Updating Database and Course Resources Evaluation Report

Project for Principles of Database (H)

Yanqiao Chen(12412115), SUSTech, Shenzhen

Instructor: Dr. Shiqi Yu

December 10, 2025

Contents

1 项目基本信息

1.1 使用工具

工具	版本	说明
Python	3.8.10	编程语言
Scrapy	2.5.1	爬虫框架
PostgesSQL	13.3	数据库管理系统

1.2 项目目标

通过爬虫爬取电影数据库 tmdb 电影和演员数据，更新旧有教学电影数据库中信息（自 2019 以后，根据查询可知电影数据库信息截至 2019 年）。限于 API 爬取速度限制，我只爬取了 2019 年以后的部分电影数据和其部分演员数据，如有需要更新之前的数据，只需要调整爬虫代码中的年份范围以及爬取限制即可，但是这可能需要更长的事件或者使用反爬机制，有可能有法律风险，因此此处并不尝试。

限于 API 速率限制，以及为“教学用”（轻量化、小型化）数据库的考量，我们只爬取每个年份的部分电影数据（数量不等）。如果后续需要增量式更新，本爬虫亦可以通过修改 peopleid 起始以及年份范围来进行增量式更新。

1.3 更新内容

更新了三个表单：movies, credits, people, 并且向 countries 中插入了“??”作为数据库未知国家的占位符。同时注意到，现有数据库中的国家代码”sp” 不符合 ISO 3166-1 标准，但是为了统一修改，将读取到的西班牙国家代码”es” 全部替换为”sp”。同时，我们添加了巴勒斯坦”ps” 作为国家代码。

2 项目过程

整体项目过程如下：

1. 向 tmdb 数据库申请 api 密钥
2. 编写 Scrapy 爬虫，爬取 2019 年以后的电影数据
3. 使用 psycopg2 导入数据库

2.1 申请 API

在 tmdb 官网注册账号后，进入设置页面，申请 API 密钥。申请过程中需要填写一些基本信息以及使用目的等。申请成功后，可以在设置页面查看到 API 密钥。

2.2 编写爬虫

我们本次使用 Scrapy 框架编写爬虫。首先创建一个新的 Scrapy 项目，然后创建一个新的爬虫，命名为 tmdb_spider 并存储在 spiders 文件夹下。然后，我们配置 settings.py 文件和 pipeline.py 文件，以便将爬取的数据存储到数据库中。最后，我们编写 tmdb_spider.py 文件，定义爬虫的行为和数据处理逻辑。

一个典型的爬虫框架如下：

- start_requests: 定义初始请求，通常是从某个 URL 开始爬取数据。
- parse: 处理响应数据，提取所需信息，并生成新的请求。
- item_pipeline: 处理提取的数据，进行清洗、存储等操作。

根据主要的几个表格的关系，我们首先从电影本身的信息开始爬取，然后对于每个电影，爬取其演员和工作人员的信息。最后，将所有数据存储到数据库中。

本次项目中，我进行了如下的操作：

1. 发起请求，获取 2019 年以后的电影列表。
2. 将请求转换为 JSON 格式，便于数据处理。
3. 解析电影数据，提取电影 ID、标题、简介、发布日期、评分等信息。
4. 对于每部电影，发起新的请求，获取其演员和工作人员信息。
5. 解析演员和工作人员数据，提取演员 ID、姓名、角色等信息。
6. 将提取的数据存储到数据库中。同时，为了避免重复存入演员信息（保证唯一的 peopleid），我们使用构造 sql 查询语句检查演员是否已经存在于数据库中。如果已经存在，那么 credit 表中应当使用已有的 peopleid，否则插入新的演员信息并获取新的 peopleid。

在实际爬取过程中，因为一些设置上的问题，我们进行了多次不同年份的爬取。由于单个年份的电影数据较多，且电影网站往往将某一年份的电影持续推送，因此我们在以后的爬取中可能需要设置单个年份的爬取上限以满足各年份均匀分布的要求。否则，爬取到的电影数据可能会集中在某些年份，导致数据库更新不均衡；如果需要更新全部 2019 年以后的数据，则需要更长的时间和更复杂的反爬机制。

2.3 数据库更新结果

最后结果显示，我们成功更新了 2019 年电影 4426 条，涉及演员和导演近两万人，并且成功将数据存储到数据库中。数据库的完整性和一致性得到了保证，所有数据均符合预期格式和要求。对于那些国家信息缺失的电影，我们使用了“??”作为占位符，确保数据库的完整性。

数据详见附录中的 SQL 文件。本 SQL 文件已经在 Opengauss 上测试过，能够成功且完整的插入数据。SQL 文件通过 pg_dump 工具生成，包含了插入数据的完整 SQL 语句。为了适配 OpenGauss 数据库，我们删除了生成的 SQL 脚本中的一些参数设置和语法，以确保兼容性。

3 课程资源的评估

3.1 Some Comments on the Lecture Notes

Summary Slides 在阅读 Lecture Notes 的时候，我发现在大部分 Slides 中，内容比较分散且不够系统化。在重新阅读复习的过程中，需要花费很大的精力区分例子和关键定义与注意项目。建议在 Lecture Notes 中，根据 Chapter 中不同的概念，增加一页 Summary 进行重点内容的总结和归纳，以此方便在课堂中以及复习过程中方便阅读和理解。

Slides 中的显示错误

- Chapter 2 Slide 13, 14, 15, 29, 61: 文字标题显示重叠；
- Chapter 2 Slide 23: 文字显示重叠；
- Chapter 9 Slide 1, 标题 Trigger 少打了一个 g；

笔误

- Chapter 2 Slide 38, 34: not null 前面错误的添加了逗号，导致语法错误；

一些补充更新

- Chapter 2 Slide 20: 实际上，许多的 SQL 方言可以开启严格模式，从而阻止一些可能不符合逻辑或者可能导致错误的插入，例如 MySQL 的 STRICT_ALL_TABLES 模式；不过，数据库在严格模式下仍然不会符合关系模型中的所有约束；
- Chapter 2 Slide 36: 长段注释语法“/* */”可以补充说明；

- Chapter 3 Slide 30, 第三行在部分数据库的日期转换中可能返回 null，移植性较差；
- Chapter 4 Slide 19，可以在 PPT 中明确附加 where 和 having 的执行顺序（where 对原始行过滤，having 对 group by 后的各组进行过滤）；
- Chapter 4 Slide 27, 可以在 PPT 中附加 count(*) 和 count(col) 对比：count(*) 不会检查数据中是否保存了 null，count(col) 会检查数据行对应的列中是否包含 null；
- Chapter 5 Slide 61, 可以在 PPT 上强调关联子查询和普通子查询效率上存在区别的原因，例如给出一个具体的 SQL 查询例子；
- Chapter 5 Slide 75，可以增加对于此类运算和解释：由于逻辑短路，对于 and 运算，如果第一个是 False，呢么就是 False；如果第一个是 True，还需要检验第二个；
- Chapter 7 Slide 8，介绍全文搜索的时候，可以介绍其优势，例如，全文搜索可以利用倒排索引提高搜索效率，因此其比 like 的搜索效率更高；
- Chapter 7，可以增加对于 MVCC 的介绍；
- Chapter 7 Slide 81, 可以增加对于 with 语法中，指定分隔符等额外选项的语法；

关于 Opengauss 的增补 由于 OpenGauss 根据 PostgreSQL 较早版本开发，许多功能上为了与 PostgreSQL 兼容，因此 OpenGauss 本身并没有太多创新的特性。但是，OpenGauss 在 PostgreSQL 的基础上，增加了一些新的特性和功能，以满足现代数据库系统的需求。以下是一些我发现的 OpenGauss 的新增特性：

- OpenGauss 增加了一些特性，例如支持 AI4DB，也就是通过机器学习算法辅助开发人员进行数据库运维监控。这一功能主要通过华为开发的数据库运维平台 DBMind 进行。项目开源在 Gitcode 上。
- GaussMaster 则支持基于 opengauss 数据库本身，构建用户的知识库，支持用户直接通过自然语言询问数据库相关问题，并且通过大模型进行回答。
- OpenGauss 支持使用 CREATE MODEL 等 DB4AI 语法，支持在数据库内训练模型，并且集成了部分经典 AI 算法，例如线性回归、逻辑回归等。
- OpenGauss 支持 SQL 预处理，通过预处理客户在其它数据库中编写的 SQL 转化为 OpenGauss 支持的 SQL，方便客户转移业务数据库。
- OpenGauss 内置了 stack 工具，支持运维人员直接调用指定线程的调用栈，分析死锁等问题。

- OpenGauss 具有特有的 DB4AI Schema。不过，OpenGauss 的官方文档对此介绍不多。

- OpenGauss 原生支持向量数据类型，而 PostgreSQL 需要通过插件进行支持。

如上等等，我们发现，OpenGauss 的开发有两个特点：一是兼容性，支持开发人员将原有的数据库业务转移到 OpenGauss 上面；二是智能化，支持 AI4DB 和 DB4AI 等功能，迎合当前 AI 发展的潮流。这实际上是 OpenGauss 的一种策略。一方面，OpenGauss 处于初步发展阶段，当前已有很多开源数据库，例如 PostgreSQL、MySQL 等，OpenGauss 需要通过兼容性来吸引用户使用；另一方面，AI 技术的发展为数据库系统带来了新的机遇，OpenGauss 通过集成 AI 功能，提升数据库的智能化水平，从而增强其竞争力。

事实上，近些年来 PostgreSQL 虽然没有在数据库原生支持 AI 功能，但是在各类插件中，已经有许多支持机器学习的插件，例如 pgvector 插件。

事实上，在机器学习和 AI 技术的发展下，数据库也大概率会在人工智能的帮助下进化出更加丰富的功能，例如更好的 SQL 优化器、更加智能的索引选择、更加快速的文本搜索和语义理解，甚至最终我们将能够完全通过自然语言进行数据库查询操作，从而实现对于大量数据的控制。

关于 Lecture Slide 的行文建议 在原始的 Lecture Slide 中，许多内容的行文比较零散且不够系统化。尤其是对于一些重点的细节进行介绍的时候，重点内容缺乏加粗和强调；对于一些特性进行介绍的时候，学生往往很难区分哪些是重点内容，哪些是次要内容。另外，Lecture Slide 牺牲了大量系统性以换取内容的风趣性和生动性（例如，提供了很多图片和生动的例子，这是很好的），这对于学生再次翻看 Slide 的时候，理解和记忆内容造成了一定的困难。

3.2 Some Comments on the Original Filmdb

关于国家编号不符合 ISO-3166-1 的提示 在更新数据库的过程中，我发现现有 filmdb 数据库中的国家编号不符合 ISO-3166-1 标准。例如，西班牙的国家代码应为"es"，但在数据库中使用了"sp"。为了保持一致性，我在爬取数据时将西班牙的国家代码统一替换为"sp"。建议在数据库设计和维护过程中，严格遵守国际标准，以避免混淆和错误。由于 filmdb 中的数据库中原始数据来源不明，因此为了减小对于数据库现有数据的影响，在更新的过程中保持与数据库内部一致。但是建议在后续的数据库设计中，遵循国际标准。

对于数据库中其它表的疑问 在课程的练习中，Filmdb 中存在一些我们从来不会使用的表格，里面往往只存储了少量的数据，有些数据甚至是法语标题，在练习的时候一般不会用到。在后续的维护中，这些表似乎是可以删除的。

4 一些课程建议

4.1 关于 Lecture

在 Lecture 部分，我认为可以增加一些对于数据库本身架构开发的内容，例如，对于 PostgreSQL 代码的观察和分析，以及这些代码如何实现数据库的基本功能和特性。可以适当地减少对于 SQL 语法本身细节的讲解，而增加在 Lab 和 Project 中增加对于 SQL 语法细节的练习（例如，复杂查询、语法细节等等），如果能够更多地上手练习，对于理解 SQL 语法和数据库设计会有更大的帮助。

另外，从 CMU-15-455 课程中，我发现可以增加一些对于数据库系统内部实现的内容，而不是仅仅停留在 SQL 语法和数据库架构的层面。通过了解数据库系统的内部实现原理，可以更好地理解数据库的性能优化和设计原则。例如，CMU 数据库课程的第一个 Project 是编写一个 Hyperloglog 算法来估计数据的基数，这对于理解数据库中的数据处理和优化有很大的帮助，我们可以将此类算法的实践作为 Lab 的一部分内容。

此外，增加一些对于现代数据库前沿的内容，例如分布式数据库、并发控制等内容，可以帮助学生了解数据库领域的最新发展和趋势。

另外，我觉得这门课开在图灵班大二上实际上不太好，因为大二的计算机学生通常对于计算机软件开发的知识尚不成熟，语言熟练度尚且不足，对于数据库系统的理解也比较浅显。正因为此，许多同学对于数据库的理解止步于 SQL 语言本身，对于数据库的重要性和实际应用缺乏深入了解。在 CMU-15-455 课程中，该课程要求学生先修计算机系统课程以及 C++ 语言课程，并且在课程初期有一个 Pass or Fail Project （只有拿到满分，才能够继续课程，否则将被劝退）。我认为我们在课程的初期也可以设置类似的门槛，充分利用南科大前四周的退课时间窗口，确保选课的同学对于数据库系统有足够的兴趣和基础，从而提高课程的整体水平和学习效果。；同时，如果学生学习过计算机组成原理和计算机系统，他会知道为什么把需要查询的数据缓存在内存上会更快，从而更好理解数据库设计的动机；如果学生学习过操作系统，他会更好地理解并发控制和事务管理的原理，从而更好地理解数据库系统的设计和实现；如果学生学习过编译原理，他会更好地理解 SQL 查询优化器的工作原理，从而更好地理解数据库系统的性能优化。

如上，可以表明数据库本身实际上是一个需要大量前置课程的课程，因此我建议将数据库课程安排在大三上学期，这样，按照图灵班的进度，学生在大三上学期之前已经学习过计算机系统、操作系统、编译原理等课程，从而为数据库课程打下坚实的基础。

我相信这样的调整会显著提升学生对数据库系统的理解和兴趣，从而提高课程的整体质量和学习效果。

4.2 关于 Lab

在 Lab 部分，我认为可以分为两个部分：SQL 语法练习和数据库编程练习。这些练习不必上交评分，而是作为学生自我练习和巩固知识的机会。通过实际操作，学生可以更好地理解 SQL 语法和数据库设计的原理。同时，我认为这一门课需要几个助教来协助完成 Lab 的设计和实现工作。

详细地说，SQL 语法练习不仅仅是简单的查询练习，而是需要和工程实践结合起来，例如通过 Java/Python 等语言连接数据库，进行实际的数据操作和查询。这不仅可以帮助学生理解 SQL 语法的实际应用，还可以提高他们的编程能力和数据库操作技能。另外，也可以使用一些教学用数据库（如有），让学生在 Lab 中进行实际的数据库设计和实现，从而更好地理解数据库系统的工作原理和设计思路。

4.3 关于 Project

实际上，我们可能需要设计一个课程数据库，让学生在 Lab 中进行实际的数据库设计和实现。通过实际操作，学生可以更好地理解数据库的设计原则和实现方法。例如，CMU-15-455 课程中，课程项目组提供了一个名为 Bustub 的教学数据库系统，学生需要在此基础上实现各种数据库功能和特性。这种实践性的学习方式可以帮助学生更好地理解数据库系统的工作原理和设计思路；至于国内，华中科技大学在数据库课程中提供了一个基于 oceanbase 开发的 miniob 系统，学生需要在此基础上实现各种数据库功能和特性。我认为我们也可以设计一个类似的教学数据库系统，让学生在 Lab 中进行实际的数据库设计和实现。这一份工作可以作为课程助教的长期接续工作。

另外，CMU-15-455 课程中还提供了很多需要阅读的 Paper，我们可以效仿这一点，在 Project 中增加“综述”这一部分内容，让学生阅读一些数据库前沿的论文，并撰写综述报告。这不仅可以帮助学生了解数据库领域的最新发展和趋势，还可以提高他们的学术阅读和写作能力。

如上，我们可以列出设计出的若干 Project 供学生任选:

- 实现数据库的关键优化功能，例如 Hyperloglog 等算法（事实上，这一算法已经被 PostgreSQL 在源代码中使用，是对于数据基数估计的一种有效方法）；
- 让学生阅读指定及自行查找指定的论文，撰写综述报告；
- 对数据库感兴趣的同学，可以根据自己的兴趣，进行设计和探索，并且尝试在课程结束后发表论文，这可以作为 Bonus；
- 协助教师设计和实现一个教学数据库系统并开源，为学弟学妹们提供一个更好的学习平台。

- 实现爬虫，爬取指定网站的数据并存储到数据库中。
- 使用小型框架构建一个网站，连接数据库，并且通过模拟攻击测试数据库的安全性。

5 总结

综上，我们对本课程的练习数据库的核心四个表进行了完全的更新，并且通过考察其它表的作用，给出了一些建议。同时，我们对于课程材料进行了从头到尾的检查，并给出了一些相关的提醒和建议。最后，我们对课程的全过程：开课时间和课程准备、Lecture 和 Lab、Project 等方面，都给出了一些建议，希望能够帮助课程的改进和提升。

附录中后附了爬虫代码，如果后续有更新需要，仍然可以使用此爬虫进行更新爬取最新的电影。

总而言之，希望本次的 Project 对于数据库原理（H）这门课的开设、维护和改进有所帮助。另外，我们的确非常需要几位助教来答疑、协助完成 Lab 和 Project 答疑，这将对数据库原理这门课的顺利进行有很大的帮助。

A Python Scrapy 爬虫代码

Listing 1: tmdb_spider.py 爬虫代码

```
import scrapy
import re


class TmdbSpider(scrapy.Spider):
    name = "tmdb"
    allowed_domains = ["api.themoviedb.org"]

    API_KEY = "Your API Key"

    def start_requests(self):
        url = "https://api.themoviedb.org/3/discover/movie"
        base_params = {
            'api_key': self.API_KEY,
            'sort_by': 'popularity.desc',
```

```
            'page': 1,
            'with_release_type': '2|3',
        }
        years = range(2020, 2021)

        for year in years:
            year_params = base_params.copy()
            year_params['primary_release_year'] = year

            query_string = '&'.join([f"{k}={v}" for k, v in year_params.items
                ()])
            full_url = f"{url}?{query_string}"

            self.logger.info(f"Starting␣crawl␣for␣year:␣{year}")
            yield scrapy.Request(url=full_url, callback=self.parse_discover)

    def parse_discover(self, response):
        try:
            data = response.json()
        except Exception as e:
            self.logger.error(f"Failed␣to␣parse␣JSON␣from␣Discover:␣{e}")
            return

        for movie in data.get('results', []):
            movie_id = movie.get('id')
            detail_url = f"https://api.themoviedb.org/3/movie/{movie_id}?
                api_key={self.API_KEY}"
            yield scrapy.Request(
                url=detail_url,
                callback=self.parse_details,
                meta={'initial_data': movie}
            )

        current_page = data.get('page')
```

```
48          total_pages = data.get('total_pages')
49
50          if current_page is not None and current_page < total_pages:
51              if current_page >= 500:
52                  self.logger.warning("Reached␣TMDB␣API␣limit␣(Page␣500).␣
                        Stopping␣pagination␣for␣this␣query.")
53              else:
54                  next_page = current_page + 1
55                  next_url = re.sub(r'page=\d+', f'page={next_page}', response.
                        url)
56                  self.logger.info(f"Paging:␣Requesting␣page␣{next_page}␣of␣{
                        total_pages}")
57                  yield scrapy.Request(url=next_url, callback=self.
                        parse_discover)
58
59      def parse_details(self, response):
60          initial_data = response.meta['initial_data']
61          try:
62              detail_data = response.json()
63          except Exception as e:
64              self.logger.error(f"Failed␣to␣parse␣JSON␣from␣Details:␣{e}")
65              return
66
67          item_data = {
68              'movieid': detail_data.get('id'),
69              'title': initial_data.get('title'),
70              'release_date': initial_data.get('release_date'),
71              'runtime': detail_data.get('runtime', 0) or 0
72          }
73
74          countries = detail_data.get('production_countries', [])
75          if countries:
76              code = countries[0].get('iso_3166_1')
77              if code == 'ES' or code == 'es':
```

```
78                code = 'sp'
79            if code == 'KN' or code == 'kn':
80                code = 'ke'
81            item_data['country'] = code.lower() if code else '??'
82        else:
83            item_data['country'] = '??'
84
85        credits_url = f"https://api.themoviedb.org/3/movie/{item_data['
              movieid']}/credits?api_key={self.API_KEY}"
86        yield scrapy.Request(
87            url=credits_url,
88            callback=self.parse_credits,
89            meta={'item_data': item_data}
90        )
91
92    def parse_credits(self, response):
93        from ..items import TmdbMovieItem
94
95        item_data = response.meta['item_data']
96        try:
97            credits_data = response.json()
98        except Exception as e:
99            self.logger.error(f"Failed to parse JSON from Credits: {e}")
100           return
101
102       item = TmdbMovieItem(**item_data)
103       processed_people = []
104
105       for p in credits_data.get('cast', [])[:5]:
106           processed_people.append({
107               'tmdb_id': p.get('id'),
108               'name': p.get('name'),
109               'job': 'A',
110               'gender': None,
```

```
                'born': None,
                'died': None
            })

        for p in credits_data.get('crew', []):
            if p.get('job') == 'Director':
                processed_people.append({
                    'tmdb_id': p.get('id'),
                    'name': p.get('name'),
                    'job': 'D',
                    'gender': None,
                    'born': None,
                    'died': None
                })

        if processed_people:
            item['cast_crew'] = processed_people

            first_person = processed_people[0]
            person_url = f"https://api.themoviedb.org/3/person/{first_person
                ['tmdb_id']}?api_key={self.API_KEY}"

            yield scrapy.Request(
                url=person_url,
                callback=self.parse_person_details,
                meta={
                    'item': item,
                    'people_list': processed_people,
                    'current_index': 0
                }
            )
        else:
            item['cast_crew'] = []
            yield item
```

```
144
145     def parse_person_details(self, response):
146         meta = response.meta
147         try:
148             person_data = response.json()
149         except Exception as e:
150             self.logger.error(f"Failed to parse JSON from Person: {e}")
151             return
152
153         item = meta['item']
154         people_list = meta['people_list']
155         current_index = meta['current_index']
156
157         person = people_list[current_index]
158
159         gender_code = person_data.get('gender')
160         if gender_code == 1:
161             person['gender'] = 'F'
162         elif gender_code == 2:
163             person['gender'] = 'M'
164         else:
165             person['gender'] = '?'
166
167         birthday = person_data.get('birthday')
168         if birthday:
169             person['born'] = int(birthday[:4])
170         else:
171             person['born'] = 0
172
173         deathday = person_data.get('deathday')
174         if deathday:
175             person['died'] = int(deathday[:4])
176         else:
177             person['died'] = None
```

```

        next_index = current_index + 1

        if next_index < len(people_list):
            next_person = people_list[next_index]
            person_url = f"https://api.themoviedb.org/3/person/{next_person
                ['tmdb_id']}?api_key={self.API_KEY}"

            yield scrapy.Request(
                url=person_url,
                callback=self.parse_person_details,
                meta={
                    'item': item,
                    'people_list': people_list,
                    'current_index': next_index
                }
            )
        else:
            yield item
```

Listing 2: pipeline.py 数据库存储代码

```
import psycopg2
from scrapy.exceptions import DropItem
from .items import TmdbMovieItem


class PostgresPipeline:
    START_PEOPLE_ID = 26319 # 此处根据数据库实际调整
    people_cache = {}

    current_people_id = START_PEOPLE_ID

    @classmethod
    def from_crawler(cls, crawler):
```

```
14          # 从 settings.py 中加载数据库配置
15          db_settings = crawler.settings.getdict('DATABASE')
16          return cls(db_settings)
17
18      def __init__(self, db_settings):
19          self.db_settings = db_settings
20          self.conn = None
21          self.cursor = None
22
23      def open_spider(self, spider):
24          """爬虫开启时连接数据库"""
25          try:
26              self.conn = psycopg2.connect(
27                  host=self.db_settings['host'],
28                  port=self.db_settings['port'],
29                  database=self.db_settings['database'],
30                  user=self.db_settings['username'],
31                  password=self.db_settings['password']
32              )
33              # 关闭自动提交，手动控制事务
34              self.conn.autocommit = False
35              self.cursor = self.conn.cursor()
36              spider.logger.info("Database connection established successfully.
                    ")
37
38              try:
39                  self.cursor.execute(f"SELECT setval('people_peopleid_seq', {
                        self.START_PEOPLE_ID - 1}, true);")
40                  self.conn.commit()
41                  spider.logger.info(f"peopleid sequence set to start at {self.
                        START_PEOPLE_ID}.")
42              except psycopg2.Error as e:
43                  self.conn.rollback()
44                  spider.logger.warning(f"Could not set sequence (might not
```

```
                exist):␣{e}")
45
46          except psycopg2.Error as e:
47              spider.logger.error(f"Database␣connection␣failed:␣{e}")
48              raise
49
50      def close_spider(self, spider):
51          """爬虫关闭时关闭连接"""
52          if self.conn:
53              self.conn.close()
54              spider.logger.info("Database␣connection␣closed.")
55
56
57      def _execute_sql(self, sql, params=None):
58          """执行␣SQL␣语句，出错时抛出异常由␣process_item␣捕获"""
59          self.cursor.execute(sql, params)
60
61      def process_item(self, item, spider):
62          if not isinstance(item, TmdbMovieItem):
63              return item
64
65          try:
66              # 1. 插入或更新 Movies 表 (父表)
67              self._insert_movie(item)
68
69              # 2. 遍历演职员，处理 People 和 Credits
70              for person_data in item.get('cast_crew', []):
71                  # 获取有效的人物 ID (查找现有或插入新人物)
72                  people_id = self._insert_or_lookup_person(person_data)
73
74                  # 3. 插入 Credits 表 (子表)
75                  self._insert_credit(item['movieid'], people_id, person_data['
                        job'])
76
```

```
77              # 4. 提交整个事务 (只有当所有步骤都成功时)
78              self.conn.commit()
79              # spider.logger.debug(f"Committed transaction for movie: {item['
                    movieid']}")
80
81          except psycopg2.Error as e:
82              self.conn.rollback()
83              spider.logger.error(
84                  f"DB␣Error␣processing␣movie␣{item.get('movieid')}:␣{e.pgerror
                        .strip()␣if␣e.pgerror␣else␣e}")
85
86          except Exception as e:
87              self.conn.rollback()
88              spider.logger.error(f"General␣Error␣processing␣movie␣{item.get('
                    movieid')}:␣{e}")
89
90          return item
91
92      def _insert_movie(self, item):
93          """插入或更新␣Movies␣表"""
94
95          country_code = item['country']
96
97          sql = """
98  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣movies␣(movieid,␣title,␣country,␣year_released,␣
        runtime)
99  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣VALUES␣(%s,␣%s,␣%s,␣%s,␣%s)␣ON␣CONFLICT␣(movieid)␣DO␣\
100 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣UPDATE␣\
101 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SET␣title␣=␣EXCLUDED.title,␣runtime␣=␣EXCLUDED.runtime;␣\
102 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
103         params = (
104             item['movieid'],
105             item['title'],
106             country_code,
```

```
107             int(item['release_date'][:4]) if item['release_date'] else 0,
108             item['runtime']
109         )
110         self._execute_sql(sql, params)
111 
112     def _insert_or_lookup_person(self, person_data):
113         """
114 ␣␣␣␣␣␣␣␣查找或插入人物，并返回␣peopleid。
115 ␣␣␣␣␣␣␣␣逻辑:
116 ␣␣␣␣␣␣␣␣1.␣检查本地缓存。
117 ␣␣␣␣␣␣␣␣2.␣检查数据库␣(SELECT)。
118 ␣␣␣␣␣␣␣␣3.␣如果不存在，插入新记录␣(INSERT)。
119 ␣␣␣␣␣␣␣␣"""
120         full_name = person_data.get('name')
121         parts = full_name.strip().split('␣', 1)
122         first_name = parts[0] if len(parts) > 1 else None
123         surname = parts[-1]
124         if len(parts) == 1:
125             first_name = None
126 
127         people_key = (surname, first_name)
128 
129         if people_key in self.people_cache:
130             return self.people_cache[people_key]
131 
132         lookup_sql = """
133 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣peopleid␣\
134 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣people
135 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣surname␣=␣%s␣\
136 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣AND␣first_name␣IS␣NOT␣DISTINCT␣\
137 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣%s;␣\
138 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
139         self._execute_sql(lookup_sql, (surname, first_name))
140         result = self.cursor.fetchone()
```

```
141
142         if result:
143             people_id = result[0]
144             self.people_cache[people_key] = people_id
145             return people_id
146
147         assigned_id = self.current_people_id
148
149         born_year = person_data.get('born', 0)
150         died_year = person_data.get('died')
151         gender_char = person_data.get('gender', '?')
152
153         insert_sql = """
154 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣people␣(peopleid,␣first_name,␣surname,␣
    born,␣died,␣gender)
155 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣VALUES␣(%s,␣%s,␣%s,␣%s,␣%s,␣%s);␣\
156 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
157         params = (
158             assigned_id,
159             first_name,
160             surname,
161             born_year,
162             died_year,
163             gender_char
164         )
165         self._execute_sql(insert_sql, params)
166
167         self.current_people_id += 1
168         self.people_cache[people_key] = assigned_id
169
170         return assigned_id
171
172     def _insert_credit(self, movieid, peopleid, credited_as):
173         sql = """
```

```
174                INSERT INTO credits (movieid, peopleid, credited_as)
175                VALUES (%s, %s, %s) ON CONFLICT DO NOTHING; \
176                """
177            params = (movieid, peopleid, credited_as)
178            self._execute_sql(sql, params)
```

Listing 3: items.py Item 定义代码

```
1  import scrapy
2
3
4  class TmdbMovieItem(scrapy.Item):
5      # Movies 表字段
6      movieid = scrapy.Field()
7      title = scrapy.Field()
8      release_date = scrapy.Field()
9      country = scrapy.Field()
10     runtime = scrapy.Field()
11
12     cast_crew = scrapy.Field()
```

Listing 4: settings.py 数据库配置代码

```
1
2  BOT_NAME = 'tmdb_scraper'
3
4  SPIDER_MODULES = ['project2.spiders']
5  NEWSPIDER_MODULE = 'project2.spiders'
6
7  USER_AGENT = 'tmdb_scraper'
8
9  ROBOTSTXT_OBEY = True
10
11
12 CONCURRENT_REQUESTS = 4
```

```
(4 * 0.25 = 1秒)
DOWNLOAD_DELAY = 0.25

AUTOTHROTTLE_ENABLED = True
AUTOTHROTTLE_START_DELAY = 5.0
AUTOTHROTTLE_MAX_DELAY = 60.0
AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0

DATABASE = {
    'drivername': 'postgresql',
    'host': 'localhost',
    'port': '5432',
    'database': 'project2',
    'username': 'postgres',
    'password': '1234'
}

ITEM_PIPELINES = {
    'project2.pipelines.PostgresPipeline': 300,
}

LOG_LEVEL = 'INFO'

RETRY_TIMES = 3

COOKIES_ENABLED = False
```

B 更新后的数据库导出的 SQL 文件

见后附文件 filmdb.sql。