华中科技大学

数据库系统原理实践报告

专　　业：计算机科学与技术

班　　级：**ACM1901**

学　　号：**U201914918**

姓　　名：何佳林

指导老师：杨茂林

分数	
教师签名	

2022 年 7 月 6 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目录

1 课程任务概述

“数据库系统原理实践”是配合“数据库系统原理”课程独立开设的实践课，注重理论与实践相结合。本课程以 MySQL 为例，系统性地设计了一系列的实训任务，内容涉及以下几个部分：

1. 数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的管理与编程；
2. 数据查询，数据插入、删除与修改等数据处理相关任务；
3. 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4. 数据库的设计与实现；
5. 数据库应用系统的开发 (JAVA 篇)。

课程依托头歌实践教学平台，实践课程链接为：数据库系统原理实践-以 MySQL 为例。实验环境为 Linux 操作系统下的 MySQL 8.0.28 （主要为 8.028 版本，部分关卡使用 8.022 版本，使用中基本无差别）。在数据库应用开发环节，使用 JAVA 1.8 。

2 任务实施过程与分析

2.1 数据库、表与完整性约束的定义 (Create)

本节的六个关卡围绕数据库和表的创建展开，分别完成了数据库的创建、表的创建以及表中主码、外码、CHECK、DEFAUT 和 UNIQUE 等约束的建立。

2.1.1 创建数据库

本关任务：创建用于 2022 年北京冬奥会信息系统的数据库: beijing2022。

在开始操作前，需要连接到数据库，在命令行输入:

```
mysql -h host -u user -ppassword dbname
```

然后使用如下命令创建数据库：

```
create database beijing2022;
```

2.1.2 创建表及主码约束

本关任务：在指定的数据库中创建一个表，并为表指定主码。

创建数据库 TestDb ，在 TestDb 下使用 **create table** 命令创建表 t_emp ，并对主码 id 附加 **primary key** 约束。

```
create database TestDb;
use TestDb;
create table t_emp(
    id int primary key,
    deptId int,
    name varchar(32),
    salary float
);
```

2.1.3 创建外码约束

本关任务：创建外码约束（参照完整性约束）。

按关卡要求创建 MyDb 数据库，并在其中创建 dept 表和 staff 表，并给表 staff 创建外键，代码如下：

```
create database MyDb;
use MyDb;
create table dept(
    deptNo int primary key,
    deptName varchar(32)
);
create table staff(
    staffNo int primary key,
    staffName varchar(32),
```

```
    gender char(1),
    dob date,
    salary numeric(8, 2),
    deptNo int,
    constraint FK_staff_deptNo foreign key(deptNo) references dept(deptNo)
);
```

代码中的第 14 行创建了表 staff 的 deptNo 到表 dept 的 deptNo 的外键约束，约束名为 FK_staff_deptNo 。

2.1.4 CHECK 约束

本关任务：为表创建 CHECK 约束

根据任务要求，创建表并添加 check 约束，代码如下：

```
create table products(
    pid char(10) primary key,
    name varchar(32),
    brand char(10) constraint CK_products_brand check(brand in ('A', 'B')),
    price int constraint CK_products_price check(price > 0)
);
```

2.1.5 DEFAULT 约束

该关卡任务已完成，实施情况本报告略过。

2.1.6 UNIQUE 约束

该关卡任务已完成，实施情况本报告略过。

2.2 表结构与完整性约束的修改 (ALTER)

本节的四个关卡围绕数据库中表的基本修改操作展开，设计用 alter 语句对表的定义进行修改，如更换/修改表名、列名、列的类型、列约束、表约束；添加或删除列、约束等。

2.2.1 修改表名

本关任务：修改表的名称，将表名 your_table 更改为 my_table 。

ALTER TABLE 的简化的语法为：

```
ALTER TABLE 表名
[修改事项 [, 修改事项] ...]
```

其中常用的修改事项有：

```
修改事项 ::=
    ADD [COLUMN] 列名 数据类型 [列约束]
       [FIRST | AFTER col_name]
  | ADD {INDEX|KEY} [索引名] [类型] (列 1,...)
```

```
    | ADD [CONSTRAINT [约束名]] 主码约束
    | ADD [CONSTRAINT [约束名]] UNIQUE 约束
    | ADD [CONSTRAINT [约束名]] 外码约束
    | ADD [CONSTRAINT [约束名]] CHECK 约束
    | DROP {CHECK|CONSTRAINT} 约束名
    | ALTER [COLUMN] 列名 {SET DEFAULT {常量 | (表达式)} | DROP DEFAULT}
    | CHANGE [COLUMN] 列名 新列名 数据类型 [列约束]
        [FIRST | AFTER col_name]
    | DROP [COLUMN] 列名
    | DROP {INDEX|KEY} 索引名
    | DROP PRIMARY KEY
    | DROP FOREIGN KEY fk_symbol
    | MODIFY [COLUMN] 列名 数据类型 [列约束]
        [FIRST | AFTER col_name]
    | RENAME COLUMN 列名 TO 新列名
    | RENAME {INDEX|KEY} 索引名 TO 新索引名
    | RENAME [TO|AS] 新表名
```

根据任务要求选择“**RENAME [TO|AS]** 新表名”修改事项，代码如下：

```
alter table your_table rename my_table;
```

2.2.2 添加与删除字段

本关任务：删除表 orderDetail 中的列 orderDate ，添加列 unitPrice 。

根据任务要求选择“**DROP [COLUMN]** 列名”和“**ADD [COLUMN]** 列名 数据类型 [列约束]”修改事项，代码如下：

```
alter table orderDetail drop orderDate;
alter table orderDetail add unitPrice decimal(10, 2);
```

2.2.3 修改字段

本关任务：修改指定字段的类型和名称。

根据任务要求选择“**MODIFY [COLUMN]** 列名 数据类型 [列约束]”和“**RENAME COLUMN** 列名 **TO** 新列名”修改事项，代码如下：

```
alter table addressBook
    modify QQ char(12),
    rename column weixin to wechat;
```

2.2.4 添加、删除与修改约束

本关任务：添加、删除与修改约束。

对每个要求修选取合适的修改事项即可，代码如下：

```
#(1) 为表 Staff 添加主码
alter table Staff add primary key (staffNo);
#(2) Dept.mgrStaffNo 是外码，对应的主码是 Staff.staffNo, 请添加这个外码，名字为 FK_Dept_mgrStaffNo:
alter table Dept add constraint FK_Dept_mgrStaffNo foreign key(mgrStaffNo) references
↪  Staff(staffNo);
```

```sql
#(3) Staff.dept 是外码，对应的主码是 Dept.deptNo. 请添加这个外码，名字为 FK_Staff_dept:
alter table Staff add constraint FK_Staff_dept foreign key(dept) references Dept(deptNo);
#(4) 为表 Staff 添加 check 约束，规则为：gender 的值只能为 F 或 M；约束名为 CK_Staff_gender:
alter table Staff add constraint CK_Staff_gender check (gender in ('F', 'M'));
#(5) 为表 Dept 添加 unique 约束：deptName 不允许重复。约束名为 UN_Dept_deptName:
alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

2.3 数据查询 (Select)

本节的 19 个关卡由浅到深围绕 select 语句在不同场景下的应用展开。

2.3.1 查询客户主要信息

本关任务：查询所有客户的名称、手机号和邮箱信息，查询结果按照客户编号排序。
基础的 select 查询，代码如下：

```sql
select
    c_name, c_phone, c_mail
from client
order by c_id;
```

2.3.2 邮箱为 null 的客户

本关任务：查询客户表 (client) 中没有填写邮箱的客户的编号、名称、身份证号、手机号。
使用 where 字句即可完成该条件查询，代码如下：

```sql
select
    c_id,
    c_name,
    c_id_card,
    c_phone
from client
where c_mail is null;
```

2.3.3 既买了保险又买了基金的客户

本关任务：查询既买了保险又买了基金的客户的名称和邮箱。
使用 exists 子查询即可，代码如下：

```sql
select
    c_name,
    c_mail,
    c_phone
from client
where
    exists (
        select
            *
        from property
        where
            pro_c_id = c_id
            and pro_type = 2
```

```
    )
    and exists (
        select
            *
        from property
        where
            pro_c_id = c_id
            and pro_type = 3
    )
order by c_id;
```

2.3.4 办理了储蓄卡的客户信息

该关卡任务已完成，实施情况本报告略过。

2.3.5 每份金额在 30000~50000 之间的理财产品

该关卡任务已完成，实施情况本报告略过。

2.3.6 商品收益的众数

本关任务：查询资产表中所有资产记录里商品收益的众数和它出现的次数。

在 property 表中基于 pro_income 分组，利用 count 函数统计数量，如果某个分组数量大于等于所有分组或者数量最大的分组，则为众数。代码如下：

```
select
    pro_income,
    count(*) as presence
from property
group by pro_income
having count(*) >= all(
    select
        count(*)
    from property
    group by pro_income
);
```

2.3.7 未购买任何理财产品的武汉居民

本关任务：查询未购买任何理财产品的武汉居民的信息。

湖北省武汉市居民的身份证号开头为“4201”，利用“`like "4201%"`”即可筛选出武汉居民，再利用 exists 查询是否购买理财产品即可。代码如下：

```
select
    c_name,
    c_phone,
    c_mail
from client
where
    c_id_card like "4201%"
    and not exists (
        select
            *
```

```
        from property
        where
            pro_c_id = c_id
            and pro_type = 1
    )
order by c_id;
```

2.3.8 持有两张信用卡的用户

该关卡任务已完成，实施情况本报告略过。

2.3.9 购买了货币型基金的客户信息

该关卡任务已完成，实施情况本报告略过。

2.3.10 投资总收益前三名的客户

本关任务：查询投资总收益前三名的客户。

利用“**inner join**”对用户表 client 和资产表 property 做自然连接，然后按 c_id 分组统计收益和并降序排序，利用“**limit** 3”输出前 3 名用户即可。代码如下：

```
select
    c_name,
    c_id_card,
    sum(pro_income) total_income
from
    client
    inner join property
    on
        pro_c_id = c_id
        and pro_status = " 可用"
group by c_id
order by sum(pro_income) desc
limit 3;
```

2.3.11 黄姓客户持卡数量

该关卡任务已完成，实施情况本报告略过。

2.3.12 客户理财、保险与基金投资总额

该关卡任务已完成，且与下一关思路一致，实施情况本报告略过。

2.3.13 客户总资产

本关任务：查询客户在本行的总资产。

对于每个客户每种类型的资产总和，可以用一条 select 语句获取，将全部的结果利用 union all 取可重并集，再与用户表 client 做自然连接，最后再按 c_id 分组，统计资产和并降序排序即可。

值得注意的一个常识：如果银行卡类型的信用卡，那么总资产要减去信用卡的 b_balance，即透支金额。最终代码如下：

```
select
    c_id,
    c_name,
    ifnull(sum(amount), 0) as total_property
from
    client
    left join (
        select
            pro_c_id,
            pro_quantity * p_amount as amount
        from property, finances_product
        where pro_pif_id = p_id
            and pro_type = 1
        union all
        select
            pro_c_id,
            pro_quantity * i_amount as amount
        from property, insurance
        where pro_pif_id = i_id
            and pro_type = 2
        union all
        select
            pro_c_id,
            pro_quantity * f_amount as amount
        from property, fund
        where pro_pif_id = f_id
            and pro_type = 3
        union all
        select
            pro_c_id,
            sum(pro_income) as amount
        from property
        group by pro_c_id
        union all
        select
            b_c_id,
            sum(if(b_type = " 储蓄卡", b_balance, -b_balance)) as amount
        from bank_card
        group by b_c_id
    ) pro
    on c_id = pro.pro_c_id
group by c_id
order by c_id;
```

2.3.14 第 N 高问题

本关任务：查询每份保险金额第 4 高保险产品的编号和保险金额。

在保险表 insurance 中利用 “**distinct** i_amount” 选出不同的保险金额并降序，再利用 “**limit** 3, 1” 即可获取 select 结果第 4 行的内容，即第 4 高的保险金额。最后在外部查询中判断保险金额是否等于该金额即可，代码如下：

```
select
    i_id,
    i_amount
from insurance
```

```sql
where i_amount = (
    select
        distinct i_amount
    from insurance
    order by i_amount desc
    limit 3, 1
);
```

2.3.15 基金收益两种方式排名

本关任务：对客户基金投资收益实现两种方式的排名次。

以降序的基金投资收益和为关键字，直接调用 MySQL 库函数即可：函数 rank() 是名次不连续的排名，函数 dense_rank() 是名次连续的排名。

以名次不连续为例，代码如下：

```sql
select
    pro_c_id,
    sum(pro_income) as total_revenue,
    rank() over(order by sum(pro_income) desc) as "rank"
from property
where pro_type = 3
group by pro_c_id
order by total_revenue desc, pro_c_id;
```

2.3.16 持有完全相同基金组合的客户

本关任务：查询持有完全相同基金组合的客户。

先按用户 id 分组，再利用 group_concat() 函数将用户所购买的基金 id 去重排序后拼接成一个字符串 f_id 。由于需要重复用到上述表，所以利用 with as 创建一个公用临时表，如果表中两个客户的 f_id 相同则说明这两个客户的基金组合完全相同。代码如下：

```sql
with pro(c_id, f_id) as (
    select
        pro_c_id c_id,
        group_concat(distinct pro_pif_id order by pro_pif_id) f_id
    from property
    where pro_type = 3
    group by pro_c_id
)
select
    t1.c_id c_id1,
    t2.c_id c_id2
from pro t1, pro t2
where
    t1.c_id < t2.c_id
    and t1.f_id = t2.f_id;
```

2.3.17 购买基金的高峰期

本关任务：查询 2022 年 2 月购买基金的高峰期，如果连续三个交易日，投资者购买基金的总金额超过 100 万，则称这连续的几日为投资者购买基金的高峰期。

利用 “datediff(pro_purchase_time, "2021-12-31")” 获取该交易日是 2022 年的第几天，再利用 week() 函数获取该天是当年的第几周，由于每周有 2 天不是交易日，所以前者减去后者即可得到不考虑节假日的情况下，该天是当年的第几个交易日，记为 workday 。由于 2022 年 2 月第 1 个交易日为 2 月 7 日，是周一，所以按照上述方法计算 “该天是当年的第几个交易日”，对于 2 月的交易日而言只存在一个相同的偏移量，不影响结果。然后再附加上交易时间是 2022 年 2 月且交易类型是基金的条件筛选即可。

获取上述表后再从中筛选出交易总金额超过 100 万的行，再利用 row_number() 函数获取筛选后该行的行号，记为 rownum 。可以得出结论，如果两行 workday – rownum 的结果是相同的，那么这两行就属于同一个连续段中，按这个结果分类调用 count() 函数就可以获取每个连续段的长度。根据任务要求，如果连续段的长度大于等于 3 ，则属于该连续段的交易日都是 “投资者购买基金的高峰期”。代码如下：

```sql
select
    t3.t as pro_purchase_time,
    t3.amount as total_amount
from (
    select
        *,
        count(*) over(partition by t2.workday - t2.rownum) cnt
    from (
        select
            *,
            row_number() over(order by workday) rownum
        from (
            select
                pro_purchase_time t,
                sum(pro_quantity * f_amount) amount,
                @row := datediff(pro_purchase_time, "2021-12-31") - 2 * week(pro_purchase_time)
                ↪  workday
            from
                property, fund, (select @row) a
            where
                pro_purchase_time like "2022-02-%"
                and pro_type = 3
                and pro_pif_id = f_id
            group by pro_purchase_time
        ) t1
        where amount > 1000000
    ) t2
) t3
where t3.cnt >= 3;
```

对 finance3 数据集进行自测运行，输出表 t3 的所有内容，结果如表2-1所示。

表 2-1: 购买基金的高峰期

pro_purchase_time	total_amount	workday	rownum	cnt
2022-02-15	2230000	32	1	2
2022-02-16	1050000	33	2	2
2022-02-18	3762500	35	3	3
2022-02-21	7030000	36	4	3
2022-02-22	3950000	37	5	3

表中第 1,2 行 workday – rownum 的结果均为 31 ，属于同一个连续段，长度为 2 不是高

峰期；表中第 3,4,5 行 workday – rownum 的结果均为 32，属于同一个连续段，长度为 3，是高峰期。

2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总金额

该关卡任务已完成，实施情况本报告略过。

2.3.19 以日历表格式显示每日基金购买总金额

本关任务：以日历表格式显示 2022 年 2 月每周每日基金购买总金额。

对资产表 property 和基金表 fund 做自然连接，并按交易时间分组计算出购买总金额，再利用 week() 函数和 weekday() 函数获取当天是当年的第几周和当天是周几。最后按照周次分组，对 weekday() 的结果使用 if 语句判断将金额放在哪一列计算即可。代码如下：

```
select
    wk week_of_trading,
    sum(if(dayId = 0, amount, null)) Monday,
    sum(if(dayId = 1, amount, null)) Tuesday,
    sum(if(dayId = 2, amount, null)) Wednesday,
    sum(if(dayId = 3, amount, null)) Thursday,
    sum(if(dayId = 4, amount, null)) Friday
from (
    select
        week(pro_purchase_time) - 5 wk,
        weekday(pro_purchase_time) dayId,
        sum(pro_quantity * f_amount) amount
    from
        property
        join fund
        on pro_pif_id = f_id
    where
        pro_purchase_time like "2022-02-%"
        and pro_type = 3
    group by pro_purchase_time
) t
group by wk;
```

2.4 数据的插入、修改与删除

本节的 6 个关卡围绕 Insert, Update, Delete 语句在不同场景下的应用展开。

2.4.1 插入多条完整的客户信息

本关任务：向客户表 client 插入数据。

用一条 insert 语句完成，代码如下：

```
insert into client values
    (1," 林惠雯","960323053@qq.com","411014196712130323","15609032348","Mop5UPkl"),
    (2," 吴婉瑜","1613230826@gmail.com","420152196802131323","17605132307","QUTPhxgVNlXtMxN"),
    (3," 蔡贞仪","252323341@foxmail.com","160347199005222323","17763232321","Bwe3gyhEErJ7");
```

2.4.2 插入不完整的客户信息

本关任务：向客户表 client 插入一条数据不全的记录。

插入不完整的数据记录需要在 values 前指定插入的列，代码如下：

```
insert into client
    (c_id, c_name, c_phone, c_id_card, c_password)
values
    (33, " 蔡依婷", "18820762130", "350972199204227621", "MKwEuc1sc6");
```

2.4.3 批量插入数据

本关任务：向客户表 client 批量插入数据。

将 insert 语句中的 value 字段更换为 select 查询语句即可，代码如下：

```
insert into client select * from new_client;
```

2.4.4 删除没有银行卡的客户信息

本关任务：删除在本行没有银行卡的客户信息。

直接使用 delete 语句和 not exists 子查询即可，代码如下：

```
delete from client
where not exists (
    select
        *
    from bank_card
    where client.c_id = bank_card.b_c_id
);
```

2.4.5 冻结客户资产

本关任务：冻结客户的投资资产。

使用一条 update 语句结合 exists 子查询即可实现。

```
update property
set pro_status = " 冻结"
where exists (
    select
        *
    from client
    where
        c_phone = "13686431238"
        and c_id = pro_c_id
);
```

2.4.6 连接更新

本关任务：根据客户表的内容修改资产表的内容。

根据任务要求使用 update 语句更新即可，代码如下：

```sql
update
    property,
    client
set pro_id_card = c_id_card
where pro_c_id = c_id;
```

2.5 视图

本节的 2 个关卡围绕视图的创建与使用展开。

2.5.1 创建所有保险资产的详细记录视图

本关任务：创建所有保险资产的详细记录视图。

根据任务要求编写 select 语句，并在前面加上“**create view ... as**”即可，代码如下：

```sql
create view v_insurance_detail as
select
    c_name,
    c_id_card,
    i_name,
    i_project,
    pro_status,
    pro_quantity,
    i_amount,
    i_year,
    pro_income,
    pro_purchase_time
from
    client, property, insurance
where
    c_id = pro_c_id
    and pro_type = 2
    and pro_pif_id = i_id;
```

2.5.2 基于视图的查询

本关任务：基于视图 v_insurance_detail 查询每位客户保险资产的总额和保险总收益。

将视图当做普通的表一样 select 查询即可，代码如下：

```sql
select
    c_name,
    c_id_card,
    sum(pro_quantity * i_amount) as insurance_total_amount,
    sum(pro_income) as insurance_total_revenue
from v_insurance_detail
group by c_id_card
order by insurance_total_amount desc;
```

2.6 存储过程与事务

本节的 3 个关卡分别涉及使用流程控制语句的存储过程、使用游标的存储过程和使用事务的存储过程。

2.6.1 使用流程控制语句的存储过程

本关任务：创建一个存储过程，向表 fibonacci 插入斐波拉契数列的前 n 项。

利用“**with recursive ... as**”定义一个递归查询子句，定义初值和递归过程即可，注意特判第一项为 1，代码如下：

```
drop procedure if exists sp_fibonacci;
delimiter $$
create procedure sp_fibonacci(in m int)
begin
    set m = m - 1;
    with recursive cte(id, cur, pre) as (
        select
            0, 0, 0
        union all
        select
            id + 1,
            if (id < 2, 1, cur + pre),
            cur
        from cte
        where id < m
    )
    select
        id n,
        cur fibn
    from cte;
end $$
delimiter ;
```

2.6.2 使用游标的存储过程

该关卡任务已完成，实施情况本报告略过。

2.6.3 使用事务的存储过程

本关任务：编写实现转账功能的存储过程。

根据题意编写判断条件，如果不合法则将返回值置为 0，并直接利用 leave 退出事务。如果合法，则先判断付款方和收款方的卡类型，如果是信用卡则要把转账金额设置为负数，最后使用 update 更新即可。代码如下：

```
delimiter $$
create procedure sp_transfer(
    IN applicant_id int,
    IN source_card_id char(30),
    IN receiver_id int,
    IN dest_card_id char(30),
    IN amount numeric(10,2),
    OUT return_code int)
pro:
BEGIN
    declare s_id, r_id int;
    declare s_type, r_type char(20);
    declare s_b, rcv_amount numeric(10, 2) default amount;
    select
        b_c_id, b_balance, b_type
    into
```

```
        s_id, s_b, s_type
    from bank_card
    where b_number = source_card_id;
    select
        b_c_id, b_type
    into
        r_id, r_type
    from bank_card
    where b_number = dest_card_id;
    if s_id != applicant_id or r_id != receiver_id  or (s_type = " 信用卡" and r_type = " 储蓄卡")
    ↪  or (s_type = " 储蓄卡" and s_b < amount) then
        set return_code = 0;
        leave pro;
    end if;
    if s_type = " 信用卡" then
        set amount = -amount;
    end if;
    if r_type = " 信用卡" then
        set rcv_amount = -rcv_amount;
    end if;
    update bank_card set b_balance = b_balance - amount where b_number = source_card_id;
    update bank_card set b_balance = b_balance + rcv_amount where b_number = dest_card_id;
    set return_code = 1;
END$$
delimiter ;
```

2.7 触发器

本节的 1 个关卡涉及触发器的使用。

2.7.1 为投资表 property 实现业务约束规则-根据投资类别分别引用不同表的主码

本关任务：为资产表 property 编写一个触发器，以实现任务所要求的完整性业务规则。

根据任务要求声明“**BEFORE INSERT ON** property”类型的触发器，然后根据任务要求判断并设置相应的报错信息即可，如果报错信息为空则说明没有错误。代码如下：

```
drop trigger if exists before_property_inserted;
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
    declare tp int default new.pro_type;
    declare id int default new.pro_pif_id;
    declare msg varchar(50);
    if tp = 1 then
        if id not in (select p_id from finances_product) then
            set msg = concat("finances product #", id, " not found!");
        end if;
    elseif tp = 2 then
        if id not in (select i_id from insurance) then
            set msg = concat("insurance #", id, " not found!");
        end if;
    elseif tp = 3 then
        if id not in (select f_id from fund) then
            set msg = concat("fund #", id, " not found!");
        end if;
    else
        set msg = concat("type ", tp, " is illegal!");
    end if;
```

```
    if msg is not null then
        signal sqlstate "45000" set message_text = msg;
    end if;
END$$
delimiter ;
```

2.8 用户自定义函数

本节的 1 个关卡涉及用户自定义函数的定义和使用。

2.8.1 创建函数并在语句中使用它

本关任务：编写一个依据客户编号计算其在本金融机构的存储总额的函数，并在 SELECT 语句使用这个函数。

根据任务要求，函数定义部分的代码如下：

```
delimiter $$
create function get_deposit(client_id int)
returns numeric(10, 2)
begin
^^Ireturn (
        select
            sum(b_balance)
        from bank_card
        where b_type = " 储蓄卡"
        group by b_c_id
        having b_c_id = client_id
    );
end$$
delimiter ;
```

在 select 语句中需要的地方直接调用自定义函数即可，代码如下：

```
select
    *
from (
    select
        c_id_card,
        c_name,
        get_deposit(c_id) total_deposit
    from client
) a
where total_deposit >= 1000000
order by total_deposit desc;
```

2.9 安全性控制

本节的 2 个关卡涉及数据库中的用户、角色和权限等内容。

2.9.1 用户和权限

本关任务：在金融应用场景数据库环境中，创建用户，并给用户授予指定的权限。

根据任务要求合理使用 create, grant, revoke 语句编写指令即可，代码如下：

```
#(1) 创建用户 tom 和 jerry，初始密码均为'123456'；
create user tom identified by "123456";
create user jerry identified by "123456";
#(2) 授予用户 tom 查询客户的姓名，邮箱和电话的权限，且 tom 可转授权限；
grant select (c_mail, c_name, c_phone) on client to tom with grant option;
#(3) 授予用户 jerry 修改银行卡余额的权限；
grant update (b_balance) on bank_card to jerry;
#(4) 收回用户 Cindy 查询银行卡信息的权限。
revoke select on bank_card from Cindy;
```

2.9.2 用户、角色与权限

本关任务：创建角色，授予角色一组权限，并将角色代表的权限授予指定的一组用户。

根据任务要求合理使用 create, grant 语句编写指令即可，代码如下：

```
# (1) 创建角色 client_manager 和 fund_manager；
create user client_manager;
create user fund_manager;
# (2) 授予 client_manager 对 client 表拥有 select,insert,update 的权限；
grant select, insert, update on client to client_manager;
# (3) 授予 client_manager 对 bank_card 表拥有查询除银行卡余额外的 select 权限；
grant select (b_c_id, b_number, b_type) on bank_card to client_manager;
# (4) 授予 fund_manager 对 fund 表的 select,insert,update 权限；
grant select, insert, update on fund to fund_manager;
# (5) 将 client_manager 的权限授予用户 tom 和 jerry；
grant client_manager to tom, jerry;
# (6) 将 fund_manager 权限授予用户 Cindy.
grant fund_manager to Cindy;
```

2.10 并发控制与事务的隔离级别

本节的 6 个关卡涉及数据库中并发控制与事务的隔离级别相关内容，包括隔离级别的设置，事务的开启、提交和回滚等，还通过添加等待代码实现了读脏、不可重复读、幻读等出错场景。

2.10.1 并发控制与事务的隔离级别

本关任务：设置事务的隔离级别。

MySQL 的事务隔离级别从低到高分以下四级：

1. 读未提交（READ UNCOMITTED ）
2. 读已提交（READ COMMITTED ）
3. 可重复读（REPEATABLE READ ）
4. 可串行化（SERIALIZABLE ）

低隔离级别可以支持更高的并发处理，同时占用的系统资源更少，但可能产生数据不一致的情形也更多一些。MySQL 事务隔离级别及其可能产生的问题如表2-2所示。

表 2-2: MySQL 事务隔离级别说明

隔离级别	读脏	不可重复读	幻读
READ UNCOMITTED	√	√	√
READ COMMITTED	×	√	√
REPEATABLE READ	×	×	√
SERIALIZABLE	×	×	×

表2-2说明，最低的隔离级别不能避免读脏、不可重复读和幻读，而最高的隔离级别，可保证多个并发事务的任何调度，都不会产生数据的不一致性，但其代价是并发度最低。

根据任务要求将事务的隔离级别设置为 read uncommitted ，并以 rollback 语句结束事务即可，代码如下：

```
# 设置事务的隔离级别为 read uncommitted
set session transaction isolation level read uncommitted;
-- 开启事务
start transaction;
insert into dept(name) values('运维部');
# 回滚事务:
rollback;
```

2.10.2 读脏

本关任务：选择合适的事务隔离级别，构造两个事务并发执行时，发生“读脏”现象。

根据表2-2，要发生“读脏”需要将隔离级别设置为 read uncommitted 。同时，确保事务 1 读航班余票发生在在事务 2 修改之后，事务 2 撤销发生在事务 1 读取之后，据此设置事务休眠时间，即可发生“读脏”。

事务 1 代码如下：

```
-- 事务 1:
use testdb1;
-- 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;

start transaction;

-- 时刻 2 - 事务 1 读航班余票, 发生在事务 2 修改之后
-- 添加等待代码, 确保读脏
set @n = sleep(1);

select tickets from ticket where flight_no = 'CA8213';
commit;
```

事务 2 代码如下：

```
-- 事务 2
use testdb1;
-- 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;
start transaction;
-- 时刻 1 - 事务 2 修改航班余票
```

```
update ticket set tickets = tickets - 1 where flight_no = 'CA8213';

-- 时刻 3 - 事务 2 取消本次修改
-- 请添加代码，使事务 1 在事务 2 撤销前读脏;
set @n = sleep(2);

rollback;
```

2.10.3 不可重复读

该关卡任务已完成，实施情况本报告略过。

2.10.4 幻读

该关卡任务已完成，实施情况本报告略过。

2.10.5 主动加锁保证可重复读

本关任务：在事务隔离级别较低的 read uncommitted 情形下，通过主动加锁，保证事务的一致性。

事务 2 尝试在事务 1 两次读之间出票一张，但由于事务 1 代码中第 6 行加了锁，事务 2 的更新代码无法在事务 1 的锁释放之前执行，因此实现了可重复读，即事务 1 的两次读取结果是一致的。

事务 1 代码如下：

```
-- 事务 1:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
-- 第 1 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455' for update;
set @n = sleep(5);
-- 第 2 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455' for update;
commit;
-- 第 3 次查询所有航班的余票，发生在事务 2 提交后
set @n = sleep(1);
select * from ticket;
```

事务 2 代码如下：

```
-- 事务 2:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
set @n = sleep(1);
-- 在事务 1 的第 1, 2 次查询之间，试图出票 1 张 (航班 MU2455):
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

2.10.6 可串行化

本关任务：选择除 serializable (可串行化) 以外的任何隔离级别，保证两个事务并发执行的结果是可串行化的。

直接让事务 1 休眠到事务 2 执行完后再执行即可。

事务 1 代码如下：

```
-- 事务 1:
use testdb1;
start transaction;
set @n = sleep(5);
select tickets from ticket where flight_no = 'MU2455';
select tickets from ticket where flight_no = 'MU2455';
commit;
```

事务 2 代码如下：

```
-- 事务 2:
use testdb1;
start transaction;
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

2.11 备份 + 日志：介质故障与数据库恢复

本节的 2 个关卡涉及数据库的备份与恢复方法。

2.11.1 备份与恢复

本关任务：备份数据库，然后再恢复它。

使用 mysqldump 指令将服务器上的数据库 residents 备份至文件 residents_bak.sql 中：

```
mysqldump -h127.0.0.1 -uroot --databases residents > residents_bak.sql
```

使用 mysql 指令根据备份文件 residents_bak.sql 还原数据库：

```
mysql -h127.0.0.1 -uroot < residents_bak.sql
```

2.11.2 备份 + 日志：介质故障的发生与数据库的恢复

本关任务：模拟介质故障的发生，以及如何利用备份和备份之后的日志恢复数据库。

由于需要对数据库 train 作逻辑备份并新开日志文件,所以需要在上一关的基础上在 mysqldump 指令中加入 `--flush-logs` 参数来新开日志文件：

```
mysqldump -h127.0.0.1 -uroot --flush-logs --databases train > train_bak.sql;
```

系统在故障发生的时间点会将备份后新开的日志文件保存为 log/binlog.000018 。为了保证两次发生的业务数据都不丢失，除了根据备份文件 train_bak.sql 恢复数据库外，还需要利用 mysqlbinlog 指令从 log/binlog.000018 文件中恢复数据库：

```
mysql -h127.0.0.1 -uroot < train_bak.sql;
mysqlbinlog --no-defaults log/binlog.000018 | mysql -uroot;
```

值得注意的细节是 MySQL 有一个默认设置 “default-character-set=utf8” 与 binlog 文件冲突，所以需要在 mysqlbinlog 指令中加入 `--no-defaults` 参数，表示不使用 MySQL 默认的设置

2.12 数据库设计与实现

本节的 3 个关卡涉及数据库设计与实现相关内容，包括从概念模型到 MySQL 实现、E-R 图的构建、建模工具的使用等。

2.12.1 从概念模型到 MySQL 实现

该关卡任务已完成，实施情况本报告略过。

2.12.2 从需求分析到逻辑模型

本关任务：根据应用场景业务需求描述，完成 E-R 图，并转换成关系模式。

根据任务要求描述转换得到的 E-R 图如图2.1所示。

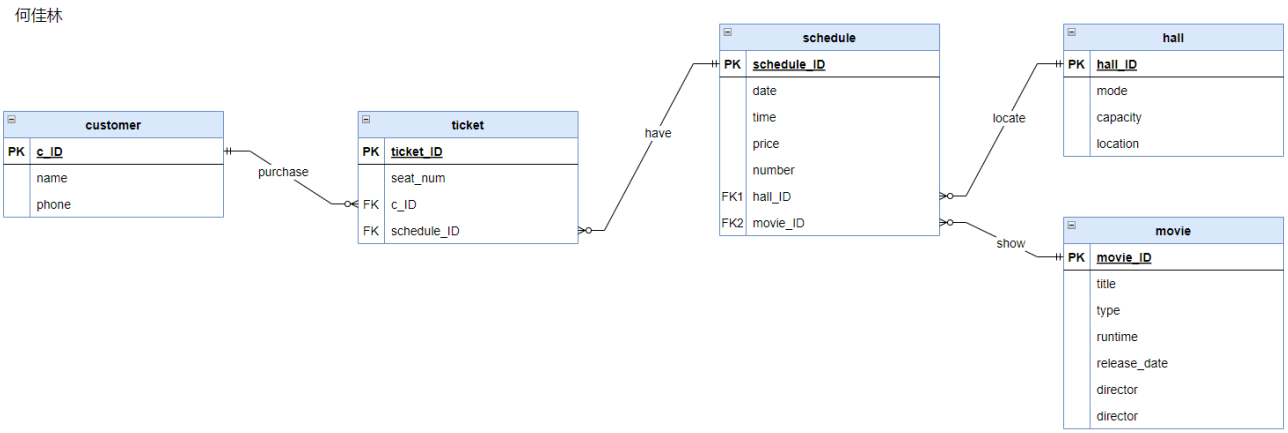

图 2.1: 影院管理系统 E-R 图

该 E-R 图对应的关系模式描述如下：

```
movie(movie_ID, title, type, runtime, release_date, director, starring), primary key:(movie_ID);
customer(c_ID, name, type, phone), primary key:(c_ID);
hall(hall_ID, mode, capacity, location), primary key:(hall_ID);
schedule(schedule_ID, date, time, price, number, movie_ID, hall_ID), primary key:(schedule_ID),
↪  foreign key(movie_ID, hall_ID);
ticket(ticket_ID, seat_num, schedule_ID), primary key(ticket_ID), foreign key(schedule_ID);
```

2.12.3 建模工具的使用

该关卡任务已完成，实施情况本报告略过。

2.12.4 制约因素分析与设计

在从实际问题的建模到数据库的概念模型和逻辑模型的构建过程中，需要考虑若干制约因素。以机票订票系统为例，系统需要考虑到旅客的实际情况，旅客可以多次乘坐飞机，一张机票肯定是某个用户为某个特定的旅客购买的特定航班的机票，所以机票信息不仅跟乘坐人有关，同时还需要记录购买人信息 (虽然两者有时是同一人) 。此外，对于系统的权限也存在若干要求，例如实体 “用户” 就刻印根据权限分成两类，用户分两类：普通用户可以订票，管理用户有权限维护和管理整个系统的运营。

2.12.5 工程师责任及其分析

社会方面，工程师应该能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。安全方面，工程师应该尽可能考虑系统中存在的安全漏洞，安全性是所有系统用户关心的重要命题；科学发展方面，工程师应该能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有效的结论。

2.13 数据库应用开发 (JAVA 篇)

本节的 7 个关卡从 JDBC 体系结构出发，涉及 JAVA 开发数据库应用的基本知识。

2.13.1 JDBC 体系结构和简单的查询

本关任务：查询 client 表中邮箱非空的客户信息，列出客户姓名，邮箱和电话。

使用 Java 的 Class.forName() 方法，将驱动程序的类文件动态加载到内存中，并将其自动注册。

加载驱动程序后，使用 `DriverManager.getConnection(String url, String user, String password)` 方法建立连接。

在使用 Statement 对象执行 SQL 语句之前，需要使用 Connection 对象的 createStatement() 方法创建 Statement 的一个实例。

创建 Statement 对象后，可以使用它来执行一个 SQL 语句，其中有三个执行方法：

1. boolean execute(String SQL) ：如果可以检索到 ResultSet 象，则返回一个布尔值 true ；否则返回 false 。使用此方法执行 SQL DDL 语句或需要使用真正的动态 SQL 时。
2. int executeUpdate(String SQL) ：返回受 SQL 语句执行影响的行数。使用此方法执行预期会影响多个行的 SQL 语句，例如 INSERT, UPDATE 或 DELETE 语句。
3. ResultSet executeQuery(String SQL) ：返回一个 ResultSet 对象。当希望获得结果集时，使用此方法，就像使用 SELECT 语句一样。

本关需要选用第 3 个执行方法，使用 resultSet.next() 遍历 ResultSet 输出相应的信息即可，代码如下：

```
public class Client {
    static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    static final String DB_URL =
    ↪  "jdbc:mysql://127.0.0.1:3306/finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezon
    static final String USER = "root";
    static final String PASS = "123123";

    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            Class.forName(JDBC_DRIVER);
            connection = DriverManager.getConnection(DB_URL, USER, PASS);
            statement = connection.createStatement();
            resultSet = statement.executeQuery("select c_name, c_mail, c_phone from client where
            ↪  c_mail is not null");
            System.out.println(" 姓名\t 邮箱\t\t\t\t 电话");
            while (resultSet.next()) {
                System.out.print(resultSet.getString("c_name") + "\t");
                System.out.print(resultSet.getString("c_mail") + "\t\t");
                System.out.println(resultSet.getString("c_phone"));
            }
         } catch (ClassNotFoundException e) {
            System.out.println("Sorry,can`t find the JDBC Driver!");
            e.printStackTrace();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        } finally {
            try {
                if (resultSet != null)
                    resultSet.close();
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }
    }
}
```

2.13.2 用户登录

本关任务：编写客户登录程序，提示用户输入邮箱和密码，并判断正确性，给出适当的提示信息。

用 PreparedStatement 类执行 SQL 语句，把 SQL 语句中变化的部分当成参数，以防御 SQL 注入攻击。代码如下：

```
statement = connection.createStatement();
String sql = "select * from client where c_mail = ? and c_password = ?";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setString(1, loginName);
ps.setString(2, loginPass);
```

```
resultSet = ps.executeQuery();
if (resultSet.next())
    System.out.println(" 登录成功。");
else
    System.out.println(" 用户名或密码错误！");
```

2.13.3 添加新用户

本关任务：编程完成向客户表 client 插入记录的方法。

编写 SQL 语句，用 PreparedStatement 类执行即可，代码如下：

```
public static int insertClient(Connection con, int c_id, String c_name, String c_mail,
                               String c_id_card, String c_phone, String c_password) {
    String sql = "insert into client values (?, ?, ?, ?, ?, ?)";
    try {
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, c_id);
        ps.setString(2, c_name);
        ps.setString(3, c_mail);
        ps.setString(4, c_id_card);
        ps.setString(5, c_phone);
        ps.setString(6, c_password);
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
```

2.13.4 银行卡销户

该关卡任务已完成，实施情况本报告略过。

2.13.5 客户修改密码

本关任务：编写修改客户登录密码的的方法。

先判断用户是否存在，再判断密码是否正确，最后利用 update 语句修改密码。代码如下：

```
public static int passwd(Connection connection, String mail, String password, String newPass) {
    String sql = "select * from client where c_mail = ?";
    try {
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setString(1, mail);
        ResultSet res = ps.executeQuery();
        if (res.next()) {
            if (password.equals(res.getString("c_password"))) {
                sql = "update client set c_password = ? where c_mail = ? and c_password = ?";
                ps = connection.prepareStatement(sql);
                ps.setString(1, newPass);
                ps.setString(2, mail);
                ps.setString(3, password);
                ps.executeUpdate();
                return 1;
            } else
                return 3;
        } else
```

```
            return 2;

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}
```

2.13.6 事务与转账操作

本关任务：编写一个银行卡转账的方法。

根据题意编写判断条件，如果不合法则直接**return false**，然后判断收款方卡类型是否为信用卡决定是否要把收款金额设置为负数。代码如下：

```
public static boolean transferBalance(Connection con, String sourceCard, String destCard, double amount) {
    try {
        String sql = "select * from bank_card where b_number = ?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, sourceCard);
        ResultSet res = ps.executeQuery();
        if (!res.next() || res.getString("b_type").equals(" 信用卡") || res.getDouble("b_balance") < amount)
            return false;
        ps = con.prepareStatement(sql);
        ps.setString(1, destCard);
        res = ps.executeQuery();
        if (!res.next())
            return false;
        double rcv_amount = res.getString("b_type").equals(" 信用卡") ? -amount : amount;
        sql = "update bank_card set b_balance = b_balance + ? where b_number = ?";
        ps = con.prepareStatement(sql);
        ps.setDouble(1, -amount);
        ps.setString(2, sourceCard);
        ps.executeUpdate();
        ps = con.prepareStatement(sql);
        ps.setDouble(1, rcv_amount);
        ps.setString(2, destCard);
        ps.executeUpdate();
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

2.13.7 把稀疏表格转为键值对存储

本关任务：将一个稀疏的表中有保存数据的列值，以键值对 “(列名, 列值)” 的形式转存到另一个表中，这样可以直接丢失没有值列。

选出 entrance_exam 表中所有数据，对每个学生，并枚举所有学科，如果该学生存在该学科的成绩，就将其插入 sc 表中。代码如下：

```
public static void insertSC(Connection con, int sno, String col_name, int col_value) {
    try {
```

```
        String sql = "insert into sc values (?, ?, ?)";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, sno);
        ps.setString(2, col_name);
        ps.setInt(3, col_value);
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) throws Exception {
    Class.forName(JDBC_DRIVER);
    Connection con = DriverManager.getConnection(DB_URL, USER, PASS);
    String[] subject = {"chinese", "math", "english", "physics", "chemistry", "biology",
    ↪  "history", "geography", "politics"};
    try {
        ResultSet res = con.createStatement().executeQuery("select * from entrance_exam");
        while (res.next()) {
            int sno = res.getInt("sno"), score;
            for (String sub : subject) {
                score = res.getInt(sub);
                if (!res.wasNull())
                    insertSC(con, sno, sub, score);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

---

3 课程总结

本次课程实验从数据库、表的定义出发，分别完成了表的完整性约束的创建和修改、数据查询、数据的插入、修改与删除、视图的创建与使用、存储过程与事务、触发器、用户自定义函数、安全性控制、并发控制与事务的隔离级别、数据库的备份与日志、数据库的设计与实现和数据库应用开发等 13 个实训实验，总计 62 个关卡。

在课程实验的实施过程中，通过合理使用搜索引擎查阅相关资料，学习 MySQL 的一些复杂语法和一些经典问题的简单处理方法简化编程，让我能够熟练使用 MySQL 实现一些复杂的查询，同时也感受到了 MySQL 的灵活性。在基于 JDBC 的数据库应用开发中，我切实体会到了开发过程中代码安全的重要性，如果不遵守规范，则有可能会因为 SQL 注入攻击造成非常严重的后果。

此次课程实验内容充实完整，引导性强，完成实验的收获也很大，希望之后可以引入“数据库的索引 B+ 树实现”，能够让学生更加深入地理解数据库是如何运作的。

最后，衷心地感谢课程组老师们对实验内容的精心打造以及实验过程中的悉心指导！