

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | C |
| 学 号： | U |
| 姓 名： |  |
| 指导教师： | 胡贯荣 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2025年 6月 26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](#_Toc202374700)

[2 任务实施过程与分析 2](#_Toc202374701)

[2.1 基于金融应用的数据查询(Select) 3](#_Toc202374702)

[2.1.1 客户理财、保险与基金投资总额 3](#_Toc202374703)

[2.1.2 持有完全相同基金组合的客户 4](#_Toc202374704)

[2.1.3购买基金的高峰期 6](#_Toc202374705)

[2.2 数据查询（Select）-新增2 8](#_Toc202374706)

[2.2.1 统计各单位的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资 8](#_Toc202374707)

[2.2.2 对指定身份证号的客户2023年的酬劳代扣税 9](#_Toc202374708)

[2.3 存储过程与事务 10](#_Toc202374709)

[2.3.1使用事务的存储过程 10](#_Toc202374710)

[2.4 数据库设计与实现 12](#_Toc202374711)

[2.4.1 从概念模型到MySQL实现 12](#_Toc202374712)

[2.4.2 从需求分析到逻辑模型 13](#_Toc202374713)

[2.4.3 建模工具的使用 15](#_Toc202374714)

[2.4.4 制约因素分析与设计 16](#_Toc202374715)

[2.4.5 工程师责任及其分析 17](#_Toc202374716)

[2.5 数据库应用开发(JAVA篇) 17](#_Toc202374717)

[2.5.1 客户修改密码 17](#_Toc202374718)

[2.5.2 事务与转账操作 19](#_Toc202374719)

[2.5.3 把稀疏表格转为键值对存储 20](#_Toc202374720)

[3 课程总结 23](#_Toc202374721)

[附录 25](#_Toc202374722)

# 1 课程任务概述

本实践课程通过一系列基于MySQL的实训任务，将理论知识应用于实践。课程内容涵盖了从基础的数据库对象管理到核心的系统机制，再到上层的应用开发与数据库设计。课程共分为16个实训模块，具体分解如下：

1. **基础数据定义 (实训1-2):**
2. 实训1 (Create): 学习使用CREATE语句创建数据库、表以及主键、外键、CHECK等完整性约束。
3. 实训2 (Alter): 学习使用ALTER语句修改已有的表结构和约束。
4. **数据查询 (实训3-5):**
5. 实训3 (Select基础): 在一个金融应用数据库背景下，进行多样的单表、多表数据查询。
6. 实训4 & 5 (Select进阶): 在金融背景上进行扩展，完成与统计、推荐、薪酬计算等相关的复杂查询任务。
7. **数据操作与高级对象 (实训6-10):**
8. 实训6 (DML): 掌握数据的插入、修改和删除操作。
9. 实训7 (视图): 学习创建和使用视图。
10. 实训8 (存储过程与事务): 学习编写存储过程并进行事务管理。
11. 实训9 (触发器): 学习创建和使用触发器。
12. 实训10 (自定义函数): 学习创建和使用用户自定义函数。
13. **数据库系统核心机制 (实训11-13 & 16):**
14. 实训11 (安全性): 实践数据库的安全性控制。
15. 实训12 (并发控制): 理解事务隔离级别，处理并发事务冲突。
16. 实训13 (备份与恢复): 实践数据库的逻辑备份与恢复。
17. 实训16 (存储管理): 深入内核，实现缓冲池管理器等底层存储组件。
18. **数据库设计与应用开发 (实训14-15):**
19. 实训14 (设计与实现): 包含从E-R图到MySQL实现，以及从需求分析到逻辑模型设计的全过程。
20. 实训15 (Java应用开发): 使用JDBC连接和操作数据库，完成查询、登录、增删改数据、事务处理（转账）等实际应用功能。

# 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。如图2.1所示，本次实践最终完成了课程平台中的第1~15实训任务，仅跳过实训8的第2关，总计126分。下面将重点针对其中的基于金融应用的数据查询（实训3）、数据查询（Select）-新增2（实训5）、存储过程与事务（实训8）、数据库设计与实现（实训14）、Java数据库应用开发（实训15）等任务阐述其完成过程中的具体工作。



图2.1 实验完成情况总览

## 2.1 基于金融应用的数据查询(Select)

本关的核心任务是在一个模拟的银行金融应用数据库上，练习并掌握SQL中SELECT查询语句的使用。该任务要求根据提供的六个表（客户表 client、银行卡表 bank\_card、理财产品表 finances\_product、保险表 insurance、基金表 fund 和资产表 property），编写相应的查询语句来完成各个关卡指定的查询需求。

本关共19个子任务，已完成19个子任务。以下选出一些具有代表性的、有较强的综合性子任务来进行阐释说明。

### 2.1.1 客户理财、保险与基金投资总额

本关要求综合查询客户表、资产表以及三种产品表，统计出每位客户的姓名、身份证号以及其投资的总本金（total\_amount）。查询结果需要包含所有客户（即使没有投资记录），并按总投资金额降序排列。

本关关键在于，客户的资产记录分散在property表中，而计算单笔投资本金所需的“产品单价”又分别存储在三个不同的产品表中，因此需要根据property.pro\_type字段来决定关联哪张表。本关代码见附录中的代码 3，流程示意图见图2.2。

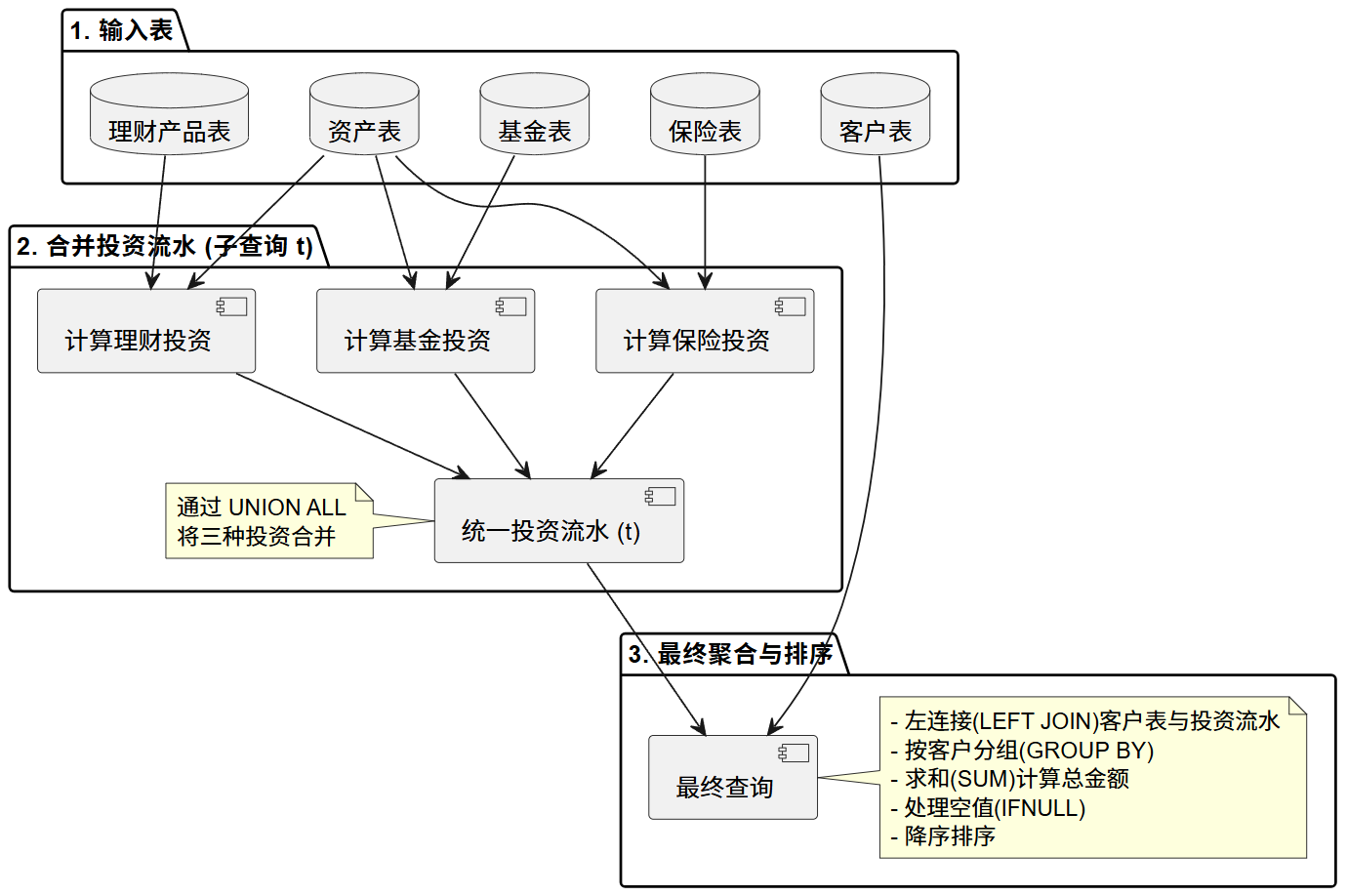


图2.2 客户理财、保险与基金投资总额查询流程

首先，我们通过一个子查询来创建一个临时的、标准化的“投资流水表”，别名设为t。这个表只有两列：客户ID (pro\_c\_id) 和单笔投资金额 (amount)。为了汇集所有类型的投资，子查询内部使用了 UNION ALL 将三个独立的查询结果合并。每个独立查询负责一种资产类型。使用 UNION ALL 而非 UNION 是因为我们明确知道三部分查询结果不会有交集，所以无需进行额外的去重操作，优化了性能。

主查询从 client 表开始，使用 LEFT JOIN 与上述的投资流水子查询 t 进行关联。选用 LEFT JOIN 是为了确保：即使某个客户在子查询 t 中没有任何投资记录，该客户的信息依然会出现在最终结果中。通过 GROUP BY c.c\_id 对结果按客户进行分组，然后使用 SUM(t.amount) 将每个客户名下的所有投资流水金额求和，从而得到投资总额。

需要注意的是，对于没有投资记录的客户，SUM(t.amount) 的结果会是 NULL。使用 IFNULL(SUM(t.amount), 0) 可以将这些 NULL 值优雅地转换成0，使之符合业务逻辑。

最后，使用 ORDER BY total\_amount DESC 对结果集按总投资金额进行降序排序，完成任务要求。

### 2.1.2 持有完全相同基金组合的客户

本关要求在资产表(property)中找出所有持有完全相同基金组合的客户对。客户对(A, B)有效的条件是：当客户A持有的所有基金，客户B也全部持有，并且客户B持有的所有基金，客户A也全部持有。为避免结果重复，如(A, B)和(B, A)，查询结果只应包含客户编号较小者在前的一对。

本关是一个比较典型的“集合相等”判断问题。判断两个集合A和B相等，等价于判断“A是B的子集”且“B是A的子集”。见代码 1，本关中我使用了双重相关子查询NOT EXISTS来实现这一逻辑。

|  |
| --- |
| SELECT DISTINCT a.pro\_c\_id AS c\_id1, b.pro\_c\_id AS c\_id2  FROM property a, property b  WHERE a.pro\_type = 3 AND b.pro\_type = 3 AND a.pro\_c\_id < b.pro\_c\_id  -- a有的基金b都有  AND NOT EXISTS (  SELECT 1 FROM property pa  WHERE pa.pro\_c\_id = a.pro\_c\_id AND pa.pro\_type = 3  AND NOT EXISTS (  SELECT 1 FROM property pb  WHERE pb.pro\_c\_id = b.pro\_c\_id AND pb.pro\_type = 3  AND pb.pro\_pif\_id = pa.pro\_pif\_id  )  )  -- b有的基金a都有  AND NOT EXISTS (  SELECT 1 FROM property pb  WHERE pb.pro\_c\_id = b.pro\_c\_id AND pb.pro\_type = 3  AND NOT EXISTS (  SELECT 1 FROM property pa  WHERE pa.pro\_c\_id = a.pro\_c\_id AND pa.pro\_type = 3  AND pa.pro\_pif\_id = pb.pro\_pif\_id  )  ); |

代码 1 持有完全相同基金组合的客户SQL代码

首先，使用双重NOT EXISTS来进行双向子集检查。以“客户a的基金集是客户b的子集”为例，第一个 NOT EXISTS 结构可以解读为：“不存在这样一种基金，它被客户a持有，但客户b却没有持有”。内层 NOT EXISTS 负责验证这个反例是否存在：对于a持有的每一只基金(pa.pro\_pif\_id)，检查b是否也持有(pb.pro\_pif\_id = pa.pro\_pif\_id)。如果b不持有，内层NOT EXISTS为真，导致外层NOT EXISTS为假，该客户对被排除。只有当上述两个双重NOT EXISTS条件同时满足时，才能证明两个客户的基金集合完全相等。

最后，通过 FROM property a, property b 对 property 表进行自连接，生成所有可能的客户对。WHERE子句通过 a.pro\_type = 3 AND b.pro\_type = 3 确保只关注基金投资。其中，关键条件 a.pro\_c\_id < b.pro\_c\_id 一举两得：既排除了客户与自身的比较（如 A,A），也避免了结果中出现重复的客户对（如 (1,2) 和 (2,1)），确保了结果的唯一性。

### 2.1.3购买基金的高峰期

本关要求在2022年2月的交易日中，识别出“基金购买高峰期”。高峰期被定义为：至少连续三个交易日，每个交易日的基金总购买金额都达到或超过100万元。最终需列出所有属于高峰期内的日期及当日的总购买金额。

本关是一个时间序列分析问题，核心在于如何判断“连续三个交易日”且“每日金额达标”。在本关中我通过多层嵌套子查询和窗口函数来解决，见代码 2。

|  |
| --- |
| SELECT t2.pro\_purchase\_time, total\_amount  FROM(  SELECT t1.pro\_purchase\_time, total\_amount,  IF(incL1=1 AND incL2=2, MIN(t1.total\_amount) OVER (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW), 0) AS min1,  IF(incL1=1 AND incR1=4, MIN(t1.total\_amount) OVER (ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING), 0) AS min2,  IF(incR1=4 AND incR2=3, MIN(t1.total\_amount) OVER (ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING), 0) AS min3  FROM(  SELECT pro\_purchase\_time,  SUM(pro\_quantity\*f\_amount) AS total\_amount,  (weekday(pro\_purchase\_time)-weekday(LAG(pro\_purchase\_time, 1) OVER (ORDER BY pro\_purchase\_time)) +5)%5 AS incL1,  (weekday(pro\_purchase\_time)-weekday(LAG(pro\_purchase\_time, 2) OVER (ORDER BY pro\_purchase\_time)) +5)%5 AS incL2,  (weekday(pro\_purchase\_time)-weekday(LEAD(pro\_purchase\_time, 1)OVER (ORDER BY pro\_purchase\_time)) +5)%5 AS incR1,  (weekday(pro\_purchase\_time)-weekday(LEAD(pro\_purchase\_time, 2)OVER (ORDER BY pro\_purchase\_time)) +5)%5 AS incR2  FROM property, fund  WHERE pro\_type=3 AND pro\_pif\_id=f\_id AND pro\_purchase\_time BETWEEN '2022-02-01' AND '2022-02-28'  GROUP BY pro\_purchase\_time  ORDER BY pro\_purchase\_time ASC  ) t1  ) t2  WHERE t2.min1>=1000000 OR t2.min2>=1000000 OR t2.min3>=1000000; |

代码 2 购买基金的高峰期SQL代码

首先，内层子查询 t1进行数据准备与日期间隔计算，计算出2022年2月每一天（有交易记录）的基金总购买额，并计算出当前交易日与前后交易日之间的“工作日”间隔。第一步通过聚合计算出每日的基金总购买额，然后使用窗口函数LAG和LEAD获取当前行的前一个、前两个、后一个、后两个交易日的日期。通过weekday()函数和模运算 (weekday(date1) - weekday(date2) + 5) % 5 来计算两个日期之间的工作日差值，其中，incL1, incL2 分别计算与前1、前2个交易日的间隔；incR1, incR2 计算与后1、后2个交易日的间隔。

中层子查询 t2用于识别连续三日窗口并计算窗口内最低金额。基于t1计算出的日期间隔，找出所有满足“连续三个交易日”的窗口，并计算该窗口内三天的最低日交易额。代码考虑了三种构成连续三日窗口的情况，并为每种情况计算一个最低值：

1. min1：向后看。IF(incL1=1 AND incL2=2, ...) 判断当前日期、前一交易日、前二交易日是否连续。如果是，则计算这三天的最低交易额。
2. min2：看中间。IF(incL1=1 AND incR1=4, ...) 判断当前日期、前一交易日、后一交易日是否连续。（注意：incR1用+5和%5计算后，后一个工作日的差值会是4）。如果是，则计算这三天的最低交易额。
3. min3：向前看。IF(incR1=4 AND incR2=3, ...) 判断当前日期、后一交易日、后二交易日是否连续。如果是，则计算这三天的最低交易额。

其中，使用了MIN(...) OVER (...)窗口函数，可以高效地在滑动窗口内计算最小值，避免了复杂的自连接。

最后，从t2的结果中筛选出属于高峰期的日期。

## 2.2 数据查询（Select）-新增2

本关的核心任务是对客户表(client)和薪资表(wage)进行复杂的薪酬数据统计与分析。要求熟练运用子查询、聚合函数、条件逻辑(IF)、窗口函数(WITH...AS, ROW\_NUMBER)以及多表连接等高级SQL查询技巧，来解决一系列实际的薪酬计算问题。

本关共有6个子关卡，实际完成6个子关卡。以下选择其中具有代表性的2关进行详细阐释。

### 2.2.1 统计各单位的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资

本关要求在不考虑兼职收入的情况下，对每个单位的全职薪酬数据进行深度统计。需要计算的指标包括：薪资总额、单位月平均薪资、单笔最高/最低薪资，以及最复杂的中位薪资。

本关查询的复杂性主要体现在需要计算多个不同维度的统计指标，特别是中位数的计算。本关中我的思路是通过WITH子句构建了多个公共表表达式（CTE），将复杂问题分解为一系列清晰的、可管理的步骤。详细代码见附录部分代码 4，代码流程示意见下图 2.3。

第一步，需要对数据进行预处理。将wage表和client表连接，预先筛选出所有与有效客户关联的全职薪酬记录，避免“悬浮元组”的干扰。

第二步，分步计算各项指标。

1. personal：计算每个员工的总薪酬，以及每个员工在每个单位的“个人月平均工资”，用以在后续计算中位数。
2. minmax：直接从薪酬流水中找出每个单位的单笔最高和最低薪资。
3. stats：计算单位级别的宏观指标，如总薪酬、总人数和单位月平均薪资。
4. ranks：使用窗口函数 ROW\_NUMBER() 对每个单位内的员工按其“个人月平均工资”进行排序并编号。
5. 第三步，将上述所有CTE的结果聚合在一起，进行中位数计算与最终聚合。由于中位数在奇数人数和偶数人数时计算方式不同，通过IF判断单位总人数的奇偶性，来执行不同的中位数计算逻辑。

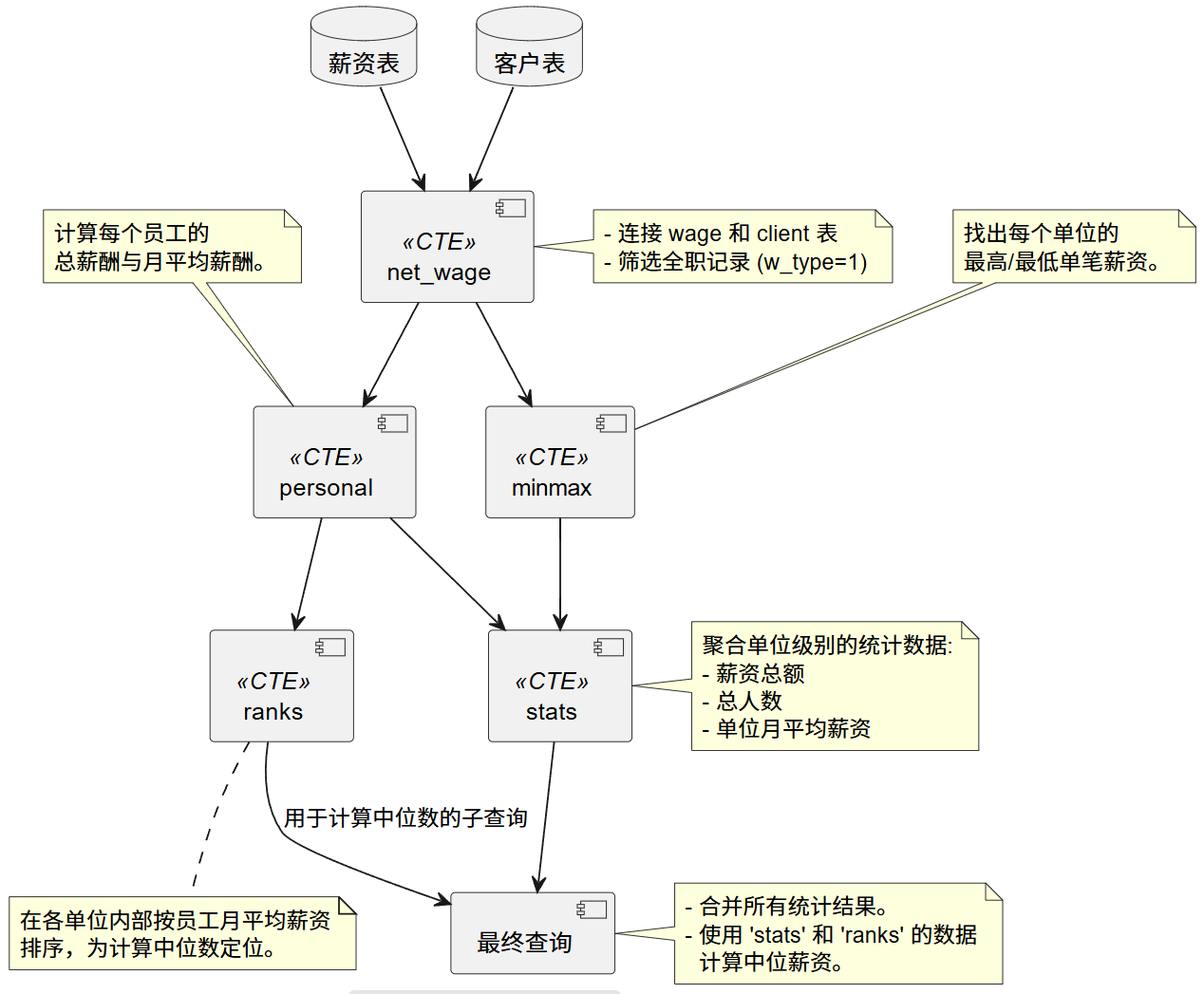


图 2.3 统计各单位的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资流程示意图

### 2.2.2 对指定身份证号的客户2023年的酬劳代扣税

本任务要求为指定身份证号的客户计算其2023年度的个人所得税，并将其更新到该年度的每一笔薪资记录中。税收规则为：年收入超过6万元的部分，按20%的税率征税。计算出的总税额需要按比例分摊到当年的每一笔薪资记录中进行扣除，并更新扣税标志。

本关的复杂之处在于更新操作依赖于对同一批数据预先计算出的聚合值（年总收入）。本关代码中运用了MySQL的会话变量（Session Variables）来解决这个问题，避免了复杂的子查询和连接，使得整个过程清晰且高效。本关代码见附录，操作流程示意见图2.4。

首先，准备阶段获取关键信息并存入变量。通过身份证号查询client表，找到对应的客户内部ID (c\_id)，并将其存入会话变量@cid。

然后，利用@cid变量，从wage表中聚合计算出该客户在2023年的总收入，并将结果存入会话变量@total\_salary。至此，后续更新操作所需的所有前提条件（客户ID、年总收入）都已准备就绪。

执行阶段，首先WHERE w\_c\_id=@cid AND year(w\_time)='2023' 条件准确地锁定了需要更新的所有薪资记录。然后使用IF函数，判断预先计算好的@total\_salary是否超过60000，直接为所有相关记录设置正确的扣税标志。同样，使用IF函数判断是否需要扣税。如果年收入未超6万，则减去0，金额不变，否则根据要求给出的规则进行扣税，通过w\_amount/@total\_salary计算当前这条薪资记录占年度总收入的比例。

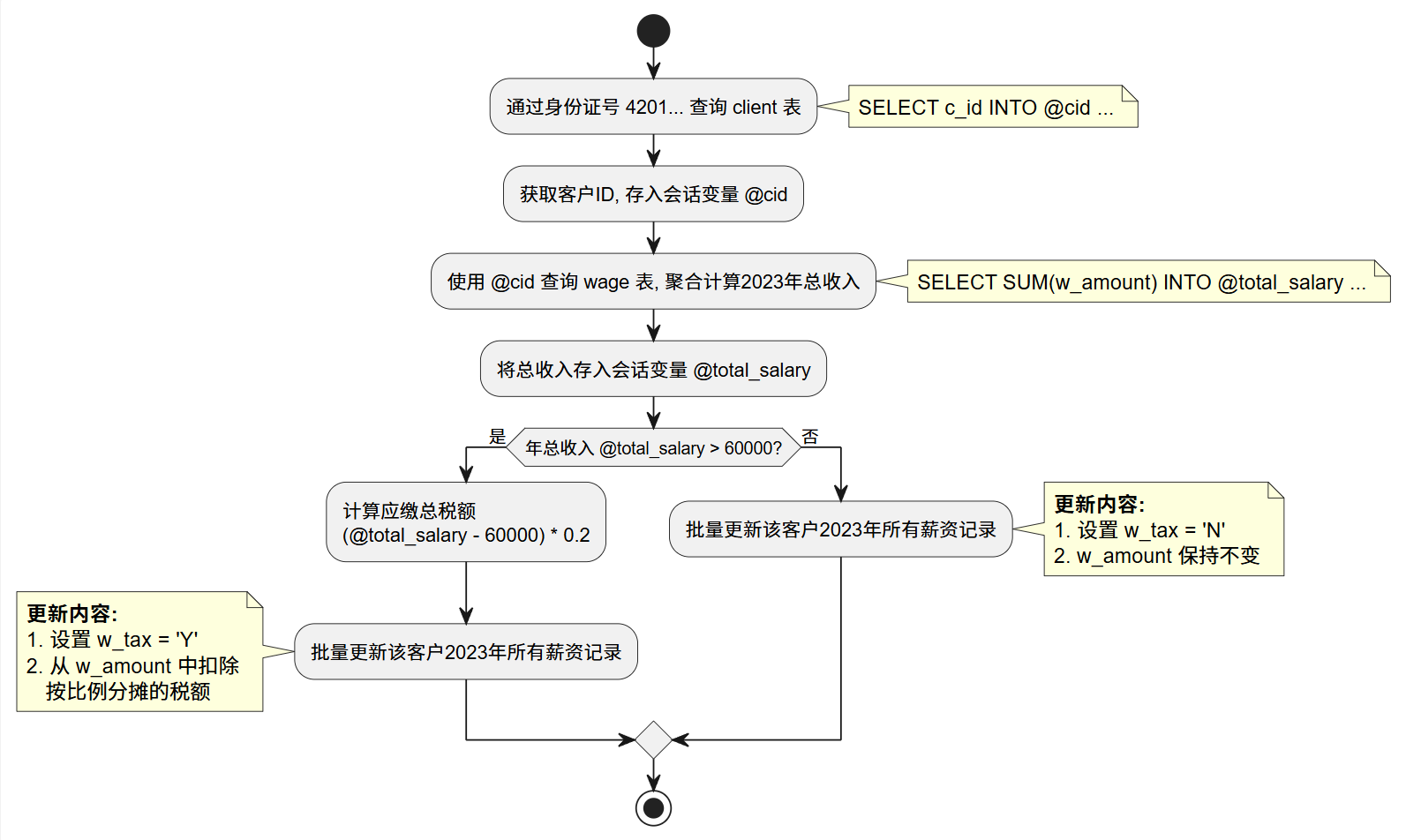


图2.4 对指定客户扣税流程图

## 2.3 存储过程与事务

本关通过三个递进的编程任务，练习MySQL中定义和使用存储过程的核心技能，并理解事务在保证数据一致性中的关键作用。

本关共有3个子任务，实际完成2个。以下选择第3子关进行详细阐释。

### 2.3.1使用事务的存储过程

本关任务要求编写存储过程sp\_transfer ，用于实现在两个银行账户之间进行转账。此过程需处理复杂的业务规则，并确保操作的原子性，即转账要么完全成功，要么完全失败，数据库状态不会处于中间状态。本关代码见附录代码 6，流程示意见图2.5。

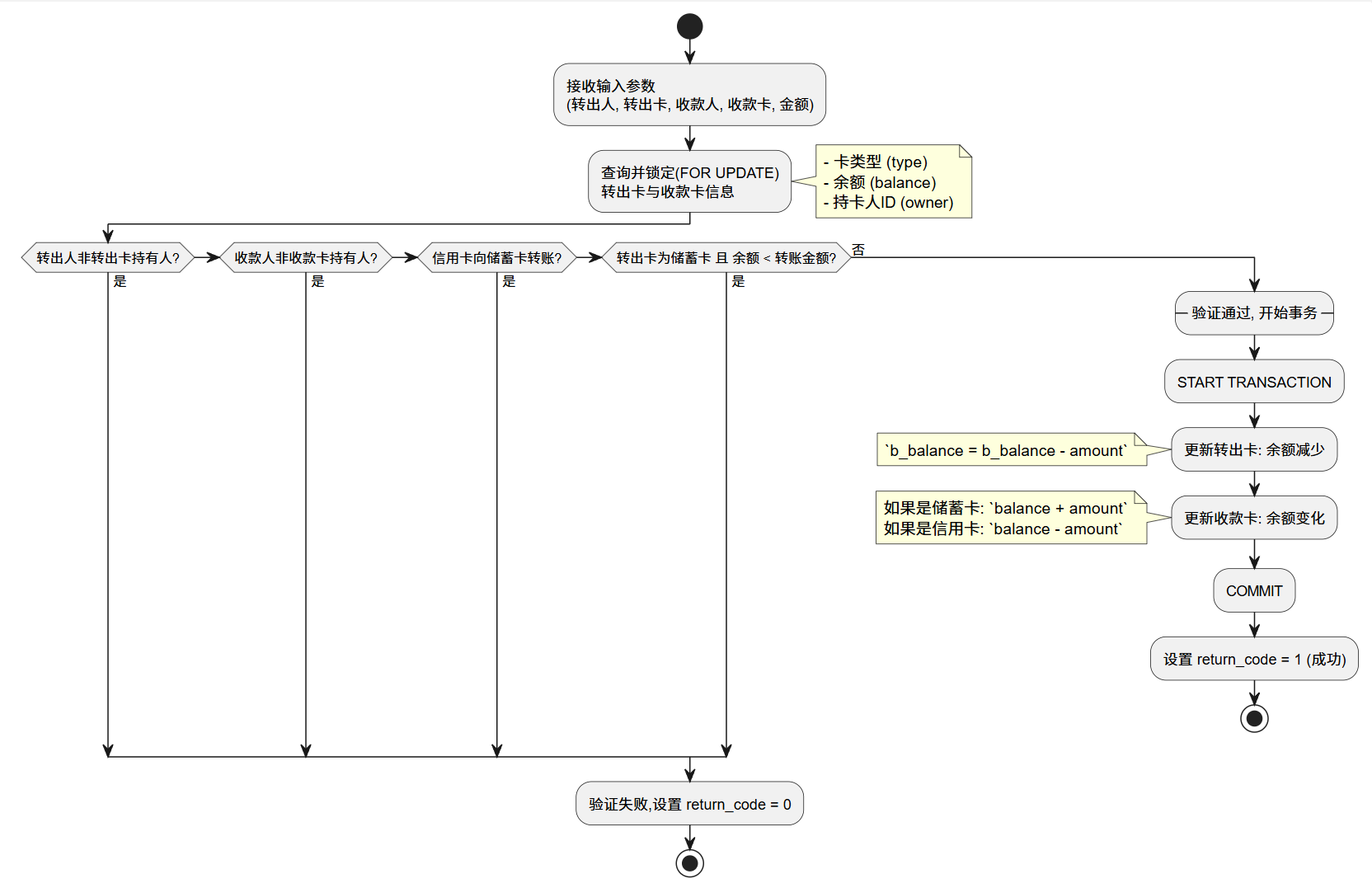


图2.5 使用事务的存储过程流程图

存储过程sp\_transfer接收五个输入（IN）参数，包括付款人、收款人、双方卡号及转账金额。输出（OUT）参数 return\_code用于向调用者反馈操作结果（1为成功，0为失败）。

执行更新操作之前，过程首先使用 SELECT ... INTO ... 语句查询转出卡和转入卡的信息。这里的关键点在于查询语句末尾使用 FOR UPDATE 子句，它会对查询结果中的相应行施加一个排他锁，防止在本次转账事务完成之前有其他并发事务修改这两张卡的余额，从而可以避免“脏读”和“不可重复读”等并发问题，保证了数据的一致性。

在启动事务前，还需进行一系列严格的条件检查，提前拦截无效操作，减少不必要的事务开销。首先，进行所有权验证，检查发起转账的客户是否是转出卡的合法持有人，以及收款人是否是收款卡的持有人。然后，进行转账规则验证，禁止从“信用卡”向“储蓄卡”转账，且确保“储蓄卡”作为转出方时，其余额充足。任何一条规则不满足，都会立即将 return\_code 设为0，并终止执行。

在所有前置校验通过后，才通过 START TRANSACTION 显式地开启一个事务。所有后续的数据库更新操作将作为一个不可分割的单元执行。根据卡的类型更新双方账户余额。对于储蓄卡是简单的加减法。对于信用卡还款（向信用卡转账），余额是做减法，因为其 b\_balance 字段代表的是已透支金额。当所有更新语句成功执行后，使用 COMMIT 提交事务，使所有更改永久生效。同时设置 return\_code 为1，表示成功。

## 2.4 数据库设计与实现

本关主要围绕数据库设计与实现的完整流程展开，主要包含两大模块：

一是从需求到模型，基于给定的影院管理系统业务场景，进行需求分析，绘制E-R图完成概念与逻辑建模，最终产出规范化的关系模式。

二是从模型到实现，将一个已定义好的逻辑模型转化为MySQL上的物理数据库。

本关共有3个子关卡，实际完成3个子关卡。以下将详细阐述这3关。

### 2.4.1 从概念模型到MySQL实现

本关要求根据提供的关于机票订票系统逻辑模型的E-R图（见图 2.6）及文字描述，编写一套完整的SQL DDL脚本。该脚本需在MySQL中创建名为flight\_booking的数据库，并精确构建user, passenger, airport, airline, airplane, flightschedule, flight, ticket等八个表，同时定义各表的主键、外键、非空、唯一、默认值以及索引等约束，从而将抽象的逻辑设计转化为具体的物理数据库实现。



图 2.6 机票订票系统概念模型ER图

本关的核心在于：将E-R图中定义的实体、属性和关系，逐一映射为SQL中的表、列和约束。整体思路遵循“先独立实体，后关联实体”的原则，以确保在定义外键时，其引用的主键已经存在。

1. **数据库初始化**

为保证脚本的可重复执行性，在脚本开头使用DROP DATABASE IF EXISTS flight\_booking;和CREATE DATABASE flight\_booking;，确保每次运行都能在一个干净的环境中进行。随后使用USE flight\_booking;指定后续操作的上下文。

1. **表结构创建**

严格遵循“先独立实体，后关联实体”的创建顺序。通过梳理E-R图中的依赖关系，我们可以规划出一条无冲突的创建路径：user, passenger, airport -> airline -> airplane, flightschedule -> flight -> ticket。

首先，创建独立实体表user, passenger, airport。这些表的创建相对直接，主要是将模型中的属性精确转换为MySQL的列定义，并根据文字描述中的约束需求，附加PRIMARY KEY AUTO\_INCREMENT, NOT NULL, UNIQUE, DEFAULT等约束。

然后，依次创建关联实体表airline -> airplane, flightschedule -> flight -> ticket。

在airline表中，还根据要求为name创建索引，提升查询性能。

需要注意的是，在flightschedule和flight表中，表示出发地和到达地的外键列被命名为from和to。这两个词是SQL的保留关键字。为了避免语法错误，需要使用反引号（`）将它们括起来，如`from`和`to`。

由于篇幅限制，最终实现的SQL代码详见源代码附件。

### 2.4.2 从需求分析到逻辑模型

本关任务要求根据影院管理系统的业务需求，先绘制E-R图，再将其转换为关系模式。

1. **E-R图绘制**

首先，我们需要仔细分析本关任务描述中的业务功能，识别出核心的实体和它们之间的关系。根据描述，可以明确地识别出五个实体：电影(movie)、顾客(customer)、放映厅(hall)、排场(schedule) 和 电影票(ticket)。每个实体的属性也已在描述中清晰列出。其中，顾客与电影票是“购买”关系，一个顾客可购买多张票，一张票只被一个顾客购买，是1:N关系；电影票与排场是“属于”关系，多张票属于一个排场，是N:1关系；排场与电影是“放映”关系，一个排场放映一部电影，一部电影可有多个排场，是N:1关系；排场与放映厅是“位于”关系，一个排场位于一个放映厅，一个放映厅可有多个排场，是N:1关系。

基于以上分析，使用tikz工具绘制出E-R图，见图 2.7。

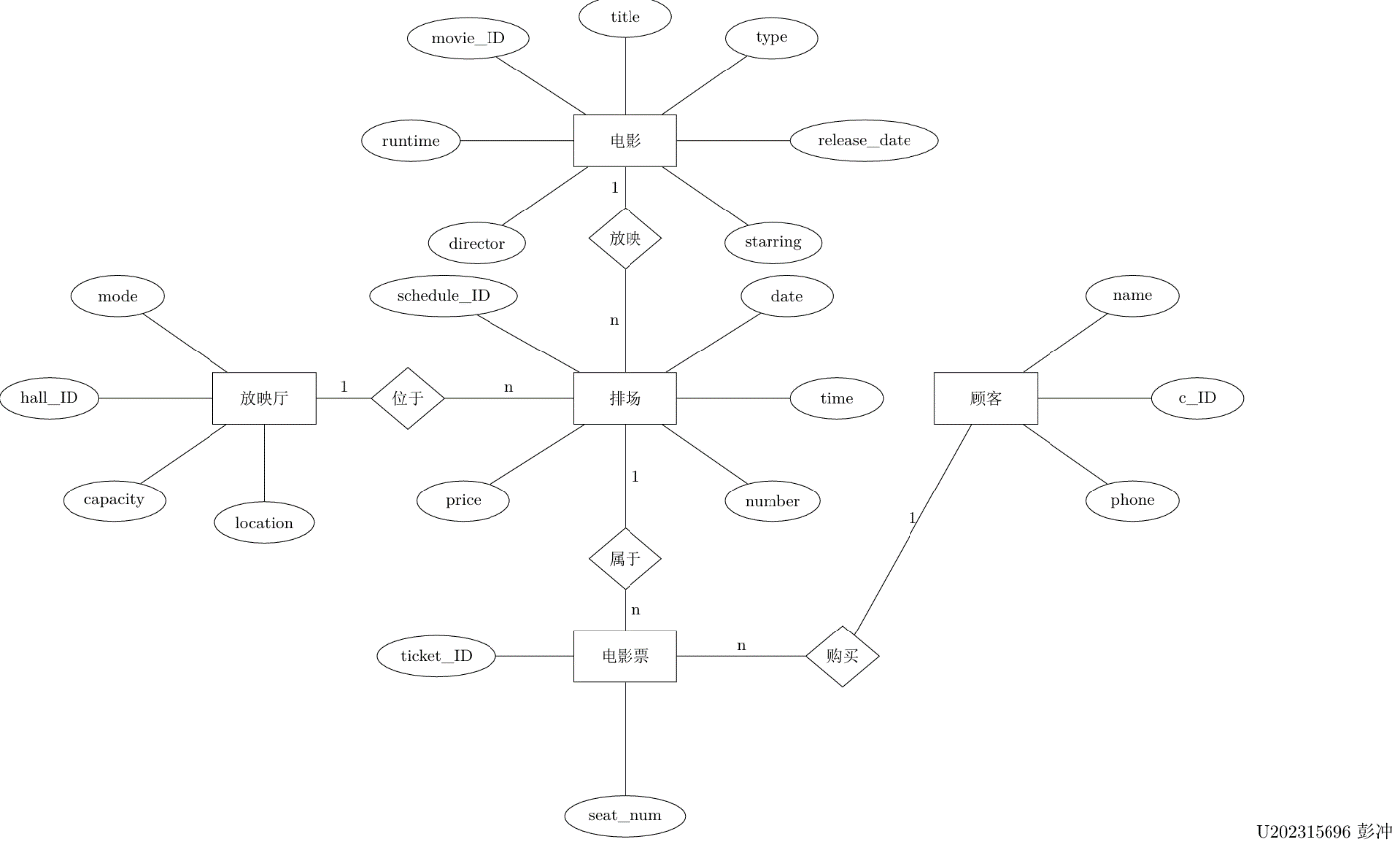


图 2.7 影院管理系统E-R图模型

1. **转换关系模式**

有以上分析可知，各实体间均为1:N关系，而无多对多关系，因此无须为关系本身创建新的、独立的关联表。所有的联系都可以通过在“N”端实体对应的关系模式中，添加“1”端实体的主码作为外码来实现。

因此，每个实体集都转换为一个独立的关系模式。实体的属性成为关系模式的属性，实体的标识符成为关系模式的主码。最终生成的关系模式见下图 2.8。

|  |
| --- |
| movie(movie\_ID, title, type, runtime, release\_date, director, starring), 主码: (movie\_ID)  customer(c\_ID, name, phone), 主码: (c\_ID)  hall(hall\_ID, mode, capacity, location), 主码: (hall\_ID)  schedule(schedule\_ID, date, time, price, number, movie\_ID, hall\_ID), 主码: (schedule\_ID); 外码: (movie\_ID)→movie(movie\_ID), (hall\_ID)→hall(hall\_ID)  ticket(ticket\_ID, seat\_num, schedule\_ID, c\_ID), 主码: (ticket\_ID); 外码: (schedule\_ID)→schedule(schedule\_ID), (c\_ID)→customer(c\_ID) |

图 2.8 关系模式

### 2.4.3 建模工具的使用

本关任务旨在了解使用专业的数据库建模工具（如MySQL Workbench）进行数据库设计，并通过其“Forward Engineering”功能，从可视化模型自动生成SQL DDL脚本。

第一步，如图图 2.9，使用MySQL Workbench打开rbac.mwb模型文件（File -> Open Model..->在文件浏览器窗口选中rbac.mwb文件）。

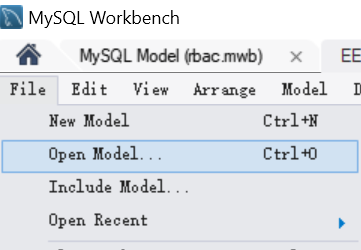


图 2.9 第一步

第二步，打开后，点击菜单上的Database-> Forward Engineering，如图 2.10。点击“Forward Engineering”，在弹出的导引窗口中一直点Next，直至Review SQL Script步骤，便可见到自动生成的SQL脚本，如图 2.11。

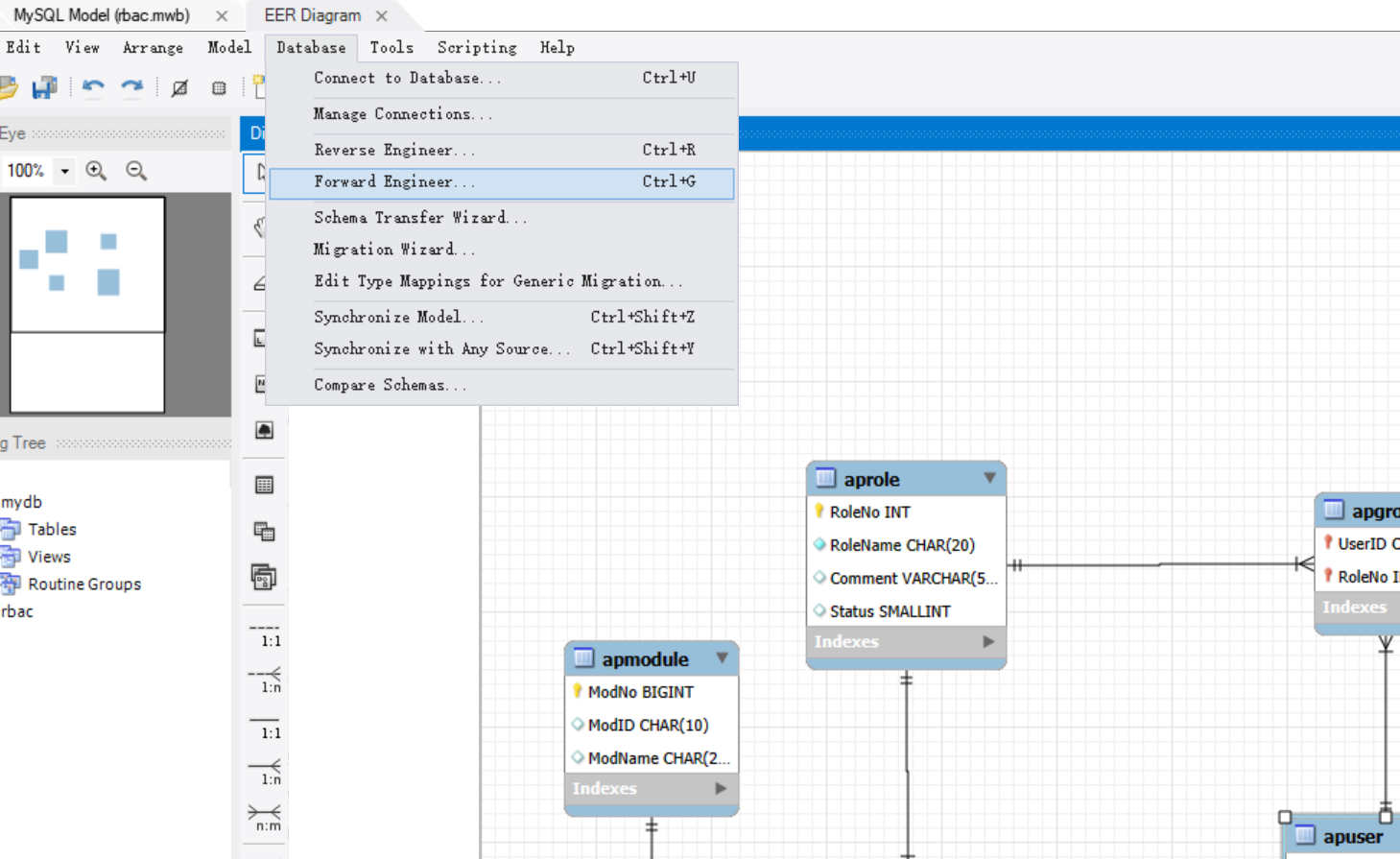


图 2.10 第二步

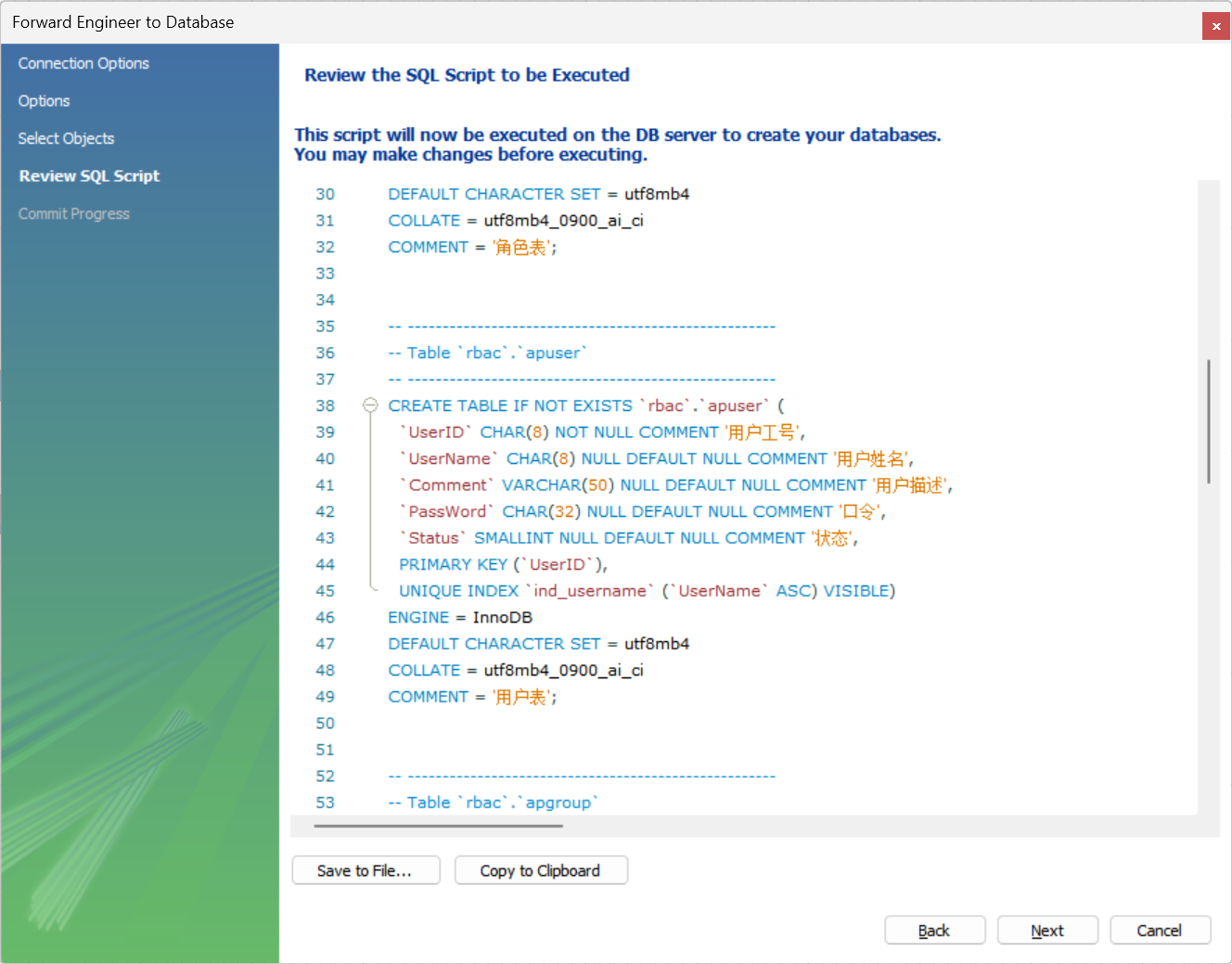


图 2.11 第二步

### 2.4.4 制约因素分析与设计

在数据库设计与实现的实训任务中，尽管我们的核心任务是技术实现，但在方案设计层面，我们必须将社会、安全、法律等现实世界的制约因素纳入考量。一个成功的系统不仅功能完备，更应是负责任、安全可靠且合规的。以下是在设计过程中，对这些因素的一些考虑。

1. **数据隐私**

系统中收集了大量个人身份信息，例如，user和passenger表中的姓名、身份证号、电话、邮箱、生日等。在设计上，通过UNIQUE约束保证了身份证、手机号等信息的唯一性，这既是业务要求，也符合数据管理的准确性原则；在实际应用中，对这些敏感数据的存储、传输和访问必须进行加密和严格的权限控制，以遵守有关的法律法规。

1. **文化差异**

user表中设置firstname和lastname字段，而非单独的name字段，是为了适应东西方对姓名的不同表达习惯。airport表中设置country和city字段，也是为了适应国际化需求。

1. **访问安全与权限控制**

在第一关机票预订系统的user表中，通过admin\_tag字段明确区分了普通用户和管理员。第3关的RBAC模型通过将权限（apright）赋予角色（aprole），再将角色分配给用户（apgroup），遵循了“最小权限原则”，确保用户只能访问其职责所需的数据和功能，有效防止了越权操作和内部数据泄露，保障了系统的整体安全。

### 2.4.5 工程师责任及其分析

在设计和实现机票预订及权限管理系统的过程中，我们必须认识到，每一行代码、每一个数据表的设计，都可能与社会、安全、法律及文化等因素产生复杂的相互影响。

1. **安全与法律责任**

在机票预订系统中，我们处理的是包括姓名、身份证号、联系方式在内的高度敏感个人信息。若数据库设计不周，如密码明文存储、缺少访问控制，可能导致大规模数据泄露，不仅会给旅客带来财产损失和隐私侵害，更可能触犯有关法律法规，使企业和个人面临严重的法律后果。

1. **社会与文化责任**

软件系统是社会运行的工具，其设计应体现对社会多样性和人文需求的关怀。本次设计中将user（用户）与passenger（旅客）分离，是考虑到了现实社会中的互助行为，使系统更具人情味和实用性。

## 2.5 数据库应用开发(JAVA篇)

本关旨在通过一系列递进的编程任务，锻炼使用Java JDBC（Java DataBase Connectivity,java数据库连接）进行数据库应用开发的核心技能。该实训共包含七个关卡，内容涵盖了数据库的增删改查（CRUD）操作、用户身份验证、事务处理以及数据结构转换。

本关共有7个子关卡，实际完成7个子关卡。以下是对其中代表性的关卡的详细阐述。

### 2.5.1 客户修改密码

本关任务要求实现用于修改客户的登录密码的passwd方法。该方法需要接收数据库连接、客户邮箱、旧密码和新密码作为参数。在执行修改前，必须先验证客户邮箱是否存在以及旧密码是否正确。根据执行结果，方法需返回一个特定的整数值来表示不同状态：1代表成功，2代表用户不存在，3代表密码错误，-1代表程序异常。

本关的关键逻辑是“查询-验证-更新”，为保证数据操作的准确性和安全性，不能直接执行UPDATE。

本关代码详见附件，执行流程见示意图 2.12。流程说明如下：

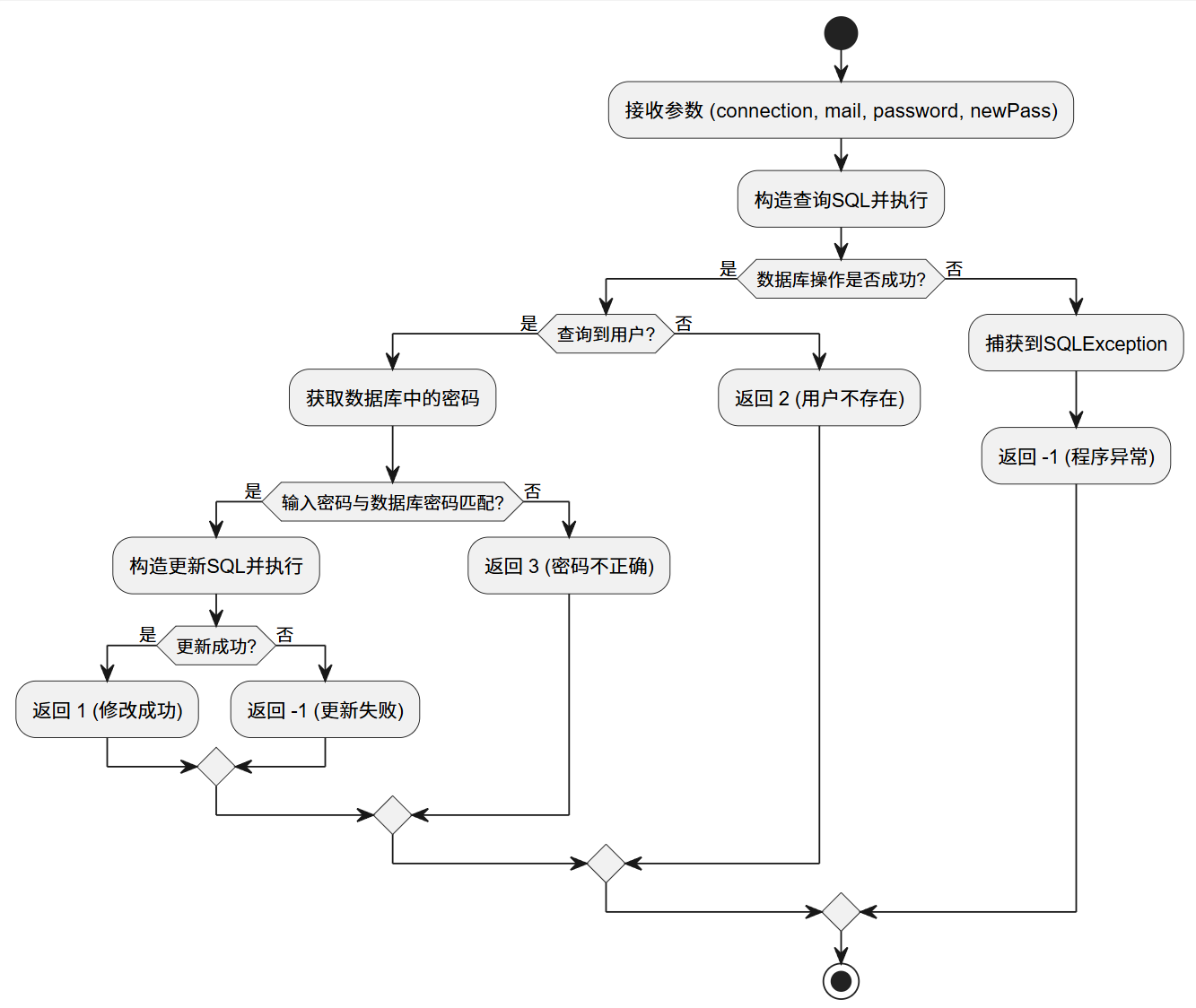


图 2.12 passwd方法流程图

首先，查询用户并验证身份，根据传入的客户邮箱（mail）参数查询client表。如果查询结果为空，说明该邮箱对应的用户不存在，方法应返回2。如果用户存在，则从查询结果中取出该用户存储在数据库中的密码。将取出的密码与用户传入的旧密码（password）进行比对，如果密码不匹配，说明旧密码错误，方法应返回3。如果旧密码验证通过，则说明身份正确。

然后，更新密码。执行UPDATE语句，将该邮箱对应用户的密码更新为新密码（newPass）。根据UPDATE操作影响的行数判断是否更新成功。若影响行数大于0，则表示修改成功，方法返回1。

在整个数据库操作过程中，使用try-catch块捕获可能发生的SQLException。一旦捕获到异常，说明数据库连接或SQL执行出现问题，方法应返回-1。

出于安全性考虑，代码全部使用了PreparedStatement，通过参数占位符?来设置查询和更新的条件值，而非使用Statement，避免SQL注入攻击的风险。

### 2.5.2 事务与转账操作

本关任务要求实现银行卡转账方法 transferBalance，模拟完整的转账流程：从一个储蓄卡账户（转出方）扣除指定金额，并将该金额增加到另一个账户（转入方）。

本关关键是利用JDBC的事务处理机制，确保转账操作的原子性，即转出账户的扣款和转入账户的存款必须同时成功或同时失败。转账失败的条件包括：转出或转入卡号不存在、转出卡为信用卡、或转出卡余额不足。

本关代码详见附件，执行流程见示意图 2.13。流程说明如下：

1. **关闭事务自动提交**

在所有数据库操作开始前，调用 connection.setAutoCommit(false) 关闭自动提交。

1. **校验转出方及转入方**

查询转出卡信息，检查卡是否存在、是否为储蓄卡、以及余额是否足够支付转账金额。任一条件不满足，则调用 connection.rollback() 回滚事务并返回 false。查询转入卡信息，检查卡是否存在。若不存在，同样回滚事务并返回 false。

1. **执行转账**

执行 UPDATE 语句，从转出卡余额中减去转账金额。根据转入卡的类型，若为储蓄卡，则余额b\_balance = b\_balance + amount；若为信用卡，则代表还款，透支额b\_balance = b\_balance - amount。

1. **提交或回滚**

如果所有 UPDATE 操作都成功执行（影响行数均为1），则调用 connection.commit() 提交事务，使所有更改永久生效，并返回 true。

如果在任何步骤中出现SQL异常或更新影响的行数不为1，则在 catch 块中调用 connection.rollback() 撤销该事务内所有已执行的操作，并返回 false。

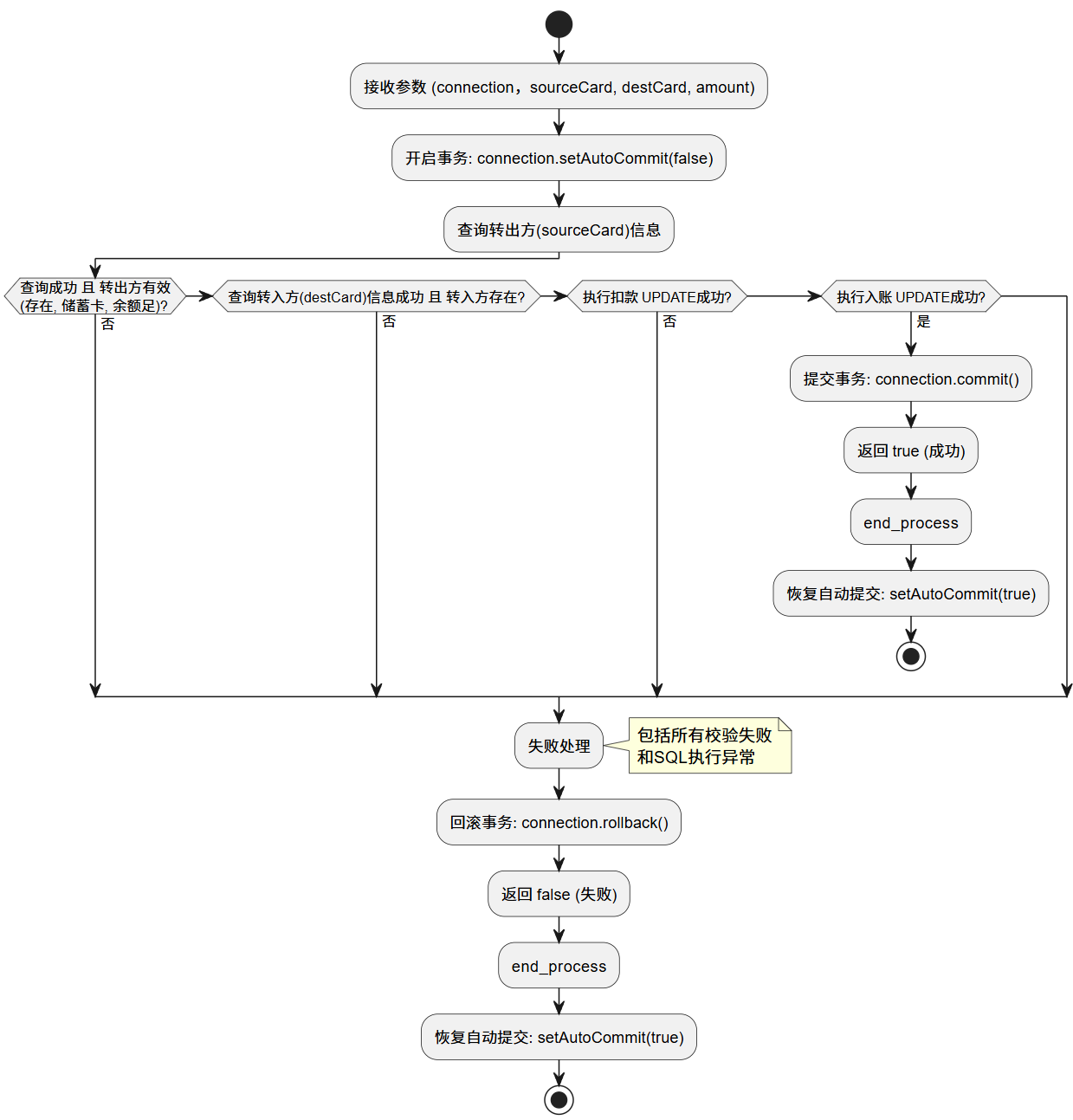


图 2.13 transferBalance流程图

### 2.5.3 把稀疏表格转为键值对存储

本关任务要求实现一个Java方法，将一个多列的、数据稀疏的“宽表” (entrance\_exam) 中的数据，转换并存储到一个“键-值”对形式的“长表” (sc) 中。

具体来说，需要读取entrance\_exam表中的每一行记录，并将其中的非空成绩（如语文、数学、英语等）作为独立的行插入到sc表中，从而实现数据结构的优化，节省存储空间并便于后续的统计分析。

本关的关键在于将表的列转换为行，需要对查询结果进行二次处理和循环插入。

本关代码详见源文件，执行流程图见图 2.14，下对执行过程进行说明。

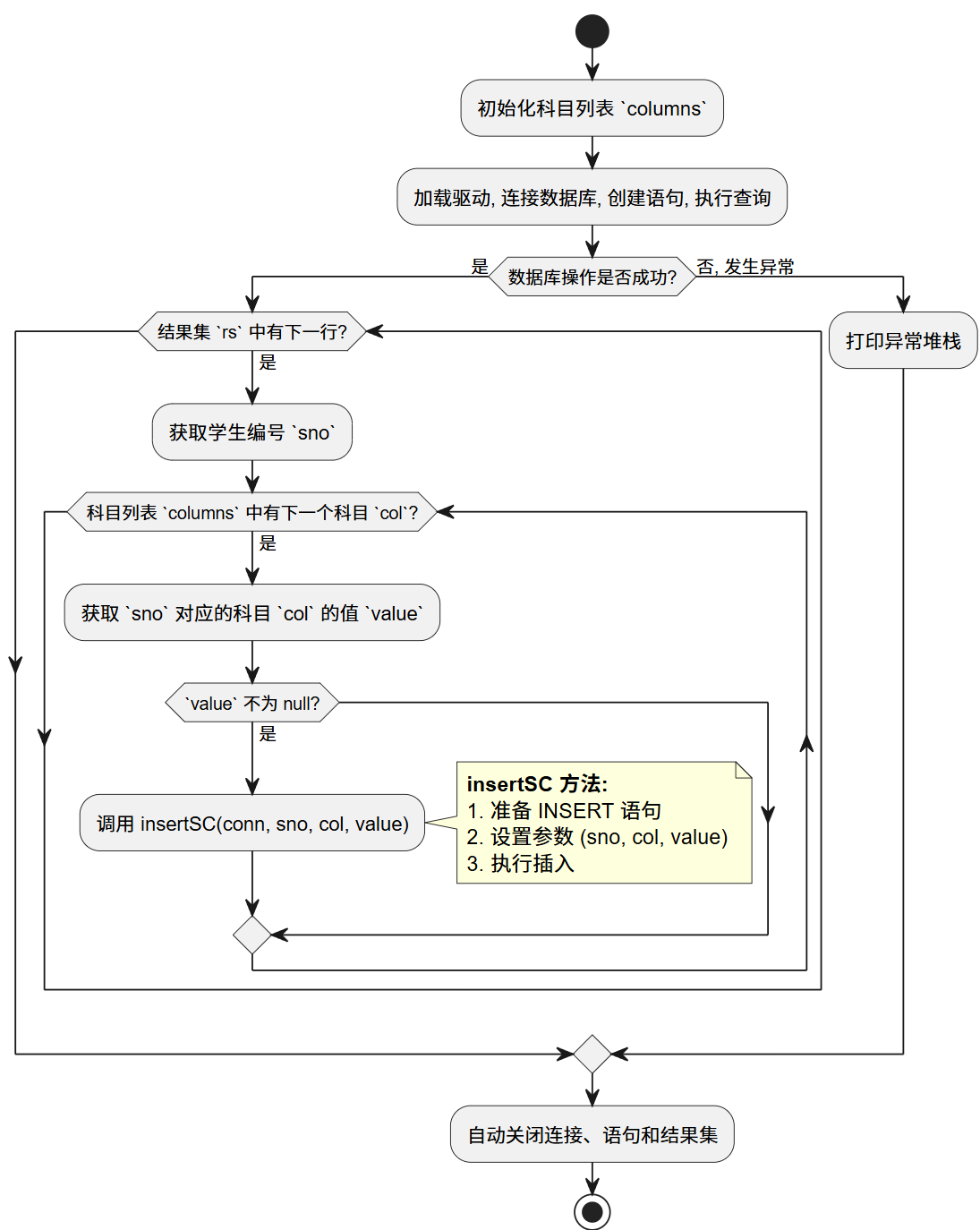


图 2.14 稀疏表格转储流程图

首先，查询源数据，获取源表中的所有学生及其各科成绩记录，结果存放在rs(ResultSet)中。

然后，使用while (rs.next())循环遍历查询结果中的每一行，即每一个学生。对于每个学生，使用for循环遍历预先定义的科目名称数组。通过rs.getString(科目名)，获取当前学生在当前科目下的成绩。

接下来，关键步骤是判断该成绩值是否为null。如果成绩值不为null，则调用独立的insertSC方法，将学生编号（sno）、科目名（col）和成绩值（value）作为一条新记录插入到目标表sc中，若成绩值为null则跳过，继续遍历下一科目。

当两层循环结束，则所有数据转换完成。如果在此过程中的任何步骤发生错误（如数据库连接失败、SQL语法错误等），catch 块会捕获这个异常并将其信息打印到控制台。

# 3 课程总结

本次《数据库系统原理实践》课程是一次理论与实践紧密结合的综合性训练。课程以主流关系型数据库MySQL为平台，通过一系列精心设计的、由浅入深的实训任务，让我将《数据库系统概论》课程中学到的理论知识付诸实践。

在为期数周的实践中，我完成了课程平台中的实训任务1-15，其中跳过了实训8的第2关，总计126分。完成情况归纳如下：

**数据对象管理与编程：**熟练掌握了使用SQL的数据定义语言（DDL）部分。包括创建数据库、设计表结构，并为其添加主键、外键、唯一、检查等多种完整性约束，以及视图、过程（Procedure）、用户自定义函数和触发器的创建与使用。

**数据处理与查询：**熟练掌握了使用SQL的数据操纵语言（DML）部分。包括对数据的增删改查操作，以及单表查询、多表连接、子查询、聚合函数以及窗口函数等复杂查询技巧。

**数据库系统内核机制：**理解了数据库的安全性控制；亲手实践并观察了脏读、不可重复读等并发问题，对事务的ACID特性与并发控制有了直观且深刻的认识；掌握了保障数据可靠性的基本方法。

**数据库设计与实现：**根据给定的需求，完成了从需求分析到概念模型（E-R图），再到逻辑模型（关系模式）的完整设计流程。

**数据库应用开发：**将数据库技能与编程语言相结合，完成了使用Java程序连接数据库、执行CRUD操作、处理事务以及进行数据结构转换等任务。

由于学期时间安排较少等原因，没来得及探讨实训16关于存储管理器的内容，我深感遗憾。实际上，在上这门课之前，我很期待这种DBMS开发实现的内容，但课程重点落在了SQL的使用以及数据库的设计与应用，想必这背后有着课程组的考量。

通过本次课程实践，我最大的收获是实现了从“知道”到“会用”的转变。理论课上的抽象概念，如事务、索引等，在一次次亲手调试和验证中变得具体而生动。在这个过程中，我深刻体会到严谨的逻辑思维和关注细节的重要性。一个微小的SQL错误或设计缺陷，都可能导致数据不一致或系统性能问题。特别是实训12的并发控制和实训15的事务处理，让我感叹多用户环境下保证数据正确性的复杂与精妙。

在此，我谨向负责本课程的老师及课程团队致以最诚挚的感谢。得益于老师们对课程实践体系的精心设计，我们才能通过一系列系统且层层递进的任务，将抽象的理论知识有效地应用于实践。在整个学习过程中，老师的耐心指导与悉心答疑，为我们攻克难关、顺利完成所有实训任务提供了巨大的支持与帮助。这门课程不仅让我掌握了宝贵的专业技能，更点燃了我对数据库技术的探索热情。对此，我表示由衷的感谢！

# 附录

|  |
| --- |
| SELECT c.c\_id,c\_name, IFNULL(SUM(t.prop), 0) AS total\_property  FROM client c  LEFT JOIN (  SELECT b\_c\_id AS tid, SUM(b\_balance) AS prop  FROM bank\_card  WHERE b\_type='储蓄卡'  GROUP BY b\_c\_id  UNION ALL  SELECT b\_c\_id AS tid, SUM(b\_balance\*-1) AS prop  FROM bank\_card  WHERE b\_type='信用卡'  GROUP BY b\_c\_id  UNION ALL    SELECT p.pro\_c\_id AS tid, p.pro\_quantity \* fp.p\_amount + p.pro\_income AS prop  FROM property p  JOIN finances\_product fp ON p.pro\_type = 1 AND p.pro\_pif\_id = fp.p\_id  UNION ALL  SELECT p.pro\_c\_id AS tid, p.pro\_quantity \* i.i\_amount + p.pro\_income AS prop  FROM property p  JOIN insurance i ON p.pro\_type = 2 AND p.pro\_pif\_id = i.i\_id  UNION ALL  SELECT p.pro\_c\_id AS tid, p.pro\_quantity \* f.f\_amount + p.pro\_income AS prop  FROM property p  JOIN fund f ON p.pro\_type = 3 AND p.pro\_pif\_id = f.f\_id  ) AS t ON c.c\_id=t.tid  GROUP BY c.c\_id; |

代码 3 客户理财、保险与基金投资总额SQL代码

|  |
| --- |
| WITH net\_wage AS(  SELECT \*  FROM wage  JOIN client ON client.c\_id=wage.w\_c\_id ),  personal AS(  SELECT w\_org, w\_c\_id,  SUM(w\_amount) AS person\_total,  ROUND(SUM(w\_amount)/COUNT(DISTINCT DATE\_FORMAT(w\_time, '%Y-%m')), 2) AS person\_avg  FROM net\_wage  WHERE w\_type=1  GROUP BY w\_org, w\_c\_id ),  minmax AS (  SELECT w\_org,  MAX(w\_amount) AS max\_wage,  MIN(w\_amount) AS min\_wage  FROM net\_wage  WHERE w\_type=1  GROUP BY w\_org ),  months AS (  SELECT w\_org, COUNT(DISTINCT DATE\_FORMAT(w\_time, '%Y-%m')) AS cnt\_month  FROM net\_wage  WHERE w\_type=1  GROUP BY w\_org),  stats AS (  SELECT p.w\_org,  SUM(p.person\_total) AS total\_amount,  COUNT(DISTINCT p.w\_c\_id) AS cnt\_person,  ROUND(SUM(p.person\_total)/COUNT(DISTINCT p.w\_c\_id)/m.cnt\_month,2) AS average\_wage,  mm.max\_wage,min\_wage  FROM personal p, months m, minmax mm  WHERE p.w\_org=m.w\_org AND p.w\_org=mm.w\_org  GROUP BY w\_org),  ranks AS (  SELECT w\_org, person\_avg,  ROW\_NUMBER() OVER(PARTITION BY w\_org ORDER BY person\_total) AS rn\_asc  FROM personal)  SELECT p.w\_org, s.total\_amount,average\_wage,max\_wage,min\_wage,  ROUND(IF(s.cnt\_person%2=1,  -- 奇数，取中间  (SELECT person\_avg FROM ranks WHERE w\_org = s.w\_org AND rn\_asc = (s.cnt\_person + 1) / 2),  -- 偶数，取中间两人平均  (  (SELECT person\_avg FROM ranks WHERE w\_org = s.w\_org AND rn\_asc = (s.cnt\_person / 2)) +  (SELECT person\_avg FROM ranks WHERE w\_org = s.w\_org AND rn\_asc = (s.cnt\_person / 2) + 1)  ) / 2  ),2) AS mid\_wage  FROM personal p, months m, stats s  WHERE p.w\_org=m.w\_org AND p.w\_org=s.w\_org  GROUP BY s.w\_org, s.total\_amount, s.cnt\_person, m.cnt\_month  ORDER BY s.total\_amount DESC; |

代码 4 统计各单位的薪资总额、月平均薪资、最高薪资、最低薪资、中位薪资SQL代码

|  |
| --- |
| SELECT SUM(w\_amount) INTO @total\_salary  FROM wage  WHERE w\_c\_id = @cid AND YEAR(w\_time) = 2023;  UPDATE wage  SET w\_tax=IF(@total\_salary>60000,'Y','N'),  w\_amount=w\_amount-IF(@total\_salary>60000,ROUND((@total\_salary-60000)\*0.2\*(w\_amount/@total\_salary), 2),0)  WHERE w\_c\_id=@cid AND year(w\_time)='2023'; |

代码 5 对指定客户扣税SQL代码

|  |
| --- |
| use finance1;  -- 在金融应用场景数据库中，编程实现一个转账操作的存储过程sp\_transfer\_balance，实现从一个帐户向另一个帐户转账。  -- 请补充代码完成该过程：  delimiter $$  create procedure sp\_transfer(  IN applicant\_id int,  IN source\_card\_id char(30),  IN receiver\_id int,  IN dest\_card\_id char(30),  IN amount numeric(10,2),  OUT return\_code int)  proc\_label: BEGIN  DECLARE source\_type CHAR(20);  DECLARE dest\_type CHAR(20);  DECLARE source\_balance NUMERIC(10,2);  DECLARE dest\_balance NUMERIC(10,2);  DECLARE source\_owner INT;  DECLARE dest\_owner INT;  -- 检查付款卡  SELECT b\_type, b\_balance, b\_c\_id INTO source\_type, source\_balance, source\_owner  FROM bank\_card WHERE b\_number = source\_card\_id FOR UPDATE;  -- 检查收款卡  SELECT b\_type, b\_balance, b\_c\_id INTO dest\_type, dest\_balance, dest\_owner  FROM bank\_card WHERE b\_number = dest\_card\_id FOR UPDATE;  -- 仅当转款人是转出卡的持有人时，才可转出  IF source\_owner != applicant\_id THEN  SET return\_code = 0;  LEAVE proc\_label;  END IF;  -- 仅当收款人是收款卡的持有人时，才可转入  IF dest\_owner != receiver\_id THEN  SET return\_code = 0;  LEAVE proc\_label;  END IF;  -- 信用卡不能向储蓄卡转账  IF source\_type = '信用卡' AND dest\_type = '储蓄卡' THEN  SET return\_code = 0;  LEAVE proc\_label;  END IF;  -- 转账金额不能超过储蓄卡余额  IF source\_type = '储蓄卡' AND source\_balance < amount THEN  SET return\_code = 0;  LEAVE proc\_label;  END IF;  -- 开始事务  START TRANSACTION;  -- 储蓄卡扣款  IF source\_type = '储蓄卡' THEN  UPDATE bank\_card SET b\_balance = b\_balance - amount WHERE b\_number = source\_card\_id;  END IF;  -- 储蓄卡或信用卡收款  IF dest\_type = '储蓄卡' THEN  UPDATE bank\_card SET b\_balance = b\_balance + amount WHERE b\_number = dest\_card\_id;  ELSE  -- 信用卡收款(还款)，余额可以为负  UPDATE bank\_card SET b\_balance = b\_balance - amount WHERE b\_number = dest\_card\_id;  END IF;  COMMIT;  SET return\_code = 1;  END$$  delimiter ; |

代码 6 使用事务的存储过程SQL代码