哈尔滨工业大学

<<数据库系统>>

实验报告一

(2024 年度秋季学期**)**

姓名:	许坤彬
学号:	2022113586
学院:	计算学部
教师:	张浩

实验一

一、实验目的

在熟练掌握 MySQL 基本命令、SQL 语言以及用 C 语言编写 MySQL 操作程序的基础上，学习简单数据库系统的设计方法，包括数据库概要设计、逻辑设计。

二、实验环境

操作系统：Windows 11
数据库：MySQL
编程语言：python
开发环境：PyCharm2023.3.5 专业版、Navicat 17

三、实验过程及结果

1. 数据库的设计

在实验过程中我们设计了一个实体图书馆管理系统，一共包含 8 个实体、七个联系，它们分别是：

八个实体：

图书（Book）：图书编号（主键）、书名、作者、出版日期、分类

会员（Member）：会员编号（主键）、姓名、性别、出生日期、注册日期

管理员（Librarian）：管理员编号（主键）、姓名、职位、入职日期

借阅记录（Borrowing_Record）：记录编号（主键）、借书日期、还书日期

书架（Bookshelf）：书架编号（主键）、位置

出版社（Publisher）：出版社编号（主键）、名称、地址

罚款记录（Fine_Record）：罚款编号（主键）、罚款金额、罚款原因

作者（Author）：作者编号（主键）、姓名、国籍

七个联系：

图书-书架：多对一（一个书架上可以有多本书，但一本书只能放在一个书

架上）

图书-作者：多对多（一本书可以由多个作者编写，一个作者可以写多本书，使用中间表 Book_Author）

会员-借阅记录：一对多（一个会员可以有多个借阅记录）

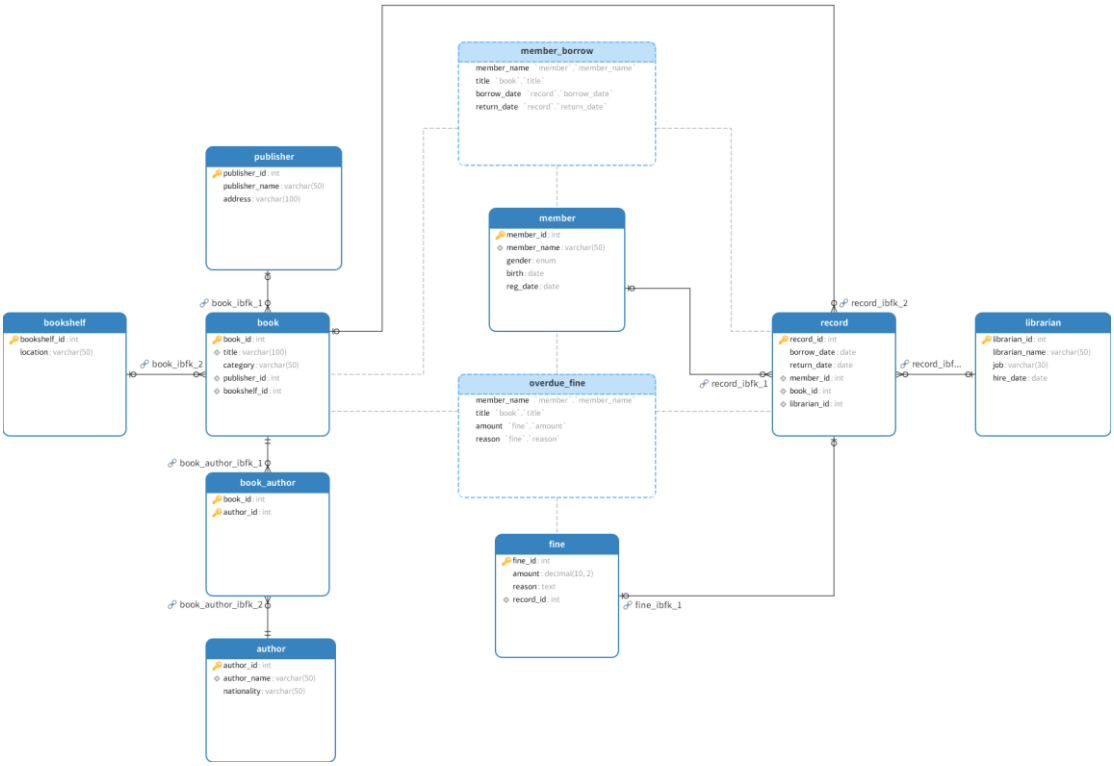
图书-借阅记录：一对多（一本书可以有多个借阅记录）

管理员-借阅记录：一对多（一个管理员可以管理多个借阅记录）

图书-出版社：多对一（一本书只能有一个出版社，但一个出版社可以出版多本书）

借阅记录-罚款记录：一对一（每个借阅记录对应一个罚款记录，如果存在超期情况）

2. 跟据实体和联系，绘制 E-R 图



3. 关系表：

（1）member 表：用于存储会员信息

属性：

member_id：主键，会员的唯一标识

member_name：会员姓名，不允许为空

gender：性别，限定为'M'（男）或'F'（女）

birth：会员出生日期

reg_date：会员注册日期

（2）librarian 表：用于存储管理员信息

属性：

librarian_id：主键，管理员的唯一标识

librarian_name：管理员姓名，不允许为空

job：管理员的职务

hire_date：管理员的雇佣日期

（3）bookshelf 表：用于存储书架信息

属性：

bookshelf_id：主键，书架的唯一标识

location：书架位置，不允许为空

（4）publisher 表：用于存储出版社信息

属性：

publisher_id：主键，出版社的唯一标识

publisher_name：出版社名称，不允许为空

address：出版社地址

（5）book 表：用于存储图书信息

属性：

book_id：主键，图书的唯一标识

title：图书标题，不允许为空

category：图书类别

publisher_id：外键，引用 publisher 表的 publisher_id

bookshelf_id：外键，引用 bookshelf 表的 bookshelf_id

（6）author 表：用于存储作者信息

属性：

author_id：主键，作者的唯一标识

author_name：作者姓名，不允许为空

nationality：作者国籍

（7）book_author 表：用于存储图书与作者的多对多关系

属性：

book_id：主键部分，外键，引用 book 表的 book_id

author_id：主键部分，外键，引用 author 表的 author_id

（8）record 表：用于存储借阅记录

属性：

record_id：主键，借阅记录的唯一标识

borrow_date：借书日期

return_date：还书日期

member_id：外键，引用 member 表的 member_id

book_id：外键，引用 book 表的 book_id

librarian_id：外键，引用 librarian 表的 librarian_id

（9）fine 表：用于存储罚款记录

属性：

fine_id：主键，罚款记录的唯一标识

amount：罚款金额，不允许为空

reason：罚款原因

record_id：外键，引用 record 表的 record_id，且具有唯一性约束

4. 完整性约束

（1）主键约束

member(member_id): member_id 是 member 表的主键，确保每个会员有唯一的标识。

librarian(librarian_id): librarian_id 是 librarian 表的主键，确保每个管理员有唯

一的标识。

bookshelf(bookshelf_id): bookshelf_id 是 bookshelf 表的主键，确保每个书架有唯一的标识。

publisher(publisher_id): publisher_id 是 publisher 表的主键，确保每个出版社有唯一的标识。

book(book_id): book_id 是 book 表的主键，确保每本图书有唯一的标识。

author(author_id): author_id 是 author 表的主键，确保每个作者有唯一的标识。

book_author(book_id, author_id): book_id 和 author_id 的组合是 book_author 表的主键，确保图书与作者的组合关系唯一。

record(record_id): record_id 是 record 表的主键，确保每条借阅记录有唯一的标识。

fine(fine_id): fine_id 是 fine 表的主键，确保每条罚款记录有唯一的标识。

（2）外键约束

book(publisher_id): 作为外键引用 publisher(publisher_id)，确保图书的出版商信息在 publisher 表中存在。

book(bookshelf_id): 作为外键引用 bookshelf(bookshelf_id)，确保图书的书架信息在 bookshelf 表中存在。

book_author(book_id): 作为外键引用 book(book_id)，确保图书在 book 表中存在。

book_author(author_id): 作为外键引用 author(author_id)，确保作者在 author 表中存在。

record(member_id): 作为外键引用 member(member_id)，确保借阅的会员在 member 表中存在。

record(book_id): 作为外键引用 book(book_id)，确保借阅的图书在 book 表中存在。

record(librarian_id): 作为外键引用 librarian(librarian_id)，确保借阅的管理员在 librarian 表中存在。

fine(record_id): 作为外键引用 record(record_id)，确保罚款记录对应的借阅记录在 record 表中存在。

（3）唯一性约束

fine(record_id): record_id 在 fine 表中是唯一的，即一个借阅记录最多只能对应一条罚款记录。

（4）其他约束

member(gender): 限定为枚举类型('M', 'F')，确保性别只能填 M 或 F。

5. 创建的视图

依据会员表、图书表和借阅记录表创建视图 member_borrow，用来呈现会员的借阅记录，其属性包括会员姓名、书名、借阅日期和归还日期。

```
CREATE VIEW member_borrow AS
SELECT member.member_name, book.title, record.borrow_date,
record.return_date
FROM record
JOIN member ON record.member_id = member.member_id
JOIN book ON record.book_id = book.book_id;
```

6. 最关键代码

（1）数据库连接

```
import pymysql
import gradio as gr
import datetime

# 连接到 MySQL 数据库
conn = pymysql.connect(
    host="localhost",  # 数据库主机
    port=3306,  # 端口号，默认为 3306
    user="root",  # 数据库用户名
    password="751016xkb",  # 数据库密码
    database="library",  # 数据库名称
    connect_timeout=60,  # 将超时时间设置为 60 秒
    charset="utf8mb4"  # 使用 utf8mb4 字符集
)

# 创建游标对象
cursor = conn.cursor()
```

（2）增删操作（部分）

```
# sql 的基本使用嵌套函数
def sql_insert(sql, values):
```

```
    try:
        cursor.execute(sql, values)
        conn.commit()
        msg = f"插入成功"
    except Exception as e:
        conn.rollback()  # 出现错误时回滚
        msg = f"插入失败：{e}"

    print(msg)
    return msg

def sql_delete(sql, values):
    try:
        cursor.execute(sql, values)
        conn.commit()
        msg = f"删除成功"
    except Exception as e:
        conn.rollback()  # 出现错误时回滚
        msg = f"删除失败：{e}"

# author 表的基础增删改查
def insert_author(author_id, author_name, nationality):
    if not author_name:
        return "作者名不能为空"

    sql = "INSERT INTO author (author_id, author_name, nationality) VALUES
(%s, %s, %s)"
    values = (author_id, author_name, nationality)
    return sql_insert(sql, values)

def delete_author(author_id):

    check_sql = "SELECT author_id FROM author WHERE author_id = %s"
    check_values = (author_id,)
    result = sql_search_one(check_sql, check_values)

    if result is None:
        msg = f"数据库中不存在序列号为{author_id}的作者"
        print(msg)
        return msg

    sql = "DELETE FROM author WHERE author_id = %s"
    values = (author_id,)
    return sql_delete(sql, values)
```

（3）视图查询

```
# 查询部分
# 利用视图查询，使用视图查询每一个会员借的的每一本书
def search_member_borrow():
    sql = "SELECT * FROM member_borrow;"
    result = sql_search_all(sql)
    return tuple_to_list_string(result)
```

（4）连接查询

```
# 设计一个利用到了连接查询的方法
# 查询会员的借阅书籍和还书情况
def search_record_from_member_name(member_name):
    sql = ("SELECT member.member_id, member.member_name, record.record_id,
record.book_id, book.title, record.borrow_date, record.return_date "
           "FROM record "
           "JOIN book ON record.book_id = book.book_id "
           "JOIN member ON record.member_id = member.member_id "
           "WHERE member.member_name = %s;")
    values = (member_name,)

    try:
        cursor.execute(sql, values)
        result = cursor.fetchall()

        # 如果查询结果为空，返回提示信息
        if not result:
            return "没有找到相关借阅记录。"

        return tuple_to_list_string(result)

    except Exception as e:
        return f"查询失败：{e}"
```

（5）嵌套查询

```
# 查询借阅了两本书以上的用户
def search_member_n_books(book_num):
    sql = ("SELECT member.member_id, member.member_name,
COUNT(record.record_id) "
           "FROM member "
```

```
            "JOIN record ON member.member_id = record.member_id "
            "WHERE member.member_id IN (SELECT record.member_id FROM record
GROUP BY record.member_id HAVING COUNT(record.record_id) >= %s) "
            "GROUP BY member.member_id")
    values = (book_num,)

    try:
        cursor.execute(sql, values)
        result = cursor.fetchall()

        # 如果查询结果为空，返回提示信息
        if not result:
            return "没有找到相关借阅记录。"

        return tuple_to_list_string(result)

    except Exception as e:
        return f"查询失败: {e}"
```

（6）分组查询

```
# 分组查询，按照类别对书籍数量进行统计
def search_books_num_by_category(num):
    sql = "SELECT book.category, COUNT(book.book_id) FROM book GROUP BY
book.category HAVING COUNT(book.book_id) >= %s"
    values = (num,)

    try:
        cursor.execute(sql,values)
        result = cursor.fetchall()

        # 如果查询结果为空，返回提示信息
        if not result:
            return "没有找到相关借阅记录。"

        return tuple_to_list_string(result)

    except Exception as e:
        return f"查询失败: {e}"
```

7. 运行界面

四、实验心得

1. 数据库使用过程中突然断连，且不是代码的问题；

从任务管理器的服务管理模块重启 mysql 服务可以解决问题，但为什么出现断连我也无法查找到原因。猜测可能与数据库的超时设定有关

2. 数据库的插入、删除操作常收到完整性约束的限制，需要根据需求确定功能。并且需要处理好错误时的回滚事务。