# Lab 1 Report

## 实验目的

- 熟悉并掌握MySQL编程方法
- 熟悉并掌握SQL和过程化SQL在设计数据库方面的方法和思路
- 理解MySQL编程逻辑和编程规范，设计一个简单的图书馆数据库系统

## 实验环境

- 系统：Ubuntu 22.04.2 LTS
- 数据库服务器：MySQL Server 8.0
- IDE开发环境：DataGrip

## 实验过程

### 创建图书馆数据库基本表

本次实验一共要创建4个表，每一个表都只需要按照题目中的规定写代码即可

- 图书表：

```
create table Book
(
    ID            char(8) primary key,
    name          varchar(10) not null,
    author        varchar(10),
    price         float,
    status        int check ( status in (0, 1, 2, 3) ),
    borrow_Times  int,
    reserve_Times int default 0
);
```

这个表注意要限制status的范围，并将ID设为主码

- 读者表：

```
create table Reader
(
    ID      char(8) primary key,
    name    varchar(10),
    age     int check ( age >= 0 ),
    address varchar(20)
);
```

这个表没有特别需要注意的， 只需要注意隐含的age大于等于0

- 借阅表

```
create table Borrow
(
    book_ID     char(8) references Book (ID),
    reader_ID   char(8) references Reader (ID),
    borrow_Date date,
    return_Date date,
    primary key (book_ID, reader_ID, borrow_Date),
    check ( return_Date >= borrow_Date )
);
```

借阅表引用了图书号、读者号两个外键，同时有图书号、读者号、借阅日期三个主码。这里我为了严谨，检查了归还日期必须要不早于借阅日期。

- 预约表

```
create table Reserve
(
    book_ID      char(8) references Book (ID),
    reader_ID    char(8) references Reader (ID),
    reserve_Date date default (curdate()),
    take_Date    date,
    primary key (book_ID, reader_ID, reserve_Date),
    check ( take_Date >= reserve_Date )
);
```

预约表是有明确的规定：预约取书日期必须晚于预约日期，这里做一个检查即可。对于预约日期的默认值，可以用curdate来生成当前日期

## 用SQL语言完成特定查询

- 查询读者 Rose 借过的读书（包括已还和未还）的图书号、书名和借期

```
select b.ID, b.name, w.borrow_Date
from Borrow w
         join Book b on w.book_ID = b.ID
         join Reader r on w.reader_ID = r.ID
where r.name = 'Rose';
```

这个查询非常简单，只需要两次连接即可。

- 查询从没有借过图书也从没有预约过图书的读者号和读者姓名

```
select r.ID, r.name
from Reader r
where r.ID not in (select distinct reader_ID
                   from Borrow)
  and r.ID not in (select distinct reader_ID
                   from Reserve);
```

这里可以使用not in来实现“没有”的操作，并做两次子查询即可实现

- 查询被借阅次数最多的作者（注意一个作者可能写了多本书）

```
select b.author
from Book b
group by b.author
having SUM(b.borrow_Times)
order by SUM(b.borrow_Times) DESC
limit 1;
```

对于一个作者写了多本书的情况，我们只需要对图书表按照作者进行分组，并求借阅量的和即可。最后可以用limit 1 来输出借阅量最多的读者

- 查询目前借阅未还的书名中包含“MySQL”的的图书号和书名

```
select b.ID, b.name
from Book b
         join Borrow w on w.book_ID = b.ID
where b.name like '%MySQL%'
  and w.return_Date is null;
```

使用借阅表的归还日期为空则表示未还的特点，找到对应书名即可

- 查询借阅图书数目超过 10 本的读者姓名

```
select r.name
from Reader r
         join Borrow w on w.reader_ID = r.ID
group by r.ID
having count(*) > 10;
```

这个查询和第三个查询类似，不过这里可以使用having来限制书数超过10本

- 查询没有借阅过任何一本 John 所著的图书的读者号和姓名

```
select re.ID, re.name
from Reader re
where re.ID not in (select r.ID
                    from Reader r
                             join Borrow w on w.reader_ID = r.ID
                             join Book b on w.book_ID = b.ID
                    where b.author = 'John');
```

这个查询我们可以先找出借阅过John所著图书的所有读者，之后再找到不在这个集合里面的所有读者即可

- 查询 2022 年借阅图书数目排名前 10 名的读者号、姓名以及借阅图书数

```
from Reader r
         join Borrow w on w.reader_ID = r.ID
where w.borrow_Date like '2022%'
group by r.ID
order by num DESC
limit 10;
```

这里我们用like来表示2022年借阅情况，同时对读者编号进行分组，用limit 10 来返回前十位即可

- 创建一个读者借书信息的视图,该视图包含读者号、姓名、所借图书号、图书名和借期，并使用该视图查询最近一年所有读者的读者号以及所借阅的不同图书数

  首先来创建一个视图：

```
create view BorrowInfo as
select r.ID as reader_ID, r.name as reader_name, b.ID as book_ID, b.name as
book_name, w.borrow_Date
from Reader r
         join Borrow w on r.ID = w.reader_ID
         join Book b on b.ID = w.book_ID;
```

  接下来来查询：

```
select reader_ID, count(distinct book_ID) as num
from BorrowInfo
where borrow_Date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
group by reader_ID;
```

  为了满足“最近一年”，这列使用了日期减法。

## 设计数据库存储过程

- 设计一个存储过程 updateReaderID,实现对读者表的 ID 的修改

```
delimiter //
create procedure updateReaderID(in oldID char(8), in newID char(8))
begin
    -- check if the new ID is already in the table
    select count(*) into @count from Reader where ID = newID;
    if @count > 0 then
        signal sqlstate '45000'
            set message_text = 'The new ID is already in the table';
    end if;

    -- check if the old ID is already in the table
    select count(*) into @count from Reader where ID = oldID;
    if @count = 0 then
        signal sqlstate '45000'
            set message_text = 'The old ID is not in the table';
    end if;
```

```
    start transaction;
    update Borrow set Reader_ID = newID where Reader_ID = oldID;
    update Reserve set Reader_ID = newID where Reader_ID = oldID;
    update Reader set ID = newID where ID = oldID;
    commit;
end;
delimiter ;
```

更新读者ID的时候，自然要检查几个内容：

- 新的ID是否已经被读者使用了？如果被使用了，那么这样的ID就是非法的。
- 旧的ID是否确实被某个读者使用了？如果没有被使用，那么这样的修改也是非法的

同时，在更新ID的时候，我们要逐个更新读者表、借阅表和预约表的相应内容，这些可以用一个事务来原子地完成。

- 设计一个存储过程 borrowBook, 当读者借书时调用该存储过程完成借书处理

  这个存储过程内容较多，我们来逐一分析：

  - 检查读者和图书是否是确实存在的：

    ```
    -- 检查读者是否存在
    select count(*) into @reader from Reader where ID = readerId;
    if @reader = 0 then
        signal sqlstate '45000' set message_text = 'Reader is not exist';
    end if;

    -- 检查图书是否存在
    select count(*) into @book from Book where ID = bookId;
    if @book = 0 then
        signal sqlstate '45000' set message_text = 'Book is not exist';
    end if;
    ```

  - 检查图书是否是可借的状态：这里需要注意，图书不可借，要么这本图书在预约表中没有被还，要么自己没预约但被别人预约了：

    ```
    -- 检查读者是否存在
    select count(*) into @reader from Reader where ID = readerId;
    if @reader = 0 then
        signal sqlstate '45000' set message_text = 'Reader is not exist';
    end if;

    -- 检查图书是否存在
    select count(*) into @book from Book where ID = bookId;
    if @book = 0 then
        signal sqlstate '45000' set message_text = 'Book is not exist';
    end if;
    ```

  - 同一天不允许同一个读者重复借阅同一本读书：这个只需要查阅当天该读者借该书的记录即可

```
    select count(*)
    into @count
    from Borrow
    where book_ID = bookId
      and reader_ID = readerId
      and borrow_date = borrowDate;
    if @count > 0 then
        signal sqlstate '45000' set message_text = 'Reader has already
borrowed this book today';
    end if;
```

- 一个读者最多只能借阅 3 本图书：这个只需要查阅借阅表，数一下该读者有多少未环书即可

```
    select count(*) into @count from Borrow where reader_ID = readerId and
return_Date is null;
    if @count >= 3 then
        signal sqlstate '45000' set message_text = 'Reader has borrowed 3
books';
    end if;
```

- 如果借阅者已经预约了该图书，则允许借阅，但要求借阅完成后删除借阅者对该图书的预约记录：

```
    select count(*) into @count from Reserve where book_ID = bookId and
reader_ID = readerId and take_Date is null;
    if @count = 0 then
        select count(*) into @count from Reserve where book_ID = bookId and
take_Date is null;
        if @count > 0 then
            signal sqlstate '45000' set message_text = 'Reader has not
reserved this book and it is reserved by others';
        end if;
    end if;
```

- 这些条件都满足后，就允许借阅了。这里需要把图书表的times加一，并修改status。注意，在我的实现中，借出状态享有最高优先级。不管这本书是否被预约，只要被借出，那么状态就是1。

```
    insert into Borrow(reader_ID, book_ID, borrow_Date) values (readerId,
bookId, borrowDate);
    set @row = row_count();
    if @row = 0 then
        signal sqlstate '45000' set message_text = 'Failed to insert new
borrowing record';
    end if;
    update Book set borrow_Times = Book.borrow_Times + 1, status = 1 where
ID = bookId;
    set @row = row_count();
    if @row = 0 then
        signal sqlstate '45000' set message_text = 'Failed to update borrow
times and status';
    end if;
    delete from Reserve where book_ID = bookId and reader_ID = readerId and
take_Date is null;
```

- 设计一个存储过程 returnBook，当读者还书时调用该存储过程完成还书处理

  为了达成这个目标，我们需要先做四个检查：

  - 检查该书是否存在

    ```
        select count(*) into @count from Book where ID = bookId;
        if @count = 0 then
            signal sqlstate '45000' set message_text = 'This book does not
    exist.';
        end if;
    ```

  - 检查该读者是否存在

    ```
        select count(*) into @count from Reader where ID = readerId;
        if @count = 0 then
            signal sqlstate '45000' set message_text = 'This reader does not
    exist.';
        end if;
    ```

  - 检查该书是否已经被借出或已经归还

    ```
        select status into @status from Book where ID = bookId;
        if @status != 1 then
            signal sqlstate '45000' set message_text = 'This book is not
    borrowed or has been returned.';
        end if;
    ```

  - 检查该读者是否已经借过该书

```
    select count(*) into @count from Borrow where bookId = book_ID and
readerId = reader_ID;
    if @count = 0 then
        signal sqlstate '45000' set message_text = 'This reader has not
borrowed this book.';
    end if;
```

四个检查全部通过后，就可以更新借阅记录和图书状态了。这里特别是对图书状态的更新，依然沿用了之前的约定：借出状态享有最高的优先级：

```
    -- 更新借阅记录
    select count(*)
    into @count
    from Borrow
    where bookId = book_ID
      and readerId = reader_ID
      and return_Date is null
      and borrow_Date > returnDate;
    if @count != 0 then
        signal sqlstate '45000' set message_text = 'The return date is earlier
than the borrow date.';
    end if;
    update Borrow set return_Date = returnDate where bookId = book_ID and
readerId = reader_ID and return_Date is null;

    -- 更新图书状态
    select count(*) into @count from Reserve where book_ID = bookId and
take_Date is null;
    if @count != 0 then
        update Book set status = 2 where ID = bookId;
    else
        update Book set status = 0 where ID = bookId;
    end if;
```

## 设计触发器

在本次实验中，我们使用了两个触发器

- 第一个触发器：当一本书被预约的时候，自动增加预约次数并自动修改图书的status，但若这本书已经被借出，那么依然保持为借出状态。

  这里我加入了一个新的检查：不允许一个人重复预约同一本书，同时小心地修改status

```
create trigger reserve_book
    before insert
    on Reserve
    for each row
begin
    -- 不允许同一个人重复预约同一本书
    select count(*) into @reserve from Reserve where book_ID = new.book_id and
reader_ID = new.reader_id;
```

```
    if @reserve != 0 then
        signal sqlstate '45000' set message_text = 'You have already reserved
this book.';
    end if;
    -- 如果这本书被借出，那么status=1
    select count(*) into @borrow from Borrow where book_ID = new.book_id and
return_date is null;
    if @borrow != 0 then
        update Book set status = 1 where ID = new.book_id;
    else
        update Book set status = 2 where ID = new.book_id;
    end if;

    update Book set reserve_Times = reserve_Times + 1 where ID = new.book_id;
end;
```

- 第二个触发器：当读者取消预约，自动减少预约次数，同时修改图书状态：

```
create trigger cancel_reserve
    after delete
    on Reserve
    for each row
begin
    -- 如果这本书被借出，那么status=1
    select count(*) into @borrow from Borrow where book_ID = old.book_ID;
    select count(*) into @reserve from Reserve where book_ID = old.book_ID;
    update Book set reserve_Times = reserve_Times - 1 where ID = old.book_ID;
    update Book set status = 1 where ID = old.book_ID;
    if @borrow != 0 then
        update Book set status = 1 where ID = old.book_ID;
        -- 如果这本书还有预约，那么status=2
    elseif @reserve != 0 then
        update Book set status = 2 where ID = old.book_ID;
        -- 如果这本书没有预约，那么status=0
    else
        update Book set status = 0 where ID = old.book_ID;
    end if;
end;
```

# 实验结果

这里我主要来展示一下给出的测试的结果：

- 查询读者 Rose 借过的读书（包括已还和未还）的图书号、书名和借期

| | ID | name | borrow_Date |
|---|---|---|---|
| 1 | b1 | 数据库系统实现 | 2022-02-22 |
| 2 | b11 | 三体 | 2022-01-11 |
| 3 | b16 | 中国2185 | 2022-01-11 |
| 4 | b19 | HowWeThink | 2023-04-08 |
| 5 | b2 | 数据库系统概念 | 2022-02-22 |

- 查询从没有借过图书也从没有预约过图书的读者号和读者姓名

| | ID | name |
|---|---|---|
| 1 | r10 | 汤大晨 |
| 2 | r22 | 张悟 |

- 查询被借阅次数最多的作者

| | author |
|---|---|
| 1 | 刘慈欣 |

- 查询目前借阅未还的书名中包含“MySQL”的的图书号和书名

| | ID | name |
|---|---|---|
| 1 | b14 | Perl&MySQL |

- 查询借阅图书数目超过 10 本的读者姓名

| | name |
|---|---|
| 1 | 王林 |
| 2 | 王林 |
| 3 | David |

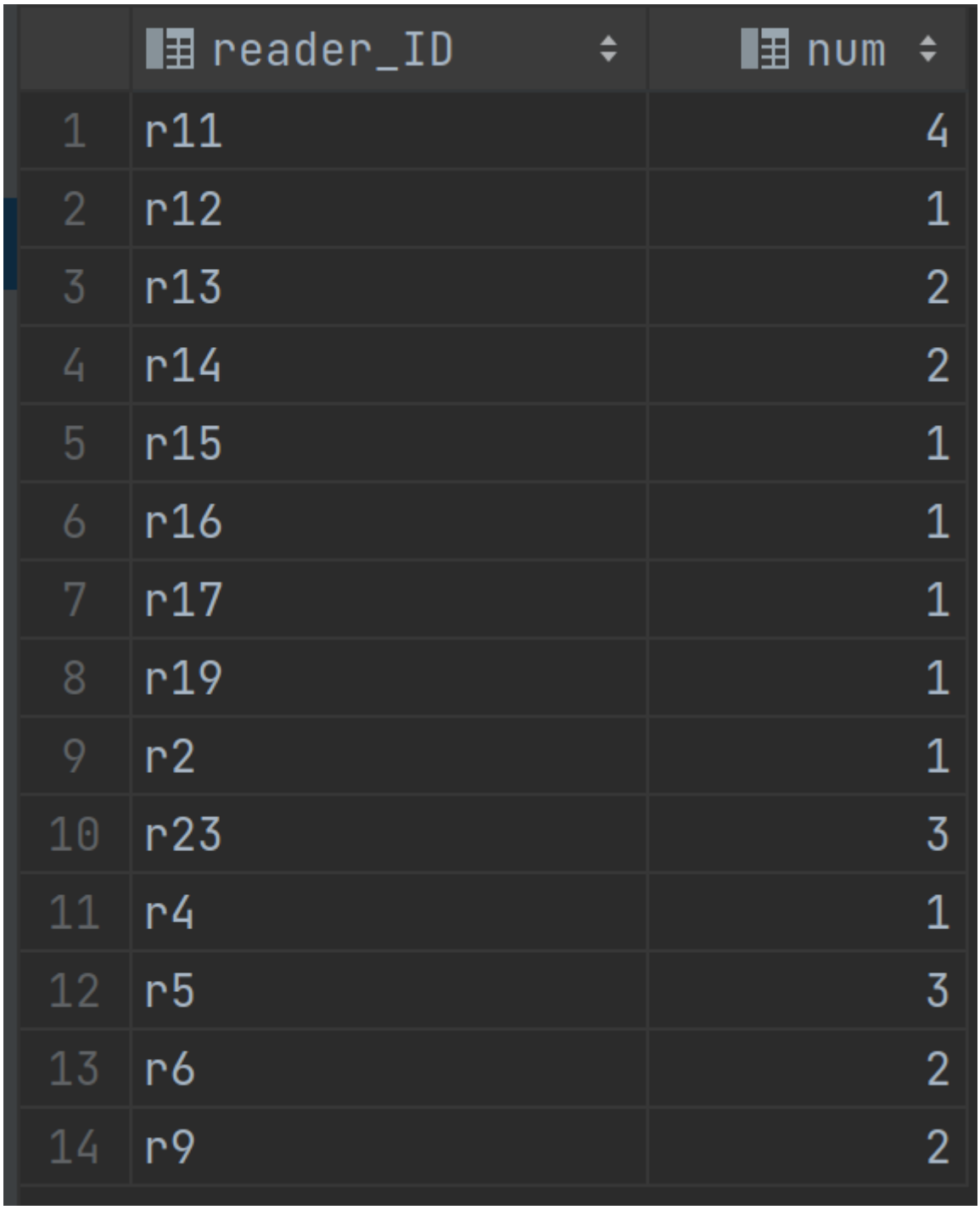
- 查询没有借阅过任何一本 John 所著的图书的读者号和姓名

| | ID | name |
|---|---|---|
| 1 | r1 | 王林 |
| 2 | r10 | 汤大晨 |
| 3 | r11 | 李平 |
| 4 | r12 | Lee |
| 5 | r14 | Bob |
| 6 | r15 | 李晓 |
| 7 | r17 | Mike |
| 8 | r18 | 范维 |
| 9 | r19 | David |
| 10 | r20 | Vipin |
| 11 | r21 | 林立 |
| 12 | r22 | 张悟 |
| 13 | r23 | 袁平 |
| 14 | r4 | Mora |
| 15 | r6 | 李—— |
| 16 | r8 | 赵四 |

- 查询 2022 年借阅图书数目排名前 10 名的读者号、姓名以及借阅图书数

| | ID | name | num |
|---|---|---|---|
| 1 | r11 | 李平 | 4 |
| 2 | r2 | Rose | 4 |
| 3 | r3 | 罗永平 | 4 |
| 4 | r1 | 王林 | 3 |
| 5 | r7 | 王二狗 | 3 |
| 6 | r9 | 魏心 | 3 |
| 7 | r8 | 赵四 | 3 |
| 8 | r23 | 袁平 | 3 |
| 9 | r4 | Mora | 2 |
| 10 | r6 | 李—— | 2 |

- 创建一个读者借书信息的视图,该视图包含读者号、姓名、所借图书号、图书名和借期,并使用该视图查询最近一年所有读者的读者号以及所借阅的不同图书数



| | reader_ID | num |
|---|---|---|
| 1 | r11 | 4 |
| 2 | r12 | 1 |
| 3 | r13 | 2 |
| 4 | r14 | 2 |
| 5 | r15 | 1 |
| 6 | r16 | 1 |
| 7 | r17 | 1 |
| 8 | r19 | 1 |
| 9 | r2 | 1 |
| 10 | r23 | 3 |
| 11 | r4 | 1 |
| 12 | r5 | 3 |
| 13 | r6 | 2 |
| 14 | r9 | 2 |

有关几个存储过程的结果和报错情况，我已在检查时详细向助教延时，这里也不方便做过多展示，故不在这里放出了。

## 总结与思考

- MySQL给予我们的功能是非常多的，很多时候我们可以用很简单的语句就完成了一个非常复杂的功能
- 在设计数据库的时候，特别要注意对一些非法的情况作出检查。宗旨就是：永远不能相信用户的操作
- 这次实验让我对几种触发器有了深刻理解，不同类型触发器实现相同功能的复杂度可能大相径庭。