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：	CS2306
学　　号：	U202315721
姓　　名：	黎欣雨
指导教师：	胡贯荣

分数	
教师签名	

2025 年 6 月 10 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

“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，基础内容涉及以下几个部分，并可结合实际对 DBMS 原理的掌握情况向内核设计延伸：

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 url 见相关课堂教师发布链接及其邀请码。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

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 1~8、11，12，14，15 实训任务，下面将重点针对其中的 3，4，5，14，15 任务阐述其完成过程中的具体工作。

2.1 基于金融应用的数据查询(Select)

本小节实训任务聚焦于金融应用场景下的 MySQL 数据查询操作，要求掌握多种 Select 查询技巧。内容包括基础客户信息筛选、复杂条件查询（如多产品持有、金额范围、空值处理）、数据统计（如众数、排名、总额计算）、高级查询（如第 N 高、组合匹配、时间段分析）以及特定格式输出（如日历表）。旨在通过实际金融业务场景，提升数据查询与分析能力。

本任务已完成全部 19 个关卡。

2.1.1 基金收益两种方式排名

对于名次不连续的收益排名，该查询通过三层嵌套结构实现不连续排名功能。最内层从 property 表中筛选基金类产品（pro_type=3），按客户分组计算总收益并降序排列。中间层使用三个用户变量实现核心排名逻辑：@rankcount 作为行号计数器逐行递增；@current_revenue 记录前一行收益值用于比较；@rank 动态维护当前排名。

```
SELECT pro_c_id,total_revenue,@rankcount := @rankcount + 1,
 IF(@current_revenue = total_revenue, @rank, @rank := @rankcount) AS trank
 @current_revenue := total_revenue
```

当检测到收益变化时，将当前行号赋值给排名变量（产生名次跳跃），否则保持原排名。这种设计使得相同收益的客户获得并列排名后，后续排名会跳过相应位次（如 1,2,2,4），通过变量状态传递和条件判断实现了精确的名次控制。

连续排名方案采用更简洁的两变量架构，去除了行号计数器@rankcount。其内层查询同样计算客户基金总收益并排序，但在排名逻辑上改用单调递增策略：仅当收益值变化时才增加排名计数器@rank（如 1,2,2,3）。这种实现通过 IF 函数直接比较当前收益与缓存值@current_revenue，避免了行号介入，使并列排名后的名次保持连续。相比不连续方案，连续排名减少了变量维护成本，但需要确保 ORDER BY 子句的排序稳定性。

```
SELECT pro_c_id,total_revenue,
 IF(@current_revenue = total_revenue, @rank, @rank := @rank + 1) AS trank,
    @current_revenue := total_revenue
```

最终代表不连续名次和连续名次排序的代表性结果截图如图所示。

1300	43100	7
700	42400	8
400	35200	9
1900	35200	9
200	33200	11
1000	29200	12
1700	24900	13
800	24200	14
1200	24200	14
600	17600	16

图 2.1 不连续名次

1300	43100	7
700	42400	8
400	35200	9
1900	35200	9
200	33200	10
1000	29200	11
1700	24900	12
800	24200	13

图 2.2 不连续名次

2.1.2 持有完全相同基金组合的客户

该查询通过自连接和嵌套子查询实现客户基金组合的精确匹配，核心逻辑分为三个部分：首先通过 A.pro_c_id < B.pro_c_id 确保每对客户只输出一次；

其次通过双重 NOT EXISTS 子查询验证双向基金持有的一致性——条件 1 检查 A 持有的所有基金（pro_type=3）必须存在于 B 的持有中（通过 NOT IN 反向筛选差异项）；

```
NOT EXISTS( SELECT *  FROM property C WHERE A.pro_c_id = C.pro_c_id AND C.pro_type = 3 AND C.pro_pif_id NOT IN (SELECT pro_pif_id FROM property D WHERE D.pro_c_id = B.pro_c_id AND D.pro_type = 3)  )
```

条件 2 对称地验证 B 的基金必须全部被 A 持有；

```
NOT EXISTS( SELECT *  FROM property E WHERE B.pro_c_id = E.pro_c_id AND E.pro_type = 3 AND E.pro_pif_id NOT IN ( SELECT pro_pif_id FROM property F WHERE F.pro_c_id = A.pro_c_id AND F.pro_type = 3 ) )
```

最后通过 EXISTS 确保客户 A 至少持有一只基金，避免与无基金持有的客户形成无效匹配。

```
EXISTS( SELECT *  FROM property G WHERE G.pro_c_id = A.pro_c_id AND G.pro_type = 3 );
```

2.1.3 以日历表格式显示每日基金购买总金额

该查询通过多阶段数据处理将基金购买记录转换为日历格式展示，核心逻辑分为三个层次：最内层子查询筛选 2022 年 2 月 7 日至 28 日的基金购买记录（pro_type=3），关联 fund 表计算每笔交易的总金额（f_amount*pro_quantity），

并通过 week()函数计算交易周数（减去 5 以适配 2 月起始周序）、weekday()函数获取星期几（0=周一至 4=周五）；中间层使用条件聚合函数 SUM(IF(days=N,amount,NULL))将每日金额按星期几横向展开，其中 IF 函数实现行转列的逻辑过滤；最终按周数分组排序输出日历表。

```
--   请用一条SQL语句实现该查询：
SELECT week_of_trading, SUM(IF(days=0,amount,NULL)) AS Monday, SUM(IF(days=1,amount,NULL))
AS Tuesday, SUM(IF(days=2,amount,NULL)) AS Wednesday, SUM(IF(days=3,amount,NULL)) AS
Thursday, SUM(IF(days=4,amount,NULL)) AS Friday
FROM
    (SELECT week(pro_purchase_time)-5 AS week_of_trading,    #返回周数（第一周次在当年是第六
周）
    SUM(f_amount*pro_quantity) AS amount,    #返回基金购买总金额
    weekday(pro_purchase_time) AS days   #返回星期几
    FROM fund, property
    WHERE f_id = pro_pif_id AND
    pro_type = 3 AND
    pro_purchase_time >= "2022-2-7" AND
    pro_purchase_time <= "2022-2-28"   #2月的时间范围
    GROUP BY pro_purchase_time) AS a
GROUP BY week_of_trading
ORDER BY week_of_trading ASC;
```

图 2.3 以日历表格式显示每日基金购买总金额

最终得到一个完整的日历表结构如下图所示：

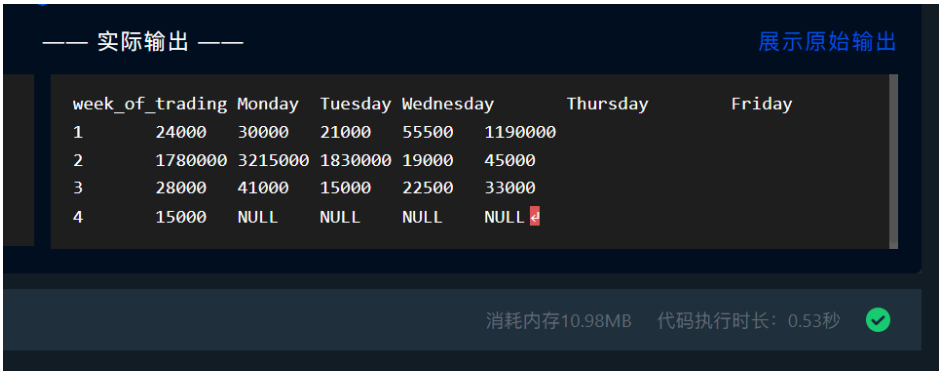

图 2.4 以日历表格式显示每日基金购买总金额数出结果

2.2 数据查询(Select)-新增

本小节实训任务围绕理财产品的销售与客户行为分析展开，重点训练高级数据查询技能。内容包括：销售额排名、客户投资偏好分析、畅销产品全覆盖查询、理财产品相似性匹配、客户间产品重叠统计以及相似客户识别。旨在通过理财业务场景，掌握复杂条件筛选、聚合计算、集合运算及关联分析等查询技术。

本实训任务已完成全部 6 个关卡。

2.2.1 查询购买了所有畅销理财产品的客户

该查询采用 NOT EXISTS 双重否定逻辑来找出购买了全部畅销理财产品的

客户，核心思路是将"购买所有畅销产品"转化为"不存在任何一个畅销产品未被购买"的条件判断。首先在子查询中筛选出被超过 2 个客户购买的理财产品（pro_type=1）定义为畅销产品，然后通过 NOT EXISTS 确保目标客户(p1.pro_c_id)不存在任何畅销产品(fin_pos.pro_pif_id)不在其购买记录中的情况。

```sql
-- 3) 查询购买了所有畅销理财产品的客户
--   请用一条SQL语句实现该查询：
SELECT DISTINCT pro_c_id
FROM property p1
WHERE NOT EXISTS(       #NOT EXISTS：不存在某个畅销产品是当前客户没有购买的则该客户会被选中
   SELECT *
   FROM
      (SELECT pro_pif_id
      FROM property
      WHERE pro_type = 1
      GROUP BY pro_pif_id
      HAVING COUNT(*)>2) fin_pos  #识别出畅销的理财产品作为子查询表
   WHERE fin_pos.pro_pif_id NOT IN   #子查询表示客户购买的所有理财产品，not in表示找出当前客户(p1.pro_c_id)没有购买的畅销产品
      (SELECT pro_pif_id
      FROM property p2
      WHERE p1.pro_c_id = p2.pro_c_id AND
      p2.pro_type = 1)
);
```

图 2.5 查询购买了所有畅销理财产品的客户

2.2.2 查找相似的理财产品

首先分析题目中给出的概念，得到几个有助于理解的公式：相似度 = 同时持有产品 14 和其他理财产品的客户数量，也等于产品 14 的核心客户群（持有 A 数量最多前三名）购买其他理财产品的总人数，相似的理财产品 = 相似度最高的 3 款理财产品。

首先在最内层子查询中，通过 dense_rank()窗口函数按持有量(pro_quantity)降序排列购买产品 14 的客户，筛选出持有量前三的核心客户群；

```
SELECT  pro_c_id
FROM  (SELECT pro_c_id, dense_rank() over(order by pro_quantity) AS rk
FROM  property
WHERE  pro_type = 1 AND pro_pif_id = 14) fin_rk
WHERE  fin_rk.rk <= 3))
```

中间层子查询获取这些核心客户购买的所有非 14 号理财产品；

```
SELECT DISTINCT pro_pif_id  FROM  property
   WHERE  pro_type = 1 AND  pro_pif_id <> 14 AND  pro_c_id in (SELECT pro_c_id  FROM fin_rk  WHERE fin_rk.rk <= 3)
```

最外层统计每款理财产品被核心客户购买的次数(COUNT(*))作为相似度指标，并使用 dense_rank()生成相似度排名(prank)。

最后得到的结果如图所示：

图 2.6 查找相似的理财产品

2.2.3 查询任意两个客户的相同理财产品数

该查询通过自连接和分组聚合实现了客户间理财产品相似度的计算，核心思路是通过 property 表的自连接(p1 和 p2)找出持有相同理财产品(pro_pif_id)的不同客户对(pro_c_id)。查询首先在 FROM 子句中进行自连接操作，连接条件设置为 p1.pro_pif_id = p2.pro_pif_id 且 p1.pro_c_id ≠ p2.pro_c_id，确保比较的是不同客户持有的相同理财产品；然后通过 GROUP BY pid_1,pid_2 对客户对进行分组，使用 COUNT()计算每对客户共同持有的理财产品数量；HAVING COUNT() >= 2 子句过滤掉共同持有产品少于 2 个的客户对；最后按第一个客户 ID 升序排序输出结果。该方案的复杂度主要来自于自连接操作，当客户数量和产品数量较大时会产生较大的中间结果集，但通过 WHERE 条件提前过滤非理财产品(pro_type=1)和相同客户的无效比较，有效优化了查询性能。技术亮点在于利用简单的自连接和分组操作实现了复杂的关系网络分析，结果直观展示了客户间的理财产品持有相似度。

```sql
-- 5) 查询任意两个客户的相同理财产品数
--    请用一条SQL语句实现该查询：
# 第一列和第二列输出客户编号(pro_c_id,pro_c_id)，第三列输出他们持有的相同理财产品数(total_count)
，按照第一列的客户编号的升序排列
SELECT pid_1 AS pro_c_id, pid_2 AS pro_c_id, COUNT(*) AS total_count
FROM       #通过自连接找出持有相同理财产品的不同客户对
    (SELECT p1.pro_c_id AS pid_1, p2.pro_c_id AS pid_2, p1.pro_pif_id
    FROM property p1, property p2
    WHERE p1.pro_type = 1 AND
        p2.pro_type = 1 AND
        p1.pro_c_id <> p2.pro_c_id AND   #排除客户与自身的比较
        p1.pro_pif_id = p2.pro_pif_id) same_fin  #确保比较的是同一产品
GROUP BY pid_1,pid_2
HAVING COUNT(*) >= 2
ORDER BY pid_1;
```

图 2.7 查询任意两个客户的相同理财产品数

2.2.4 查找相似的理财客户

该查询通过四层嵌套结构实现了基于共同持有理财产品的客户相似度分析，核心逻辑可分为四个处理阶段：

最内层查询通过property表的自连接(p1和p2)找出所有客户对及其共同持有的理财产品(pro_pif_id)，其中通过 p2.pro_pif_id IN 子查询确保只保留主客户(p1.pro_c_id)也持有的产品；

```
(SELECT DISTINCT p1.pro_c_id AS pac, p2.pro_c_id AS pbc, p2.pro_pif_id
FROM property p1, property p2
WHERE p1.pro_c_id <> p2.pro_c_id AND
    p2.pro_type = 1 AND
    p2.pro_pif_id IN
    (SELECT p3.pro_pif_id
    FROM property p3
    WHERE p3.pro_c_id = p1.pro_c_id AND   #配对客户持有的产品必须是主客户也持有的产

        p3.pro_type = 1)) common_c   # 找出所有客户对 及其 共同持有的理财产品
```

图 2.8 查找相似的理财客户

第二层通过 GROUP BY pac,pbc 计算每对客户共同持有的理财产品数量(common)；

```
SELECT pac, pbc, COUNT(*) AS common FROM (...) common_c GROUP BY pac,pbc
```

第三层使用窗口函数 rank()按共同产品数降序(隐含在 order by common)和客户 ID 升序(pbc)为每个主客户(pac)的相似客户分配排名(crank)；

```
SELECT pac, pbc, common, rank() over(partition by pac order by common,pbc) AS crank  FROM (...) common_b
```

最外层通过 WHERE crank <= 2 筛选出每个客户最多 2 个最相似的客户。

```
SELECT pac, pbc, common, crank  FROM (...) common_a  WHERE crank <= 2
```

该方案的复杂度主要来自多层嵌套和自连接操作，特别是最内层的 IN 子查询可能导致性能瓶颈，但通过提前过滤非理财产品(pro_type=1)和相同客户的无效比较优化了查询效率。技术亮点在于将复杂的相似度计算分解为多个逻辑清晰的步骤，并利用窗口函数实现了灵活的排名机制，最终结果精确展示了每位客户与其最相似的两位客户及其共同持有的理财产品数量。

2.3 数据查询（Select）-新增 2

本小节实训任务聚焦于客户酬劳数据的统计与操作，要求掌握薪资和兼职酬劳的查询与处理技能。包括：客户年度酬劳汇总、单位薪资统计、兼职酬劳排名、批量酬劳发放操作，以及特定客户的数据更新。旨在通过酬劳管理场景，强化聚合计算、条件筛选、批量操作等数据查询与更新能力。

本实训任务已完成 1~4，6 关卡。

2.3.1 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

该查询通过六个 CTE(Common Table Expressions)分阶段处理工资数据，最终输出各机构薪资统计报表。

首先在 filtered_data 中筛选全职员工的有效工资记录并关联客户表；

```
WITH filtered_data AS (SELECT w.w_org, w.w_c_id, w.w_amount, w.w_time
FROM    wage w,client c    WHERE w.w_c_id = c.c_id AND    w.w_type = 1),
```

monthly_salary 按月汇总每位员工的工资总额；

```
monthly_salary AS (SELECT w_org,w_c_id, DATE_FORMAT(w_time, '%Y-%m') AS month, SUM(w_amount) AS monthly_total FROM filtered_data GROUP BY w_org, w_c_id, month ),
```

employee_avg 计算员工平均月薪；

```
employee_avg AS (SELECT w_org, w_c_id, AVG(monthly_total) AS avg_monthly_salary FROM monthly_salary GROUP BY w_org, w_c_id),
```

org_summary 统计机构层面的薪资总额、员工数、月份数和极值；

```
org_summary AS ( SELECT ms.w_org,
    SUM(ms.monthly_total) AS total_amount,
    COUNT(DISTINCT ms.w_c_id) AS employee_count,
    COUNT(DISTINCT ms.month) AS month_count,
    MAX(ms.monthly_total) AS max_wage,
    MIN(ms.monthly_total) AS min_wage
    FROM monthly_salary ms
    GROUP BY    ms.w_org ),
```

ranked_employees 为员工薪资排序并编号；

```
ranked_employees AS (SELECT ea.w_org, ea.avg_monthly_salary, ROW_NUMBER() OVER (PARTITION BY ea.w_org ORDER BY ea.avg_monthly_salary) AS row_num,COUNT(*) OVER (PARTITION BY ea.w_org) AS total_employees FROM employee_avg ea),
```

median_calculation 通过计算中位数位置(考虑奇偶情况)确定薪资中位数。

```
median_calculation AS (SELECT re.w_org,AVG(re.avg_monthly_salary) AS mid_wage FROM ranked_employees re
WHERE re.row_num IN (
            (re.total_employees + 1) DIV 2,      #奇数则相等，偶数则为中间两个
```

```
            (re.total_employees + 2) DIV 2 )
GROUP BY re.w_org)
```

技术亮点包括：1)使用(DIV 2)运算高效处理中位数位置计算；2)ROW_NUMBER()窗口函数实现薪资排序；3)COALESCE 处理空值情况；4)多级聚合确保统计精度。该方案通过模块化 CTE 将复杂的统计分析分解为逻辑清晰的步骤，特别是中位数计算采用数学方法而非简单排序，大幅提升了大数集处理效率，最终输出包含总额、人均薪资、极值和中位数的完整机构薪资画像，按薪资总额降序排列。

最后输出结果展示为：

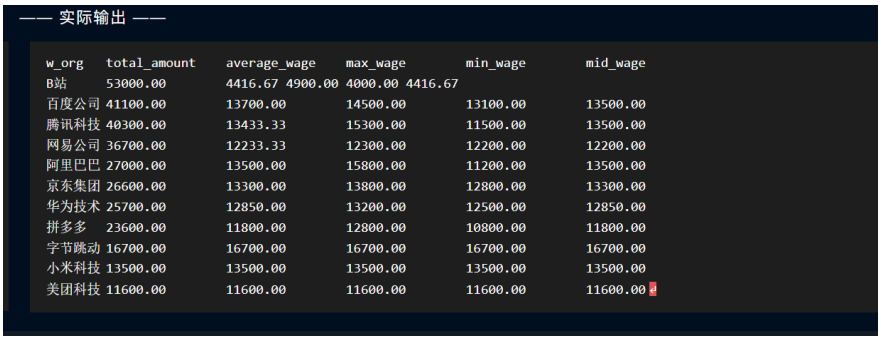

图 2.9 统计各单位不计兼职的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资数出结果

2.3.2 对身份证号为 420108199702144323 的客户 2023 年的酬劳代扣税

该 SQL 语句通过 JOIN 子查询和条件计算实现了客户工资记录的税务处理，主要分为三个逻辑阶段：首先在子查询中获取指定身份证号客户('420108199702144323')2023 年的总酬劳(SUM(w.w_amount))；

然后在主查询中通过 INNER JOIN 关联原工资表进行更新操作，其中税款计算采用累进税制逻辑——当总酬劳超过 6 万元时，对超出部分按 20%税率计算总税额，并按各月工资占比分摊到具体记录中(GREATEST 函数确保只对超阈值部分计税)；

最后通过 IF 条件设置税务标志位(w_tax)。

```sql
UPDATE wage
INNER JOIN (
    #获取指定身份证号客户的c_id及其2023年总酬劳
    SELECT w.w_c_id,SUM(w.w_amount) AS total_salary
    FROM wage w
    INNER JOIN client c ON w.w_c_id = c.c_id
    WHERE c.c_id_card = '420108199702144323' AND YEAR(w.w_time) = 2023
    GROUP BY w.w_c_id
) AS t ON wage.w_c_id = t.w_c_id
SET
    #计算并扣除税款（若总酬劳 > 60000）
    wage.w_amount = wage.w_amount - (
        GREATEST(t.total_salary - 60000, 0) * 0.2 * (wage.w_amount / t.total_salary)
    ),
    #设置扣税标志：总酬劳 > 60000 时标记为 'Y'
    wage.w_tax = IF(t.total_salary > 60000, 'Y', 'N')
WHERE
    YEAR(wage.w_time) = 2023  #仅更新2023年的记录
    AND wage.w_c_id = t.w_c_id;  #仅仅更新指定用户的酬劳
```

图 2.10 对身份证号为 420108199702144323 的客户 2023 年的酬劳代扣税

技术关键在于：1)使用 GREATEST 函数实现免税额阈值控制；2)通过比例分摊(wage.w_amount/t.total_salary)确保税款精确分配到各月；3)YEAR 函数限定时间范围保障数据准确性。该方案在单条语句中高效完成了税务计算、金额调整和状态标记的复合操作，但需要注意子查询的聚合结果与主表记录的精确匹配，以及浮点数计算可能带来的精度问题。

2.4 数据库设计与实现

本小节实训任务围绕数据库设计与实现流程展开，重点训练从业务需求到数据库落地的全流程能力。内容包括：概念模型到 MySQL 的转换实现、需求分析至逻辑模型的推导方法，以及专业建模工具的实际操作。旨在通过完整的数据库设计案例，掌握数据建模的核心步骤和技术工具的应用。

本实训任务已完成关卡 1。

2.4.1 从概念模型到 MySQL 实现

根据任务要求，按需对各个主体创建表，并按需指定各个主体的主码、外码和其他码的缺省值、不可空性、不可重性等。

将 ER 图转换为关系模式后，共有 8 个关系，分别是 user，passenger，airport，airline，airplane，flightschedule，flight，ticket。

```sql
CREATE DATABASE IF NOT EXISTS flight_booking;
USE flight_booking;

DROP TABLE IF EXISTS user;
CREATE TABLE user(
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    sex CHAR(1) NOT NULL,
    email VARCHAR(50),
    phone VARCHAR(30),
    username VARCHAR(20) NOT NULL UNIQUE,
    password CHAR(32) NOT NULL,
    admin_tag TINYINT NOT NULL DEFAULT 0
);

DROP TABLE IF EXISTS passenger;
CREATE TABLE passenger(
    passenger_id INT PRIMARY KEY AUTO_INCREMENT,
    id CHAR(18) NOT NULL UNIQUE,
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    mail VARCHAR(50),
    phone VARCHAR(20) NOT NULL,
    sex CHAR(1) NOT NULL,
    dob DATE
);
```

图 2.11 从概念模型到 MySQL 实现（1）

```sql
DROP TABLE IF EXISTS airport;
CREATE TABLE airport(
    airport_id INT PRIMARY KEY AUTO_INCREMENT,
    iata CHAR(3) NOT NULL UNIQUE,
    icao CHAR(4) NOT NULL UNIQUE,
    name VARCHAR(50) NOT NULL,
    city VARCHAR(50),
    country VARCHAR(50),
    latitude DECIMAL(11,8),
    longitude DECIMAL(11,8),
    index(name)
);

DROP TABLE IF EXISTS airline;
CREATE TABLE airline(
    airline_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    iata CHAR(2) NOT NULL UNIQUE,
    airport_id INT NOT NULL,
    CONSTRAINT FK_airline_airport FOREIGN KEY (airport_id) REFERENCES airport(airport_id)
);
```

图 2.12 从概念模型到 MySQL 实现（2）

```sql
DROP TABLE IF EXISTS airplane;
CREATE TABLE airplane(
    airplane_id INT PRIMARY KEY AUTO_INCREMENT,
    type VARCHAR(50) NOT NULL,
    capacity SMALLINT NOT NULL,
    identifier VARCHAR(50) NOT NULL,
    airline_id INT NOT NULL,
    CONSTRAINT FK_airplane_airline FOREIGN KEY (airline_id) REFERENCES airline(airline_id)
);
```

图 2.13 从概念模型到 MySQL 实现（3）

```sql
DROP TABLE IF EXISTS flightschedule;
CREATE TABLE flightschedule(
    flight_no CHAR(8) PRIMARY KEY,
    departure TIME NOT NULL,
    arrival TIME NOT NULL,
    duration SMALLINT NOT NULL,
    monday TINYINT DEFAULT 0,
    tuesday TINYINT DEFAULT 0,
    wednesday TINYINT DEFAULT 0,
    thursday TINYINT DEFAULT 0,
    friday TINYINT DEFAULT 0,
    saturday TINYINT DEFAULT 0,
    sunday TINYINT DEFAULT 0,
    airline_id INT NOT NULL,
    `from` INT NOT NULL,
    `to` INT NOT NULL,
    CONSTRAINT FK_flightschedule_airline FOREIGN KEY (airline_id) REFERENCES airline
(airline_id),
    CONSTRAINT FK_flightschedule_airport1 FOREIGN KEY (`from`) REFERENCES airport
(airport_id),
    CONSTRAINT FK_flightschedule_airport2 FOREIGN KEY (`to`) REFERENCES airport(airport_id)
);
```

图 2.14 从概念模型到 MySQL 实现（4）

```sql
DROP TABLE IF EXISTS flight;
CREATE TABLE flight(
    flight_id INT PRIMARY KEY AUTO_INCREMENT,
    departure DATETIME NOT NULL,
    arrival DATETIME NOT NULL,
    duration SMALLINT NOT NULL,
    airline_id INT NOT NULL,
    airplane_id INT NOT NULL,
    flight_no CHAR(8) NOT NULL,
    `from` INT NOT NULL,
    `to` INT NOT NULL,
    CONSTRAINT FK_flight_airline FOREIGN KEY (airline_id) REFERENCES airline(airline_id),
    CONSTRAINT FK_flight_airplane FOREIGN KEY (airplane_id) REFERENCES airplane(airplane_id)
,
    CONSTRAINT FK_flight_flightschedule FOREIGN KEY (flight_no) REFERENCES flightschedule
(flight_no),
    CONSTRAINT FK_flight_airport1 FOREIGN KEY (`from`) REFERENCES airport(airport_id),
    CONSTRAINT FK_flight_airport2 FOREIGN KEY (`to`) REFERENCES airport(airport_id)
);
```

图 2.15 从概念模型到 MySQL 实现（5）

```sql
DROP TABLE IF EXISTS ticket;
CREATE TABLE ticket(
    ticket_id INT PRIMARY KEY AUTO_INCREMENT,
    seat CHAR(4),
    price DECIMAL(10,2) NOT NULL,
    flight_id INT NOT NULL,
    passenger_id INT NOT NULL,
    user_id INT NOT NULL,
    CONSTRAINT FK_ticket_flight FOREIGN KEY (flight_id) REFERENCES flight(flight_id),
    CONSTRAINT FK_ticket_passenger FOREIGN KEY (passenger_id) REFERENCES passenger
(passenger_id),
    CONSTRAINT FK_ticket_user FOREIGN KEY (user_id) REFERENCES user(user_id)
);
```

图 2.16 从概念模型到 MySQL 实现（6）

2.4.2 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”就可以根据权限分成两类，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。

2.4.3 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。

2.5 数据库应用开发(JAVA 篇)

本小节实训任务聚焦于使用 Java 进行 MySQL 数据库应用开发，涵盖 JDBC 基础操作和典型金融业务功能实现。内容包括：JDBC 体系结构理解、用户登录验证、客户信息增删改查、银行卡销户处理、密码修改、转账事务控制，以及数据存储格式转换。旨在通过实际金融业务场景，掌握 Java 与 MySQL 交互的核心开发技能和事务处理能力。

本实训任务已完成全部 7 个关卡。

2.5.1 JDBC 体系结构和简单的查询

本关任务是查询 client 表中邮箱非空的客户信息，列出客户姓名，邮箱和电话。 应使用 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，用 ResultSet executeQuery(String SQL)返回一个 ResultSet 对象实现，使用 resultSet.next()遍历 ResultSet 输出相应的信息即可。

```
        try { Class.forName("com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/finance?useUnicode=true&
characterEncoding=UTF8&useSSL=false&serverTimezone=UTC","root","123123");
            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.println(resultSet.getString("c_name")+"\t"+resultSet.getString("c_mail")+"\t\t"+resultSet.
getString("c_phone"));
            }



        } catch (ClassNotFoundException e) {
```

图 2.17 JDBC 体系结构和简单的查询

2.5.2 用户登录

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：

```
        // 补充实现代码:
        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！");
    } catch (ClassNotFoundException e) {
```

图 2.18 用户登录

2.5.3 添加新用户

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;
}
```

图 2.19 添加新用户

2.5.4 银行卡销户

实现向银行卡销号的方法，只要客户编号和银行卡号匹配，即从 bank_card 表中删除该银行卡。使用预编译 SQL 语句防止 SQL 注入,通过 PreparedStatement 设置两个参数：客户 ID 和银行卡号,执行删除操作并返回影响的行数(成功删除返回 1，失败返回 0)。

```java
public static int removeBankCard(Connection connection,
                                 int b_c_id, String b_number){
    String sql = "delete from bank_card where b_c_id = ? and b_number = ?";
    try {
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setInt(1, b_c_id);
        ps.setString(2, b_number);
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
```

图 2.20 银行卡销户

2.5.5 客户修改密码

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：

```java
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;
}
```

图 2.21 客户修改密码

2.5.6 事务与转账操作

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：

```java
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;
}
```

图 2.22 事务与转账操作

2.5.7 把稀疏表格转为键值对存储

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:

```java
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();
    }
}
```

图 2.23 把稀疏表格转为键值对存储（1）

```
    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();
        }

    }
}
```

图 2.24 把稀疏表格转为键值对存储（2）

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、数据库应用开发、数据库的备份与日志、数据库的设计与实现等 15 个实训实验。

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 1~8、11，12，14，15 实训任务。从中我学会了如何创建数据库，表，完整性约束；如何进行数据查询；如何对数据进行插入，修改，删除；如何在表的基础上建立视图；怎么进行存储过程和事务的编写；还有数据库的设计与实现任务中教会我们怎么绘制 E_R 图以及最后的数据库的应用开发。

在课程实验的实施过程中，我们通过合理使用搜索引擎查阅相关资料，学习了一些复杂语法和一些经典问题的简单处理方法简化编程，让我能够实现一些复杂的查询。很多关卡中实现的方法都不是固定的，有时候熟知内置函数会带来意想不到的简化。编程的重点主要在于逻辑的设计，首先要在脑海中有一个完整的查询框架，不能走一步想一步。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。