HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên hướng dẫn	**: Kim Ngọc Bách**
Nhóm	**: 01-N9**
Họ và tên	***: Mã Sinh Viên***
Phan Văn Hoàn	**: B22DCCN329**
Hoàng Minh Nghĩa	**: B22DCCN605**
Phùng Đình Dũng	**: B22DCCN137**

Hà Nội – 2025

Mục lục

Lời cảm ơn 4
I. Phân chia công việc 5
II. Tóm tắt đề bài 5
III. Trình bài và giải thích cách giải quyết vấn đề 6
1. Tổng quan cách tiếp cận bài toán 6
2. Triển khai chi tiết. 7
2.1. Môi trường triển khai 7
2.2. Giải pháp cho từng nhiệm vụ 7
IV. Kết quả thực nghiệm 15

Bảng hình ảnh:

Hình 1: Ảnh kết quả cho hàm loadratings 8
Hình 2: Ảnh quá trình phân mảnh cho hàm rangepartiton 9
Hình 3: Ảnh kết quả sau khi phân mảnh của range_part0 9
Hình 4: Ảnh kết quả sau khi phân mảnh của range_part1 9
Hình 5: Ảnh kết quả sau khi phân mảnh của range_part2 10
Hình 6: Ảnh kết quả sau khi phân mảnh của range_part3 10
Hình 7: Ảnh kết quả sau khi phân mảnh của range_part4 11
Hình 8: Ảnh quá trình phân mảnh cho hàm roundrobinpartition với n = 3 12
Hình 9: Ảnh quá trình cho hàm rangeinsert 13
Hình 10: Ảnh kết quả của hàm roundrobininsert 14
Hình 11: Ảnh tạo bảng của hàm create_metadata_table 14
Hình 12: Ảnh kết quả cho hàm count_partitions 15
Hình 13: Nhập đầu vào là file rating.dat với 10 triệu lẻ 54 bản ghi 15
Hình 14: Ảnh kết quả trả ra khi chạy file Assignment1Tester.py 15
Hình 15: Bảng ratings được tạo sau khi chạy hàm loadratings 16
Hình 16: Bảng range_part0 được tạo sau khi chạy hàm rangepartition 17
Hình 17: Bảng range_part1 được tạo sau khi chạy hàm rangepartition 17
Hình 18: Bảng range_part2 được tạo sau khi chạy hàm rangepartition 18
Hình 19: Bảng range_part3 được tạo sau khi chạy hàm rangepartition 18
Hình 20: Bảng range_part4 được tạo sau khi chạy hàm rangepartition 19
Hình 21: Bảng rrobin_part0 được tạo sau khi chạy hàm roundrobinpartition 19
Hình 22: Bảng rrobin_part1 được tạo sau khi chạy hàm roundrobinpartition 20
Hình 23: Bảng rrobin_part2 được tạo sau khi chạy hàm roundrobinpartition 20
Hình 24: Bảng rrobin_part3 được tạo sau khi chạy hàm roundrobinpartition 21
Hình 25: Bảng rrobin_part4 được tạo sau khi chạy hàm roundrobinpartition 21
Hình 26: Dữ liệu được chèn vào bảng range_part0 sau khi chạy hàm rangeinsert 22
Hình 27: Dữ liệu được chèn vào bảng ratings sau khi chạy hàm rangeinsert 22
Hình 28: Dữ liệu được chèn vào bảng rrobin_part4 sau khi chạy hàm roundrobininsert 23
Hình 29: Dữ liệu được chèn vào bảng ratings sau khi chạy hàm roundrobininsert 23

Lời cảm ơn

Chúng em xin trân trọng gửi lời cảm ơn chân thành nhất đến thầy Kim Ngọc Bách.

Trong suốt quá trình học tập và hoàn thiện đề tài này, sự hướng dẫn tận tình, những kiến thức chuyên sâu và những góp ý quý báu của thầy đã giúp chúng em rất nhiều. Thầy không chỉ truyền đạt kiến thức mà còn khơi gợi niềm say mê học hỏi, giúp chúng em hiểu rõ hơn về các khái niệm phức tạp và áp dụng vào thực tế một cách hiệu quả.

Sự hỗ trợ và định hướng của thầy là yếu tố quan trọng giúp nhóm chúng em có thể hoàn thành bài tập lớn này.

Một lần nữa, chúng em xin chân thành cảm ơn thầy!

I. Phân chia công việc

Thành Viên	Công việc

Phan Văn Hoàn (Trưởng nhóm)	• Lập kế hoạch, phân chia công việc.
	• Trình bày, duyệt báo cáo.
	• Đề xuất cách tiếp cần và cách thức triển khai chung để giải quyết vấn đề.
	• Đề xuất giải pháp và triển khai chi tiết các hàm:
	  ○ create_metadata_table
	  ○ Roundrobinpartition ().
	  ○ RoundRoundRobin_Insert
	• Tối ưu cú pháp

Hoàng Minh Nghĩa	• Chỉnh sửa, bổ sung báo cáo
	• Đề xuất giải pháp triển khai chi tiết hàm:
	  ○ LoadRatings ()
	  ○ Range_Insert ()
	  ○ count_partitions ()

Phùng Đình Dũng	• Chỉnh sửa, bổ sung báo cáo
	• Đề xuất giải pháp và triển khai chi tiết hàm:
	  ○ Range_Partition (),
	  ○ Hàm tạo cơ sở dữ liệu (create_db)
	• Kiểm tra rà soát lỗi và đánh giá tối ưu.

II. Tóm tắt đề bài

Bài tập lớn yêu cầu mô phỏng các phương pháp phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở (như PostgreSQL hoặc MySQL), sử dụng ngôn ngữ Python để xử lý.

Cụ thể, cần:

- Viết các hàm Python để:

    - Tải dữ liệu đánh giá phim từ tệp ratings.dat vào bảng Ratings.
    - Thực hiện phân mảnh ngang theo Range Partitioning và Round Robin Partitioning.
    - Cài đặt các hàm chèn bản ghi mới vào đúng phân mảnh tương ứng.
- Dữ liệu sử dụng là từ trang MovieLens, chứa hơn 10 triệu lượt đánh giá của người dùng với định dạng UserID::MovieID::Rating::Timestamp.

Ngoài ra, cần tuân thủ các ràng buộc:

- Không thay đổi tên bảng phân mảnh mặc định.
- Không mã hóa cứng tên file, tên cơ sở dữ liệu hoặc đóng kết nối DB trong các hàm.
- Tuân thủ đúng lược đồ bảng và thuật toán phân mảnh được mô tả.

III. Trình bài và giải thích cách giải quyết vấn đề

1. Tổng quan cách tiếp cận bài toán

Để giải quyết bài toán phân mảnh dữ liệu ngang trên hệ quản trị cơ sở dữ liệu PostgreSQL, nhóm đã tiến hành phân tích kỹ lưỡng yêu cầu đề bài và xây dựng các hàm Python sử dụng thư viện psycopg2 để tương tác hiệu quả với cơ sở dữ liệu. Quy trình thực hiện bao gồm năm bước chính:

1. *Thiết lập môi trường: Cài đặt Python 3.12.x, PostgreSQL và thư viện psycopg2. Tạo cơ sở dữ liệu dds_assgn1 và bảng meta-data partition_metadata để lưu trữ thông tin về số bản ghi và số lượng phân mảnh.*
2. *Tải dữ liệu: Sử dụng hàm loadratings để nạp dữ liệu từ tệp ratings.dat vào bảng Ratings một cách nhanh chóng, đảm bảo schema đúng với yêu cầu (userid, movieid, rating).*
3. *Phân mảnh dữ liệu: Triển khai hai phương pháp phân mảnh ngang:*
4. *Phân mảnh theo khoảng (range partitioning): Chia bảng Ratings thành N bảng con (range_part0, range_part1, ...) dựa trên các khoảng giá trị đều nhau của trường rating.*
5. *Phân mảnh vòng tròn (round-robin partitioning): Phân bổ bản ghi lần lượt vào N bảng con (rrobin_part0, rrobin_part1, ...) theo thứ tự vòng tròn.*
6. *Chèn dữ liệu mới: Xây dựng các hàm rangeinsert và roundrobininsert để chèn bản ghi mới vào bảng Ratings và đúng bảng phân mảnh, sử dụng bảng partition_metadata để theo dõi số bản ghi và trạng thái phân mảnh.*
7. *Kiểm tra kết quả: Kiểm tra thủ công nội dung các bảng trong PostgreSQL để đảm bảo dữ liệu được phân mảnh và chèn chính xác.*

Để tuân thủ yêu cầu không sử dụng biến toàn cục và tăng hiệu suất, nhóm đã thiết kế bảng partition_metadata để lưu trữ thông tin về số lượng phân mảnh (partition_count), tổng số bản ghi (total_rows), và chỉ số phân mảnh cuối cùng (last_insert_idx cho round-robin). Bảng này được cập nhật trong các hàm rangepartition, roundrobinpartition, rangeinsert, và roundrobininsert và đảm bảo tính nhất quán khi chèn dữ liệu. Phương pháp này không chỉ đáp ứng yêu cầu đề bài mà còn tăng cường khả năng mở rộng cho các tập dữ liệu lớn, như tập MovieLens với 10 triệu bản ghi.

2. Triển khai chi tiết.

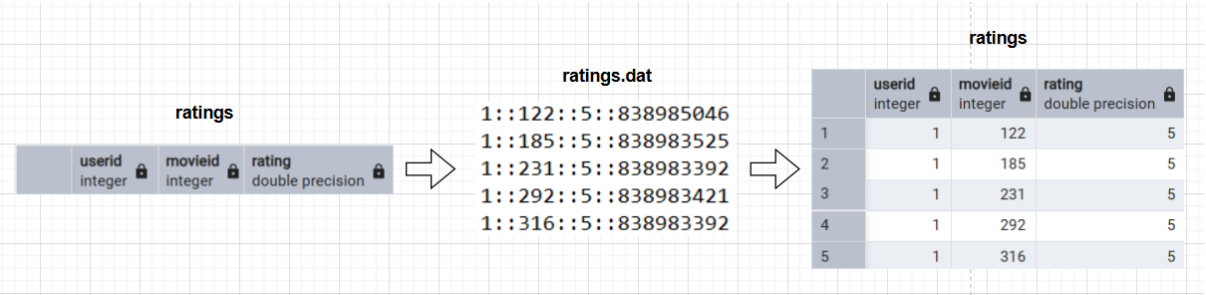
2.1. Môi trường triển khai

- Cấu hình máy ảo: Python 3.12.x
- Hệ điều hành: Window 11
- Hệ quản trị cơ sở dữ liệu: PostgreSQL

2.2. Giải pháp cho từng nhiệm vụ

a. Hàm loadratings

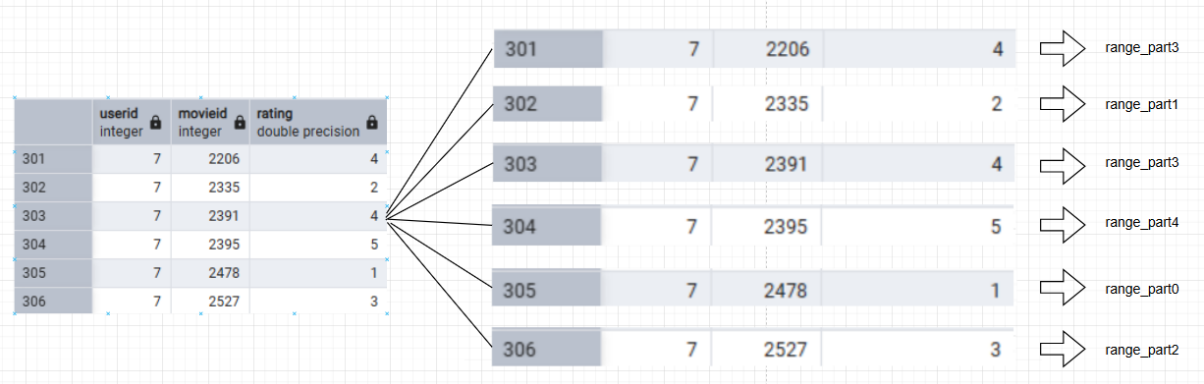
1. Nhận tham số kết nối, tên bảng và đường dẫn file
2. Tạo cursor từ kết nối
3. Xóa bảng ratings nếu đã tồn tại và tạo lại với cấu trúc gồm userid, movieid và rating
4. Khởi tạo buffer trong bộ nhớ để xử lý dữ liệu
5. Mở và đọc file dữ liệu theo từng dòng:
    - Tách trường dữ liệu bằng dấu ':'
    - Trích xuất 3 trường cần thiết (userid, movieid, rating)
    - Ghi vào buffer với dấu tab làm phân cách
6. Đưa con trỏ về đầu buffer
7. Sử dụng copy_from để nạp dữ liệu trực tiếp từ buffer vào bảng
8. Đóng buffer để giải phóng bộ nhớ
9. Đóng cursor (vẫn duy trì kết nối)
10. Commit thay đổi



Hình 1: Ảnh kết quả cho hàm loadratings

b. Hàm phân mảnh theo khoảng rangepartition

1. Nhận tham số kết nối, tên bảng và số phân mảnh
2. Tạo cursor từ kết nối
3. Gọi hàm tạo bảng metadata nếu chưa tồn tại
4. Tính toán khoảng delta = 5.0 / số phân mảnh
5. Với mỗi phân mảnh i:
    - Tính giới hạn dưới minRange = i * delta
    - Tính giới hạn trên maxRange = (i+1) * delta hoặc 5.0 nếu là phân mảnh cuối
    - Tạo tên bảng phân mảnh (range_part + i)
    - Tạo bảng phân mảnh mới
    - Nếu là phân mảnh đầu tiên (i=0):
        - Chèn dữ liệu từ bảng gốc với điều kiện rating >= minRange và <= maxRange
    - Nếu không:
        - Chèn dữ liệu với điều kiện rating > minRange và <= maxRange
6. Đếm tổng số bản ghi trong bảng gốc
7. Thêm thông tin vào bảng metadata (loại phân mảnh, số phân mảnh, tổng số bản ghi)
8. Commit thay đổi
9. Đóng cursor (không đóng kết nối)



Hình 2: Ảnh quá trình phân mảnh cho hàm rangepartiton

	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

Hình 3: Ảnh kết quả sau khi phân mảnh của range_part0

	userid integer	movieid integer	rating double precision
1	2	648	2
2	2	802	2
3	2	858	2
4	3	1552	2
5	3	5505	2
6	4	344	2
7	6	4053	2
8	6	4369	2
9	7	541	2
10	7	1895	1.5

Hình 4: Ảnh kết quả sau khi phân mảnh của range_part1

	userid integer	movieid integer	rating double precision
1	2	151	3
2	2	376	3
3	2	539	3
4	2	719	3
5	2	733	3
6	2	736	3
7	2	780	3
8	2	786	3
9	2	1049	3
10	2	1073	3

Hình 5: Ảnh kết quả sau khi phân mảnh của range_part2

	userid integer	movieid integer	rating double precision
1	2	1210	4
2	3	590	3.5
3	3	1148	4
4	3	1246	4
5	3	1252	4
6	3	1276	3.5
7	3	1408	3.5
8	3	3408	4
9	3	4535	4
10	3	4677	4

Hình 6: Ảnh kết quả sau khi phân mảnh của range_part3

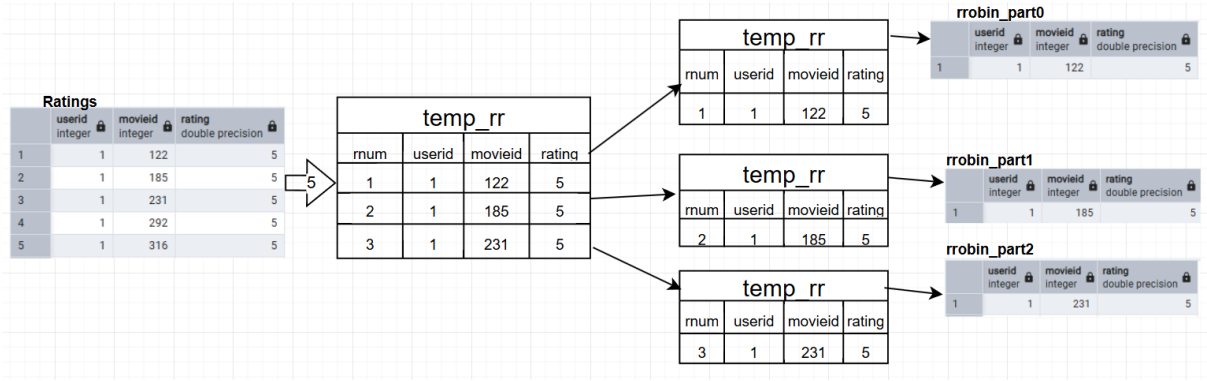
	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5

Hình 7: Ảnh kết quả sau khi phân mảnh của range_part4

c. Hàm phân mảnh theo roundrobinpartition

1. Nhận tham số kết nối, tên bảng và số phân mảnh
2. Tạo cursor từ kết nối
3. Gọi hàm tạo bảng metadata nếu chưa tồn tại
4. Với mỗi phân mảnh i:
    - Tạo tên bảng phân mảnh (rrobin_part + i)
    - Tạo bảng phân mảnh mới
5. Tạo bảng tạm lưu dữ liệu từ bảng gốc, thêm cột số thứ tự (ROW_NUMBER)
6. Với mỗi phân mảnh i:
    - Chèn dữ liệu từ bảng tạm với điều kiện MOD(số thứ tự, số phân mảnh) = i
7. Xóa bảng tạm
8. Đếm tổng số bản ghi trong bảng gốc
9. Tính chỉ số phân mảnh cuối cùng: (total_rows - 1) % numberofpartitions
10. Thêm thông tin vào bảng metadata (loại phân mảnh, số phân mảnh, tổng số bản ghi, chỉ số cuối)
11. Commit thay đổi

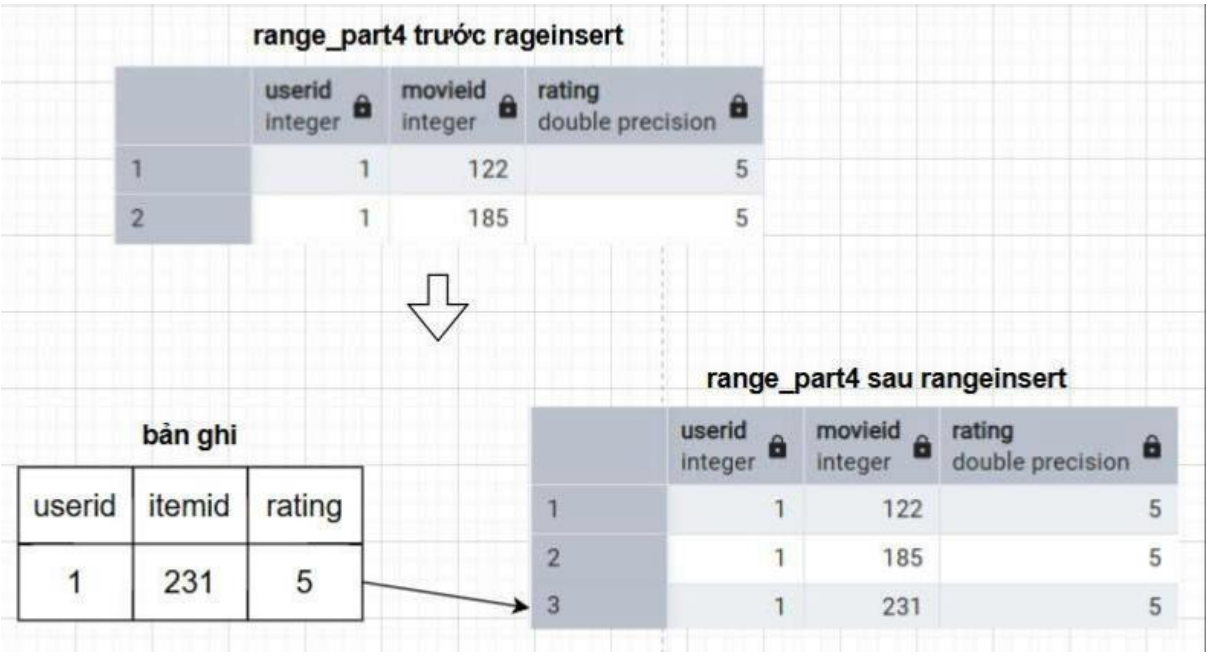
12.Đóng cursor (không đóng kết nối)



Hình 8: Ảnh quá trình phân mảnh cho hàm roundrobinpartition với n = 3

d. Hàm rangeinsert

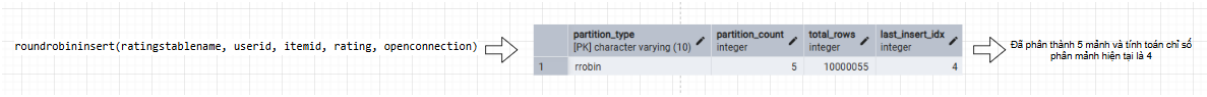
1. Nhận tham số kết nối, tên bảng, userid, itemid, rating
2. Tạo cursor từ kết nối
3. Lấy thông tin từ bảng metadata cho loại phân mảnh 'range'
4. Nếu không có thông tin:
    - Chèn vào bảng gốc
    - Commit thay đổi
    - Đóng cursor (không đóng kết nối)
    - Kết thúc hàm
5. Chèn dữ liệu vào bảng gốc
6. Tính toán chỉ số phân mảnh phù hợp:
    - Tính delta = 5.0 / numberofpartitions
    - Tính index = int(rating / delta)
    - Xử lý trường hợp đặc biệt khi rating là bội số của delta
    - Xử lý trường hợp index vượt quá số phân mảnh
7. Xác định tên bảng phân mảnh dựa trên chỉ số
8. Chèn dữ liệu vào bảng phân mảnh
9. Cập nhật metadata: tăng total_rows
10. Commit thay đổi
11. Đóng cursor (không đóng kết nối)



Hình 9: Ảnh quá trình cho hàm rangeinsert

e. Hàm roundrobininsert

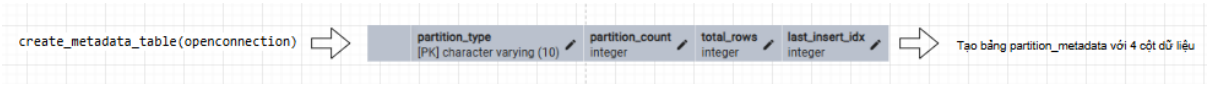
1. Nhận tham số kết nối, tên bảng, userid, itemid, rating
2. Tạo cursor từ kết nối
3. Lấy thông tin từ bảng metadata cho loại phân mảnh 'rrobin'
4. Nếu không có thông tin:
    - Chèn vào bảng gốc
    - Commit thay đổi
    - Đóng cursor (không đóng kết nối)
    - Kết thúc hàm
5. Chèn dữ liệu vào bảng gốc
6. Tính toán chỉ số phân mảnh tiếp theo: (last_insert_idx + 1) % numberofpartitions
7. Xác định tên bảng phân mảnh dựa trên chỉ số
8. Chèn dữ liệu vào bảng phân mảnh
9. Cập nhật metadata: tăng total_rows và cập nhật last_insert_idx
10. Commit thay đổi
11. Đóng cursor (không đóng kết nối)



Hình 10: Ảnh kết quả của hàm roundrobininsert

f. Hàm tạo bảng Metadata (create_metadata_table)

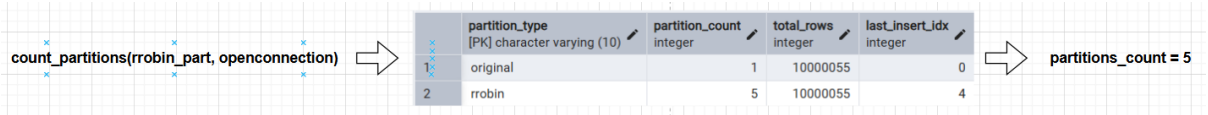
1. Nhận tham số kết nối (openconnection) từ bên ngoài
2. Tạo cursor từ kết nối để thực hiện truy vấn
3. Thực thi câu lệnh SQL để tạo bảng partition_metadata
4. Thiết kế bảng với các cột:
   - partition_type: kiểu phân mảnh (range hoặc rrobin)
   - partition_count: số lượng phân mảnh
   - total_rows: tổng số bản ghi
   - last_insert_idx: chỉ số phân mảnh cuối cùng được sử dụng (cho roundrobin)
5. Commit thay đổi để lưu vào cơ sở dữ liệu
6. Đóng cursor (không đóng kết nối)



Hình 11: Ảnh tạo bảng của hàm create_metadata_table

g. Hàm đếm phân mảnh (count_partitions)

1. Nhận tham số loại phân mảnh và kết nối
2. Tạo cursor từ kết nối
3. Xác định kiểu tìm kiếm dựa trên tham số đầu vào:
   - 'range' -> 'range'
   - 'rrobin_part' -> 'rrobin'
   - Nếu không khớp, đóng cursor và trả về 0
4. Truy vấn số lượng phân mảnh từ bảng metadata
5. Xử lý kết quả truy vấn
6. Đóng cursor (không đóng kết nối)
7. Trả về số lượng phân mảnh



Hình 12: Ảnh kết quả cho hàm count_partitions

h. Hàm tạo cơ sở dữ liệu (create_db)

1. Nhận tham số tên database (dbname) từ bên ngoài
2. Kết nối đến database mặc định của PostgreSQL
3. Thiết lập isolation level để tự động commit
4. Tạo cursor từ kết nối để thực hiện truy vấn
5. Kiểm tra xem database đã tồn tại chưa
6. Tạo database mới nếu chưa tồn tại
7. Đóng cursor và kết nối

IV. Kết quả thực nghiệm

```
RATING_COLNAME = 'rating'
INPUT_FILE_PATH = 'C:\\Users\\hoany\\Desktop\\ml-10M100K\\ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 10000054  # Number of lines in the input file
```

Hình 13: Nhập đầu vào là file rating.dat với 10 triệu lẻ 54 bản ghi

Sau khi thực thi file Assignment1Tester.py, chương trình chạy trong vòng 58 giây và đưa ra kết quả:

```
D:\CSDLPT\BTL-CSDLPT-main>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 14: Ảnh kết quả trả ra khi chạy file Assignment1Tester.py

Phân tích kết quả:

- loadratings function pass!: Hàm loadratings đã tải thành công dữ liệu từ tệp ratings.dat vào bảng ratings. Quá trình này bao gồm tạo bảng, tải dữ liệu hiệu quả bằng io.StringIO và cur.copy_from() và cập nhật siêu dữ

liệu của bảng gốc. Điều này chứng minh khả năng xử lý chính xác lượng lớn dữ liệu đầu vào của hàm.

- rangepartition function pass!: Hàm rangepartition đã hoàn tất việc tạo và phân mảnh dữ liệu vào các bảng con.
- rangeinsert function pass!: Hàm rangeinsert đã chèn thành công một bản ghi mới vào bảng chính và bảng phân mảnh theo khoảng chính xác dựa trên giá trị đánh giá của bản ghi.
- roundrobinpartition function pass Hàm roundrobinpartition tạo và mảnh dữ liệu thành công vào các bảng con bằng cách sử dụng phân vùng round-robin.
- roundrobininsert function pass!: Hàm roundrobininsert chèn thành công một bản ghi mới vào bảng cha và bảng phân vùng round-robin chính xác dựa trên chỉ mục chèn tiếp theo được lưu trữ trong siêu dữ liệu. Điều này xác nhận rằng nó có thể định tuyến dữ liệu mới vào cấu trúc phân vùng round-robin theo thứ tự và cập nhật siêu dữ liệu cần thiết.

Kết quả trên cho thấy tất cả các hàm trong "Interface.py" đã vượt qua các bài kiểm tra.

Các bảng được tạo sau khi chạy chương trình:

1. Sau hàm loadratings:

	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

Hình 15: Bảng ratings được tạo sau khi chạy hàm loadratings

2. Sau hàm rangepartition:

- Với n = 5:

	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

Hình 16: Bảng range_part0 được tạo sau khi chạy hàm rangepartition

	userid integer	movieid integer	rating double precision
1	2	648	2
2	2	802	2
3	2	858	2
4	3	1552	2
5	3	5505	2
6	4	344	2
7	6	4053	2
8	6	4369	2
9	7	541	2
10	7	1895	1.5

Hình 17: Bảng range_part1 được tạo sau khi chạy hàm rangepartition

	userid integer	movieid integer	rating double precision
1	2	151	3
2	2	376	3
3	2	539	3
4	2	719	3
5	2	733	3
6	2	736	3
7	2	780	3
8	2	786	3
9	2	1049	3
10	2	1073	3

Hình 18: Bảng range_part2 được tạo sau khi chạy hàm rangepartition

	userid integer	movieid integer	rating double precision
1	2	1210	4
2	3	590	3.5
3	3	1148	4
4	3	1246	4
5	3	1252	4
6	3	1276	3.5
7	3	1408	3.5
8	3	3408	4
9	3	4535	4
10	3	4677	4

Hình 19: Bảng range_part3 được tạo sau khi chạy hàm rangepartition

	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

Hình 20: Bảng range_part4 được tạo sau khi chạy hàm rangepartition

3. Sau hàm roundrobinpartition:

- Với n = 5:

	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

Hình 21: Bảng rrobin_part0 được tạo sau khi chạy hàm roundrobinpartition

	userid integer	movieid integer	rating double precision
1	1	185	5
2	1	355	5
3	1	377	5
4	1	539	5
5	1	616	5
6	2	539	3
7	2	736	3
8	2	1049	3
9	2	1544	3
10	3	1148	4

Hình 22: Bảng rrobin_part1 được tạo sau khi chạy hàm roundrobinpartition

	userid integer	movieid integer	rating double precision
1	1	231	5
2	1	356	5
3	1	420	5
4	1	586	5
5	2	110	5
6	2	590	5
7	2	780	3
8	2	1073	3
9	3	110	4.5
10	3	1246	4

Hình 23: Bảng rrobin_part2 được tạo sau khi chạy hàm roundrobinpartition

	userid integer	movieid integer	rating double precision
1	1	292	5
2	1	362	5
3	1	466	5
4	1	588	5
5	2	151	3
6	2	648	2
7	2	786	3
8	2	1210	4
9	3	151	4.5
10	3	1252	4

Hình 24: Bảng rrobin_part3 được tạo sau khi chạy hàm roundrobinpartition

	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

Hình 25: Bảng rrobin_part4 được tạo sau khi chạy hàm roundrobinpartition

4. Sau hàm rangeinsert:

- Với testcase n = 5, userid = 100, movieid = 2, rating = 0.

	userid integer	movieid integer	rating double precision
479154	71564	442	1
479155	71564	553	1
479156	71564	653	1
479157	71564	2987	1
479158	71564	5945	1
479159	71565	2167	1
479160	71565	2683	1
479161	71567	891	1
479162	71567	1717	1
479163	71567	1982	1
479164	71567	1983	1
479165	71567	1984	1
479166	71567	1985	1
479167	71567	1986	1
479168	71567	2107	1
479169	100	2	0

Hình 26: Dữ liệu được chèn vào bảng range_part0 sau khi chạy hàm rangeinsert

	userid integer	movieid integer	rating double precision
10000040	71567	1909	2
10000041	71567	1917	4
10000042	71567	1920	4
10000043	71567	1982	1
10000044	71567	1983	1
10000045	71567	1984	1
10000046	71567	1985	1
10000047	71567	1986	1
10000048	71567	2012	3
10000049	71567	2028	5
10000050	71567	2107	1
10000051	71567	2126	2
10000052	71567	2294	5
10000053	71567	2338	2
10000054	71567	2384	2
10000055	100	2	0

Hình 27: Dữ liệu được chèn vào bảng ratings sau khi chạy hàm rangeinsert

5. Sau roundrobininsert:
   - Với testcase n = 5, userid = 100, movieid = 1, rating = 3.

	userid integer	movieid integer	rating double precision
2000001	71567	256	3
2000002	71567	589	4
2000003	71567	898	4
2000004	71567	1210	4
2000005	71567	1396	3
2000006	71567	1598	2
2000007	71567	1769	3
2000008	71567	1909	2
2000009	71567	1984	1
2000010	71567	2107	1
2000011	100	1	3

Hình 28: Dữ liệu được chèn vào bảng rrobin_part4 sau khi chạy hàm roundrobininsert

	userid integer	movieid integer	rating double precision
10000042	71567	1920	4
10000043	71567	1982	1
10000044	71567	1983	1
10000045	71567	1984	1
10000046	71567	1985	1
10000047	71567	1986	1
10000048	71567	2012	3
10000049	71567	2028	5
10000050	71567	2107	1
10000051	71567	2126	2
10000052	71567	2294	5
10000053	71567	2338	2
10000054	71567	2384	2
10000055	100	1	3

Hình 29: Dữ liệu được chèn vào bảng ratings sau khi chạy hàm roundrobininsert