

Improve the database FilmDB
and Course Materials

SUSTech
CS213/Project 2: Filmdb 数据库补
全，课程材料更新与建议

火明贤

12/2025

摘要

本项目旨在更新 filmdb 教学数据库，使其包含 2018-2025 年的热门电影数据，并针对 PostgreSQL 和 openGauss 环境优化课程内容。针对 Database Enhancement (Task 1)，我开发了一套基于 Python 的 ETL (Extract, Transform, Load) 自动化流程。该流程利用 TMDB API 获取数据，并采用“Staging Table (中转表)”策略来确保数据的安全性与完整性。为了解决新旧数据冲突，我设计了内存哈希索引 (In-memory Hash Indexing) 来实现演员数据的智能去重与 ID 映射，并基于 origin_country 字段修复了国家代码的准确性。最终，数据库成功新增了 6k+ 电影。此外，本项目还审查了现有课件，并提出了关于 openGauss 架构的新课程模块建议，以紧跟数据库技术的最新发展与国家发展需求。

1. 引言

背景 (Background)：CS213 提供的 filmdb 数据库主要包含 2018 年之前的电影数据。随着数据库技术的演进和电影行业的变迁，过时的数据限制了学生在查询优化、数据分析等场景下的实践体验与课程严谨性。此外，随着国产数据库的兴起，课程需要引入 openGauss 相关的教学内容以满足新的技术标准。

目标 (Objectives)：本项目的主要目标是：

1. 实施数据增

强策略，通过外部 API 引入 2018-2025 年的高热度电影数据。2. 确保更新后的数据库严格遵守参照完整性 (Referential Integrity)，并兼容 PostgreSQL 和 openGauss。3. 提出符合当前技术趋势（如 AI 与国产化）的课程新模块。

方法论 (Methodology)：为保证现有数据的安全，我没有采用直接插入 (Direct Insertion) 的方式，而是设计了“三层中转架构”：先将清洗后的数据存入 Staging Tables，经过去重校验后，再合并至主表。这种方法模拟了工业界真实的数据仓库更新流程。

2. FilmDB 数据库更新

2.1. 数据获取

为了获取高质量的现代电影数据，我开发了一套定制化的 Python ETL (Extract, Transform, Load) 脚本，该脚本充当了外部 API 与本地数据库之间的桥梁。

API 集成：数据源选用了业界标准的 TMDB (The Movie Database) API。我使用了 Python 的 requests 库来处理 HTTP 请求，并实现了自动重试机制以应对网络波动或 API 限流 (Rate Limiting, HTTP 429)。

定向提取策略: 脚本并非盲目抓取，而是通过 `/discover/movie` 端点配置了特定的过滤器：

Temporal Scope: `primary_release_year` 设定为 2018–2025。

Relevance: 使用 `sort_by=popularity.desc` 确保优先获取高热度电影。

Deep Fetching: 对于每部电影，脚本会进一步查询详情接口以获取准确的 `runtime`，并解析 `credits` 结构提取导演及前两名主演信息。

Staging Architecture: 最关键的方法论创新在于使用了 Staging Tables。数据没有直接写入生产环境，而是先存入结构相同的临时表 (`new_movies_staging` 等)。这种“缓冲区”设计允许我们在最终提交 (Commit) 之前进行复杂的 SQL 清洗和验证操作。

```
> └─ new_credits_staging  
> └─ new_movies_staging  
> └─ new_people_staging
```

2.2. 数据合并

在将数据从 Staging 表合并至主表 (Production Tables) 过程中果然遇到了一系列问题，根据 bug 提示一步步解决。

2.1.1 实体完整性与去重: 原数据库中已包含少量 2018 年的电影 (如 `Ready Player One`)，直接合并会导致主键冲突。

```
[23505] 错误: 重复键违反唯一约束"movies_title_country_year_released_key"  
详细: 键值"(title, country, year_released)=(Ready Player One, us, 2018)" 已经存在
```

我实施了“合并前清洗”策略，使用 `DELETE ... WHERE EXISTS` 语句，从 `Staging` 表中移除了那些 `(title, country, year)` 组合已存在于主表的记录。

针对新数据中出现的知名演员（如 `Vincent D'Onofrio`），我编写了 SQL 逻辑将 `Staging` 表中的新 ID 映射回主表中已存在的旧 ID。这防止了同一演员在数据库中出现重复记录（与上电影重复相似），维护了数据的唯一性。

2.2.2 遵守约束： 新数据与旧 Schema 定义存在冲突，通过数据转换解决了以下问题：

针对 `Error 22001 (String Data Right Truncation)`，使用 `LEFT(column, 100)` 和 `LEFT(column, 30)` 函数，分别对超长的电影标题和人名进行了截断，以符合 `VARCHAR(100)` 和 `VARCHAR(30)` 的列定义。

针对缺失出生年份的次要演职员，使用 `COALESCE(born, 0)` 填充默认值，既满足了非空约束，又保留了记录的完整性。

2.2.3 ID 连续性与参照完整性 为了保证主键 ID 的整洁与连续：使用 `ROW_NUMBER()` 窗口函数对新加入的 `people` 进行了 ID 重排，使其紧接在原数据库最大 ID (16489) 之后（即从 16490 开始），消除了 ID 碎片。

最终的合并操作 (`Merge`) 被封装在一个 SQL 事务

(BEGIN . . . COMMIT) 中。这意味着电影、人员和演职员关系的插入要么全部成功，要么在遇到错误时全部回滚。这一机制确保了数据库永远不会处于“只有电影没有演员”的中间不一致状态。

3. 课程讲义更新建议

3. 1 Lec2:

①位置: Slide 17 – "The main variants of SQL"

问题: 目前列表中缺少国产及教学用数据库 openGauss。

新增内容: 在 Oracle, MySQL, PostgreSQL 旁增加 openGauss 的 Logo 和介绍。

描述: “openGauss: 一款企业级开源关系型数据库管理系统, 源自 PostgreSQL, 针对高性能和高并发场景进行了优化。”

②位置: Slide 28 – "Text datatypes"

openGauss 补充: 建议补充说明: 在 openGauss 中, TEXT 类型通常比 VARCHAR(n) 更受推荐用于存储变长字符串, 因为它没有性能惩罚, 这与某些旧式 DBMS 不同。

③位置: Slide 44 & 52 – "Primary Key" & "Foreign Key"

问题: 对键 (Key) 的解释主要停留在定义层面。

更新建议:

新增案例: 引入 错误代码 [23505] (Duplicate Key)

Violation / 重复键违反唯一约束)。

案例描述：“在合并新数据时，系统拦截了电影《头号玩家》

(Ready Player One, 2018)，因为该记录已存在于旧库中。

这次 unique (title, country, year_released) 约束的触发，成功保护了数据库免受脏数据(重复记录)的污染，证明了约束是数据质量的第一道防线。”

3. 2 Lec3:

①位置： Slide 41 - 日期运算

问题：列出了 SQL Server, DB2, Postgres, MySQL,

Oracle, SQLite，但缺失 openGauss。

更新建议：PostgreSQL 旁加入 openGauss (openGauss 与 PostgreSQL 语法兼容)。

②位置： Slide 57 - 字符串连接

问题：缺少 openGauss 的拼接语法。

更新建议：PostgreSQL 旁加入 openGauss (openGauss 与 PostgreSQL 语法兼容)。

3. 3 Lec4:

①位置： Slide 26 - "Count(*)"

问题：仅讨论了计数功能。

更新建议：

增加应用场景：说明 COUNT(*) 是 ETL (Extract,

Transform, Load) 流程中验证数据完整性的核心工具。

3. 3 Lec8:

位置： 索引类型介绍。

更新建议（针对新数据的优化）：

场景引入： 我们在 Task 1 中爬取了大量长文本电影标题（如 Borat...），使用标准的 like '%keyword%' 查询会全表扫描，非常慢。

openGauss 特性： 介绍 openGauss 支持的 GIN 索引，这对于文本搜索（全文检索）非常有效。

3. 4 Lec9:

位置： Isolation Levels

问题： 需要明确默认级别。

更新建议（openGauss 特性）：

说明： 指出 openGauss 的默认隔离级别是 Read Committed（读已提交），这与 Oracle/PostgreSQL 一致，能有效防止“脏读”。

3. 5 Lec10:

位置： 用户权限管理

问题： 缺少 OpenGauss 的特性介绍

更新建议（openGauss 特性）：

新增内容： 介绍 openGauss 的“三权分立”安全模型（系统管理员、安全管理员、审计管理员），这是其作为企业级数据库在安全性上的重要增强。

3.5 Lec11:

位置：范式与反范式

问题：缺乏权衡的讨论

更新建议（设计决策）：

案例引入：讨论我们在 Task 1 中遇到的“**截断策略**”。

分析：为了符合 3NF 或 Schema 定义，我们不得不将超长的电影标题截断为 100 字符。这是一个典型的“工程妥协”：在修改表结构（DDL 变更成本高）与清洗数据（可能会丢失信息）之间做出的选择。

4. 课程建议

建议主题：引入“openGauss 企业级架构与高性能调优”专题模块

1. 当前的课程主要关注通用的 SQL 语法，在某种程度上将 openGauss 仅视作 MySQL 或标准 PostgreSQL 的替代品来使用。

痛点：学生虽然学会了写查询，但并不理解为什么在金融、电信等国家关键基础设施中，我们需要自主可控的国产企业级数据库。

缺失：课程忽略了 openGauss 作为企业级数据库的核心竞争

优势——高性能存储引擎、高并发处理能力以及企业级安全特性。学生在完成作业时（如批量更新数据卡顿），往往是因为不懂底层原理（如事务日志机制），而不仅仅是 SQL 写得不对。

2. 核心建议（Core Proposal）建议新增一个“企业级数据库架构”模块，深度挖掘 openGauss 的特有功能，从“应用开发者”视角转向“数据库架构师”视角。这不仅是技术的提升，更是对国产软硬件生态的深度理解。

具体实施方案：

A. 存储引擎深度解析：行存 vs. 列存：

现状：学生默认使用行存储（Row Store），但在大数据分析场景下效率低下。

改进：利用 openGauss 支持多种存储引擎的特性，设计对比实验（类似 Project1）。

实验设计：让学生分别在“行存表”和“列存表（Cstore）”中对百万级电影数据执行聚合查询（例如计算 $\text{AVG}(\text{runtime})$ ）。

教学目标：让学生亲身体验到列式存储在 OLAP（分析型）场景下的性能飞跃，理解企业级数据库如何通过架构设计解决海量数据难题。

B. 高级性能调优：

现状：学生遇到慢查询（如 Project 中的批量 Update）时

束手无策。

改进：引入 openGauss 特有的 EXPLAIN PERFORMANCE 工具。

实验设计：对比“单条提交”与“批量事务提交”的执行计划与 CPU/I/O 消耗。

教学目标：培养学生的性能意识，理解 I/O 瓶颈与锁机制，掌握企业级数据库调优的“硬核”技能。

C. 企业级安全与三权分立：

现状：目前仅涉及基础的 GRANT/REVOKE。

改进：介绍 openGauss 的三权分立安全模型（系统管理员、安全管理员、审计管理员）。

教学目标：让学生理解为什么在银行和政府系统中，国产数据库的这种安全架构设计是不可替代的，从而提升对国产数据库战略意义的认知。

3. 预期价值

战略高度：将课程从单纯的“编程课”提升为符合国家信创战略的“架构课”，增强学生对国产技术的自信心和认同感。

就业竞争力：掌握列存储、执行计划分析和企业级安全配置，是互联网大厂和国企对后端/DBA 岗位的核心要求。

理论结合实践：通过 openGauss 的独有特性，将抽象的数据库原理（ACID、I/O 模型）具象化，让学生知其然，更知其所以然。

5. 总结

本项目通过三个阶段的系统性工作，成功实现了一个过时电影数据库的现代化重构，并以此为基础对《数据库原理》课程体系提出了具有战略意义的优化方案。

在 数据工程层面，通过 Python ETL 引入了 2018 至 2025 年的热门电影数据，更重要的是，我在处理真实世界的“脏数据”时，实践了严格的数据治理原则。从解决主键冲突到处理字段截断，再到利用事务机制（ACID）保护数据一致性，这一过程证明了约束（Constraints）是数据库质量的最后一道防线。

在 课程体系优化层面，将项目中的实战经验反哺于教学。通过引入 openGauss 的企业级特性（如列存储、三权分立、性能分析），我建议将课程重心从单纯的 SQL 语法教学转移到企业级架构与高性能调优上来。这一转变不仅填补了理论与实践的鸿沟，更紧扣国产化信创的时代需求。

综上所述，本项目超越了简单的数据更新任务，它是一次关于数据完整性、系统性能与架构设计的完整实践。它表明，在现代计算机科学教育中，培养学生解决复杂工程问题的能力与培养理论基础同等重要。