



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： CS2302

学 号： U202315663

姓 名： 刘世峰

指导教师： 袁平鹏

分数	
教师签名	

2025 年 06 月 25 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	

总分	
----	--

目录

1 课程任务概述.....	1
2 任务实施过程与分析.....	2
2.1 MySQL-数据库、表与完整性约束的定义(Create)	2
2.1.1 CHECK约束（第4关）	2
2.2 MySQL-表结构与完整性约束的修改	3
2.2.1 添加、删除与修改约束	3
2.3 MySQL-基于金融应用的数据查询(Select)	4
2.3.1 客户理财、保险与基金投资总额	4
2.3.2 购买基金的高峰期	5
2.3.3 以日历表格式显示每日基金购买总金额	6
2.4 MySQL-数据查询(Select)-新增	6
2.4.1 投资积极且偏好理财类产品的客户	6
2.4.2 查找相似的理财产品	7
2.4.3 查找相似的理财客户	9
2.5 数据查询(Select)-新增 2	10
2.5.1 统计各单位不计兼职的薪资总额、月平均薪资、最高薪资、最低薪资、 中位薪资	10
2.5.2 客户年度从各单位获得的酬劳总额	11
2.6 MySQL-数据的插入、修改与删除(Insert、Update、Delete)	12
2.6.1 插入多条完整的客户信息	12
2.6.2 插入不完整的客户信息	12
2.6.3 批量插入数据	12

2.6.4	删除没有银行卡的客户信息	12
2.6.5	冻结客户资产	12
2.6.6	连接更新	12
2.7	MySQL-视图.....	12
2.7.1	创建所有保险资产的详细记录视图	13
2.7.2	基于视图的查询	13
2.8	MySQL-存储过程与事务	13
2.8.1	使用游标的存储过程	13
2.8.2	使用事务的存储过程	14
2.9	MySQL-触发器.....	15
2.9.1	为投资表porperty 实现业务约束规则-根据投资类别分别引用不同表的主 码	15
2.10	MySQL-用户自定义函数.....	16
2.10.1	创建函数并在语句中使用它	16
2.11	MySQL-安全性控制.....	17
2.11.1	用户和权限	17
2.11.2	用户、角色与权限	17
2.12	MySQL-并发控制与事务的隔离级别	17
2.12.1	不可重复读	17
2.13	数据库设计与实现.....	17
2.13.1	从概念模型到 MySQL 实现	18
2.14	MySQL-数据库应用开发(JAVA 篇)	19
2.14.1	JDBC 体系结构和简单的查询	19
2.14.2	把稀疏表格转为键值对存储	19

3 课程总结.....	20
-------------	----

附录.....	21
---------	----

1 课程任务概述

“数据库系统原理实践”是“数据库系统原理”课程的配套实践环节，旨在通过实际操作帮助学生深入理解数据库系统的核心原理和技术。课程以 MySQL 8.0 为主要实践环境，结合 Java 1.8 开发工具，系统性地设计了涵盖数据库设计、管理、编程及应用的实训任务。

核心内容

数据库对象与约束管理

创建与修改数据库、表、索引、视图等对象。

实现主键、外键、CHECK、DEFAULT、UNIQUE 等完整性约束。

数据操作与查询

使用 SELECT、INSERT、UPDATE、DELETE 等语句完成数据操作。

实践复杂查询场景，如金融应用中的多表关联、聚合统计、相似性推荐等。

高级数据库功能 视图的创建与应用。

存储过程、事务、触发器、用户自定义函数的开发与调试。

并发控制（如事务隔离级别）与数据库恢复机制（备份与日志）。

数据库设计与实现

从概念模型到物理模型的转换（E-R 图与关系模式设计）。使用 MySQL Workbench 等工具完成数据库建模与脚本生成。

5.数据库应用开发

基于 JDBC 的 Java 程序开发，实现用户登录、数据增删改查、事务处理等功能。

6.系统内核扩展（选学）

存储管理（缓冲池、页面替换算法）。索引管理（B+树等结构）。

实践要求

必做任务：实训 1~6、14（数据库设计与实现）的全部或部分关卡，实训 15（Java 开发）最多跳过一关。

2 任务实施过程与分析

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了课程平台中的第 1~15 实训任务，下面将重点针对其中的具有代表性的重要任务阐述其完成过程中的具体工作。

2.1 MySQL-数据库、表与完整性约束的定义(Create)

本实训主要考察对数据库对象或约束条件的创建语法的掌握情况，包括创建数据库、表基表的主码约束、外码约束、CHECK 约束 DEFAULT 约束和 UNIQUE 约束。

已完成的关卡：创建数据库；创建表及表的主码约束；创建外码约束(foreign key)；CHECK 约束；DEFAULT 约束；UNIQUE 约束。

2.1.1 CHECK约束（第4关）

解题思路：在本实验中，首先在 MySQL 8.0.16 及以上版本中创建数据库 **MyDb**，并在其中建立表 **products**，设计字段 **pid** (CHAR (10)) 为主码、**name** (VARCHAR (32)) 为产品名称、**brand** (CHAR (10)) 与 **price** (INT) 分别用于记录品牌和价格；随后通过两个表级 CHECK 约束精确控制数据合法性：CK_products_brand 保证 brand 只能取 'A' 或 'B'，CK_products_price 要求 price 必须大于 0，从而同时实现用户定义完整性与业务规则的自动校验，具体实现见图2.1。

```
1  # 请在以下空白处填写完成任务的语句，空白行可通过回车换行添加。
2  # 你也可以在命令行窗口完成任务，不过，在命令行键入的语句不会被保存。
3
4  create DataBase MyDb;
5  use MyDb;
6  create table products(
7      pid char(10) primary key,
8      name varchar(32),
9      brand char(10),
10     price int,
11     constraint CK_products_brand check(brand in ('A','B')),
12     constraint CK_products_price check(price > 0)
13 );
14
15
16 # 结束
```

图2.1 CHECK约束的实现

2.2 MySQL-表结构与完整性约束的修改

本实训要求用 `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）。

已完成的关卡：修改表名；添加与删除字段；修改字段；添加、删除与修改约束。

2.2.1 添加、删除与修改约束

解题思路：在实现过程中先利用 `ALTER TABLE` 语句按“先被引用、后引用者”的顺序逐步补齐约束：

- ① 先给 **Staff** 表添加 `staffNo` 主键，使其具备被引用资格；
- ② 随后在 **Dept** 表上添加外键 `FK_Dept_mgrStaffNo` 使 `mgrStaffNo` 指向 `Staff.staffNo`；
- ③ 再为 **Staff** 表的 `dept` 列添加外键 `FK_Staff_dept` 指向 `Dept.deptNo`，保证双向关联完整性；
- ④ 接着通过 `ADD CONSTRAINT CK_Staff_gender CHECK (gender IN ('F','M'))` 为 **Staff** 强化性别取值合法性；
- ⑤ 最后在 **Dept** 表上增加唯一约束 `UN_Dept_deptName` 以防部门重名。整个流程严格依赖 `ALTER TABLE ... ADD CONSTRAINT ...` 或 `ADD PRIMARY KEY` 与 `ADD UNIQUE` 语法，不涉及数据迁移，确保引用列先具备唯一性（主键或唯一键）再建立外键，从而一次性通过所有完整性与唯一性校验。

实现代码如图2.2

```
1 use MyDb;
2 #请在以下空白处填写适当的语句，实现编程要求。
3 #(1) 为表Staff添加主码
4 alter table Staff add primary key(staffNo);
5 #(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:
6 alter table Dept add constraint FK_Dept_mgrStaffNo foreign key(mgrStaffNo) references Staff(staffNo);
7 #(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:
8 alter table Staff add constraint FK_Staff_dept foreign key (dept) references Dept(deptNo);
9 #(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:
10 alter table Staff add constraint CK_Staff_gender check(gender in ('F','M'));
11 #(5) 为表Dept添加unique约束：deptName不允许重复。约束名为UN_Dept_deptName:
12 alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

图2.2 约束相关操作的实现代码

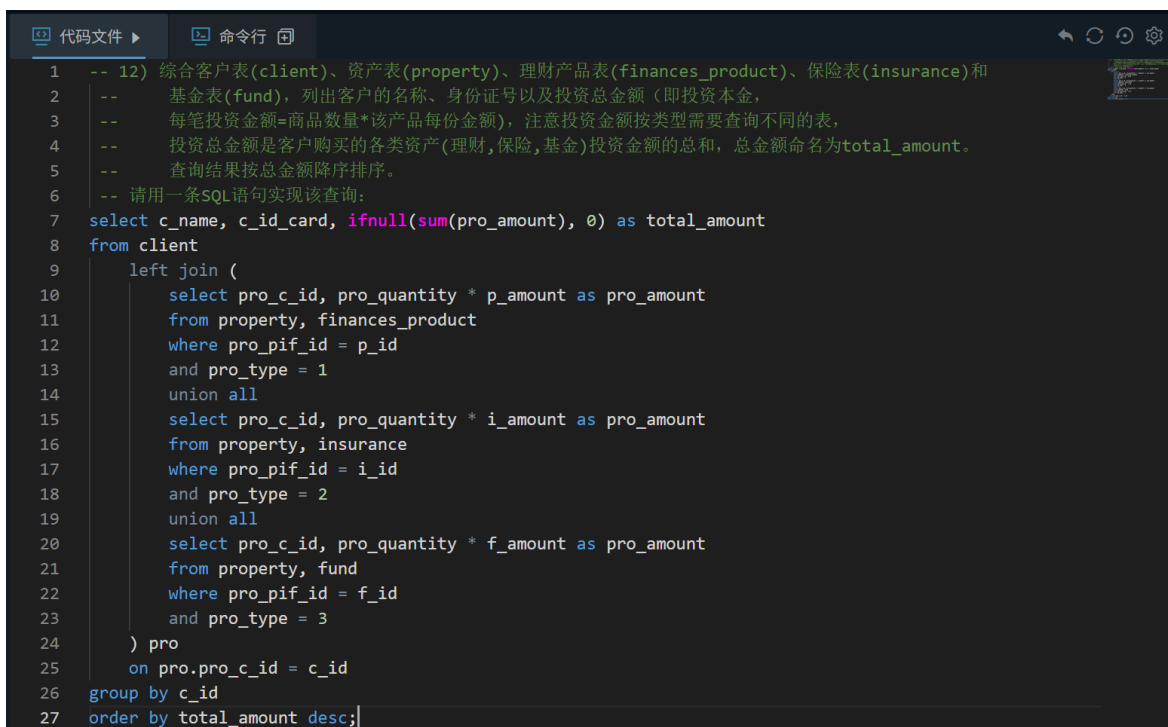
2.3 MySQL-基于金融应用的数据查询(Select)

本实训采用的是某银行的一个金融场景应用的模拟数据库，共 19 个关卡任务均围绕数据查询动作(Select)来实现。任务包括对数据库中数据的简单查询（例如直接使用select语句配合简单的条件选择语句实现的查询）、对查询结果排序、多层嵌套查询与多条件查询、多表连接、外连接、对结果进行升序或降序排序、LIKE、NOT EXISTS的用法、子查询、集函数与分组统计、top n查询、数据消重、rank函数的使用等等。

已完成关卡：1~19

2.3.1 客户理财、保险与基金投资总额

解题思路：先将 property 表按 pro_type 分别和 finances_product、insurance、fund 三张产品表连接，计算每条资产记录的投资本金 = pro_quantity × 每份金额，再用 UNION ALL 把三种类型的结果合并成临时表 (pro_c_id, pro_amount)；随后把该临时表与 client 做 LEFT JOIN，以保证无投资客户也列出，并用 SUM(pro_amount) 聚合得到各客户的投资总额 total_amount（对 NULL 用 IFNULL 转 0）；最后按 total_amount 降序排序，一条 SQL 即可输出客户姓名、身份证号和总投资金额，实现代码见图2.3。



```
1  -- 12) 综合客户表(client)、资产表(property)、理财产品表(finances_product)、保险表(insurance)和
2  --      基金表(fund)，列出客户的名称、身份证号以及投资总金额（即投资本金，
3  --      每笔投资金额=商品数量*该产品每份金额），注意投资金额按类型需要查询不同的表，
4  --      投资总金额是客户购买的各类资产（理财，保险，基金）投资金额的总和，总金额命名为total_amount。
5  --      查询结果按总金额降序排序。
6  --      请用一条SQL语句实现该查询：
7  select c_name, c_id_card, ifnull(sum(pro_amount), 0) as total_amount
8  from client
9  left join (
10     select pro_c_id, pro_quantity * p_amount as pro_amount
11     from property, finances_product
12     where pro_pif_id = p_id
13     and pro_type = 1
14     union all
15     select pro_c_id, pro_quantity * i_amount as pro_amount
16     from property, insurance
17     where pro_pif_id = i_id
18     and pro_type = 2
19     union all
20     select pro_c_id, pro_quantity * f_amount as pro_amount
21     from property, fund
22     where pro_pif_id = f_id
23     and pro_type = 3
24 ) pro
25 on pro.pro_c_id = c_id
26 group by c_id
27 order by total_amount desc;
```

图2.3 求客户理财、保险与基金投资总额实现

2.3.2 购买基金的高峰期

解题思路：先用递归 CTE calendar 生成 2022-02-07 至 02-28 的所有日期，再过滤掉周六周日得到完整“交易日序列”并给每个交易日编号 rn_td；随后将 property 与 fund 表关联，按交易日求出当日基金申购金额 total_amount（缺值置 0），得到 day_amount。接着只保留单日金额 ≥100 万的记录，重新编号为 rn_high，对比 rn_td 与 rn_high 的差值把相邻高额日归入同一组 (grp_id)；利用 HAVING count(*) ≥ 3 选出至少连续三日的高峰组号，再回查其日期与金额并按日期升序输出，实现高峰期识别与展示，如图2.4

```
4 with recursive calendar as (  
5     select date '2022-02-07' as d  
6     union all  
7     select date_add(d, interval 1 day)  
8     from calendar  
9     where d < '2022-02-28'  
10 ),  
11 trading_day as (  
12     select d,  
13         row_number() over (order by d) as rn_td  
14     from calendar  
15     where dayofweek(d) between 2 and 6  
16 ),  
17 day_amount as (  
18     select t.d,  
19         coalesce(sum(f.f_amount * p.pro_quantity), 0) as total_amount,  
20         t.rn_td  
21     from trading_day t  
22     left join property p  
23         on date(p.pro_purchase_time) = t.d  
24         and p.pro_type = 3  
25     left join fund f  
26         on f.f_id = p.pro_pif_id  
27     group by t.d, t.rn_td  
28 ),  
29 high_day as (  
30     select d,  
31         total_amount,  
32         rn_td,  
33         row_number() over (order by d) as rn_high  
34     from day_amount  
35     where total_amount >= 1000000  
36 ),  
37 tagged as (  
38     select d,  
39         total_amount,  
40         rn_td - rn_high as grp_id  
41     from high_day  
42 ),  
43 qualified_grp as (  
44     select grp_id  
45     from tagged  
46     group by grp_id  
47     having count(*) >= 3  
48 ),  
49 select d as pro_purchase_time,  
50     total_amount as total_amount  
51 from tagged  
52 where grp_id in (select grp_id from qualified_grp)  
53 order by d;
```

图2.4 求高峰期实现

2.3.3 以日历表格式显示每日基金购买总金额

解题思路：先按 property 与 fund 表关联，限定 2022-02 且 pro_type = 3，把单日基金申购本金汇总为 amount；随后借助 WEEK() 和 WEEKDAY() 把每条日期映射到“交易周次”(2 月 7 日所在周记为 1，因此整体作 WEEK(date)-5) 与周内序号 0-4；接着用内层查询产生结果集 (week, id, amount)，再在最外层利用条件聚合：
SUM(IF(id=0,amount,NULL)) AS Monday, ... SUM(IF(id=4,amount,NULL)) AS Friday
把列转行，生成一行五列的日历格式；最后按 week_of_trading 分组输出，即得 2 月每周工作日的基金购买总金额，如图2.5。

```
1  -- 19) 以日历表格式列出2022年2月每周每日基金购买总金额，输出格式如下：
2  -- week_of_trading Monday Tuesday Wednesday Thursday Friday
3  --           1
4  --           2
5  --           3
6  --           4
7  -- 请用一条SQL语句实现该查询：
8  select
9      `week` week_of_trading,
10     sum(if(`id` = 0, amount, null)) Monday,
11     sum(if(`id` = 1, amount, null)) Tuesday,
12     sum(if(`id` = 2, amount, null)) Wednesday,
13     sum(if(`id` = 3, amount, null)) Thursday,
14     sum(if(`id` = 4, amount, null)) Friday
15  from (
16      select week(pro_purchase_time) - 5 `week`,
17             weekday(pro_purchase_time) `id`,
18             sum(pro_quantity * f_amount) amount
19      from property join fund on pro_pif_id = f_id
20      where pro_purchase_time like "2022-02-%" and pro_type = 3
21      group by pro_purchase_time
22  ) as t
23  group by `week`;
```

图2.5 日历格式显示基金购买总额

2.4 MySQL-数据查询(Select)-新增

本小节子任务仍然以上述数据库内容为背景，但业务内容与统计、相似性推荐相关，其中的6个关卡分别实践在表中进行数据查询所需要的不同技巧与指令。

已完成关卡：查询销售总额前三的理财产品；投资积极且偏好理财类产品的客户；查询购买了所有畅销理财产品的客户；查找相似的理财产品；查询任意两个客户的相同理财产品数；查找相似的理财客户。

2.4.1 投资积极且偏好理财类产品的客户

解题思路：先分别统计每位客户在 property 表中持有的理财产品资产（pro_type = 1）与基金类资产（pro_type = 3）的不同产品编号个数：用两个 CTE table_cnt_1 与 table_cnt_3 求得 cnt_1 与 cnt_3。随后把这两张中间表按客户编号 (pro_c_id) 连接，并筛选 理财产品种类 ≥ 3 且大于基金产品种类 (cnt_1 > cnt_3) 的客户，即“投资积

极且偏好理财”的定义；最后 SELECT DISTINCT pro_c_id ORDER BY pro_c_id 输出满足条件的客户编号并去重、按升序排列，如图2.6。

```
1  -- 2) 投资积极且偏好理财类产品的客户
2  -- 请用一条SQL语句实现该查询：
3  with table_cnt_1 as
4  (
5      select pro_c_id, count(distinct(pro_pif_id)) as cnt_1
6      from property
7      where pro_type = 1
8      group by pro_c_id
9  ),
10 table_cnt_3 as
11 (
12     select pro_c_id, count(distinct(pro_pif_id)) as cnt_3
13     from property
14     where pro_type = 3
15     group by pro_c_id
16 )
17 select distinct property.pro_c_id
18 from property
19 inner join table_cnt_1 on property.pro_c_id = table_cnt_1.pro_c_id
20 inner join table_cnt_3 on property.pro_c_id = table_cnt_3.pro_c_id
21 where pro_type = 1 and cnt_1 > cnt_3
22 order by pro_c_id;
```

图2.6 投资积极且偏好理财类产品的客户

2.4.2 查找相似的理财产品

解题思路：先锁定**产品 14**的“头部客户”——统计每位客户持有 14 号理财产品的份额后用 DENSE_RANK() 按份额降序排名，保留名次 ≤ 3 的客户（并列算同一名次）。接着在 property 表中找出这批客户买过的**其它理财产品**(pro_type = 1 且 pro_pif_id \neq 14) 作为候选集合；对每个候选产品统计被这批客户持有的去重人数 cc，这就是与 14 号产品的“相似度”。再用第二次 DENSE_RANK() 按 cc 降序给出相似度排名 prank，保留排名 ≤ 3 （含并列）的产品。最后按相似度降序、产品编号升序输出 pro_pif_id、cc、prank，即得到产品 14 的“前 3 名（含并列）”相似理财产品，如图 2.7。

```

1  √ with prod14_cust as (
2  √      select pro_c_id,
3  √          |      sum(pro_quantity) qty
4  √      from property
5  √      where pro_type = 1
6  √          |      and pro_pif_id = 14
7  √      group by pro_c_id
8  √  ),
9  √ ranked as (
10 √      select pro_c_id,
11 √          |      dense_rank() over (order by qty desc) rk
12 √      from prod14_cust
13 √  ),
14 √ top_cust as (
15 √      select pro_c_id
16 √      from ranked
17 √      where rk <= 3
18 √  ),
19 √ cand as (
20 √      select distinct pro_pif_id
21 √      from property
22 √      where pro_type = 1
23 √          |      and pro_pif_id <> 14
24 √          |      and pro_c_id in (select pro_c_id from top_cust)
25 √  ),
26 √ sim as (
27 √      select c.pro_pif_id,
28 √          |      count(distinct p.pro_c_id) cc
29 √      from cand c
30 √      join property p
31 √          |      on p.pro_type = 1
32 √          |      and p.pro_pif_id = c.pro_pif_id
33 √      group by c.pro_pif_id
34 √  )
35 √ select pro_pif_id,
36 √      |      cc,
37 √      |      prank
38 √ from (
39 √      select pro_pif_id,
40 √          |      cc,
41 √          |      dense_rank() over (order by cc desc) prank
42 √      from sim
43 √  ) t
44 √ where prank <= 3
45 √ order by cc desc,
46 √      |      pro_pif_id;

```

图2.7 查找相似理财产品

2.4.3 查找相似的理财客户

解题思路：先把 property 表中过滤出理财资产（pro_type = 1），得到每位客户-理财产品对应关系 (cid, pid) 作为 CTE cp；随后将 cp 自连接（剔除自身）按产品编号配对，统计两位不同客户在同一理财产品上的交集计数 common，得到 CTE common_pairs。接着在该结果上使用 RANK() OVER (PARTITION BY pac ORDER BY common DESC, pbc ASC) 给每位“左边客户”pac 按相似度（共同理财产品数）降序、同分按客户编号升序分配名次 crank。最后筛选 crank < 3 以保留每位客户最相似的前两位理财同好，输出 pac、pbc、共同理财产品数 common 及相似度排名 crank，并按 pac 升序、其内部按 crank 排序，满足题意，如图2.8。

```
4 with cp as (  
5     select distinct pro_c_id as cid,  
6         pro_pif_id as pid  
7     from property  
8     where pro_type = 1  
9 ),  
10 common_pairs as (  
11     select a.cid as pac,  
12         b.cid as pbc,  
13         count(*) as common  
14     from cp a  
15     join cp b  
16         on a.pid = b.pid  
17         and a.cid <> b.cid  
18     group by a.cid, b.cid  
19 ),  
20 ranked as (  
21     select pac,  
22         pbc,  
23         common,  
24         rank() over (partition by pac order by common desc, pbc asc) as crank  
25     from common_pairs  
26 )  
27 select pac,  
28     pbc,  
29     common,  
30     crank  
31 from ranked  
32 where crank < 3  
33 order by pac,  
34     crank,  
35     pbc;
```

图2.8 查找相似理财客户

2.5 数据查询(Select)-新增2

本小节子任务在上述数据库内容的基础上，拓展相关内容和表格，包括综合客户表(client)、薪资表(wage)，列出客户的名称、年份、身份证号、全职酬劳总金额（full_t_amount）、兼职酬劳总金额（part_t_amount），查询业务主要涉及酬劳信息及其发放。

已完成的任务：客户年度从各单位获得的酬劳总额；统计各单位不计兼职的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资；获得兼职总酬劳前三名的客户；查找兼职酬劳的前三单位；批量发放酬劳；对身份证号码为 420108199702144323 的客户2023 年的酬劳代扣税。

2.5.1 统计各单位不计兼职的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资

解题思路：先用 full_w CTE 过滤出有效客户的正式工资（w_type = 1），同时引入年月字段 ym，确保后续可按人-月维度统计；接着在 org_base 中对每个单位汇总工资总额、人数、月份数以及极值，随后用 org_stat 计算单位月平均工资 $average_wage = total / (人数 \times 月份)$ 并保留两位小数。为了求中位数，先在 emp_month_avg 求出每名员工在本单位的月平均工资，再在 ranked 里对同一单位按工资升序编号，统计员工总数 cnt；median CTE 根据奇偶条目数选取恰当行（偶数取两行平均）得到 mid_wage。最后把 org_stat 与 median 合并输出 w_org, total_amount, average_wage, max_wage, min_wage, mid_wage，并按工资总额降序排序，实现了各单位薪资总额、均值、极值与中位值的一次性查询，如图2.9。

```
2 with full_w as (
3     select w.w_org,
4           w.w_c_id,
5           w.w_amount,
6           date_format(w.w_time, '%Y.%m') as ym
7     from wage w
8     join client c on c.c_id = w.w_c_id
9     where w.w_type = 1
10 ),
11 org_base as (
12     select w_org,
13           sum(w_amount) as total_amount,
14           count(distinct w_c_id) as people_cnt,
15           count(distinct ym) as month_cnt,
16           max(w_amount) as max_wage,
17           min(w_amount) as min_wage
18     from full_w
19     group by w_org
20 ),
21 org_stat as (
22     select w_org,
23           total_amount,
24           round(total_amount / (people_cnt * month_cnt), 2) as average_wage,
25           max_wage,
26           min_wage
27     from org_base
28 ),
29 emp_month_avg as (
30     select w_org,
31           w_c_id,
32           sum(w_amount) / count(distinct ym) as emp_avg
33     from full_w
34     group by w_org, w_c_id
35 ),
```

```

36 ranked as (
37     select w_org,
38           emp_avg,
39           row_number() over(partition by w_org order by emp_avg) as rn,
40           count(*) over(partition by w_org) as cnt
41     from emp_month_avg
42 ),
43 median as (
44     select w_org,
45           round(avg(emp_avg),2) as mid_wage
46     from ranked
47     where (cnt mod 2 = 1 and rn = cnt div 2 + 1)
48           or (cnt mod 2 = 0 and rn in (cnt div 2, cnt div 2 + 1))
49     group by w_org
50 )
51 select s.w_org,
52       s.total_amount,
53       s.average_wage,
54       s.max_wage,
55       s.min_wage,
56       m.mid_wage
57 from org_stat s
58 join median m on m.w_org = s.w_org
59 order by s.total_amount desc;

```

图2.9 统计各种薪资

2.5.2 客户年度从各单位获得的酬劳总额

解题思路：针对身份证号 420108199702144323 的客户，先在 client 表取到其 c_id，再统计其 2023 年在 wage 表的全年酬劳总额 total_salary；若 total_salary 不超过 60000 元则免税，否则按超额部分 × 20 % 计算全年应扣税额，并按“当月酬劳 ÷ 全年酬劳”的比例摊入当年的每条薪资记录，将 w_amount 减去相应税额，同时把 w_tax 标志改为 'Y'；整个过程通过一次 UPDATE ... JOIN ... 语句完成，对该客户 2023 年所有正式工资记录实现了按比例代扣个人所得税并设置扣税标志。

实现见图2.10

```

3  update wage w
4  join (
5      select c_id from client where c_id_card = '420108199702144323'
6  ) c on w.w_c_id = c.c_id
7  join (
8      select w_c_id, sum(w_amount) as total_salary
9      from wage
10     where w_c_id = (select c_id from client where c_id_card = '420108199702144323')
11           and year(w_time) = 2023
12     group by w_c_id
13 ) s on w.w_c_id = s.w_c_id
14 set
15     w.w_amount = w.w_amount - (
16         greatest(s.total_salary - 60000, 0) * 0.2
17         * (w.w_amount / s.total_salary)
18     ),
19     w.w_tax = if(s.total_salary > 60000, 'Y', 'N')
20 where year(w.w_time) = 2023;

```

图2.10 年度酬劳

2.6 MySQL-数据的插入、修改与删除(Insert、Update、Delete)

本小节子任务在上述数据库内容的基础上，通过 Insert、Update、Delete 完成对客户信息表的插入、修改与删除等相关操作。

已完成的关卡：插入多条完整的客户信息；插入不完整的客户信息；批量插入数据；删除没有银行卡的客户信息；冻结客户资产；连接更新。

2.6.1 插入多条完整的客户信息

该任务通过 INSERT 语句向 client 表一次性插入了三条客户数据，包含客户 ID、姓名、邮箱、身份证号、电话和密码等完整信息，具体实现代码省略。

2.6.2 插入不完整的客户信息

该任务通过 INSERT 语句向 client 表新增了 33 号客户蔡依婷的基本信息，包含必填字段并设置邮箱为 NULL 值，具体实现代码略。

2.6.3 批量插入数据

该任务通过 INSERT 语句将 new_client 表中的所有记录批量插入到 client 表中，实现新客户数据的完整迁移，具体实现代码省略。

2.6.4 删除没有银行卡的客户信息

该任务使用 DELETE 语句且通过 NOT EXISTS 子查询找出并删除 client 表中没有任何银行卡记录的客户信息，确保数据一致性，具体实现代码省略。

2.6.5 冻结客户资产

该任务使用 UPDATE 语句且通过子查询定位手机号为"13686431238"的客户 ID，并将其名下所有理财、保险和基金资产状态更新为"冻结"，具体实现代码省略。

2.6.6 连接更新

该任务使用 UPDATE 语句且通过关联 property 表和 client 表，将客户身份证号从 client 表同步更新到 property 表的 pro_id_card 字段，完善资产记录中的客户身份信息，具体实现代码省略。

2.7 MySQL-视图

使用 MySQL 语言实现视图 view 的创建和基于视图的查询。掌握 create view 及其相关语句的使用。

已完成的关卡：创建所有保险资产的详细记录视图；基于视图的查询。

2.7.1 创建所有保险资产的详细记录视图

该任务通过 SQL 语句创建了一个名为 v_insurance_detail 的视图，通过多表关联整合保险资产的完整信息。视图将 property 表（资产表）与 client 表（客户表）、insurance 表（保险表）进行连接，筛选出保险类型资产（pro_type=2），并提取包括客户姓名、身份证号、保险名称、保障项目等关键字段，具体实现代码省略。

2.7.2 基于视图的查询

该查询基于视图 v_insurance_detail，通过 SELECT 语句选取客户姓名和身份证号，使用 SUM 聚合函数分别计算每位客户的保险投资总额（保险金额×数量）和总收益。GROUP BY 子句按客户分组，ORDER BY 按投资总额降序排列结果，具体实现代码见图 2.9 所示。

2.8 MySQL-存储过程与事务

使用 MySQL 语言实现存储过程的创建和使用。掌握使用流程控制语句、游标和事务三种不同的存储过程。

已完成的关卡：使用流程控制语句的存储过程；使用游标的存储过程；使用事务的存储过程。

2.8.1 使用游标的存储过程

解题思路：先用两条游标把人员按类别分成 护士队列（e_type = 3）和 医生队列（e_type = 1-主任；e_type = 2-普通医生），各按工号升序取出，借助 CONTINUE HANDLER FOR NOT FOUND 在游标走到末尾时自动回环；随后从 start_date 到 end_date 按天遍历，先轮出两名护士填 n_nurse1/2_name，再确定当天医生：若是周末且轮到了主任（e_type = 1），则把主任“暂存”并改由下一位普通医生顶替；到下一个周一优先安排这名暂存主任；其余日期直接按游标顺序取医生。完成一日排班后向 night_shift_schedule 写入 (日期, 医生, 护士1, 护士2) 并把日期自增一天，循环直至结束日期，保证了①每天 1 医生 2 护士；②全员按工号循环；③主任不值周末且自动顺延到周一，如图2.11。

```

2 delimiter $$
3 create procedure sp_night_shift_arrange(in start_date date, in end_date date)
4 begin
5     declare finish, type, `week` int default false;
6     declare doctor, nurse1, nurse2, head char(30);
7     declare cursor1 cursor for select e_name from employee where e_type = 3;
8     declare cursor2 cursor for select e_type, e_name from employee where e_type < 3;
9     declare continue handler for not found set finish = true;
10    open cursor1;
11    open cursor2;
12    while start_date <= end_date do
13        fetch cursor1 into nurse1;
14        if finish then
15            close cursor1;
16            open cursor1;
17            set finish = false;
18            fetch cursor1 into nurse1;
19        end if;
20        fetch cursor1 into nurse2;
21        if finish then
22            close cursor1;
23            open cursor1;
24            set finish = false;
25            fetch cursor1 into nurse2;
26        end if;
27
28        set `week` = weekday(start_date);
29        if `week` = 0 and head is not null then
30            set doctor = head;
31            set head = null;
32        else
33            fetch cursor2 into type, doctor;
34            if finish then
35                close cursor2;
36                open cursor2;
37                set finish = false;
38                fetch cursor2 into type, doctor;
39            end if;
40            if `week` > 4 and type = 1 then
41                set head = doctor;
42                fetch cursor2 into type, doctor;
43                if finish then
44                    close cursor2;
45                    open cursor2;
46                    set finish = false;
47                    fetch cursor2 into type, doctor;
48                end if;
49            end if;
50        end if;
51        insert into night_shift_schedule values
52        (start_date, doctor, nurse1, nurse2);
53        set start_date = date_add(start_date, interval 1 day);
54    end while;
55 end$$
56
57
58 delimiter ;

```

图2.11 排班表

2.8.2 使用事务的存储过程

解题思路：该存储过程先在 bank_card 表中读取转出卡与转入卡的持有人编号、卡类型和余额，用来逐项校验业务规则：①申请人必须是转出卡持有人且收款人必须是转入卡持有人；②信用卡不能向储蓄卡转出，但储蓄卡可向信用卡“还款”；③若转出卡是储蓄卡则其可用余额必须≥转账金额。校验失败立即将 return_code 置 0 并退出。校验通过后启动显式事务，将储蓄卡余额按“减-加”方式更新；若目标是信

用卡则把金额作为负值累加，允许出现负余额（表示透支已被偿还）。两次 UPDATE 均成功后 COMMIT 并置 return_code 为 1；若任何一步出错触发异常则 ROLLBACK，同样把 return_code 设为 0，确保资金安全与数据一致性，具体实现代码省略。

2.9 MySQL-触发器

通过触发器的相关知识完成对触发器的相关操作。

已完成的关卡：全部的1关

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

解题思路：要强制 property 表的 pro_type 与 pro_pif_id 始终保持“一对一”来源关系，可在该表上编写 **BEFORE INSERT**（若还允许改号则再建 BEFORE UPDATE）触发器，在行写入前逐条校验：①首先判定 pro_type 只能取 1/2/3；若不在合法集合，立即用 SIGNAL SQLSTATE '45000' 抛出自定义异常并拼接“type x is illegal!” 错误信息；②随后根据 pro_type 分支，用 NOT EXISTS 子查询验证 pro_pif_id 是否存在于对应产品主表——finances_product(p_id)、insurance(i_id) 或 fund(f_id)；任一检查失败则同样抛出异常并返回“finances product #x not found!”、“insurance #x not found!”或“fund #x not found!”；③所有校验通过才允许插入。该触发器体仅含读操作、无事务或 DDL，故满足 MySQL 触发器限制；借助它可在数据库层面彻底阻断不合规数据写入，保障资产行和三张产品表之间的引用完整性及业务逻辑一致性，实现如图2.12。

```
1 use finance1;
2 drop trigger if exists before_property_inserted;
3 -- 请在适当的地方补充代码，完成任务要求;
4 delimiter $$
5 CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
6 FOR EACH ROW
7 BEGIN
8     IF (NEW.pro_type NOT IN (1,2,3)) THEN
9         SET @msg = CONCAT('type ', NEW.pro_type, ' is illegal!');
10        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
11    END IF;
12
13    IF (NEW.pro_type = 1) AND (SELECT COUNT(*) FROM finances_product WHERE p_id = NEW.pro_pif_id) = 0
14    THEN
15        SET @msg = CONCAT('finances product #', NEW.pro_pif_id, ' not found!');
16        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
17    END IF;
18
19    IF (NEW.pro_type = 2) AND (SELECT COUNT(*) FROM insurance WHERE i_id = NEW.pro_pif_id) = 0 THEN
20        SET @msg = CONCAT('insurance #', NEW.pro_pif_id, ' not found!');
21        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
22    END IF;
23
24    IF (NEW.pro_type = 3) AND (SELECT COUNT(*) FROM fund WHERE f_id = NEW.pro_pif_id) = 0 THEN
25        SET @msg = CONCAT('fund #', NEW.pro_pif_id, ' not found!');
26        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
27    END IF;
28
29 END$$
30
31 delimiter ;
```

图2.12 触发器

2.10 MySQL-用户自定义函数

本小节包括函数的定义，在SELECT语句中应用自定义函数等内容。

本实训已完成全部1个关卡。

2.10.1 创建函数并在语句中使用它

解题思路：先用 CREATE FUNCTION 定义标量函数 **get_deposit(client_id INT)**：在函数体里直接返回一条 SELECT SUM(b_balance)，对 bank_card 过滤 b_type='储蓄卡' 且 b_c_id = client_id，若客户无储蓄卡则 SUM 自动返 NULL（可按需用 IFNULL(...,0)）；设返回类型为 NUMERIC(10,2)。定义前后利用 DELIMITER 改变界符并适当 DROP FUNCTION 避免重名。随后在业务查询中把该函数当内部函数用：SELECT c_id_card, c_name, get_deposit(c_id) AS total_deposit FROM client WHERE get_deposit(c_id) >= 1000000 ORDER BY total_deposit DESC; 这样就按客户身份证、姓名列出所有储蓄余额≥100 万的客户，并按总额从高到低排序，实现“一条 SQL 调用自写函数完成过滤与统计”的要求，实现见图2.13。

```
1  use financel;
2
3  set global log_bin_trust_function_creators=1;
4  drop function IF EXISTS get_deposit;
5  /*
6   用create function语句创建符合以下要求的函数：
7   依据客户编号计算该客户所有储蓄卡的存款总额。
8   函数名为：get_Records。函数的参数名可以自己命名：*/
9  delimiter $$
10
11 CREATE FUNCTION GET_DEPOSIT(CLIENT_ID INT) RETURNS
12 NUMERIC(10, 2) BEGIN
13     return (
14         select sum(b_balance)
15         from bank_card
16         where b_type = "储蓄卡"
17         group by b_c_id
18         having b_c_id = client_id
19     );
20 END$$
21
22 delimiter ;
23
24 select
25     c_id_card,
26     c_name,
27     get_deposit(c_id) as total_deposit
28 from client
29 where
30     get_deposit(c_id) >= 1000000
31 order by total_deposit desc;
```

图2.13 用户自定义函数

2.11 MySQL-安全性控制

本小节包含了授予收回用户权限，角色创建收回等内容

本实训完成了全部2个关卡

2.11.1 用户和权限

解题思路：创建 tom、jerry 两用户；授 tom 对 client 表姓名/邮箱/电话列的 SELECT（含 WITH GRANT OPTION）；授 jerry 对 bank_card 表 b_balance 列的 UPDATE；用 REVOKE 收回 Cindy 对 bank_card 的 SELECT 权限，具体实现代码省略。

2.11.2 用户、角色与权限

解题思路：先用 CREATE ROLE 定义 client_manager 与 fund_manager；随后把操作权限直接授予角色：给 client_manager 分配对 client 表的 SELECT/INSERT/UPDATE 和对 bank_card 除 b_balance 外各列的 SELECT；给 fund_manager 分配对 fund 表的 SELECT/INSERT/UPDATE；最后用 GRANT <role> TO <user> 把 client_manager 授给 tom、jerry，把 fund_manager 授给 Cindy，实现统一授权、集中管理，具体实现代码省略。

2.12 MySQL-并发控制与事务的隔离级别

本小节包括并发控制中的并发控制与事务的隔离级别，读脏，不可重复读、幻读、主动加锁保证可重复读、可串行化等内容。

本任务已完成全部6个关卡。

2.12.1 不可重复读

解题思路：把两个会话都降到 READ COMMITTED（或更低的 READ UNCOMMITTED）隔离级别，使同一事务内的多次读不会被快照锁定；在 t2 里先 SELECT 余额得到值 A，然后 SLEEP；此间 t1 立即 UPDATE 同一行并 COMMIT；t2 醒来再次 SELECT 发现值变为 $B \neq A$ ，于是同一事务中出现两次读结果不一致的 **不可重复读**；随后各事务再按需要更新 / 提交并把每次查询结果写到辅助表 result 供评测比对，具体实现代码省略。

2.13 数据库设计与实现

本小节主要涉及了数据库设计与实现的相关任务，包括从概念模型到MySQL实现、从需求分析到逻辑模型以及建模工具的使用等。

本任务已完成3个关卡。

2.13.1 从概念模型到 MySQL 实现

解题思路：

先 CREATE DATABASE flight_booking CHARACTER SET=utf8mb4 并 USE flight_booking; 然后按“无外键→有外键”次序逐表实现。①基础主表：

- **airport:** airport_id INT AUTO_INCREMENT PRIMARY KEY, 唯一码 iata CHAR(3) UNIQUE、icao CHAR(4) UNIQUE, 其余列加 BTREE INDEX(name); 经纬度用 DECIMAL(11,8)。
- **airline:** airline_id INT AUTO_INCREMENT PK, iata CHAR(2) UNIQUE, name VARCHAR(30) NOT NULL。
- **user:** 包含 user_id INT AI PK 及登录字段, admin_tag TINYINT NOT NULL DEFAULT 0, 对 username、email 加 UNIQUE KEY; sex CHAR(1) 可加 CHECK (sex IN ('M','F'))。
- **passenger:** passenger_id INT AI PK, id CHAR(18) UNIQUE NOT NULL 等个人信息列, 同样对 sex 加校验。
- **airplane:** airplane_id INT AI PK, type VARCHAR(50) NOT NULL, capacity SMALLINT NOT NULL, identifier VARCHAR(50) NOT NULL 并唯一索引; 外键 airline_id 指向 airline。

②计划层表

- **flightschedule:** flight_no CHAR(8) PK; 周一至周日 7 列 TINYINT DEFAULT 0; departure TIME、arrival TIME、duration SMALLINT NOT NULL; from INT、to INT 两列作为外键分别指向 airport.airport_id (为避免同表两列同名, 用 from / to 而非 airport_id), 并在 (from)、(to) 建索引。airline_id 同样外键到 airline。

③实际航班表

- **flight:** flight_id INT AI PK, flight_no CHAR(8) 指向 flightschedule; departure DATETIME、arrival DATETIME、duration SMALLINT; airline_id、airplane_id、from、to 均设置外键, 且分别建索引。

④业务票据表

- **ticket:** ticket_id INT AI PK, flight_id INT NOT NULL 外键到 flight, user_id INT 外键到 user, passenger_id INT 外键到 passenger; seat CHAR(4), 为防重

复在 (flight_id, seat) 上建联合唯一； price DECIMAL(10,2) NOT NULL。

⑤所有索引显式指定 USING BTREE。对外键可加 ON UPDATE CASCADE ON DELETE RESTRICT 以保持参照完整性。完成后脚本整体顺序为：建库→各基础表→计划表→实际航班表→票表；每段用 ENGINE=InnoDB 保障事务与外键支持，从而把逻辑模型完整落地为 MySQL 物理模式，具体代码省略。

2.14 MySQL-数据库应用开发(JAVA 篇)

该实训任务通过 JDBC 技术实现金融场景下的数据库操作，包括客户信息查询、用户登录验证、数据增删改（如添加客户、销户、修改密码）、事务处理（如转账操作）以及稀疏表转键值对存储的优化设计。任务要求严格遵循输入输出格式，处理异常情况，并确保事务的原子性和数据一致性。

已完成的关卡：JDBC 体系结构和简单的查询；用户登录；添加新客户；银行卡销户；客户修改密码；事务与转账操作；把稀疏表格转为键值对存储。

2.14.1 JDBC 体系结构和简单的查询

解题思路：先加载 MySQL 驱动并用 DriverManager.getConnection 连到 finance 库；创建 Statement 执行

```
SELECT c_name, c_mail, c_phone FROM client WHERE c_mail IS NOT NULL;
```

获取结果集后先打印标题行（姓名·tab·邮箱·四个 tab·电话），接着循环 ResultSet，按“姓名 + tab + 邮箱 + 两个 tab + 电话”输出每条记录；最后在 finally 块里依次关闭 ResultSet、Statement、Connection 释放资源即可，具体代码省略。

2.14.2 把稀疏表格转为键值对存储

解题思路：用 JDBC 先查整行：SELECT * FROM entrance_exam；对每行取主键 sno 后按一个 列名数组 依次读取 9 门科目分数；用 ResultSet.getInt(col) 结合 wasNull() 判断该列是否为空——只有非空才调用封装好的 insertSC(conn, sno, col_name, score) 把 (sno, 列名, 列值) 写入稠密表 sc。这样逐行-逐列遍历即可把所有有效数据转存为键值对，空列自动丢弃，实现稀疏表到键值表的转换。

3 课程总结

本轮实践将数据库技术的全栈能力贯通到底。首先，在对象管理层面，独立完成了库、表、索引与视图的全生命周期设计和迭代，为关键字段配置主键、外键等完整性约束，确保数据一致性与可维护性。针对金融业务场景，熟练编写 SELECT、INSERT、UPDATE 等 DML 语句，实现多表聚合、收益排行等复杂分析任务。

高级功能方面，已掌握视图、存储过程、事务及触发器的深度用法：依托流程控制、游标及事务语义，编写的存储过程可自动生成斐波那契数列、安排夜班班次并完成加锁转账；触发器则在资产表写入时实时校验业务规则，杜绝脏数据。针对实际业务，我能够从需求出发绘制 E-R 模型，并自动输出 MySQL 脚本，已成功落地于影院管理系统等项目。

在应用层，基于 JDBC 开发 Java 组件，支持客户查询、登录验证、密码重置以及跨表转账；所有 SQL 均通过预编译语句执行，以防注入风险，并采用 try-catch-finally 严格释放资源、收口异常。依托头歌平台完成 15 个实训任务，重点攻克了商品收益众数、投资组合相似度等高阶查询，以及事务隔离控制。查询阶段综合运用子查询、窗口函数（ROW_NUMBER、DENSE_RANK）和自连接技术，为客户投资偏好分析与产品推荐提供支撑。

项目实践进一步印证：数据库设计必须兼顾业务逻辑与性能优化——例如将稀疏表改为键值对存储可显著加速检索；而事务控制是金融数据一致性的核心保障。下一步计划深入研究缓冲池机制与 B+ 树索引调优，完善分布式事务逻辑，以提升高并发场景下的稳定性；并补齐多语言字符集与细粒度权限管理，确保系统在法规与文化多元环境中的合规运行。

附录