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：	校交 1902 班
学　　号：	U201912633
姓　　名：	张睿
指导教师：	袁平鹏

分数	
教师签名	

2022　年　6 月　29 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)。

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

使用相关的 sql 语句对数据库，表和完整性约束进行定义。

2.1.1 创建数据库

该关卡任务已完成，实施情况本报告略过

2.1.2 创建表及表的主码约束

```
CREATE TABLE staff (
 staffNo INT,
 staffName VARCHAR(32),
 gender CHAR(1),
 dob date,
 Salary numeric(8,2),
 deptNo INT,
 CONSTRAINT PK_staff PRIMARY KEY (staffNo),
 CONSTRAINT FK_staff_deptNo  FOREIGN KEY (deptNo) REFERENCES dept(deptNo)
);
```

注意约束的约定命名规则：

- 主码约束以“PK_”打头，后跟表名，一个表只会有一个主码约束；
- 外码约束以“FK_”打头，后跟表名及列名；
- CHECK 约束以“CK_”打头，后跟表名及列名

2.1.3 创建外码约束

该关卡任务已完成，实施情况本报告略过

2.1.4 CHECK 约束

该关卡任务已完成，实施情况本报告略过

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 表结构和完整性约束的修改

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

ALTER TABLE your_table **RENAME TO** my_table;

2.2.2 添加与删除字段

ALTER TABLE orderDetail **DROP COLUMN** orderDate;
ALTER TABLE orderDetail **add COLUMN** unitPrice numeric(10,2);

2.2.3 修改字段

ALTER TABLE addressBook **MODIFY** QQ CHAR(12)

2.2.4 添加、删除与修改约束

该关卡任务已完成，实施情况本报告略过

2.3 数据查询

这个实践项目主要是数据的查询。每个查询任务都需要用一条 SQL 语句完成某个查询（允许嵌套,允许使用衍生表，但只能是一条 SQL 语句，而不是两条或多条语句，除非关卡有多个任务要求）。数据集是一个事先创建并已初始化的数据

库：某银行金融服务场景模拟数据库。你只需写出能实现查询要求的那条 SQL 语句即可，评测程序每次评测时都会初始化实验环境，然后在多个数据集上运行你的代码文件，并与预期的结果进行比对。

2.3.1 金融应用场景介绍,查询客户主要信息

该关卡任务已完成，实施情况本报告略过

2.3.2 邮箱为 null 的客户

该关卡任务已完成，实施情况本报告略过

2.3.3 既买了保险又买了基金的客户

该关卡任务已完成，实施情况本报告略过

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

该关卡任务已完成，实施情况本报告略过

2.3.7 未购买任何理财产品的武汉居民

LIKE 进行字符串匹配， %匹配多个字符 _ 匹配单个字符

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

TopN 查询核心思想，找有几个比当前的高。

```
SELECT a1.c_name, a1.c_id_card, a1.total_income
FROM (
    SELECT c_id, c_name, c_id_card, SUM(pro_income) AS total_income
    FROM client, property
    WHERE c_id=pro_c_id AND pro_status='可用'
    GROUP BY c_id
) a1
WHERE c_id in (
    SELECT first.c_id
```

```
    FROM (
        SELECT c_id, c_name, c_id_card, SUM(pro_income) AS total_income
        FROM client, property
        WHERE c_id=pro_c_id
        GROUP BY c_id

    ) first left join    (
        SELECT c_id, c_name, c_id_card, SUM(pro_income) AS total_income
        FROM client, property
        WHERE c_id=pro_c_id AND pro_status='可用'
        GROUP BY c_id
    ) second on first.total_income < second.total_income
    GROUP BY first.c_id
    HAVING COUNT(*)<=2      //最多仅有两个比当前的高→top3
)
ORDER BY a1.total_income DESC;
```

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

COALASCE 可以将空值转为特定值

GROUP BY 后 SELECT 只能 select group by 的属性，或者使用聚集函数。

2.3.13 客户总资产

该关卡任务已完成，实施情况本报告略过

2.3.14 第 N 高问题

该关卡任务已完成，实施情况本报告略过

2.3.15 基金收益两种方式排名

名次连续或者不连续，只用连接之后看有几个比他大的时候去重即可。

比如：

- 同数同名次，总排名不连续。例如 300、200、200、150、150、100 的排名结果为 1, 2, 2, 4, 4, 6；对于 150，前面有 3 个比他大的，所以排第 4 名
- 同数同名次，总排名连续。例如 300、200、200、150、150、100 的排名结果为 1, 2, 2, 3, 3, 4。对于 150， 前面有 3 个比他大的，但去重后只有两个，所以排第 3 名。

```
-- (1) 基金总收益排名(名次不连续)
SELECT t1.c_id AS pro_c_id, t1.total_revenue, COUNT(t2.c_id)+1 AS rank
FROM (
    SELECT c_id, SUM(pro_income) AS total_revenue
    FROM client, property
    WHERE c_id=pro_c_id AND pro_type=3
    GROUP BY c_id
) t1 LEFT JOIN (
    SELECT c_id, SUM(pro_income) AS total_revenue
    FROM client, property
    WHERE c_id=pro_c_id AND pro_type=3
    GROUP BY c_id
) t2 ON t1.total_revenue < t2.total_revenue
GROUP BY t1.c_id, t1.total_revenue
ORDER BY rank, t1.c_id;
```

```
-- (2) 基金总收益排名(名次连续)
SELECT t1.c_id AS pro_c_id, t1.total_revenue, COUNT(t2.total_revenue)+1 AS rank
FROM (
    SELECT c_id, SUM(pro_income) AS total_revenue
    FROM client, property
    WHERE c_id=pro_c_id AND pro_type=3
    GROUP BY c_id
) t1 LEFT JOIN (
    SELECT DISTINCT SUM(pro_income) AS total_revenue    // DISTINCT 去重
    FROM client, property
    WHERE c_id=pro_c_id AND pro_type=3
    GROUP BY c_id
) t2 ON t1.total_revenue < t2.total_revenue
GROUP BY t1.c_id, t1.total_revenue
ORDER BY rank, t1.c_id;
```

2.3.16 持有完全相同基金组合的客户

元组合并使用

array_to_string(ARRAY(SELECT DISTINCT unnest(array_agg(pro_pif_id order by pro_pif_id))), ',')

其中，array_agg 将 pro_pif_id 变为数组并排序，unnest 将数组展开变成表，表中有一列元素，该列元素为数组元素，select distinct 将其去重，转换成 array 后使用 array_to_string 以，分割变成字符串输出。

```
SELECT t1.pro_c_id AS c_id1, t2.pro_c_id AS c_id2
FROM (
   SELECT pro_c_id, array_to_string(ARRAY(SELECT DISTINCT unnest(array_ag
g(pro_pif_id order by pro_pif_id))), ',') AS fund
   FROM property
   WHERE pro_type=3
   GROUP BY pro_c_id
) t1, (
   SELECT pro_c_id, array_to_string(ARRAY(SELECT DISTINCT unnest(array_ag
g(pro_pif_id order by pro_pif_id))), ',') AS fund
   FROM property
   WHERE pro_type=3
   GROUP BY pro_c_id
) t2
WHERE t1.pro_c_id<t2.pro_c_id AND t1.fund=t2.fund
```

2.3.17 购买基金的高峰期

日期函数：extract('year' from t1.pro_purchase_time) 其中‘year’可以换为 month 等

找连续 3 天思想：按日期顺序编号，然后将 3 张相同的表连起来，求编号连续且满足条件的 3 天即可。

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

注意 case 的使用:

Case
When …then…
End

```
SELECT
    CASE
        WHEN t3.week=6 THEN 1
        WHEN t3.week=7 THEN 2
        WHEN t3.week=8 THEN 3
        WHEN t3.week=9 THEN 4
        END AS week_of_trading,
    CASE t3.dow WHEN 1 THEN t3.amount END AS monday,
    CASE t3.dow WHEN 2 THEN t3.amount END AS tuesday,
    CASE t3.dow WHEN 3 THEN t3.amount END AS wendnesday,
    CASE t3.dow WHEN 4 THEN t3.amount END AS thursday,
    CASE t3.dow WHEN 5 THEN t3.amount END AS friday
from (
    SELECT *
    FROM (
        SELECT t1.pro_purchase_time, SUM(t1.pro_quantity*t2.f_amount) AS amount, extract('dow' from t1.pro_purchase_time) AS dow, extract('week' from t1.pro_purchase_time) AS week
        FROM property t1, fund t2
        WHERE t1.pro_pif_id=t2.f_id AND t1.pro_type=3 AND extract('month' from t1.pro_purchase_time)=2 AND extract('year' from t1.pro_purchase_time)=2022
        GROUP BY t1.pro_purchase_time
    ) t4
    ORDER BY week, t4.pro_purchase_time
) t3
```

2.4 数据的插入、修改与删除

2.4.1 插入多条完整的客户信息

该关卡任务已完成，实施情况本报告略过

2.4.2 插入不完整的客户信息

```
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

```
INSERT INTO client
  (SELECT * FROM new_client);
```

2.4.4 删除没有银行卡的客户信息

该关卡任务已完成，实施情况本报告略过

2.4.5 冻结客户资产

该关卡任务已完成，实施情况本报告略过

2.4.6 连接更新

这里需要注意 set 可以接子查询

```
UPDATE property
SET pro_id_card=(SELECT c_id_card FROM client WHERE property.pro_c_id=client.c_id)
```

2.5 视图

基于 openGauss 的视图的创建与使用

2.5.1 创建所有保险资产的详细记录视图

该关卡任务已完成，实施情况本报告略过

2.5.2 基于视图的查询

该关卡任务已完成，实施情况本报告略过

2.6 存储过程与事务

存储过程（Stored Procedure）是一种可编程的数据对象，是由一组为了完成特定功能的 SQL 语句和控制结构组成的复杂程序。存储过程经编译后存储在数据库中，外部程序通过指定存储过程的名字并给定参数（如果存储过程定义了参数）来调用它。

2.6.1 使用流程控制语句的存储过程

这一关需要注意：

1. 变量的定义与赋值；

定义：DECLARE i int default 2

赋值：a. i := i+1 b. select .. into i from …

2. 存储过程的定义；

CREATE PROCEDURE proc1()

as

BEGIN

insert into test values(1);

END;

/

对于参数有三种类型：

- IN：输入参数，也是默认模式，表示该参数的值必须在调用存储过程时指定，在存储过程中修改该参数的值不能被返回；
- OUT：输出参数，该值可在存储过程内部被改变，并可返回；
- IN OUT：输入输出参数，调用时指定，并且可被改变和返回。

2.6.2 使用游标的存储过程

这一关需要注意游标的使用。

首先是定义的顺序，DECLARE 定义的顺序要求如下： 变量->游标->特情处理。

其次是游标的定义与使用：

- 定义游标：

  CURSOR cursor_name FOR select_statement
- OPEN cursor_name
- FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name]
- EXIT WHEN C1%NOTFOUND;
- CLOSE cursor_name

需要注意的是，游标会有特情处理的，因为有可能取空。

对于这一关来说，主要的逻辑如下：

-- 编写一存储过程，自动安排某个连续期间的大夜班的值班表:

```
create procedure sp_night_shift_arrange(in start_date date, in end_date date)
AS
    ….
    CURSOR cur_doctor FOR (
```

```
        SELECT e_name,e_type
        FROM employee
        WHERE e_type=1 or e_type=2
    );
    CURSOR cur_nurse FOR (
        SELECT e_name
        FROM employee
        WHERE e_type=3
    );
// 这里定义了两个游标，游标的表是 employee 的派生表，通过这样的方式，实现把护士和医生分离开来。
begin
    ……
    WHILE cur_date<=end_date LOOP
        IF extract('DOW' FROM cur_date)>=1 AND extract('DOW' FROM cur_date)<=5    // 周一到周五
        THEN
            IF NOT exists(SELECT * FROM night_shift_schedule WHERE cur_date=n_date)    /
            THEN
                FETCH cur_doctor INTO doc_name,doc_type;   // 找一个医生
                IF cur_doctor%FOUND=0   // 找完了需要重新轮班
                THEN
                    close cur_doctor;
                    open cur_doctor;
                    FETCH cur_doctor INTO doc_name,doc_type;
                END IF;
            END IF;

            FETCH cur_nurse INTO nurse1; // 找两个护士
            IF cur_nurse%FOUND=0
            THEN
                close cur_nurse;
                open cur_nurse;
                FETCH cur_nurse INTO nurse1;
```

```
            END IF;
            FETCH cur_nurse INTO nurse2;
            IF cur_nurse%FOUND=0
            THEN
                close cur_nurse;
                open cur_nurse;
                FETCH cur_nurse INTO nurse2;
            END IF;
            IF NOT exists(SELECT * FROM night_shift_schedule WHERE
cur_date=n_date)
            THEN
                INSERT INTO night_shift_schedule VALUES(cur_date, doc_name,
nurse1, nurse2);
            ELSE
                UPDATE night_shift_schedule
                SET n_nurse1_name=nurse1, n_nurse2_name=nurse2
                WHERE n_date=cur_date;
            END IF;
        ELSE
            FETCH cur_doctor INTO doc_name,doc_type;
            IF cur_doctor%FOUND=0
            THEN
                close cur_doctor;
                open cur_doctor;
                FETCH cur_doctor INTO doc_name,doc_type;
            END IF;
            FETCH cur_nurse INTO nurse1;
            IF cur_nurse%FOUND=0
            THEN
                close cur_nurse;
                open cur_nurse;
                FETCH cur_nurse INTO nurse1;
            END IF;
            FETCH cur_nurse INTO nurse2;
            IF cur_nurse%FOUND=0
```

```
            THEN
                close cur_nurse;
                open cur_nurse;
                FETCH cur_nurse INTO nurse2;
            END IF;

            IF doc_type=2
            THEN
                INSERT INTO night_shift_schedule VALUES(cur_date, doc_name,
nurse1, nurse2);
            ELSE
                IF extract('DOW' FROM cur_date)=0
                THEN
                    INSERT INTO night_shift_schedule
VALUES(cur_date+integer'1', doc_name, 'temp', 'temp');
                ELSE
                    INSERT INTO night_shift_schedule
VALUES(cur_date+integer'2', doc_name, 'temp', 'temp');
                END IF;

                FETCH cur_doctor INTO doc_name,doc_type;
                IF cur_doctor%FOUND=0
                THEN
                    close cur_doctor;
                    open cur_doctor;
                    FETCH cur_doctor INTO doc_name,doc_type;
                END IF;
                INSERT INTO night_shift_schedule VALUES(cur_date, doc_name,
nurse1, nurse2);
            END IF;

        END IF;
        cur_date := cur_date + integer'1';
    END LOOP;
```

```
    DELETE FROM night_shift_schedule
    WHERE n_date > end_date;

end;
```
// 这里需要注意，如果当前日期是周末，此时不能安排主任，所以如果医生不是主任则直接插入，如果医生是主任，则需要调下一个非主任医生，并把当前的主任根据今天是周日或周六往后推 1 天或者 2 天，先把这个主任插入进去，这也是为什么前面需要 not exists 的原因，主要是因为这里提前插入了之后的记录， 并且由于只有医生，所以护士需要利用 update 更改记录。

2.6.3 使用事务的存储过程

该关卡任务已完成，实施情况本报告略过

2.7 触发器

触发器(trigger)是对约束(constraints)的补充，它用程序来完成 constraint 实现不了的业务规则。我们知道，当执行 insert,delete 或 update 语句时，DBMS 将检查该语句是否会导致数据违返约束。同样的，如果在表上定义了 insert,delete 或 update 事件驱动的触发器，那么当执行 insert,delete 或 update 语句时，还会激活相应的触发器，以检查数据的完整性。

约束和触发器共同构成 DBMS 的完整性子系统。

本实训的任务是用 create trigger 语句，创建符合任务要求的触发器。

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

这一关主要是要熟悉触发器的语法：

```
CREATE  TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
EXECUTE PROCEDURE TRI_INSERT_FUNC();
```
其中，before 指定了触发器的触发时机在插入之前触发，本关触发器的业务逻辑较为简单，不过多赘述。

2.8 户自定义函数

2.8.1 创建函数并在语句中使用它

注意创建函数的语法。

```
create function function_name([para data_type[,...]])
returns data_type
begin
    function_body;
    return expression;
end
```

2.9 安全性控制

安全性一般要从全方位、多层次考虑具体的安全措施，比如不仅要考虑 OpenGauss 的安全性，还要考虑服务器主机的安全性(如防 eavesdropping, altering, playback, 以及 DoS 攻击等）。本实训仅考察 OpenGauss 安全性中的一部分：存取控制。

OpenGauss 采用自主存取控制策略，主要机制有：用户，数据对象，权限，角色，授权，收回权限等。

2.9.1 用户和权限

该关卡任务已完成，实施情况本报告略过

2.9.2 用户、角色与权限

这一关主要要熟悉创建角色并授权的语法。

```
-- 请填写语句，完成以下功能：
-- (1) 创建角色 client_manager 和 fund_manager；
CREATE ROLE client_manager IDENTIFIED BY 'hust_1234';
CREATE ROLE fund_manager IDENTIFIED BY 'hust_1234';
-- (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_number,b_type,b_c_id)
ON bank_card
TO client_manager;
-- (4) 授预 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

由于并发在实训中不可控制，本关主要是通过设置睡眠时间来观察并发的情况并模拟不同并发控制可能出现的问题。

2.10.1 不可重复读

时刻	事务 1	事务 2
1	insert into result(t,tickets) select 1 t, tickets from ticket where fli ght_no = 'CZ5525';	Pg_sleep(1)
2	Pg_sleep(1)	insert into result(t,tickets) select 2 t, tickets from ticket where fli ght_no = 'CZ5525';
3	update ticket set tickets = tickets - 1 w here flight_no = 'CZ5525'; insert into result(t,tickets) select 1 t, tickets from ticket where fli ght_no = 'CZ5525';	Pg_sleep(1)
4	Pg_sleep(1)	insert into result(t,tickets) select 2 t, tickets from ticket where fli ght_no = 'CZ5525';

		update ticket set tickets = tickets - 1 w here flight_no = 'CZ5525';
5	Pg_sleep(1)	insert into result(t,tickets) select 2 t, tickets from ticket where fli ght_no = 'CZ5525'; commit
6	commit insert into result(t,tickets) select 1 t, tickets from ticket where fli ght_no = 'CZ5525';	

2.10.2 幻读

该关卡任务已完成，实施情况本报告略过

2.10.3 主动加锁保证可重复读

需要注意加锁的语法：

select * from t1,t2 for share of t1 for update of t2

即加锁是对表这个粒度进行加锁，锁在事务结束时释放。

2.10.4 可串行化

主要是模拟可串行化，t1 最开始睡眠即可，不多赘述。

2.11备份+日志：介质故障与数据库恢复

和大多数 DBMS 一样，OpenGauss 在事务机制的保障下，利用备份、日志文件实现恢复。在生产环境中，通常需要部署多台服务器，可以利用复制、镜像、master-slave 等机制保障数据的安全性、可用性，同时提供高并发服务。

本实训在一个单一服务器实例上进行，只作一个基本的训练。

2.11.1 备份与恢复

gs_dump -U gaussdb -W'Passwd123@123' -h localhost -p5432 residents -f

residents_bak.sql -F t
gs_restore -U gaussdb -W'Passwd123@123' residents_bak.sql -p5432 -d residents -F t

参数	参数说明	举例
-U	连接数据库的用户名。	-U gaussdb
-W	指定用户连接的密码。	-W 'Passwd123@123'
-f	将导出文件发送至指定目录文件夹。	无
-p	指定服务器所侦听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
-d	需要导出的数据库名称.	-d residents
-F	选择导出文件格式。-F 参数值如下：（1）p：纯文本格式（2）c：自定义归档（3）d：目录归档格式（4）t：tar 归档格式	-F t

2.12数据库设计与实现

2.12.1 从概念模型到 MySQL 实现

该任务关卡跳过

2.12.2 从需求分析到逻辑模型

该任务关卡跳过

2.12.3 建模工具的使用

该任务关卡跳过

2.12.4 制约因素分析与设计

该任务关卡跳过

2.12.5 工程师责任及其分析

该任务关卡跳过

2.13数据库应用开发(JAVA 篇)

数据库应用开发，系指利用高级语言开发数据库应用系统，这些应用系统可是 C/S 架构的，也可以是 B/S 架构的(即 web 应用)，开发的语言可是 C++,JAVA，PHP，JSP，Python 等。

本实训通过几个简单的练习让大家掌握 JAVA 开发数据库应用的基本知识。

实训环境为 Java1.8 和 OpenGauss 2.1.0。平台提供的操作系统为 Ubuntu 18.0.4 。

2.13.1 JDBC 体系结构和简单的查询

```
import java.sql.*;

public class Client {
    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            Class.forName("org.postgresql.Driver");   // 注册类

            String URL = "jdbc:postgresql://localhost:5432/postgres";
            String USER = "gaussdb";
            String PASS = "Passwd123@123";
            connection = DriverManager.getConnection(URL, USER, PASS);

            statement = connection.createStatement();

            resultSet = statement.executeQuery("select * from client where c_mail is
not null;");   // 可以直接查询

            System.out.printf("姓名\t 邮箱\t\t\t\t 电话\n");
            while(resultSet.next()) {     // 果果 resultset.next 获取元组
                System.out.print(resultSet.getString("c_name"));
```

```
                System.out.print("\t");
                System.out.print(resultSet.getString("c_mail"));
                System.out.print("\t\t");
                System.out.print(resultSet.getString("c_phone"));
                System.out.print("\n");
            }

        } 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();    // 全部都要显示关闭
                }

                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }
    }
}
```

2.13.2 用户登录

还有另一种可设参数的查询方法

```
        String sql = "select * from client where c_mail = ? and c_password = ?;";
// ? 为占位符
        preparedStatement = connection.prepareStatement(sql);
```

```
preparedStatement.setString(1,loginName);  // 也可以设置参数，从 1 开始编号
preparedStatement.setString(2,loginPass);
resultSet = preparedStatement.executeQuery();
```

2.13.3 添加新客户

该关卡任务已完成，实施情况本报告略过

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

```
if(rs.next()){
        if(rs.getString("c_password").trim().compareTo(password) == 0){
// 这里需要注意要 trim 一下，因为数据库对该列的定义有可能是定长 char，会用空格或 tab 等填充满导致对比出错
            sql = "update client set c_password = ? where c_mail = ?";
            ps = connection.prepareStatement(sql);
            ps.setString(1, newPass);
            ps.setString(2, mail);
            n = ps.executeUpdate();
```

2.13.6 事务与转账操作

JDBC 缺省情形下，Statement 和 PreapredStatement 均自动为一个事务。因此，当一个 Statement 调了 executeUpdate()方法后，所执行的 SQL 语句自动提交，其对数据库的修改不可再撤销。

但调用 Connection.setAutoCommit(boolean autoCommit)方法可以改变缺省设置：

```
connection.setAutoCommit(false);
```

将使本会话期内的语句不再自动提交，必须调用 Connection 的以下方法手动提交：

- commit()
- rollback()

前者为正常提交，后者表示事务回滚，撤销所有修改。

JDBC 支持以下隔离级别：

- static int TRANSACTION_NONE
- static int TRANSACTION_READ_UNCOMMITTED
- static int TRANSACTION_READ_COMMITTED
- static int TRANSACTION_REPEATABLE_READ

- static int TRANSACTION_SERIALIZABLE

它们对应的 int 值依次为 0,1,2,4 和 8。

2.13.7 把稀疏表格转为键值对存储

这一关要了解一些 resultset metadata 的相关操作。

……

```
            try {
                ResultSet rs = stmt.executeQuery(sql);
                ResultSetMetaData rsmd = rs.getMetaData();
                while(rs.next()) {
                    for(int i=2; i<=rsmd.getColumnCount(); i++) {   // 获得列数
                        if(rs.getInt(i) != 0){
                            n = insertSC(conn,
rs.getInt("sno"),rsmd.getColumnName(i), rs.getInt(i));     // 获得列号
                        }
                }
                }
            }catch (SQLException e){
                System.out.print("error");
            }
    }
}
```

2.13.8 制约因素分析与设计

从社会角度，转账操作关系到多方的利益关系，需要保证其准确、及时。
从安全角度，转账需要保证非常高的安全性，以防欺诈等存在。
从法律来说，转账也涉及多方的协同与竞争，包括不同银行之间，银行与个人之间，个人与个人之间等。

2.13.9 工程师责任及其分析

工程师应当平衡各个制约因素，去完善程序，考虑各种可能的突发情况，包括转账操作的正确性，网络的安全性，以及可能需要的及时性，人机交互的方式等，还需考虑法律法规的存在，才能写出能“用”的代码

3 课程总结

本次实验主要是基于 openGauss 完成了一系列实验，包括：数据库、表与完整性约束的定义、表结构和完整性约束的修改、数据查询、数据的插入、修改与删除、视图、存储过程与事务、用户自定义函数、触发器、安全性控制、备份+日志、并发控制与事务的隔离级别、数据库应用开发，简单来说除了设计其余都已完成。

具体而言，有如下工作：

1. 完成了基于 opengauss 进行相关定义包括数据库、表、视图、约束等和复杂增删查改的工作
2. 学会了使用存储过程、事务、函数、游标以及触发器完成相关操作。
3. 了解并学会控制对数据库内核相关的模块，包括安全性控制、恢复机制、完整性控制并发控制等机制。
4. 学会利用高级语言（java）操作数据库。

纵观整个数据库实验，个人觉得数据库实验还是有比较大的改进空间的，可能是因为刚刚开始用 educoder，不管是从内容还是形式都体验比较一般。一方面对于内容，个人觉得与课程的耦合性不够高，而且感觉一直都在了解语法，虽说有几关深入到了内核，但是太简单了，只是浅浅地了解了一番，但是对内在机理还是不够清楚。从形式来说，我觉得 educoder 这个平台大大限制了数据库这个实验，本来数据库就是很灵活的，现在变成非常死板地比对文本结果，只能说 educoder 的优势在这个实验上没怎么反应出来，反映出来的大多是劣势。简单来说，就是深入内核、难度加大、形式可变。最后，还是希望数据库实验以后越来越好，以及感谢老师们对我的帮助。