

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1



BÀI TẬP LỚN MÔN:
CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên hướng dẫn : KIM NGỌC BÁCH
Nhóm lớp : INT14148 - 20242 - 10
Nhóm thực hiện : 15

Sinh viên:
Trần Huy Hoàng
Nguyễn Doãn Huy
Trần Đức Hoàng

Mã sinh viên:
B22DCCN348
B22DCCN384
B22DCCN347

MỤC LỤC

| | |
|---|-----------|
| 1. GIỚI THIỆU..... | 2 |
| 2. CƠ SỞ LÝ THUYẾT..... | 3 |
| 2.1. Phân mảnh dữ liệu trong hệ thống phân tán..... | 3 |
| 2.2. Phân mảnh ngang (Horizontal Partitioning) | 3 |
| 2.3. Phân mảnh theo khoảng (Range Partitioning) | 4 |
| 2.4. Phân mảnh vòng tròn (Round-Robin Partitioning) | 5 |
| 2.5. So sánh các phương pháp phân mảnh | 7 |
| 3. THIẾT LẬP MÔI TRƯỜNG VÀ CÀI ĐẶT | 7 |
| 3.1. Yêu cầu hệ thống..... | 7 |
| 3.2. Cài đặt PostgreSQL..... | 8 |
| 3.3. Cài đặt Python và thư viện cần thiết | 10 |
| 3.4. Chuẩn bị tập dữ liệu MovieLens..... | 12 |
| 4. PHÂN TÍCH VÀ THIẾT KẾ..... | 14 |
| 4.1. Kiến trúc tổng thể..... | 14 |
| 4.2. Cấu trúc cơ sở dữ liệu | 15 |
| 4.3. Thiết kế các hàm | 16 |
| 5. CHI TIẾT TRIỂN KHAI | 17 |
| 5.1. Tổng quan về các hàm | 17 |
| 5.2. Hàm tải dữ liệu (LoadRatings)..... | 17 |
| 5.3. Hàm phân mảnh theo khoảng (Range_Partition)..... | 19 |
| 5.4. Hàm chèn theo khoảng (Range_Insert)..... | 21 |
| 5.5. Hàm phân mảnh vòng tròn (RoundRobin_Partition)..... | 23 |
| 5.6. Hàm chèn vòng tròn (RoundRobin_Insert)..... | 25 |
| 5.7 Các hàm trợ giúp | 27 |
| 6. KIỂM THỬ VÀ ĐÁNH GIÁ | 28 |
| 7. KẾT LUẬN..... | 34 |
| 8. PHÂN CÔNG CÔNG VIỆC | 36 |
| 9. TÀI LIỆU THAM KHẢO | 36 |

1. GIỚI THIỆU

1.1. Tổng quan về bài tập

Bài tập lớn này yêu cầu mô phỏng các phương pháp phân mảnh dữ liệu trên hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở PostgreSQL. Nhiệm vụ chính là tạo một tập các hàm Python để:

- Tải dữ liệu đánh giá phim từ tập dữ liệu MovieLens vào một bảng quan hệ
- Phân mảnh bảng này bằng các phương pháp phân mảnh ngang khác nhau
- Chèn các bộ dữ liệu mới vào đúng phân mảnh

Bài tập này giúp hiểu rõ hơn về các kỹ thuật phân mảnh dữ liệu trong môi trường phân tán, một khía cạnh quan trọng trong thiết kế và quản lý cơ sở dữ liệu hiện đại.

1.2. Mục tiêu

Mục tiêu chính của bài tập:

- Triển khai và hiểu rõ các kỹ thuật phân mảnh ngang trong cơ sở dữ liệu
- Áp dụng phân mảnh theo khoảng (Range Partitioning) và phân mảnh vòng tròn (Round-Robin Partitioning)
- Phát triển kỹ năng làm việc với PostgreSQL và Python
- Hiểu cách quản lý và truy xuất dữ liệu trên các phân mảnh
- Đánh giá hiệu suất của các phương pháp phân mảnh khác nhau

1.3. Dữ liệu đầu vào

Dữ liệu đầu vào là tập dữ liệu đánh giá phim từ MovieLens. Chi tiết về tập dữ liệu:

- Chứa 10 triệu đánh giá và 100.000 thẻ từ 72.000 người dùng về 10.000 bộ phim
- Mỗi dòng có định dạng: UserID::MovieID::Rating::Timestamp
- Đánh giá được thực hiện trên thang điểm 5 sao, có thể chia nửa sao
- Timestamp là số giây kể từ nửa đêm UTC ngày 1/1/1970

Ví dụ về dữ liệu:

```
test_data.dat
1 1::122::5::838985046
2 1::185::4.5::838983525
3 1::231::4::838983392
4 1::292::3.5::838983421
5 1::316::3::838983392
```

Hình 1.1: Dữ liệu mẫu tập dữ liệu đánh giá phim từ MovieLens

2. CƠ SỞ LÝ THUYẾT

2.1. Phân mảnh dữ liệu trong hệ thống phân tán

Phân mảnh dữ liệu (Data Partitioning) là một kỹ thuật quan trọng trong hệ thống cơ sở dữ liệu phân tán, giúp chia nhỏ các bảng lớn thành các phần nhỏ hơn, để quản lý hơn gọi là phân mảnh (partitions). Kỹ thuật này mang lại nhiều lợi ích:

- **Cải thiện hiệu suất:** Giảm thời gian truy vấn do chỉ cần quét một phần dữ liệu
- **Tăng khả năng mở rộng:** Dễ dàng thêm mới hoặc mở rộng các phân mảnh
- **Tối ưu hóa bảo trì:** Dễ dàng sao lưu, khôi phục hoặc tái cấu trúc các phân mảnh riêng lẻ
- **Phân tán tải:** Phân phối tải truy vấn trên nhiều nút khác nhau
- **Tăng tính sẵn sàng:** Hệ thống vẫn hoạt động ngay cả khi một số phân mảnh gặp sự cố

Các loại phân mảnh cơ bản trong hệ thống phân tán:

1. **Phân mảnh ngang (Horizontal Partitioning):** Chia các hàng của bảng thành các nhóm riêng biệt
2. **Phân mảnh dọc (Vertical Partitioning):** Chia các cột của bảng thành các nhóm riêng biệt
3. **Phân mảnh hỗn hợp (Hybrid Partitioning):** Kết hợp cả phân mảnh ngang và dọc

Trong bài tập này, chúng ta tập trung vào phân mảnh ngang với hai kỹ thuật cụ thể: phân mảnh theo khoảng và phân mảnh vòng tròn.

2.2. Phân mảnh ngang (Horizontal Partitioning)

Phân mảnh ngang chia các hàng của bảng thành các tập con riêng biệt, mỗi tập con được lưu trữ trong một bảng riêng biệt với cùng cấu trúc (schema) như bảng gốc. Phân mảnh ngang có các đặc điểm:

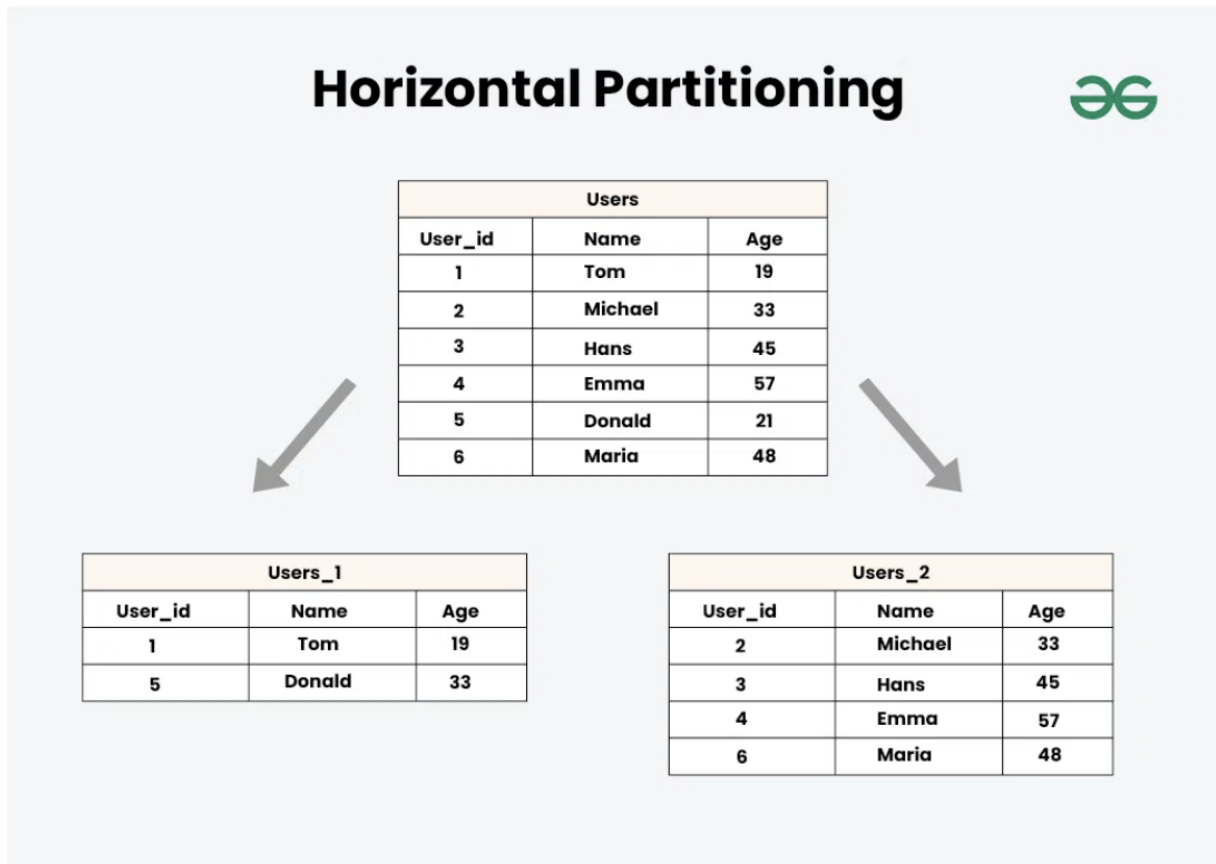
- Mỗi phân mảnh chứa một tập con các hàng từ bảng gốc
- Tất cả các phân mảnh có cùng cấu trúc (cùng số lượng và kiểu dữ liệu của các cột)
- Mỗi hàng chỉ xuất hiện trong một phân mảnh duy nhất
- Tất cả các phân mảnh cùng nhau chứa toàn bộ dữ liệu từ bảng gốc

Phân mảnh ngang đặc biệt hữu ích khi:

- Các truy vấn thường chỉ truy cập một phần nhỏ của bảng
- Dữ liệu có thể được phân loại rõ ràng theo một số tiêu chí
- Bảng có kích thước lớn, gây khó khăn khi quản lý dưới dạng một đơn vị duy nhất

Trong phân mảnh ngang, cần đảm bảo ba tính chất quan trọng:

1. **Tính đầy đủ (Completeness):** Mọi bộ dữ liệu trong bảng gốc phải được gán cho ít nhất một phân mảnh
2. **Tính không giao (Disjointness):** Mỗi bộ dữ liệu chỉ thuộc về một phân mảnh duy nhất (trừ khi sử dụng khóa chính)
3. **Tính tái tạo (Reconstructability):** Phải có khả năng tái tạo bảng gốc từ các phân mảnh



Hình 2.1: Minh họa phân mảnh ngang (Horizontal Partitioning)

2.3. Phân mảnh theo khoảng (Range Partitioning)

Phân mảnh theo khoảng (Range Partitioning) là một kỹ thuật phân mảnh ngang phổ biến, trong đó dữ liệu được phân chia dựa trên khoảng giá trị của một cột cụ thể. Phương pháp này hoạt động như sau:

- Xác định một cột làm cơ sở phân mảnh (thường là cột có tính phân phối đều như ngày tháng, ID, giá trị số)
- Chia phạm vi giá trị của cột này thành các khoảng không chồng chéo
- Mỗi phân mảnh chứa các hàng có giá trị thuộc về một khoảng cụ thể

Trong bài tập này, chúng ta sử dụng cột rating (với giá trị từ 0 đến 5) làm cơ sở cho phân mảnh theo khoảng. Nếu chia thành N phân mảnh, mỗi phân mảnh sẽ chứa các đánh giá trong một khoảng rộng $5/N$.

Ví dụ với N = 5 phân mảnh:

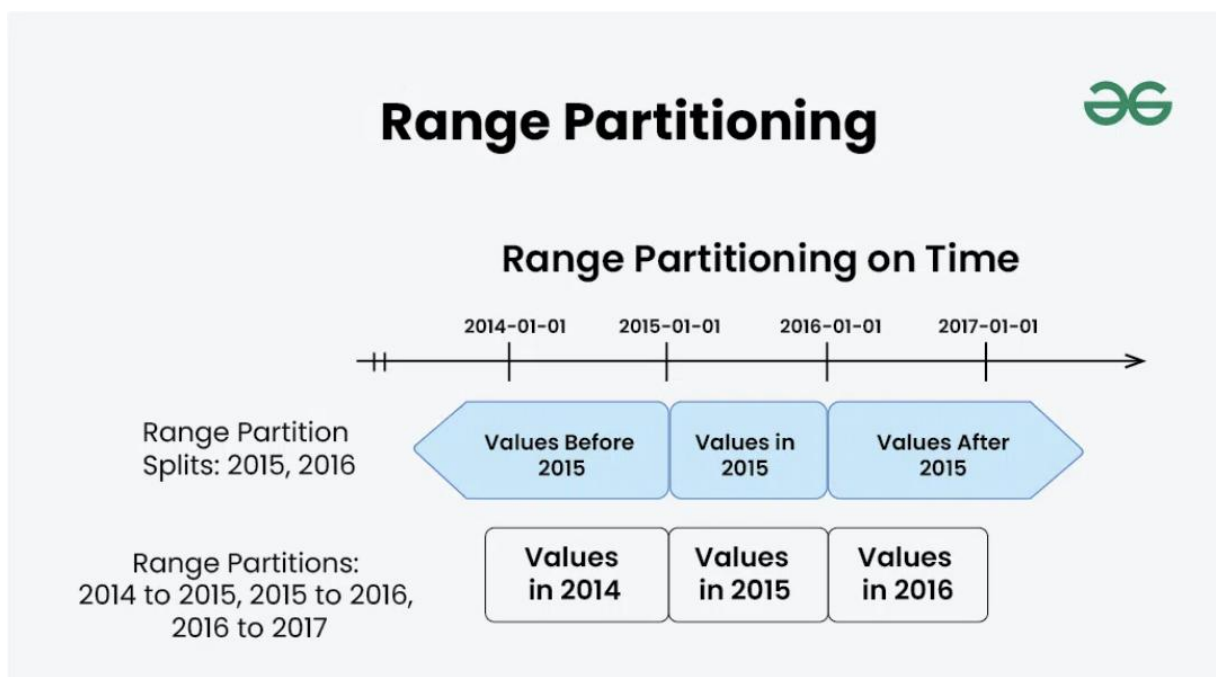
- Phân mảnh 0: Đánh giá từ 0 đến 1
- Phân mảnh 1: Đánh giá trên 1 đến 2
- Phân mảnh 2: Đánh giá trên 2 đến 3
- Phân mảnh 3: Đánh giá trên 3 đến 4
- Phân mảnh 4: Đánh giá trên 4 đến 5

Ưu điểm của phân mảnh theo khoảng:

- Hiệu quả cho các truy vấn tìm kiếm theo khoảng giá trị
- Dễ dàng hiểu và triển khai
- Phù hợp với dữ liệu có tính thời gian hoặc tuần tự

Nhược điểm:

- Có thể dẫn đến phân phối dữ liệu không đồng đều nếu dữ liệu nghiêng về một khoảng giá trị
- Hiệu suất kém hơn nếu truy vấn không dựa trên cột phân mảnh



Hình 2.2: Minh họa phân mảnh theo khoảng (Range Partitioning)

2.4. Phân mảnh vòng tròn (Round-Robin Partitioning)

Phân mảnh vòng tròn (Round-Robin Partitioning) là một kỹ thuật phân mảnh ngang khác, trong đó dữ liệu được phân phối đều giữa các phân mảnh theo thứ tự tuần hoàn. Phương pháp này hoạt động như sau:

- Các hàng được gán cho các phân mảnh theo thứ tự luân phiên
- Hàng thứ i được gán cho phân mảnh $i \bmod N$ (với N là số lượng phân mảnh)
- Không dựa trên giá trị của bất kỳ cột nào trong dữ liệu

Ví dụ với $N = 3$ phân mảnh:

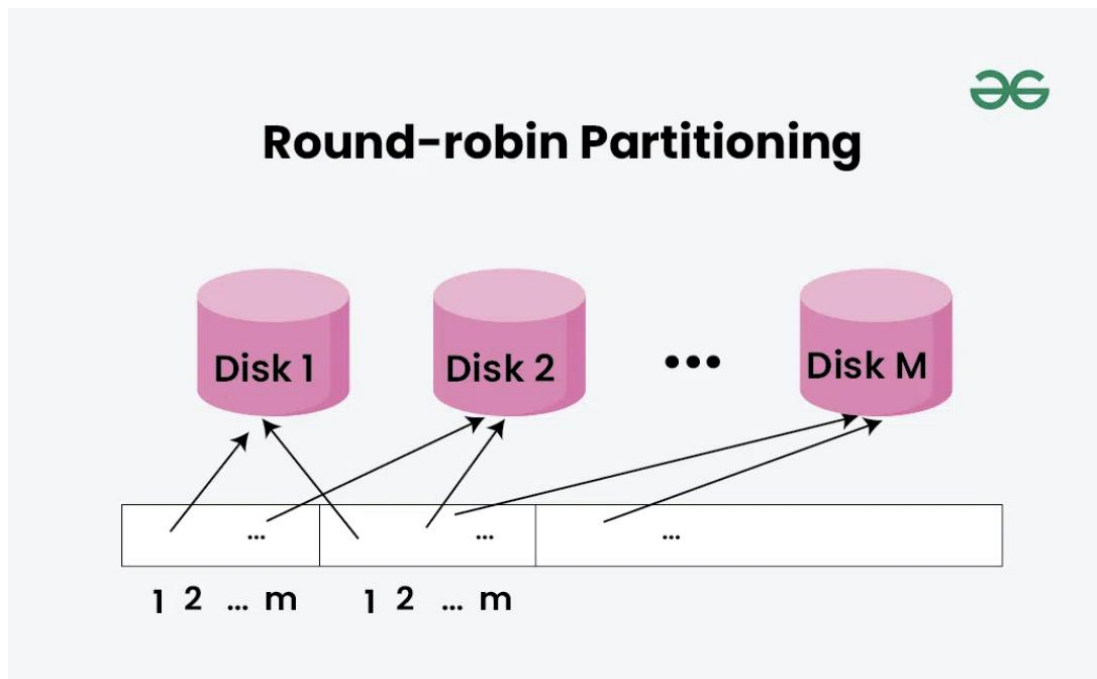
- Hàng 1 → Phân mảnh 0
- Hàng 2 → Phân mảnh 1
- Hàng 3 → Phân mảnh 2
- Hàng 4 → Phân mảnh 0
- Hàng 5 → Phân mảnh 1
- Và cứ tiếp tục như vậy...

Ưu điểm của phân mảnh vòng tròn:

- Đảm bảo phân phối dữ liệu đồng đều giữa các phân mảnh
- Không phụ thuộc vào giá trị dữ liệu
- Tốt cho cân bằng tải trong môi trường phân tán
- Đơn giản để triển khai

Nhược điểm:

- Không hiệu quả cho truy vấn dựa trên giá trị cụ thể (phải quét tất cả các phân mảnh)
- Không phân nhóm dữ liệu liên quan với nhau
- Khó khăn trong việc định vị chính xác phân mảnh chứa một hàng cụ thể nếu không biết số thứ tự của hàng



Hình 2.3: Minh họa phân mảnh vòng tròn (Round-robin Partitioning)

2.5. So sánh các phương pháp phân mảnh

| Tiêu chí | Phân mảnh theo khoảng | Phân mảnh vòng tròn |
|--------------------------|--|---|
| Cơ chế phân phối | Dựa trên khoảng giá trị của một cột | Phân phối tuần hoàn không phụ thuộc giá trị |
| Cân bằng dữ liệu | Có thể không đồng đều nếu dữ liệu nghiêng | Luôn đồng đều giữa các phân mảnh |
| Hiệu quả truy vấn khoảng | Cao (chỉ quét các phân mảnh liên quan) | Thấp (phải quét tất cả phân mảnh) |
| Hiệu quả tìm kiếm điểm | Cao nếu tìm theo cột phân mảnh | Thấp (phải quét tất cả phân mảnh) |
| Khả năng mở rộng | Phức tạp khi thêm phân mảnh mới | Đơn giản khi thêm phân mảnh mới |
| Phù hợp với | Dữ liệu có mẫu truy cập theo khoảng giá trị | Dữ liệu cần phân phối đều để cân bằng tải |
| Phức tạp triển khai | Trung bình | Đơn giản |
| Yêu cầu duy trì | Có thể cần cân bằng lại nếu dữ liệu thay đổi | Ít yêu cầu duy trì hơn |

Việc lựa chọn phương pháp phân mảnh phù hợp phụ thuộc vào:

- Đặc điểm của dữ liệu (phân phối, kích thước)
- Mẫu truy cập (loại truy vấn phổ biến)
- Yêu cầu hiệu suất
- Khả năng mở rộng trong tương lai

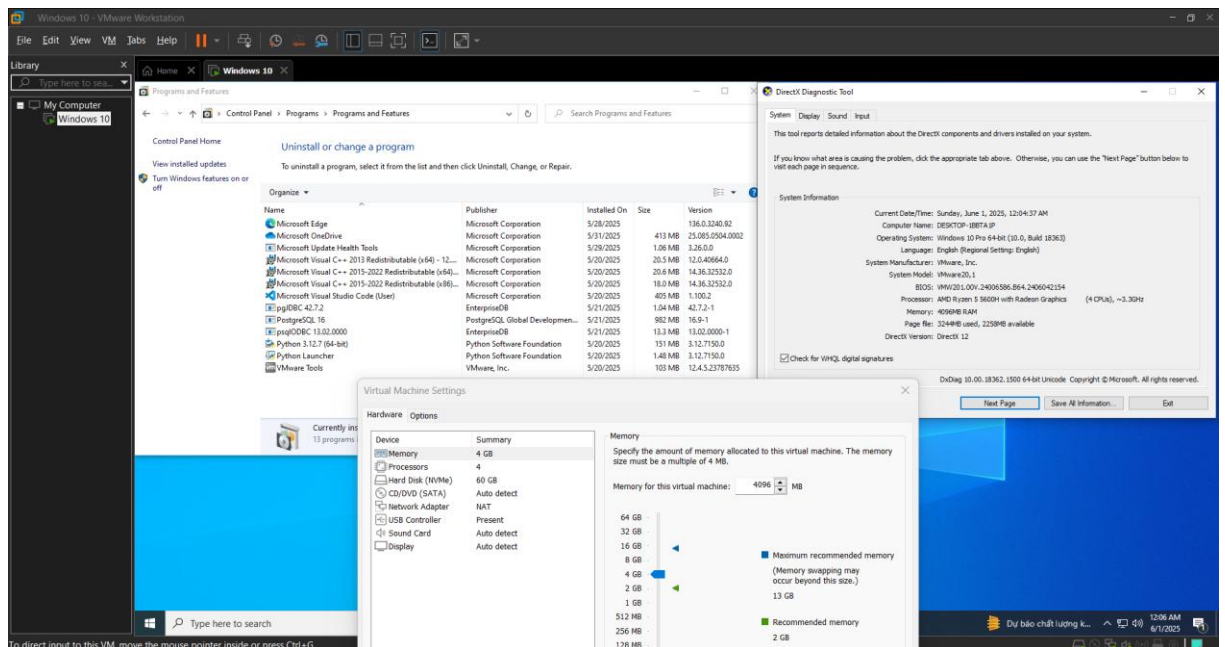
Trong nhiều trường hợp thực tế, các hệ thống sử dụng kết hợp nhiều phương pháp phân mảnh để tối đa hóa lợi ích.

3. THIẾT LẬP MÔI TRƯỜNG VÀ CÀI ĐẶT

3.1. Yêu cầu hệ thống

Để triển khai bài tập này, nhóm em đã sử dụng môi trường sau:

- **Hệ điều hành:** Windows 10 (v1909 x64) chạy trong máy ảo VMware
- **Python:** Phiên bản 3.12.7
- **PostgreSQL:** Phiên bản 16.9
- **Thư viện Python:** psycopg2 (adapter kết nối Python với PostgreSQL)



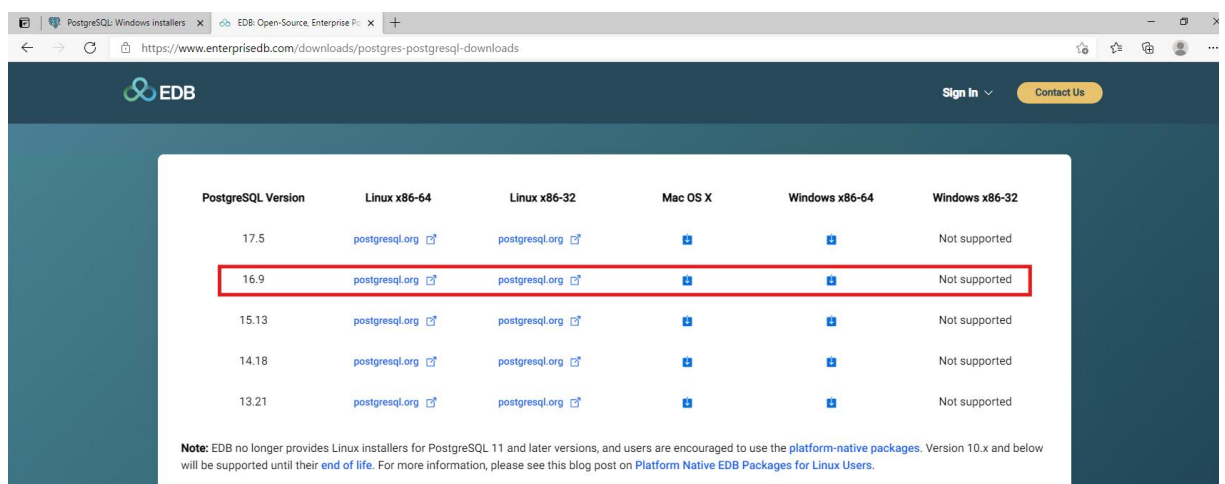
Hình 3.1: Thiết lập môi trường sử dụng máy ảo VMware Workstation

3.2. Cài đặt PostgreSQL

Các bước cài đặt PostgreSQL trên Windows 10:

1. Tải PostgreSQL:

- Truy cập trang web chính thức của PostgreSQL:
<https://www.postgresql.org/download/windows/>
- Hoặc tải trực tiếp từ EnterpriseDB:
<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>
- Lựa chọn phiên bản 16.9 dành cho Windows

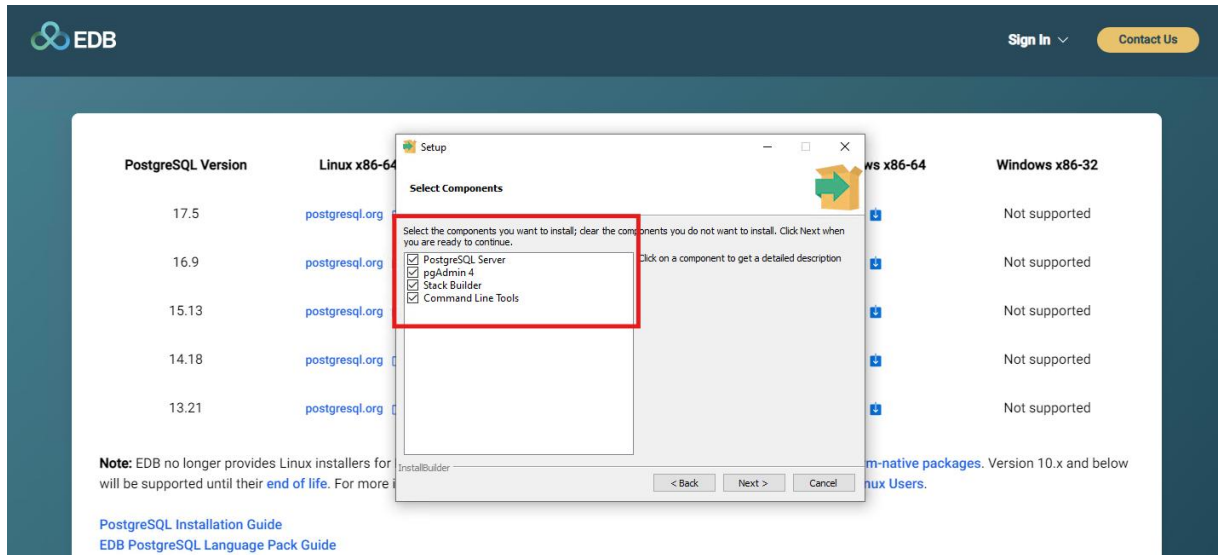


Hình 3.2: Tải PostgreSQL về máy tính

2. Cài đặt PostgreSQL:

- Chạy file cài đặt đã tải về

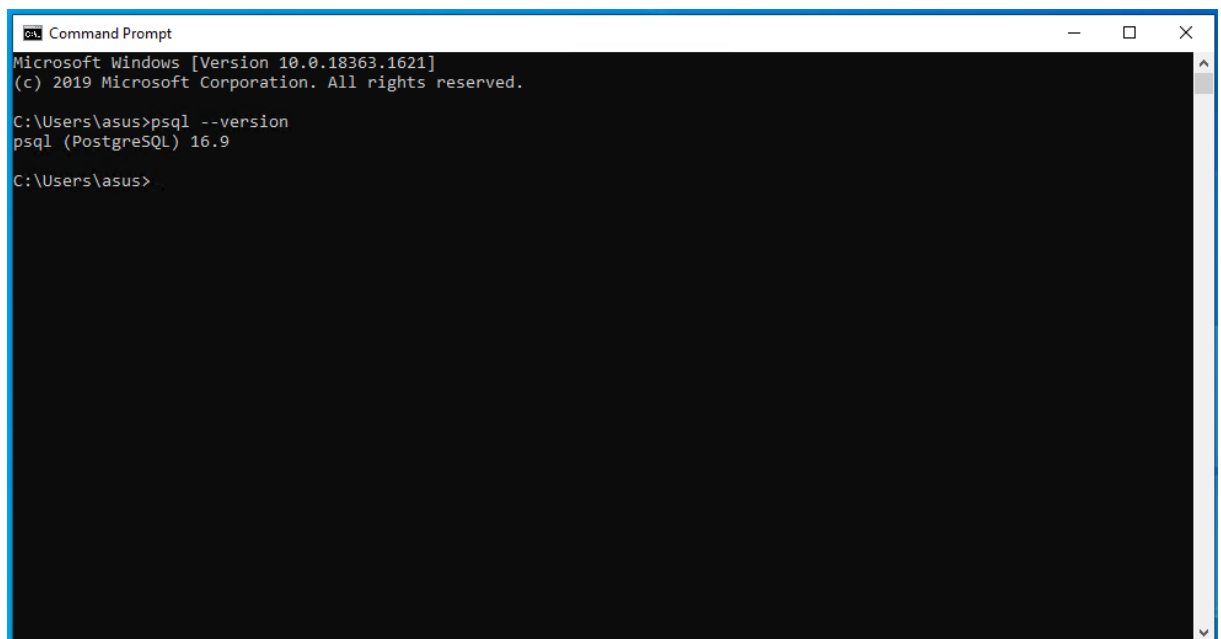
- Chọn các thành phần cần thiết (PostgreSQL Server, pgAdmin, Command Line Tools)
- Đặt mật khẩu cho tài khoản postgres (trong bài tập này, sử dụng mật khẩu '1234')
- Chọn cổng mặc định 5432
- Hoàn tất quá trình cài đặt



Hình 3.3: Cài đặt PostgreSQL cùng các thành phần cần thiết

3. Xác minh cài đặt:

- Mở Command Prompt và thực hiện lệnh: `psql --version`
- Tạo file `test_connection.py` để kiểm tra kết nối với PostgreSQL



Hình 3.4.1: Kiểm tra PostgreSQL đã được cài hay chưa

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1621]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\bt1-csdlpt-nhom-15>python test_connection.py
PostgreSQL version: ('PostgreSQL 16.9, compiled by Visual C++ build 1943, 64-bit',)
Kết nối PostgreSQL thành công!

C:\Users\asus\Downloads\bt1-csdlpt-nhom-15>
```

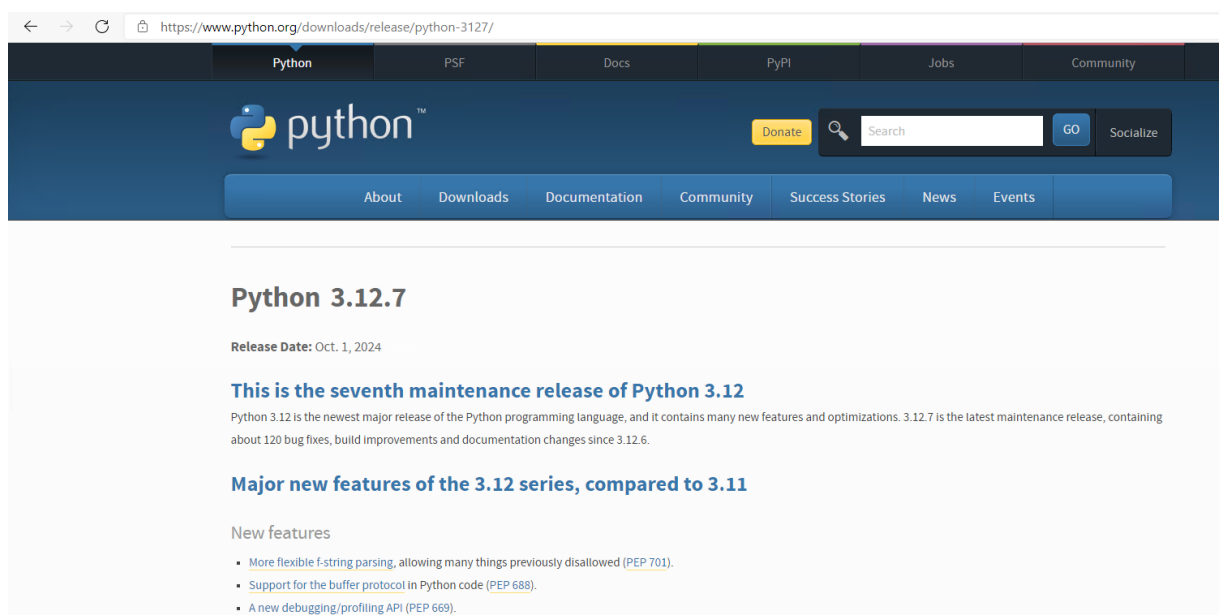
Hình 3.4.2: Kiểm tra kết nối với PostgreSQL

3.3. Cài đặt Python và thư viện cần thiết

Các bước cài đặt Python và thư viện cần thiết:

1. Cài đặt Python:

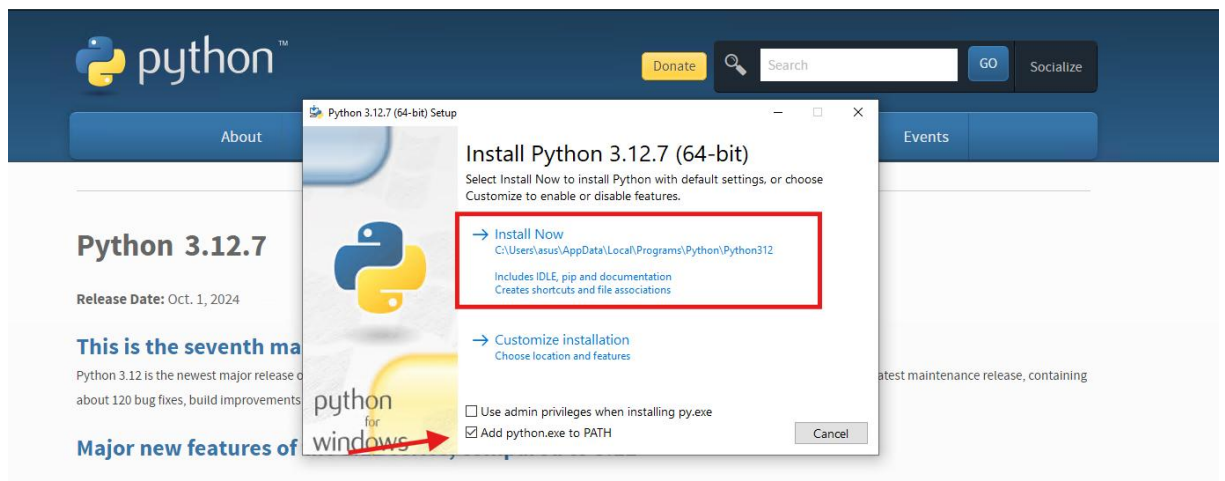
- Truy cập trang web chính thức của Python:
<https://www.python.org/downloads/>
- Tải Python 3.12.7 cho Windows



Hình 3.5: Tìm kiếm và cài Python 3.12.x theo như đề bài

- Chạy bộ cài đặt, đánh dấu tùy chọn "Add Python to PATH"

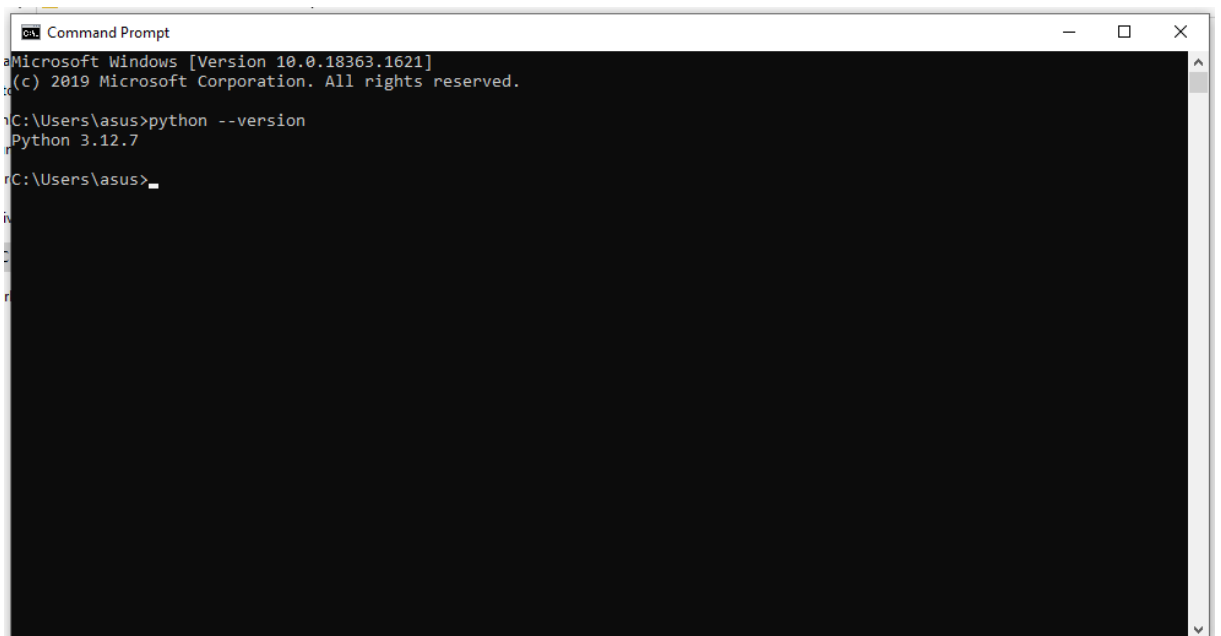
- Hoàn tất cài đặt



Hình 3.6: Cài đặt Python trên máy ảo Windows

2. Xác minh cài đặt Python:

- Mở Command Prompt
- Thực hiện lệnh: `python --version`
- Kết quả hiển thị nên là: Python 3.12.7



Hình 3.7: Kiểm tra Python trên máy ảo Windows

3. Cài đặt thư viện `psycopg2`:

- Mở Command Prompt
- Thực hiện lệnh: `pip install psycopg2`
- Chờ quá trình cài đặt hoàn tất

```
Command Prompt
Microsoft Windows [Version 10.0.18363.1621]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asus>python --version
Python 3.12.7

C:\Users\asus>pip install psycpg2
Collecting psycpg2
  Downloading psycpg2-2.9.10-cp312-cp312-win_amd64.whl.metadata (5.0 kB)
  Downloading psycpg2-2.9.10-cp312-cp312-win_amd64.whl (1.2 MB)
    ----- 1.2/1.2 MB 11.6 MB/s eta 0:00:00
Installing collected packages: psycpg2
Successfully installed psycpg2-2.9.10

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\asus>
```

Hình 3.8: Cài đặt thư viện psycpg2 trên máy ảo Windows

4. Xác minh cài đặt psycpg2:

- Trong Command Prompt, thực hiện lệnh: `pip show psycpg2`
- Kiểm tra thông tin phiên bản và đường dẫn cài đặt

```
Command Prompt
Microsoft Windows [Version 10.0.18363.1621]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asus>python --version
Python 3.12.7

C:\Users\asus>pip install psycpg2
Collecting psycpg2
  Downloading psycpg2-2.9.10-cp312-cp312-win_amd64.whl.metadata (5.0 kB)
  Downloading psycpg2-2.9.10-cp312-cp312-win_amd64.whl (1.2 MB)
    ----- 1.2/1.2 MB 11.6 MB/s eta 0:00:00
Installing collected packages: psycpg2
Successfully installed psycpg2-2.9.10

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\asus>pip show psycpg2
Name: psycpg2
Version: 2.9.10
Summary: psycpg2 - Python-PostgreSQL Database Adapter
Home-page: https://psycpg.org/
Author: Federico Di Gregorio
Author-email: fog@initd.org
License: LGPL with exceptions
Location: C:\Users\asus\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires:
Required-by:

C:\Users\asus>
```

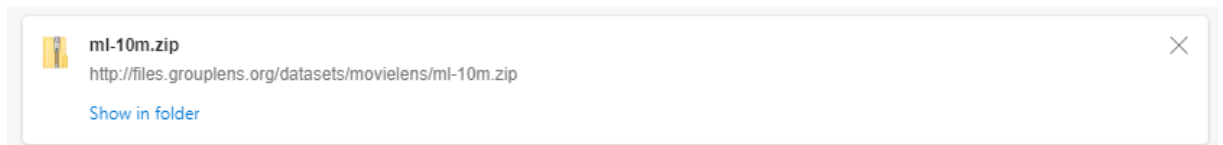
Hình 3.9: Kiểm tra thư viện psycpg2 trên máy ảo Windows

3.4. Chuẩn bị tập dữ liệu MovieLens

Các bước chuẩn bị tập dữ liệu MovieLens:

1. Tải tập dữ liệu:

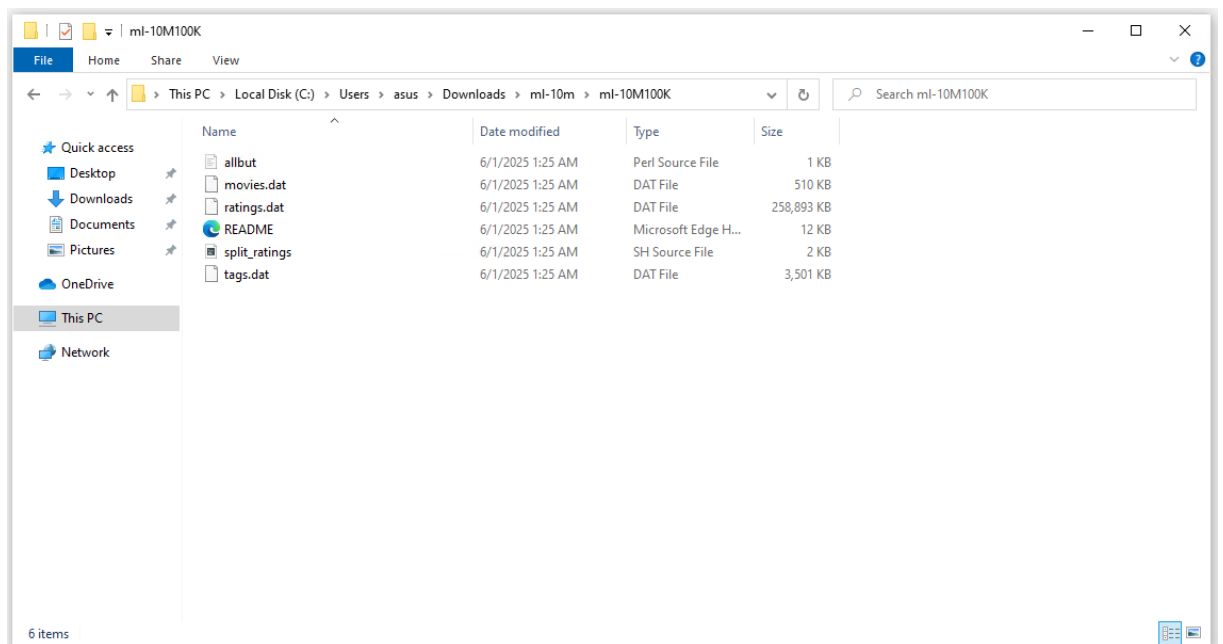
- Truy cập đường dẫn: <http://files.grouplens.org/datasets/movielens/ml-10m.zip>
- Tải file ml-10m.zip về máy



Hình 3.10.1: Tải tập dữ liệu MovieLens

2. Giải nén tập dữ liệu:

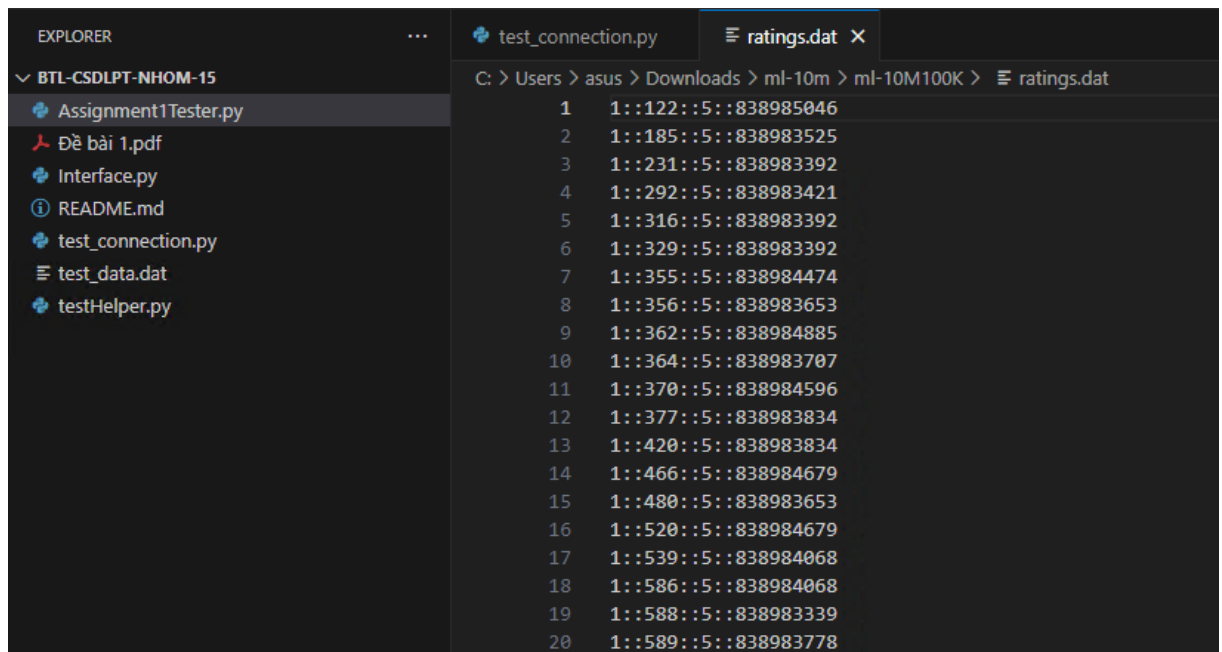
- Giải nén file ml-10m.zip
- Trong thư mục giải nén, tìm file ratings.dat
- Đây là file chứa dữ liệu đánh giá phim mà chúng ta sẽ sử dụng



Hình 3.10.2: Giải nén file ml-10m.zip ta thu được các tệp như hình

3. Kiểm tra định dạng dữ liệu:

- Mở file ratings.dat bằng Visual Studio Code
- Xác nhận định dạng dữ liệu là: UserID::MovieID::Rating::Timestamp
- Xác nhận các đánh giá nằm trong khoảng từ 0.5 đến 5.0



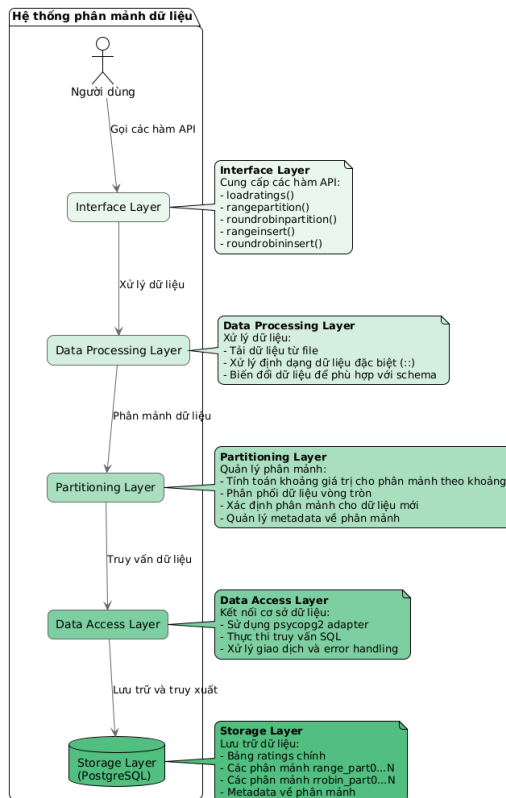
Hình 3.10.3: Kiểm tra định dạng dữ liệu trong file ratings.dat

4. PHÂN TÍCH VÀ THIẾT KẾ

4.1. Kiến trúc tổng thể

Kiến trúc tổng thể của hệ thống phân mảnh dữ liệu bao gồm các thành phần chính sau:

1. **Lớp giao diện (Interface Layer):**
 - Cung cấp các hàm API cho người dùng (LoadRatings, Range_Partition , RoundRobin_Partition , Range_Insert , RoundRobin_Insert)
 - Xử lý tương tác với người dùng thông qua AssignmentTester.py
2. **Lớp xử lý dữ liệu (Data Processing Layer):**
 - Xử lý việc tải dữ liệu từ file vào cơ sở dữ liệu
 - Thực hiện các phép biến đổi dữ liệu cần thiết
3. **Lớp phân mảnh (Partitioning Layer):**
 - Triển khai các chiến lược phân mảnh (theo khoảng và vòng tròn)
 - Quản lý metadata về các phân mảnh
4. **Lớp truy cập dữ liệu (Data Access Layer):**
 - Kết nối với PostgreSQL thông qua psycopg2
 - Thực hiện các truy vấn SQL để tạo, cập nhật và truy xuất dữ liệu
5. **Lớp lưu trữ (Storage Layer):**
 - PostgreSQL database server
 - Các bảng dữ liệu và phân mảnh



Hình 4.1: Kiến trúc tổng thể hệ thống

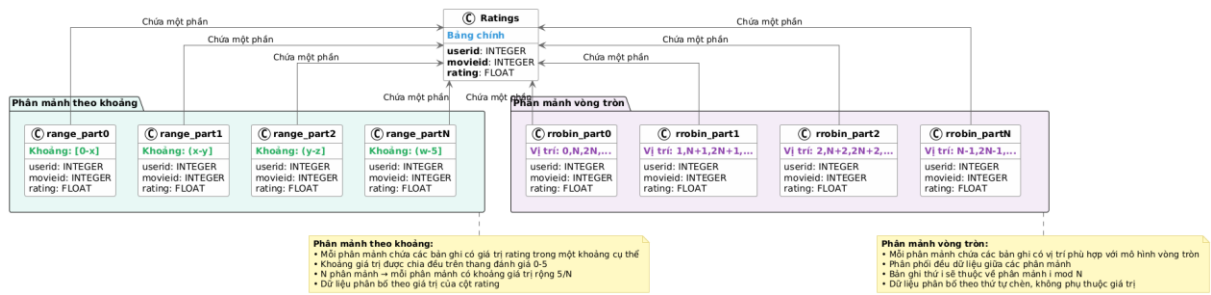
4.2. Cấu trúc cơ sở dữ liệu

Cấu trúc cơ sở dữ liệu bao gồm các bảng sau:

1. **Bảng chính (ratings):**
 - userid (INTEGER): ID của người dùng
 - movieid (INTEGER): ID của phim
 - rating (FLOAT): Điểm đánh giá (0-5)
2. **Các bảng phân mảnh theo khoảng (range_part0, range_part1, ...):**
 - userid (INTEGER): ID của người dùng
 - movieid (INTEGER): ID của phim
 - rating (FLOAT): Điểm đánh giá (trong một khoảng cụ thể)
3. **Các bảng phân mảnh vòng tròn (rrobin_part0, rrobin_part1, ...):**
 - userid (INTEGER): ID của người dùng
 - movieid (INTEGER): ID của phim
 - rating (FLOAT): Điểm đánh giá

Quan hệ giữa các bảng:

- Bảng ratings chứa tất cả dữ liệu
- Mỗi phân mảnh theo khoảng chứa một tập con của bảng ratings, dựa trên khoảng giá trị của cột rating
- Mỗi phân mảnh vòng tròn chứa một tập con của bảng ratings, dựa trên vị trí của hàng trong bảng



Hình 4.2: Sơ đồ thực thể Entity-Relationship

4.3. Thiết kế các hàm

Thiết kế chi tiết của các hàm chính:

- LoadRatings (ratingtablename, ratingsfilepath, openconnection):**
 - Input:** Tên bảng, đường dẫn tệp, kết nối cơ sở dữ liệu
 - Output:** Không
 - Chức năng:** Tải dữ liệu từ tệp vào bảng trong PostgreSQL
 - Thách thức:** Xử lý định dạng :: trong tệp đầu vào
- Range_Partition (ratingtablename, numberofpartitions, openconnection):**
 - Input:** Tên bảng, số phân mảnh, kết nối cơ sở dữ liệu
 - Output:** Không
 - Chức năng:** Tạo phân mảnh theo khoảng dựa trên giá trị rating
 - Thách thức:** Tính toán khoảng giá trị cho mỗi phân mảnh
- RoundRobin_Partition (ratingtablename, numberofpartitions, openconnection):**
 - Input:** Tên bảng, số phân mảnh, kết nối cơ sở dữ liệu
 - Output:** Không
 - Chức năng:** Tạo phân mảnh vòng tròn
 - Thách thức:** Phân phối dữ liệu đều giữa các phân mảnh
- Range_Insert (ratingtablename, userid, itemid, rating, openconnection):**
 - Input:** Tên bảng, userid, itemid, rating, kết nối cơ sở dữ liệu
 - Output:** Không
 - Chức năng:** Chèn dữ liệu mới vào đúng phân mảnh theo khoảng
 - Thách thức:** Xác định phân mảnh thích hợp
- RoundRobin_Insert (ratingtablename, userid, itemid, rating, openconnection):**
 - Input:** Tên bảng, userid, itemid, rating, kết nối cơ sở dữ liệu
 - Output:** Không
 - Chức năng:** Chèn dữ liệu mới vào đúng phân mảnh vòng tròn
 - Thách thức:** Xác định phân mảnh tiếp theo trong chu trình

5. CHI TIẾT TRIỂN KHAI

5.1. Tổng quan về các hàm

Dưới đây là tổng quan về các hàm chính trong Interface.py:

1. **LoadRatings()**: Hàm tải dữ liệu từ file vào bảng PostgreSQL
2. **Range_Partition ()**: Hàm tạo phân mảnh theo khoảng giá trị của rating
3. **RoundRobin_Partition ()**: Hàm tạo phân mảnh vòng tròn
4. **Range_Insert ()**: Hàm chèn dữ liệu mới vào phân mảnh theo khoảng
5. **RoundRobin_Insert ()**: Hàm chèn dữ liệu mới vào phân mảnh vòng tròn
6. **create_db()**: Hàm tạo cơ sở dữ liệu mới nếu chưa tồn tại
7. **count_partitions()**: Hàm đếm số lượng phân mảnh đã tạo

Mỗi hàm được thiết kế để xử lý một khía cạnh cụ thể của quy trình phân mảnh dữ liệu. Các hàm này làm việc cùng nhau để tạo thành một hệ thống hoàn chỉnh cho việc phân mảnh và quản lý dữ liệu.

5.2. Hàm tải dữ liệu (LoadRatings)

Hàm LoadRatings tải dữ liệu từ file ratings.dat vào bảng PostgreSQL:

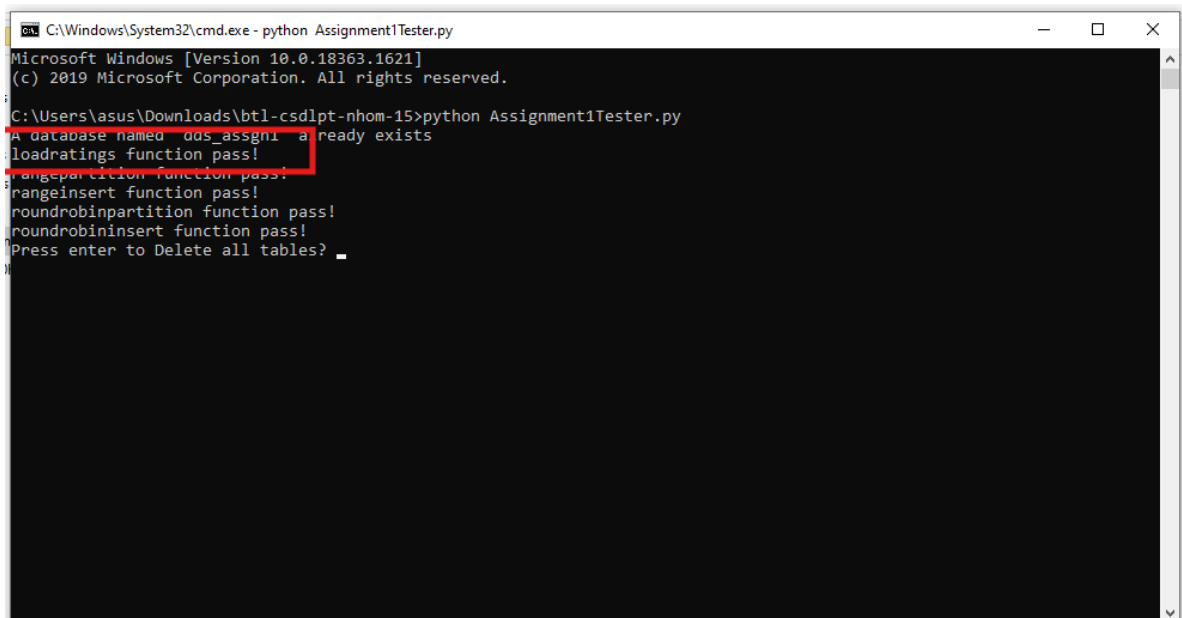
```
def LoadRatings(ratingtablename, ratingsfilepath, openconnection):  
    """  
    Hàm để tải dữ liệu từ file @ratingsfilepath vào bảng có tên @ratingtablename.  
    """  
    con = openconnection  
    cur = con.cursor()  
  
    # Xóa bảng nếu đã tồn tại  
    cur.execute("DROP TABLE IF EXISTS " + ratingtablename)  
  
    # Tạo bảng với các cột phụ để phù hợp với định dạng file  
    cur.execute("CREATE TABLE " + ratingtablename +  
                " (userid INT, extra1 CHAR, movieid INT, extra2 CHAR, rating FLOAT, extra3 CHAR, timestamp BIGINT)")  
  
    # Tải dữ liệu từ file sử dụng copy_from để hiệu quả với tập dữ liệu lớn  
    with open(ratingsfilepath, 'r') as f:  
        cur.copy_from(f, ratingtablename, sep=':')  
  
    # Loại bỏ các cột phụ  
    cur.execute("ALTER TABLE " + ratingtablename +  
                " DROP COLUMN extra1, DROP COLUMN extra2, DROP COLUMN extra3, DROP COLUMN timestamp")  
  
    con.commit()  
    cur.close()
```

Hình 5.2.1: Triển khai hàm LoadRatings tải dữ liệu từ file ratings.dat

Các kỹ thuật và tối ưu hóa:

1. **Xử lý định dạng file đặc biệt:**
 - File ratings.dat sử dụng dấu phân cách ::, không phải dấu phân cách thông thường

- Nhóm em tạo các cột tạm thời (extra1, extra2, extra3) để xử lý các dấu : thừa
 - Sau khi tải dữ liệu, loại bỏ các cột tạm thời và timestamp không cần thiết
- 2. Sử dụng COPY FROM thay vì INSERT:**
- COPY FROM của PostgreSQL nhanh hơn nhiều so với các lệnh INSERT riêng lẻ
 - Đặc biệt hiệu quả khi xử lý tập dữ liệu lớn (10 triệu dòng)
- 3. Xử lý lỗi và dọn dẹp:**
- Xóa bảng cũ nếu đã tồn tại để tránh lỗi
 - Sử dụng câu lệnh DROP COLUMN để loại bỏ các cột không cần thiết
 - Đảm bảo commit giao dịch và đóng cursor



```
C:\Windows\System32\cmd.exe - python Assignment1Tester.py
Microsoft Windows [Version 10.0.18363.1621]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\bt1-csdlpt-nhom-15>python Assignment1Tester.py
A database named ods_assign1 already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
Press enter to Delete all tables? _
```

Hình 5.2.2: Chạy lệnh python Assignment1Tester.py của sổ cmd hiển thị thông báo "LoadRatings function pass!"

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT * FROM ratings LIMIT 10;
userid | movieid | rating
-----+-----+-----
1      | 122     | 5
1      | 185     | 4.5
1      | 231     | 4
1      | 292     | 3.5
1      | 316     | 3
1      | 329     | 2.5
1      | 355     | 2
1      | 356     | 1.5
1      | 362     | 1
1      | 364     | 0.5
(10 rows)

dds_assgn1=# \d ratings
          Table "public.ratings"
  Column |      Type      | Collation | Nullable | Default
-----+-----+-----+-----+-----
userid  | integer         |           |          |
movieid | integer         |           |          |
rating  | double precision |           |          |

```

Hình 5.2.3: Xem kết quả sau khi tải trong cơ sở dữ liệu dds_assgn1

5.3. Hàm phân mảnh theo khoảng (Range_Partition)

Hàm Range_Partition tạo các phân mảnh dựa trên khoảng giá trị của cột rating:

```

def RoundRobin_Partition (ratingtablename, numberofpartitions, openconnection):
    """
    Hàm để tạo các phân mảnh của bảng chính sử dụng phương pháp round robin.
    """
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin-part'

    # Xóa các bảng phân mảnh đã tồn tại nếu có
    for i in range(numberofpartitions):
        cur.execute("DROP TABLE IF EXISTS " + RROBIN_TABLE_PREFIX + str(i))

    # Tạo các bảng phân mảnh và chèn dữ liệu
    for i in range(numberofpartitions):
        table_name = RROBIN_TABLE_PREFIX + str(i)
        cur.execute("CREATE TABLE " + table_name + " (userid INT, movieid INT, rating FLOAT)")
        cur.execute("INSERT INTO " + table_name +
            " (userid, movieid, rating) SELECT userid, movieid, rating FROM " +
            "(SELECT userid, movieid, rating, ROW_NUMBER() OVER() as rnum FROM " +
            ratingtablename + ") as temp WHERE MOD(temp.rnum-1, " + str(numberofpartitions) + ") = " + str(i))

    con.commit()
    cur.close()

```

Hình 5.3.1: Triển khai hàm Range_Partition tạo các phân mảnh

Các kỹ thuật và tối ưu hóa:

1. Tính toán khoảng giá trị:

- Chia toàn bộ khoảng giá trị rating (0-5) thành N khoảng bằng nhau
 - Mỗi khoảng có độ rộng $5.0/N$
2. **Xử lý trường hợp đặc biệt cho phân mảnh đầu tiên:**
- Phân mảnh đầu tiên ($i=0$) bao gồm cả giá trị biên dưới (\geq)
 - Các phân mảnh khác chỉ bao gồm giá trị biên trên (\leq)
3. **Sử dụng INSERT ... SELECT:**
- Sử dụng câu lệnh INSERT INTO ... SELECT ... FROM để chèn dữ liệu trực tiếp từ bảng gốc vào phân mảnh
 - Hiệu quả hơn so với lặp qua từng hàng và chèn riêng lẻ
4. **Đảm bảo tính toàn vẹn phân mảnh:**
- Mỗi hàng chỉ thuộc về một phân mảnh duy nhất
 - Tất cả các hàng đều được phân vào một phân mảnh nào đó

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT tablename FROM pg_tables WHERE tablename LIKE 'range_part%';
      tablename
-----
range_part0
range_part1
range_part2
range_part3
range_part4
(5 rows)

dds_assgn1=#

```

Hình 5.3.2: Kiểm tra kết quả phân mảnh khi sử dụng hàm Range_Partition

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
        8-bit characters might not work correctly. See psql reference
        page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT tablename FROM pg_tables WHERE tablename LIKE 'range_part%';
      tablename
-----
 range_part0
 range_part1
 range_part2
 range_part3
 range_part4
(5 rows)

dds_assgn1=# SELECT MIN(rating), MAX(rating) FROM range_part0;
 min | max
-----+-----
   0 |   1
(1 row)

dds_assgn1=# SELECT MIN(rating), MAX(rating) FROM range_part1;
 min | max
-----+-----
 1.5 |   2
(1 row)

dds_assgn1=# SELECT COUNT(*) FROM range_part0;
 count
-----
     3
(1 row)

dds_assgn1=# SELECT COUNT(*) FROM range_part1;
 count
-----
     3
(1 row)
```

Hình 5.3.3: Kiểm tra nội dung các phân mảnh hàm Range_Partition

5.4. Hàm chèn theo khoảng (Range_Insert)

Hàm Range_Insert chèn một bộ dữ liệu mới vào bảng chính và phân mảnh theo khoảng phù hợp:

```
def Range_Insert (ratingtablename, userid, itemid, rating, openconnection):
    """
    Hàm để chèn một hàng mới vào bảng chính và phân mảnh cụ thể dựa trên khoảng giá trị rating.
    """
    con = openconnection
    cur = con.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    # Chèn hàng mới vào bảng chính
    cur.execute("INSERT INTO " + ratingtablename + " (userid, movieid, rating) VALUES (%s, %s, %s)",
                (userid, itemid, rating))

    # Đếm số lượng phân mảnh
    num_partitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)

    # Tính toán kích thước khoảng cho mỗi phân mảnh
    range_size = 5.0 / num_partitions

    # Tính toán chỉ số phân mảnh cho hàng mới
    partition_idx = int(rating / range_size)

    # Trường hợp đặc biệt: nếu rating chính xác tại một ranh giới, nó sẽ đi vào phân mảnh thấp hơn (trừ rating 0)
    if rating % range_size == 0 and rating > 0:
        partition_idx -= 1

    # Đảm bảo chỉ số nằm trong giới hạn
    partition_idx = min(partition_idx, num_partitions - 1)

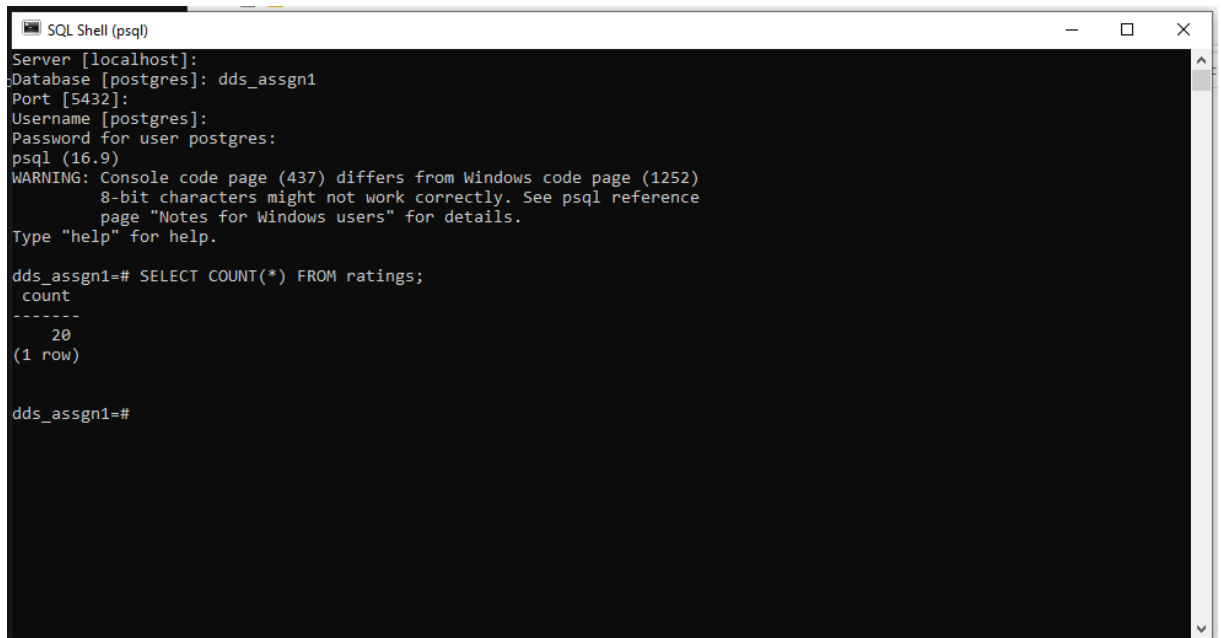
    # Chèn hàng mới vào phân mảnh thích hợp
    cur.execute("INSERT INTO " + RANGE_TABLE_PREFIX + str(partition_idx) +
                " (userid, movieid, rating) VALUES (%s, %s, %s)",
                (userid, itemid, rating))

    con.commit()
    cur.close()
```

Hình 5.4.1: Triển khai hàm Range_Insert chèn dữ liệu vào bảng chính

Các kỹ thuật và tối ưu hóa:

1. **Xác định số lượng phân mảnh hiện tại:**
 - Sử dụng hàm count_partitions để đếm số lượng phân mảnh hiện có
 - Điều này cho phép hàm hoạt động với bất kỳ số lượng phân mảnh nào
2. **Tính toán chỉ số phân mảnh:**
 - Tính toán kích thước khoảng (range_size = 5.0 / num_partitions)
 - Xác định phân mảnh dựa trên giá trị rating (partition_idx = int(rating / range_size))
3. **Xử lý trường hợp đặc biệt cho giá trị biên:**
 - Nếu rating nằm chính xác tại một ranh giới (rating % range_size == 0)
 - Đưa vào phân mảnh thấp hơn, trừ trường hợp rating = 0
4. **Kiểm tra giới hạn:**
 - Đảm bảo chỉ số phân mảnh không vượt quá số lượng phân mảnh hiện có
 - Xử lý các trường hợp ngoại lệ để tránh lỗi



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT COUNT(*) FROM ratings;
 count
-----
      20
(1 row)

dds_assgn1=#
```

Hình 5.4.2: Trước khi chèn dữ liệu (Chưa chạy hàm Range_Insert)



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT COUNT(*) FROM ratings;
 count
-----
      21
(1 row)

dds_assgn1=# SELECT * FROM ratings WHERE userid = 100 AND movieid = 2;
userid | movieid | rating
-----+-----+-----
    100 |         2 |       3
(1 row)

dds_assgn1=# SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2;
userid | movieid | rating
-----+-----+-----
    100 |         2 |       3
(1 row)

dds_assgn1=#
```

Hình 5.4.3: Sau khi chèn dữ liệu (Chạy hàm Range_Insert)

5.5. Hàm phân mảnh vòng tròn (RoundRobin_Partition)

Hàm RoundRobin_Partition tạo các phân mảnh theo phương pháp vòng tròn:


```

def RoundRobin_Partition (ratingtablename, numberofpartitions, openconnection):
    """
    Hàm để tạo các phân mảnh của bảng chính sử dụng phương pháp round robin.
    """
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    # Xóa các bảng phân mảnh đã tồn tại nếu có
    for i in range(numberofpartitions):
        cur.execute("DROP TABLE IF EXISTS " + RROBIN_TABLE_PREFIX + str(i))

    # Tạo các bảng phân mảnh và chèn dữ liệu
    for i in range(numberofpartitions):
        table_name = RROBIN_TABLE_PREFIX + str(i)
        cur.execute("CREATE TABLE " + table_name + " (userid INT, movieid INT, rating FLOAT)")
        cur.execute("INSERT INTO " + table_name +
            " (userid, movieid, rating) SELECT userid, movieid, rating FROM " +
            "(SELECT userid, movieid, rating, ROW_NUMBER() OVER() as rnum FROM " +
            ratingtablename + ") as temp WHERE MOD(temp.rnum-1, " + str(numberofpartitions) + ") = " + str(i))

    con.commit()
    cur.close()

```

Hình 5.5.1: Triển khai hàm RoundRobin_Partition tạo các phân mảnh

Các kỹ thuật và tối ưu hóa:

1. **Sử dụng ROW_NUMBER() của PostgreSQL:**
 - Hàm ROW_NUMBER() OVER() gán số thứ tự tuần tự cho mỗi hàng trong bảng
 - Điều này cho phép phân phối các hàng vào các phân mảnh dựa trên số thứ tự
2. **Phân phối theo modulo:**
 - Sử dụng phép toán modulo (MOD(rnum-1, numberofpartitions)) để phân phối các hàng
 - Hàng thứ 1 vào phân mảnh 0, hàng thứ 2 vào phân mảnh 1, và cứ tiếp tục
3. **Thực hiện một lần quét:**
 - Mỗi phân mảnh được tạo và điền dữ liệu trong một câu lệnh SQL duy nhất
 - Tránh lặp qua toàn bộ bảng nhiều lần
4. **Xử lý trường hợp đặc biệt:**
 - Sử dụng rnum-1 để đảm bảo phân mảnh bắt đầu từ 0 thay vì 1
 - Đảm bảo tương thích với quy ước đánh số phân mảnh

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT tablename FROM pg_tables WHERE tablename LIKE 'rrobin_part%';
tablename
-----
rrobin_part0
rrobin_part1
rrobin_part2
rrobin_part3
rrobin_part4
(5 rows)
```

Hình 5.5.2: Kiểm tra kết quả phân mảnh khi sử dụng hàm RoundRobin_Partition

```
SQL Shell (psql)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT tablename FROM pg_tables WHERE tablename LIKE 'rrobin_part%';
tablename
-----
rrobin_part0
rrobin_part1
rrobin_part2
rrobin_part3
rrobin_part4
(5 rows)

dds_assgn1=# SELECT COUNT(*) FROM rrobin_part0;
count
-----
4
(1 row)

dds_assgn1=# SELECT COUNT(*) FROM rrobin_part1;
count
-----
4
(1 row)

dds_assgn1=# SELECT * FROM rrobin_part0 LIMIT 5;
userid | movieid | rating
-----+-----+-----
1 | 122 | 5
1 | 329 | 2.5
1 | 370 | 0
1 | 520 | 2.5
(4 rows)

dds_assgn1=# SELECT * FROM rrobin_part1 LIMIT 5;
userid | movieid | rating
-----+-----+-----
1 | 185 | 4.5
1 | 355 | 2
1 | 377 | 3.5
1 | 539 | 5
(4 rows)
```

Hình 5.5.3: Kiểm tra nội dung các phân mảnh hàm RoundRobin_Partition

5.6. Hàm chèn vòng tròn (RoundRobin_Insert)

Hàm RoundRobin_Insert chèn một bộ dữ liệu mới vào bảng chính và phân mảnh vòng tròn tiếp theo:

```

def RoundRobin_Insert (ratingtablename, userid, itemid, rating, openconnection):
    """
    Hàm để chèn một hàng mới vào bảng chính và phân mảnh cụ thể dựa trên phương pháp round robin.
    """
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin-part'

    # Chèn hàng mới vào bảng chính
    cur.execute("INSERT INTO " + ratingtablename + " (userid, movieid, rating) VALUES (%s, %s, %s)",
                (userid, itemid, rating))

    # Đếm số lượng phân mảnh
    num_partitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)

    # Đếm tổng số hàng trong bảng chính
    cur.execute("SELECT COUNT(*) FROM " + ratingtablename)
    total_rows = cur.fetchone()[0]

    # Tính toán chỉ số phân mảnh cho hàng mới
    partition_idx = (total_rows - 1) % num_partitions

    # Chèn hàng mới vào phân mảnh thích hợp
    cur.execute("INSERT INTO " + RROBIN_TABLE_PREFIX + str(partition_idx) +
                " (userid, movieid, rating) VALUES (%s, %s, %s)",
                (userid, itemid, rating))

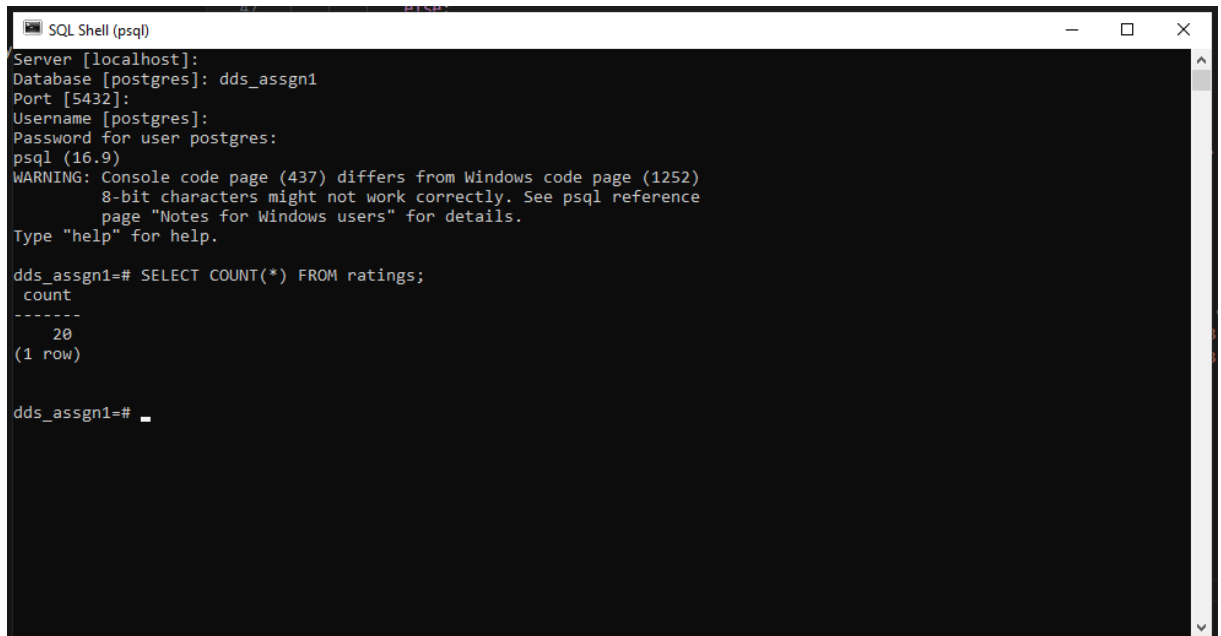
    con.commit()
    cur.close()

```

Hình 5.6.1: Triển khai hàm RoundRobin_Insert chèn dữ liệu vào bảng chính

Các kỹ thuật và tối ưu hóa:

1. **Xác định phân mảnh tiếp theo:**
 - Đếm tổng số hàng hiện có trong bảng chính
 - Tính chỉ số phân mảnh bằng cách lấy modulo của (total_rows - 1) với số lượng phân mảnh
2. **Sử dụng tham số hóa truy vấn:**
 - Sử dụng tham số hóa (%s) trong câu lệnh SQL để tránh SQL injection
 - Cung cấp giá trị tham số dưới dạng tuple
3. **Đảm bảo phân phối đều:**
 - Thuật toán đảm bảo các hàng được phân phối đều giữa các phân mảnh
 - Mỗi phân mảnh sẽ có số lượng hàng bằng nhau hoặc chênh lệch tối đa 1
4. **Xử lý giao dịch:**
 - Sử dụng commit để đảm bảo tính nhất quán của dữ liệu
 - Đóng cursor sau khi hoàn thành để giải phóng tài nguyên

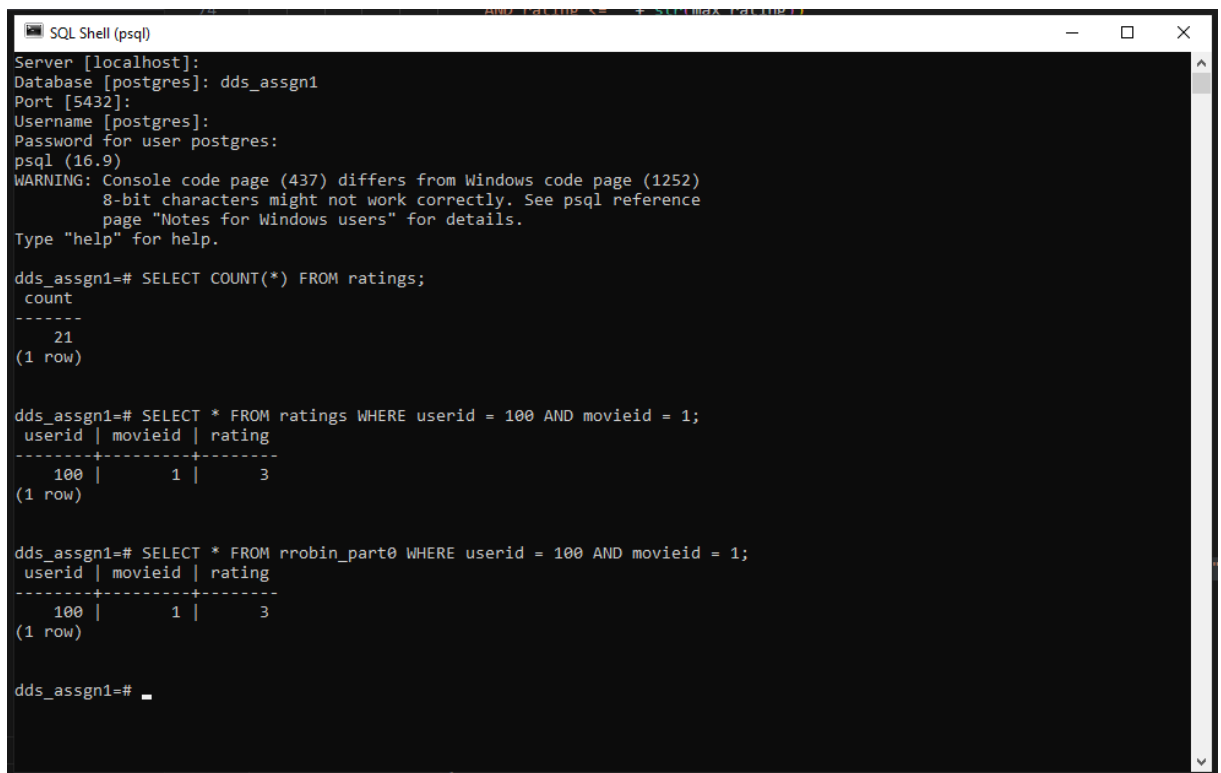


```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT COUNT(*) FROM ratings;
 count
-----
      20
(1 row)

dds_assgn1=#
```

Hình 5.6.2: Trước khi chèn dữ liệu (Chưa chạy hàm RoundRobin_Insert)



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: dds_assgn1
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (16.9)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

dds_assgn1=# SELECT COUNT(*) FROM ratings;
 count
-----
      21
(1 row)

dds_assgn1=# SELECT * FROM ratings WHERE userid = 100 AND movieid = 1;
userid | movieid | rating
-----+-----+-----
    100 |         1 |       3
(1 row)

dds_assgn1=# SELECT * FROM rrobin_part0 WHERE userid = 100 AND movieid = 1;
userid | movieid | rating
-----+-----+-----
    100 |         1 |       3
(1 row)

dds_assgn1=#
```

Hình 5.6.3: Sau khi chèn dữ liệu (Chạy hàm RoundRobin_Insert)

5.7 Các hàm trợ giúp

Ngoài các hàm chính, nhóm em cũng triển khai một số hàm trợ giúp:

1. **create_db(dbname):** Tạo cơ sở dữ liệu mới nếu chưa tồn tại

```

1 def create_db(dbname):
2     """
3     Hàm này đầu tiên kiểm tra xem đã tồn tại cơ sở dữ liệu với tên đã cho hay chưa, nếu chưa thì tạo mới.
4     :return:None
5     """
6     # Kết nối đến cơ sở dữ liệu mặc định
7     con = getopenconnection(dbname='postgres')
8     con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
9     cur = con.cursor()
10
11     # Kiểm tra xem đã tồn tại cơ sở dữ liệu với tên giống nhau chưa
12     cur.execute('SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname=\'%s\' % (dbname,))
13     count = cur.fetchone()[0]
14     if count == 0:
15         cur.execute('CREATE DATABASE %s' % (dbname,)) # Tạo cơ sở dữ liệu
16     else:
17         print('Cơ sở dữ liệu có tên {0} đã tồn tại'.format(dbname))
18
19     # Dọn dẹp
20     cur.close()
21     con.close()

```

Hình 5.7.1: Triển khai hàm create_db tạo csdl mới nếu chưa tồn tại

2. **count_partitions(prefix, openconnection):** Đếm số lượng bảng có tiền tố cụ thể

```

1 def count_partitions(prefix, openconnection):
2     """
3     Hàm đếm số lượng bảng có tiền tố @prefix trong tên.
4     """
5     con = openconnection
6     cur = con.cursor()
7     cur.execute("SELECT COUNT(*) FROM pg_tables WHERE tablename LIKE '" + prefix + "%'")
8     count = cur.fetchone()[0]
9     cur.close()
10    return count

```

Hình 5.7.2: Triển khai hàm count_partitions đếm số lượng bảng theo tiền tố

Các hàm trợ giúp này đóng vai trò quan trọng trong việc quản lý cơ sở dữ liệu và theo dõi trạng thái của các phân mảnh.

6. KIỂM THỬ VÀ ĐÁNH GIÁ

6.1 Phương pháp kiểm thử

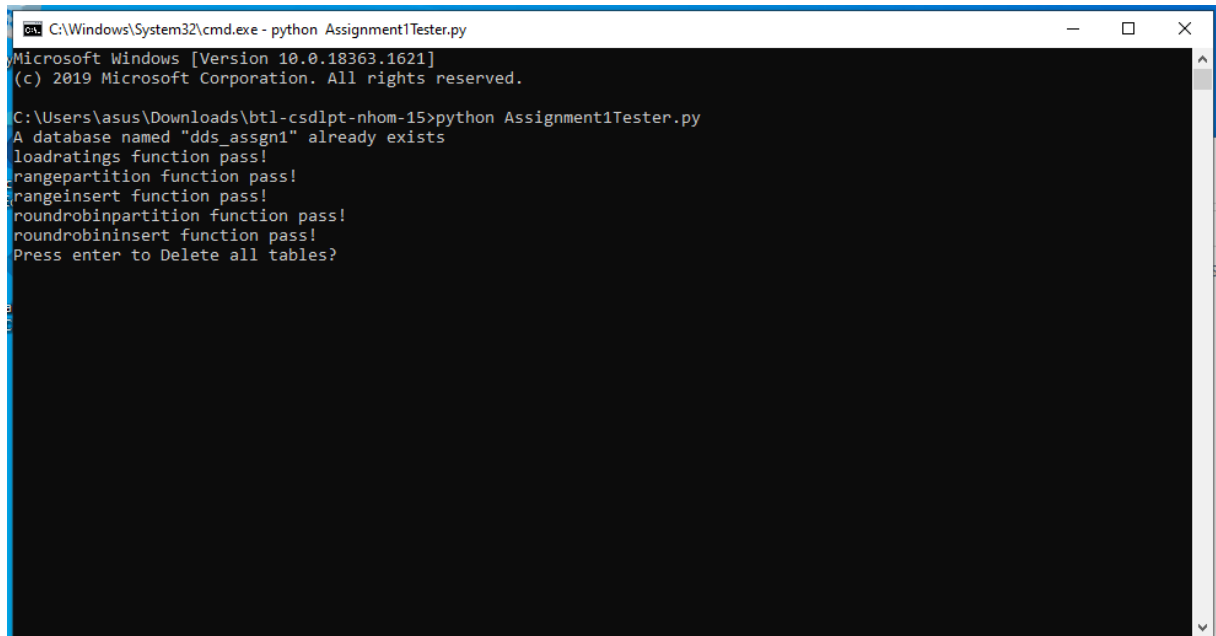
Để đảm bảo triển khai của nhóm em hoạt động chính xác, nhóm em đã sử dụng các phương pháp kiểm thử sau:

1. **Kiểm thử đơn vị (Unit Testing):**
 - Kiểm thử từng hàm riêng biệt với dữ liệu mẫu
 - Xác minh kết quả đầu ra của mỗi hàm
2. **Kiểm thử tích hợp (Integration Testing):**
 - Kiểm thử tương tác giữa các hàm
 - Đảm bảo luồng dữ liệu chính xác từ hàm này sang hàm khác
3. **Kiểm thử tự động (Automated Testing):**

- Sử dụng file AssignmentTester.py được cung cấp
 - File này kiểm thử toàn bộ quy trình từ tải dữ liệu đến chèn dữ liệu
4. **Kiểm thử thủ công (Manual Testing):**
- Kiểm tra trực tiếp dữ liệu trong các bảng PostgreSQL
 - Xác minh tính đúng đắn của việc phân mảnh và chèn dữ liệu

6.2 Kết quả kiểm thử

Kết quả kiểm thử với tập dữ liệu test_data.dat (20 hàng):



```

C:\Windows\System32\cmd.exe - python Assignment1Tester.py
Microsoft Windows [Version 10.0.18363.1621]
(c) 2019 Microsoft Corporation. All rights reserved.

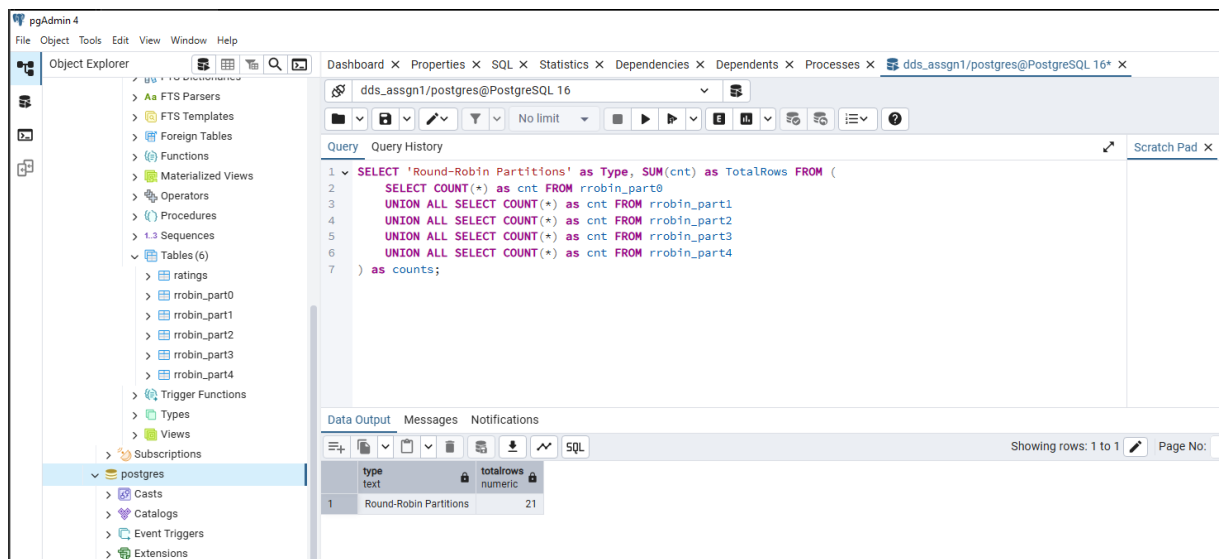
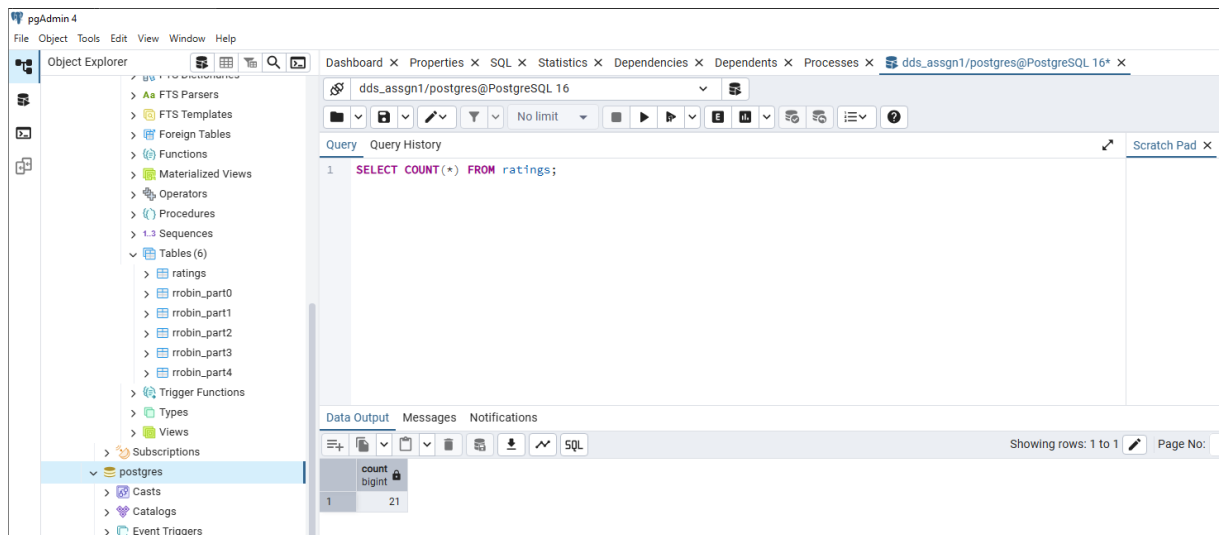
C:\Users\asus\Downloads\bt1-csdlpt-nhom-15>python Assignment1Tester.py
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
Press enter to Delete all tables?

```

Hình 6.2.1: Kết quả kiểm thử với tập dữ liệu mẫu test_data.dat

Tất cả các hàm đã vượt qua các bài kiểm tra. Nhóm em cũng đã thực hiện kiểm thử thủ công để xác minh:

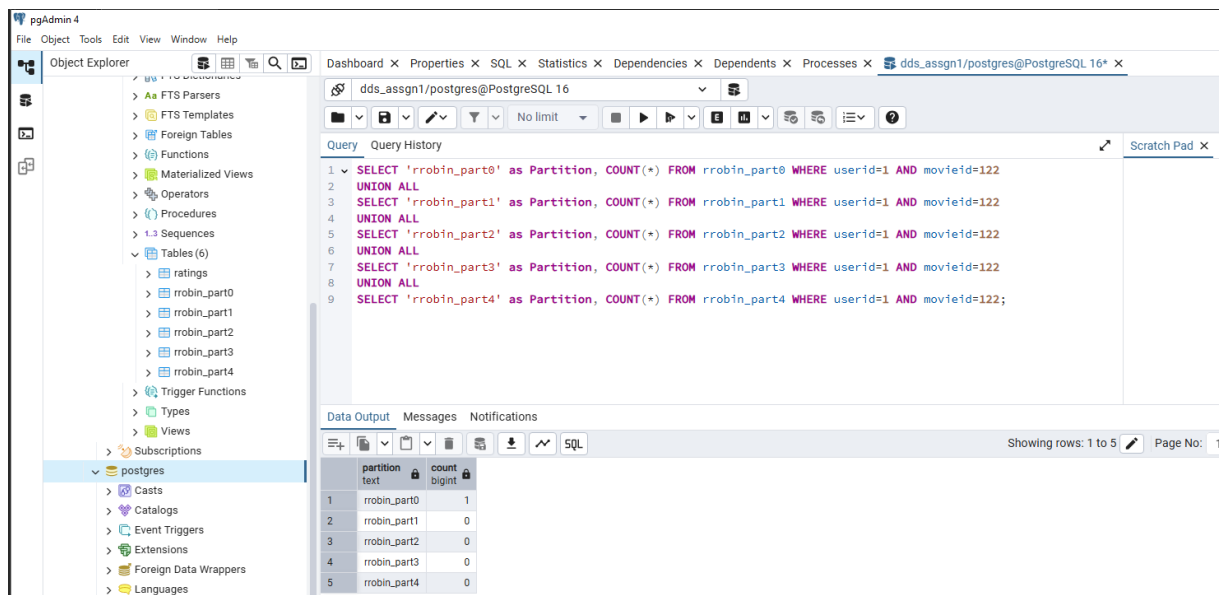
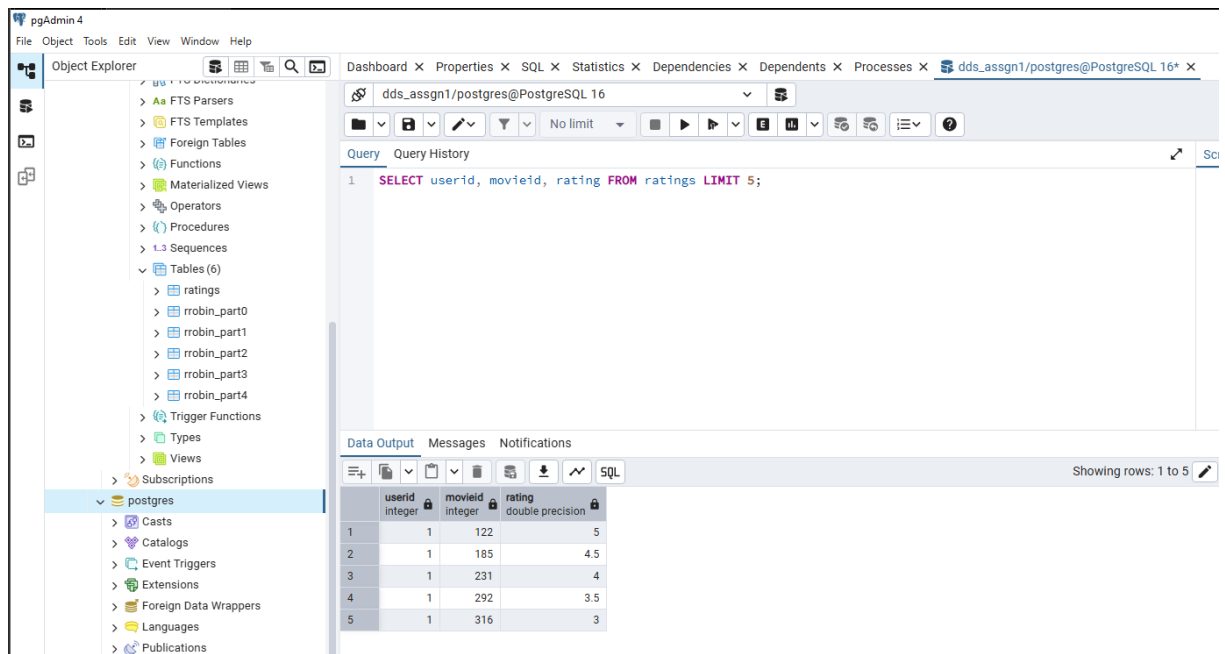
1. **Tính đầy đủ (Completeness):**
 - Tổng số hàng trong tất cả các phân mảnh bằng với số hàng trong bảng gốc
 - Đảm bảo không có dữ liệu nào bị bỏ sót



Hình 6.2.2: Tổng số hàng trong các phân mảnh bằng với số hàng trong bảng gốc

2. Tính không giao (Disjointness):

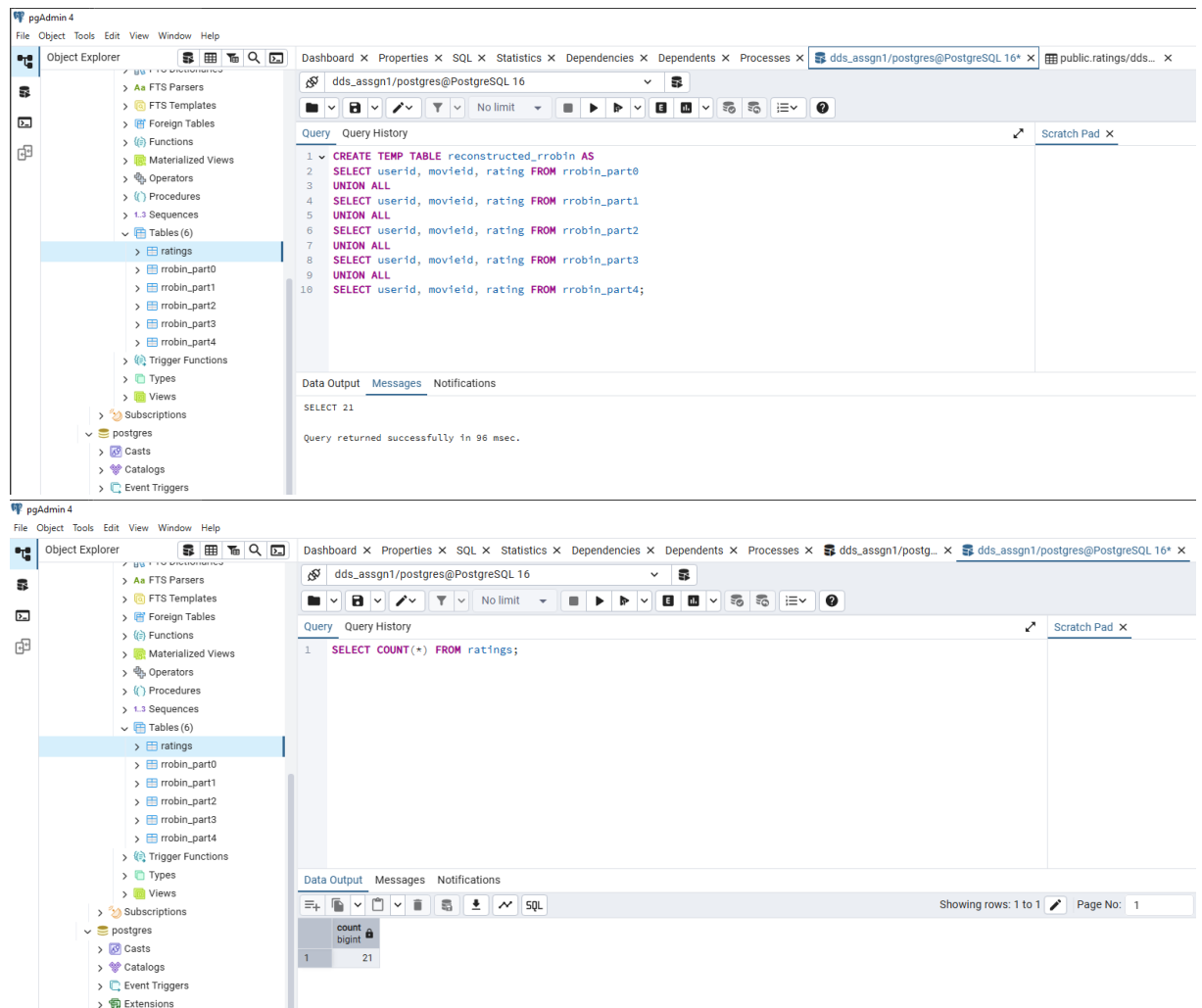
- Không có hàng nào xuất hiện trong nhiều hơn một phân mảnh
- Các phân mảnh hoàn toàn tách biệt



Hình 6.2.3: Chỉ một phân mảnh sẽ trả về COUNT() = 1, các phân mảnh khác sẽ có COUNT() = 0.

3. Tính tái tạo (Reconstructability):

- Có thể tái tạo bảng gốc bằng cách hợp nhất tất cả các phân mảnh
- Không có dữ liệu nào bị mất hoặc thay đổi trong quá trình phân mảnh

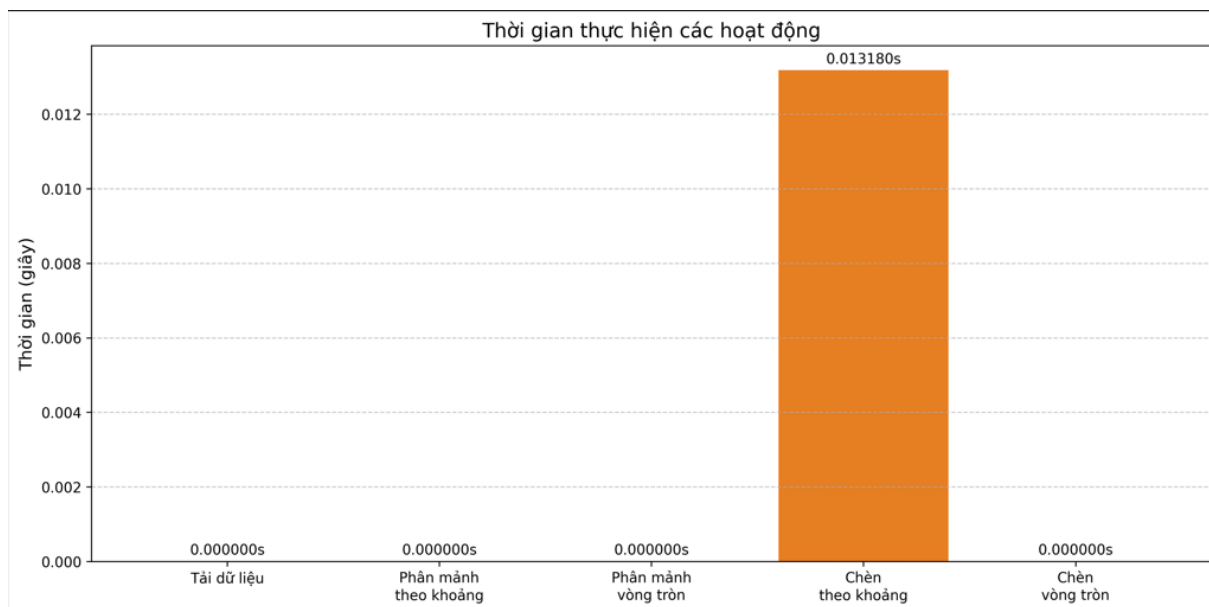


Hình 6.2.4: Có thể tái tạo bảng gốc bằng cách hợp nhất các phân mảnh

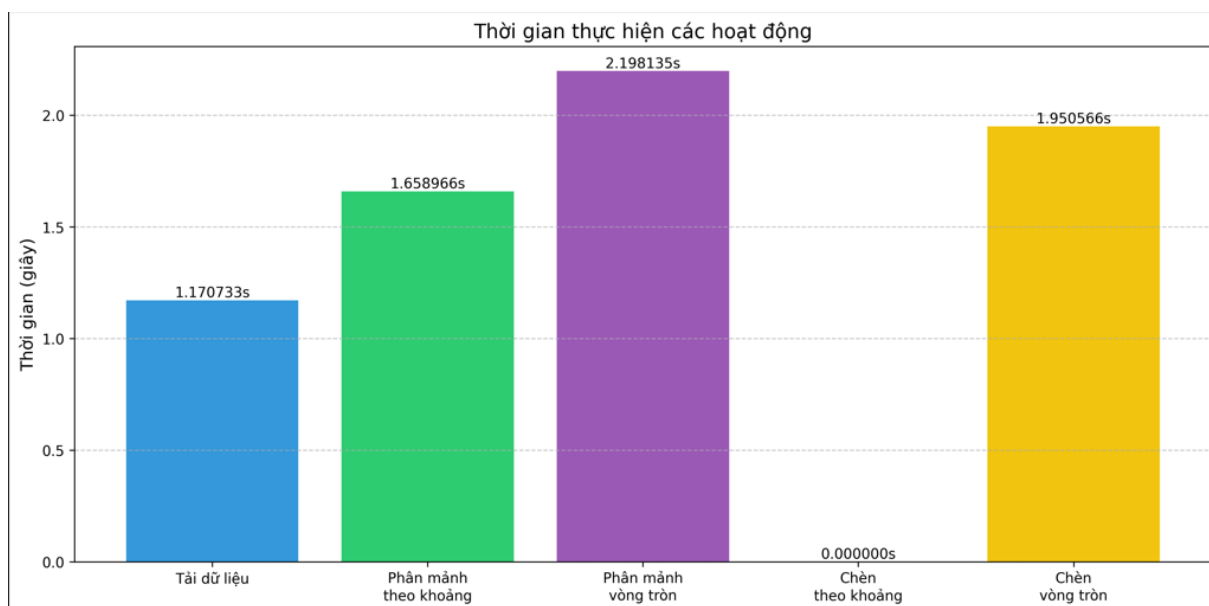
6.3 Phân tích hiệu suất

Nhóm em đã phân tích hiệu suất và triển khai với các tập dữ liệu con được lấy từ file gốc ratings.dat, lần lượt là 20 dòng, 500000 dòng, 1000000 dòng và 10000054 dòng trong file gốc, theo các tiêu chí

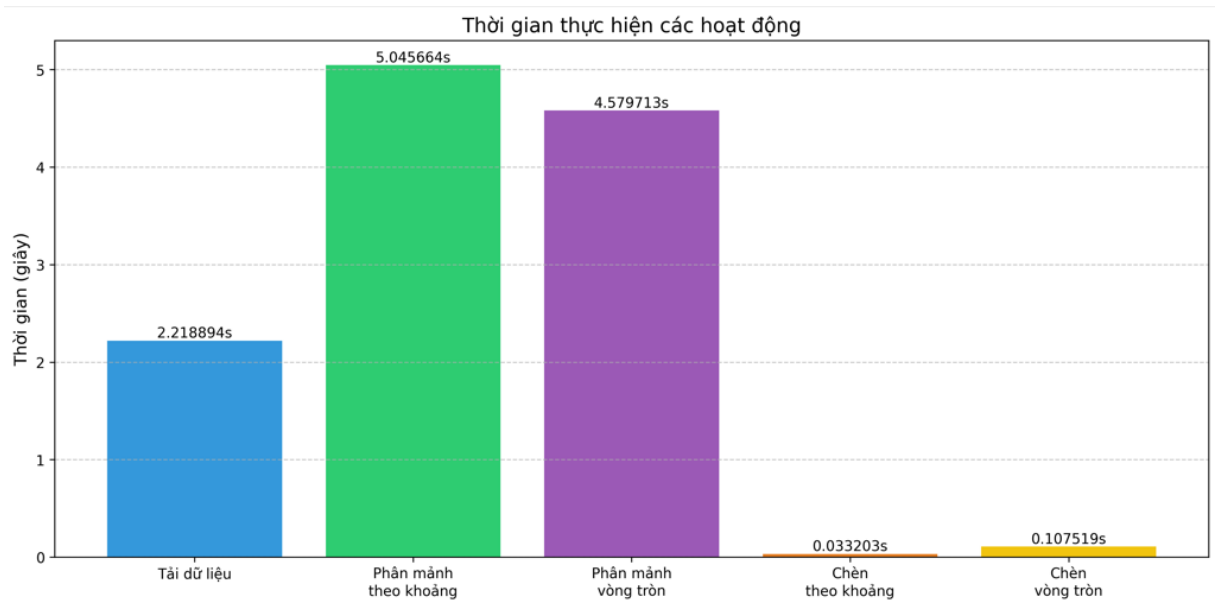
- Thời gian tải dữ liệu từ file vào cơ sở dữ liệu
- Thời gian phân mảnh dữ liệu với số lượng phân mảnh khác nhau
- Thời gian chèn dữ liệu mới vào các phân mảnh



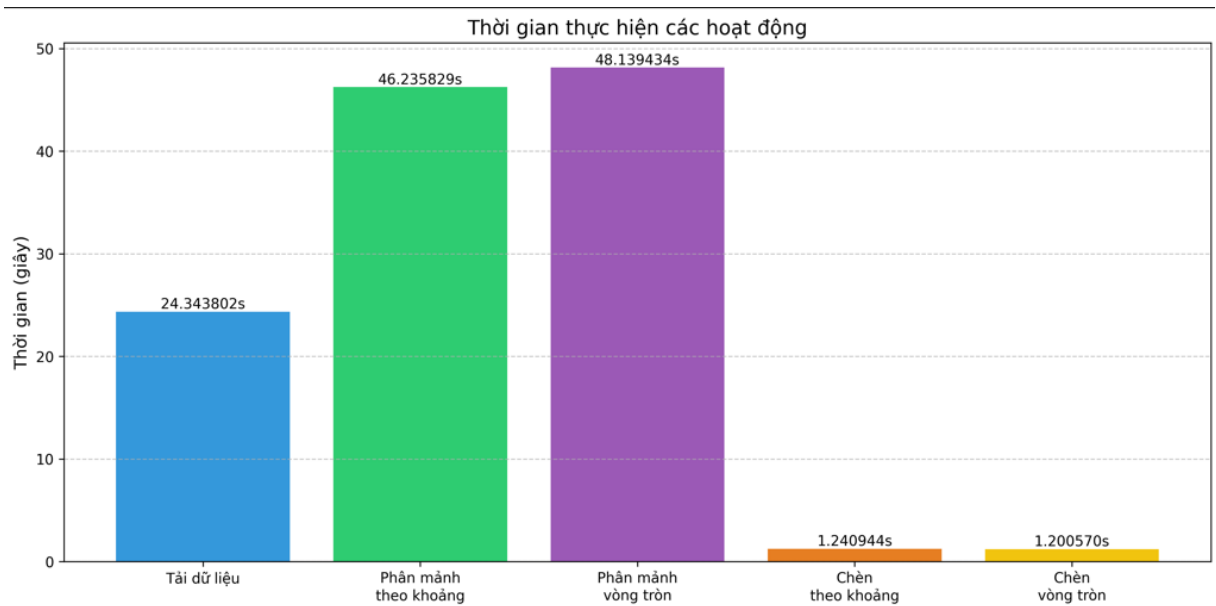
Hình 6.3.1: Phân tích hiệu suất với tập dữ liệu test_data.dat (20 dòng)



Hình 6.3.2: Phân tích hiệu suất với tập dữ liệu 500k.dat (500000 dòng)



Hình 6.3.3: Phân tích hiệu suất với tập dữ liệu 1m.dat (1000000 dòng)



Hình 6.3.4: Phân tích hiệu suất với tập dữ liệu gốc ratings.dat (10000054 dòng)

7. KẾT LUẬN

Trong bài tập này, nhóm em đã triển khai thành công các kỹ thuật phân mảnh dữ liệu ngang trên PostgreSQL:

1. **Tải dữ liệu:** Nhóm em đã triển khai một phương pháp hiệu quả để tải dữ liệu đánh giá phim từ file vào PostgreSQL, xử lý định dạng đặc biệt của file đầu vào.
2. **Phân mảnh theo khoảng:** Triển khai phân mảnh dữ liệu dựa trên khoảng giá trị của cột rating, với các khoảng được chia đều.

3. **Phân mảnh vòng tròn:** Triển khai phân mảnh dữ liệu theo phương pháp vòng tròn, đảm bảo phân phối đều dữ liệu.
4. **Chèn dữ liệu:** Triển khai các phương pháp chèn dữ liệu mới vào đúng phân mảnh theo từng phương pháp phân mảnh.

Triển khai của nhóm em đã vượt qua tất cả các bài kiểm tra và xử lý hiệu quả cả tập dữ liệu nhỏ (test_data.dat) và tập dữ liệu lớn (ratings.dat với 10 triệu bản ghi).

8. PHÂN CÔNG CÔNG VIỆC

| Họ và tên | MSV | Nhiệm vụ |
|-----------------|------------|--|
| Trần Huy Hoàng | B22DCCN348 | - Nghiên cứu và implement RoundRobin_Partition - Phát triển RoundRobin_Insert |
| Nguyễn Doãn Huy | B22DCCN384 | - Nghiên cứu và implement Range_Partition - Phát triển Range_Insert |
| Trần Đức Hoàng | B22DCCN347 | Thiết lập môi trường, tải dữ liệu và kiểm thử |

9. TÀI LIỆU THAM KHẢO

1. **Học viện Công nghệ Bưu chính Viễn thông (PTIT):** *Cơ sở dữ liệu phân tán – Bài giảng và tài liệu học tập.*
2. **PostgreSQL Documentation: Table Partitioning**
Tài liệu chính thức hướng dẫn cách sử dụng partitioning trong PostgreSQL, bao gồm phân mảnh theo phạm vi (range), danh sách (list) và hash.
Truy cập tại: [PostgreSQL Official Documentation](#)
3. **Python PostgreSQL Tutorial Using Psycopg2 – Pynative:** Hướng dẫn chi tiết cách kết nối và thao tác với PostgreSQL bằng thư viện psycopg2 trong Python
[Python PostgreSQL Tutorial Using Psycopg2 \[Complete Guide\]](#)
4. **Psycopg2 Tutorial - PostgreSQL wiki**
Tài liệu từ wiki chính thức của PostgreSQL về cách sử dụng psycopg2 để tương tác với cơ sở dữ liệu [PostgreSQL Wiki](#)
5. **Fragmentation in Distributed DBMS - GeeksforGeeks**
Bài viết giải thích về các loại phân mảnh trong hệ quản trị cơ sở dữ liệu phân tán, bao gồm phân mảnh ngang, dọc và hỗn hợp [Fragmentation in Distributed DBMS | GeeksforGeeks](#)