

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

o0o



BTL MÔN CƠ SỞ DỮ LIỆU PHÂN TÁN

**Đề tài: Phân mảnh theo khoảng và phân mảnh vòng
tròn**

Link GitHub:

https://github.com/quethaibinh/btl_cddlpt_nhom_23

Số thứ tự nhóm: 23

Vũ Đức Thành	MSSV: B22DCCN801
Ngô Đức Sơn	MSSV: B22DCCN693
Hoàng Sơn Hải	MSSV: B22DCCN261

Giảng viên hướng dẫn: Ts. Kim Ngọc Bách

HÀ NỘI, 06/2025

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến thầy Kim Ngọc Bách – giảng viên bộ môn cơ sở dữ liệu phân tán, đã tận tình giảng dạy, hướng dẫn và cung cấp những kiến thức quý báu giúp chúng em hoàn thành bài tập lớn này.

Chúng em cũng xin gửi lời cảm ơn đến các thầy cô trong khoa công nghệ thông tin 1, Học viện Công nghệ Bưu chính Viễn thông, đã tạo điều kiện thuận lợi cho chúng em trong suốt quá trình học tập và nghiên cứu.

Mặc dù đã cố gắng hết sức, nhưng bài báo cáo không thể tránh khỏi những thiếu sót. Chúng em mong nhận được sự góp ý từ thầy cô và các bạn để bài báo cáo được hoàn thiện hơn.

Trân trọng cảm ơn!

PHÂN CÔNG NHIỆM VỤ THỰC HIỆN

TT	SV thực hiện	Công việc / Nhiệm vụ	Hoàn thành
1	Vũ Đức Thành	RangePartition	Đúng hạn
2	Ngô Đức Sơn	RoundRobinPartition	Đúng hạn
3	Hoàng Sơn Hải	LoadRatings	Đúng hạn

Nhóm thực hiện tự đánh giá

TT	SV thực hiện	Thái độ tham gia	Mức hoàn thành CV	Kỹ năng giao tiếp	Kỹ năng hợp tác	Kỹ năng lãnh đạo
1	Vũ Đức Thành	5	5	4	5	
2	Ngô Đức Sơn	5	5	4	5	
3	Hoàng Sơn Hải	5	4	5	5	

Ghi chú:

- **Thái độ tham gia:** Đánh giá điểm thái độ tham gia công việc chung của nhóm (từ 0: không tham gia, đến 5: chủ động, tích cực).
- **Mức hoàn thành CV:** Đánh giá điểm mức độ hoàn thành công việc được giao (từ 0: không hoàn thành, đến 5: hoàn thành xuất sắc).
- **Kỹ năng giao tiếp:** Đánh giá điểm khả năng tương tác, giao tiếp trong nhóm (từ 0: không hoặc giao tiếp rất yếu, đến 5: giao tiếp xuất sắc).
- **Kỹ năng hợp tác:** Đánh giá điểm khả năng hợp tác, hỗ trợ lẫn nhau, giải quyết mâu thuẫn, xung đột.
- **Kỹ năng lãnh đạo:** Đánh giá điểm khả năng lãnh đạo (từ 0: không có khả năng lãnh đạo, đến 5: có khả năng lãnh đạo tốt, tổ chức và điều phối công việc trong nhóm hiệu quả).

MỤC LỤC

PHẦN 1. MỞ ĐẦU.	1
PHẦN 2. CƠ SỞ LÝ THUYẾT.	3
2.1 Mục tiêu và lợi ích của phân mảnh dữ liệu	3
2.2 Phân loại các phương pháp phân mảnh dữ liệu	3
2.2.1 Phân mảnh ngang (Horizontal Fragmentation)	4
2.2.2 Phân mảnh dọc (Vertical Fragmentation).....	4
2.2.3 Phân mảnh hỗn hợp (Hybrid Fragmentation)	4
2.3 Hai kỹ thuật phân mảnh ngang được mô phỏng trong báo cáo.....	4
2.3.1 Phân mảnh theo khoảng (Range Partitioning)	4
2.3.2 Phân mảnh vòng tròn (Round Robin Partitioning)	5
2.4 Giới thiệu về PostgreSQL – hệ quản trị cơ sở dữ liệu mã nguồn mở.....	5
PHẦN 3. CÀI ĐẶT VÀ TRIỂN KHAI.....	7
3.1 Môi trường thực hiện	7
3.1.1 Cài đặt Python 3.12.x và thiết lập môi trường	7
3.1.2 Cài đặt PostgreSQL và pgAdmin 4	7
3.2 Chuẩn bị dữ liệu	9
3.2.1 Tải và xử lý dữ liệu.....	9
3.2.2 Định dạng dữ liệu.....	9
3.2.3 Phân tích bộ dữ liệu.....	10
3.3 Xây dựng các hàm xử lý.	11
3.3.1 Hàm LoadRatings	11
3.3.2 Hàm Range_Partition	14
3.3.3 Hàm RoundRobin_Partition	18
3.3.4 Hàm RoundRobin_Insert	21

3.3.5 Hàm Range_Insert	24
3.3.6 Các Hàm Hỗ Trợ.....	27
PHẦN 4. KIỂM THỬ VÀ ĐÁNH GIÁ.....	29
4.1 LoadRatings.....	29
4.2 RangePartition.....	30
4.3 RangeInsert.....	31
4.4 RoundRobinPartition.....	34
4.5 RoundRobinInsert.....	36
PHẦN 5. KẾT LUẬN.....	39

PHẦN 1. MỞ ĐẦU.

Giới thiệu.

Trong bối cảnh dữ liệu ngày càng bùng nổ với sự phát triển nhanh chóng của các hệ thống thông tin phân tán và ứng dụng web quy mô lớn, việc tổ chức và quản lý dữ liệu hiệu quả trở thành một yếu tố then chốt nhằm đảm bảo hiệu suất truy xuất, tính sẵn sàng cũng như khả năng mở rộng của hệ thống. Đặc biệt, trong các hệ thống phân tán, việc lưu trữ và xử lý dữ liệu cần được tối ưu hóa nhằm giảm thiểu độ trễ, cân bằng tải và đảm bảo tính toàn vẹn dữ liệu.

Một trong những kỹ thuật được áp dụng phổ biến để đạt được các mục tiêu trên là **phân mảnh dữ liệu** (data fragmentation). Phân mảnh dữ liệu là quá trình chia nhỏ một bảng dữ liệu lớn thành nhiều phần nhỏ hơn gọi là các phân mảnh (fragments), mỗi phân mảnh có thể được lưu trữ và xử lý độc lập tại các nút khác nhau trong hệ thống. Kỹ thuật này giúp cải thiện hiệu năng hệ thống, tăng cường khả năng song song hóa truy vấn và dễ dàng quản lý dữ liệu phân tán. Có hai hướng tiếp cận chính trong phân mảnh dữ liệu là: phân mảnh ngang (horizontal fragmentation) và phân mảnh dọc (vertical fragmentation). Trong đó, phân mảnh ngang là phương pháp phổ biến, đặc biệt phù hợp với các ứng dụng có quy mô lớn và yêu cầu truy cập theo dòng dữ liệu (record-based access).

Trong khuôn khổ bài tập lớn này, nhóm chúng em tập trung nghiên cứu và mô phỏng hai kỹ thuật phân mảnh ngang tiêu biểu là: **phân mảnh ngang theo khoảng giá trị** (range partitioning) và **phân mảnh ngang theo vòng tròn** (round robin partitioning). Cả hai kỹ thuật đều được triển khai trên hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở **PostgreSQL**, một nền tảng mạnh mẽ và phổ biến trong nghiên cứu cũng như phát triển hệ thống thực tế.

Nội dung báo cáo được trình bày trong ba chương chính như sau:

- **Phần 2: Cơ sở lý thuyết** – Trình bày các khái niệm nền tảng về phân mảnh dữ liệu, phân loại các kỹ thuật phân mảnh, mục tiêu, lợi ích, cùng với phần giới thiệu về hệ quản trị cơ sở dữ liệu PostgreSQL và vai trò của nó trong hệ thống phân tán.
- **Phần 3: Cài đặt và triển khai** – Hướng dẫn chi tiết quá trình thiết lập môi trường, cài đặt PostgreSQL, xây dựng cơ sở dữ liệu mẫu, và triển khai các hàm xử lý phục vụ cho hai kỹ thuật phân mảnh ngang đã chọn.
- **Phần 4: Thử nghiệm và đánh giá kết quả** – Trình bày kết quả thực nghiệm thu được khi áp dụng các kỹ thuật phân mảnh, phân tích hiệu quả thực thi và

so sánh đặc điểm giữa hai phương pháp trên cơ sở dữ liệu mẫu.

Thông qua việc triển khai và thử nghiệm các kỹ thuật phân mảnh dữ liệu, bài tập lớn không chỉ giúp chúng em hiểu rõ hơn về lý thuyết hệ phân tán mà còn rèn luyện kỹ năng làm việc thực tiễn với hệ quản trị cơ sở dữ liệu, từ đó nâng cao năng lực phân tích và thiết kế hệ thống lưu trữ dữ liệu hiệu quả.

PHẦN 2. CƠ SỞ LÝ THUYẾT.

2.1 Mục tiêu và lợi ích của phân mảnh dữ liệu

Phân mảnh dữ liệu (Data Fragmentation) là một kỹ thuật cốt lõi trong thiết kế và triển khai các hệ quản trị cơ sở dữ liệu phân tán (Distributed Database Management Systems - DDBMS). Mục tiêu chính của phân mảnh là chia nhỏ một bảng dữ liệu lớn thành các phần nhỏ hơn gọi là các phân mảnh (fragments), nhằm tăng hiệu quả lưu trữ và truy vấn trong môi trường có nhiều nút dữ liệu phân tán về mặt địa lý hoặc chức năng.

Việc phân mảnh không làm thay đổi ngữ nghĩa dữ liệu, đảm bảo rằng tập hợp tất cả các phân mảnh sẽ tái cấu trúc được bảng gốc một cách đầy đủ. Tùy thuộc vào chiến lược thiết kế, mỗi phân mảnh có thể được lưu trữ tại một vị trí vật lý khác nhau (trên các máy chủ khác nhau), hoặc cùng một vị trí nhưng phục vụ các mục tiêu tối ưu khác nhau.

Các mục tiêu và lợi ích chính của phân mảnh dữ liệu bao gồm:

- **Tăng hiệu suất truy vấn:** Các truy vấn chỉ cần thực thi trên các phân mảnh liên quan, thay vì quét toàn bộ bảng gốc. Điều này giúp giảm thời gian xử lý và tăng tốc độ phản hồi.
- **Tối ưu hóa lưu trữ:** Việc chia nhỏ bảng dữ liệu thành các phân mảnh giúp dễ dàng kiểm soát dung lượng từng phần, từ đó phân bổ hợp lý tài nguyên lưu trữ trên các máy chủ khác nhau.
- **Tăng khả năng mở rộng:** Các phân mảnh có thể được phân phối và xử lý song song trên nhiều nút, hỗ trợ khả năng mở rộng theo chiều ngang (horizontal scalability).
- **Cải thiện tính sẵn sàng và tin cậy:** Trong trường hợp một phân mảnh gặp sự cố, các phân mảnh khác vẫn có thể tiếp tục hoạt động bình thường. Việc sao lưu và phục hồi cũng dễ dàng hơn khi dữ liệu đã được phân tán.
- **Hỗ trợ phân quyền và bảo mật:** Việc quản lý phân mảnh độc lập cho phép áp dụng các chính sách bảo mật và phân quyền truy cập khác nhau tùy theo vị trí hoặc chức năng của phân mảnh.

2.2 Phân loại các phương pháp phân mảnh dữ liệu

Dựa trên cách dữ liệu được chia nhỏ, phân mảnh dữ liệu có thể được chia thành ba loại chính:

2.2.1 Phân mảnh ngang (Horizontal Fragmentation)

Phân mảnh ngang là quá trình chia bảng dữ liệu theo từng dòng (record). Mỗi phân mảnh sẽ chứa một tập con các bản ghi (rows) của bảng gốc, với đầy đủ các cột. Phương pháp này thường dựa trên điều kiện lọc (predicate) áp dụng trên một hoặc nhiều thuộc tính.

Ví dụ: Chia bảng `ratings` thành các phân mảnh theo khoảng giá trị của `userid`:

- `part_1`: chứa các dòng có `userid` từ 1 đến 10,000
- `part_2`: chứa các dòng có `userid` từ 10,001 đến 20,000

2.2.2 Phân mảnh dọc (Vertical Fragmentation)

Phân mảnh dọc chia bảng dữ liệu theo từng cột. Mỗi phân mảnh chứa một tập con các thuộc tính (columns), đồng thời vẫn giữ lại khóa chính của bảng để đảm bảo khả năng kết nối và tái tạo bảng gốc khi cần thiết.

Ví dụ: Bảng `users` có thể được phân mảnh thành:

- `fragment_login`: chứa các cột `userid`, `username`, `password`
- `fragment_profile`: chứa các cột `userid`, `name`, `email`, `birthdate`

2.2.3 Phân mảnh hỗn hợp (Hybrid Fragmentation)

Phân mảnh hỗn hợp là sự kết hợp giữa phân mảnh ngang và phân mảnh dọc. Dữ liệu có thể được phân mảnh theo dòng trước, sau đó tiếp tục phân mảnh theo cột, hoặc ngược lại. Phương pháp này thường được áp dụng trong các hệ thống có yêu cầu cao về tối ưu hóa dữ liệu theo nhiều chiều truy cập khác nhau.

2.3 Hai kỹ thuật phân mảnh ngang được mô phỏng trong báo cáo

Trong phạm vi bài tập này, nhóm tập trung mô phỏng và đánh giá hai kỹ thuật phân mảnh ngang tiêu biểu là: **phân mảnh theo khoảng** (Range Partitioning) và **phân mảnh vòng tròn** (Round Robin Partitioning). Đây là hai phương pháp thường được sử dụng trong các hệ thống cơ sở dữ liệu phân tán nhằm cải thiện hiệu suất và tính cân bằng tải.

2.3.1 Phân mảnh theo khoảng (Range Partitioning)

Phân mảnh theo khoảng là kỹ thuật chia dữ liệu thành các phân mảnh dựa trên khoảng giá trị của một thuộc tính cụ thể (thường là thuộc tính định danh như `userid`, `timestamp`, `rating`...). Mỗi phân mảnh sẽ lưu trữ các bản ghi thỏa mãn điều kiện nằm trong một khoảng giá trị nhất định.

Ví dụ: Nếu phân mảnh theo thuộc tính `userid`, ta có thể xây dựng các phân

mảnh như:

- `range_part0`: `userid` từ 1 đến 10,000
- `range_part1`: `userid` từ 10,001 đến 20,000
- `range_part2`: `userid` từ 20,001 trở đi

Kỹ thuật này phù hợp với các truy vấn theo vùng giá trị cụ thể và dễ dàng quản lý khi dữ liệu có tính phân bố theo khoảng.

2.3.2 Phân mảnh vòng tròn (Round Robin Partitioning)

Phân mảnh vòng tròn là phương pháp chia dữ liệu theo cách phân phối tuần tự từng bản ghi vào các phân mảnh, theo vòng lặp. Mỗi dòng dữ liệu được gán vào một phân mảnh dựa trên thứ tự xuất hiện, đảm bảo rằng dữ liệu được phân bố đồng đều giữa các phân mảnh.

Ví dụ: Với ba phân mảnh `rrobin_part0`, `rrobin_part1`, `rrobin_part2`:

- Bản ghi thứ 1 → `rrobin_part0`
- Bản ghi thứ 2 → `rrobin_part1`
- Bản ghi thứ 3 → `rrobin_part2`
- Bản ghi thứ 4 → `rrobin_part0`
- ...

Phương pháp này thích hợp cho các hệ thống yêu cầu xử lý song song và cân bằng tải giữa các nút dữ liệu.

2.4 Giới thiệu về PostgreSQL – hệ quản trị cơ sở dữ liệu mã nguồn mở

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở được phát triển và duy trì bởi cộng đồng toàn cầu. Với lịch sử phát triển hơn 30 năm, PostgreSQL được biết đến như một trong những hệ thống cơ sở dữ liệu mạnh mẽ, ổn định và linh hoạt nhất hiện nay.

Các đặc điểm nổi bật của PostgreSQL bao gồm:

- **Hỗ trợ phân mảnh (Partitioning):** Cho phép người dùng định nghĩa và triển khai các chiến lược phân mảnh (`range`, `list`, `hash`, `composite...`) một cách linh hoạt.
- **Hỗ trợ đa dạng kiểu dữ liệu:** Bao gồm các kiểu dữ liệu mở rộng như `JSON`, `XML`, `mảng`, `ENUM`, và hỗ trợ kiểu do người dùng định nghĩa.
- **Khả năng mở rộng cao:** Hỗ trợ các `trigger`, `stored procedure`, `user-defined functions`, `foreign data wrappers (FDW)` và mô-đun mở rộng.

- **Tuân thủ chuẩn SQL:** Hỗ trợ gần như đầy đủ các đặc tả SQL chuẩn (SQL:2011 và sau này).
- **Miễn phí và mã nguồn mở:** PostgreSQL được cấp phép theo giấy phép PostgreSQL License – một dạng BSD license, cho phép sử dụng và phân phối linh hoạt.

Nhờ những ưu điểm trên, PostgreSQL được lựa chọn làm nền tảng triển khai các kỹ thuật phân mảnh trong bài báo cáo này. Việc mô phỏng thực nghiệm trên PostgreSQL không chỉ phản ánh khả năng triển khai thực tế của các phương pháp phân mảnh, mà còn giúp người học hiểu rõ hơn về cấu trúc và đặc trưng của hệ quản trị cơ sở dữ liệu hiện đại.

PHẦN 3. CÀI ĐẶT VÀ TRIỂN KHAI.

3.1 Môi trường thực hiện

Để thực hiện việc mô phỏng hai phương pháp phân mảnh dữ liệu phổ biến trong hệ quản trị cơ sở dữ liệu phân tán, nhóm đã tiến hành xây dựng một môi trường lập trình trên máy tính cá nhân. Môi trường này bao gồm các công cụ và phần mềm mã nguồn mở, đảm bảo hỗ trợ đầy đủ cho việc xử lý dữ liệu, triển khai cơ sở dữ liệu, và thực hiện phân mảnh. Cụ thể, môi trường bao gồm:

- Hệ điều hành: Windows 10 hoặc 11 (64-bit)
- Ngôn ngữ lập trình: Python 3.12.x
- Hệ quản trị cơ sở dữ liệu: PostgreSQL (phiên bản từ 13 trở lên)
- Công cụ quản lý cơ sở dữ liệu: pgAdmin 4
- Mã nguồn mô phỏng: **quethaibinh/btl_csdlpt_nhom_23**

3.1.1 Cài đặt Python 3.12.x và thiết lập môi trường

1. **Cài đặt Python 3.12.x:** Truy cập trang chủ python.org và tải phiên bản Python 3.12.x phù hợp với hệ điều hành. Trong quá trình cài đặt, cần đánh dấu tùy chọn `Add Python to PATH` để sử dụng Python từ dòng lệnh.
2. **Tải và cấu hình mã nguồn dự án:** Thực hiện clone mã nguồn từ GitHub bằng lệnh:
 - `git clone https://github.com/quethaibinh/btl_csdlpt_nhom_23.git`
 - `cd btl_csdlpt_nhom_23`
3. **Cài đặt thư viện phụ thuộc:** Sử dụng công cụ `pip` để cài đặt thư viện kết nối PostgreSQL:
 - `pip install psycopg2`
 - `pip install pandas`

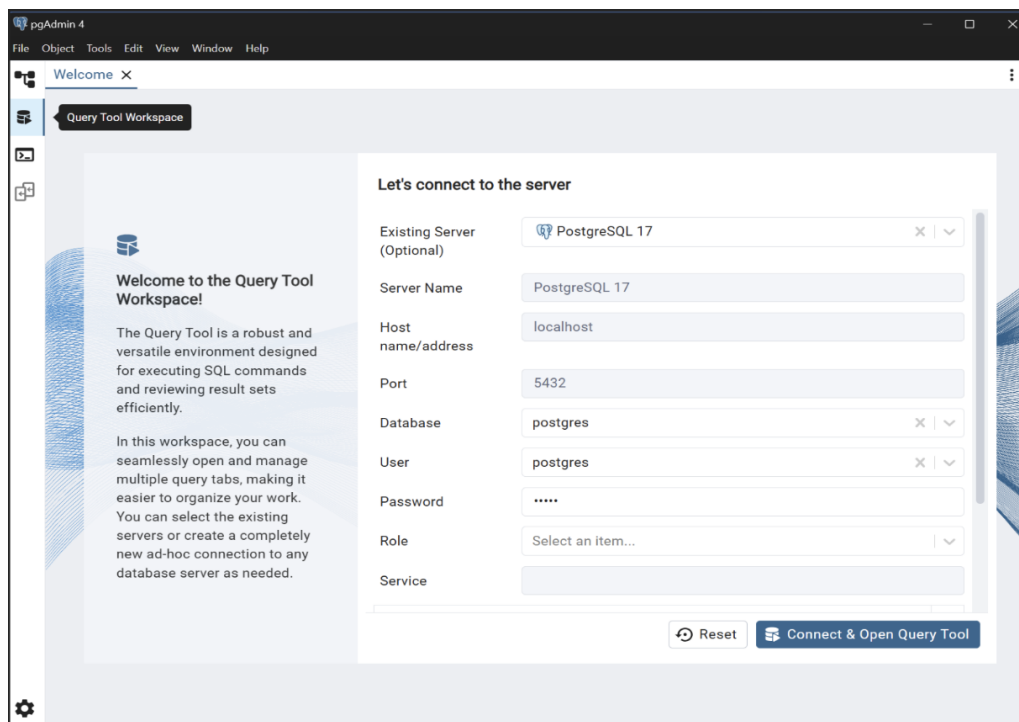
3.1.2 Cài đặt PostgreSQL và pgAdmin 4

PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở mạnh mẽ, được lựa chọn để thực hiện mô phỏng vì tính ổn định, hiệu năng cao, và hỗ trợ tốt các thao tác SQL nâng cao.

1. **Tải và cài đặt PostgreSQL:** Truy cập postgresql.org/download và chọn hệ điều hành phù hợp. Bộ cài đặt bao gồm cả pgAdmin 4, công cụ giao diện đồ

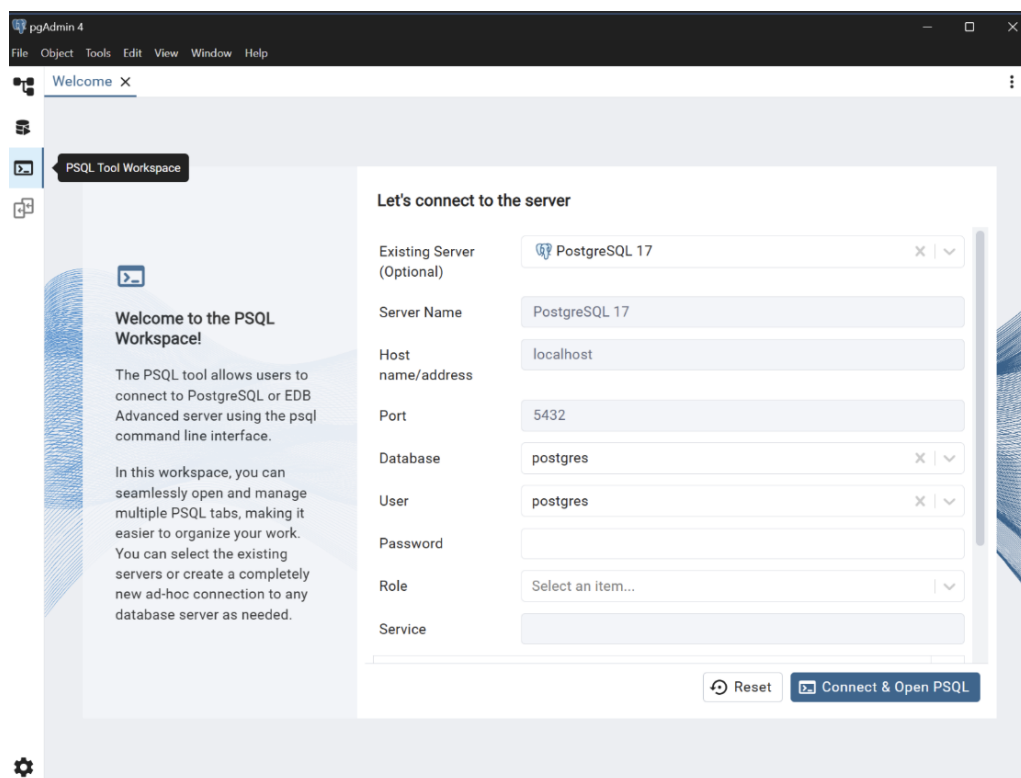
họa dùng để thao tác với cơ sở dữ liệu.

2. **Khởi tạo cơ sở dữ liệu:** Sau khi cài đặt, mở ứng dụng pgAdmin 4, đăng nhập bằng mật khẩu đã thiết lập. Tiến hành tạo một cơ sở dữ liệu mới để test xem pgadmin đã hoạt động chưa (ví dụ: `csdlpt_test`).
3. **Sử dụng Query Tool:** Truy cập Query Tool để thực hiện các truy vấn SQL, như tạo bảng, chèn dữ liệu, và kiểm tra kết quả phân mảnh.



Hình 3.1: Giao diện Query Tool trong pgAdmin 4

4. **Sử dụng PSQL Tool (nếu cần):** Đây là công cụ dòng lệnh cung cấp cách tương tác trực tiếp với PostgreSQL, phù hợp cho các tác vụ tự động hóa hoặc kiểm tra nhanh.



Hình 3.2: Giao diện PSQL Tool trong pgAdmin 4

Sau khi hoàn tất các bước trên, môi trường đã sẵn sàng để tiến hành nhập dữ liệu, cài đặt logic phân mảnh và thực hiện mô phỏng các phương pháp phân mảnh dữ liệu.

3.2 Chuẩn bị dữ liệu

Để mô phỏng các phương pháp phân mảnh, nhóm sử dụng tập dữ liệu **MovieLens 10M Dataset**, một bộ dữ liệu lớn và uy tín được cung cấp bởi GroupLens. Bộ dữ liệu này thường được sử dụng trong các nghiên cứu về hệ thống gợi ý.

3.2.1 Tải và xử lý dữ liệu

1. Truy cập liên kết: <http://files.grouplens.org/datasets/movielens/ml-10m.zip>
2. Giải nén và tìm đến tệp `ratings.dat`
3. Di chuyển tệp `ratings.dat` vào thư mục chứa mã nguồn dự án để dễ dàng thao tác

3.2.2 Định dạng dữ liệu

Tệp `ratings.dat` chứa khoảng 10 triệu dòng, mỗi dòng là một đánh giá của người dùng đối với một bộ phim. Định dạng mỗi dòng như sau:

UserID::MovieID::Rating::Timestamp

Các trường dữ liệu:

- **UserID**: Mã định danh người dùng (kiểu số nguyên)
- **MovieID**: Mã định danh bộ phim (kiểu số nguyên)
- **Rating**: Điểm đánh giá (thang điểm 0.5 đến 5.0, bước 0.5)
- **Timestamp**: Dấu thời gian đánh giá (tính bằng giây kể từ thời điểm 00:00 UTC ngày 1/1/1970 - chuẩn UNIX time)

Ví dụ một số dòng dữ liệu:

1::122::5::838985046

1::185::5::838983525

1::231::5::838983392

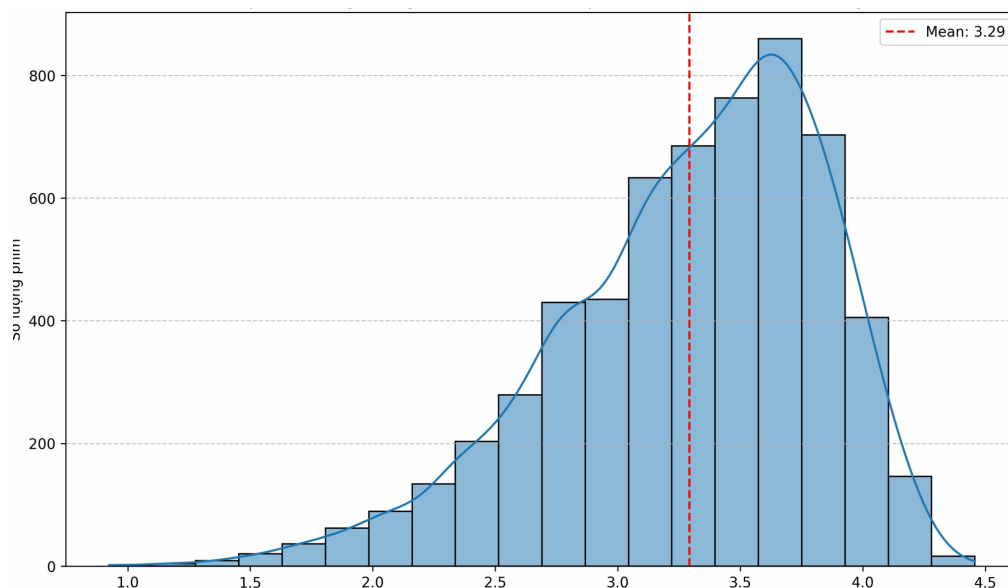
3.2.3 Phân tích bộ dữ liệu

Nhóm chúng em có sử dụng thuật toán Data Visualize để phân tích data đầu vào, từ đó có cái nhìn tổng quan hơn về một lượng data tương đối lớn trong bài tập lớn này.

a, Tổng quan về dữ liệu.

- **Tổng số đánh giá**: 10,000,054
- **Số lượng người dùng**: 69,878
- **Số lượng phim**: 10,677
- **Thang điểm đánh giá**: 0.5 đến 5.0
- **Thời gian**: 09/01/1995 đến 05/01/2009

b, Biểu đồ phân phối rating trung bình của các bộ phim.



Hình 3.3: Biểu đồ rating trung bình.

Biểu đồ thể hiện phân phối điểm đánh giá trung bình của các phim (chỉ tính các phim có ít nhất 100 đánh giá):

- Điểm đánh giá trung bình hầu hết nằm trong khoảng 3.0-4.0
- Rất ít phim có điểm đánh giá trung bình dưới 2.5 hoặc trên 4.5
- Phân phối có dạng chuông, cho thấy tính đại diện của mẫu

Phân phối này có dạng gần với phân phối chuẩn nhưng có độ lệch nhẹ về phía điểm cao hơn, với giá trị trung bình khoảng 3.5 (được đánh dấu bằng đường đỏ trên biểu đồ). Đặc biệt, hầu như không có phim nào có điểm trung bình dưới 2.0 hoặc trên 4.7, cho thấy hiệu ứng trung bình hóa khi có nhiều người dùng đánh giá. Điều này gợi ý rằng thang điểm thực tế đang được sử dụng là khoảng 2.0-4.7 thay vì 0.5-5.0 như thiết kế ban đầu, và điều này nên được tính đến khi thiết kế thuật toán gợi ý

Sau khi nạp dữ liệu vào hệ thống, nhóm tiến hành xây dựng các hàm tiền xử lý, phân mảnh và lưu trữ dữ liệu phân mảnh trong các bảng tương ứng để phục vụ cho các bước mô phỏng tiếp theo.

3.3 Xây dựng các hàm xử lý.

3.3.1 Hàm LoadRatings

Hàm `loadratings` đóng vai trò khởi đầu của quy trình, với nhiệm vụ đọc dữ liệu từ tệp nguồn và tải vào cơ sở dữ liệu.

a, Mục đích

Hàm này thực hiện quá trình ETL (Extract, Transform, Load) cơ bản để tải dữ liệu từ tệp `ratings.dat` vào một bảng chính trong cơ sở dữ liệu, chuẩn bị cho việc phân mảnh tiếp theo.

b, Các tham số

- `ratingstablename`: Tên bảng đích sẽ chứa dữ liệu
- `ratingsfilepath`: Đường dẫn đến tệp dữ liệu nguồn
- `openconnection`: Kết nối cơ sở dữ liệu đang mở

c, Luồng hoạt động của mã nguồn

Hàm `loadratings` được thiết kế để thực hiện một chuỗi các thao tác trên cơ sở dữ liệu một cách tuần tự và hiệu quả. Dưới đây là diễn giải chi tiết về luồng hoạt động của từng dòng lệnh trong hàm.

```
def loadratings(ratingstablename, ratingsfilepath,
               openconnection):
```

```

2      """
3      Function to load data in @ratingsfilepath file to a
table called @ratingstablename.
4      """
5      create_db(DATABASE_NAME)
6      con = openconnection
7      cur = con.cursor()
8
9      # Create table with all columns initially
10     cur.execute(f"""
11         CREATE TABLE {ratingstablename} (
12             userid INTEGER,
13             extra1 CHAR,
14             movieid INTEGER,
15             extra2 CHAR,
16             rating FLOAT,
17             extra3 CHAR,
18             timestamp BIGINT
19         )
20     """)
21
22     # Use COPY command for fast data loading
23     with open(ratingsfilepath, 'r') as f:
24         cur.copy_from(f, ratingstablename, sep=':')
25
26     # Drop columns in a single command
27     cur.execute(f"ALTER TABLE {ratingstablename} DROP
28         COLUMN extra1, DROP COLUMN extra2, DROP COLUMN extra3
29         , DROP COLUMN timestamp")
30
31     con.commit()
32     cur.close()

```

Giải thích luồng hoạt động: Toàn bộ quá trình được thực hiện trong một giao dịch (transaction) duy nhất để đảm bảo tính toàn vẹn dữ liệu. Nếu bất kỳ bước nào thất bại, toàn bộ thao tác sẽ được hủy bỏ.

1. Bước 1: Khởi tạo môi trường làm việc.

- `con = openconnection`: Hàm nhận một đối tượng kết nối psycopg2 đã được mở sẵn, gán vào biến cục bộ `con`.

- `cur = con.cursor()`: Từ đối tượng kết nối, một đối tượng `cursor` được tạo ra. Cursor đóng vai trò là một con trỏ, cho phép duyệt và thực thi các câu lệnh SQL trên cơ sở dữ liệu. Mọi tương tác với CSDL sau đó đều thông qua cursor này.

2. Bước 2: Tạo bảng với cấu trúc trung gian.

- `cur.execute(f"CREATE TABLE ...")`: Một câu lệnh DDL (Ngôn ngữ Định nghĩa Dữ liệu) `CREATE TABLE` được thực thi.
- **Lý do thiết kế**: Bảng được tạo với một cấu trúc rộng hơn mức cần thiết, bao gồm các cột `extra1`, `extra2`, `extra3`. Đây là một kỹ thuật xử lý dữ liệu thô. Tập nguồn `ratings.dat` sử dụng dấu hai chấm đôi (‘::’) làm dấu phân cách. Tuy nhiên, hàm `copy_from` được cấu hình với dấu phân cách là một dấu hai chấm (‘sep=’). Do đó, một dòng như `1::1193::5::978300760` sẽ được đọc thành các trường: `'1'`, `':'`, `'1193'`, `':'`, `'5'`, `':'`, `'978300760'`. Các cột `extra` được tạo ra chính là để "hứng" các ký tự hai chấm dư thừa này trong quá trình nạp dữ liệu.

3. Bước 3: Tải dữ liệu số lượng lớn (Bulk Loading).

- `with open(ratingsfilepath, 'r') as f::` Mở tệp dữ liệu tại đường dẫn `ratingsfilepath` ở chế độ chỉ đọc (‘r’). Việc sử dụng câu lệnh `with` đảm bảo tệp sẽ được tự động đóng lại sau khi hoàn tất, kể cả khi có lỗi xảy ra.
- `cur.copy_from(f, ratingstablename, sep=':')`: Đây là bước quan trọng nhất về mặt hiệu năng. Thay vì đọc từng dòng của tệp và thực thi lệnh `INSERT` (gây tốn kém về thời gian và chi phí giao tiếp mạng), hàm sử dụng `copy_from`. Phương thức này của `psycopg2` gọi trực tiếp lệnh `COPY` của PostgreSQL, một tiện ích được tối ưu hóa ở mức CSDL để nạp dữ liệu từ một luồng (stream) vào bảng với tốc độ rất cao.

4. Bước 4: Chuẩn hóa lại cấu trúc bảng.

- `cur.execute(f"ALTER TABLE ... DROP COLUMN ...")`: Sau khi dữ liệu đã nằm an toàn trong bảng, một câu lệnh DDL `ALTER TABLE` được thực thi để loại bỏ các cột đệm không cần thiết (`extra1`, `extra2`, `extra3`) và cột `timestamp`. Kết quả là bảng `ratingstablename` có được cấu trúc cuối cùng, tinh gọn và đúng với yêu cầu nghiệp vụ, chỉ bao gồm `userid`, `movieid`, `rating`.

5. Bước 5: Xác nhận giao dịch và giải phóng tài nguyên.

- `con.commit()`: Tất cả các thao tác từ Bước 2 đến Bước 4 (tạo bảng, sao chép dữ liệu, thay đổi cấu trúc) được thực hiện trong một khối giao dịch. Lệnh này sẽ xác nhận và lưu vĩnh viễn tất cả các thay đổi vào cơ sở dữ liệu.
- `cur.close()`: Giải phóng các tài nguyên đã được cấp cho đối tượng `cursor`. Đây là một hành động dọn dẹp cần thiết để tránh rò rỉ tài nguyên trong ứng dụng.

Hàm bắt đầu bằng việc tạo kết nối và `cursor` từ tham số kết nối được truyền vào, cho phép thực thi các lệnh SQL. Tiếp theo, hàm tạo một bảng ban đầu với schema rộng hơn schema cuối cùng để tương ứng với định dạng tệp nguồn có dấu phân cách `::`. Dữ liệu được tải nhanh chóng bằng lệnh `copy_from` của PostgreSQL, sau đó loại bỏ các cột không cần thiết, chỉ giữ lại `userid`, `movieid` và `rating`. Cuối cùng, hàm `commit` giao dịch và đóng `cursor`.

3.3.2 Hàm Range_Partition

Hàm `rangepartition` thực hiện phân mảnh ngang bảng chính dựa trên các khoảng giá trị của thuộc tính `rating`.

a, Mục đích

Hàm này chia dữ liệu từ bảng chính thành nhiều phân mảnh dựa trên các khoảng giá trị `rating`, giúp tối ưu hóa các truy vấn liên quan đến khoảng giá trị `rating` cụ thể.

b, Các tham số

- `ratingtablename`: Tên bảng chính chứa dữ liệu nguồn
- `numberofpartitions`: Số lượng phân mảnh cần tạo
- `openconnection`: Kết nối cơ sở dữ liệu đang mở

c, Luồng hoạt động của mã nguồn

Hàm `rangepartition` được thiết kế để phân chia một bảng chính (chứa dữ liệu xếp hạng) thành nhiều bảng con dựa trên các khoảng giá trị của cột `"rating"`. Mục tiêu là tạo ra các phân vùng dữ liệu để tối ưu hóa việc truy vấn và quản lý.

```

1 def rangepartition(ratingtablename, numberofpartitions,
2   openconnection):
3     """
4     Function to create partitions of main table based on
5     range of ratings.
6     """
7     con = openconnection

```

```
6     cur = con.cursor()
7     delta = 5.0 / numberofpartitions
8
9     # Create all tables first for better transaction
    handling
10    for i in range(numberofpartitions):
11        table_name = f"{RANGE_TABLE_PREFIX}{i}"
12        cur.execute(f"CREATE TABLE {table_name} (userid
    INTEGER, movieid INTEGER, rating FLOAT)")
13
14    # Insert data using direct approach
15    for i in range(numberofpartitions):
16        minRange = i * delta
17        maxRange = minRange + delta
18        table_name = f"{RANGE_TABLE_PREFIX}{i}"
19
20        if i == 0:
21            cur.execute(f"""
22                INSERT INTO {table_name} (userid, movieid,
rating)
23                SELECT userid, movieid, rating FROM {
ratingstablename}
24                WHERE rating >= {minRange} AND rating <= {
maxRange}
25                """)
26        else:
27            cur.execute(f"""
28                INSERT INTO {table_name} (userid, movieid,
rating)
29                SELECT userid, movieid, rating FROM {
ratingstablename}
30                WHERE rating > {minRange} AND rating <= {
maxRange}
31                """)
32
33    con.commit()
34    cur.close()
```

Giải thích luồng hoạt động: Toàn bộ quá trình tạo bảng phân vùng và chèn dữ liệu được thực hiện trong một giao dịch (transaction) duy nhất để đảm bảo tính toàn

ven dữ liệu. Nếu bất kỳ bước nào thất bại, toàn bộ thao tác sẽ được hủy bỏ, tránh tình trạng dữ liệu không nhất quán.

1. Bước 1: Khởi tạo kết nối và tính toán độ rộng khoảng.

- `con = openconnection`: Hàm nhận một đối tượng kết nối cơ sở dữ liệu đã được mở sẵn (`openconnection`) và gán nó cho biến cục bộ `con`. Đây là kết nối sẽ được sử dụng để tương tác với cơ sở dữ liệu.
- `cur = con.cursor()`: Từ đối tượng kết nối, một đối tượng con trỏ (`cursor`) được tạo ra. Con trỏ này cho phép thực thi các câu lệnh SQL trên cơ sở dữ liệu. Mọi tương tác với CSDL sau đó đều thông qua con trỏ này.
- `delta = 5.0 / numberofpartitions`: Giá trị `delta` được tính toán. Đây là độ rộng của mỗi khoảng xếp hạng mà chúng ta sẽ sử dụng để phân chia dữ liệu. Giả định rằng điểm xếp hạng nằm trong khoảng từ 0.0 đến 5.0. `numberofpartitions` là số lượng phân vùng mà chúng ta muốn tạo.

2. Bước 2: Tạo trước các bảng phân vùng.

- Vòng lặp `for i in range(numberofpartitions)`: Lặp lại `numberofpartitions` lần, mỗi lần lặp tương ứng với một phân vùng cần tạo.
- `table_name = f"{RANGE_TABLE_PREFIX}{i}"`: Tên của bảng phân vùng được tạo ra bằng cách kết hợp một tiền tố (`RANGE_TABLE_PREFIX`) với chỉ số hiện tại của vòng lặp (`i`). Điều này giúp đặt tên bảng một cách có hệ thống (ví dụ: `range_table_0`, `range_table_1`, v.v.).
- `cur.execute(f"CREATE TABLE {table_name} (userid INTEGER, movieid INTEGER, rating FLOAT)")`: Câu lệnh SQL `CREATE TABLE` được thực thi để tạo bảng mới. Mỗi bảng phân vùng sẽ có cấu trúc gồm ba cột: `userid` (kiểu số nguyên), `movieid` (kiểu số nguyên), và `rating` (kiểu số thực).
- **Lý do thiết kế**: Việc tạo tất cả các bảng con trước khi chèn dữ liệu là một kỹ thuật tốt để quản lý giao dịch. Nếu có bất kỳ lỗi nào xảy ra trong quá trình tạo bảng, giao dịch có thể được xử lý tốt hơn (ví dụ: `ROLLBACK` để quay lại trạng thái trước đó) thay vì để lại các bảng được tạo một cách không hoàn chỉnh hoặc gây ra lỗi sau này khi chèn dữ liệu.

3. Bước 3: Chèn dữ liệu vào các bảng phân vùng.

- Vòng lặp `for i in range(numberofpartitions)`: Vòng lặp này tiếp tục duyệt qua từng phân vùng để chọn và chèn dữ liệu phù hợp từ bảng chính vào bảng con tương ứng.
- `minRange = i * delta` và `maxRange = minRange + delta`: Tính toán giá trị xếp hạng tối thiểu và tối đa cho khoảng hiện tại của phân vùng.
- **Điều kiện chèn dữ liệu (đảm bảo mỗi rating chỉ thuộc một phân vùng duy nhất):**
 - `if i == 0`: Đối với phân vùng đầu tiên (khi `i` bằng 0), dữ liệu được chèn bao gồm các xếp hạng nằm trong khoảng `rating >= {minRange}` AND `rating <= {maxRange}`. Điều này bao gồm cả giá trị `minRange` và `maxRange`.
 - `else`: Đối với các phân vùng tiếp theo (`i` lớn hơn 0), dữ liệu được chèn bao gồm các xếp hạng nằm trong khoảng `rating > {minRange}` AND `rating <= {maxRange}`. Việc sử dụng toán tử `>` (lớn hơn) cho `minRange` thay vì `>=` (lớn hơn hoặc bằng) là rất quan trọng. Nó đảm bảo rằng một giá trị xếp hạng ở ranh giới giữa hai khoảng (ví dụ: xếp hạng 1.0 khi `delta` là 1.0) sẽ chỉ được chèn vào một bảng duy nhất (cụ thể là bảng của khoảng lớn hơn hoặc bằng nó), tránh việc trùng lặp dữ liệu giữa các phân vùng.
- `cur.execute(f"INSERT INTO {table_name} (...) SELECT ... FROM {ratingtablename} WHERE rating ...")`: Câu lệnh SQL `INSERT INTO ... SELECT ... WHERE ...` được thực thi. Nó chọn các cột `userid`, `movieid`, và `rating` từ bảng chính (`ratingtablename`) và lọc chúng dựa trên điều kiện `WHERE` của khoảng xếp hạng đã tính toán, sau đó chèn các bản ghi này vào bảng con (`table_name`) tương ứng.

4. Bước 4: Xác nhận giao dịch và giải phóng tài nguyên.

- `con.commit()`: Sau khi tất cả các bảng phân vùng đã được tạo và dữ liệu đã được chèn vào chúng, lệnh `commit()` được gọi. Lệnh này xác nhận và lưu vĩnh viễn tất cả các thay đổi (tạo bảng và chèn dữ liệu) vào cơ sở dữ liệu.
- `cur.close()`: Con trỏ (`cur`) được đóng để giải phóng các tài nguyên đã được cấp cho nó. Đây là một hành động dọn dẹp cần thiết để tránh rò rỉ tài nguyên trong ứng dụng. Kết nối cơ sở dữ liệu (`con`) không được đóng

trong hàm này, vì nó được nhận dưới dạng tham số và người gọi hàm có trách nhiệm quản lý việc đóng kết nối chính.

Hàm tính toán độ rộng mỗi khoảng (delta) dựa trên số phân mảnh, tạo trước các bảng phân mảnh để tối ưu giao dịch và phân phối dữ liệu sao cho mỗi giá trị rating chỉ thuộc một phân mảnh duy nhất.

3.3.3 Hàm RoundRobin_Partition

Hàm `roundrobinpartition` thực hiện phân mảnh ngang bảng chính sử dụng phương pháp vòng tròn (round robin).

a, Mục đích

Phân phối dữ liệu từ bảng chính vào các phân mảnh theo thứ tự luân phiên, cân bằng tải đều đặn.

b, Luồng hoạt động của mã nguồn

Hàm `roundrobinpartition` được thiết kế để phân chia dữ liệu từ một bảng chính thành nhiều bảng con sử dụng phương pháp Round Robin (phân phối luân phiên). Phương pháp này đảm bảo các bản ghi được phân phối đều giữa các phân vùng, giúp cân bằng tải và tối ưu hóa hiệu suất truy vấn.

```

1 def roundrobinpartition(ratingstablename,
2   numberofpartitions, openconnection):
3     """
4     Function to create partitions of main table using round
5     robin approach.
6     """
7     con = openconnection
8     cur = con.cursor()
9
10    # Create all tables first
11    for i in range(numberofpartitions):
12        table_name = f"{RROBIN_TABLE_PREFIX}{i}"
13        cur.execute(f"CREATE TABLE {table_name} (userid
14                     INTEGER, movieid INTEGER, rating FLOAT)")
15
16    # Add row numbers to the source table for efficient
17    partitioning
18    cur.execute(f"""
19        CREATE TEMPORARY TABLE temp_table AS
20        SELECT userid, movieid, rating,
21               ROW_NUMBER() OVER() AS row_id
22        FROM {ratingstablename}

```



```

19         """
20
21     # Insert into partitions based on mod calculation
22     for i in range(numberofpartitions):
23         table_name = f"{RROBIN_TABLE_PREFIX}{i}"
24         cur.execute(f"""
25             INSERT INTO {table_name} (userid, movieid,
26             rating)
27             SELECT userid, movieid, rating
28             FROM temp_table
29             WHERE MOD((row_id - 1), {numberofpartitions}) =
30             {i}
31         """)
32
33     # Drop temporary table
34     cur.execute("DROP TABLE temp_table")
35
36     con.commit()
37     cur.close()

```

Giải thích luồng hoạt động: Toàn bộ quá trình tạo bảng phân vùng và chèn dữ liệu được thực hiện trong một giao dịch (transaction) duy nhất để đảm bảo tính toàn vẹn dữ liệu. Nếu bất kỳ bước nào thất bại, toàn bộ thao tác sẽ được hủy bỏ, tránh tình trạng dữ liệu không nhất quán.

1. Bước 1: Khởi tạo kết nối và tạo các bảng phân vùng.

- `con = openconnection`: Hàm nhận một đối tượng kết nối cơ sở dữ liệu đã được mở sẵn (`openconnection`) và gán nó cho biến cục bộ `con`. Kết nối này sẽ được sử dụng để tương tác với cơ sở dữ liệu.
- `cur = con.cursor()`: Từ đối tượng kết nối, một đối tượng con trỏ (`cursor`) được tạo ra. Con trỏ này cho phép thực thi các câu lệnh SQL.
- Vòng lặp `for i in range(numberofpartitions)`: Đoạn mã này lặp lại `numberofpartitions` lần. Trong mỗi lần lặp, nó sẽ tạo ra một bảng con mới.
- `table_name = f"{RROBIN_TABLE_PREFIX}{i}"`: Tên của bảng con được tạo ra bằng cách ghép một tiền tố (ví dụ: `rrobin_table_`) với chỉ số của phân vùng (`i`).
- `cur.execute(f"CREATE TABLE {table_name} (userid IN-`

TEGER, movieid INTEGER, rating FLOAT)"): Câu lệnh SQL CREATE TABLE được thực thi để tạo bảng mới. Mỗi bảng con sẽ có ba cột: userid (kiểu số nguyên), movieid (kiểu số nguyên), và rating (kiểu số thực). Việc tạo tất cả các bảng trước khi chèn dữ liệu giúp đảm bảo tính nhất quán của giao dịch.

2. Bước 2: Thêm số thứ tự hàng vào bảng tạm thời.

- `cur.execute(f"""CREATE TEMPORARY TABLE temp_table AS SELECT ..., ROW_NUMBER() OVER() AS row_id FROM ...""")`: Một bảng tạm thời có tên `temp_table` được tạo ra. Bảng này chứa tất cả các cột từ bảng chính (`userid`, `movieid`, `rating`) cùng với một cột mới là `row_id`.
- `ROW_NUMBER() OVER() AS row_id`: Đây là một hàm cửa sổ (window function) của SQL. Nó gán một số thứ tự duy nhất cho mỗi hàng trong bảng chính `ratingtablename`. Số thứ tự này sẽ bắt đầu từ 1 và tăng dần cho mỗi hàng. Việc sử dụng bảng tạm với `row_id` là cốt lõi của phương pháp Round Robin, cho phép chúng ta dễ dàng phân phối các hàng dựa trên chỉ số của chúng.

3. Bước 3: Chèn dữ liệu vào các bảng phân vùng dựa trên phép toán modulo.

- Vòng lặp `for i in range(numberofpartitions)`: Vòng lặp này duyệt qua từng phân vùng để chèn dữ liệu.
- `table_name = f"{RROBIN_TABLE_PREFIX}{i}"`: Xác định tên của bảng phân vùng đích.
- `cur.execute(f"""INSERT INTO {table_name} (...) SELECT ... FROM temp_table WHERE MOD((row_id - 1), {numberofpartitions}) = {i}""")`: Đây là bước chính của phân vùng Round Robin.
 - `MOD((row_id - 1), {numberofpartitions})`: Phép toán modulo được sử dụng. 'row_id' là số thứ tự hàng (bắt đầu từ 1). Chúng ta trừ đi 1 ('row_id - 1') để chuyển chỉ số hàng về hệ số 0 (tức là 0, 1, 2,...). Sau đó, chúng ta thực hiện phép chia lấy dư (modulo) với 'numberofpartitions'. Kết quả của phép toán này sẽ là một số nguyên từ 0 đến 'numberofpartitions - 1'.
 - `= {i}`: Điều kiện này so khớp kết quả của phép toán modulo với chỉ số `i` của phân vùng hiện tại. Điều này đảm bảo rằng các hàng sẽ được phân phối luân phiên: hàng có 'row_id - 1' chia hết cho 'numberof-

partitions' sẽ vào bảng 'i=0', hàng tiếp theo vào bảng 'i=1', và cứ thế tiếp tục.

– Ví dụ: Nếu 'numberofpartitions' là 3:

* Hàng 1 (row_id=1): $\text{MOD}((1-1), 3) = \text{MOD}(0, 3) = 0 \rightarrow$ vào bảng 'rrobin_table_0'.

* Hàng 2 (row_id=2): $\text{MOD}((2-1), 3) = \text{MOD}(1, 3) = 1 \rightarrow$ vào bảng 'rrobin_table_1'.

* Hàng 3 (row_id=3): $\text{MOD}((3-1), 3) = \text{MOD}(2, 3) = 2 \rightarrow$ vào bảng 'rrobin_table_2'.

* Hàng 4 (row_id=4): $\text{MOD}((4-1), 3) = \text{MOD}(0, 3) = 0 \rightarrow$ lại vào bảng 'rrobin_table_0'.

Quá trình này đảm bảo mỗi bản ghi được chèn vào một và chỉ một bảng phân vùng, và các bản ghi được phân phối tương đối đều giữa các bảng.

4. Bước 4: Xóa bảng tạm thời và hoàn tất giao dịch.

- `cur.execute("DROP TABLE temp_table")`: Sau khi tất cả dữ liệu đã được chèn vào các bảng phân vùng, bảng tạm thời `temp_table` không còn cần thiết và được xóa để giải phóng tài nguyên.
- `con.commit()`: Tất cả các thao tác từ Bước 1 đến Bước 3 (tạo bảng, tạo bảng tạm, chèn dữ liệu) được thực hiện trong một khối giao dịch. Lệnh này sẽ xác nhận và lưu vĩnh viễn tất cả các thay đổi vào cơ sở dữ liệu.
- `cur.close()`: Giải phóng các tài nguyên đã được cấp cho đối tượng `con` trở. Đây là một hành động dọn dẹp cần thiết để tránh rò rỉ tài nguyên. Kết nối cơ sở dữ liệu (`con`) không được đóng trong hàm này, cho phép người gọi hàm quản lý việc đóng kết nối chính.

Sử dụng bảng tạm với cột `row_id` để xác định thứ tự và phép toán modulo để phân phối đều, sau đó xóa bảng tạm để dọn dẹp.

3.3.4 Hàm RoundRobin_Insert

Hàm `roundrobininsert` chèn một bộ dữ liệu mới vào cả bảng chính và phân mảnh vòng tròn thích hợp.

a, Mục đích

Duy trì tính nhất quán giữa bảng chính và các phân mảnh vòng tròn khi thêm dữ liệu mới.

b, Luồng hoạt động của mã nguồn

Hàm `roundrobininsert` được thiết kế để chèn một bản ghi mới vào bảng chính và đồng thời vào một bảng phân vùng cụ thể dựa trên phương pháp Round Robin (phân phối luân phiên). Điều này đảm bảo rằng mỗi bản ghi mới được phân phối đều đặn giữa các phân vùng hiện có.

```

1 def roundrobininsert(ratingtablename, userid, itemid,
2   rating, openconnection):
3     """
4     Function to insert a new row into the main table and
5     specific partition based on round robin
6     approach.
7     """
8     con = openconnection
9     cur = con.cursor()
10
11    # Insert into main table
12    cur.execute(f"INSERT INTO {ratingtablename} (userid,
13              movieid, rating) VALUES (%s, %s, %s)",
14              (userid, itemid, rating))
15
16    # Get row count and calculate partition
17    cur.execute(f"SELECT COUNT(*) FROM {ratingtablename}")
18    total_rows = cur.fetchone()[0]
19
20    numberofpartitions = count_partitions(
21        RROBIN_TABLE_PREFIX, con)
22    index = (total_rows - 1) % numberofpartitions
23
24    # Insert into partition
25    table_name = f"{RROBIN_TABLE_PREFIX}{index}"
26    cur.execute(f"INSERT INTO {table_name} (userid, movieid
27              , rating) VALUES (%s, %s, %s)",
28              (userid, itemid, rating))
29
30    con.commit()
31    cur.close()

```

Giải thích luồng hoạt động: Toàn bộ quá trình chèn dữ liệu vào cả bảng chính và bảng phân vùng được thực hiện trong một giao dịch (transaction) duy nhất. Điều

này đảm bảo tính toàn vẹn dữ liệu: nếu một trong hai thao tác chèn thất bại, cả giao dịch sẽ được hoàn tác, ngăn chặn tình trạng dữ liệu không đồng bộ giữa bảng chính và các bảng phân vùng.

1. Bước 1: Khởi tạo kết nối và chèn dữ liệu vào bảng chính.

- `con = openconnection`: Hàm nhận một đối tượng kết nối cơ sở dữ liệu đã được mở sẵn (`openconnection`) và gán nó cho biến cục bộ `con`. Đây là kết nối sẽ được sử dụng để tương tác với cơ sở dữ liệu.
- `cur = con.cursor()`: Từ đối tượng kết nối, một đối tượng con trỏ (`cursor`) được tạo ra. Con trỏ này cho phép thực thi các câu lệnh SQL.
- `cur.execute(f"INSERT INTO {ratingtablename} (userid, movieid, rating) VALUES (%s, %s, %s)", (userid, itemid, rating))`: Một câu lệnh INSERT được thực thi để chèn bản ghi mới vào bảng chính `ratingtablename`. Dữ liệu `userid`, `itemid` (được ánh xạ thành `movieid` trong bảng) và `rating` được truyền vào câu lệnh SQL một cách an toàn thông qua tham số (%s), giúp ngăn chặn các cuộc tấn công SQL injection.

2. Bước 2: Lấy tổng số hàng và tính toán chỉ mục phân vùng.

- `cur.execute(f"SELECT COUNT(*) FROM {ratingtablename}")`: Sau khi bản ghi mới đã được chèn vào bảng chính, một câu lệnh SELECT COUNT(*) được thực thi để lấy tổng số hàng hiện có trong bảng chính `ratingtablename`.
- `total_rows = cur.fetchone()[0]`: Kết quả của câu lệnh COUNT(*) (là một hàng duy nhất chứa một giá trị số) được truy xuất và gán vào biến `total_rows`.
- `numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, con)`: Hàm `count_partitions` (không được định nghĩa trong đoạn mã này nhưng được giả định là tồn tại) được gọi để xác định tổng số lượng bảng phân vùng hiện có dựa trên tiền tố tên bảng (`RROBIN_TABLE_PREFIX`). Số lượng phân vùng này là cần thiết để tính toán chỉ mục Round Robin.
- `index = (total_rows - 1) % numberofpartitions`: Đây là bước tính toán chỉ mục phân vùng đích theo phương pháp Round Robin.
 - `total_rows - 1`: Vì COUNT(*) trả về tổng số hàng (bắt đầu từ 1), chúng ta trừ đi 1 để chuyển nó thành chỉ số dựa trên 0 (tức là 0, 1, 2, ...).

- `% numberofpartitions`: Phép toán modulo được thực hiện với tổng số phân vùng. Kết quả của phép toán này sẽ là một số nguyên từ 0 đến `numberofpartitions - 1`, đại diện cho chỉ mục của bảng phân vùng mà bản ghi mới này sẽ được chèn vào. Điều này đảm bảo rằng các bản ghi được phân phối đều đặn theo chu kỳ giữa các phân vùng.

3. Bước 3: Chèn dữ liệu vào bảng phân vùng đích.

- `table_name = f"{RROBIN_TABLE_PREFIX}{index}":` Tên của bảng phân vùng đích được xây dựng dựa trên tiền tố và chỉ mục đã tính toán ở Bước 2.
- `cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) VALUES (%s, %s, %s)", (userid, itemid, rating))`: Bản ghi mới lại được chèn vào bảng phân vùng cụ thể đã xác định. Các giá trị `userid`, `itemid`, và `rating` tương tự như khi chèn vào bảng chính.

4. Bước 4: Xác nhận giao dịch và giải phóng tài nguyên.

- `con.commit()`: Lệnh này xác nhận và lưu vĩnh viễn tất cả các thay đổi đã thực hiện trong giao dịch (bao gồm cả việc chèn vào bảng chính và bảng phân vùng) vào cơ sở dữ liệu.
- `cur.close()`: Con trỏ (`cur`) được đóng để giải phóng các tài nguyên đã được cấp cho nó. Đây là một hành động dọn dẹp cần thiết để tránh rò rỉ tài nguyên. Kết nối cơ sở dữ liệu (`con`) không được đóng trong hàm này, vì nó được nhận dưới dạng tham số và người gọi hàm có trách nhiệm quản lý việc đóng kết nối chính.

Dựa vào tổng số hàng trong bảng chính để tính chỉ mục phân mảnh đích và chèn thêm vào bảng phân mảnh tương ứng.

3.3.5 Hàm `Range_Insert`

Hàm `rangeinsert` chèn một bộ dữ liệu mới vào phân mảnh theo khoảng giá trị thích hợp.

a, Mục đích

Duy trì tính nhất quán giữa bảng chính và các phân mảnh theo khoảng khi thêm dữ liệu mới.

b, Luồng hoạt động của mã nguồn

Hàm `rangeinsert` được thiết kế để chèn một bản ghi mới trực tiếp vào bảng phân vùng thích hợp dựa trên giá trị của cột `rating`. Điều này giúp duy trì cấu trúc dữ liệu đã được phân mảnh theo khoảng giá trị, đảm bảo tính nhất quán và hiệu quả truy vấn.

```

1 def rangeinsert(ratingtablename, userid, itemid, rating,
2     openconnection):
3     """
4     Function to insert a new row into the main table and
5     specific partition based on range rating.
6     """
7     con = openconnection
8     cur = con.cursor()
9
10    # Calculate appropriate range partition
11    numberofpartitions = count_partitions(
12        RANGE_TABLE_PREFIX, con)
13
14    delta = 5.0 / numberofpartitions
15    index = int(rating / delta)
16    if rating % delta == 0 and index != 0:
17        index -= 1
18
19    # Insert into partition
20    table_name = f"{RANGE_TABLE_PREFIX}{index}"
21    cur.execute(f"INSERT INTO {table_name} (userid, movieid
22        , rating) VALUES (%s, %s, %s)",
23        (userid, itemid, rating))
24
25    con.commit()
26    cur.close()

```

Giải thích luồng hoạt động: Toàn bộ quá trình chèn dữ liệu vào bảng phân vùng được thực hiện trong một giao dịch duy nhất. Điều này đảm bảo tính toàn vẹn dữ liệu: nếu có bất kỳ lỗi nào xảy ra trong quá trình chèn, giao dịch sẽ được hoàn tác, ngăn chặn tình trạng dữ liệu không nhất quán. Lưu ý rằng hàm này chỉ chèn vào bảng phân vùng mà không chèn vào bảng chính, giả định rằng bảng chính đã được phân mảnh hoàn toàn hoặc việc chèn vào bảng chính được xử lý riêng.

1. Bước 1: Khởi tạo kết nối.

- `con = openconnection`: Hàm nhận một đối tượng kết nối cơ sở dữ liệu đã được mở sẵn (`openconnection`) và gán nó cho biến cục bộ `con`. Đây là kết nối sẽ được sử dụng để tương tác với cơ sở dữ liệu.
- `cur = con.cursor()`: Từ đối tượng kết nối, một đối tượng con trỏ (`cursor`) được tạo ra. Con trỏ này cho phép thực thi các câu lệnh SQL.

2. Bước 2: Tính toán chỉ mục phân vùng thích hợp.

- `numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, con)`: Hàm `count_partitions` (không được định nghĩa trong đoạn mã này nhưng được giả định là tồn tại) được gọi để xác định tổng số lượng bảng phân vùng hiện có dựa trên tiền tố tên bảng (`RANGE_TABLE_PREFIX`). Số lượng phân vùng này là cần thiết để tính toán khoảng giá trị.
- `delta = 5.0 / numberofpartitions`: Giá trị `delta` được tính toán. Đây là độ rộng của mỗi khoảng xếp hạng, dựa trên giả định thang điểm từ 0.0 đến 5.0.
- `index = int(rating / delta)`: Chỉ mục ban đầu của phân vùng được tính bằng cách chia giá trị `rating` cho `delta` và lấy phần nguyên. Ví dụ, nếu `rating` là 2.5 và `delta` là 1.0, `index` sẽ là 2.
- `if rating % delta == 0 and index != 0: index -= 1`: Đây là một điều kiện quan trọng để xử lý các giá trị `rating` nằm đúng ranh giới của một khoảng.
 - Khi `rating` là bội số của `delta` (tức là `rating % delta == 0`), điều đó có nghĩa là `rating` nằm chính xác tại điểm kết thúc của một khoảng và cũng là điểm bắt đầu của khoảng tiếp theo.
 - Điều kiện `index != 0` được thêm vào để không áp dụng logic này cho trường hợp ‘`rating`’ là 0, vì 0 là điểm bắt đầu của khoảng đầu tiên và không cần lùi lại.
 - Logic này đảm bảo rằng các giá trị xếp hạng nằm ở ranh giới trên của một khoảng (ví dụ: xếp hạng 1.0, 2.0, ...) sẽ được xếp vào phân vùng **trước đó** (ví dụ, 1.0 sẽ nằm trong khoảng [0.0, 1.0] thay vì khoảng (1.0, 2.0]). Điều này khớp với cách phân vùng đã được tạo ra trong hàm ‘`rangepartition`’ (phân vùng đầu tiên sử dụng ‘`>= minRange AND <= maxRange`’, các phân vùng sau sử dụng ‘`> minRange AND <= maxRange`’).

3. Bước 3: Chèn dữ liệu vào bảng phân vùng đích.

- `table_name = f"{RANGE_TABLE_PREFIX}{index}"`: Tên của bảng phân vùng đích được xây dựng dựa trên tiền tố và chỉ mục đã tính toán ở Bước 2.
- `cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) VALUES (%s, %s, %s)", (userid, itemid, rating))`: Bản ghi mới (bao gồm `userid`, `itemid` được ánh xạ thành `movieid`, và `rating`) được chèn vào bảng phân vùng cụ thể đã xác định.

4. Bước 4: Xác nhận giao dịch và giải phóng tài nguyên.

- `con.commit()`: Lệnh này xác nhận và lưu vĩnh viễn tất cả các thay đổi đã thực hiện trong giao dịch (chèn dữ liệu vào bảng phân vùng) vào cơ sở dữ liệu.
- `cur.close()`: Con trỏ (`cur`) được đóng để giải phóng các tài nguyên đã được cấp cho nó. Đây là một hành động dọn dẹp cần thiết để tránh rò rỉ tài nguyên. Kết nối cơ sở dữ liệu (`con`) không được đóng trong hàm này, vì nó được nhận dưới dạng tham số và người gọi hàm có trách nhiệm quản lý việc đóng kết nối chính.

Hàm tính toán phân mảnh dựa trên giá trị `rating`, với điều chỉnh để xử lý các giá trị nằm đúng ranh giới khoảng.

3.3.6 Các Hàm Hỗ Trợ

a, Hàm `GetOpenConnection`

```
1 def getopenconnection(user=DB_USER, password=DB_PASSWORD,
2     dbname=DB_NAME) :
3     connection = psycopg2.connect("dbname='" + dbname + "' user="
4         + user + "' host='" + DB_HOST + "' password='" +
5         password + "'")
6     return connection
```

Hàm thiết lập kết nối đến cơ sở dữ liệu PostgreSQL sử dụng thông tin xác thực cho trước, trả về đối tượng kết nối để sử dụng trong các hàm khác.

b, Hàm `Create_DB`

```
1 def create_db(dbname) :
2     """
3     We create a DB by connecting to the default user and
4     database of Postgres
```

```

4 The function first checks if an existing database exists
   for a given name, else creates it.
5 :return:None
6 """
7 con = getopenconnection(dbname='postgres')
8 con.set_isolation_level(psycopg2.extensions.
   ISOLATION_LEVEL_AUTOCOMMIT)
9 cur = con.cursor()
10
11 cur.execute('SELECT COUNT(*) FROM pg_catalog.pg_database
   WHERE datname=%s', (dbname,))
12 count = cur.fetchone()[0]
13 if count == 0:
14     cur.execute(f'CREATE DATABASE {dbname}')
15 else:
16     print(f'A database named {dbname} already exists')
17
18 cur.close()
19 con.close()

```

Hàm kiểm tra sự tồn tại của cơ sở dữ liệu và tạo mới nếu chưa có, sử dụng mức isolation autocommit cho lệnh DDL.

c, Hàm Count_Partitions

```

1 def count_partitions(prefix, openconnection):
2     """
3     Function to count the number of tables which have the
4     @prefix in their name somewhere.
5     """
6     cur = openconnection.cursor()
7     cur.execute("SELECT COUNT(*) FROM pg_stat_user_tables WHERE
   relname LIKE %s", (prefix + '%',))
8     count = cur.fetchone()[0]
9     cur.close()
10    return count

```

Hàm đếm số bảng trong cơ sở dữ liệu có tên bắt đầu bằng tiền tố cho trước, hỗ trợ xác định số phân mảnh hiện có.

PHẦN 4. KIỂM THỬ VÀ ĐÁNH GIÁ.

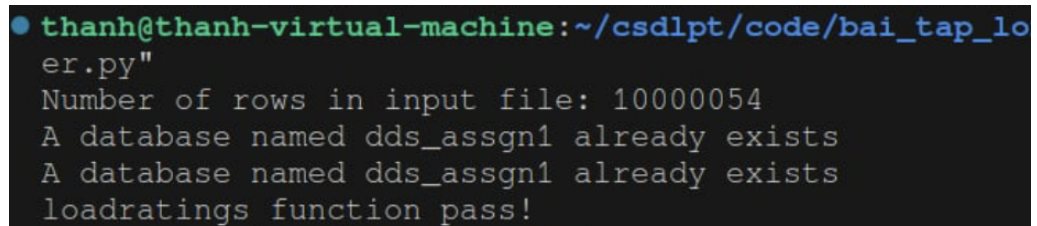
4.1 LoadRatings.

Khi chạy file Assignment1Tester.py, với test:

```
1 [result, e] = testHelper.testloadratings(LoadRating,  
    RATINGS_TABLE, INPUT_FILE_PATH, conn,  
    ACTUAL_ROWS_IN_INPUT_FILE)  
2 if result :  
3     print("loadratings function pass!")  
4 else:  
5     print("loadratings function fail!")
```

Kết quả thu được:

- Console hiển thị kết quả pass như hình 4.1.



```
● thanh@thanh-virtual-machine:~/csdlpt/code/bai_tap_lo  
er.py"  
Number of rows in input file: 10000054  
A database named dds_assgn1 already exists  
A database named dds_assgn1 already exists  
loadratings function pass!
```

Hình 4.1: console hiển thị kết quả loadRating.

- Kiểm tra kết quả trong pgadmin4 thì đã tạo được bảng với số dòng bằng với số dòng trong file ratings.dat là 10000054 dòng như hình 4.2.

Data Output Messages Notifications			
<div> <div>☰</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>			
	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5
13	1	420	5
14	1	466	5
15	1	480	5
16	1	520	5
Total rows: 10000054		Query complete 00:00:05.870	

Hình 4.2: pgadmin4 hiển thị kết quả bảng ratings.

Kết luận hàm LoadRatings được cài đặt đúng để load dữ liệu từ ratings.dat vào bảng ratings trong postgresql.

4.2 RangePartition.

Khi chạy file Assignment1Tester.py, với test:

```

1 [result, e] = testHelper.testrangepartition(range,
      RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
2 if result :
3     print("rangepartition function pass!")
4 else:
5     print("rangepartition function fail!")

```

Kết quả thu được:

- Console hiển thị kết quả `pass` cho quá trình tải dữ liệu như minh họa trong Hình 4.3.

```
Number of rows in input file: 10000054
A database named dds_assgn1 already exists
A database named dds_assgn1 already exists
loadratings function pass!
rangepartition function pass!
```

Hình 4.3: Kết quả hiển thị trên console sau khi thực hiện hàm `RangePartition()`.

- Kiểm tra kết quả trong pgAdmin 4 cho thấy hệ thống đã tạo thành công 5 bảng phân mảnh. Tổng số dòng của cả 5 bảng đúng bằng tổng số dòng của tệp dữ liệu đầu vào (10.000.054 dòng). Thống kê số lượng dòng trên từng bảng như sau:

- **range_part0:** 479,168 dòng
- **range_part1:** 908,584 dòng
- **range_part2:** 2,726,854 dòng
- **range_part3:** 3,755,614 dòng
- **range_part4:** 2,129,834 dòng



Hình 4.4: Kết quả trong pgAdmin4 hiển thị số lượng bản ghi trong các bảng phân mảnh sau khi thực hiện hàm `rangePartition()`.

Kết luận hàm `RangePartition()` được cài đặt đúng.

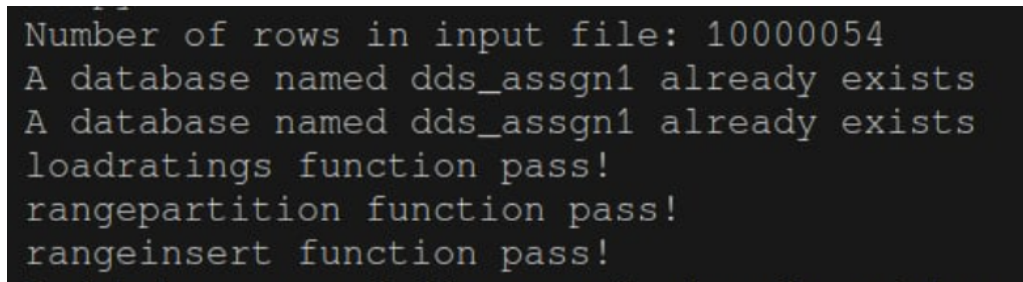
4.3 RangeInsert.

Khi chạy file `Assignment1Tester.py`, với test:

```
1 [result, e] = testHelper.testrangeinsert(range,  
    RATINGS_TABLE, 100, 2, 3, conn, '2')
```

Kết quả thu được:

- Console hiển thị kết quả pass như hình 4.5.



```
Number of rows in input file: 10000054  
A database named dds_assgn1 already exists  
A database named dds_assgn1 already exists  
loadratings function pass!  
rangepartition function pass!  
rangeinsert function pass!
```

Hình 4.5: console hiển thị kết quả test hàm **RangeInsert()**.

- Kiểm tra kết quả trong pgadmin4 trong bảng range_part2 có được thêm 1 dòng với dữ liệu như đã được insert, bảng range_part2 có số dòng là 2,726,855.

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>			
	userid integer	movieid integer	rating double precision
985	34	2195	3
986	34	2231	3
987	34	2278	3
988	34	2279	3
989	34	2294	3
990	34	2302	3
991	34	2322	3
992	34	2335	3
993	34	2355	3
994	34	2391	3
995	34	2424	3
996	34	2432	3
997	34	2443	3
998	34	2467	3
999	34	2502	3
1000	34	2548	3
Total rows: 2726855		Query complete 00:00:01.593	

Hình 4.6: pgadmin4 hiển thị kết quả bảng range_part2.

Khi chạy file Assignment1Tester.py, với test:

```
[result, e] = testHelper.testrangeinsert(range,
    RATINGS_TABLE, 100, 2, 0, conn, '0')
```

Kết quả thu được:

- Console hiển thị kết quả pass như hình 4.7.

```

Number of rows in input file: 10000054
A database named dds_assgn1 already exists
A database named dds_assgn1 already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!

```

Hình 4.7: console hiển thị kết quả test hàm **RangeInsert()**.

- Kiểm tra kết quả trong pgadmin4 trong bảng range_part0 có được thêm 1 dòng với dữ liệu như đã được insert, bảng range_part0 có số dòng là 479,169.

	userid integer	movieid integer	rating double precision
1	4	231	1
2	5	1	1
3	5	708	1
4	5	736	1
5	5	780	1
6	5	1391	1
7	6	3986	1
8	6	4270	1
9	7	1917	1
10	7	2478	1
11	7	5094	1
12	8	590	0.5
13	8	1035	0.5
14	8	1721	1
15	8	2378	0.5
16	8	2379	1
Total rows: 479169		Query complete 00:00:00.548	

Hình 4.8: pgadmin4 hiển thị kết quả bảng range_part0.

Vậy kết luận hàm RangeInsert() được cài đặt đúng.

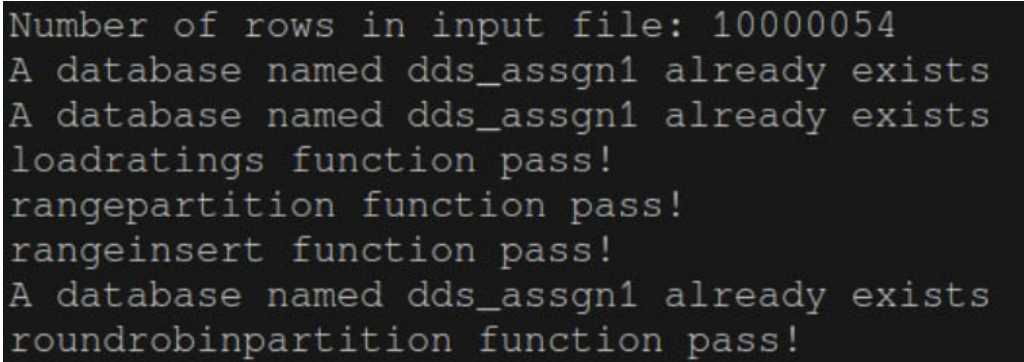
4.4 RoundRobinPartition.

Khi chạy file Assignment1Tester.py, với test:


```
1 [result, e] = testHelper.testroundrobinpartition(  
    roundRobin, RATINGS_TABLE, 5, conn, 0,  
    ACTUAL_ROWS_IN_INPUT_FILE)  
2 if result :  
3     print("roundrobinpartition function pass!")  
4 else:  
5     print("roundrobinpartition function fail")
```

Kết quả thu được:

- Console hiển thị kết quả pass cho quá trình tải dữ liệu như minh họa trong Hình 4.9.



```
Number of rows in input file: 10000054  
A database named dds_assgn1 already exists  
A database named dds_assgn1 already exists  
loadratings function pass!  
rangepartition function pass!  
rangeinsert function pass!  
A database named dds_assgn1 already exists  
roundrobinpartition function pass!
```

Hình 4.9: Kết quả hiển thị trên console sau khi thực hiện hàm RoundRobinPartition().

- Kiểm tra kết quả trong pgAdmin 4 cho thấy hệ thống đã tạo thành công 5 bảng phân mảnh. Tổng số dòng của cả 5 bảng đúng bằng tổng số dòng của tệp dữ liệu đầu vào (10.000.054 dòng). Thống kê số lượng dòng trên từng bảng như sau:
 - **rrobin_part0:** 2,000,011 dòng
 - **rrobin_part1:** 2,000,011 dòng
 - **rrobin_part2:** 2,000,011 dòng
 - **rrobin_part3:** 2,000,011 dòng
 - **rrobin_part4:** 2,000,010 dòng



Hình 4.10: Kết quả trong pgAdmin4 hiển thị số lượng bản ghi trong các bảng phân mảnh sau khi thực hiện hàm `RoundRobinPartition()`.

Kết luận hàm `RoundRobinPartition()` được cài đặt đúng.

4.5 RoundRobinInsert.

Khi chạy file `Assignment1Tester.py`, với test:

```
[result, e] = testHelper.testroundrobininsert(
    roundRobin, RATINGS_TABLE, 100, 1, 3, conn, '0')
```

Kết quả thu được:

- Console hiển thị kết quả fail.

```
Number of rows in input file: 10000054
A database named dds_assgn1 already exists
A database named dds_assgn1 already exists
loadratings function pass!
roundrobinpartition function pass!
Traceback (most recent call last):
  File "/home/thanh/csd1pt/code/bai_tap_lon_CSDL_phan_tan/testHelper.py", line 244, in testroundrobininsert
    raise Exception(
Exception: Round robin insert failed! Couldnt find (100, 1, 3) tuple in rrobin_part0 table
roundrobininsert function fail!
```

Hình 4.11: console hiển thị kết quả test hàm `RoundRobinInsert()`.

- Kiểm tra kết quả trong pgadmin4 trong bảng `rrobin_part0` không được thêm 1 dòng với dữ liệu như đã được insert, bảng `rrobin_part0` vẫn có số dòng là 2,000,011. Do dữ liệu được insert vào bảng `rrobin_part4`.

Data Output Messages Notifications			
	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	329	5
3	1	370	5
4	1	520	5
5	1	594	5
6	2	376	3
7	2	733	3
8	2	858	2
9	2	1391	3
10	3	590	3.5
11	3	1288	3
12	3	1674	4.5
13	3	4995	4.5
14	3	6287	3
15	3	8529	4
16	4	21	3
Total rows: 2000011		Query complete 00:00:01.223	

Hình 4.12: pgadmin4 hiển thị kết quả bảng rrobin_part0 với số dòng vẫn như cũ.

Khi chạy file Assignment1Tester.py, với test:

```
[result, e] = testHelper.testroundrobininsert(
    roundRobin, RATINGS_TABLE, 100, 1, 3, conn, '4')
```

Kết quả thu được:

- Console hiển thị kết quả pass.

```
A database named dds_assgn1 already exists
roundrobinpartition function pass!
roundrobininsert function pass!
```

Hình 4.13: console hiển thị kết quả test hàm **RoundRobinInsert()**.

- Kiểm tra kết quả trong pgadmin4 trong bảng rrobin_part4 có được thêm 1 dòng với dữ liệu như đã được insert, bảng rrobin_part4 có số dòng là 2,000,011.

	userid integer	movieid integer	rating double precision
1	1	316	5
2	1	364	5
3	1	480	5
4	1	589	5
5	2	260	5
6	2	719	3
7	2	802	2
8	2	1356	3
9	3	213	5
10	3	1276	3.5
11	3	1597	4.5
12	3	4677	4
13	3	5952	3.5
14	3	7155	3.5
15	3	33750	3.5
16	4	150	5
Total rows: 2000011		Query complete 00:00:01.075	

Hình 4.14: pgadmin4 hiển thị kết quả bảng rrobin_part4 với 2000011 dòng.

Vậy Kết luận hàm RoundRobinInsert() được cài đặt đúng

PHẦN 5. KẾT LUẬN.

Trong khuôn khổ bài tập lớn này, nhóm đã nghiên cứu, triển khai và mô phỏng thành công hai phương pháp phân mảnh dữ liệu phổ biến trong hệ quản trị cơ sở dữ liệu quan hệ, bao gồm: **phân mảnh ngang theo khoảng giá trị (Range Partition)** và **phân mảnh ngang vòng tròn (Round Robin Partition)**, trên nền tảng hệ quản trị cơ sở dữ liệu mã nguồn mở **PostgreSQL**.

Báo cáo đã trình bày một cách có hệ thống từ phần cơ sở lý thuyết về phân mảnh dữ liệu — bao gồm các khái niệm, phân loại và vai trò của phân mảnh trong hệ thống cơ sở dữ liệu phân tán — đến quá trình triển khai thực tế. Cụ thể, nhóm đã tiến hành cài đặt môi trường, chuẩn hóa dữ liệu đầu vào, xây dựng và kiểm thử các hàm xử lý bằng ngôn ngữ **Python** để thực hiện việc phân mảnh và chèn dữ liệu tương ứng vào các bảng con được sinh ra từ mỗi phương pháp phân mảnh.

Kết quả thử nghiệm cho thấy:

- **Đối với phương pháp Range Partition:** Dữ liệu được phân chia chính xác vào các bảng con (`range_partX`) dựa trên giá trị của thuộc tính `rating`. Cách phân mảnh này cho thấy hiệu quả rõ rệt trong các truy vấn có điều kiện theo phạm vi, giúp tăng tốc độ truy vấn và giảm thiểu lượng dữ liệu cần quét. Việc chèn dữ liệu mới cũng được thực hiện đúng vào phân mảnh tương ứng, đảm bảo tính nhất quán và toàn vẹn dữ liệu trong hệ thống phân tán.
- **Đối với phương pháp Round Robin Partition:** Dữ liệu được phân phối đồng đều theo vòng lặp giữa các bảng con (`rrobin_partX`). Phương pháp này mang lại sự cân bằng tải tốt hơn trong các tình huống truy cập hoặc xử lý song song, đặc biệt khi không có sự tập trung truy vấn vào một phân mảnh cụ thể. Kết quả thử nghiệm cũng cho thấy cơ chế chèn dữ liệu mới tiếp tục đảm bảo sự phân phối đồng đều giữa các phân mảnh, từ đó hỗ trợ hiệu quả cho các hệ thống cần khả năng mở rộng.

Thông qua quá trình thực hiện, nhóm đã củng cố kiến thức lý thuyết về phân mảnh dữ liệu, hiểu rõ hơn về cơ chế hoạt động và khả năng hỗ trợ phân mảnh của PostgreSQL, đồng thời tích lũy được nhiều kinh nghiệm thực tế trong việc xây dựng các hàm xử lý dữ liệu bằng Python và tương tác với hệ quản trị cơ sở dữ liệu.

Mặc dù chúng em đã cố gắng để làm tốt nhất có thể, dự án vẫn còn tồn tại một số hạn chế cần được cải thiện. Tuy nhiên, những thách thức này là cơ hội để em tiếp tục học hỏi, cải tiến và phát triển dự án trong tương lai.

TÀI LIỆU THAM KHẢO

- [1] PostgreSQL Documentation. (2021). Truy cập tại: <https://www.postgresql.org/docs/current/ddl-partitioning.html>
- [2] DeWitt, D., & Gray, J. (1992). "Parallel database systems: The future of high performance database systems." *Communications of the ACM*, 35(6), 85-98.
- [3] Stonebraker, M. (1986). "The case for shared nothing." *IEEE Database Engineering Bulletin*, 9(1), 4-9.
- [4] Pavlo, A., et al. (2009). "A comparison of approaches to large-scale data analysis." *Proceedings of the 2009 ACM SIGMOD*.
- [5] Wikipedia contributors. (2025). Data fragmentation. Wikipedia, The Free Encyclopedia. Truy cập tại: https://en.wikipedia.org/wiki/Data_fragmentation
- [6] Moodle Docs. (2025). Moodle Developer Documentation. Truy cập tại: <https://moodledev.io>
- [7] stackoverflow. Truy cập tại: <https://stackoverflow.com/>
- [8] Github. Truy cập tại: <https://github.com/>