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：	CS2207
学　　号：	U202215554
姓　　名：	付名扬
指导教师：	赵小松

分数	
教师签名	

2024 年 6 月 16 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。

课程依托头歌实践教学平台，实验环境为 Linux 操作系统下的 MySQL 8.0.28。在数据库应用开发环节，使用 JAVA 1.8。具体的实验条件如表 1-1 所示。

表 1-1 实验的具体条件和环境

课程平台	实验环境	开发环境	本机操作系统	实验时间
头歌/Educoder	MySQL 8.0.28	JAVA 1.8	Windows11 64 位	2024.3~2024.6

实验整体采用的是“逻辑分析→代码实现→结果评估”的结构。其流程如图 1-1 所示。

- **逻辑分析：**在实施之前，对每个实验章节的关卡进行了深入的设计要求分析。这一阶段的关键是从数据库的实际应用场景中抽象出概念和理论，并针对性地提炼出底层实现逻辑。这包括对数据结构、算法和数据库操作的细致审视，以确保在代码实现阶段能够高效地转化为具体的程序逻辑。
- **代码实现：**将逻辑分析阶段得出的逻辑转化为 MySQL 语言的代码。这一过程涉及到对数据库模式的设计和实现，以及对数据库操作的编程实现。编码规范和最佳实践的遵循至关重要。
- **结果评估：**完成代码后，将其提交到头歌的测试平台。评估过程旨在验证代码的正确性和性能，并识别任何潜在的缺陷或改进空间。此外，根据测试结果，对代码进行优化调整。

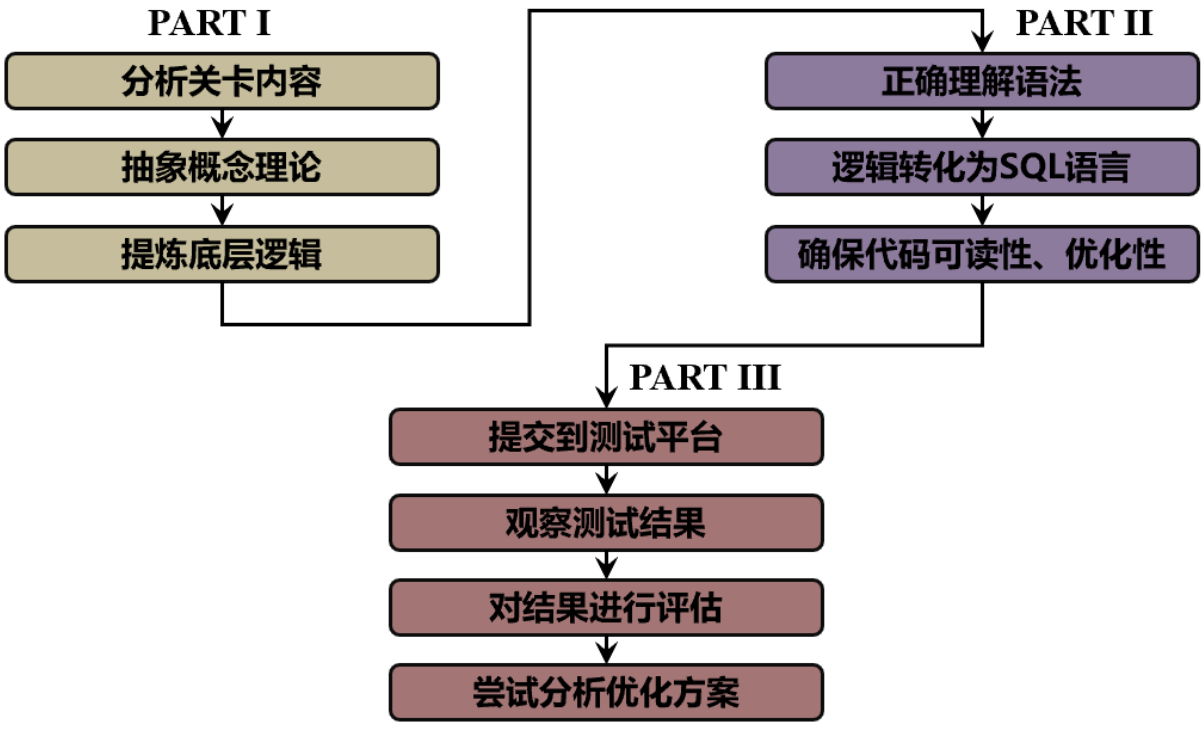

图 1-1 实验整体流程结构

2. 任务实施过程与分析

本次实验分为两大板块：实验操作部分，实验报告部分。

对于实验操作，完成了[数据库、表与完整性的定义]的 6/6 个关卡、[表结构与完整性约束的修改]的 4/4 个关卡、[基于金融应用的数据查询]的 19/19 个关卡、[数据查询新增]的 1/6 个关卡、[数据的插入、删除和修改]的 6/6 个关卡、[视图]的 2/2 个关卡、[存储过程与事务]的 1/3 个关卡、[触发器]的 1/1 个关卡、[用户自定义函数]的 1/1 个关卡、[数据库设计与实现]的 2/3 个关卡、[数据库应用与开发(JAVA 篇)]的 6/7 个关卡、[存储管理(Storage Manager)]的 4/4 个关卡、[索引管理(Indexing)]的 3/3 个关卡。最后总成绩为 113 分 ＞100 分。

对于实验报告，首先对课程任务进行了概述，阐述了自己通关采取的逻辑思路流程；接着在本文对整个实验的任务实施过程进行分析；之后选取了 2 + 9 道有分析价值的题目进行详细的复盘和重现，力求展现做题时的思路和过程，包含 2 道数据库完整性定义问题（主要阐述注意事项）、5 道难度略高的 SQL.select 查询题目、1 道用户自定义函数题、1 道存储过程题、1 道触发器题、1 道数据库设计与实现题，其中在 select 查询题目小节之前加入了难度评估折线图；最后对全文进行了总结，阐述收获心得、学到的内容、自身的优势和不足。为避免过于复杂的描述，任务的实施过程如图 2-1 所示。

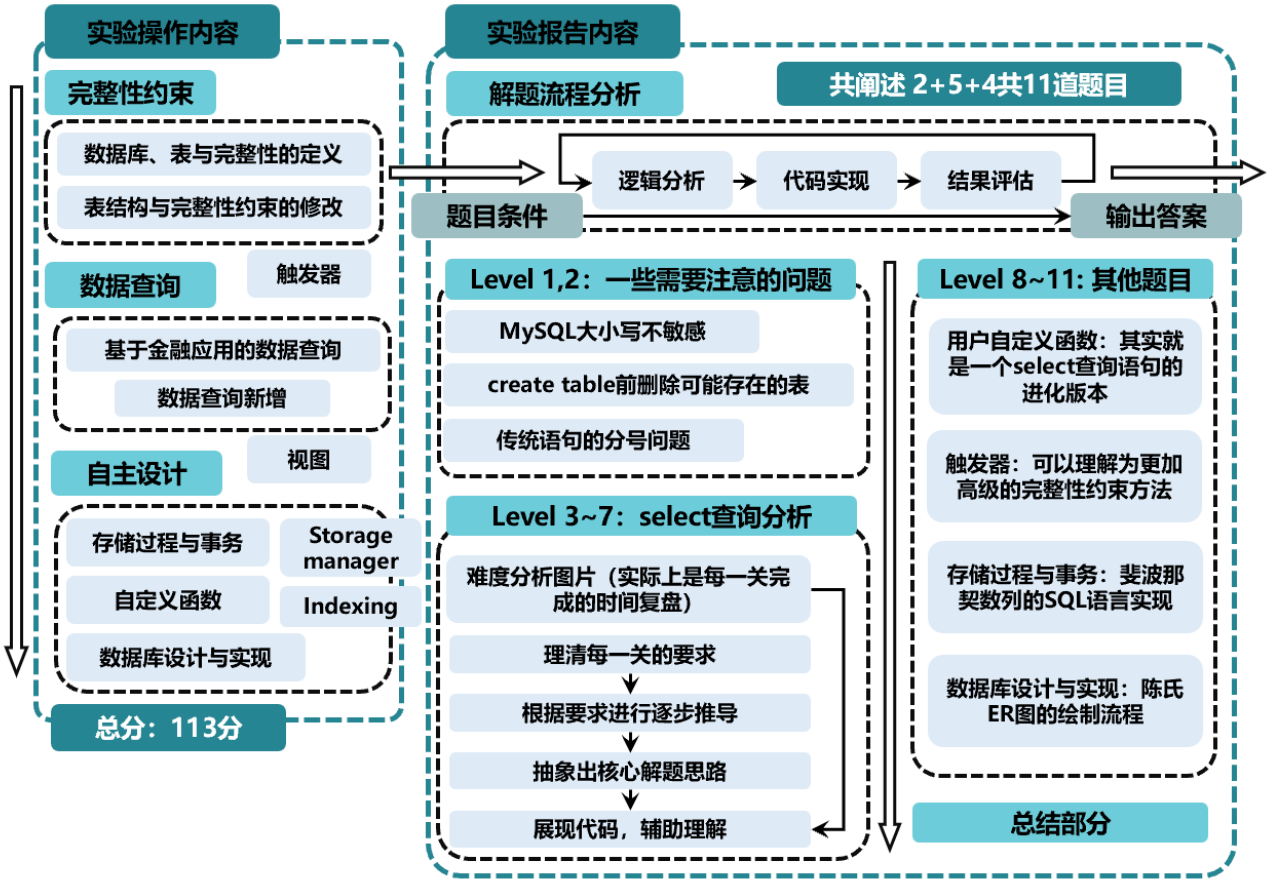

图 2-1 任务实施过程分析概况图

2.1 数据库、表与完整性约束的定义(Create)

梦开始的地方。这一部分的内容相对简单，但不要小瞧简单，简单意味着坚固。

这一块所学习的内容，将成为我们后面实现各种纷繁复杂的功能的基石，笔者认为有必要做一些记录，**主要目的是记录一些初学者阶段的易错点和注意点**，并进行了一些归纳总结。本章节已完成 6 道题目。

2.1.1 创建数据库

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

实现代码仅一行“**create database beijing2022;**”，但笔者足足用了 9 分钟半，原因是对全新环境的不熟悉。这里列出一些值得注意的点：

- 实验代码框架上方的“命令行”只是可选项而已。更推荐初学者直接在头歌平台的代码栏目编辑，因为在之前的计算机理论课程的实验学习中，对头歌平台的使用大多是代码行，这样更加符合我们的编程习惯。以上只是笔者的个人见解。
- MySQL 的关键字是**大小写不敏感**的，但平台采用 Linux 系统，数据库名、列名等用户定义的各种对象的名字是**大小写敏感**的！这一点实际上在平台上也有提示。课程的理论课 PPT 上，呈现的 MySQL 语句均为大写字母呈现，但符合人们日常表达，个人认为小写更加适合语义理解。
- *语句结束必须加上分号！！！*

2.1.2 创建表及表的主码约束

本关任务：在指定的数据库中创建一个表，并为表指定主码。

学习到的主要内容为主码的两种不同创建方式：

① 列定义里用 PRIMARY KEY 约束指定主码。实验最开始，笔者采用的是这种方法，原因是简单好记。即直接在某列的最后加上“primary key”即可。

② 表约束方法。表约束以关键词 **CONSTRAINT** 打头，后跟约束名，约束名可以省略，甚至连同关键词 CONSTRAINT 一同省略。这时，系统将自动为约束命名，一般可读性不强，会给将来可能的修改约束、删除约束等操作带来麻烦，总是给约束取一个有意义的名字是个好习惯。

只顾着创建表并添加主码，实则难以通关。本关卡还需要注意的点是：可能存在相同的表。因此一个良好的习惯是在 create table 前删除可能存在的表。在 create databases 时带上 phrase：if not exists。

以上均是初学者的易错点，特在此记录，可提醒自己和他人予以重视。

2.2 基于金融应用的数据查询(Select)

透过 SELECT 查询，将数据库中所需数据提取并可视化，成为数据库操作中至关重要的一环。本章节乃实验核心，亦为至关重要之部分，其所含关卡数量即为其重要性之体现。所涵盖 SQL 语言纯熟且至关关键，应用于金融场景，具综合性高、内容考察广泛、题型灵活多变、语句繁复且实践意义深远。因此，有必要花费较大篇幅来详细介绍。

本次实训所用数据库为某银行金融场景模拟数据库，详见附录中表格及其结构与字段说明。整个实验包含 19 个关卡，每个关卡都是在实际金融运作中具有重要意义的一环，涵盖了数据库操作的各个方面，从基础的数据提取到复杂的数据分析与可视化。通过完成这些关卡，学习者将对数据库操作有着更加深入的理解，并能够熟练运用 SQL 语言进行金融数据处理与管理。本章节共完成 19 个关卡。

此外，如图 2-2 所示，统计了在 19 关中，每一关的所用时间，目的是对整个章节的关卡难度进行一个粗略的评估，同时为实验本身提供一个微不足道的参考样本。可见，实验的难度总体上呈现出递增的趋势：从第 11 关开始，后面的难度远远大于前面；在 13 关出现一个峰值。注意这只是通过实验时间来直接推算难度的简单过程，因此参考价值略有限。

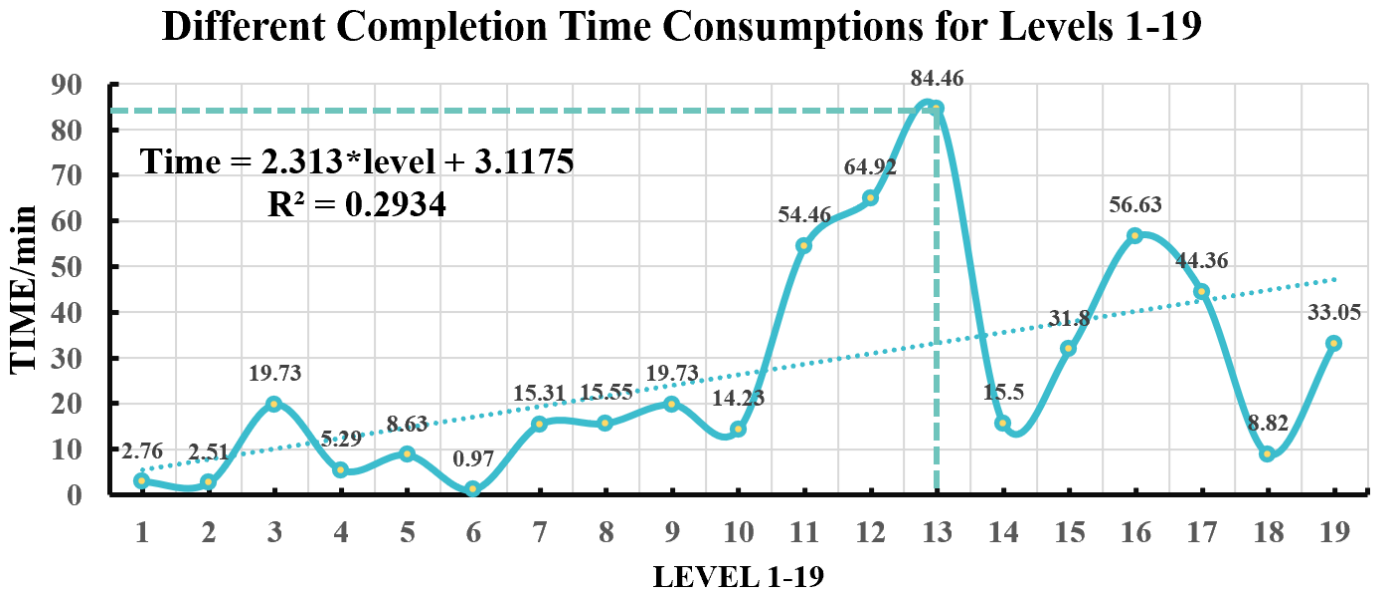

图 2-2 本章节内容中不同关卡所耗费的时间

2.2.1 黄姓用户持卡数量

本关卡的耗费时间为 54.46min，是第一个耗费时间超过 50 分钟的关卡。

原设计要求为：给出黄姓用户的编号、名称、办理的银行卡的数量(没有办卡的卡数量计为 0)，持卡数量命名为 number_of_cards，按办理银行卡数量降序输出，持卡数量相同的，依客户编号排序。

本题目思维方法较为线性，给出的要求也层层递进，非常适合基于要求进行实现的流程手段。因

此接下来逐条分析设计要求：

- **要求 1：编号、名称、办理的银行卡的数量：**

  实现方案：在 SELECT 语句中，我们首先要指定所需选取的属性。在这个情况下，我们需要选择编号（c_id）、名称（c_name）以及办理银行卡数量。但是，办理银行卡数量这一属性需要通过后续操作来提取，因此我们采用聚集函数的处理方法。此外，仅对 client 表进行操作是不足以达到我们的目标的；我们还需要对银行卡关系进行连接操作，以便提取相关信息。因此，在 SELECT 语句中，我们将需要包含连接操作，以确保我们能够从客户（client）表和银行卡关系表中检索到所需的属性。

- **要求 2：没有办卡的卡数量计为 0**

  实现方案：在处理用户可能未办理银行卡的情况时，**选择左连接（left join）是解决方案的关键。**左连接确保左侧（client 关系）的所有信息都被保留，即使在右侧（银行卡关系）中没有相应的记录。这样一来，**即使某些用户没有办理银行卡，其在客户表中的信息也会被保留**，并且办理银行卡数量将会显示为 NULL 或者其他指定的默认值。这一细节的考虑是本关卡中至关重要的一部分，因为在真实世界的数据操作中，经常会遇到部分用户未进行某项操作的情况，如未办理银行卡。因此，左连接的实现确保了数据操作的完整性和准确性。

- **要求 3：“黄姓用户”**

  实现方案：考察 WHERE 子句中字符串选择功能时，需要特别注意处理“黄姓用户”这样的需求，即选择以“黄”字开头的字符串，其后可能跟随其他类型的文字。在 MySQL 语法中，我们可以使用‘黄%’来表示这种含义。此外，需要注意字符串判断操作语句应该使用“LIKE”而不是“=”。这是一个容易出错的地方，因为在处理字符串匹配时，使用“=”会导致严格的匹配，而使用“LIKE”则更为灵活，可以进行模糊匹配。因此，在筛选“黄姓用户”的时候，应该使用类似于“WHERE c_name LIKE '黄%'”这样的语句来实现。

- **要求 4：按办理银行卡数量降序输出,持卡数量相同的,依客户编号排序**

  实现方案： 这一步就很简单了，在整个查询语句之后添上相应的排序语句即可。

代码实现如 SQL_coding1 所示，这里对每一行的核心思想都添加了注释，用双横杠前缀和斜体语言，和正式代码加以区分。

SQL_coding1: Number of cards held by users with the surname Huang

```
1  SELECT c_id, c_name, COUNT(b_number) AS number_of_cards
2  -- Join the client table with the bank_card table based on client ID
3  -- This left join ensures that all clients are included in the result even if they do not have any.
4  FROM client
```

```
5  LEFT JOIN bank_card ON c_id = b_c_id
6  -- Filter clients whose name starts with '黄'
7  -- The % sign is a wildcard character representing any number of characters.
8  WHERE c_name LIKE '黄%'
9  GROUP BY c_id
10 -- Sort the results by client ID
11 -- This ORDER BY clause sorts the results by client ID in ascending order.
12 ORDER BY c_id;
```

2.2.2 客户总资产

原设计要求为：综合客户表(client)、资产表(property)、理财产品表(finances_product)、 保险表(insurance)、基金表(fund)和投资资产表(property)， 列出所有客户的编号、名称和总资产，总资产命名为 total_property。 总资产为储蓄卡余额，投资总额，投资总收益的和，再扣除信用卡透支的金额 --(信用卡余额即为透支金额)。客户总资产包括被冻结的资产。

这道题目要求综合多个表格的信息，计算出每个客户的总资产。具体步骤如下：

连接表格：首先，需要将客户表（client）与资产表（property）、理财产品表（finances_product）、保险表（insurance）、基金表（fund）以及投资资产表（property，注意这里与客户表不同别名）进行连接。这可以通过使用左连接（LEFT JOIN）实现，以确保即使某些客户没有相关记录，也能够包含在结果中。这一部分的操作步骤如 SQL_coding2.1 所示。需要注意的是，这里较为创新地在求和语句里使用了 bool 表达式，即 true = 1 而 false = 0，用这种方法，就可以求出每个客户的**投资总收益**的和（也即 fund 类型收益、insurance 类型收益和 finance_product 类型收益）

SQL_coding2.1: Sum of total investment returns per client

```
1  select c_id, c_name,
2  SUM(IFNULL(p_amount * pro_quantity * (pro_type = 1), 0) + IFNULL(i_amount * pro_quantity
   *(pro_type = 2), 0) + IFNULL(f_amount * pro_quantity * (pro_type = 3), 0)
3  ) as t
4  from client
5  LEFT JOIN property ON client.c_id = property.pro_c_id
6  LEFT JOIN finances_product ON property.pro_pif_id = finances_product.p_id
7  LEFT JOIN insurance ON property.pro_pif_id = insurance.i_id
8  LEFT JOIN fund ON property.pro_pif_id = fund.f_id
9  group by c_id, c_name
```

计算总资产：在连接后的结果集中，需要计算每个客户的总资产。总资产包括储蓄卡余额、投资总额、投资总收益和信用卡透支金额。储蓄卡余额、投资总额和投资总收益可以通过对资产表中的相关字段进行求和来计算；而信用卡透支金额则需要单独计算，因为它是信用卡余额的负值。**刚刚已经**

计算出了投资总收益，这一步则还需要在 SQL_coding2.1 中得出的结果（设为表格 temp）再对各资产类型的表格再次进行左连接操作，以得到储蓄卡余额、投资总额和信用卡透支金额。这里仅以投资收益为例展现连接副表格的内容。

SQL_coding2.2: Linking the contents of the sub-tables

```
1  select c_id, c_name, ifnull(sum(pro_income), 0) as tt
2  from client
3  LEFT JOIN property ON client.c_id = property.pro_c_id
4  group by c_id, c_name
```

整合结果：最后，将每个客户的编号、名称以及计算得到的总资产整合在一起，形成最终的结果。**注意信用卡的金额总和的符号为负数**。在实现过程中，需要注意使用 IFNULL 函数来处理可能的空值，以及合适地使用别名来区分连接的不同表格和计算出的不同字段。此外，要确保使用合适的聚合函数对结果进行正确的汇总，以获得每个客户的正确总资产。

解决问题的流程抽象如图 2-3 所示。

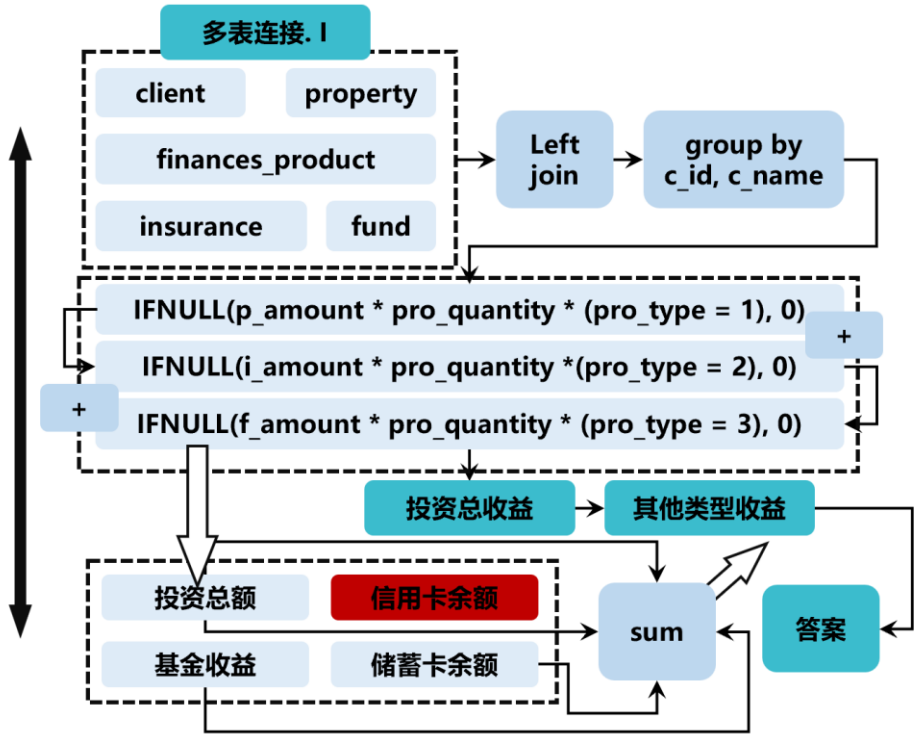

图 2-3 客户总资产的解决问题概况

2.2.3 Level 16 持有完全相同基金组合的客户

题目要求为：查询持有相同基金组合的客户对，如编号为 A 的客户持有的基金，编号为 B 的客户也持有，反过来，编号为 B 的客户持有的基金，编号为 A 的客户也持有，则(A,B)即为持有相同基金组合的二元组，请列出这样的客户对。为避免过多的重复，如果(1,2)为满足条件的元组，则不必显

示(2,1)，即只显示 编号小者在前的那一对，这一组客户编号分别命名为 c_id1,c_id2。

在解决这个问题时，最初可能会遇到一些挑战。为了解决这个问题，我们首先尝试将其简化为一个更易处理的形式。我们的目标是找到持有完全相同基金组合的客户对。为了达到这个目标，我们需要先选择一个特定的客户，称之为 c1，并尝试找到与 c1 持有完全相同基金组合的其他客户，称之为 c2。如果我们能够成功地找到这样的 c2，那么我们有可能找到所有符合条件的客户对。

接下来，我们需要对问题进行进一步分析。

首先，我们需要从现有的数据中提取必要的信息，并将其组织成一个临时表格，命名为 tb。考虑到问题涉及到客户和基金之间的关系，我们需要基于 property 和 fund 表格来构建 tb。在构建 tb 的过程中，我们只关注两个重要属性：客户 id 和基金 id。因此，我们将 property 表格和 fund 表格按照属性 pro_pif_id 和 f_id 进行等值连接，并且筛选出 pro_type = 3 的记录，表示这些资产是基金类。这样，我们就得到了一个包含了客户持有的基金信息的临时表格 tb。tb 的建立如 SQL_coding3.1 所示。

SQL_coding3.1: Establishment of tb

```
1  WITH tb AS (
2        SELECT pro_c_id, f_id
3        FROM property, fund
4        WHERE pro_pif_id = f_id AND pro_type = 3
5  )
```

对于简化版的表格 tb，我们可以将其抽象为一个包含两个列的数据结构，分别是客户 id（pro_c_id）和基金 id（f_id）。这个表格记录了每个客户持有的基金情况。

在固定用户 c1 的情况下，我们需要思考如何查询到另一个客户 c2，使得 c1 与 c2 的基金组合完全一致。

首先，对于 tb 中的任意一个客户 ci，我们需要进行扫描和筛选。为了定位到可能符合条件的客户，我们可以按照 pro_c_id 进行分组，并考察每一组的基金数量。为此，我们使用了 HAVING 语句来**筛选掉那些基金数量不符合要求的客户组**。具体地，我们使用 HAVING COUNT(*) = (SELECT COUNT(*) FROM tb WHERE pro_c_id = c1)这样的条件来保证了基金数量与 c1 相同。

然而，仅仅保证基金数量相同还不足以满足要求。我们还需要确保 ci 的所有基金 id 都出现在 c1 的基金 id 集合中。为了实现这一点，我们在 WHERE 语句中添加了条件 f_id IN (SELECT f_id FROM tb WHERE pro_c_id = c1)，这样就**保证了 ci 的基金 id 在 c1 的基金 id 集合中存在**。

同时，我们还需要排除 c1 与自己相同的情况，因此需要添加一个条件 AND ci != c1。

综合起来，我们可以使用 SQL.coding3.2 所呈现的思路来实现这一逻辑（注意这里 c1 是 p1.pro_c_id，与当时做题采用的命名方法有些出入，使用 c1 只是助于理解同时节省不必要的篇幅）：

SQL_coding3.2: Choice from tb

```
SELECT pro_c_id
    FROM tb
    WHERE f_id IN (
    SELECT f_id FROM tb WHERE pro_c_id = p1.pro_c_id
    ) AND pro_c_id != p1.pro_c_id
    GROUP BY pro_c_id
    HAVING COUNT(*) = (SELECT COUNT(*) FROM tb WHERE pro_c_id = p1.pro_c_id)
```

然而，仅仅依赖现有的 SELECT 语句逻辑可能会导致问题。这是因为该语句首先使用 WHERE 子句对元组进行筛选，接着使用 GROUP BY 子句进行分组，最后再通过 HAVING 子句筛选掉不符合条件的组。**也就是说，HAVING 子句所筛选的是已经经过 WHERE 子句一轮筛选的元组**。这种筛选流程可能会导致问题：例如，假设客户 c1 的基金组合为 001, 002, 003，而客户 c2 的基金组合为 001, 002, 003, 004。在第一次 WHERE 子句执行后，客户 c2 的基金组合将被筛选为 001, 002, 003，而基金号码 004 将因为不在 c1 的基金组合中而被“IN”子句筛选掉。因此，如果仅仅采用现有的查询逻辑，就可能导致客户 c2 的基金组合与客户 c1 完全一致的情况被错误地筛选出来。实际上，这一步骤只能选出客户 c1 的基金组合为客户 c2 的基金组合的子集的情况，而无法确保完全一致。

为了解决上述问题，我们需要引入另一个辅助表格。这个辅助表格不会执行先前的 WHERE 子句筛选操作，**而是仅执行后续的 GROUP 分组操作和 HAVING 选择**。这样一来，这个辅助表格里筛选出的就是基金数目与 c1 的基金数目相同的客户的客户号，但辅助表格不能保证其内部组合也完全相同。因此，我们需要将这个辅助表格与刚刚的关键逻辑表格进行内连接操作，类似于取交集。这样我们就能构建出一个“基金组合是 c1 客户的超集”且“基金组合数量与 c1 客户一致”的客户集合。根据集合论的原理，**如果集合 A 是集合 B 的子集，且集合 B 的元素等于集合 A，那么 A = B**，这是显而易见的。

所以，通过这两个表格的**内连接**操作，我们可以得到与 c1 客户的基金组合完全相同的客户。这一步骤能够确保我们找到的客户 c2 与 c1 持有完全相同的基金组合。

注意，现在只完成了简化版问题：与客户 c1 持有完全相同基金组合的客户。现在还需要解决问题本身，即查找所有基金组合完全相同的客户对。毫无疑问，这还需要在刚刚完成的问题的基础上再叠上一层 select 嵌套。可以考虑将两个 property 表格进行内连接操作，类似于 for 循环的内外层，对于第一个 property 表格，记作 p1，则对每个 p1.pro_c_id，去枚举 p2.pro_c_id，这里 p1.pro_c_id 就是 c1，p2.pro_c_id 就是 c2。对每一个 p1.pro_c_id 都执行一次刚刚的查询，那么查询结果就是与他的基金组合相同的客户代号，接下来只需要判断 p2.pro_c_id 是否存在于刚刚查询的结果即可。注意这里 c1 需要小于 c2，因此加上约束条件 p1.pro_c_id < p2.pro_c_id 即可。

方法总结：在解决查询持有相同基金组合的客户对的问题时，遇到了一些挑战。最初的查询逻辑可能会导致客户对的基金组合并不完全一致，**因为 WHERE 子句的筛选可能会造成部分信息的丢失**。为了解决这个问题，我们引入了一个辅助表格，该表格**只进行了后续的 GROUP 分组和 HAVING 选择操作**，筛选出了**基金数量与参考客户相同**的客户号。然后，我们将这个辅助表格与关键逻辑表格进行内连接操作，以确保客户的基金组合是参考客户基金组合的**超集**，并且基金**数量也相同**。通过这个步骤，我们可以找到与参考客户持有完全相同基金组合的客户。这样的处理方式避免了之前的问题，确保了查询结果的准确性和完整性。

这个问题的解决过程较为复杂，其解决方案的流程概况如图 2-4 所示。

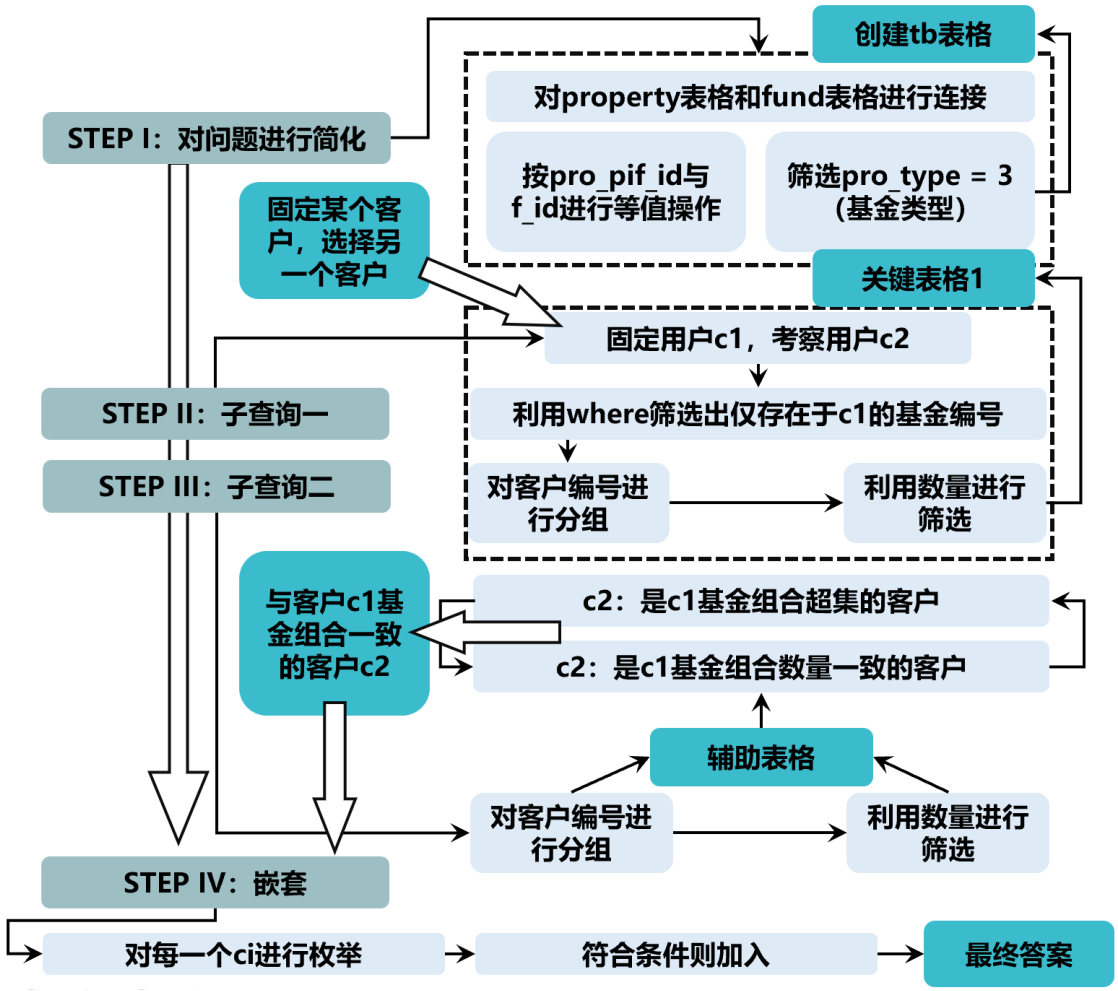

图 2-4 持有完全相同基金组合的客户的解决方案概况

该问题可以算得上是 SQL 语言查询关卡章节一的 19 关里，综合性和难度最大的一道题目。笔者当时花费了很久的时间来构想这个方法。在本文的撰写中，也花费了很长时间来重新研究本问题。这个方法的主要不足之处在于效率和复杂性方面。引入了辅助表格和内连接操作增加了查询的复杂度。这意味着查询的执行时间可能会变长，尤其是当数据量较大时。虽然这种方法可以解决问题，但它的复杂性和可能带来的性能问题需要仔细考虑。

2.2.4 至少有一张信用卡余额超过 5000 元的客户

本题目是本章节里笔者选择讲解的最后一道题目。虽然这题是第 18 关，但相比刚刚的几关来讲已经是比较简单的类型了；这里介绍这道题的目的主要是，它是我在验收过程中为助教讲解的题目。因此在实验报告进行一个汇报和复盘。

题目要求：查询至少有一张信用卡余额超过 5000 元的客户编号，以及该客户持有的信用卡总余额，总余额命名为 credit_card_amount。查询结果依客户编号排序。本题的要求也是极其线性的，因此可以将每条要求列出并逐步求解，这里也采用与第一个被介绍的关卡所相同的分析方法。

- **要求 1：至少有一张信用卡余额超过 5000 元的客户编号**

  实现方案：由于本问题涉及信用卡相关内容，因此涉及到银行卡表格的操作。本次查询的主要对象是名为 bank_card（临时别名为 b1）的表格。为了确保仅考虑信用卡，首先使用 where 子句对当前 select 语句进行限定，即 b1.b_type = '信用卡'。随后的目标是筛选出至少拥有一张信用卡余额超过 5000 元的客户，为此可以使用 exists 语句。exists 语句本质上是一个布尔判断，如果其内部的查询返回非空结果，则外层的元组将被选择。这种操作在当前语境下非常适用。

  在 exists 语句内部，我们建立了一个子查询（基于 bank_card 表格，命名为 b2），设置其内部查询的客户号等于外部查询的客户号，从而确保内部查询对象的正确性。我们进一步限定内部查询的类型为信用卡，并且要求其余额大于 5000 元。

- **要求 2：持有的信用卡总余额，命名 credit_card_amount**

  实现方案：这一步其实非常简单了，只需要基于客户编号进行分组，利用 sum 聚集函数求和。结果需要排序，也即加入 order by b1.b_c_id 即可。

2.3 数据查询 (select) – 新增

该部分是基于金融应用的数据查询章节的扩展内容，旨在进一步加深对金融数据查询的理解，是上个章节所学内容的进一步加深和拓展。本章节的难度和综合性将得到显著提升，具体表现在更为复杂的表结构、更严格的查询条件以及实际操作中更为复杂的实现方法，如多层循环嵌套等。这种评价源自于笔者的同学和室友，他们认为这一部分的挑战性较高。然而，由于时间分配原因，笔者未能完成该章节的所有任务，只完成了第一个关卡。因此，本文仅讨论了第一个关卡的解决方案，而未对后续关卡进行详细讨论。

此外，按老师的建议，依旧以折线图方式列出了本章节六个不同关卡的完成时间图，这数据源于笔者的其他班级的同学。如图 2-5 所示。在本章节中，各个关卡的难度呈现了逐渐增加的趋势，但是需要注意的是，由于只选择了两位同学的数据，并且这两位同学的完成时间略有不同，因此对于参考价值的影响可能较小。

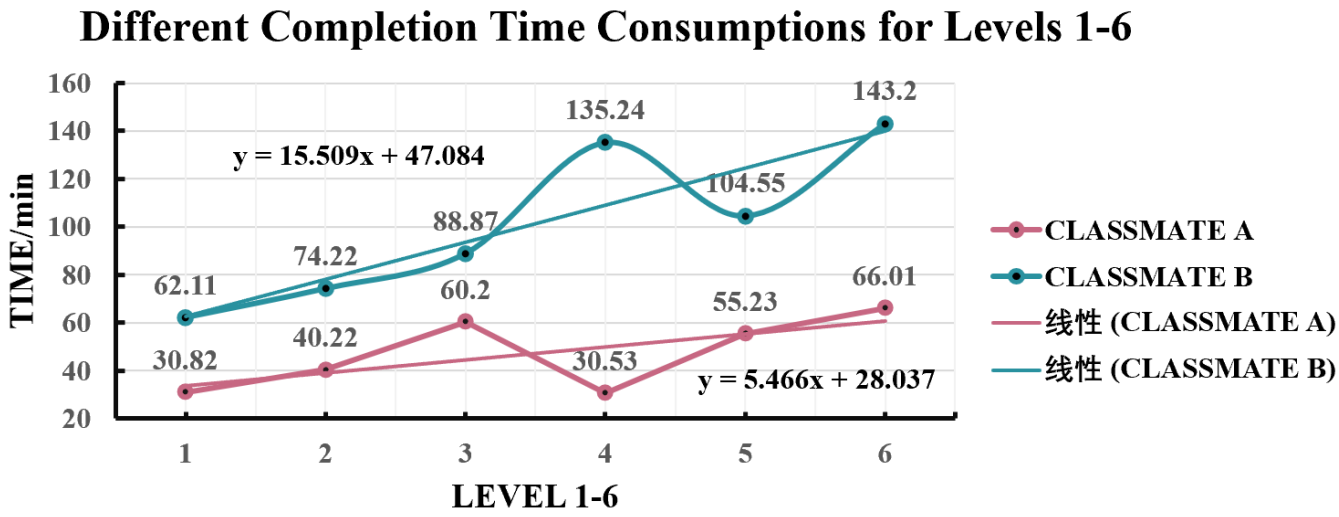

图 2-5 第三章节的不同关卡通关时间

2.3.1 查询销售总额前三的理财产品

本关任务：查询销售总额前三的理财产品。

在这里，系统已经给出了提示，也即 rank()函数的利用，以下是头歌系统原话：-rank() over(partition by ...order by) partition by 用于给结果集分组，如果没有指定那么它把整个结果集作为一个分组. 例如：select name,age,score,rank() over(partition by age order by score desc) as rank_num from players。

因此，**本关卡必然使用 rank()函数**，且需要用到该函数的 partition 约束，因为题目要求按年来分组。

同时，按头歌给出的通关条件，进行归纳总结。本关卡的详细要求为：

- 查询 2010 年和 2011 年这两年每年销售总额前 3 名（如果有并列排名，则后续排名号跳过之前的并列排名个数，例如 1、1、3）的统计年份（pyear）、销售总额排名值(rk)、理财产品编号(p_id)、销售总额(sumamount)。
- 按照年份升序排列，同一年份按照销售总额的排名值升序排列，如遇到并列排名则按照理财产品编号升序排列;
- 属性显示：统计年份（pyear）、销售总额排名值(rk)、理财产品编号(p_id)、销售总额(sumamount)
  结果显示顺序：先按照统计年份（pyear）升序排,同一年份按照销售总额排名值（rk）升序排,同一排名值的按照理财产品编号（p_id ）升序排。

接下来对本题进行分析。首先必须留意的是，这里的“2010 年，2011 年”代指的是表格 **property 里的购入时间 pro_purchase_time，而不是理财产品 finances_product(理财产品表)里的 p_year**，然而题目最后又要求统计年份的数据命名为 pyear，这是一个极其容易混淆的点，个人认为是题目没有说清楚，没有把要求讲明白导致的，如果老师看到这里，可以进行更改。**这里 pro_purchase_time 是一**

个时间类型的变量，需要利用 year()函数将其年份提取出来。

根据题目的要求，我们首先需要将属性表（property）和理财产品表（financial_product）进行等值连接或自然连接。连接操作的关键在于使用连接条件，即属性表中的 pro_pif_id 与理财产品表中的 p_id 相等，并且需要添加一个约束条件，即**属性表中的 pro_type 等于 1**，以确保我们只获取到理财产品相关的信息。

连接完成后，我们将从连接后的表格中提取所需的信息。这些信息包括购入时间（p_year）、产品代号（p_id）和产品销售额（s）。其中，产品销售额（s）是一个新的属性，它的值是通过属性表中的 **pro_quantity 与理财产品表中的 p_amount 相乘**得到的。这个新属性 s 对我们后续的分析和查询非常重要，因为它表示了每个理财产品的销售额。

在 where 语句里，最后还需要加入这样一个限制条件：即 **pro_purchase_time 必须是 2010 年和 2011 年的年份。**这样才能保证理财产品的购买时间是正确的。

对于临时表格 tb 的创建如 SQL_coding4.1 所示。

SQL_coding4.1: Establishment of tb

```
1  with tb as (
2        select year(pro_purchase_time) as p_year, p_id, pro_quantity * p_amount as s
3        from property, finances_product
4        where pro_pif_id = p_id and pro_type = 1
5        and year(pro_purchase_time) in (2010, 2011)
6  )
```

有了 tb 的辅助之后，我们理解题目的思路受限变小了。接下来将通过 tb 构建一个含有排名的表格 temp1，至于题目需要的“排名前三”，实际上只需要对 temp1 再进行一次全选择，在 where 语句里加入修饰词**“rk <= 3”**即可。因此我们只需要着眼于如何实现排名和销售总额即可。

针对 tb 表格的情况，需要考虑到同一个代码编号的理财产品可能会有多次购买，而每次购买的数量也可能不同，因此仅仅依靠单个属性 s 并不能准确反映出全部的销售总额。为了得到每个理财产品的完整销售总额，**我们需要利用 GROUP BY 语句按照 p_id 进行分组**，并利用**聚集函数 SUM()**对每个理财产品的销售额进行求和运算。

然而，这种方法在实际应用中存在一个问题：题目要求输出的内容除了销售总额外还包括购入时间（p_year），但是使用聚集函数对 p_id 进行分组后，就**无法直接输出购入时间（p_year）**，因为在分组的过程中，**p_year 并没有作为分组属性参与到查询结果中。这可能导致查询错误或不完整的结果**。

为了解决这个问题，我们需要调整 GROUP BY 分组的方式，在分组过程中同时考虑 p_year 和 p_id，即将这两个属性作为组合键进行分组。这样一来，我们就可以确保每个理财产品的销售额和购入时间都能正确地与结果关联，从而得到准确且完整的查询结果。

至于分组的排名结果，实际上题目已经提示得很清楚了，只需要在查询结果里添加一列 rk，采用 rank()函数，**写入语句 rank() over(partition by p_year order by sum(s) desc) as rk 即可。**

完成这一段查询之后，在外层按之前的描述，嵌套一层全查询，并设置条件“rk <= 3”，即可完成任务。本关卡代码难度不大，关键在于对问题的充分剖析和抽象。对应代码如 SQL_coding4.2 所示，对应的关键逻辑和刚刚提及的需要注意的点已经加粗显示。

SQL_coding4.1: Enquire about the top three financial products in terms of total sales volume

```
1   with tb as (
2         select year(pro_purchase_time) as p_year, p_id, pro_quantity * p_amount as s
3         from property, finances_product
4         where pro_pif_id = p_id and pro_type = 1
5         and year(pro_purchase_time) in (2010, 2011))
6   select *
7   from (
8         select p_year as pyear,
9         rank() over(partition by p_year order by sum(s) desc) as rk,
10        p_id, sum(s) as sumamount
11        from tb
12        group by p_year, p_id
13  ) as temp1
14  where rk <= 3
```

解决问题的流程概况如图 2-6 所示。

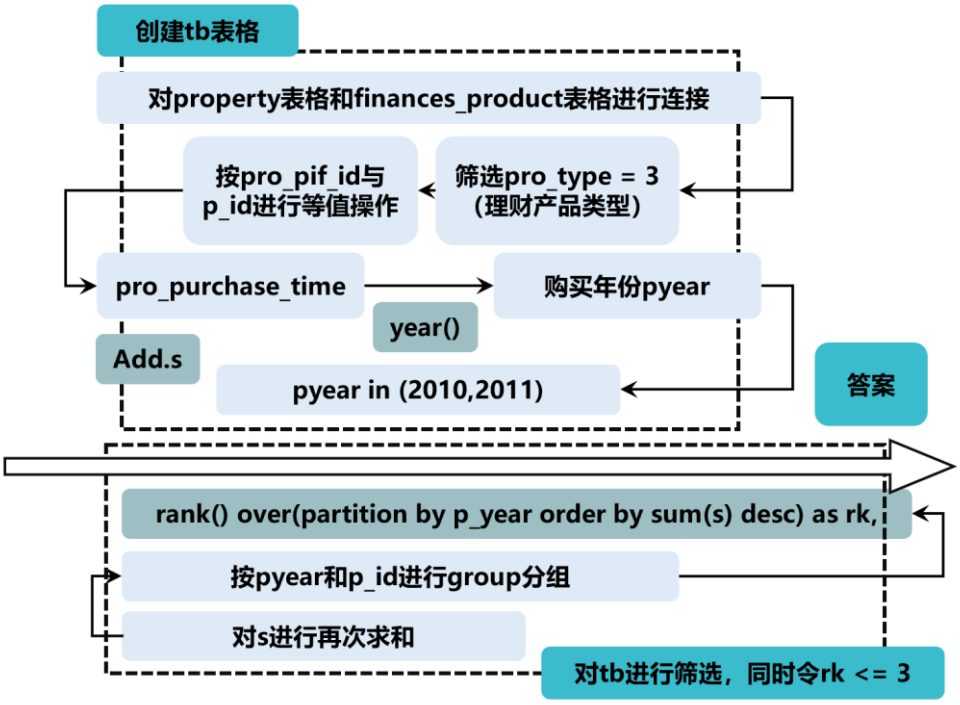

图 2-6 查询销售总额前三的理财产品的解决流程概况

2.4 用户自定义函数（创建函数并在语句中使用它）

MySQL 的函数定义与存储过程的定义一样，在定义函数之前要用“delimiter 界符”语句指定函数定义的结束界符，并在函数定义后，再次使用“delimiter 界符”语句恢复 MySQL 语句的界符(分号)。本关任务是编写一个依据客户编号计算其在本金融机构的存储总额的函数,并在 SELECT 语句使用这个函数。也就是需要我们利用 SQL 语句实现一个函数，并且马上对该函数进行实践运用。本章节仅这样一道题目，因此就不分小节来阐述了。

编码栏目里，函数的基本框架其实已经给出，实际上需要我们实现的只有函数体内部的具体功能。函数需要根据客户编号来计算其在本金融机构的存储总额，其函数参数就是客户的编号。因此这道题**实际上只是一个披着函数定义的外衣的查询语句题目**，其难度也不如之前的关卡高，关键在于克服畏惧情绪。接下来仍旧按照查询语句的分析方法来分析这道题。

- **用 create function 语句创建符合要求的函数**

函数的返回值为客户在本金融机构的存储总额，因此函数体内只需要一个 return 即可，return 的内部可以嵌套一个 select 查询语句。这查询语句需要查询的信息来自于银行卡，因此**选择 bank_card 作为查询表**；考虑到传入函数**参数为 client_id**，因此需要筛选的**条件即可设置为 b_c_id = client_id**；其类型需要设置为“储蓄卡”；由于查询的结果需要求和，即利用聚集函数按 **b_c_id 进行分组，对 b_balance 也即额度进行求和**即可；我们**返回的值即为 sum(b_balance)**。

函数体代码如 SQL_coding5 所示。

SQL_coding5: A function to calculate total balance of a customer based on the customer ID

```
1  delimiter $$
2  create function get_deposit(client_id int)
3  returns numeric(10,2)
4  begin
5        return (
6              select sum(b_balance)
7              from bank_card
8              where b_type = "储蓄卡"
9              group by b_c_id
10             having client_id = b_c_id
11       ); //注意分号!!!!!!!!!
12 end$$
13 delimiter ;
```

- **利用创建的函数进行查询操作**

接下来还需要利用创建的函数，仅用一条 SQL 语句查询存款总额在 100 万(含)以上的客户身份证

号，姓名和存款总额(total_deposit)，结果依存储总额从高到低排序。

有了上一步的函数，实际上也很简单了。接下来我们只需要对 client 原始表格进行操作，已经不再需要进行等值连接等操作，只需要传入参数 c_id 即可得到这位客户的储蓄卡总额。首先按题目要求进行选择，设置 **select c_id_card, c_name, get_deposit(c_id) as total_deposit** 为查询内容，其查询表格为 client；接着按题目给出的选择条件，设置约束条件为 **where get_deposit(c_id) >= 1000000** 即可达到总额高于一百万的目的；最后只需要加入排序要求 **order by get_deposit(c_id) desc** 即可。这代码层次简单，就不展示了。

此外，还需要注意的一个易错点是：***函数体内部逻辑完成后，必须要加一个分号！***

2.5 存储过程与事务

存储过程（Stored Procedure）是一种在数据库中存储复杂程序，以便外部程序调用的一种数据库对象。存储过程是为了完成特定功能的 SQL 语句集，经编译创建并保存在数据库中，用户可通过指定存储过程的名字并给定参数（需要时）来调用执行。存储过程思想上很简单，就是数据库 SQL 语言层面的代码封装与重用。简单的说存储过程就是具有名字的一段代码，用来完成一个特定的功能。本章节完成了第一关的内容。

程序的三种基本结构是顺序、选择、循环。上述语句可以用来构建复杂的程序。接下来就第一关的斐波那契数列对存储过程进行分析。

2.5.1 使用流程控制语句的存储过程

在 create procedure sp_fibonacci(in m int)已经限定了我们插入的范围，也即斐波那契数列的前 m 项。观察测试集输出可知，这斐波那契数列的第一项为 0，第二项为 1，以此类推，后面为 1, 2, 3, 5, 8, 11…因此，这是一个初值设置为 0 的斐波那契数列。

可以通过定义一个递归存储过程来实现斐波拉契数列的计算。

接着定义递归存储过程 cal_fib。cal_fib 有 id, res, pre_res 三个参数，分别代表当前行号、前一行的结果以及前一行的上一行的结果。在递归开始时，3 个参数都置 0。

在递归过程中，我们需要对行号进行递增操作，以确保下一个行号符合要求。同时，我们需要更新 res，使其成为前一行结果 pre_res 与当前行结果 res 的和。需要注意的是，当 id < 2 时，表示我们处于斐波那契数列的前两项，此时 res 直接被置为 1。我们还需要更新 pre_res 为当前行结果 res。这一系列的赋值操作可以通过使用 select 语句进行实现，即可以使用表达式 select id + 1, if(id < 2, 1, res + pre_res), res。当行号已经不小于 m - 1 时将弹出递归的操作。

实现方案如 SQL_coding6 所示。

SQL_coding6: Fibonacci

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

2.6 触发器（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）

触发器是与某个表绑定的命名存储对象，与存储过程一样，它由一组语句组成，当这个表上发生某个操作(insert,delete,update)时，触发器被触发执行。触发器一般用于实现业务完整性规则。当 primary key,foreigh key, check 等约束都无法实现某个复杂的业务规则时，可以考虑用触发器来实现。本章节仅一道题目，因此不再分小节介绍。

本关任务：为表 property(资产表)编写一个触发器，以实现以下完整性业务规则：

- 如果 pro_type = 1, 则 pro_pif_id 只能引用 finances_product 表的 p_id；
- 如果 pro_type = 2, 则 pro_pif_id 只能引用 insurance 表的 i_id；
- 如果 pro_type = 3, 则 pro_pif_id 只能引用 fund 表的 f_id；
- pro_type 不接受(1,2,3)以外的值。
- 各投资品种一经销售，不会再改变；
- 也不需考虑 finances_product，insurance，fund 的业务规则(一经销售的理财、保险和基金产品信息会永久保存，不会被删除或修改，即使不再销售该类产品)。

接下来分步骤陈述实现方法：

- **触发器名称和定义：**

  触发器名为 before_property_inserted，在每次向 property 表插入新记录之前触发。触发器使用 BEFORE INSERT ON property 来指定在插入操作执行前执行触发器逻辑。

- **变量声明和初始化：**

pro_type_value 和 pro_pif_id_value 是用来存储新插入行的 pro_type 和 pro_pif_id 值的变量。error_message 是用来存储错误消息的变量。

- **条件检查和错误处理：**

  首先，检查 pro_type_value 是否超出允许的范围（即大于 3 或小于等于 0）。如果超出范围，则设置相应的错误消息。

  如果 pro_type_value 是 1，并且 pro_pif_id_value 不在 finances_product 表的 p_id 中，则设置错误消息，指示找不到相应的理财产品。

  如果 pro_type_value 是 2，并且 pro_pif_id_value 不在 insurance 表的 i_id 中，则设置错误消息，指示找不到相应的保险产品。

  如果 pro_type_value 是 3，并且 pro_pif_id_value 不在 fund 表的 f_id 中，则设置错误消息，指示找不到相应的基金产品。

- **异常抛出：**

  如果 error_message 不为空，则使用 signal 语句抛出 SQL 异常，错误码为 "45000"，并设置异常消息为 error_message，以中断当前的插入操作。

实现代码如 SQL_coding7 所示（未列出所有，只列出了触发器实现部分内容）。

SQL_coding7 Flip-flop

```
1  -- Implement business constraint rules for the investment table property - refer to the master code of different tables according to the investment category.
2  BEGIN
3      declare pro_type_value int default new.pro_type;
4      declare pro_pif_id_value int default new.pro_pif_id;
5      declare error_message varchar(50);
6      if pro_type_value > 3 or pro_type_value <= 0 then
7          set error_message = concat("pro_type ", pro_type_value, " is illegal!");
8      elseif pro_type_value = 1 and pro_pif_id_value not in (select p_id from finances_product) then
9          set error_message = concat("finances product #", pro_pif_id_value, " not found!");
10     elseif pro_type_value = 2 and pro_pif_id_value not in (select i_id from insurance) then
11         set error_message = concat("insurance #", pro_pif_id_value, " not found!");
12     elseif pro_type_value = 3 and pro_pif_id_value not in (select f_id from fund) then
13         set error_message = concat("fund #", pro_pif_id_value, " not found!");
14     end if;
15     if error_message is not null then
16         signal sqlstate "45000" set message_text = error_message;
17     end if;
18 END$$
```

2.7 数据库设计与实现

本实验要求掌握数据库设计方法与实现过程。数据库设计具体过程可以概括为在精准需求分析的基础上，设计概念模型，再到逻辑模型，最后进行物理模型的设计，以此完成数据库的建模工作。本章节完成了第一关、第三关的任务；这里选择阐述第一关。

2.7.1 从概念模型到 MySQL 实现

这是一个机票订票系统。现在需要根据逻辑结构，复现其数据库建立过程。系统考虑的实体参见附录。选择这题的原因是与课程内容紧密联系，笔者当时自己完成的时候，一直无法将自己的代码提交成功，在老师的协助下依旧难以提交，因此最后是靠老师的代码通关的。这一关不再讲解代码的流程，因为实际上需要写的内容都在实体定义里，只需要将其转化为代码语言就好；笔者在这里想呈现的是如何根据题目的约束条件来进行陈氏 ER 图的绘制。Crow's foot 模型 ER 图已经给出，这里只是作为参考，如图 2-7 所示。

先列出实体之间的联系：

- 每个航空公司都有一个母港(机场)，又叫基地。大的机场可能会是多家公司的基地，小型机场可能不是任何航空公司的基地。
- 每个航班属于一家航空公司，航空公司可以很多航班。
- 任何一驾民航飞机属于一家航空公司，航空公司可以有多驾飞机。
- 每驾飞机可以执飞多个航班，一个飞行航班由一架飞机执飞。
- 一个航班根据执飞机型可以售出若干机票。一张机票是某个特定航班的机票。

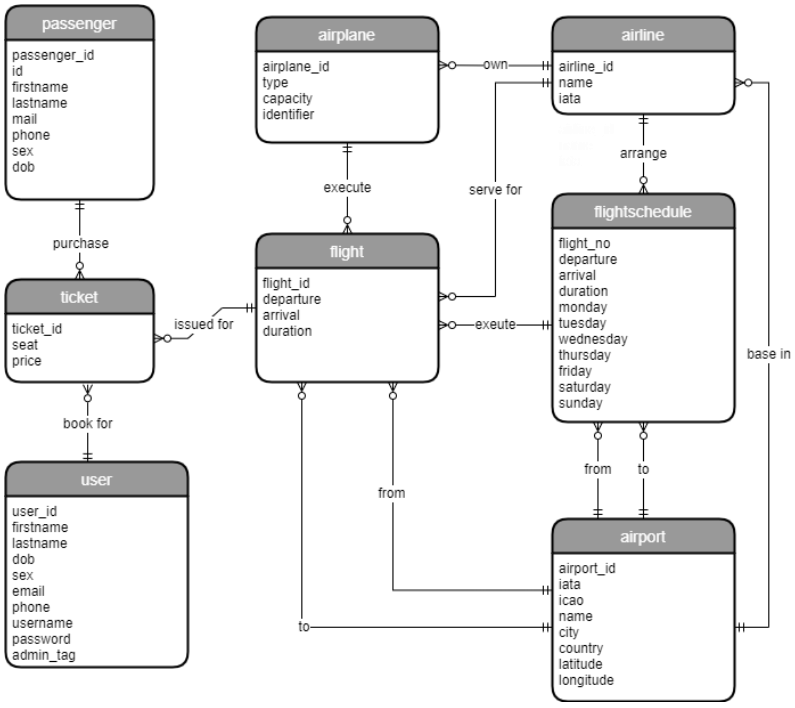

图 2-7 Crow's foot ER 图

- 用户可以多次订票，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。即机票信息不仅跟乘坐人有关，同时记录购买人信息(虽然两者有时是同一人)。为简单起见，订购时间没有考虑。
- 无论常规计划的航班，还是实际飞行航班，都是从某个机场出发，到达另一个机场。但一个机场可以是很多个航班的出发地，也是很多航班的到达地。

接下来分步骤阐明 ER 图的绘制流程。

每个航空公司都有一个母港(机场)，又叫基地。大的机场可能会是多家公司的基地，小型机场可能不是任何航空公司的基地：航空公司（airline）与机场（airport）将存在 n：1 的关系，关系名为 based in，表示机场是航空公司的基地。但难以体现“小型机场可能不是任何航空公司的基地”的语义约束。本步骤的绘制如图 2-8 所示。

图 2-8 ER 图绘制流程 1

每个航班属于一家航空公司，航空公司可以很多航班：航班将涉及两个关系，分别是航班常规调度表（flightschedule）与航班表（flight）；航班与这两个关系都是 1：n 类型，关系名分别为 arrange 和 serve for；本步骤的绘制如图 2-9 所示。

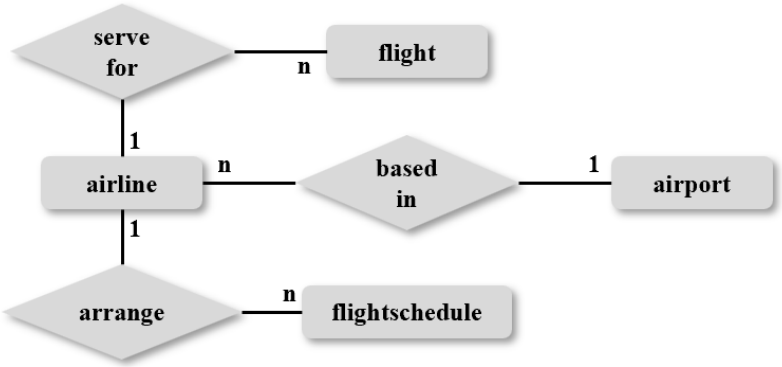

图 2-9 ER 图绘制流程 2

任何一驾民航飞机属于一家航空公司，航空公司可以有多驾飞机：加入民航飞机（airplane），可与航空公司（airline）构成一个 n：1 的关系，关系名为 own；本步骤的绘制如图 2-10 所示。

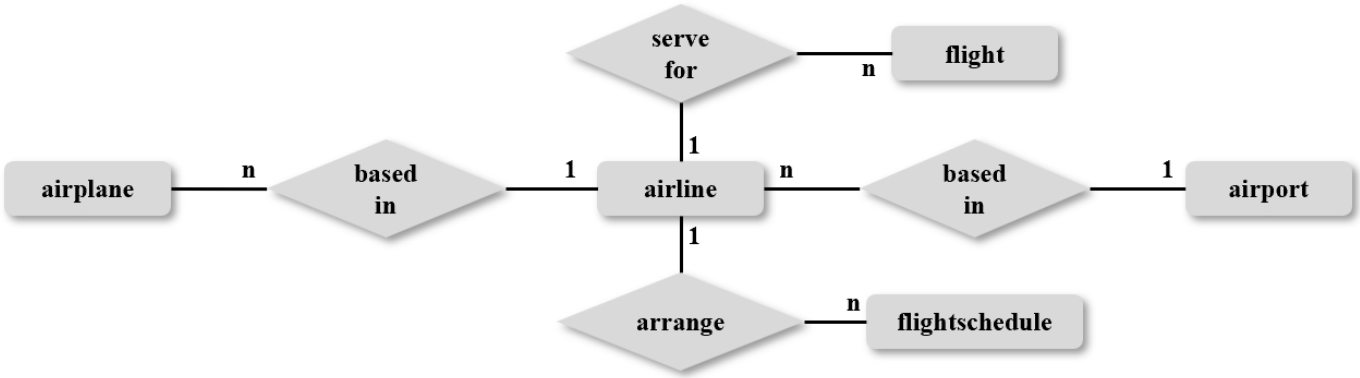

图 2-10 ER 图绘制流程 3

每驾飞机可以执飞多个航班，一个飞行航班由一架飞机执飞：在民航飞机（airplane）与航班（flight）之间加入一个新关系 execute，类型为 1：n；本步骤的绘制如图 2-11 所示。

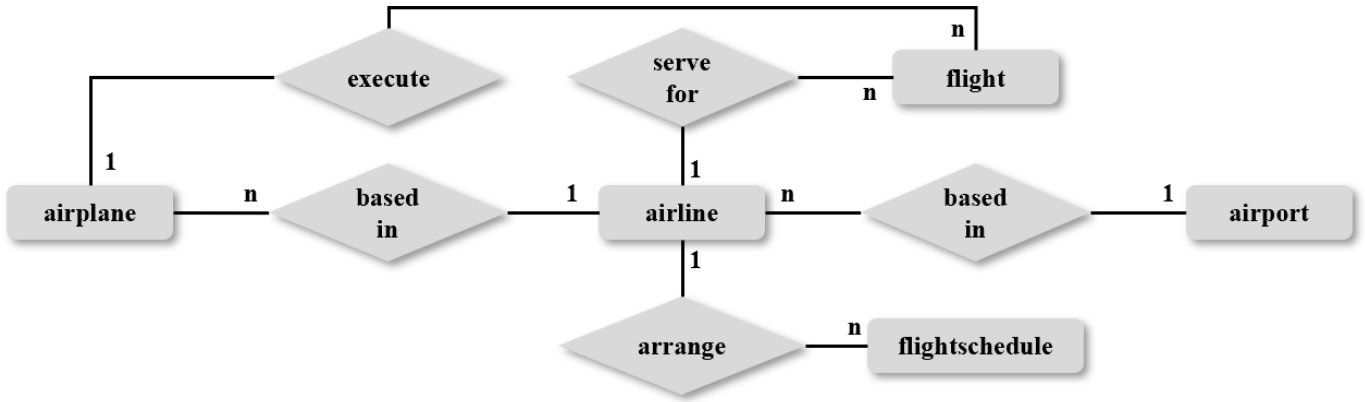

图 2-11 ER 图绘制流程 4

一个航班根据执飞机型可以售出若干机票。一张机票是某个特定航班的机票：加入机票关系（ticket），与航班（flight）构成 n：1 关系；关系名称为 issued for；步骤的绘制如图 2-12 所示。

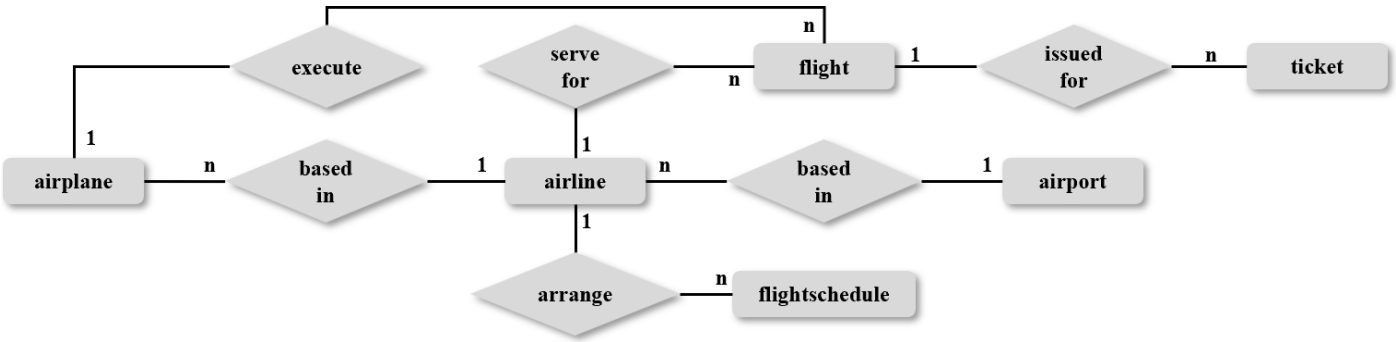

图 2-12 ER 图绘制流程 5

用户可以多次订票，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。即机票信息不仅跟乘坐人有关，同时记录购买人信息(虽然两者有时是同一人)。为简单起见，订购时间没有考虑：加入用户关系（user）和旅客关系（passager）；根据“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。即机票信息不仅跟乘坐人有关，同时记录购买人信息(虽然两者有时是同一人)”，机票（ticket）将同时与用户（user）和旅客（passager）都存在关系。这两个关系分别命名为 book for 和 purchase，其对应关系都是 1：n；步骤的绘制如图 2-13 所示。

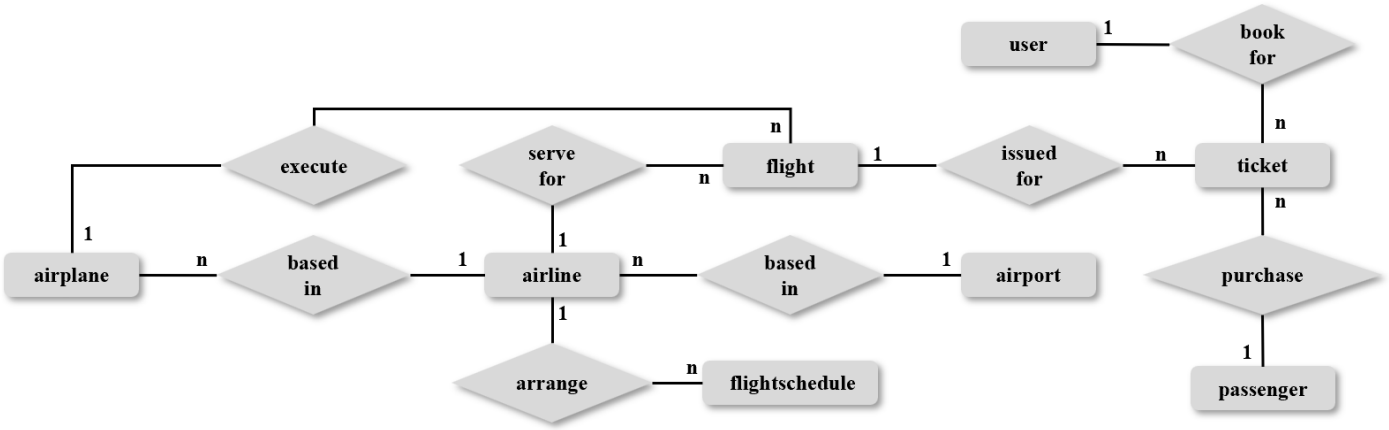

图 2-13 ER 图绘制流程 6

无论常规计划的航班，还是实际飞行航班，都是从某个机场出发，到达另一个机场。但一个机场可以是很多个航班的出发地，也是很多航班的到达地：flight、flightschedule 与 airport 之间都存在 from 关系和 in 关系，都为 n：1；步骤的绘制如图 2-14 所示。

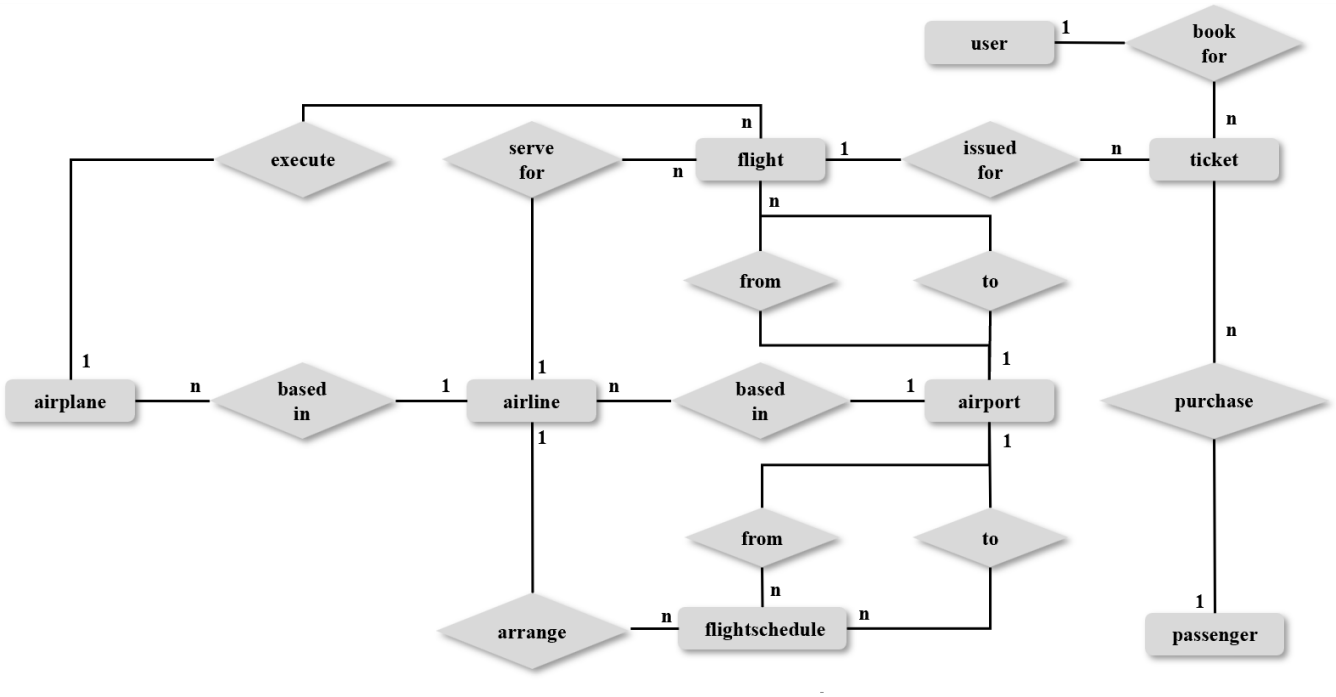

图 2-14 ER 图绘制流程 7

为了表示简便，绘制陈氏 ER 图时都没有给出关系和实体的内在属性，实际上这些内在属性均已经在题目里给出了，本章节给出的绘制流程便不再展示。选择这种阐述的原因是陈氏 ER 图与理论课教学内容高度相关，理论课也需要我们掌握这种从文字描述中提出抽象数据的能力；在绘制的过程中，复习了课堂的知识点的同时也对关系数据库有了更深的理解。

3. *Summary*

本次实验开始于 2024 年 4 月 20 日，结束于 2024 年 6 月 15 日，前后大约花费了 2 个月的时间。在实验过程中，完成了[数据库、表与完整性的定义]的 6/6 个关卡、[表结构与完整性约束的修改]的 4/4 个关卡、[基于金融应用的数据查询]的 19/19 个关卡、[数据查询新增]的 1/6 个关卡、[数据的插入、删除和修改]的 6/6 个关卡、[视图]的 2/2 个关卡、[存储过程与事务]的 1/3 个关卡、[触发器]的 1/1 个关卡、[用户自定义函数]的 1/1 个关卡、[数据库设计与实现]的 2/3 个关卡、[数据库应用与开发(JAVA 篇)]的 6/7 个关卡、[存储管理(Storage Manager)]的 4/4 个关卡、[索引管理(Indexing)]的 3/3 个关卡。最后总成绩为 113 分。在这个过程中，大大加强了我的代码能力，提高了自我素养，同时对于数据库有了系统的认识。这对我的实践能力有了很大的帮助，同时也与老师和同学们积极沟通与合作。

在本次实验中，有一些表现突出的地方：

- 较高的完成效率：实验中表现出了高效的工作速度，快速完成了任务。
- 行之有效的分析：通过采用了系统性的，科学的分析方法，成功完成实验所有内容。
- 联系课堂学习内容：能够将课堂学到的知识应用到实践中，提高了对课程内容的理解和运用能力。
- 多种多样的报告呈现手段：采取了图、流程、概况、折线图等多种呈现形式，体现了自己的理解。
- 与老师和同学们的沟通和交流：积极与老师和同学们交流，共同讨论问题并分享经验，促进了学习氛围的形成。

同样的，依旧存在一些问题：

- 功利性太强：可能出现了只追求完成任务而忽略了深入理解和探索的情况，缺乏对问题的全面思考。
- 缺乏灵活变通能力：在遇到难题或变化时，可能过于依赖已有的方法和思路，缺乏灵活的应变和创新解决方案的能力。
- 赶工痕迹明显：在最后的报告撰写中，认真程度不如之前。同时效率略有下降。

整体来看，我还是很顺利地完成了所有实验内容，学习到了应该收获的知识。在这个过程中，我也积极与老师同学们沟通，建立了深厚的友谊，加强了合作关系。这些以前未曾拥有的，如今都为我所用，都使得我受益良多。这是一次很有意义的实验！感谢您的阅读，再见！

付名扬

2024 年 6 月 16 日夜

附录

表 1 client(客户表)

字段名称	数据类型	约束	说明
c_id	INTEGER	PRIMARY KEY	客户编号
c_name	VARCHAR(100)	NOT NULL	客户名称
c_mail	CHAR(30)	UNIQUE	客户邮箱
c_id_card	CHAR(20)	UNIQUE NOT NULL	客户身份证
c_phone	CHAR(20)	UNIQUE NOT NULL	客户手机号
c_password	CHAR(20)	NOT NULL	客户登录密码

表 2 bank_card(银行卡)

字段名称	数据类型	约束	说明
b_number	CHAR(30)	PRIMARY KEY	银行卡号
b_type	CHAR(20)	无	银行卡类型(储蓄卡/信用卡)
b_c_id	INTEGER	NOT NULL FOREIGN KEY	所属客户编号,引用自 client 表的 c_id 字段。
b_balance	NUMERIC(10,2)	NOT NULL	余额,信用卡余额系指已透支的金额

表 3 finances_product(理财产品表)

字段名称	数据类型	约束	说明
p_name	VARCHAR(100)	NOT NULL	产品名称
p_id	INTEGER	PRIMARY KEY	产品编号
p_description	VARCHAR(4000)	无	产品描述
p_amount	INTEGER	无	购买金额
p_year	INTEGER	无	理财年限

表 4 insurance(保险表)

字段名称	数据类型	约束	说明
i_name	VARCHAR(100)	NOT NULL	保险名称
i_id	INTEGER	PRIMARY KEY	保险编号
i_amount	INTEGER	无	保险金额
i_person	CHAR(20)	无	适用人群
i_year	INTEGER	无	保险年限
i_project	VARCHAR(200)	无	保障项目

表 5 fund(基金表)

字段名称	数据类型	约束	说明
f_name	VARCHAR(100)	NOT NULL	基金名称
f_id	INTEGER	PRIMARY KEY	基金编号
f_type	CHAR(20)	无	基金类型
f_amount	INTEGER	无	基金金额
risk_level	CHAR(20)	NOT NULL	风险等级
f_manager	INTEGER	NOT NULL	基金管理者

表 6 property(资产表)

字段名称	数据类型	约束	说明
pro_id	INTEGER	PRIMARY KEY	资产编号
pro_c_id	INTEGER	NOT NULL	客户编号 FOREIGN KEY
pro_pif_id	INTEGER	NOT NULL	业务约束
pro_type	INTEGER	NOT NULL	商品类型:1 表示理财产品;2 表示保险;3 表示基金
pro_status	CHAR(20)	无	商品状态
pro_quantity	INTEGER	无	商品数量
pro_income	INTEGER	无	商品收益
pro_purchase_time	DATE	无	购买时间

数据库设计与实现-第一关-实体类型

1.用户(user) 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。为简单起见，两类用户合并，用 admin_tag 标记区分。用户的属性(包括业务约束)有： 用户编号: user_id int 主码,自动增加 名字: firstname varchar(50) 不可为空 姓氏: lastname varchar(50) 不可为空 生日: dob date 不可为空 性别: sex char(1) 不可为空 邮箱: email varchar(50) 联系电话: phone varchar(30) 用户名: username varchar(20) 不可空,不可有重 密码: password char(32) 不可空 管理员标志: admin_tag tinyint 缺省值 0(非管理员),不能空

2. 旅客(passenger) 用户登录系统不一定是替自己买票，所以用户和旅客信息是分开存储的。属性有：旅客编号: passenger_id int 自增，主码 证件号码: id char(18) 不可空 不可重 名字：firstname varchar(50) 不可空 姓氏: lastname varchar(50) 不可空 邮箱：mail varchar(50) 电话: phone varchar(20) 不可空 性别: sex char(1) 不可空 生日: dob date

3.机场(airport) 有以下属性： 编号: airport_id int 自增，主码 国际民航组织编码：iata char(3) 不可空，全球唯一 国际航运协会编码: icao char(4) 不可空，全球唯一 机场名称: name varchar(50) 不可空，

普通索引 所在城市: city varchar(50) 所在国家: country varchar(50) 纬度: latitude decimal(11,8) 经度: longitude decimal(11,8) 全球每个机场都有唯一 IATA 编码和 ICAO 编码，IATA 为 3 个字符，ICAO 为 4 个字符。例如首都机场的(IATA,ICAO)分别为(PEK,ZBAA)，大兴机场为(PKX,ZBAD),天河机机场为(WUH,ZHHH)。在飞机登记牌上出发地和到达地均用 IATA 表示。为能在地图上显示机场位置，需要记录经纬度信息。

4.航空公司(airline) 有以下属性: 编号：airline_id int 自增，主码 名称：name varchar(30) 不可空 国际民航组织编码: iata char(2) 不可空，具全球唯一性 航空公司的 IATA 编码为 2 位，如东航为 MU，国航为 CA,南航为 CZ 等，航班号一般以所属航空公司的 IATA 码为前缀。

5.民航飞机(airplane) 有属性： 编号：airplane_id int 自增，主码 机型：type varchar(50) 不可空，如 B737-300,A320-500 等 座位: capacity smallint 不可空 标识: identifier varchar(50) 不可空

6.航班常规调度表(flightschedule) 航班一般以周次安排，例如：每周两个班次，周一和周五。 有属性： 航班号: flight_no char(8) 主码 起飞时间: departure time 非空 到达时间: arrival time 非空 飞行时长: duration smalint 非空 周一：monday tinyint 缺省 0 周二：tuesday tinyint 缺省 0 周三：wednesday tinyint 缺省 0 周四：thursday tinyint 缺省 0 周五：friday tinyint 缺省 0 周六：saturday tinyint 缺省 0 周日：sunday tinyint 缺省 0 飞行时长一般以分种计。

7.航班表(flight) 航班依航班常规调度表为基准安排。但调度表不是一成不变，也不是每个既定的航都实际起飞，也不总是按既定的时间起飞，所以实飞航班必须单独安排并记录。 要记录的信息有： 飞行编号：flight_id int 自增，主码 起飞时间: departure datetime 非空 到达时间: arrival datetime 非空 飞行时长：duration smallint 非空 有的系统还会实时显示航班经纬度和高度位置，这里我们作了简化，去掉了实时飞行信息。

8.机票(ticket) 用户可替自己或其亲友购买某个航班的机票。机票的属性有： 票号：ticket_id int 自增，主码 坐位号：seat char(4) 价格：price decimal(10,2) 不能空