HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN HỌC: CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên :TS Kim Ngọc Bách

Nhóm :11

Lớp : CSDLPT nhóm 9

Các thành viên : Đinh Quốc Đại - B22DCCN175

Hoàng Văn Linh- B22DCCN487

Nguyễn Thị Như Quỳnh-B22DCCN679

Hà Nội - 6/2025

MỤC LỤC

LỜI CẢM ƠN

Đầu tiên, chúng em xin gửi lời cảm ơn sâu sắc đến Học viện nghệ Bưu chính Viễn thông và khoa CNTT1 đã đưa môn học Cơ sở dữ liệu phân tán vào trong chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ tán Kim Ngọc Bách đã hướng dẫn và truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua.

Trong thời gian làm việc với thầy, chúng em đã được tiếp thu thêm nhiều kiến thức bổ ích, học tập được tinh thần làm việc hiệu quả, nghiêm túc. Đây thực là những điều rất cần thiết cho quá trình học tập và công tác sau này của chúng em. Dưới sự dẫn dắt tận tình của thầy, chúng em hiểu rõ những nguyên lý cốt lõi của một hệ thống cơ sở dữ liệu phân tán như: phân mảnh dữ liệu, phân phối truy vấn, đảm bảo tính nhất quán và tính sẵn sàng của dữ liệu trong môi trường hệ thống phân tán. Đặc biệt, thông qua bài tập lớn và hướng dẫn triển khai chi tiết, chúng em đã có cơ hội vận dụng kiến thức để xây dựng các giải pháp kỹ thuật như phân mảnh range, round-robin và xử lý chèn dữ liệu đúng vào từng phân vùng tương ứng – những nội dung có tính ứng dụng cao trong thực tiễn phát triển hệ thống.. Điều này không chỉ nâng cao khả năng vận dụng mà còn giúp chúng em thêm tự tin khi bước ra môi trường làm việc thực tế.

Một lần nữa, chúng em xin gửi tới thầy lời cảm ơn sâu sắc nhất vì tất cả những gì thầy đã truyền đạt và dìu dắt chúng em trong suốt học kỳ vừa qua. Kính chúc thầy luôn mạnh khỏe, hạnh phúc, gặt hái được nhiều thành công trong sự nghiệp giảng dạy cũng như trong cuộc sống. Mong rằng trong tương lai, chúng em sẽ tiếp tục được học hỏi thầy nhiều hơn nữa!

I. GIỚI THIỆU TỔNG QUAN VỀ BÀI TOÁN

1. Mô tả bài toán

Bài toán tập trung vào việc nghiên cứu và triển khai các phương pháp **phân chia dữ liệu ngang (horizontal fragmentation)** trong hệ quản trị cơ sở dữ liệu mã nguồn mở, kết hợp với lập trình xử lý bằng **ngôn ngữ Python.**

Dữ liệu sử dụng trong bài toán là tập đánh giá phim từ bộ dữ liệu MovieLens, cụ thể là tệp ratings.dat, bao gồm các thông tin: UserID, MovieID, Rating, Timestamp.

2. Mục tiêu bài toán

- Hiểu và vận dụng kiến thức lý thuyết về phân mảnh dữ liệu ngang trong hệ thống cơ sở dữ liệu phân tán.
- Thực hành xử lý dữ liệu lớn, tổ chức lưu trữ hiệu quả bằng cách phân vùng phù hợp.
- Ứng dụng thành thạo MySQL/PostgreSQL kết hợp với Python trong xử lý dữ liệu, chia phân vùng, và thực hiện các thao tác chèn dữ liệu vào phân mảnh phù hợp.
- Xây dựng nền tảng kỹ thuật để triển khai các hệ thống xử lý phân tán trong thực tế.

3. Phạm vi và giới hạn bài toán

a, Phạm vi của bài toán

- **Dữ liệu** : Sử dụng tập dữ liệu đánh giá phim từ tệp rating.dat của MovieLens.
- **Môi trường** : Phát triển bằng Python trên hệ điều hành Ubuntu hoặc Windows, tích hợp với cơ sở dữ liệu (PostgreSQL hoặc MySQL).
- **Nhiệm vụ** :
    - Tải dữ liệu vào cơ sở dữ liệu bảng.
    - Thực hiện phân mảnh theo hai phương pháp: dựa trên giá trị khoảng và vòng tròn.
    - Xây dựng các chức năng chèn mới dữ liệu vào phân vùng phù hợp.
- **Đánh giá** : Kiểm tra tính chính xác của testcase tự động và cơ sở dữ liệu bảng nội dung.

b, Giới hạn của bài toán

- **Dữ liệu** : Chỉ sử dụng dữ liệu tệp được cung cấp, không sửa đổi hoặc mã hóa thông tin cứng.

- **Cơ sở dữ liệu** : Cơ sở dữ liệu tên cứng hoặc kết nối không được mã hóa; Phân mảnh bảng tên phải được đính kèm theo định dạng quy định.
- **Triển khai** : Không sử dụng toàn cục; các chức năng chèn phải đảm bảo tính nhất quán khi gọi nhiều lần.
- **Kiểm tra** : Testcase chỉ kiểm tra trình biên dịch lỗi, cần kiểm tra nội dung bảng để đảm bảo tính chính xác.

II.TỔNG QUAN LÝ THUYẾT

1. Phân mảnh ngang

- Phân mảnh cơ sở dữ liệu là quá trình lưu trữ một cơ sở dữ liệu lớn trên nhiều máy. Mỗi một máy hoặc máy chủ cơ sở dữ liệu chỉ có thể lưu trữ và xử lý một lượng dữ liệu giới hạn. Quá trình phân mảnh cơ sở dữ liệu khắc phục hạn chế này bằng cách tách dữ liệu thành các đoạn nhỏ hơn, được gọi là các phân mảnh và lưu trữ chúng trên một số máy chủ cơ sở dữ liệu. Tất cả máy chủ cơ sở dữ liệu thường có cùng công nghệ cơ sở, đồng thời, những máy chủ này hoạt động cùng nhau để lưu trữ và xử lý lượng dữ liệu lớn.
- Phân mảnh ngang (phân vùng ngang) là một kỹ thuật trong cơ sở dữ liệu quản trị, trong đó các hàng (bản ghi) của một bảng được chia thành các tập tin (phân mảnh) dựa trên một công cụ tiêu chuẩn. Mỗi mảnh chứa một tập hợp các bản ghi mãn tiêu chí nhưng vẫn giữ nguyên cấu trúc (lược đồ) của bảng gốc.Mục tiêu cải thiện hiệu suất truy vấn, quản lý dữ liệu dễ dàng hơn và hỗ trợ mở rộng trong phân tích hệ thống.
- Có 2 loại phân mảnh ngang:
    + Phân mảnh ngang nguyên thủy : là phân mảnh ngang được thực hiện trên các vị từ của chính quan hệ đó.
    + Phân mảnh ngang dẫn xuất : là phân mảnh một quan hệ dựa trên các vị từ của quan hệ khác.

2. Phân mảnh vòng tròn

- Dữ liệu được phân phối tuần tự theo các phân mảnh thứ tự, không dựa trên giá trị của thuộc tính. Mỗi bản ghi được phân bổ vào phân mảnh tiếp theo trong chu kỳ (0, 1, 2, …, N-1, rồi quay lại 0).
- Đảm bảo phân phối đều số lượng giữa các phân mảnh.
- Không phụ thuộc vào giá trị thuộc tính, phù hợp khi dữ liệu không có tính chất phân cụm

III. CÀI ĐẶT MÔI TRƯỜNG VÀ CÁC BƯỚC CHUẨN BỊ

1. Cài đặt môi trường.

1.1. Ngôn ngữ python.

- Python 3.12.x: Sử dụng phiên bản Python 3.12.x để đảm bảo tương thích với các thư viện cần thiết và mã nguồn được cung cấp.

- Cài đặt:

  + Sử dụng lệnh sudo apt install python3.12 trên Ubuntu.
  + Kiểm tra phiên bản Python bằng lệnh: python3 --version hoặc python --version

1.2. Hệ điều hành.

- Hệ điều hành: Ubuntu (phiên bản 24.04) .

1. 3. Hệ quản trị cơ sở dữ liệu.

- PostgreSQL:

+Cài trên Ubuntu bằng lệnh : *sudo apt install postgresql postgresql-contrib.*

+Đảm bảo dịch vụ PostgreSQL đang chạy : *sudo service postgresql start.*

- Cài đặt thư viện Python để kết nối cơ sở dữ liệu: *pip install psycopg2-binary*.

1.4. Thư viện hỗ trợ.

-Hỗ trợ kết nối Python với PostgreSQL **:** *psycopg2-binary.*

2. Các bước chuẩn bị .

2.1. Tải dữ liệu đầu vào.

- Truy cập liên kết *http://files.grouplens.org/datasets/movielens/ml-10m.zip* để tải tệp *ml-10m.zip*.
- Giải nén tệp để lấy *ratings.dat,* chứa 10 triệu đánh giá phim với định dạng **UserID::MovieID::Rating::Timestamp**.

2.2.Tạo cơ sở dữ liệu.

- Đăng nhập vào PostgreSQL: *psql -U postgres.*
- Tạo cơ sở dữ liệu: *CREATE DATABASE dds_assgn1*

2.3.Kiểm tra môi trường:

- Xác minh Python 3.12.x hoạt động: python3 --version.
- Kiểm tra kết nối cơ sở dữ liệu: Viết đoạn mã Python thử nghiệm sử dụng psycopg2 để kết nối và tạo bảng.
- Đảm bảo tệp ratings.dat có thể đọc được bằng Python

2.4.Thiết lập github repository:

- Tạo một repository công khai trên GitHub cho nhóm : https://github.com/NhuQuynh0920/BTL-N-11-CSDLPT
- Thêm các tệp mã nguồn Python và báo cáo vào repository.

2.5. Chuẩn bị dữ liệu kiểm tra:

- Sử dụng tệp kiểm tra (test_data) để thử nghiệm các hàm trước khi chạy trên toàn bộ ratings.dat.

2.6.Kiểm tra ràng buộc:

* Đảm bảo mã nguồn tuân thủ các yêu cầu:

  - Không mã hóa cứng tên cơ sở dữ liệu, thông tin kết nối, hoặc tên tệp.
  - Không sử dụng biến toàn cục; có thể dùng bảng meta-data trong cơ sở dữ liệu để quản lý phân mảnh.
  - Tên bảng phân mảnh tuân theo định dạng: range_part0, range_part1, ... và rr_part0, rr_part1, .

* Hiệu suất: Với tập dữ liệu lớn (10 triệu bản ghi), cần tối ưu hóa đọc/ghi tệp và truy vấn SQL.

*Kiểm tra: Kiểm tra nội dung bảng sau khi chạy các hàm để đảm bảo tính đúng đắn, vì testcase tự động chỉ phát hiện lỗi biên dịch.

2.7. Phân chia công việc.

Họ và tên	Nhiệm vụ
Đinh Quốc Đại	Triển khai hàm RoundRobin_Partition() và hàm RoundRobin_Insert().
Hoàng Văn Linh	Triển khai hàm LoadRatings() và hàm Range_Partition() .
Nguyễn Thị Như Quỳnh	Triển khai hàm Range_Insert() và viết báo cáo.

IV. PHƯƠNG PHÁP THỰC HIỆN.

1. Triển khai hàm LoadRatings()

- Các tham số đầu vào

  Đường dẫn tuyệt đối đến tệp rating.dat
- Yêu cầu

  Tải nội dung của tệp rating.dat vào một bảng của PostgresQL có tên là Ratings, với schema: UserId(int), MovieId(int), Rating(float)

1.1. Ý tưởng giải quyết vấn đề

- Ban đầu mã được cung cấp chỉ đơn giản là tạo ra bảng, gồm đầy đủ các trường cho từng bản ghi trong file ratings.dat, ngoài ra còn có thêm một số trường phụ cho các ký tự phân tách(các trường extra).Sau đó dùng copy form để nạp dữ liệu, cuối cùng loại bỏ các cột phụ để lấy ra 3 cột cần thiết (UserId, MovieId, Rating).Tuy nhiên, cách làm này còn những hạn chế:Không làm sạch dữ liệu, tốn tài nguyên.
- Phương án đưa ra giải quyết:Áp dụng làm sạch dữ liệu(chỉ giữ lại các trường cần dùng) trước khi chèn vào bảng, tăng tốc bằng Unlogged table để giảm thời gian load các bản ghi vào bảng. Sau đó, ghi lại logged để đảm bảo toàn vẹn dữ liệu

- Đầu tiên, tiến hành kiểm tra xem bảng cũ có tồn tại trong database không, nếu có thì xóa(để tránh xảy ra ngoại lệ như tràn bộ nhớ, không tạo bảng)

```
cur.execute(f"DROP TABLE IF EXISTS {ratingstablename};")
```

- Sử dụng Unlogged table: Đối với các bảng thông thường, nó sẽ ghi lại các thay đổi vào WAL để đảm bảo có thể khôi phục sự cố khi có.Điều này đồng nghĩa với việc nếu chẳng may khi chạy database bị lỗi thì sẽ không thể rollback .Nhưng ở trường hợp này, mỗi phiên chạy sẽ tạo và xóa DB ở cuối mỗi phiên .Chính vì thế mà dùng Unlogged table là khả thi.Cú pháp tạo bảng unlogged tên ratings(ở đây được truyền tên vào):

```
cur.execute(f"""
    CREATE UNLOGGED TABLE {ratingstablename} (
        userid INTEGER,
        movieid INTEGER,
        rating FLOAT
    );
""")
```

- Sau khi tạo bảng, tiến hành làm sạch dữ liệu trước khi load vào bảng.Ta thực hiện tạo bản sao rồi phân tách các phần ra để lấy ra UserId, MovieId, Rating.Gỉa sử có bản ghi: 1::231::5::838983392, thì sau khi tách dùng giới hạn phân tách là ‘:’ thì sẽ được kết quả: ['1', '', '231', '', '5', '', '838983392'].Cuối cùng, chỉ cần lấy UserId = mảng tại vị trí 0, MovieId tại vị trí 2 , và Rating tại vị trí 4 để chèn vào bảng ratings.
  Cú pháp làm sạch dữ liệu:

```
temp_file = ratingsfilepath + ".clean"
with open(ratingsfilepath, 'r') as infile, open(temp_file, 'w') as outfile:
    for line in infile:
        parts = line.strip().split(':')
        if len(parts) >= 5:
            outfile.write(f"{parts[0]}\t{parts[2]}\t{parts[4]}\n")
```

- Tiếp theo, lấy dữ liệu trong file tạm lưu dữ liệu làm sạch và ghi vào bảng ratings sử dụng copy_form, sau đó ghi logged lại để có thể khôi phục DB và tránh mất dữ liệu trong trường hợp xấu nhất:

```
with open(temp_file, 'r') as f:
          cur.copy_from(f, ratingstablename, sep='\t', columns=('userid', 'movieid', 'rating'))

   # Load into DB
   with open(temp_file, 'r') as f:
           cur.copy_from(f, ratingstablename, sep='\t', columns=('userid', 'movieid', 'rating'))
   end_time = time.time()
   con.commit()

    print("Thời gian thực thi loadratings: {:.2f} giây".format(end_time - start_time))
```

- Sau khi tải thành công dữ liệu, tạo một bảng info lưu thông tin về số dòng của bảng ratings. Và thêm trigger tự động cập nhật số dòng của bảng ratings khi sau khi chèn hoặc xóa bản ghi của bảng.

```
-- hàm xử lí sự kiện insert hoặc delete
CREATE OR REPLACE FUNCTION update_row_count_func()
RETURNS TRIGGER AS $$
BEGIN
   IF TG_OP = 'INSERT' THEN
      -- Tăng số lượng hàng khi có thao tác INSERT
       UPDATE info SET numrow = numrow + 1 WHERE tablename = 'ratings';
   ELSIF TG_OP = 'DELETE' THEN
      -- Giảm số lượng hàng khi có thao tác DELETE
        UPDATE info SET numrow = numrow - 1 WHERE tablename = 'ratings';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- tạo trigger
CREATE TRIGGER ratings_after_insert
```

```
AFTER INSERT ON ratings
FOR EACH ROW
EXECUTE FUNCTION update_row_count_func();
```

- Sau khi thực hiện load dữ liệu vào trong bảng xong, thực hiện commit để lưu dữ liệu trong phiên làm việc, và thực hiện giải phóng tài nguyên không cần thiết

```
    con.commit()
    cur.close()
    os.remove(temp_file)
    end_time = time.time()
    print("Thời gian thực thi loadratings: {:.2f} giây".format(end_time - start_time))
```

1.2. Mã triển khai

-Mã triển khai tuần tự như ban đầu

```
    def loadratings(ratingstablename, ratingsfilepath, openconnection):
        create_db(DATABASE_NAME)
        con = openconnection
        cur = con.cursor()
        cur.execute("create table " + ratingstablename + "(userid integer, extra1 char, movieid integer, extra2 char, rating float, extra3 char, timestamp bigint);")
        cur.copy_from(open(ratingsfilepath),ratingstablename,sep=':')
        cur.execute("alter table " + ratingstablename + " drop column extra1, drop column extra2, drop column extra3, drop column timestamp;")
        cur.close()
        con.commit()
```

-Mã triển khai theo unlogged table và làm sạch dữ liệu

```
def loadratings(ratingstablename, ratingsfilepath, openconnection):
    start_time = time.time()
    con = openconnection
    cur = con.cursor()
```

```
    cur.execute(f"DROP TABLE IF EXISTS {ratingstablename};")

    cur.execute(f"""
        CREATE UNLOGGED TABLE {ratingstablename} (
            userid INTEGER,
            movieid INTEGER,
            rating FLOAT
        );
    """)
    con.commit()

    temp_file = ratingsfilepath + ".clean"
    rows = [0]
    with open(ratingsfilepath, 'r') as infile, open(temp_file, 'w') as outfile:
        for line in infile:
            parts = line.strip().split(':')
            if len(parts) >= 5:
                outfile.write(f"{parts[0]}\t{parts[2]}\t{parts[4]}\n")
            rows[0]+=1
    # Load into DB
    with open(temp_file, 'r') as f:
        cur.copy_from(f, ratingstablename, sep='\t', columns=('userid', 'movieid',
'rating'))
    end_time = time.time()
    con.commit()

    print("Thời gian thực thi loadratings: {:.2f} giây".format(end_time -
start_time))

    # add logged to table for rollback if crash
    cur.execute(f"ALTER TABLE {ratingstablename} SET LOGGED;")

    # prepare for roundrobin insert

    cur.execute("create table info (tablename varchar, numrow integer);")
    cur.execute(f"insert into info (tablename, numrow) values
('{ratingstablename}', {rows[0]});")
    '''
    Tạo trigger để cập nhật số dòng của bảng ratings khi thực hiện insert hoặc
delete
    '''
    cur.execute(f"
        CREATE OR REPLACE FUNCTION update_row_count_func()
        RETURNS TRIGGER AS $$
```

```
        BEGIN
            IF TG_OP = 'INSERT' THEN
                UPDATE info SET numrow = numrow + 1 WHERE tablename =
'{ratingstablename}';
            ELSIF TG_OP = 'DELETE' THEN
                UPDATE info SET numrow = numrow - 1 WHERE tablename =
'{ratingstablename}';
            END IF;
            RETURN NEW;
        END;
        $$ LANGUAGE plpgsql;
    ''')
    cur.execute(f"""
        CREATE TRIGGER ratings_after_insert
        AFTER INSERT ON {ratingstablename}
        FOR EACH ROW
        EXECUTE FUNCTION update_row_count_func();
    """)

    con.commit()
    cur.close()
    os.remove(temp_file)
```

2. Triển khai hàm Range_Partition()

- Các tham số đầu vào
  - + Bảng Ratings trong PostgreSQL vừa tạo ở trên
  - + Một số nguyên N là số phân mảnh cần tạo
- Yêu cầu trả về:
  Các bảng chứa phân mảnh ngang phù hợp với điều kiện phân mảnh của khoảng N

2.1. Ý tưởng giải quyết vấn đề

- Mã ban đầu được cung cấp thực hiện phân mảnh dữ liệu theo một cách tuần tự.Tức là sau khi tính toán các khoảng phân vùng dựa trên giá trị của rating xong, nó sẽ thực hiện lần lượt phân vùng với các bản ghi, sau đó dựa trên khoảng rating đã tính toán trong mỗi phân vùng, hàm execute sẽ thực hiện câu lệnh truy vấn chèn các bản ghi với rating thỏa mãn vào các khoảng tương ứng .
- Nếu thực hiện theo cách tuần tự như ban đầu, mọi công đoạn , mọi việc sẽ diễn ra theo tuần tự, các bản ghi được lấy ra và từng phân vùng được chèn vào nối tiếp nhau.Ngoài ra việc dùng chung một cursor và một connection cho toàn bộ quá trình.Điều này là nguyên nhân gây tăng thời gian phân vùng dữ liệu

- Hướng giải quyết :thực hiện triển khai đa luồng để phân mảnh dữ liệu nhanh và hiệu quả hơn, do các bản ghi chứa các vùng rating khác nhau, nên khi thực thi riêng lẻ mỗi vùng tương ứng sẽ không ảnh hưởng đến các vùng khác.
- Đầu tiên, ở trong hàm **roundrobinpartition(),** tạo một hàm sẵn cho mỗi tiến trình sử dụng functools.partial để tạo ra .Nó nhận vào các tham số đầu vào của hàm muốn được tạo hàm sẵn:

```
func = functools.partial(create_partition_process,
ratingstablename=ratingstablename,
numberofpartitions=numberofpartitions)
```

Hàm này sau khi định nghĩa create_partition_process, sau này chỉ cần gọi func(i) hoặc truyền vào map thì nó sẽ tự động được gán tham số của create_partition_process(Hàm này sẽ đề cập ở dưới sau)

- Thực hiện tạo pool tiến trình và chạy song song bằng class ProcessPoolExecutor của thư viện chuẩn Python, nó giúp quản lý và chạy song song nhiều tiến trình.Mỗi tiến trình là một phần của nó.Cú pháp:

```
with concurrent.futures.ProcessPoolExecutor() as executor:
    executor.map(func, range(numberofpartitions))
```

ở đây, ứng với mỗi khoảng phân mảnh của N được truyền qua hàm thực thi executor.map, nó sẽ tương ứng với một tiến trình để phân mảnh bảng, bằng cách thực thi hàm create_partition_process được tạo hàm sẵn qua biến func.

- Hàm create_partition_process
  - + Hàm này dùng để chia nhỏ bảng ratings thành các bảng con tương ướng với các phân mảnh dựa trên chỉ số phân mảnh.
  - + Có 5 phân vùng cho hàm rangepartition để phân mảnh dữ liệu. Vùng partition0 trong đoạn [0,1], vùng partition1 từ khoảng (1,2], vùng partition2 từ khoảng (2,3], vùng partition3 từ (3,4] và vùng cuối cùng partition5 từ (4,5]
  - + Ta xác định được vùng dựa trên partition_index, với partition_index chạy từ 0 đến 5,sau đó đặt tên bảng phân mảnh với từng vùng tương ứng từ đó, có logic sau:

```
delta = 5 / numberofpartitions
```

```
    minRange = partition_index * delta
    maxRange = minRange + delta
    table_name = f"range_part{partition_index}"
```

+ Tiếp đến với mỗi bảng, thay vì dùng chung cursor(con trỏ giao tiếp với cơ sở dữ liệu) cho các tiến trình phân mảnh, ta thực hiện tạo mỗi cursor riêng và mỗi bảng tiêng cho mỗi phân vùng.Điều này hỗ trợ tận dụng xử lý đa luồng thay vì xử lý tuần tự như code cũ.Tiếp đó là tạo bảng với 3 cột: UserId, MovieId, Rating để lưu dữ liệu

```
        con = psycopg2.connect(database='dds_assgn1',
user='postgres', password='1234', host='localhost', port='5432')
        cur = con.cursor()
        cur.execute(f"CREATE TABLE IF NOT EXISTS
{table_name} (userid INTEGER, movieid INTEGER, rating
FLOAT);")
```

+ Lần lượt xét các giá trị range_index(thứ tự phân mảng, chạy từ 0 đến 4) được truyền vào, nếu trường hợp nó là 0, thì ta sẽ có khoảng giá trị ratings là [0,1], do đó, thực hiện chèn các bản ghi có rating trong đoạn [0,1] vào bảng range_part0.Sau đó, từ range_index từ 1 đến 4, ta chỉ cần lấy từ khoảng (minRange và maxRange).Thí dụ range_index = 1 thì lấy từ (1,2] và tương tự với các khoảng khác.Logic đoạn mã như sau:

```
if partition_index == 0:
            cur.execute(f"""
                INSERT INTO {table_name}
                SELECT userid, movieid, rating FROM
{ratingstablename}
                WHERE rating >= %s AND rating <= %s;
            """, (minRange, maxRange))
        else:
            cur.execute(f"""
                INSERT INTO {table_name}
                SELECT userid, movieid, rating FROM
{ratingstablename}
                WHERE rating > %s AND rating <= %s;
            """, (minRange, maxRange))
```

- Cuối cùng, sau khi load thành công vào các bảng, ta chỉ cần commit dữ liệu và thực hiện giải phóng tài nguyên, kết thúc phiên làm việc

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

2.2. Mã triển khai

2.2.1. Mã triển khai theo hướng tuần tự

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    """
    Function to create partitions of main table based on range of ratings.
    """
    con = openconnection
    cur = con.cursor()
    delta = 5 / numberofpartitions
    RANGE_TABLE_PREFIX = 'range_part'
    for i in range(0, numberofpartitions):
        minRange = i * delta
        maxRange = minRange + delta
        table_name = RANGE_TABLE_PREFIX + str(i)
            cur.execute("create table " + table_name + " (userid integer, movieid integer, rating float);")
        if i == 0:
                cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablename + " where rating >= " + str(minRange) + " and rating <= " + str(maxRange) + ";")
        else:
                cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablename + " where rating > " + str(minRange) + " and
```

2.2.2. Mã triển khai theo hướng đa luồng

2.2.2.1. Hàm create_partition_process(Tạo các tiến trình xử lý phân mảnh)

```
def create_partition_process(partition_index, ratingstablename, numberofpartitions):
    delta = 5 / numberofpartitions
    minRange = partition_index * delta
    maxRange = minRange + delta
    table_name = f"range_part{partition_index}"
    con = psycopg2.connect(database='dds_assgn1', user='postgres', password='1234', host='localhost', port='5432')

    cur = con.cursor()

    cur.execute(f"CREATE TABLE IF NOT EXISTS {table_name} (userid INTEGER, movieid INTEGER, rating FLOAT);")

    if partition_index == 0:
        cur.execute(f"""
            INSERT INTO {table_name}
            SELECT userid, movieid, rating FROM {ratingstablename}
            WHERE rating >= %s AND rating <= %s;
        """, (minRange, maxRange))
    else:
        cur.execute(f"""
            INSERT INTO {table_name}
            SELECT userid, movieid, rating FROM {ratingstablename}
            WHERE rating > %s AND rating <= %s;
        """, (minRange, maxRange))

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

2.2.2.2. Hàm rangepartition

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    """
    Chạy phân mảnh theo range một cách song song (mỗi tiến trình có kết nối riêng).
    """
    start_time = time.time()
```

```
    func = functools.partial(create_partition_process,
ratingstablename=ratingstablename,
numberofpartitions=numberofpartitions)
    with concurrent.futures.ProcessPoolExecutor() as executor:
        executor.map(func, range(numberofpartitions))

    end_time = time.time()
    print("Thời gian thực thi rangepartition : {:.2f}
giây".format(end_time - start_time))
```

3. Triển khai hàm RoundRobin_Partition()

- Các tham số đầu vào
  - + Bảng ratings. Bảng có 3 cột lần lượt là userid, movieid và rating.
  - + Số lượng phân mảnh. (M mảnh)
- Yêu cầu: thực hiện phân mảnh bảng đầu vào bằng phương pháp phân mảnh vòng tròn (round robin).

3.1. Ý tưởng giải quyết vấn đề

- Phương pháp phân mảnh vòng tròn là phương pháp phân mảnh dữ liệu trong đó hàng mới được thêm vào bảng sẽ được gán lần lượt vào phân vùng theo một thứ tự tuần tự và lặp lại
  → Việc chọn phân vùng cho hàng sẽ phụ thuộc vào số thứ tự của hàng trong bảng.
- Nếu bảng có N bản ghi và M phân mảnh thì bản ghi một sẽ vào phân mảnh thứ nhất, bản ghi thứ 2 sẽ vào phân vùng thứ 2, cứ như vậy cho đến khi các phân mảnh đều có một bản ghi, sau đó quy trình sẽ lặp lại từ phân vùng đầu tiên. (giả sử phân vùng đầu tiên là 0, các phân vùng đánh số từ 0 đến M-1)
  → Công thức tìm số phân vùng chứa hàng thứ i ($1 \leq i \leq N$): (i-1) % M.
- Để lấy ra số hàng của bản ghi trong bảng sử dụng lệnh: ROW_NUMBER() OVER().
- Để đảm bảo tính toàn vẹn dữ liệu cho bảng ratings, thực hiện tạo một bảng tạm (tên bảng: temp) là bản copy của bảng ratings. Vì là bảng tạm và cần tối ưu hiệu năng nên sẽ tắt chức năng LOGGED trên bảng này.

```
CREATE UNLOGGED TABLE temp AS
SELECT *, ROW_NUMBER() OVER() -1 as stt FROM ratings;
```

- Sau khi tạo xong bảng tạm, thực hiện việc tạo các phân mảnh của bảng ratings. Đặt tên phân mảnh sẽ là rrobin_part + i với $0 \leq i < M$.

```
CREATE TABLE rrobin_parti AS
SELECT userid, movieid, rating FROM temp
WHERE MOD(stt ,M) = i;
```

- Sau khi tạo xong các phân mảnh thực hiện xóa bảng bảng tạm và commit giao dịch.

```
DROP TABLE temp;
```

- Sử dụng đa luồng để tối ưu bài toán:
    + Sau khi tạo xong bảng tạm, thực hiện commit để đảm bảo các connection mở thêm sẽ đều nhìn thấy bảng tạm.
    + Tạo các câu lệnh truy vấn đẩy vào một hàng đợi.
    + Vì các câu lệnh đều lấy dữ liệu từ bảng tạm để chèn vào các phân mảnh riêng biệt nên có thể thực hiện đồng thời. → tạo thêm kết nối tới cơ sở dữ liệu để triển khai đa luồng.

3.2. Mã triển khai

- Mã triển khai tuần tự:

```
def roundrobinpartition(ratingstablename, numberofpartitions,
openconnection):
    con = openconnection
    cur = con.cursor()

    RROBIN_TABLE_PREFIX = 'rrobin_part'
     # tạo bảng tạm
    TEMP_TABLE = 'temp'
    cur.execute(f'''
        CREATE UNLOGGED TABLE {TEMP_TABLE} AS
        SELECT *, ROW_NUMBER() OVER() - 1 AS stt FROM
{ratingstablename};
    ''')
    # thực hiện các truy vấn tạo phân mảnh.
    for i in range (numberofpartitions):
        table_name = RROBIN_TABLE_PREFIX + str(i)
        cur.execute(f'''
            CREATE TABLE {table_name} AS
            SELECT userid, movieid, rating FROM {TEMP_TABLE}
            WHERE MOD(stt, {numberofpartitions}) = {i};
        ''')
    # xóa bảng tạm
    cur.execute(f 'drop table {TEMP_TABLE};')
    con.commit()
    cur.close()
```

- Mã triển khai theo đa luồng

```
# đoạn mã sử dụng 2 luồng để thực thi truy vấn
# hàm nhận và thực thi câu lệnh sql từ hàng đợi
def rrobin_execute_query(sql_list: queue):
    con = getopenconnection(dbname=DATABASE_NAME)
    while True:
        sql = sql_list.get() # lấy lệnh sql từ hàng đợi
        """
        Nếu lấy được giá trị None (tín hiệu báo hết truy vấn) từ hàng đợi thì thoát khỏi
vòng lặp, đóng kết nối.
        """
        if sql is None:
            break
        cur = con.cursor()
        cur.execute(sql)
        con.commit()
        cur.close()
    con.close()
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'
    TEMP_TABLE = 'temp_ratings'
    NUM_THREADS = 2
    cur.execute(f'''
        CREATE UNLOGGED TABLE {TEMP_TABLE} AS
        SELECT *, ROW_NUMBER() OVER() - 1 AS stt FROM {ratingstablename};
    ''')
    con.commit()
    sql_list = queue.Queue()
    # tạo các câu truy vấn và đẩy vào hàng đợi.
    for i in range(numberofpartitions):
        table_name = RROBIN_TABLE_PREFIX + str(i)
        sql= f'''
        CREATE TABLE {table_name} AS
        SELECT userid, movieid, rating FROM {TEMP_TABLE}
        WHERE MOD(stt, {numberofpartitions}) = {i};
        '''
        sql_list.put(sql)
    # đặt None là tín hiệu hết truy vấn.
    for i in range(NUM_THREADS):
        sql_list.put(None)
    threads =  []
    for t in range(NUM_THREADS):
        # tạo luồng để thực thi hàm rrobin_execute_query và truyền vào hàng đợi chứa
```

```
lệnh sql
        t = threading.Thread(target=rrobin_execute_query, args=(sql_list,))
        t.start() # thực thi luồng
        threads.append(t)
    # chờ các luồng thực hiện xong
    for t in threads:
        t.join()
    cur.execute(f'drop table {TEMP_TABLE};') # xóa bảng tạm
    con.commit()
    cur.close()
```

4. Triển khai hàm RoundRobin_Insert()

- Các tham số đầu vào:
  - + Bảng ratings
  - + UserId
  - + MovieId
  - + Rating
- Yêu cầu, chèn bản ghi (UserId, MovieId, Rating) vào bảng ratings và vào đúng phân mảnh theo cách round robin.

4.1. Ý tưởng giải quyết vấn đề

- Như đã phân tích ở phần triển khai hàm RoundRobin_Partition(), thì việc xác định phân mảnh chứa hàng mới sẽ phụ thuộc vào số dòng của bản ghi sau khi chèn vào bảng ratings
  → Có thể sử dụng hàm COUNT(*) để lấy ra số dòng của bảng nhưng có một vấn đề số lượng dòng rất lớn (cỡ 10 triệu dòng) nên việc lấy ra số dòng bằng COUNT(*) sẽ tốt rất nhiều chi phí.
  → Đề xuất lấy thông tin số dòng của bảng info được tạo khi thực hiện chạy hàm LoadRatings().
- Các bước triển khai ý tưởng:
  - + Thực hiện chèn dòng mới (UserId, MovieId, Rating) vào bảng ratings. Thực hiện xong việc chèn dữ liệu, số lượng dòng của bảng ratings sẽ đươc tự động tăng lên do đã tạo trigger để xử lý hành động chèn.

    ```
    INSERT INTO ratings (userid, moivieid, rating)
    VALUES (UserId, MovieId, Rating);
    ```

  - + Lấy ra là số lượng dòng của bảng ratings từ bảng info.

    ```
    SELECT COUNT(*) FROM info
    WHERE tablename = ‘ratings’;
    ```

+ Đặt M là số lượng phân mảnh. Để lấy ra số lượng phân mảnh, thực hiện đếm số bảng có tên bảng chứa tiền tố "rrobin_part" (đây là tiền tố được thêm vào khi thực hiện phân mảnh).

```
SELECT COUNT(*) FROM pg_stat_user_tables
WHERE relname like 'rrobin_part%' ;
```

+ Đạt i là chỉ số phân mảnh cần chèn → i = (total_rows-1) % M.
+ Cuối cùng thực hiện chèn dòng mới (UserId, MovieId, Rating) vào phân mảnh. Tên phẩn mảnh được ghép từ bởi tiền tố "rrobin_part" và chỉ số i.

```
INSERT INTO rrobin_parti (userid, moivieid, rating)
VALUES (UserId, MovieId, Rating);
```

4.2. Mã triển khai

```
def count_partitions(prefix, openconnection):
    """
    Hàm thực hiện công việc đếm số lượng phân mảnh dựa vào tiền tố.
    """
    con = openconnection
    cur = con.cursor()
    cur.execute("select count(*) from pg_stat_user_tables where relname like " + "'" +
prefix + "%';")
    count = cur.fetchone()[0]
    cur.close()
    return count

# hàm thực thi chèn dữ liệu theo round robin
def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    # thực hiện lệnh chèn dữ liệu mới vào bảng ratings
   cur.execute(f"
        INSERT INTO {ratingstablename} (userid, movieid, rating)
        VALUES ({userid}, {itemid}, {rating});
    ")

    # thực hiện lấy ra số lượng của bảng ratings
    cur.execute(f"select numrow from info where tablename = '{ratingstablename}';")
    num_row = cur.fetchall()[0][0]
```

```
    # thực hiện đếm số phân mảnh.
    num_partitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)

    # lấy ra chỉ số của phân mảnh cần chèn thêm dữ liệu
    id = (num_row-1) % num_partitions
    table_name = RROBIN_TABLE_PREFIX + str(id) # tên bảng = tiền tố + chỉ số

    # thực hiện chèn dữ liệu mới vào phân mảnh.
   cur.execute(f"
       INSERT INTO {table_name} (userid, movieid, rating)
       VALUES ({userid}, {itemid}, {rating});
   ")
    cur.close()
    con.commit()
```

5. Triển khai hàm Range_Insert()

- Các tham số đầu vào
  + Bảng ratings
  + UserId
  + MovieId
  + Rating

- Yêu cầu : Yêu cầu, chèn bản ghi (UserId, MovieId, Rating) vào bảng ratings và vào đúng phân mảnh theo cách Range.

5.1. Ý tưởng giải quyết vấn đề

- Việc chèn bản ghi mới vào bảng ratings và phân phối vào phân mảnh phụ thuộc vào giá trị rating. Phạm vi phân mảnh được xác định trước bởi hàm rangepartition(), chia khoảng [0, 5] thành N phần đều nhau (ví dụ: N = 3 → các khoảng [0-1.67), [1.67-3.34), [3.34-5]).
- Thay vì đếm số dòng hoặc truy vấn toàn bộ bảng, sử dụng logic tính toán chỉ số phân mảnh dựa trên giá trị rating và số phân mảnh (N). Số phân mảnh được lấy từ hàm count_partitions() dựa trên tiền tố range_part.
- Các bước triển khai:
  + Thực hiện chèn dòng mới vào bảng ratings: Sử dụng lệnh INSERT INTO ratings để chèn bản ghi (userid, itemid, rating) vào bảng ratings.
  + Xác định số lượng phân mảnh: Gọi hàm count_partitions() với tiền tố range_part để lấy tổng số phân mảnh (N).

+ Tính chỉ số phân mảnh: Tính index dựa trên giá trị rating và số phân mảnh N bằng cách chia rating / (5.0 / N) và điều chỉnh ranh giới (giảm index nếu rating nằm ở ranh giới và index > 0).
+ Chèn vào phân mảnh tương ứng:
    - Tạo tên bảng phân mảnh bằng cách ghép tiền tố range_part với chỉ số index ;
    - Thực hiện lệnh INSERT INTO để chèn bản ghi vào phân mảnh đó.

5.2. Mã triển khai

```
def count_partitions(prefix, openconnection):
    """
    Hàm thực hiện công việc đếm số lượng phân mảnh dựa vào tiền tố.
    """
    con = openconnection
    cur = con.cursor()
    cur.execute("SELECT COUNT(*) FROM pg_stat_user_tables WHERE
relname LIKE %s", (prefix + '%',))
    count = cur.fetchone()[0]
    cur.close()
    return count

# Hàm thực thi chèn dữ liệu theo phương pháp range
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    con = openconnection
    cur = con.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    # Thực hiện lệnh chèn dữ liệu mới vào bảng ratings
    cur.execute("INSERT INTO %s (userid, movieid, rating) VALUES (%s,
%s, %s)",
                (ratingstablename, userid, itemid, rating))

    # Thực hiện đếm số phân mảnh
  num_partitions= count_partitions(RANGE_TABLE_PREFIX,
openconnection)
```

```
    # Lấy ra chỉ số của phân mảnh cần chèn thêm dữ liệu
    delta = 5.0 / num_partitions
    index = int(rating / delta)
    if rating % delta == 0 and index > 0:
        index -= 1
    if index >= num_partitions:
        index = num_partitions - 1
    table_name = RANGE_TABLE_PREFIX + str(index)
    # Thực hiện chèn dữ liệu mới vào phân mảnh
    cur.execute("INSERT INTO %s (userid, movieid, rating) VALUES (%s,
%s, %s)",
                (table_name, userid, itemid, rating))

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