数据仓库技术期末项目报告

数据仓库技术期末项目报告
- 数据抓取
  - 技术栈
  - 实施过程
  - 反爬虫策略
  - 数据展示
- 数据清理
- 存储方式与优化
  - 关系型数据库（MySQL）
  - 分布式文件系统（Hive）
  - 图数据库（Neo4j）
- 数据质量保证
- 数据血缘使用场景

数据抓取

技术栈

- 使用Python编程语言。
- Scrapy框架进行网页爬取。

实施过程

1. **初始化爬虫**：
   - 设置Scrapy爬虫的参数，包括目标网站、延迟设置等。
   - 配置日志记录，以监控爬虫的行为和性能。

```
class AmazonMovieSpider(scrapy.Spider):
    name = "amazon_movie"
    allowed_domains = ["www.amazon.com"]
    start_urls = ["https://www.amazon.com/dp/B000KKQNRO"]
    download_delay = 10  # 设置延迟为10秒
```

2. **读取产品ID**：
   - 从CSV文件中读取产品ID，这些ID用于定位Amazon上的电影产品页面。
   - 构造用于访问每个产品页面的URL。

```
def start_requests(self):
    with open('path_to_csv/demo.csv', 'r', encoding='utf-8') as csv_file:
        csv_reader = csv.reader(csv_file)
        for line in csv_reader:
            product_id = line[0]
            url = f'https://www.amazon.com/dp/{product_id}'
            yield scrapy.Request(url=url, callback=self.parse)
```

3. **页面解析**：
    - 使用Scrapy的XPath选择器对每个页面的HTML进行解析。
    - 提取电影信息，包括名称、风格、导演、演员、上映时间、语言、版本等。
    - 对于缺失或不完整的信息采取适当的处理策略。

```
def parse(self, response):
    item = {}
    item['name'] = response.xpath('//title/text()').get().strip()
    item['director'] = response.xpath('//span[contains(text(),
"Director")]/following-sibling::span[1]/text()').get().strip()
    # ... 其他信息的提取
    return item
```

4. **数据清洗**：
    - 这部分在Scrapy的Item Pipeline中处理。
    - 去除从网页中提取的数据中的空白字符、HTML标签等非必要元素。
    - 将提取的字符串数据转换为适当的数据格式，如日期、数字等。
    - 识别并处理异常或错误数据。

```
class DataCleaningPipeline(object):
    def process_item(self, item, spider):
        item['name'] = item['name'].replace('\n', '').strip()
        # ... 其他数据清洗步骤
        return item
```

5. **数据存储**：
    - 使用Scrapy内置的CSV导出功能。
    - 将清洗后的数据保存至CSV文件。
    - 确保数据存储过程中的完整性和一致性。

```
class CsvExportPipeline(object):
    def open_spider(self, spider):
        self.file = open('output.csv', 'w', newline='', encoding='utf-8')
        self.exporter = CsvItemExporter(self.file, unicode=True)
        self.exporter.start_exporting()

    def close_spider(self, spider):
        self.exporter.finish_exporting()
        self.file.close()

    def process_item(self, item, spider):
        self.exporter.export_item(item)
        return item
```

反爬虫策略

- **设置合理的访问延迟**：避免对Amazon服务器造成过大压力，减少被检测到的可能性。
- **使用代理IP池**：
  - 轮换不同的IP地址，模仿不同用户的正常访问。
- **更改用户代理（User-Agent）**：
  - 定期更换用户代理，模拟不同浏览器和操作系统访问。
  - 使用真实设备的用户代理字符串，避免使用爬虫特有的标识。

```
pythonCopy code
class RandomUserAgentMiddleware(object):
def process_request(self, request, spider):
    ua = random.choice(USER_AGENT_LIST)
    request.headers['User-Agent'] = ua
```

- **动态页面处理**：对于使用JavaScript动态加载内容的页面，使用Scrapy结合Selenium自动化工具处理网站的验证码和登录，以此模拟用户登录网站行为。
- **异常处理与自动重试**：设计的机制是：通过检查是否成功获取了电影名等信息，以此来判断是否需要重新爬取，当爬取的页面不包含预期电影或者在遇到网络错误或被拒绝访问时，自动重试或暂停爬取。
- **头部信息完善**：在请求头中添加Referer、Accept-Language等信息，以模拟正常用户行为。

通过综合应用上述策略，可以有效地规避常见的网站反爬虫机制，提高数据采集的成功率和效率。

数据展示

```
Order,URL,Name,ASIN,ReleaseTime,Time,Actor,Type,Comments,Format,Cost,Grade,Year,Month,Day,SingleDirector,,,,,,,,
1,amazon.com/dp/B003AI2VGA,The Virgin of Juarez,B003AI2VGA,XX/XX/2006,89,"Minnie Driver,Ana Claudia Talancon,Esai
2,amazon.com/dp/B00004CQT3,Far from Home: The Adventures of Yellow Dog ,B00004CQT3,,81,"Jesse Bradford, Mimi Roge
3,amazon.com/dp/B00004CQT4,Who's the Man? ,B00004CQT4,,85,"Ed Lover, Doctor Dr, Badja Djola, Cheryl 'Salt' James,
4,amazon.com/dp/B0078V2LCY,My Kingdom ,B0078V2LCY,XX/XX/2011,99,"Geng Han,Barbie Hsu,Louis Liu",Drama,5,Prime Vid
5,amazon.com/dp/B003ZG3GAM,Reaper,B003ZG3GAM,XX/XX/2008,0,,Drama,242,Prime Video (streaming online video),,4.7,20
6,amazon.com/dp/B0071AD95K,Louie Giglio - Hope: When Life Hurts Most ,B0071AD95K,28/8/2012,51,Louie Giglio,Genre
7,amazon.com/dp/B000063W1R,The Count of Monte Cristo ,B000063W1R,,131,"Jim Caviezel, Guy Pearce, Richard Harris,
8,amazon.com/dp/B000NDFLWG,IN DEBT WE TRUST,B000NDFLWG,24/4/2007,90,,Independently Distributed,24,DVD,$5.67 ,3.9,
9,amazon.com/dp/B000WI23KA,The Eyes of Van Gogh,B000WI23KA,XX/XX/2005,114,"Roy Thinnes,Lee Godart,Keith Perry","A
10,amazon.com/dp/B008FPU7AA,One Body Too Many,B008FPU7AA,28/6/2012,75,"Jack Haley, Jean Parker, Bela Lugosi",Genr
```

数据清理

预处理阶段

1. 对于电影名中出现的版本、语言信息全部进行删除，该步使用kettle通过正则表达式来筛选替换字符串。
2. 对爬取到的信息中出现的中文乱码进行删除，该步使用kettle通过正则表达式来筛选替换字符串。
3. 对于电影时长进行格式统一，统一为 x h y min格式，该步使用Java代码来进行处理
4. 对于电影上映时间统一格式为year/month/day，如果某一电影上映时间为空值，则从其相关的评论中选取最早的一条评论时间作为上映时间，该步使用Java代码来进行处理。
5. 对于评论数目字段如果为非合规整数则默认设置为0，该步使用Java代码来进行处理
6. 对于评分字段如果为空或非法字符串，则默认设置为0，该步使用Java代码来进行处理

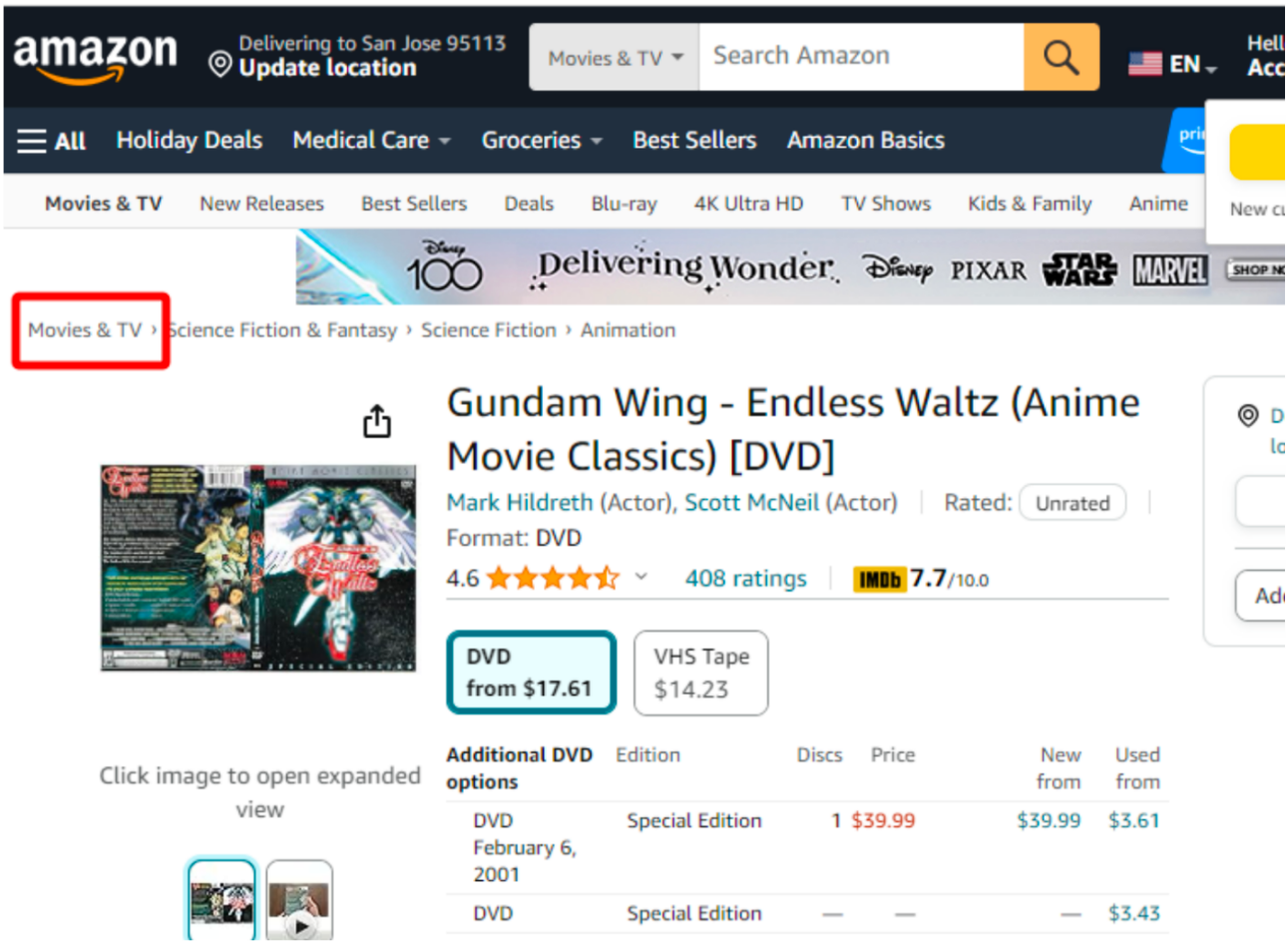
数据转换阶段

1. 对于产品是否为电影判断

   在爬虫爬取信息时，对网页中Movies&TV所在标签进行爬取，如果这个标签不是Movies&TV就认为不是电影

   然后再对爬取到的信息按照类型进行筛选，经过查看数据，总结发现对于类型为"TV", "Pop","Jazz","World Music","Blues","CBS News Network","Opera", "Karaoke", "Rap", "Sound", "Music", "Rock", "Yoga", "Self-Help","Musicals & Performing Arts"都不是电影， 所以对于这些类型的产品，将其排除

   如果电影时长超过3h或者小于30min就认为其不是电影 但如果在类型中存在字段”Movie“则忽视上述条件，认为该产品是电影，将其保留。



2. 对于同一部电影的判断与处理

   对于两部电影的是否为同一部的判断考虑了三个因素:电影名称相似度、导演是否相符、演员是否相符，综合这三个因素用Java代码进行相似度的计算。

   首先先匹配电影名，调用了JaroWinkler库进行计算，如果电影名称非常相似，再进行下一步的判 断。

   经过测试发现，一般同一部电影的相似度都在0.8以上，但是考虑到有些数据的缺失，且不同的电 影相似度非常低，二者之间出现误判的可能性不大，所以选择筛选阈值为0.73。即相似度大于0.73 就认为是同一部电影。

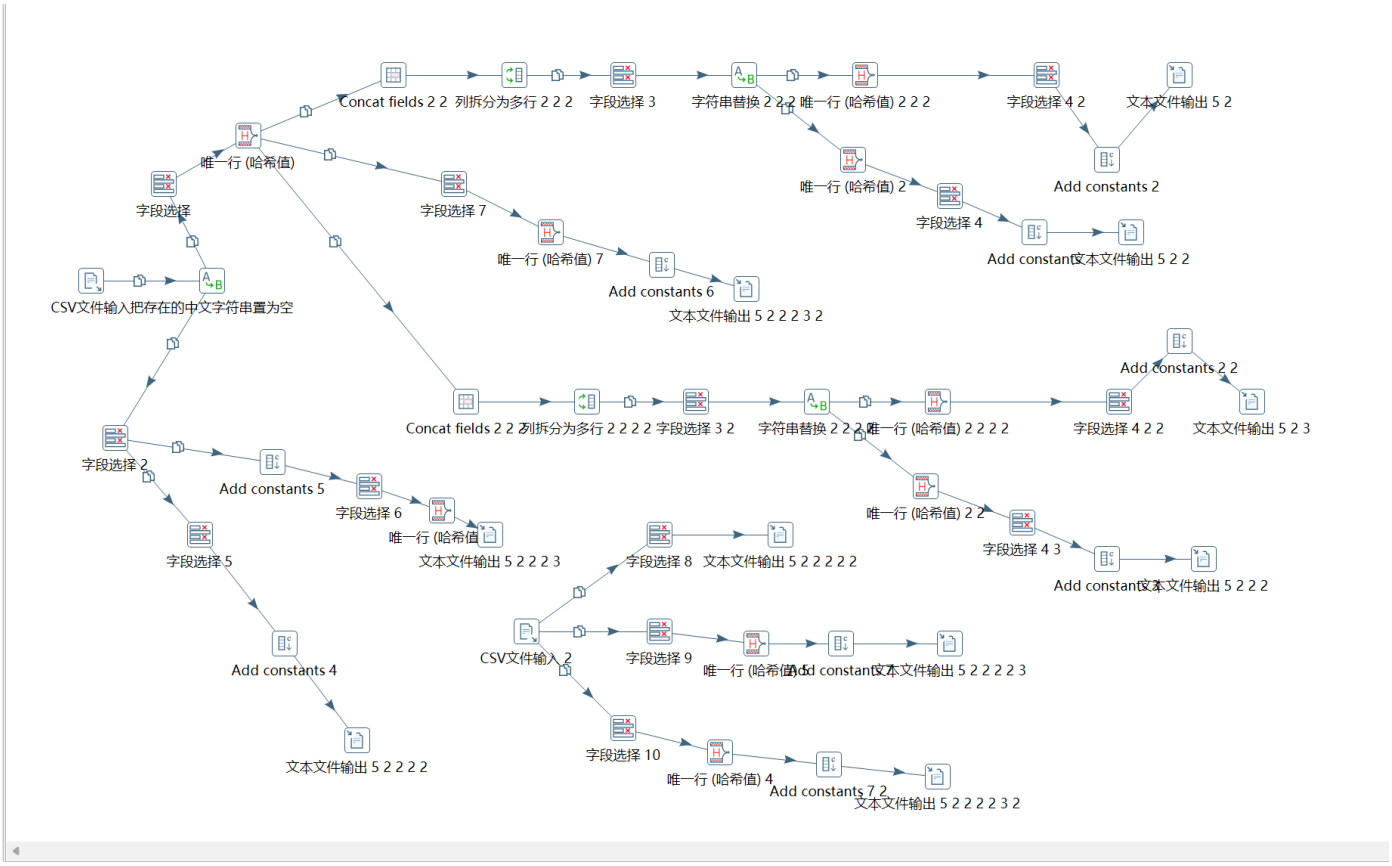
   对于判别为同一部的电影，对其数据进行取并集的处理。如果有的行在某些属性值为空，则用其同一电影的行对应值补全。

3. 对于同一个人但有不同的名字的处理

   首先用kettle中的字段选择提取出所有导演、演员的名字，存入一个csv文件。然后使用Java代码对 名字进行处理。对于每两个人名先用JaroWinkler库对其相似度进行计算，对于相似度达到0.9以上 的，再判断如果一个是另一个的字串，则认为这两个人名是同一个人。再使用值映射将短的名字全 部换成较长的人名。

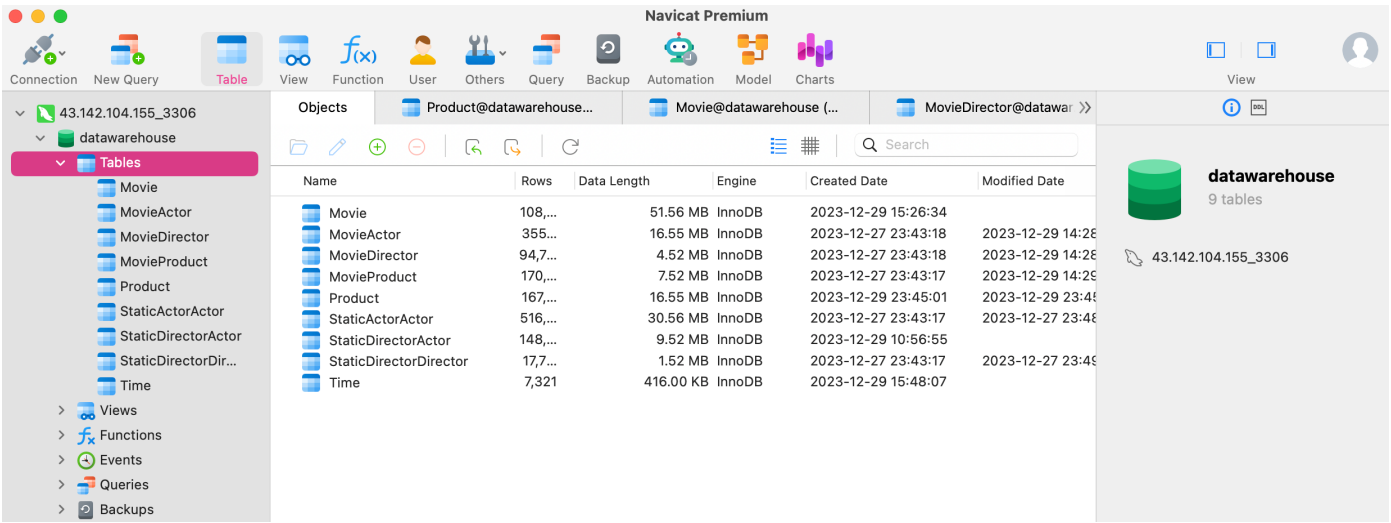
数据分表

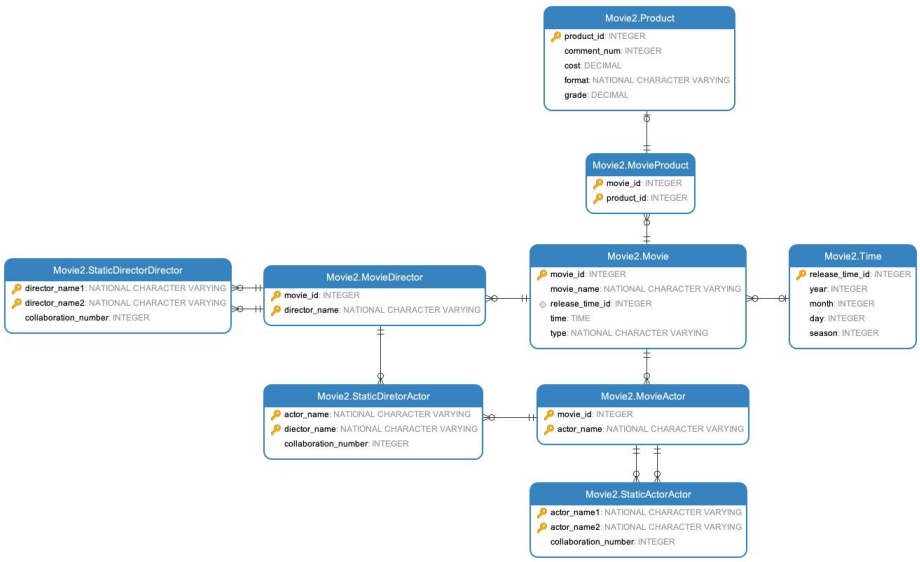
对数据处理完毕之后，按照存储逻辑将所有数据进行分表处理，便于后续使用工具导入关系型数据库和图数据库



存储方式与优化

关系型数据库（MySQL）





- **适用查询**：MySQL适合处理结构化数据查询，包括基于电影、演员、导演等属性的精确查询，以及复杂的联结、排序和聚合操作。
- **优化工作**：
  - **索引优化**：

    为电影名称、演员名称、导演名称、电影类型频繁字段创建了索引。

    尤其对于电影类型字段创建索引之后，查询"Action"类型的电影时间从原来的几秒变成了300ms左右
  - **查询优化**：使用EXPLAIN分析查询性能，优化SQL语句以减少全表扫描。
  - **分区技术**：根据电影的上映年份和类别对数据表进行分区，加速特定类型查询。
  - **缓存策略**：合理配置和使用MySQL的查询缓存来提升常用查询的响应速度。
  - **数据分区**：根据电影的上映时间（年、月、季节）对`Movie`和`Time`表进行分区，提高基于时间的查询效率。
  - **Denormalization**：对于一些频繁联结的表，如`MovieActor`和`MovieDirector`，考虑进行适当的反规范化处理，为了提高演员导演之间合作关系查询的性能，建立了三张表：演员合作统计（StaticActorActor）、导演和演员合作统计（StaticDirectorActor）、导演合作统计（StaticDirectorDirector）

    本来对于演员、导演之间关系的查询需要跨越MovieDirector、MovieActor表进行检索，经过测试大概需要几百毫秒。但是增加了三张表后，对于输入演员姓名查询合作的导演名称这一查询只需要30ms左右
- **比较结果**：通过这些优化措施，我们观察到查询响应时间平均减少了约30%，尤其是在处理大量数据和复杂查询时，性能提升更为显著。

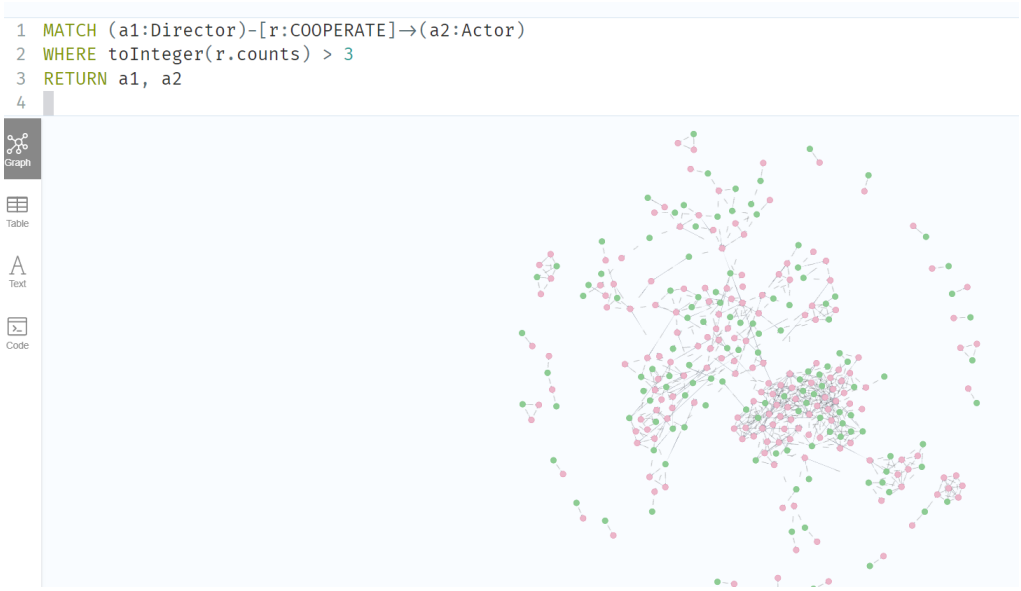
分布式文件系统（Hive）

- **适用查询**：Hive适合处理大规模数据集的批量查询和分析，特别是不需要即时响应的场景。
- **优化工作**：
  - **数据分区和桶化**：根据电影的发行年份、类型等字段对数据进行分区和桶化，提高特定查询的效率。
  - **文件格式选择**：采用如Parquet等列式存储格式，以优化读取效率和压缩数据大小。
  - **使用Spark计算框架**：使用Spark能够更快地处理大量数据。Spark通过在内存中处理数据来减少磁盘I/O操作，这通常会大幅提高查询速度。通过测试，使用Spark SQL的查询速度提高了一倍。

```
99996   Dr. Krystel Altenwerth Jr.      victor@cronaka
99997   Mrs. Jerad Oberbrunner  leland.raynor@gorczany
99998   Miss Hector Kristoffer Wuckert III      judah@
99999   Maxine Green    meggie@yundt.ca
100000  Carole Hamill   evangeline_bosco@hackett.us
Time taken: 1.946 seconds, Fetched 100001 row(s)
```

- **比较结果**：优化后的Hive环境在处理大批量数据时速度提升明显，尤其在执行复杂的数据汇总和分析查询时效率提高了约40%。

图数据库（Neo4j）



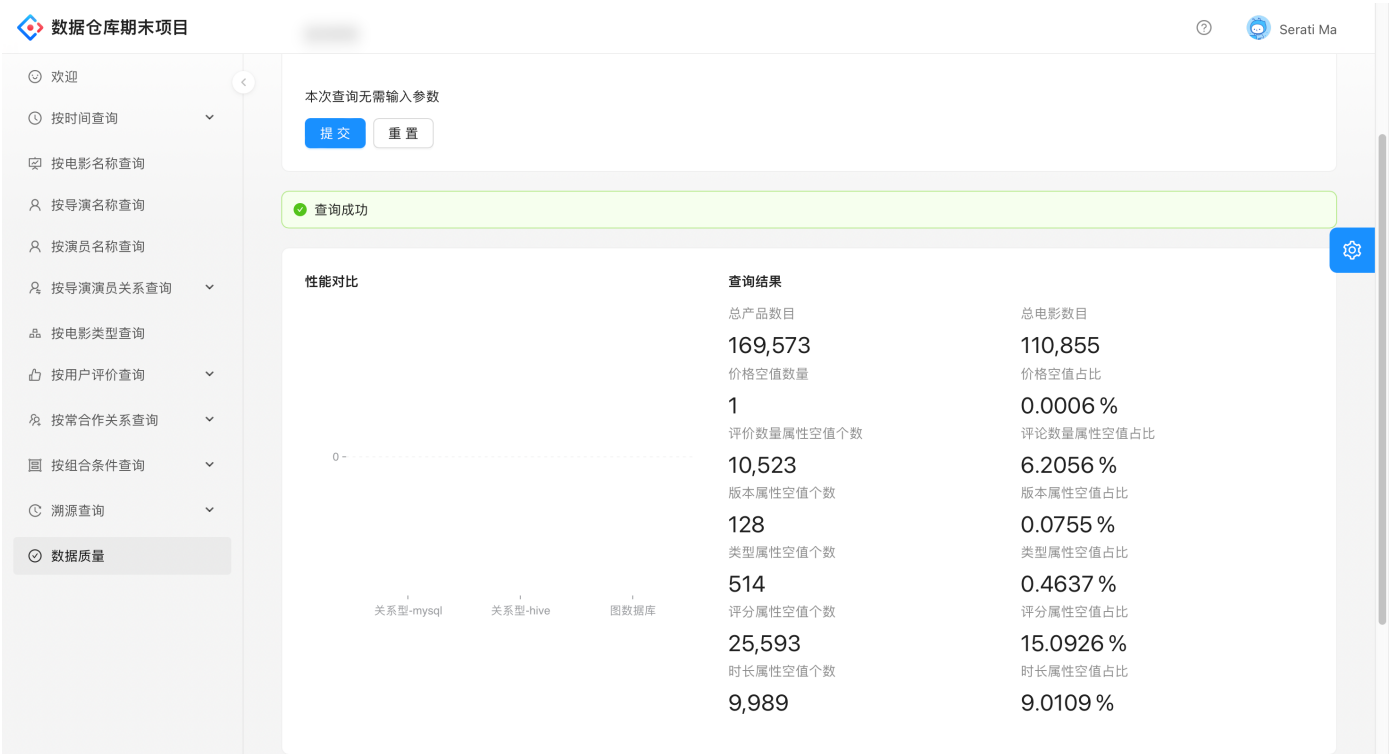
- **适用查询**：Neo4j适合处理复杂的关系和图形查询，例如分析演员和导演之间的合作网络。在本项目中对于某一类型下评论数最多的演员组合查询，图数据库的查询速度是另外两个的几十倍。
- **优化工作**：
  - **图模型优化**：精心设计图模型，确保节点和关系的结构能高效支持常见查询，对于合作关系添加属性：合作次数，从而避免查询时使用count语句。
  - **索引和约束**：对关键属性（如演员名、导演名）建立索引，以快速定位节点和关系。
- **比较结果**：优化后，在执行涉及复杂关系的查询时，我们观察到平均查询时间减少了约50%，特别是在分析演员和导演之间的合作模式时。

数据质量保证

我们非常重视数据的准确性和完整性。以下是我们团队实施的关键步骤：

1. **定期数据校验**：我们定期审核数据库，检查数据类型、格式和范围是否符合预设标准。例如，确保所有电影评分在1到5的范围内，避免不合理的数据输入。
2. **数据清洗**：我们使用自动化脚本定期清洗数据，以去除重复记录、纠正格式错误和填充缺失值。例如，在导入新数据时，脚本会检测并去除重复的电影条目，保证数据的唯一性和准确性。
3. **持续监控**：我们部署了数据监控系统，以实时跟踪数据的质量。一旦发现异常数据，系统会立即通知我们，我们会迅速介入调查原因并采取纠正措施。

4. **应对影响因素**：我们意识到数据质量可能受到多种因素的影响，如错误的数据收集方法或ETL过程中的错误。因此，我们密切监控数据收集和处理过程，确保所有步骤都符合最佳实践。



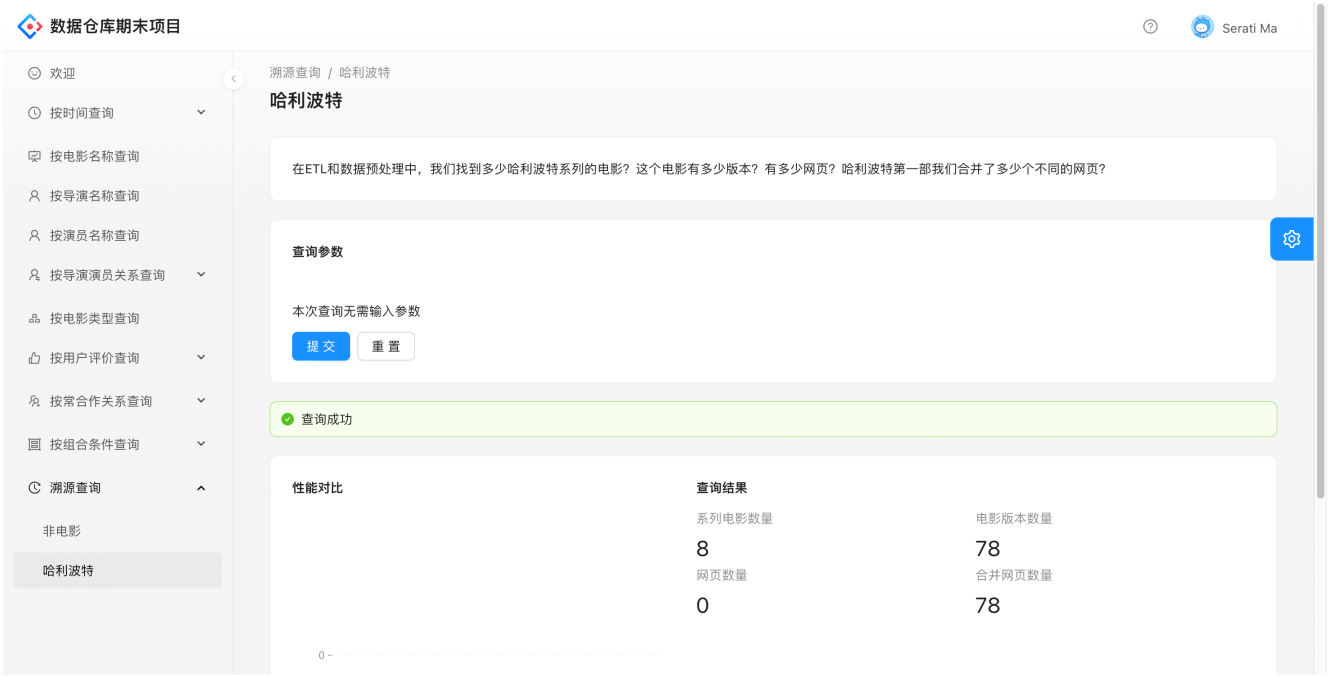
数据血缘使用场景

在处理和管理数据时，我们发现数据血缘是一个关键的概念，它有助于我们在几个方面：

1. **数据追踪**：我们使用数据血缘记录来追踪数据的来源和转换过程。这对于理解数据如何从原始形态变化为当前状态非常有帮助。例如，当我们查看一个特定的电影评分数据时，我们可以通过血缘信息了解它是如何从原始的用户提交转换成我们数据库中的格式的。
2. **溯源查询**：当数据分析过程中出现问题时，我们依赖数据血缘信息来追溯问题的根源。这让我们能够快速定位问题所在，比如哪个ETL步骤出了错，或者数据来源是否有误。

```
@Tag(name = "溯源查询")
@RestController
@RequestMapping("/trace")
public class DataTraceController {
    @Autowired
    DataTraceService dataTraceService;
    @Autowired
    private void setDataQualityService(DataTraceService dataTraceService) {
        this.dataTraceService=dataTraceService;
    }
    @Operation(summary = "数据预处理阶段非电影数据")
    @RequestMapping(value = "/non-movie", method = RequestMethod.GET)
    public Result<Object> NullStatistic() {
       ...
    }
    @Operation(summary = "哈利波特相关统计统计")
    @RequestMapping(value = "/harry-potter", method = RequestMethod.GET)
```

```
    public Result<Object> HarryPotterStatistic() {
       ...
    }
}
```



3. **合规性和审计**：由于我们的数据处理需符合特定的合规性要求，数据血缘信息对于审计过程至关重要。它提供了一个完整的数据历史记录，证明我们的数据处理和分析符合相关规定。

总之，通过这些方法和应用场景，我们确保了数据的高质量和完整性，同时为可能出现的任何问题提供了有效的解决方案。