValueError(f"{param} is not supported, it should be the " f"subset

    if isinstance(param, list):
        for p in param:
            _check_value(p, valid_values)
    else:
        _check_value(param, valid_values)
    return param
```

In [238…

```
def check_type(param, types, elem = False):
    def _check_type(param, types):
        if type(param).__name__ not in types:
            raise TypeError(f"{param} is of type{type(param)} not " f"supported, sho

    if isinstance(param, list) and elem:
        for p in param:
```

```
            _check_type(p, types)
    else:
        _check_type(param, types)
    return param
```

In [239…

```
initDB()
print("Database initialized.")
init_sample_data()
print("Sample data inserted.")
```

```
Database initialized.
Sample data inserted.
```

In [240…

```
def handle_reg_user(request, conn):
    user_nickname = check_type(request["user_nickname"], ["str"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    INSERT INTO User (Nickname)
    VALUES (?)
    RETURNING UserID
    ''', (user_nickname,)
    )
    result = cursor.fetchone()
    conn.commit()
    cursor.close()
    return {
        "user_id" : result[0]
    }
```

In [241…

```
# 1. 用户注册（示例新增用户）
request1 = {"user_nickname": "新用户"}
response1 = handle_reg_user(request1, getDB())
print("新用户ID:", response1["user_id"])  # 输出: 6
```

```
新用户ID: 6
```

In [242…

```
def handle_reg_topic(request, conn):
    topic_name = check_type(request["topic_name"], ["str"])
    cursor = conn.cursor()
    cursor.execute(
    '''
    INSERT INTO Topic (TopicName)
    VALUES (?)
    RETURNING TopicID
    ''',( topic_name,)
    )
    result = cursor.fetchone()
    conn.commit()
    cursor.close()
    return {
        "topic_id": result[0]
    }
```

In [243…

```
# 2. 创建主题（示例新增主题）
request2 = {"topic_name": "游戏"}
response2 = handle_reg_topic(request2, getDB())
print("新主题ID:", response2["topic_id"])  # 输出: 5
```

```
新主题ID: 5
```

In [244…

```
def handle_post_post(request, conn):
    user_id = check_type(request["user_id"], ["str", "int"])
    topic_id = check_type(request["topic_id"], ["str", "int"], True)
```

```
    title = check_type(request["title"], ["str"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    INSERT INTO Post (UserID, Title)
    VALUES (?, ?)
    RETURNING PostID
    ''', (int(user_id),title,)
    )
    result = cursor.fetchone()
    conn.commit()
    cursor.close()
    for topic in topic_id:
        cursor = conn.cursor()
        cursor.execute(
        '''
        INSERT INTO PostTopic (PostID, TopicID)
        VALUES (?, ?)
        ''',(result[0], int(topic),)
        )
    cursor = conn.cursor()
    cursor.execute(
    '''
    INSERT INTO UserActionLog (UserID, ActionType, TargetType, TargetID)
    VALUES (?, ?, ?, ?)
    ''',(int(user_id), "CREATE_POST", "post", int(result[0]))
    )
    return {
        "post_id": result[0]
    }
```

In [245…

```
# 3. 发布帖子（用户1在NBA主题发帖）
request3 = {
    "user_id": 1,
    "topic_id": [1],  # NBA主题
    "title": "东契奇绝杀集锦"
}
response3 = handle_post_post(request3, getDB())
print("新帖子ID:", response3["post_id"])  # 输出: 7
```

新帖子ID: 7

In [246…

```
def handle_post_target(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])
    name = check_type(request["name"], ["str"])
    description = check_type(request["description"], ["str"])
    cursor = conn.cursor()
    cursor.execute(
    '''
    INSERT INTO ScoreTarget (PostID, TargetName, Description)
    VALUES (?, ?, ?)
    RETURNING TargetID
    ''',
    (int(post_id), name, description)
    )
    result = cursor.fetchone()
    conn.commit()
    cursor.close()
    return {
        "target_id" : result[0]
    }
```

In [247…

```
# 4. 添加评分对象（在新帖添加对象）
request4 = {
```

```
    "post_id": 7,
    "name": "卢卡·东契奇",
    "description": "独行侠核心球员"
}
response4 = handle_post_target(request4, getDB())
print("新评分对象ID:", response4["target_id"])  # 输出: 8
```

新评分对象ID: 8

In [248…

```
def handle_post_comment_score(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])
    target_id = check_type(request["target_id"], ["str", "int"])
    user_id = check_type(request["user_id"], ["str", "int"])
    content = check_type(request["content"], "str")
    score = check_type(request["score"], ["int"])

    cursor = conn.cursor()
    cursor.execute(
    '''
    INSERT INTO ScoreRecord (UserID, TargetID, PostID, Score)
    VALUES (?, ?, ?, ?)
    RETURNING RecordID
    '''
    ,(int(user_id), int(target_id), int(post_id), score,)
    )
    result1 = cursor.fetchone()
    cursor.execute(
    '''
    INSERT INTO Comment (PostID, UserID, Content, TargetID)
    VALUES (?, ?, ?, ?)
    RETURNING CommentID
    '''
    ,(int(post_id), int(user_id), content, int(target_id))
    )
    result2 = cursor.fetchone()
    cursor.execute(
    '''
    INSERT INTO UserActionLog (UserID, ActionType, TargetType, TargetID)
    VALUES (?, ?, ?, ?)
    ''',(int(user_id), "SCORE", "target", int(target_id))
    )
    cursor.execute(
    '''
    INSERT INTO UserActionLog (UserID, ActionType, TargetType, TargetID)
    VALUES (?, ?, ?, ?)
    ''',(int(user_id), "COMMENT", "comment", int(result2[0]))
    )
    conn.commit()
    cursor.close()
    return {
        "record_id" : result1[0],
        "comment_id": result2[0]
    }
```

In [249…

```
# 5. 发表带评分的评论（用户2给东契奇打分）
request5 = {
    "post_id": 7,
    "target_id": 8,
    "user_id": 2,
    "content": "东契奇关键球太稳了！",
    "score": 8
}
response5 = handle_post_comment_score(request5, getDB())
print("新评分记录:", response5)  # 输出包含record_id和comment_id
```

新评分记录: {'record_id': 9, 'comment_id': 6}

In [250…

```
def handle_post_comment_comment(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])
    target_id = check_type(request["target_id"], ["str", "int"])
    user_id = check_type(request["user_id"], ["str", "int"])
    content = check_type(request["content"], ["str"])
    parent = check_type(request["parent"], ["str", "int"])
    cursor = conn.cursor()
    cursor.execute(
    '''
    INSERT INTO Comment (PostID, UserID, Content, TargetID, ParentID)
    VALUES (?, ?, ?, ?, ?)
    RETURNING CommentID
    '''
    ,(int(post_id), int(user_id), content, int(target_id), int(parent))
    )
    result2 = cursor.fetchone()
    cursor.execute(
    '''
    INSERT INTO UserActionLog (UserID, ActionType, TargetType, TargetID)
    VALUES (?, ?, ?, ?)
    ''',(int(user_id), "COMMENT", "comment", int(result2[0]))
    )
    conn.commit()
    cursor.close()
    return {
        "comment_id": result2[0]
    }
```

In [251…

```
# 6. 回复评论（用户3回复上条评论）
request6 = {
    "post_id": 7,
    "target_id": 8,
    "user_id": 3,
    "content": "确实，比库里强多了",
    "parent": response5["comment_id"]
}
response6 = handle_post_comment_comment(request6, getDB())
print("回复评论ID:", response6["comment_id"])
```

回复评论ID: 7

In [252…

```
def handle_req_topic_posts(request, conn):
    topic_id = check_type(request["topic_id"], ["str", "int"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    SELECT Post.*
    FROM Post
    JOIN PostTopic USING(PostID)
    WHERE PostTopic.TopicID = {topic_id}
    '''
    )
    result = cursor.fetchall()
    conn.commit()
    cursor.close()
    return [
            {
                "post_id": a,
                "user_id": b,
                "title": c,
                "post_time": d
            }
```

```
            for a,b,c,d in result
        ]
```

In [253…

```
# 7. 查询NBA主题帖子
request7 = {"topic_id": 1}
response7 = handle_req_topic_posts(request7, getDB())
print("NBA主题帖子:")
for post in response7:  # 应包含帖子1/2/7
    print(f"{post['post_id']}: {post['title']}")
```

NBA主题帖子:
1: 湖人vs勇士全场集锦！詹姆斯关键三分
2: 约基奇最新赛季数据分析
7: 东契奇绝杀集锦

In [254…

```
def handle_req_target_avgscore(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])
    target_id =check_type(request["target_id"], ["str", "int"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    SELECT AVG(Score)
    FROM ScoreRecord
    WHERE ScoreRecord.PostID == {str(post_id)} and ScoreRecord.TargetID == {str(targ
    '''
    )
    result = cursor.fetchone()
    conn.commit()
    cursor.close()
    return {
        "score": result[0]
    }
```

In [255…

```
# 8. 查询詹姆斯平均分（帖子1）
request8 = {"post_id": 1, "target_id": 1}
response8 = handle_req_target_avgscore(request8, getDB())
print("詹姆斯平均分:", response8["score"])  # (9.5+8.8)/2=9.15
```

詹姆斯平均分: 9.15

In [256…

```
def handle_req_target_comments(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])
    target_id =check_type(request["target_id"], ["str", "int"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    SELECT c.CommentID, c.UserID, c.Content, c.ParentID, c.CommentTime
    FROM Comment AS c
    WHERE c.PostID = {str(post_id)} and c.TargetID = {str(target_id)}
    '''
    )
    val = cursor.fetchall()
    conn.commit()
    cursor.close()
    return [
        {
            "comment_id": a,
            "user_id": b,
            "content": c,
            "parent_id": d,
            "comment_time": e
        }
```

```
        for a,b,c,d,e in val
    ]
```

In [257…

```
# 9. 获取iPhone15的评论（帖子3）
request9 = {"post_id": 3, "target_id": 3}
response9 = handle_req_target_comments(request9, getDB())
print("iPhone15评论数:", len(response9))  # 初始有1条
```

iPhone15评论数: 0

In [258…

```
def handle_req_user_actions(request, conn):
    user_id = check_type(request["user_id"], ["str", "int"])
    cursor = conn.cursor()
    cursor.execute(
    f'''
    SELECT u.ActionType, u.LogID, u.ActionTime
    FROM UserActionLog AS u
    WHERE u.UserID = {str(user_id)}
    '''
    )
    val = cursor.fetchall()
    conn.commit()
    cursor.close()
    return [
        {
            "action_type": a,
            "action_id": b,
            "action_time": c,
        }
        for a,b,c in val
    ]
```

In [259…

```
# 10. 查询用户1的行为记录
request10 = {"user_id": 1}
response10 = handle_req_user_actions(request10, getDB())
print("用户1最近行为:", [a["action_type"] for a in response10])  # 包含CREATE_POST/S
```

用户1最近行为: ['CREATE_POST', 'CREATE_POST']

In [260…

```
# 11. 修改评分（无需记录日志）
def handle_update_score(request, conn):
    record_id = check_type(request["record_id"], ["str", "int"])
    user_id = check_type(request["user_id"], ["str", "int"])
    new_score = check_type(request["new_score"], ["int", "float"])

    cursor = conn.cursor()
    # 直接更新评分记录
    cursor.execute('''
    UPDATE ScoreRecord
    SET Score = ?
    WHERE RecordID = ?
      AND UserID = ?
    RETURNING TargetID, PostID
    ''', (float(new_score), int(record_id), int(user_id)))

    updated = cursor.fetchone()
    if not updated:
        raise ValueError("评分记录不存在或无权修改")

    conn.commit()  # 移除了日志记录
    return {"target_id": updated[0], "post_id": updated[1]}
```

In [261…

```
# 11. 修改评分（用户2修改对东契奇的评分）
request_update = {
    "record_id": 5,  # 假设存在的评分记录ID
    "user_id": 2,
    "new_score": 9.0
}
resp_update = handle_update_score(request_update, getDB())
print("修改结果:", resp_update)  # 应返回目标ID和帖子ID
```

修改结果: {'target_id': 3, 'post_id': 3}

In [262…

```
# 12. 删除评论（级联删除子评论）
def handle_delete_comment(request, conn):
    comment_id = check_type(request["comment_id"], ["str", "int"])
    user_id = check_type(request["user_id"], ["str", "int"])

    cursor = conn.cursor()
    # 直接删除评论（通过外键级联）
    cursor.execute('''
    DELETE FROM Comment
    WHERE CommentID = ?
      AND UserID = ?
    RETURNING PostID, TargetID
    ''', (int(comment_id), int(user_id)))

    deleted = cursor.fetchone()
    if not deleted:
        raise ValueError("评论不存在或无权删除")

    conn.commit()  # 移除了日志记录
    return {"post_id": deleted[0], "target_id": deleted[1]}
```

In [263…

```
# 12. 删除评论（用户3删除自己的回复）
# 测试删除评论
request_delete = {
    "comment_id": 3,  # 假设存在的评论ID
    "user_id": 1
}
resp_delete = handle_delete_comment(request_delete, getDB())
print("删除结果:", resp_delete)  # 应返回关联的帖子ID和目标ID
```

删除结果: {'post_id': 3, 'target_id': 1}

In [264…

```
# 13. 查询帖子中评论最多的评分对象
def handle_req_hot_target(request, conn):
    post_id = check_type(request["post_id"], ["str", "int"])

    cursor = conn.cursor()
    cursor.execute('''
    SELECT
        st.TargetID,
        st.TargetName,
        COUNT(c.CommentID) AS comment_count
    FROM Comment c
    JOIN ScoreTarget st ON c.TargetID = st.TargetID
    WHERE c.PostID = ?
      AND c.TargetID IS NOT NULL
    GROUP BY st.TargetID
    ORDER BY comment_count DESC
    LIMIT 1
    ''', (int(post_id),))

    result = cursor.fetchone()
    return {
```

```
        "target_id": result[0],
        "target_name": result[1],
        "comment_count": result[2]
    } if result else {}
```

In [265…

```
# 13. 查询帖子1的热门对象
request13 = {"post_id": 1}
response13 = handle_req_hot_target(request13, getDB())
print("帖子1最热对象:", response13)  # 应返回詹姆斯（2条评论）
```

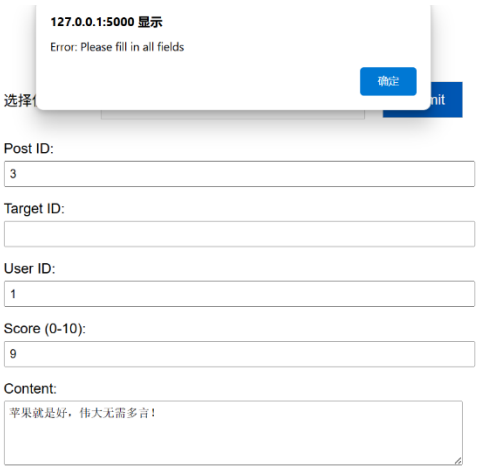
帖子1最热对象: {'target_id': 1, 'target_name': '勒布朗·詹姆斯', 'comment_count': 1}

四、 前端页面展示

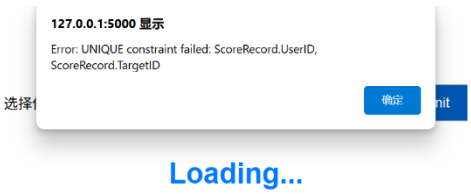
可以通过下拉菜单选择所需的场景，在对应的表单中填写有效的信息后点击提交按钮即可。得到结果后点击返回重新回到表单。

1. 成功页面展示（发送内容和查询内容）

2. 前端错误检测（填写格式）



3. 后端错误（表单与数据库要求不符）



具体运行方式可以参见 flask-frontend 中的 readme 文档。

五、 代码和文档详细

https://github.com/Litchi-Silhouette/DatabaseLab
我们的代码在 lab1 下，提交了这个文件夹