HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

___***___

BÁO CÁO BÀI TẬP LỚN CUỐI KỲ

CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên: Kim Ngọc Bách

Nhóm học phần: 10

Nhóm BTL: 3

Sinh viên thực hiện:

Họ và tên	*Mã sinh viên*
Đỗ Thị Kim Oanh	B22DCCN606
Nguyễn Sỹ Công	B22DCCN089
Nguyễn Thị Hương Giang	B22DCCN249

Hà Nội, 2025

LỜI CẢM ƠN

Lời đầu tiên, nhóm 3 chúng em muốn gửi lời cảm ơn chân thành và sâu sắc tới thầy Kim Ngọc Bách, người đã chỉ dạy và hướng dẫn nhóm 3 nói riêng, cũng như nhóm học phần 10 nói chung trong suốt quá trình thầy giảng dạy học phần Cơ sở dữ liệu phân tán.

Nhờ sự chỉ bảo tận tâm, phương pháp giảng dạy dễ hiểu, cùng với những kiến thức thực tiễn mà thầy truyền đạt, nhóm đã có cơ hội tiếp cận và hiểu rõ hơn về các khái niệm cốt lõi, cũng như kỹ thuật triển khai trong hệ thống cơ sở dữ liệu phân tán – một lĩnh vực quan trọng và thiết thực trong ngành Công nghệ Thông tin.

Thực hiện bài tập lớn này, nhóm chúng em đã được thầy tạo điều kiện để tìm hiểu sâu hơn, rõ hơn về hai loại phân mảnh, biến những lý thuyết tưởng chừng như khô khan, vào trong một bài toán thực tế. Xoay quanh quá trình thực hiện bài tập lớn này, có không ít những khó khăn mà nhóm chưa thể giải quyết triệt để, nhưng nhờ có sự giải đáp nhiệt tình, tâm huyết của thầy, đã phần nào đó giúp chúng em tiến bộ hơn.

Dẫu vậy thì có thể bài tập lớn vẫn chưa đạt tới độ hoàn thiện mà thầy mong muốn, nhưng đó cũng là đúc kết từ sự cố gắng, nỗ lực hoàn thành của nhóm, nên có thể phần nào mong thầy nhìn nhận bài tập lớn này dưới góc độ tích cực nhất.

Một lần nữa, nhóm 3 xin chân thành cảm ơn thầy, kính chúc thầy sức khỏe và luôn thành công trong sự nghiệp giảng dạy!

Nhóm 3

MỤC LỤC

PHÂN CÔNG CÔNG VIỆC

STT	MSV	Họ tên	Nhiệm vụ
1	B22DCCN089	Nguyễn Sỹ Công	Cài đặt hàm rangeinsert(), rangepartition()
2	B22DCCN249	Nguyễn Thị Hương Giang	Cài đặt hàm loadratings(), viết báo cáo
3	B22DCCN606	Đỗ Thị Kim Oanh	Cài đặt hàm roundrobininsert(), roundrobinpartition()

I. Chuẩn bị cấu hình

- Python 3.12.

- Postgres 17: Thông tin user postgres như sau:

Server: localhost, database: postgres, port: 5432, username: postgres, password: 1234

- IDE để chạy Python 3.12, ở đây nhóm chọn Pycharm.

II. Thực hiện

Trong bài tập lớn này, nhóm implement lại từ các hàm mẫu của thầy, thực hiện sửa lại tên các tham số.

Thầy đặt tên các tham số liền nhau:

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

Nhóm đổi lại về dạng phân cách giữa các từ bởi dấu "_" để tường minh hơn:

```
def loadratings(ratings_table_name, file_path, open_connection):
```

Tương tự với các hàm khác.

Các hàm được đặt trong file Interface.py.

File ratings.dat có dung lượng vượt quá dung lượng có thể đẩy lên github nên nhóm không đẩy.

1. Hàm loadratings

1.1. Phân tích

Theo yêu cầu đề bài, hàm loadratings có nhiệm vụ tải nội dung file ratings.dat vào bảng ratings. Bảng này lưu trữ dữ liệu userid (int), movieid (int) và rating (float).

Trên mỗi dòng của file ratings.dat, bộ dữ liệu có định dạng như sau:

UserID::MovieID::Rating::Timestamp

Ví dụ:

1::122::5::838985046

1::185::5::838983525

1::231::5::838983392

Nhưng bảng ratings ta cần tải nội dung vào chỉ lưu dữ liệu của 3 phần đầu (userid, movieid và rating), không lưu timestamp. Nên ý tưởng của nhóm là duyệt từng dòng trong

file, tách các phần trên từng dòng theo xâu "::", từ đó lấy được 4 phần dữ liệu, sau đó kiểm tra, nếu đủ 4 phần dữ liệu thì sẽ thêm vào trong 1 file ảo, sử dụng đối tượng StringIO. Sau đó nạp hết nội dung từ file ảo đó vào bảng ratings trong cơ sở dữ liệu.

1.2. Thực hiện

Đầu tiên, khởi tạo cursor để thiết lập kết nối với cơ sở dữ liệu.

```
cursor = open_connection.cursor()
```

Sau đó thực hiện tạo bảng ratings trong PostGres nếu bảng chưa tồn tại, sau đó commit để lưu lại trong cơ sở dữ liệu.

```
create_table_query = f"""
CREATE TABLE IF NOT EXISTS {ratings_table_name} (
    userid INT,
    movieid INT,
    rating FLOAT
);
"""
cursor.execute(create_table_query)  # tạo bảng ratings nếu chưa tồn tại
open_connection.commit()
```

Khởi tạo đối tượng StringIO (đã import thư viện io), để lưu data từ file ratings.dat vào 1 file ảo, ở đây nhóm đặt là formatted_data.

```
formatted_data = io.StringIO()
```

Mở file ratings.dat và duyệt từng dòng, dựa vào mục phân tích ở trên, nhóm tách từng dòng theo dấu "::" và chỉ lấy 3 phần đầu, bỏ qua timestamp. Kiểm tra nếu đủ 4 part, thực hiện viết vào file ảo formatted_data, với mỗi dòng là 3 phần (userid, movieid, timestamp) cách nhau bởi dấu tab "\t", hết dòng thì thực hiện xuống dòng "\n".

```
with open(file_path, 'r') as f:
    for line in f:
        # data từng dòng:  1::122::5::838985046
        parts = line.strip().split('::')  # Tách theo '::'
        if len(parts) == 4:  # kiểm tra nếu đủ 4 part
            formatted_data.write(f"{parts[0]}\t{parts[1]}\t{parts[2]}\n")  # ghi 3 phần đầu, cách nhau bởi dấu tab
        else:
            print(f"Dòng sai định dạng (userid, movieid, rating, timestamp): {line.strip()}")
```

Sau đó đưa con trỏ về đầu file ảo.

```
# đưa con trỏ StringIO về đầu trước khi chèn vào bảng ratings
formatted_data.seek(0)
```

Cuối cùng sử dụng hàm copy_from để tải tất cả data từ file ảo formatted_data (phân cách nhau bởi dấu tab) vào trong cơ sở dữ liệu, rồi commit lại.

```
cursor.copy_from(formatted_data, ratings_table_name, sep='\t')  # thêm vào bảng, phân cách bởi dấu tab
open_connection.commit()
```

2. Hàm rangepartition

2.1. Phân tích

Hàm rangepartition thực hiện phân mảnh bảng gốc thành nhiều phân mảnh theo giá trị rating:

- Trường hợp N = 1

Một bảng chứa tất cả các giá trị.

- Trường hợp N = 2

Hai bảng:

Partition 0 chứa các giá trị [0, 2.5]

Partition 1 chứa các giá trị (2.5, 5]

- Trường hợp N = 3

Ba bảng:

Partition 0 chứa [0, 1.67]

Partition 1 chứa (1.67, 3.34]

Partition 2 chứa (3.34, 5]

Từ đây, nhóm dựa vào giá trị rating, với min là 0, và max là 5, tính toán giá trị từng bước, ví dụ nếu có 3 phân mảnh, thực hiện lấy giá trị max là 5 (lưu ý phải ở dạng float) chia cho 3 (số phân mảnh), để tìm ra khoảng mà mỗi phân mảnh chiếm.

Cũng lưu ý thêm là chỉ có phân mảnh đầu tiên, sẽ lấy giá trị đầu mút nhỏ nhất (0), còn các phân mảnh sau thì không lấy, do đó cũng phải thực hiện if else để tạo truy vấn.

Ý tưởng thực hiện của nhóm là tạo một bảng trung gian tạm thời, lưu giá trị 4 cột: userid, movieid, rating, range_idx. Cột mới range_idx là để lưu phân mảnh mà bản ghi hiện tại thuộc về (tính từ index 0). Tức là ví dụ bản ghi là 1, 1, 5, 3, thì bản ghi đó thuộc vào phân mảnh 3. Sau đó lưu tất cả data từ bảng tạm thời vào các phân mảnh.

2.2. Thực hiện

Khởi tạo cursor để thiết lập kết nối với cơ sở dữ liệu và khởi tạo tên bảng trung gian.

```
cursor = open_connection.cursor()
temp_table_name = "temp_range_ratings_for_partition"
```

Xóa các bảng phân mảnh range nếu đã tồn tại.

```
# xóa bảng range partition nếu tồn tại
for i in range(N):
    drop_table_query = f"DROP TABLE IF EXISTS range_part{i} CASCADE;"
    cursor.execute(drop_table_query)
open_connection.commit()
```

Sau đó tạo lại các bảng phân mảnh mới

```
create_partition_table_query = f"""
CREATE TABLE range_part{{}} (
    userid INT,
    movieid INT,
    rating FLOAT
);
"""
for i in range(N):
    cursor.execute(create_partition_table_query.format(i))
open_connection.commit()
```

Khởi tạo max_rating là 5.0, tức là dạng float để tính toán giá trị step, khoảng rating mà mỗi phân mảnh chiếm.

```
min_rating = 0.0
max_rating = 5.0
step = max_rating / N
```

Sau đó nhóm thực hiện tạo bảng trung gian với 4 cột userid, movieid, rating và range_idx (phân mảnh mà bản ghi thuộc về). Trước hết phải tạo câu lệnh CASE WHEN, khởi tạo list case_clauses, tính toán các khoảng rating: lower_bound là cận dưới, upper_bound là cận trên, tất cả phân mảnh đều lấy giá trị rating nếu rating bằng cận trên, nhưng chỉ có phân mảnh đầu tiên, index 0 là lấy giá trị cận dưới.

```
# tạo các câu WHEN để tạo bảng trung gian
case_clauses = []
for i in range(N):
    lower_bound = min_rating + i * step
    upper_bound = min_rating + (i + 1) * step
    if i == 0:
        case_clauses.append(
            f"WHEN rating >= {lower_bound} AND rating <= {upper_bound} THEN {i}")
    else:
        case_clauses.append(
            f"WHEN rating > {lower_bound} AND rating <= {upper_bound} THEN {i}")
```

Giải thích thêm, nhóm cho min_rating vào để tường minh hơn, cận dưới được tính theo tổng min_rating và tích thứ tự phân mảnh với biến step. Ví dụ nếu là phân mảnh đầu tiên, cận dưới sẽ là 0 + 0 * step = 0, cận trên là 0 + 1 * step = step, tức là cận trên của phân mảnh thứ i trùng với cận dưới của phân mảnh thứ (i + 1).

Thực hiện tạo bảng trung gian, gồm 4 cột như đã nhắc tới ở trên, cột cuối được tính toán theo CASE WHEN, với các điều kiện WHEN đã được nối vào list case_clauses.

```
# tạo bảng trung gian, gồm 4 cột: userid, movieid, rating, range_idx (cột cuối để xác định thuộc phân mảnh nào)
# {' '.join(case_clauses)} -> nối với nhau bằng dấu cách, để tạo câu lệnh CASE WHEN
create_temp_table_query = f"""
CREATE TEMPORARY TABLE {temp_table_name} AS
SELECT
    userid,
    movieid,
    rating,
    CASE
        {' '.join(case_clauses)}
    END AS range_idx
FROM {ratings_table_name};
"""
cursor.execute(create_temp_table_query)
open_connection.commit()
```

Sau đó thực hiện tải data từ bảng trung gian sang các bảng phân mảnh, các bản ghi xác định phân mảnh theo cột cuối cùng của bảng trung gian (range_idx).

```
# chuyển data từ bảng trung gian qua các phân mảnh
for i in range(N):
    insert_query = f"""
    INSERT INTO range_part{i} (userid, movieid, rating)
    SELECT userid, movieid, rating
    FROM {temp_table_name}
    WHERE range_idx = {i};
    """
    cursor.execute(insert_query)
open_connection.commit()
```

Sau khi hoàn tất chuyển data từ bảng trung gian qua các phân mảnh, thực hiện xóa bảng trung gian vì không còn cần sử dụng nữa.

```
cursor.execute(f"DROP TABLE IF EXISTS {temp_table_name} CASCADE;") # xóa bảng trung gian
open_connection.commit()
```

3. Hàm roundrobinpartition

3.1. Phân tích

Sở dĩ round robin, cũng đã được giới thiệu qua ở môn Hệ điều hành, thuật toán thực hiện xoay vòng. Đối với yêu cầu phân mảnh của đề bài, lần lượt các dòng trong file ratings.dat sẽ phải truyền lần lượt vào các phân mảnh.

Ví dụ: có 3 phân mảnh (tính từ index 0), 10 bản ghi. Ta thực hiện phân mảnh xoay vòng như sau:

- Bản ghi 1 thuộc phân mảnh 0.
- Bản ghi 2 thuộc phân mảnh 1.
- Bản ghi 3 thuộc phân mảnh 2.
- Bản ghi 4 thuộc phân mảnh 0.
- Bản ghi 5 thuộc phân mảnh 1.
- Bản ghi 6 thuộc phân mảnh 2.
- Bản ghi 7 thuộc phân mảnh 0.
- Bản ghi 8 thuộc phân mảnh 1.
- Bản ghi 9 thuộc phân mảnh 2.

- Bản ghi 10 thuộc phân mảnh 0.

Từ đó, nhóm có ý tưởng tương tự như range partition, tạo 1 bảng trung gian, cũng có 4 cột, nhưng khác ở chỗ là cột cuối cùng bây giờ sẽ lưu số thứ tự hàng, ví dụ 10 bản ghi ở trên, số thứ tự hàng lần lượt là 1, 2, 3, … 10. Sử dụng hàm có sẵn ROW_NUMBER(), hàm này tính từ index 1.

Sau đó truyền qua các phân mảnh, sử dụng phép chia dư cho số phân mảnh. Lưu ý do hàm ROW_NUMBER() tính từ index 1, nhưng các phân mảnh tính từ index 0, nên khi chia dư, phải trừ giá trị số thứ tự hàng đi 1 rồi mới chia dư cho số phân mảnh.

3.2. Thực hiện

Khởi tạo cursor để kết nối với cơ sở dữ liệu, và biến tên của bảng trung gian.

```
cursor = open_connection.cursor()
temp_table_name = "temp_numbered_ratings_for_partition"
```

Xóa các bảng phân mảnh round robin nếu đã tồn tại.

```
# xóa bảng round robin partition nếu tồn tại
for i in range(N):
    drop_table_query = f"DROP TABLE IF EXISTS rrobin_part{i} CASCADE;"
    cursor.execute(drop_table_query)
open_connection.commit()
```

Tạo các bảng phân mảnh mới.

```
create_partition_table_query = f"""
CREATE TABLE rrobin_part{{}} (
    userid INT,
    movieid INT,
    rating FLOAT
);
"""
for i in range(N):
    cursor.execute(create_partition_table_query.format(i))
open_connection.commit()
```

Tạo bảng trung gian, với cột cuối cùng được tính toán theo hàm ROW_NUMBER(). Có sắp xếp theo ctid là vì hàm ROW_NUMBER() OVER đằng sau bắt buộc phải có

ORDER BY giá trị nào đó, nhưng ở đây muốn theo đúng thứ tự gốc, nên nhóm sắp xếp theo ctid, tức là cột đặc biệt của PostGres, lưu trữ vị trí vật lý của hàng trong bảng.

```
create_temp_table_query = f"""
CREATE TEMPORARY TABLE {temp_table_name} AS
SELECT
    userid,
    movieid,
    rating,
    ROW_NUMBER() OVER (ORDER BY ctid) AS row_number
FROM {ratings_table_name};
"""
cursor.execute(create_temp_table_query)
open_connection.commit()
```

Thực hiện chuyển data từ bảng trung gian sang các bảng phân mảnh, sử dụng câu lệnh điều kiện WHERE, thực hiện chia dư, nếu giá trị số thứ tự hàng (row_number) sau khi trừ 1, chia dư với số phân mảnh bằng giá trị của phân mảnh hiện tại thì thỏa mãn. Lí do phải trừ 1 cũng đã được giải thích ở phần phân tích, vì ROW_NUMBER() tính từ index 1, còn các phân mảnh theo đề bài tính từ index 0.

```
# truyền data từ bảng trung gian sang các phân mảnh
for i in range(N):
    insert_query = f"""
    INSERT INTO rrobin_part{i} (userid, movieid, rating)
    SELECT userid, movieid, rating
    FROM {temp_table_name}
    WHERE MOD((row_number - 1), {N}) = {i};
    """
    # row_number phải trừ 1 là vì hàm ROW_NUMBER() tính từ 1, nhưng các phân mảnh tính từ 0
    cursor.execute(insert_query)
open_connection.commit()
```

Sau khi thêm xong vào các phân mảnh, do không còn dùng tới bảng trung gian nữa nên thực hiện xóa.

```
cursor.execute(f"DROP TABLE IF EXISTS {temp_table_name} CASCADE;")
open_connection.commit()
```

4. Hàm roundrobininsert

4.1. Phân tích

Về cơ bản, logic của hàm round robin partition và round robin insert là tương tự nhau. Tức là để xác định 1 bản ghi thuộc về phân mảnh nào, phải tìm được số thứ tự hàng của bản ghi đó. Vì vậy trong hàm này, trước hết phải thêm bản ghi cần thêm vào bảng gốc.

Từ đó tính toán được số thứ tự hàng của bản ghi mới nhất, tức là tổng số hàng của bảng ratings (đã được thêm bản ghi mới nhất rồi). Sau đó lấy thứ tự hàng trừ đi 1, chia dư cho tổng số phân mảnh, là tìm được phân mảnh mà bản ghi đó thuộc về. Nhắc lại, phải trừ đi 1 là do khi tính toán thứ tự cột (dùng câu lệnh SELECT), thì vô hình chung đã tính từ index 1, còn theo yêu cầu đề bài, các phân mảnh tính từ index 0.

4.2. Thực hiện

Khởi tạo cursor kết nối với cơ sở dữ liệu.

```
cursor = open_connection.cursor()
```

Thực hiện thêm vào bảng ratings gốc.

```
# thêm vào bảng ratings chung
insert_main_query = f"""
INSERT INTO {ratings_table_name} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cursor.execute(insert_main_query, (UserID, ItemID, Rating))
open_connection.commit()
```

Sau đó đếm số phân mảnh round robin, sử dụng hàm thầy viết sẵn (do yêu cầu đề không có phần implement hàm này nên nhóm tái sử dụng luôn). Sau đó kiểm tra nếu không tồn tại phân mảnh nào thì báo lỗi và dừng hàm.

```
num_rrobin_partitions = count_partitions( prefix: 'rrobin_part', open_connection)

if num_rrobin_partitions == 0:
    print("Không tồn tại bảng phân mảnh round robin nào")
    return
```

Tính toán tổng số bản ghi sau ghi thêm bản ghi mới nhất (để tìm được thứ tự hàng của bản ghi mới nhất). Phải fetchone vì khi dùng thư viện psycopg2, khi thực hiện truy vấn SELECT, kết quả trả về cho cursor là 1 tuple, nên muốn lấy kết quả phải lấy index 0.

```
get_total_rows_query = f"SELECT COUNT(*) FROM {ratings_table_name};"
cursor.execute(get_total_rows_query)
total_rows_after_insert = cursor.fetchone()[0]
```

Tính toán xem bản ghi thuộc phân mảnh nào, bằng cách lấy tổng số hàng sau khi thêm trừ 1 rồi chia dư cho số phân mảnh.

```
# phải trừ 1 là vì hàm ROW_NUMBER() tính từ 1, nhưng các phân mảnh tính từ 0
partition_index = (total_rows_after_insert - 1) % num_rrobin_partitions
target_rrobin_table = f"rrobin_part{partition_index}"
```

Thực hiện thêm vào phân mảnh và commit.

```
# thêm vào phân mảnh
insert_rrobin_query = f"""
INSERT INTO {target_rrobin_table} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cursor.execute(insert_rrobin_query, (UserID, ItemID, Rating))
open_connection.commit()
```

5. Hàm rangeinsert

5.1. Phân tích

Logic xử lý của range partition và range insert cũng tương tự nhau. Tức là sẽ phải tính toán khoảng mà mỗi phân mảnh chiếm, nếu có 2 phân mảnh thì mỗi phân mảnh chiếm 2.5 điểm rating, nếu có 3 thì mỗi phân mảnh chiếm 1.67 điểm rating, tương tự với số phân mảnh khác. Sau đó tính giá trị cận trên và cận dưới, nếu rating của bản ghi cần thêm mới, thuộc vào khoảng cận trên, cận dưới của phân mảnh nào thì thêm vào phân mảnh đó.

5.2. Thực hiện

Khởi tạo cursor để kết nối với cơ sở dữ liệu

```
cursor = open_connection.cursor()
```

Thêm vào bảng ratings gốc.

```
# thêm vào bảng ratings chung
insert_main_query = f"""
INSERT INTO {ratings_table_name} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cursor.execute(insert_main_query, (UserID, ItemID, Rating))
open_connection.commit()
```

Tính toán số lượng phân mảnh range. Nếu không tồn tại phân mảnh nào thì dừng hàm lại.

```
num_range_partitions = count_partitions( prefix: 'range_part', open_connection)

if num_range_partitions == 0:
    print("Không tồn tại phân mảnh range nào")
    return
```

Tương tự như hàm range partition, tính giá trị step, khoảng mà mỗi phân mảnh chiếm.

```
min_rating = 0.0
max_rating = 5.0
step = max_rating / num_range_partitions
```

Thực hiện tìm xem bản ghi cần thêm thuộc phân mảnh nào, chi tiết tính cận trên và cận dưới cũng đã được giải thích kĩ ở phần range partition nên nhóm không giải thích lại ở đây. Chỉ giải thích thêm là nếu ở phân mảnh đầu, thì kiểm tra cả giá trị bằng cận dưới, còn phân mảnh cuối không kiểm tra giá trị bằng cận dưới, và khi kiểm tra đúng, cập nhật phân mảnh mà bản ghi thuộc về, rồi break để dừng vòng for.

```
target_partition_index = -1
for i in range(num_range_partitions):
    lower_bound = min_rating + i * step
    upper_bound = min_rating + (i + 1) * step

    if i == 0:
        if lower_bound <= Rating <= upper_bound:
            target_partition_index = i
            break
    else:
        if lower_bound < Rating <= upper_bound:
            target_partition_index = i
            break
```

Kiểm tra nếu khi tính giá trị cận trên, bị dính phải vấn đề floating point của python, ví dụ khi nhân (i + 1) với step, nếu giá trị cận trên của phân mảnh cuối cùng không phải 5.0 tròn, mà là 4.99, hay 1 giá trị nào đó xấp xỉ do floating point. Thì khi kiểm tra điều kiện Rating <= upper_bound sẽ không thỏa mãn (do 5.0 <= 4.99 là sai). Do đó nhóm kiểm tra lại, nếu Rating = 5 và không chạy vào câu lệnh if nào của vòng for thì phân mảnh mà bản ghi thuộc về là phân mảnh cuối cùng.

```
# kiểm tra nếu Rating = 5.0 nhưng không thỏa mãn đk if else ở vòng for trên -> do floating point
if target_partition_index == -1 and Rating == max_rating:
    target_partition_index = num_range_partitions - 1
```

Cập nhật tên phân mảnh và thêm vào phân mảnh đúng.

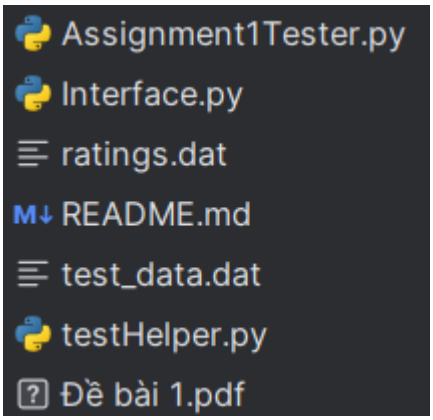
```
target_range_table = f"range_part{target_partition_index}"

# thêm vào phân mảnh
insert_range_query = f"""
INSERT INTO {target_range_table} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cursor.execute(insert_range_query, (UserID, ItemID, Rating))
open_connection.commit()
```

III. Kiểm thử

1. Cấu hình chung

Đặt file ratings.dat cùng mức với các file khác



Trong file Assignment1Tester.py, thực hiện sửa đổi đường dẫn tới file ratings.dat và số hàng thành 10.000.054

```
INPUT_FILE_PATH = 'ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 1e7 + 54
```

2. Kết quả kiểm thử lần 1

2.1. Kết quả chung

Khi thực hiện chạy file Assignment1Tester.py, với testcase như mẫu của thầy, tức là cả range partition và round robin partition đều phân thành 5 phân mảnh, range insert kiểm tra sau khi thêm bản ghi có userid là 100, movieid là 2, rating là 3 thì phân vào phân mảnh 2. Chỉ thay testcase của hàm round robin insert, thay thành sau khi thêm bản ghi có userid là 100, movieid là 1, rating là 3 thì phải phân vào phân mảnh 4.

Lí do phải thay testcase của round robin insert là vì ban đầu file test_data.dat mẫu của thầy chỉ có 20 bản ghi, khi thêm 1 bản ghi thì thành 21, sau đó dùng round robin insert, thì bản ghi mới nhất sẽ phải vào phân mảnh (21 – 1) % 5 = 0, tức là phân mảnh 0, nhưng khi thay thành file ratings.dat, có 10.000.054 bản ghi, khi thêm bản ghi mới nhất thì sẽ phải vào phân mảnh (10.000.055 – 1) % 5 = 4, tức là phân mảnh 4.

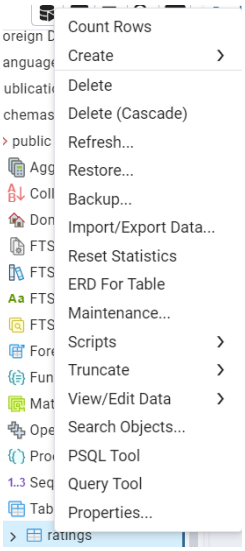
```
testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, userid: 100, itemid: 1, rating: 3, conn, expectedtableindex: '4')
```

Kết quả chung (thời gian nhóm ước lượng: khoảng 4 phút 10 giây):

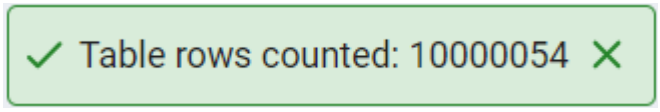
```
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
Press enter to Delete all tables?
```

2.2. Hàm loadratings

Đầu tiên là comment toàn bộ phần không liên quan tới loadratings, sau đó chạy và chọn Count Rows (hàng đầu tiên khi chuột phải vào bảng ratings) trong PostGres:



Kết quả số hàng:



Khi chạy câu lệnh SELECT, mỗi trang chứa 1000 bản ghi, kết quả trang đầu tiên (Page No. 1):

Showing rows: 1 to 1000 | Page No: 1 | of 10001

	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

Total rows: 10000054 | Query complete 00:00:03.037 | CRLF | Ln 2, Col 22

Kết quả trang cuối cùng (Page No. 10001):

Showing rows: 10000001 to 10000054 | Page No: 10001 | of 10001

	userid integer	movieid integer	rating double precision
10000001	71566	595	5
10000002	71567	32	3
10000003	71567	110	5
10000004	71567	196	4
10000005	71567	256	3
10000006	71567	260	5

Total rows: 10000054 | Query complete 00:00:03.037 | CRLF | Ln 2, Col 22

2.3. Hàm rangepartition

Comment toàn bộ phần từ rangeinsert trở đi.

Sử dụng thao tác tương tự, hiển thị được số bản ghi tổng ở bảng ratings:

✓ Table rows counted: 10000054 ×

Thực hiện truy vấn SELECT, thấy được số lượng bản ghi ở dòng cuối cùng góc trái của từng ảnh (Total rows):

Bảng ratings: có 10.000.054 bản ghi

Query Query History

```
SELECT userid, movieid, rating
    FROM public.ratings;
```

Data Output Messages Notifications

Showing rows: 10000001 to 10000054 Page No:

	userid integer	movieid integer	rating double precision
10000001	71566	595	5
10000002	71567	32	3
10000003	71567	110	5
10000004	71567	196	4
10000005	71567	256	3
10000006	71567	260	5
10000007	71567	316	4
10000008	71567	442	2
10000009	71567	480	4
10000010	71567	589	4
10000011	71567	780	4
10000012	71567	788	4
10000013	71567	829	4
10000014	71567	891	1

Total rows: 10000054 Query complete 00:00:03.037

Phân mảnh 0: có 479.168 bản ghi

Query Query History

```
SELECT userid, movieid, rating
    FROM public.range_part0;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 479168 Query complete 00:00:00.213

Phân mảnh 1: có 908.584 bản ghi

Query Query History

```
SELECT userid, movieid, rating
    FROM public.range_part1;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	7	2335	2
12	7	5184	2
13	8	345	1.5

Total rows: 908584 Query complete 00:00:00.360

Phân mảnh 2: có 2.726.854 bản ghi

Query Query History

```
SELECT userid, movieid, rating
    FROM public.range_part2;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	2	1356	3
12	2	1391	3
13	2	1544	3

Total rows: 2726854 Query complete 00:00:00.961

Phân mảnh 3: có 3.755.614 bản ghi

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part3;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	3	5952	3.5
12	3	6377	4
13	3	7153	4

Total rows: 3755614 Query complete 00:00:01.449

Phân mảnh 4: có 2.129.834 bản ghi

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part4;
```

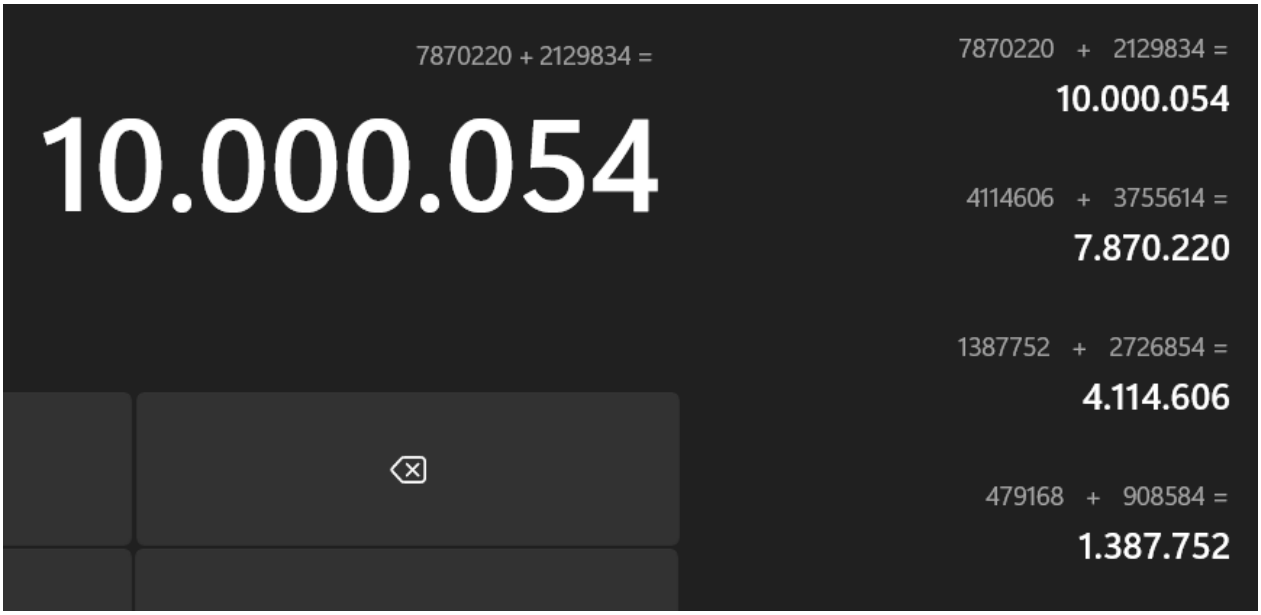
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 2129834 Query complete 00:00:01.150

Thực hiện cộng tổng số bản ghi (dùng Calculator có sẵn trong máy), lịch sử cộng xếp theo gần nhất, tức là phép tính cuối cùng là phân mảnh 0 và 1, ngay phía trên là tổng vừa tính cộng với phân mảnh 2,… Kết quả cuối cùng bằng với tổng số bản ghi của bảng ratings, vậy kết quả phân mảnh đã đúng:



2.4. Hàm rangeinsert

Comment toàn bộ phần từ roundrobinpartition đổ đi.

SELECT bảng ratings lúc sau khi thêm, số bản ghi là 10.000.055 (tăng lên 1)

Query Query History

```
1  SELECT userid, movieid, rating
2      FROM public.ratings;
```

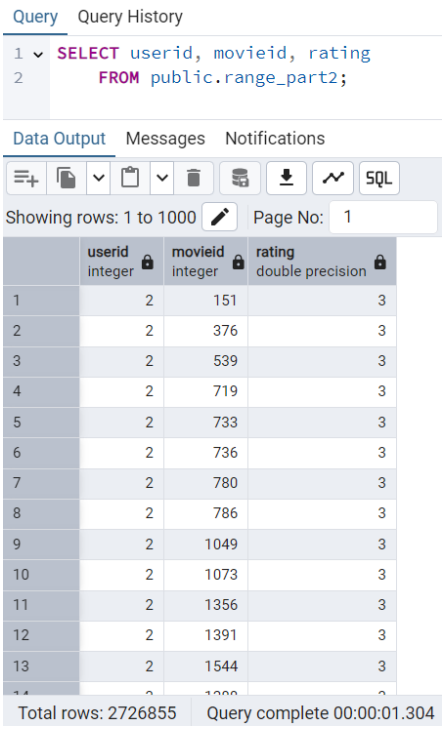
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

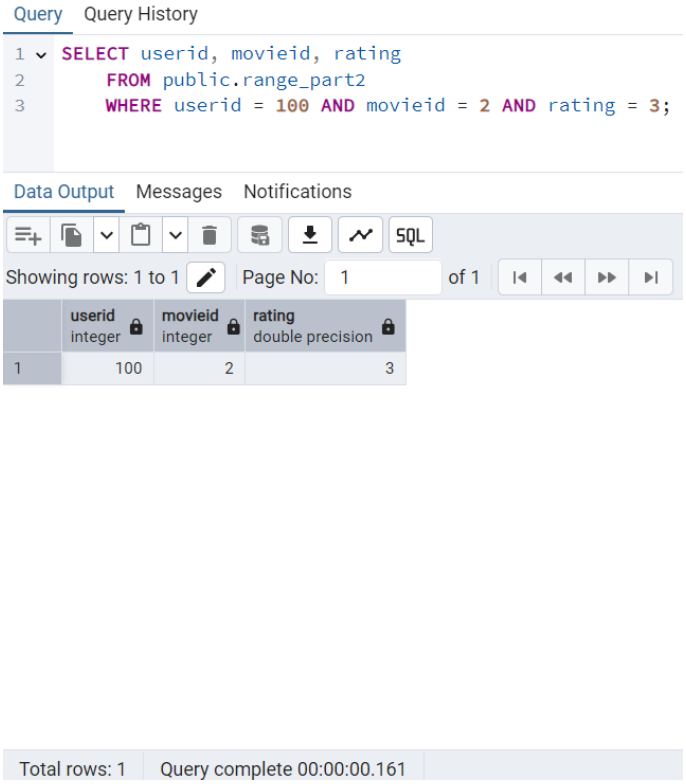
	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5
13	1	420	5
14	1	466	5

Total rows: 10000055 Query complete 00:00:05.534

Vì bản ghi mới thêm có data là userid: 100, movieid: 2, rating: 3 nên bảng dự kiến được thêm vào là phân mảnh 2. SELECT phân mảnh 2, có 2.726.855 bản ghi (tăng lên 1).



Thực hiện SELECT có điều kiện WHERE để xác định chính xác.



Như vậy là đã thêm thành công, chính xác vào phân mảnh 2.

2.5. Hàm roundrobinpartition

Comment toàn bộ phần range partition, range insert và round robin insert.

Thực hiện SELECT với từng phân mảnh, thấy số bản ghi ở dòng cuối (Total rows).

Phân mảnh 0: có 2.000.011 phân mảnh

Query Query History

```
SELECT userid, movieid, rating
    FROM public.rrobin_part0;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 2000011 Query complete 00:00:00

Phân mảnh 1: có 2.000.011 phân mảnh

Query Query History

```
SELECT userid, movieid, rating
    FROM public.rrobin_part1;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	3	1408	3.5
12	3	3408	4

Total rows: 2000011 Query complete 00:00:00.754

Phân mảnh 2: có 2.000.011 phân mảnh

Query Query History

```
SELECT userid, movieid, rating
    FROM public.rrobin_part2;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	3	1552	2
12	3	3684	4.5

Total rows: 2000011 Query complete 00:00:01

Phân mảnh 3: có 2.000.011 phân mảnh

Query Query History

```
SELECT userid, movieid, rating
    FROM public.rrobin_part3;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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
11	3	1564	4.5
12	3	4535	4

Total rows: 2000011 Query complete 00:00:00

Phân mảnh 4: có 2.000.010 phân mảnh

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.rrobin_part4;
```

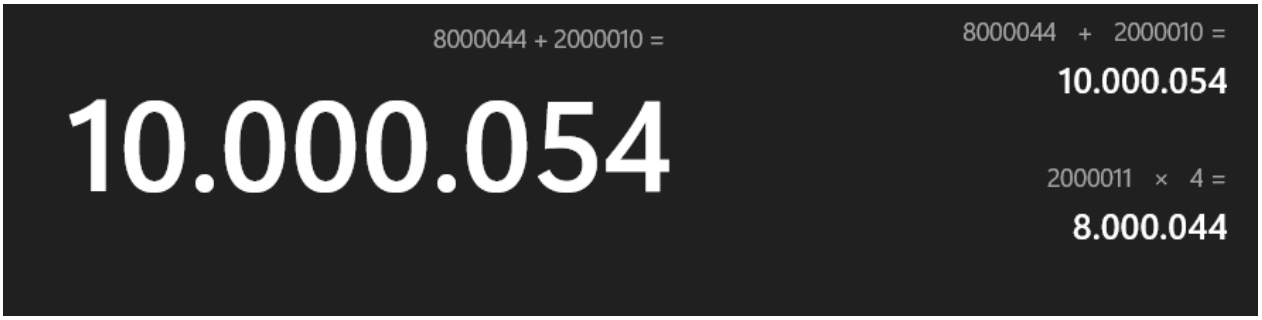
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 2000010 Query complete 00:00:0(

Thực hiện tính tổng các phân mảnh, ta được kết quả bằng với số bản ghi của bảng ratings.



2.6. Hàm roundrobininsert

Comment toàn bộ phần range partition và range insert.

Thực hiện SELECT để xem tổng số bản ghi, phần Total rows ở cuối ảnh.

Bảng ratings sau khi insert, có 10.000.055 bản ghi (đã tăng lên 1).

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.ratings;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 10000055 Query complete 00:00:0

Vì lúc sau khi tăng, số thứ tự hàng của bản ghi mới nhất là 10.000.055 nên phân mảnh nó thuộc về là (10.000.055 – 1) % 5 = 4, do vậy thực hiện SELECT với phân mảnh 4.

Phân mảnh 4 sau khi insert: có 2.000.011 bản ghi (đã tăng lên 1).

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.rrobin_part4;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 2000011 Query complete 00:00:0

Bản ghi vừa thêm có data là userid: 100, movieid: 1, rating: 3, thực hiện kiểm tra trong phân mảnh 4:

Như vậy đã thêm thành công bản ghi vào phân mảnh 4.

3. Kết quả kiểm thử lần 2

3.1. Kết quả chung

Thực hiện chạy file Assignment1Tester.py sau khi thay đổi testcase, cụ thể thay đổi:

- 2 hàm test range partition và round robin partition thay số phân mảnh thành 3.

```
testHelper.testrangepartition(MyAssignment, RATINGS_TABLE, n: 3, conn, partitionstartindex: 0, ACTUAL_ROWS_IN_INPUT_FILE)
```

```
testHelper.testroundrobinpartition(MyAssignment, RATINGS_TABLE, numberofpartitions: 3, conn, partitionstartindex: 0, ACTUAL_ROWS_IN_INPUT_FILE)
```

- Hàm test range insert vẫn giữ thông tin bản ghi thêm vào là userid: 100, movieid: 2, rating: 3, nhưng thay đổi phân mảnh dự kiến sẽ được thêm vào là phân mảnh 1. Vì khi chia 3 phân mảnh, phân mảnh 0 sẽ chiếm [0; 1,67], phân mảnh 1 sẽ chiếm (1,67;3,34], phân mảnh 2 sẽ chiếm (3,34;5]. Mà rating của bản ghi mới là 3 => thuộc phân mảnh 1.

```
testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, userid: 100, itemid: 2, rating: 3, conn, expectedtableindex: '1')
```

- Hàm test round robin insert vẫn giữ thông tin bản ghi thêm vào là userid: 100, movieid: 1, rating: 3, nhưng thay đổi phân mảnh dự kiến sẽ được thêm vào là phân mảnh 1. Vì sau khi thêm bản ghi mới, tổng số bản ghi trong bảng ratings là 10.000.055, nên số thứ tự hàng của bản ghi mới nhất là 10.000.055. Thực hiện phép tính (10.000.055 – 1) mod 3 = 1, nên bản ghi mới nhất thuộc phân mảnh 1.

```
testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, userid: 100, itemid: 1, rating: 3, conn, expectedtableindex: '1')
```

Kết quả chung (thời gian nhóm ước lượng: khoảng 4 phút 10 giây):

```
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?
```

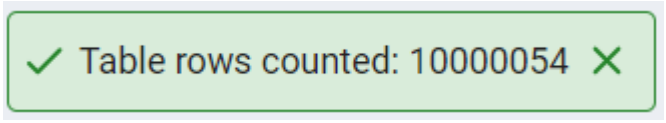
3.2. Hàm loadratings

Do không thay đổi các tham số khi truyền vào nên kết quả tương tự như kiểm thử lần 1.

3.3. Hàm rangepartition

Comment toàn bộ phần từ rangeinsert trở đi.

Sử dụng thao tác tương tự như ở lần kiểm thử 1, hiển thị được số bản ghi tổng ở bảng ratings:



Thực hiện truy vấn SELECT, thấy được số lượng bản ghi ở dòng cuối cùng góc trái của từng ảnh (Total rows):

Phân mảnh 0: có 597.446 bản ghi

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part0;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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	1895	1.5
10	7	1917	1
11	7	2478	1
12	7	5094	1
13	8	345	1.5
14	8	370	1.5

Total rows: 597446 Query complete 00:00:0

Phân mảnh 1: có 3.517.160 bản ghi

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part1;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	2	151	3
2	2	376	3
3	2	539	3
4	2	648	2
5	2	719	3
6	2	733	3
7	2	736	3
8	2	780	3
9	2	786	3
10	2	802	2
11	2	858	2
12	2	1049	3
13	2	1073	3
14	2	1356	3

Total rows: 3517160 Query complete 00:00:0

Phân mảnh 2: có 5.885.448 bản ghi

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part2;
```

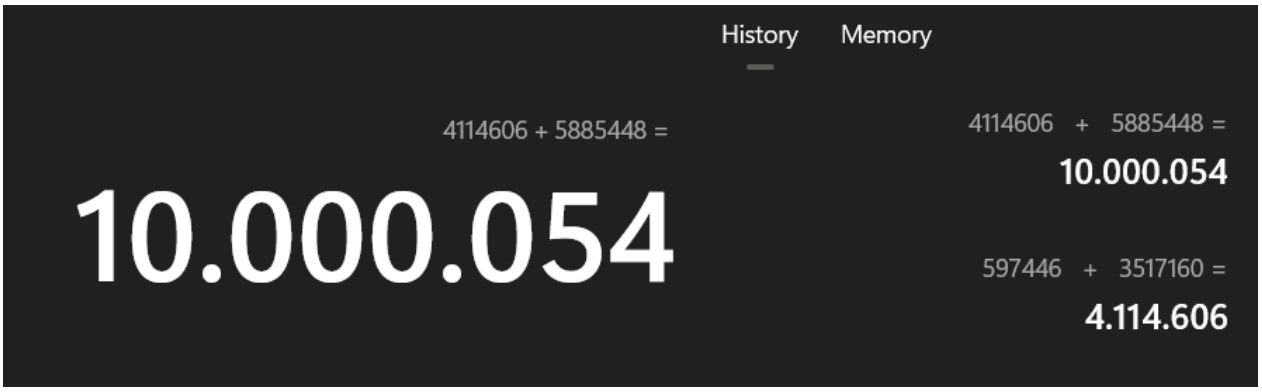
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 5885448 Query complete 00:00:0

Thực hiện cộng tổng số bản ghi (dùng Calculator có sẵn trong máy), lịch sử cộng xếp theo gần nhất, tức là phép tính cuối cùng là phân mảnh 0 và 1, ngay phía trên là tổng vừa tính cộng với phân mảnh 2. Kết quả cuối cùng bằng với tổng số bản ghi của bảng ratings, vậy kết quả phân mảnh đã đúng:



3.4. Hàm rangeinsert

Comment toàn bộ phần từ roundrobinpartition đổ đi.

SELECT bảng ratings lúc sau khi thêm, số bản ghi là 10.000.055 (tăng lên 1)

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.ratings;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
17	1	539	5
18	1	586	5
19	1	588	5
20	1	589	5
21	1	594	5
22	1	616	5
23	2	110	5
24	2	151	3
25	2	260	5
26	2	376	3
27	2	539	3
28	2	590	5
29	2	648	2

Total rows: 10000055 Query complete 00:00:0

Vì bản ghi mới thêm có data là userid: 100, movieid: 2, rating: 3 nên bảng dự kiến được thêm vào là phân mảnh 1. SELECT phân mảnh 1, có 3.517.161 bản ghi (tăng lên 1).

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.range_part1;
```

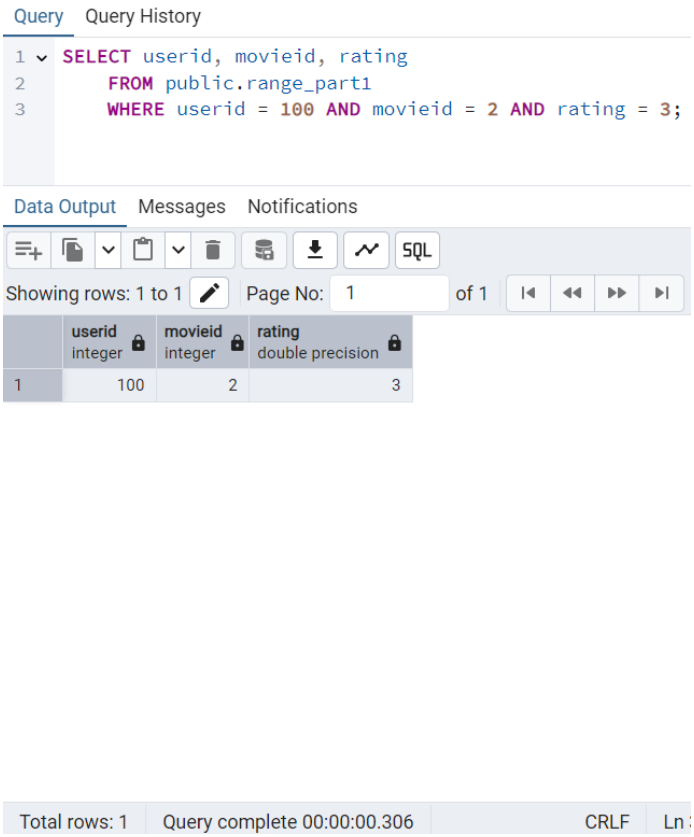
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	2	151	3
2	2	376	3
3	2	539	3
4	2	648	2
5	2	719	3
6	2	733	3
7	2	736	3
8	2	780	3
9	2	786	3
10	2	802	2
11	2	858	2
12	2	1049	3
13	2	1073	3

Total rows: 3517161 Query complete 00:00:0

Thực hiện SELECT đối với phân mảnh 1, có điều kiện WHERE để xác định chính xác.



Như vậy là đã thêm thành công, chính xác vào phân mảnh 1.

3.5. Hàm roundrobinpartition

Comment toàn bộ phần range partition, range insert và round robin insert.

Thực hiện SELECT với từng phân mảnh, thấy số bản ghi ở dòng cuối (Total rows).

Phân mảnh 0: có 3.333.352 phân mảnh

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.rrobin_part0;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	292	5
3	1	355	5
4	1	364	5
5	1	420	5
6	1	520	5
7	1	588	5
8	1	616	5
9	2	260	5
10	2	590	5
11	2	733	3
12	2	786	3
13	2	1049	3

Total rows: 3333352 Query complete 00:00:02

Phân mảnh 1: có 3.333.351 phân mảnh

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.rrobin_part1;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	1	185	5
2	1	316	5
3	1	356	5
4	1	370	5
5	1	466	5
6	1	539	5
7	1	589	5
8	2	110	5
9	2	376	3
10	2	648	2
11	2	736	3
12	2	802	2
13	2	1073	3

Total rows: 3333351 Query complete 00:00:04

Phân mảnh 2: có 3.333.351 phân mảnh

Query Query History

```
1 SELECT userid, movieid, rating
2     FROM public.rrobin_part2;
```

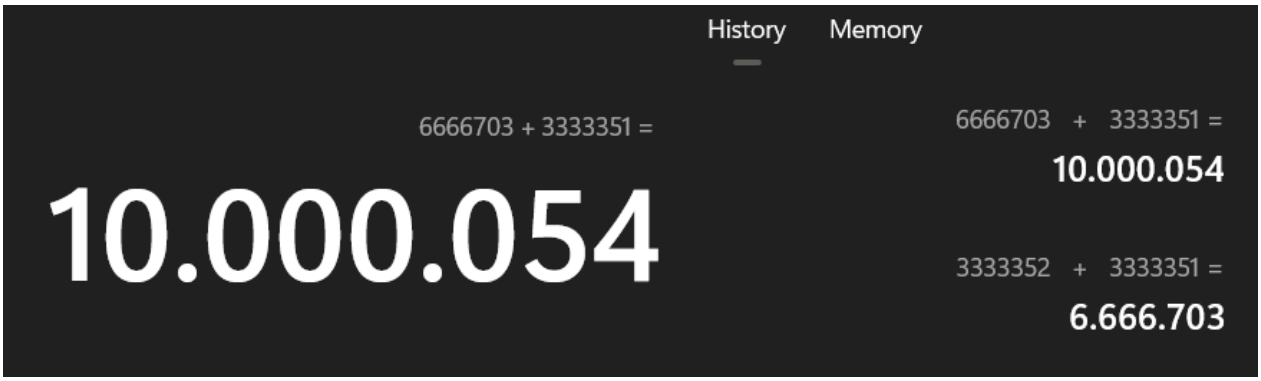
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	1	231	5
2	1	329	5
3	1	362	5
4	1	377	5
5	1	480	5
6	1	586	5
7	1	594	5
8	2	151	3
9	2	539	3
10	2	719	3
11	2	780	3
12	2	858	2
13	2	1210	4

Total rows: 3333351 Query complete 00:00:03

Thực hiện tính tổng các phân mảnh, ta được kết quả bằng với số bản ghi của bảng ratings.



3.6. Hàm roundrobininsert

Comment toàn bộ phần range partition và range insert.

Thực hiện SELECT để xem tổng số bản ghi, phần Total rows ở cuối ảnh.

Bảng ratings sau khi insert, có 10.000.055 bản ghi (đã tăng lên 1).

Query Query History

```
SELECT userid, movieid, rating
    FROM public.ratings;
```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	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

Total rows: 10000055 Query complete 00:00:15

Vì lúc sau khi tăng, số thứ tự hàng của bản ghi mới nhất là 10.000.055 nên phân mảnh nó thuộc về là (10.000.055 – 1) % 3 = 1, do vậy thực hiện SELECT với phân mảnh 1.

Phân mảnh 1 sau khi insert: có 3.333.352 bản ghi (đã tăng lên 1).

Query Query History

```
SELECT userid, movieid, rating
    FROM public.rrobin_part1;
```

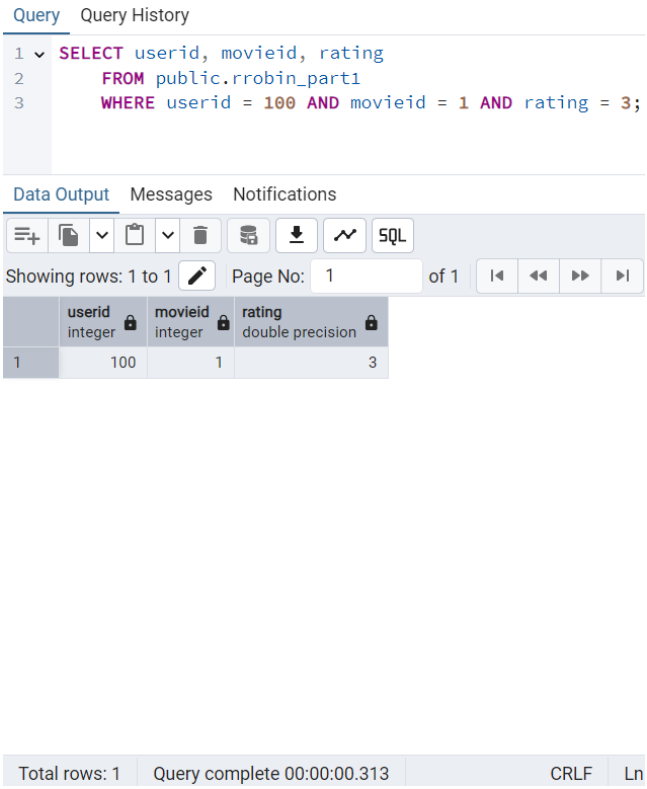
Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1

	userid integer	movieid integer	rating double precision
1	1	185	5
2	1	316	5
3	1	356	5
4	1	370	5
5	1	466	5
6	1	539	5
7	1	589	5
8	2	110	5
9	2	376	3
10	2	648	2
11	2	736	3
12	2	802	2
13	2	1073	3

Total rows: 3333352 Query complete 00:00:0

Bản ghi vừa thêm có data là userid: 100, movieid: 1, rating: 3, thực hiện kiểm tra trong phân mảnh 1 với câu lệnh điều kiện WHERE:



Như vậy đã thêm thành công bản ghi vào phân mảnh 1.

IV. Kết luận

Sau quá trình tìm hiểu và thực hiện bài tập lớn này, nhóm chúng em đã triển khai thành công hai kỹ thuật phân mảnh là phân mảnh theo khoảng giá trị (range partition) và phân mảnh vòng tròn (round robin partition) trên bảng dữ liệu ratings. Cả hai phương pháp đã được kiểm thử và chạy đúng với các trường hợp được đưa ra.

Từ đó, nhóm cũng đã hiểu rõ hơn về nguyên lý hoạt động của phân mảnh trong hệ thống cơ sở dữ liệu phân tán, cũng như rèn luyện kỹ năng thao tác với PostgreSQL và lập trình xử lý dữ liệu lớn. Đây là nền tảng quan trọng để chúng em có thể học cách xây dựng các hệ thống dữ liệu phân tán hiệu quả và có khả năng mở rộng trong thực tế, áp dụng cho công việc trong tương lai.