

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



**BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN: CƠ SỞ DỮ LIỆU PHÂN TÁN
MÃ HỌC PHẦN: INT14148**

**ĐỀ TÀI: Mô phỏng các phương pháp phân mảnh dữ liệu trên một
hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở.**

Nhóm 12

Nhóm sinh viên thực hiện:

B22DCCN469 – Hoàng Văn Khởi

B22DCCN505 – Phạm Thành Long

B22DCCN589 – Hoàng Cao Nguyên

Tên lớp: nhóm 09

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

HÀ NỘI 2024

PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN

TT	Công việc / Nhiệm vụ	SV thực hiện
1	Cài đặt hàm LoadRatings và viết báo cáo	Hoàng Văn Khởi
2	Cài đặt hàm RoundRobin_Partition	Phạm Thành Long
3	Cài đặt hàm Range_Partition	Hoàng Cao Nguyên

MỤC LỤC

MỤC LỤC	3
DANH MỤC CÁC HÌNH VẼ.....	4
CHƯƠNG 1. GIỚI THIỆU	5
1.1 Giới thiệu tổng quan.....	5
1.2 Mục tiêu của bài	5
1.3 Lý do chọn công nghệ	5
CHƯƠNG 2. TỔNG QUAN LÝ THUYẾT.....	6
2.1 Phân mảnh cơ sở dữ liệu	6
2.2 Kỹ thuật phân mảnh ngang.....	6
2.2.1 Kỹ thuật Range Partitioning	6
2.2.2 Kỹ thuật Round Robin Partitioning.....	7
2.3 Mục tiêu của phân mảnh	7
CHƯƠNG 3. THỬ NGHIỆM	8
3.1 Triển khai và thử nghiệm	8
3.1.1 Yêu cầu môi trường.....	8
3.1.2 Hàm đọc dữ liệu LoadRatings.....	9
3.1.3 Hàm phân mảnh ngang theo phạm vi của rating.....	11
3.1.4 Hàm chèn tuple dữ liệu mới vào phân mảnh ngang.....	13
3.1.5 Xây dựng hàm roundrobinpartition.....	14
3.1.6 Xây dựng hàm RoundRobin_insert.....	18

DANH MỤC CÁC HÌNH VẼ

Hình 1 – Hệ điều hành ubuntu	8
Hình 2 – Phiên bản python	8
Hình 3 – Hệ quản trị cơ sở dữ liệu PostgreSQL.....	8
Hình 4 – Tập đầu vào ratings.dat	9
Hình 5 – Hàm LoadRatings.....	10
Hình 6 - Hàm phân mảnh ngang theo phạm vi	12
Hình 7 – Kết quả chạy phân mảnh (range partition)	12
Hình 8 – Cơ sở dữ liệu sau phân mảnh (range partition)	13
Hình 9 – Kết quả kiểm tra thông qua tester (range partition)	13
Hình 10 - Hàm chèn dữ liệu vào phân mảnh ngang theo phạm vi	14
Hình 11 – Kết quả chạy và kiểm tra bằng tester (range partition)	14
Hình 12 – Hàm phân mảnh ngang theo ghép cặp xoay vòng.....	15
Hình 13 – Kết quả phân mảnh (round robin partition).....	16
Hình 14 – Kết quả phân mảnh số 1 (round robin partition)	16
Hình 15 – Kết quả phân mảnh số 2 (round robin partition)	17
Hình 16 - Kết quả phân mảnh số 3 (round robin partition).....	17
Hình 17 - Kết quả phân mảnh số 4 (round robin partition).....	18
Hình 18 - Kết quả phân mảnh số 5 (round robin partition).....	18
Hình 19 – Hàm round robin insert	19
Hình 20 – Kết nối database	19
Hình 21 – Đếm số lượng phân mảnh	19
Hình 22 – Thêm dữ liệu vào phân mảnh	20
Hình 23 – Kết quả chạy với terminal	20
Hình 24 – Bộ test kiểm tra	20
Hình 25 – Kết quả sau khi chạy	21
Hình 26 – Gọi hàm kiểm tra.....	21
Hình 27 – Hàm kiểm tra round robin partition.....	22
Hình 28 – Hàm kiểm tra insert	22

CHƯƠNG 1. GIỚI THIỆU

1.1 Giới thiệu tổng quan

Bài tập lớn tập trung vào việc mô phỏng các phương pháp phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, cụ thể là PostgreSQL. Dự án yêu cầu triển khai các hàm Python để tải dữ liệu từ tập tin ratings.dat của MovieLens, thực hiện phân mảnh ngang dữ liệu theo hai phương pháp (Range Partitioning và Round Robin Partitioning), và hỗ trợ chèn dữ liệu mới vào các phân mảnh tương ứng.

1.2 Mục tiêu của bài

Áp dụng phân mảnh ngang: Thực hiện phân mảnh dữ liệu theo hai kỹ thuật Range Partitioning và Round Robin Partitioning trên bảng Ratings.

Xây dựng các hàm thao tác dữ liệu: Triển khai các hàm Python để tải dữ liệu, phân mảnh dữ liệu, và chèn dữ liệu mới vào các phân mảnh trong cơ sở dữ liệu PostgreSQL hoặc MySQL.

Đảm bảo tính chính xác và hiệu quả: Đảm bảo các hàm hoạt động đúng trên tập dữ liệu lớn (10 triệu đánh giá) và duy trì tính toàn vẹn của dữ liệu khi chèn.

1.3 Lý do chọn công nghệ

Python: Python được chọn vì cú pháp đơn giản, dễ đọc, và có nhiều thư viện hỗ trợ làm việc với cơ sở dữ liệu như psycopg2 (cho PostgreSQL). Python cũng phù hợp để xử lý tệp dữ liệu lớn như ratings.dat.

PostgreSQL: PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở mạnh mẽ, hỗ trợ tốt các tính năng phân mảnh và quản lý dữ liệu lớn. Nó cũng có cộng đồng phát triển tích cực và tài liệu phong phú, giúp dễ dàng triển khai và kiểm tra.

CHƯƠNG 2. TỔNG QUAN LÝ THUYẾT

2.1 Phân mảnh cơ sở dữ liệu

Phân mảnh cơ sở dữ liệu (database partitioning) là kỹ thuật chia một bảng dữ liệu lớn thành các phần nhỏ hơn (phân mảnh) để tăng hiệu suất truy vấn, dễ dàng quản lý dữ liệu, và cải thiện khả năng mở rộng. Phân mảnh giúp giảm thời gian truy cập dữ liệu bằng cách giới hạn phạm vi dữ liệu cần xử lý trong mỗi truy vấn.

2.2 Kỹ thuật phân mảnh ngang

Phân mảnh ngang chính là việc chia quan hệ thành nhiều các nhóm bộ. Kết quả của quá trình phân mảnh ngang là các quan hệ con, số lượng quan hệ con phụ thuộc vào điều kiện ràng buộc của các thuộc tính. Và các bộ trong các quan hệ con là tách biệt nhau. Phân mảnh ngang thực chất là phép chọn quan hệ thỏa mãn một biểu thức điều kiện cho trước.

Trong bài tập đang sử dụng hai kỹ thuật Range Partitioning và Round Robin Partitioning.

2.2.1 Kỹ thuật Range Partitioning

Range Partitioning chia dữ liệu thành các phân mảnh dựa trên các khoảng giá trị của một thuộc tính. Mỗi phân mảnh chứa các hàng có giá trị thuộc tính nằm trong một khoảng xác định.

❖ Cách thực hiện

- Với số phân mảnh là N, khoảng giá trị của Rating (từ 0 đến 5) được chia thành N khoảng đều nhau.
- Ví dụ:
 - + Nếu N = 2: Partition 0 chứa Rating từ [0, 2.5], Partition 1 chứa từ [2.5, 5].
 - + Nếu N=3: Partition 0 chứa Rating từ [0, 1.67], Partition 1 chứa Rating từ [1.67, 3.34], Partition 2 chứa Rating từ [3.34, 5].
- Mỗi phân mảnh được lưu vào một bảng riêng (ví dụ: range_part0, range_part1, ...).

❖ Ưu điểm

- Dễ dàng thực hiện các truy vấn dựa trên phạm vi giá trị (range queries).
- Phù hợp khi dữ liệu có phân bố không đồng đều, ví dụ khi nhiều đánh giá tập trung ở một số mức Rating nhất định.

❖ Nhược điểm

- Có thể dẫn đến mất cân bằng dữ liệu nếu các giá trị Rating không phân bố đồng đều.
- Cần xác định trước các khoảng giá trị phù hợp.

2.2.2 Kỹ thuật Round Robin Partitioning

Round Robin Partitioning phân phối các hàng dữ liệu vào các phân mảnh theo cách tuần hoàn (vòng tròn). Dữ liệu được gán vào các phân mảnh theo thứ tự mà không phụ thuộc vào giá trị của bất kỳ cột nào.

❖ Cách thực hiện

- Với số phân mảnh N , các hàng được phân phối lần lượt vào các bảng (ví dụ: hàng 1 vào partition 0, hàng 2 vào partition 1, ..., hàng $N+1$ lại vào partition 0, và cứ tiếp tục).
- Mỗi phân mảnh được lưu vào một bảng riêng (ví dụ: rr_part0, rr_part1, ...).

❖ Ưu điểm

- Đảm bảo phân phối dữ liệu đồng đều giữa các phân mảnh, bất kể giá trị của dữ liệu.
- Đơn giản để triển khai và quản lý.

❖ Nhược điểm

- Không tối ưu cho các truy vấn dựa trên giá trị của một cột cụ thể, vì dữ liệu được phân bố ngẫu nhiên.
- Có thể làm tăng độ phức tạp khi cần tìm kiếm dữ liệu dựa trên một tiêu chí cụ thể.

2.3 Mục tiêu của phân mảnh

Tối ưu hiệu năng: Giảm thời gian truy vấn bằng cách chỉ xử lý một hoặc một vài phân mảnh thay vì toàn bộ bảng.

Dễ quản lý dữ liệu lớn: Với tập dữ liệu lớn như MovieLens (10 triệu đánh giá), phân mảnh giúp giảm tải cho cơ sở dữ liệu và dễ dàng mở rộng hệ thống.

Hỗ trợ hệ thống phân tán: Các phân mảnh có thể được lưu trữ trên các máy chủ khác nhau, cải thiện khả năng xử lý song song.

Tăng tính sẵn sàng: Nếu một phân mảnh gặp sự cố, các phân mảnh khác vẫn có thể hoạt động.

CHƯƠNG 3. THỬ NGHIỆM

3.1 Triển khai và thử nghiệm

3.1.1 Yêu cầu môi trường

Để triển khai các hàm trong bài tập lớn, môi trường thực hiện cần được cấu hình như sau:

- Hệ điều hành: Sử dụng máy ảo Ubuntu.

```
long@long-ThinkPad-X280:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 23.10
Release:        23.10
Codename:       mantic
long@long-ThinkPad-X280:~$
```

Hình 1 – Hệ điều hành ubuntu

- Python: Phiên bản 3.12.0, đảm bảo tương thích với các thư viện sử dụng.

```
long@long-ThinkPad-X280:~$ python3.12 --version
Python 3.12.0
long@long-ThinkPad-X280:~$
```

Hình 2 – Phiên bản python

- Hệ quản trị cơ sở dữ liệu: PostgreSQL vì tính phổ biến và hỗ trợ tốt cho việc phân mảnh.

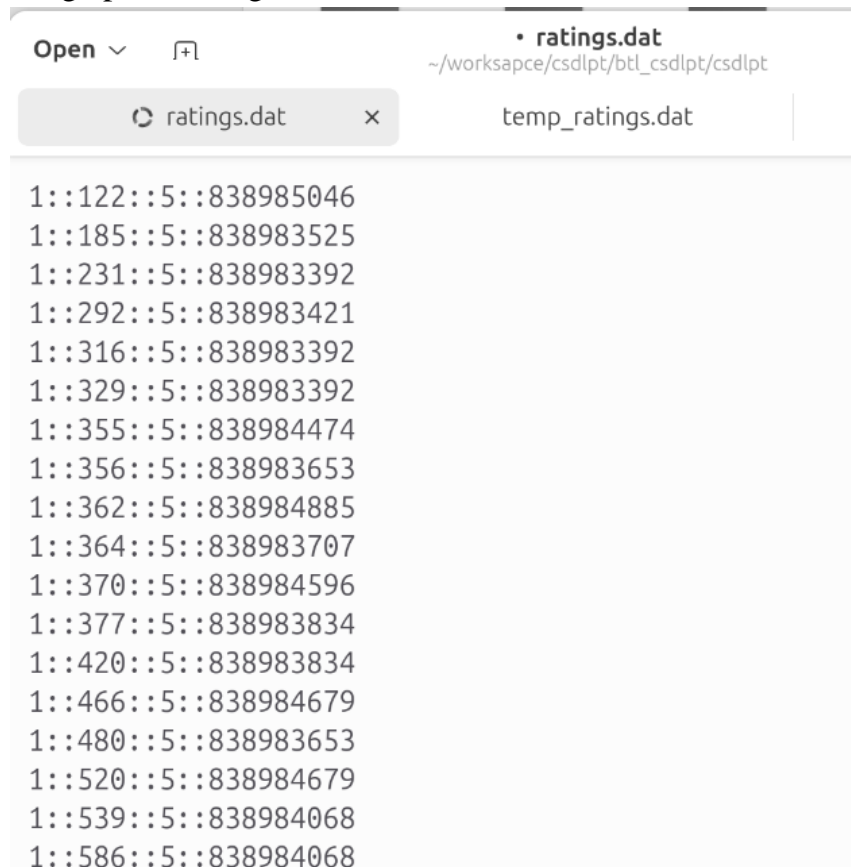
```
long@long-ThinkPad-X280:~$ sudo systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; preset: e>
   Active: active (exited) since Tue 2025-06-10 21:24:14 +07; 14min ago
     Process: 1328 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
    Main PID: 1328 (code=exited, status=0/SUCCESS)
       CPU: 3ms

Jun 10 21:24:14 long-ThinkPad-X280 systemd[1]: Starting postgresql.service - Po>
Jun 10 21:24:14 long-ThinkPad-X280 systemd[1]: Finished postgresql.service - Po>
lines 1-9/9 (END)
[3]+  Stopped                  sudo systemctl status postgresql
```

Hình 3 – Hệ quản trị cơ sở dữ liệu PostgreSQL

- Thư viện python:
 - + psycopg2 (cho PostgreSQL) để kết nối và thao tác với cơ sở dữ liệu.
 - + Không yêu cầu thêm thư viện bên ngoài để xử lý tệp ratings.dat.
- Tệp dữ liệu đầu vào: Tệp ratings.dat được tải từ trang MovieLens, là một tệp data thô có tên là rating.dat gồm các thông tin như Stt, userID, userName, rating, mục

đích là để nạp vào database và tiến hành phân mảnh theo 2 thuật toán roundrobin partitioning, range partitioning.



Hình 4 – Tập đầu vào ratings.dat

3.1.2 Hàm đọc dữ liệu LoadRatings

```

def loadratings(ratingtablename, ratingsfilepath, openconnection):
    try:
        cur = openconnection.cursor()
        cur.execute("DROP TABLE IF EXISTS " + ratingtablename)
        cur.execute("""
            CREATE TABLE """ + ratingtablename + """ (
                UserID INTEGER,
                MovieID INTEGER,
                Rating FLOAT
            )
        """)

        # Tạo file tạm thời với định dạng phù hợp
        temp_file = 'temp_ratings.dat'
        with open(ratingsfilepath, 'r', encoding='utf-8') as infile, open(temp_file, 'w', encoding='utf-8') as outfile:
            for line in infile:
                parts = line.strip().split('::')
                if len(parts) == 4:
                    outfile.write(f"{parts[0]}\t{parts[1]}\t{parts[2]}\n")

        # Sử dụng copy_from để nạp dữ liệu
        with open(temp_file, 'r', encoding='utf-8') as f:
            cur.copy_from(f, ratingtablename, sep='\t', null='')
        openconnection.commit()
        print("Data loaded successfully into " + ratingtablename)

        # Xóa file tạm thời
        os.remove(temp_file)
    except psycopg2.Error as e:
        print("Error loading ratings: " + str(e))
        openconnection.rollback()
    finally:
        cur.close()

```

Hình 5 – Hàm LoadRatings

❖ Giải thích:

Hàm loadratings đọc dữ liệu từ tệp *ratings.dat* và đưa vào bảng trong PostgreSQL:

Tham số: ratingtablename là tên bảng, ratingsfilepath là đường dẫn trỏ đến tệp ratings.dat, openconnection là đối tượng kết nối đến PostgreSQL.

Xử lý lỗi: Dùng try-except để bắt lỗi psycopg2.Error, Nếu xảy ra lỗi liên quan đến cơ sở dữ liệu thì in thông báo lỗi và gọi rollback() để hủy các thay đổi chưa được commit.

Tạo bảng: Xóa bảng cũ nếu tồn tại DROP TABLE IF EXISTS, tạo bảng mới với schema UserID (INTEGER), MovieID (INTEGER), Rating (FLOAT).

Xử lý tệp: Đọc ratings.dat, tách dòng bằng ::, lấy UserID, MovieID, Rating, ghi vào tệp tạm (temp_ratings.dat) với dấu phân cách \t.

Nạp dữ liệu: Dùng cur.copy_from để nạp từ tệp tạm vào bảng, commit giao dịch.

Thực hiện xong: Xóa tệp tạm, đóng con trỏ trong finally.

❖ Ưu điểm:

- Xử lý lỗi tốt: Bắt lỗi cơ sở dữ liệu bằng việc sử dụng try-except, in thông báo chi tiết.
- Kiểm tra bảng: DROP TABLE IF EXISTS thực hiện xóa bảng nếu bảng đã tồn tại để tránh lỗi bảng đã tồn tại.
- Hỗ trợ mã hóa: Dùng utf-8 tránh lỗi ký tự đặc biệt.

- Đóng tài nguyên: Khởi finally luôn đảm bảo con trỏ luôn được đóng.
- Hiệu quả nạp dữ liệu: Sử dụng copy_from để nạp nhanh dữ liệu lớn thay vì chèn từng dòng bằng lệnh INSERT.

3.1.3 Hàm phân mảnh ngang theo phạm vi của rating

Tiến hành triển khai phân mảnh ngang bảng *ratings* thành *n* mảnh dựa trên điều kiện độ lớn của thuộc tính rating. Mỗi mảnh được phân sẽ tạo một bảng mới có tiền tố *range_part + i*, với *i* là số thứ tự mảnh. Theo quy ước, phạm vi phân mảnh (khoảng giá trị rating) sẽ được chia đều cho mỗi mảnh.

Tham số đầu vào:

- ratingtablename: Tên bảng gốc (trong bài toán sẽ tên là ratings).
- numberofpartitions: Số lượng mảnh cần chia.
- openconnection: Kết nối đang sử dụng để thao tác với database.

Các bước triển khai phân mảnh như sau:

Tìm khoảng giá trị phân mảnh: Trước tiên cần tính độ dài khoảng phân mảnh để làm cơ sở xác định được khoảng giá trị để chia các tuple cho các mảnh. Do giá trị rating chỉ có phạm vi từ 0 đến 5, nên ta dễ dàng xác định độ dài khoảng:

$$d = 5 / <số\ lượng\ mảnh\ cần\ chia>$$

Tạo phân mảnh (tạo bảng) và chèn dữ liệu gốc: Với mỗi mảnh (0 đến numberofpartitions - 1), dựa vào thứ tự mảnh và độ dài *d* đã tìm được, ta xác định được tên phân mảnh (theo quy ước đã nêu ở trên) và khoảng giá trị dùng để lọc lấy các tuple cho mảnh này (min_rate, max_rate). Cuối cùng, tạo một phân mảnh mới với dữ liệu đến từ bảng gốc ratings sau khi đã lọc lấy các hàng phù hợp.

```

def rangepartition(ratingtablename, numberofpartitions, openconnection):
    """
    Based on range of ratings, create new partitions from main table (ratings)
    """
    try:
        cur = openconnection.cursor()

        # Tính phạm vi rating cho mỗi mảnh
        d = 5 / numberofpartitions

        # Với mỗi mảnh thứ i tạo bảng mới có tiền tố range_part + i,
        # Lấy dữ liệu từ bảng rating gốc thêm vào các mảnh
        for i in range(numberofpartitions):
            tb_name = f'range_part{i}'
            min_rate = i * d
            max_rate = min_rate + d
            if i == 0:
                cur.execute(
                    f'CREATE TABLE {tb_name} AS '
                    f'SELECT userid, movieid, rating FROM {ratingtablename} '
                    f'WHERE rating <= {max_rate};'
                )
            else:
                cur.execute(
                    f'CREATE TABLE {tb_name} AS '
                    f'SELECT userid, movieid, rating FROM {ratingtablename} '
                    f'WHERE rating > {min_rate} and rating <= {max_rate};'
                )
            cur.close()
            openconnection.commit()
    except Exception as ex:
        openconnection.rollback()
        print(f'Phân mảng ngang theo khoảng thất bại: {str(ex)}')

```

Hình 6 - Hàm phân mảnh ngang theo phạm vi

- ⇒ Nhận xét: Triển khai gộp 2 thao tác tạo bảng mới và chèn dữ liệu vào một câu lệnh truy vấn duy nhất giúp giảm đáng kể thời gian thực thi. Ngoài ra, toàn bộ quá trình phân mảnh được bọc trong khối try catch giúp khi xảy ra lỗi phân mảnh, có thể ngay lập tức rollback database về trạng thái cuối trước khi thay đổi, như vậy sẽ luôn đảm bảo tính toàn vẹn dữ liệu ngay cả khi phân mảnh bị lỗi.

Kết quả chạy phân mảnh:

```

PS C:\Users\NGUYEN\Desktop\CSDLPT\BTL\clone\csdlpt> & C:/Users/NGUYEN/Desktop/CSDLPT/BTL/clone/csdlpt/BTL_N12.py
Xóa dữ liệu và tạo mới database ...
Thời gian thực thi: 0.307s

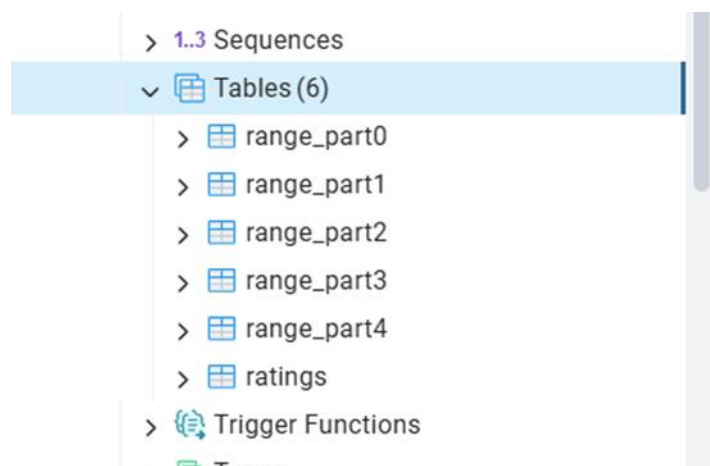
Khởi tạo dữ liệu bảng ratings ...
Data loaded successfully into ratings
Thời gian thực thi: 8.970s

Phân mảnh ngang theo khoảng ...
Thời gian thực thi: 8.476s

```

Hình 7 – Kết quả chạy phân mảnh (range partition)

Cơ sở dữ liệu sau phân mảnh:



Hình 8 – Cơ sở dữ liệu sau phân mảnh (range partition)

Kết quả kiểm tra thông qua tester:

```
CSDLPT/BTL/clone/csd1pt/Assignment1Tester.py
A database named "dds_assgn1" already exists
Data loaded successfully into ratings
loadratings function pass!
rangepartition function pass!
```

Hình 9 – Kết quả kiểm tra thông qua tester (range partition)

3.1.4 Hàm chèn tuple dữ liệu mới vào phân mảnh ngang

Hàm xử lý xác định phân mảnh và chèn dữ liệu mới vào phân mảnh đó dựa trên giá trị rating

Tham số đầu vào:

- ratingtablename: Tên bảng gốc (trong bài toán sẽ tên là ratings)
- userid: Mã định danh người đánh giá
- movieid: Mã định danh bộ phim được đánh giá
- rating: Mức độ đánh giá
- openconnection: Kết nối đang sử dụng để thao tác với database

Các bước triển khai như sau:

Xác định khoảng phân mảnh phù hợp: Vì số lượng mảnh là không biết trước, chúng ta cần đếm thủ công các bảng có tiền tố range_part nhằm xác định số lượng mảnh hiện tại. Sau khi có được số lượng mảnh, tiến hành tính khoảng d theo công thức đã được trình bày ở phần 3.1.3. Từ đó sử dụng phép chia nguyên và chia dư, xác định được thứ tự mảnh phù hợp.

Chèn dữ liệu: Sử dụng thứ tự mảnh i tìm được, xác định tên phân mảnh và chèn dữ liệu mới.

```
def rangeinsert(ratingtablename, userid, movieid, rating, openconnection):
    try:
        cur = openconnection.cursor()

        # Tìm số lượng mảnh thông qua đếm số lượng bảng có tiền tố là range_part
        cur.execute(f"SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE \'range_part%\';")
        number_of_partitions = cur.fetchone()[0]

        # Dựa vào giá trị rating của bản ghi mới, tìm được số thứ tự mảnh phù hợp
        # Từ số thứ tự, tìm được tên bảng rồi chèn bản ghi vào như thông thường
        d = 5 / number_of_partitions
        i = int(rating / d)
        if rating % d == 0 and i != 0:
            i = i - 1
        tb_name = f'range_part{i}'
        cur.execute(f"INSERT INTO {tb_name} (userid, movieid, rating) values ({userid}, {movieid}, {rating})")

        cur.close()
        openconnection.commit()
    except Exception as ex:
        openconnection.rollback()
        print(f'Chèn dữ liệu vào phân mảnh ngang theo khoảng thất bại: {str(ex)}')
```

Hình 10 - Hàm chèn dữ liệu vào phân mảnh ngang theo phạm vi

⇒ Nhận xét: Xử lý các ngoại lệ xảy ra và rollback khi có lỗi giúp đảm bảo tính toàn vẹn dữ liệu.

Kết quả chạy và kiểm tra bằng tester:

```
Chèn dữ liệu vào phân mảnh ngang theo khoảng ...
Thời gian thực thi: 0.005s
```

```
rangeinsert function pass!
```

Hình 11 – Kết quả chạy và kiểm tra bằng tester (range partition)

3.1.5 Xây dựng hàm roundrobinpartition

Ta định nghĩa hàm roundrobinpartition() nhận vào các tham số sau:

- ratingtablename: tên bảng gốc chứa ratings.
- numberofpartitions: số lượng phân mảnh muốn tạo.
- openconnection: kết nối đến CSDL PostgreSQL (đã mở)

```

def roundrobinpartition(ratingtablename, numberofpartitions, openconnection):
    """
    Based on round-robin distribution, create new partitions from main table (ratings).
    """
    try:
        cur = openconnection.cursor()
        prefix = 'rrobin_part'

        # Tạo và điền dữ liệu vào các bảng phân mảnh
        for i in range(numberofpartitions):
            tb_name = f'{prefix}{i}'
            cur.execute(f"""
                CREATE TABLE {tb_name} AS
                SELECT userid, movieid, rating
                FROM (
                    SELECT userid, movieid, rating,
                           ROW_NUMBER() OVER () - 1 AS rnum
                    FROM {ratingtablename}
                ) AS temp
                WHERE MOD(rnum, {numberofpartitions}) = {i}
            """)
        cur.close()
        openconnection.commit()
        print(f"Phân mảnh round-robin hoàn thành với {numberofpartitions} bảng.")
    except Exception as ex:
        openconnection.rollback()
        print(f'Phân mảnh ngang theo round-robin thất bại: {str(ex)}')

```

Hình 12 – Hàm phân mảnh ngang theo ghép cặp xoay vòng

Tạo biến con lấy kết nối từ openconnection, con trở cur = con.cursor() để thực hiện các truy vấn SQL.

Bao bọc bên ngoài là hàm try với mục đích rollback() lại dữ liệu nếu như không thực hiện hành động như là connect,select,tạo con trở thì sẽ trả lại trạng thái cũ

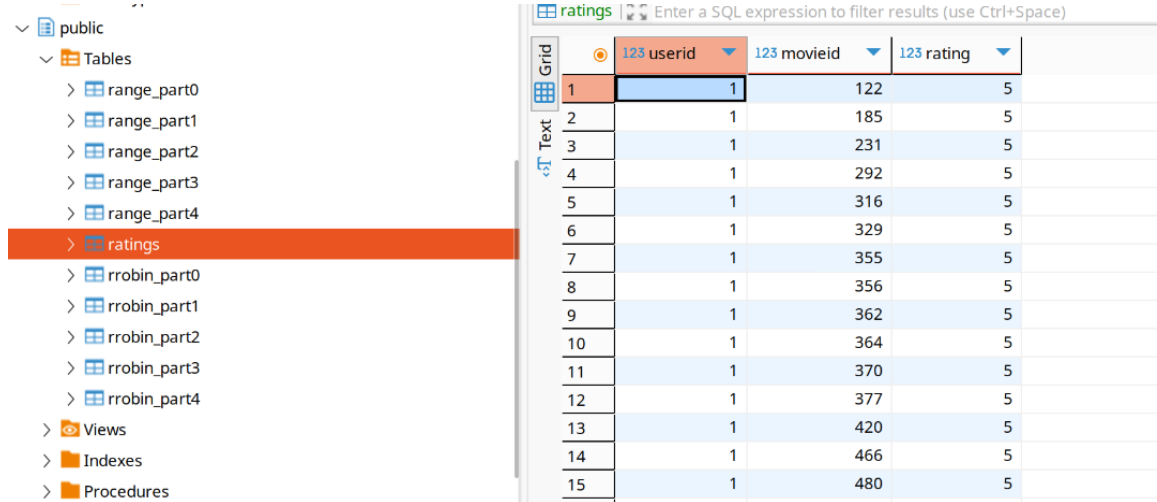
Ta sẽ duyệt qua vòng for nhằm mục đích để tạo các phân mảnh (for i in range(numberofpartitions)) và gán cho table được tạo ra và gán cho giá trị prefix là “rrobin_part” (tb_name = f’{prefix}{i}’)

Tiếp theo là câu lệnh truy vấn tạo bảng : CREATE TABLE {tb_name} ; SELECT userid,movieId,rating : chọn các trường userId,movideId, Rating; FROM (SELECT userid,movieid,rating, Row_Number() - 1 as rnum : câu lệnh này là lấy tất cả các trường như trong bảng được loading nhưng bổ sung thêm trường rnum đây là trường stt - 1; phục vụ mục đích tính toán phân mảnh. “where mod(rnum,numberofpartition)) = {i}” cái này là đoạn giúp phân mảnh dữ liệu theo modul. từng giá trị được xếp vào từng bảng theo nguyên tắc xoay vòng. Ví dụ :

- Với số thứ tự là 0 => sẽ gán với phân mảnh số 0 (0 % 3 = 0)
- Với số thứ tự là 1 => sẽ gán với phân mảnh số 1 (1 % 3 = 1)
- Với số thứ tự là 2 => sẽ gán với phân mảnh số 2 (2 % 3 = 2)
- Với số thứ tự là 3 => sẽ gán với phân mảnh số 0 (3 % 3 = 0)
- Với số thứ tự là 4 => sẽ gán với phân mảnh số 1 (4 % 3 = 1)

Nếu thành công thì sẽ in ra thông báo “phân mảnh hoàn thành với n bảng”

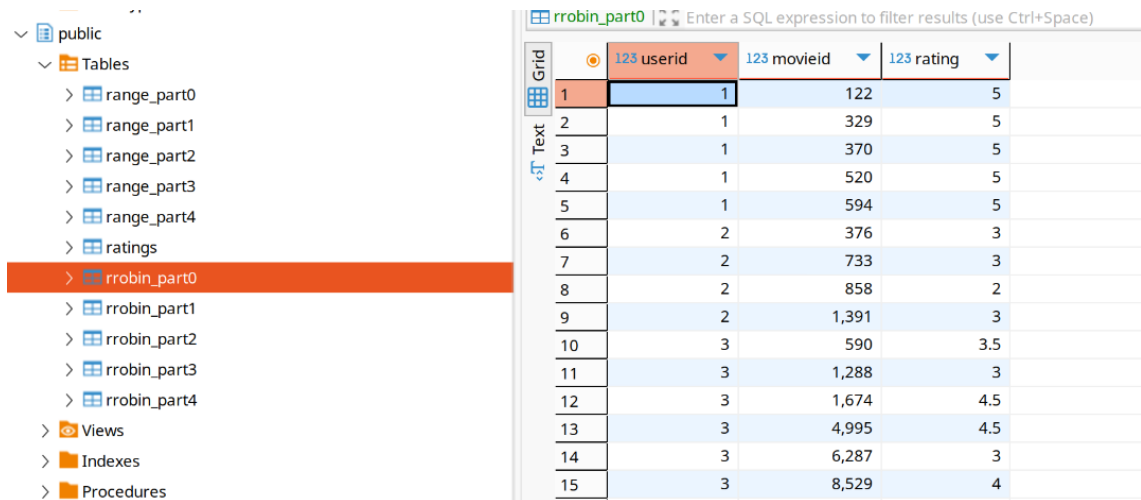
Kết quả sau khi phân mảnh: ở đây là $n = 5$; tức là có 5 site; lần lượt theo thứ tự là : rrobin_part0, rrobin_part1, rrobin_part2, rrobin_part3, rrobin_part4,



	123 userid	123 movieid	123 rating
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

Hình 13 – Kết quả phân mảnh (round robin partition)

Sau khi phân mảnh rồi hàng thứ 1 gồm userid =1; movieid = 122; rating = 5 sẽ được cho vào phân mảnh 1; hàng thứ 6 với userid =1; moviedid = 329 với rating là 5 cũng vào phân mảnh 1



	123 userid	123 movieid	123 rating
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	1,391	3
10	3	590	3.5
11	3	1,288	3
12	3	1,674	4.5
13	3	4,995	4.5
14	3	6,287	3
15	3	8,529	4

Hình 14 – Kết quả phân mảnh số 1 (round robin partition)

Sau khi phân mảnh rồi hàng thứ 2 gồm userid =1 ; movieid = 185; rating = 5 sẽ được cho vào phân mảnh 1; hàng thứ 7 với userid = 1; moviedid = 355 với rating là 5 cũng vào phân mảnh 2;

	userid	movieid	rating
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	1,049	3
9	2	1,544	3
10	3	1,148	4
11	3	1,408	3.5
12	3	3,408	4
13	3	5,299	3
14	3	6,377	4
15	3	8,533	4.5

Hình 15 – Kết quả phân mảnh số 2 (round robin partition)

Sau khi phân mảnh rồi hàng thứ 3 gồm userid =1 ; movieid =231 ; rating = 5 sẽ được cho vào phân mảnh 1; hàng thứ 8 với userid = 1; moviedid = 356 với rating là 5 cũng vào phân mảnh 3:

	userid	movieid	rating
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	1,073	3
9	3	110	4.5
10	3	1,246	4
11	3	1,552	2
12	3	3,684	4.5
13	3	5,505	2
14	3	6,539	5
15	3	8,783	5

Hình 16 - Kết quả phân mảnh số 3 (round robin partition)

Sau khi phân mảnh rồi hàng thứ 4 gồm userid =1 ; movieid =292 ; rating = 5 sẽ được cho vào phân mảnh 1; hàng thứ 9 với userid = 1; moviedid = 356 với rating là 5 cũng vào phân mảnh 4:

	userid	movieid	rating
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	1,210	4
9	3	151	4.5
10	3	1,252	4
11	3	1,564	4.5
12	3	4,535	4
13	3	5,527	4.5
14	3	7,153	4
15	3	27,821	4.5

Hình 17 - Kết quả phân mảnh số 4 (round robin partition)

Sau khi phân mảnh rồi hàng thứ 5 gồm $userid = 1$; $movieid = 316$; $rating = 5$ sẽ được cho vào phân mảnh 1; hàng thứ 10 với $userid = 1$; $movieid = 364$ với $rating$ là 5 cũng vào phân mảnh 5:

	userid	movieid	rating
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	1,356	3
9	3	213	5
10	3	1,276	3.5
11	3	1,597	4.5
12	3	4,677	4
13	3	5,952	3.5
14	3	7,155	3.5
15	3	33,750	3.5

Hình 18 - Kết quả phân mảnh số 5 (round robin partition)

3.1.6 Xây dựng hàm RoundRobin_insert

```

f roundrobininsert(ratingtablename, userid, movieid, rating, openconnection):
    """
    Insert a new record into the appropriate round-robin partition table.
    """
    try:
        cur = openconnection.cursor()
        prefix = 'rrobin_part'

        # Đếm số lượng bảng phân mảnh
        cur.execute("SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE %s", (f'{prefix}%',))
        number_of_partitions = cur.fetchone()[0]
        if number_of_partitions == 0:
            raise Exception("Không tìm thấy bảng phân mảnh round-robin.")

        # Xác định bảng đích dựa trên tổng số bản ghi hiện tại
        cur.execute("insert into " + ratingtablename + "(userid, movieid, rating) values (" + str(userid) + "," + str(
            cur.execute("SELECT COUNT (*) FROM " + ratingtablename + ";");
            total_rows = (cur.fetchall())[0][0];
            partition_index = (total_rows - 1) % number_of_partitions
            tb_name = f'{prefix}{partition_index}'

        # Chèn dữ liệu vào bảng phân mảnh
        cur.execute(f"""
            INSERT INTO {tb_name} (userid, movieid, rating)
            VALUES (%s, %s, %s)
            """, (userid, movieid, rating))

        cur.close()
        openconnection.commit()
        print(f"Chèn dữ liệu vào {tb_name} thành công.")
    except Exception as ex:
        openconnection.rollback()
        print(f"Chèn dữ liệu vào phân mảnh ngang theo round-robin thất bại: {str(ex)}")

```

Hình 19 – Hàm round robin insert

Ta định nghĩa hàm roundrobininsert(...) nhận vào các tham số sau:

- ratingtablename: tên bảng gốc chứa ratings.
- userid, itemid, rating: tương ứng các giá trị được truyền vào để inser vào các mảnh
- openconnection: kết nối đến CSDL PostgreSQL (đã mở).

Tạo biến con lấy kết nối từ openconnection, con trả cur = con.cursor() để thực hiện các truy vấn SQL.

Đặt tiền tố cho phân mảnh bảng: RROBIN_TABLE_PREFIX = 'rrobin_part'.

```

cur = openconnection.cursor()
prefix = 'rrobin_part'

```

Hình 20 – Kết nối database

Đếm số lượng phân mảnh : “pg_stat_user_tables” là view hệ thống của postgres giúp xem được số lượng các bảng:

```

# Đếm số lượng bảng phân mảnh
cur.execute("SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE %s", (f'{prefix}%',))
number_of_partitions = cur.fetchone()[0]
if number_of_partitions == 0:
    raise Exception("Không tìm thấy bảng phân mảnh round-robin.")

```

Hình 21 – Đếm số lượng phân mảnh

Xác định xem dựa trên phân mảnh nào dựa vào số thứ tự của nó trong trang:

```
# Xác định bảng đích dựa trên (parameter) ratingtablename: Any
cur.execute("insert into " + ratingtablename + "(userid, movieid, rating) values (" + str(userid) + "," + str
cur.execute("SELECT COUNT (*) FROM " + ratingtablename + ";");
total_rows = (cur.fetchall())[0][0];
partition_index = (total_rows - 1) % number_of_partitions
tb_name = f'{prefix}{partition_index}'
```

Hình 22 – Thêm dữ liệu vào phân mảnh

Giải thích: Phương pháp tận dụng chỉ số bắt đầu từ 0. Vị trí cuối cùng của tập dữ liệu tương ứng với tổng số dòng hiện tại, đồng thời là chỉ số của phần tử tiếp theo cần chèn. Ví dụ, nếu dòng cuối có chỉ số 99, tổng số dòng là 100, và bản ghi mới sẽ được chèn tại vị trí 100. Do đó index sẽ là vị trí thứ $(101 - 1) = 100$.

Kết quả thu được sau khi chạy:

Phân vùng cho thấy đang ở rrobin_part4 vì giá trị của các hàng vốn là 1000004 sau khi thêm nó sẽ là bội số chia hết cho 5 và sẽ ở site 5 ứng với rrobin_part4

```
○ (venv) long@long-ThinkPad-X280:~/worksapce/csd1pt/btl_csd1pt/csd1p
A database named "postgres" already exists
Data loaded successfully into ratings
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
Data loaded successfully into ratings
Phân mảnh round-robin hoàn thành với 5 bảng.
roundrobinpartition function pass!
Chèn dữ liệu vào rrobin_part4 thành công.
roundrobininsert function pass!
Press enter to Delete all tables? █
```

Hình 23 – Kết quả chạy với terminal

Kiểm tra trên database thứ 4 với bảng rrobin_part4 với user id = 100 movie 1 rating 3 được xuất hiện cuối bảng ứng với bộ test trên:

```
# ALERT:: Change the partition index according to your testing sequence.
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '4')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '1')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
if result :
```

Hình 24 – Bộ test kiểm tra

	123 userid	123 movieid	123 rating
18	100	1,271	0.5
19	100	1,387	3.5
20	100	1,977	1.5
21	100	2,028	3
22	100	2,122	1
23	100	2,186	3
24	100	2,302	1.5
25	100	2,531	1
26	100	2,571	2.5
27	100	2,699	2.5
28	100	2,929	1
29	100	3,015	1
30	100	3,255	1.5
31	100	3,556	3
32	100	3,893	2.5
33	100	4,069	0.5
34	100	4,508	3
35	100	4,936	1
36	100	5,378	2
37	100	6,537	2.5
38	100	7,386	2
39	100	31,770	3
40	100	1	3

Hình 25 – Kết quả sau khi chạy

Logic hàm kiểm tra:

```
[result, e] = testHelper.testroundrobinpartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN
if result :
    print("roundrobinpartition function pass!")
else:
    print("roundrobinpartition function fail!")

# ALERT:: Change the partition index according to your testing sequence.
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '4')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '1')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
```

Hình 26 – Gọi hàm kiểm tra

Gọi hai hàm test; tại đây sẽ thực hiện test và hiển thị kết quả hai hàm test này được gọi từ file TestHepler với hàm testroundrobinpartition sẽ lấy các tham số trên để thực hiện chia partition MyAsssingemnet.roundrobinpartition; trong đó MyAsssignment chính là interface để triển khai các hàm nói trên sau đó thực hiện mapping các hàm test và hiển thị kết quả:

```

def testroundrobinpartition(MyAssignment, ratingtablename, numberofpartitions, openconnection,
    partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    """
    Tests the round robin partitioning for Completeness, Disjointness and Reconstruction
    :param ratingtablename: Argument for function to be tested
    :param numberofpartitions: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param robinpartitiontableprefix: This function assumes that you tables are named in an order. Eg: robinpart1, robinpart2, ...
    :return: Raises exception if any test fails
    """
    try:
        MyAssignment.roundrobinpartition(ratingtablename, numberofpartitions, openconnection)
        testrangeandrobinpartitioning(numberofpartitions, openconnection, RROBIN_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRoundrobinPartition(ratingtablename, numberofpartitions, openconnection, RROBIN_TABLE_PREFIX)
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]

```

Hình 27 – Hàm kiểm tra round robin partition

Logic này tương tự cho hàm insert:

```

def testroundrobininsert(MyAssignment, ratingtablename, userid, itemid, rating, openconnection, expectedtableindex):
    """
    Tests the roundrobin insert function by checking whether the tuple is inserted in the Expected table you provide
    :param ratingtablename: Argument for function to be tested
    :param userid: Argument for function to be tested
    :param itemid: Argument for function to be tested
    :param rating: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param expectedtableindex: The expected table to which the record has to be saved
    :return: Raises exception if any test fails
    """
    try:
        expectedtablename = RROBIN_TABLE_PREFIX + expectedtableindex
        MyAssignment.roundrobininsert(ratingtablename, userid, itemid, rating, openconnection)
        if not teststrangerrobininsert(expectedtablename, itemid, openconnection, rating, userid):
            raise Exception(
                'Round robin insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format(userid, itemid, rating, expectedtablename)
            )
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]

```

Hình 28 – Hàm kiểm tra insert