HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÀI TẬP LỚN
CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên hướng dẫn: Kim Ngọc Bách

Nhóm 22

Danh sách sinh viên thực hiện:
Ngô Xuân Hòa - B22DCCN326
Lô Minh Huy - B22DCCN383
Nguyễn Võ Hưng - B22DCCN417

Hà Nội – 2025

LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn sâu sắc đến Thầy Kim Ngọc Bách, giảng viên bộ môn, người đã tận tâm giảng dạy và truyền đạt cho chúng em những kiến thức quý giá về hệ quản trị cơ sở dữ liệu phân tán trong suốt học kỳ vừa qua.

Với phương pháp giảng dạy khoa học, lối truyền đạt mạch lạc và những ví dụ thực tiễn sinh động, Thầy không chỉ giúp chúng em nắm vững các khái niệm chuyên môn như phân mảnh dữ liệu, tổ chức bảng, hay kỹ năng thao tác với cơ sở dữ liệu bằng Python, mà còn khơi dậy trong chúng em niềm hứng thú và tư duy hệ thống khi tiếp cận với lĩnh vực này.

Đề tài bài tập lớn lần này là cơ hội quý báu để chúng em không chỉ củng cố kiến thức đã học, mà còn rèn luyện tư duy logic, kỹ năng lập trình và khả năng giải quyết vấn đề thực tiễn – những hành trang quan trọng cho con đường phát triển trong ngành công nghệ thông tin sau này.

Một lần nữa, chúng em xin gửi lời cảm ơn chân thành nhất đến Thầy. Mong rằng trong chặng đường học tập sắp tới, chúng em sẽ tiếp tục nhận được sự đồng hành và chỉ dẫn tận tình từ Thầy trong những môn học khác.

Mục lục

I. GIỚI THIỆU

1. Bối cảnh và ý nghĩa thực tiễn

Trong bối cảnh công nghệ thông tin phát triển vượt bậc, việc xử lý và quản lý dữ liệu lớn (Big Data) đã trở thành một thách thức quan trọng đối với các tổ chức và doanh nghiệp. Với sự bùng nổ của thông tin kỹ thuật số, các hệ thống cơ sở dữ liệu truyền thống gặp phải nhiều hạn chế về hiệu suất, khả năng mở rộng và quản lý.

Database Partitioning (phân vùng cơ sở dữ liệu) là một trong những kỹ thuật quan trọng nhất để giải quyết các vấn đề này. Kỹ thuật này không chỉ giúp cải thiện hiệu suất truy vấn mà còn tối ưu hóa việc sử dụng tài nguyên hệ thống, đồng thời tăng khả năng mở rộng và bảo trì của cơ sở dữ liệu.

Việc lựa chọn đề tài này nhằm mục đích nghiên cứu sâu về các phương pháp phân vùng khác nhau và đánh giá hiệu quả của chúng trong thực tế, đặc biệt là trong việc xử lý dữ liệu đánh giá phim (movie ratings) - một loại dữ liệu phổ biến trong các hệ thống khuyến nghị.

2. Mục tiêu nghiên cứu

2.1. Mục tiêu chính

- Nghiên cứu và hiểu rõ các khái niệm cơ bản về Database Partitioning
- Cài đặt và triển khai hai phương pháp phân vùng chính: Range Partitioning và Round Robin Partitioning
- So sánh và đánh giá hiệu suất của các phương pháp phân vùng
- Rút ra những kết luận và khuyến nghị cho việc áp dụng trong thực tế

2.2. Mục tiêu cụ thể

- Xây dựng hệ thống quản lý dữ liệu ratings với khả năng phân vùng
- Đo đạc và phân tích hiệu suất của từng phương pháp
- Đánh giá tác động của phân vùng đến các thao tác CRUD (Create, Read, Update, Delete)

- Đưa ra khuyến nghị về việc lựa chọn phương pháp phân vùng phù hợp

2.3. Mục tiêu bài tập

- Làm việc với cơ sở dữ liệu PostgreSQL/MySQL: Tạo bảng, chèn và truy vấn dữ liệu; quản lý kết nối và xử lý truy cập bằng Python.
- Lập trình xử lý dữ liệu bằng Python: Đọc file, phân tích – chuẩn hóa dữ liệu; xây dựng hàm tải và phân mảnh dữ liệu theo thuật toán.
- Nắm vững thuật toán phân mảnh ngang: Áp dụng hai phương pháp phân mảnh – theo khoảng (Range Partitioning) dựa trên giá trị rating, và phân mảnh vòng tròn (Round Robin Partitioning) để chia đều dữ liệu.
- Triển khai hàm chèn động: Chèn bản ghi mới vào bảng chính và phân mảnh tương ứng; đảm bảo tính đúng đắn và nhất quán trong phân mảnh sau khi chèn.

3. Phân chia công việc

Họ và tên	MSV	Công việc
Lô Minh Huy	B22DCCN383	Tìm hiểu và xử lý tối ưu loadratings. Tìm hiểu, xử lý phân mảnh Range. Cài đặt, triển khai các chương trình chính như create database, count partition. Viết báo cáo.
Ngô Xuân Hòa	B22DCCN326	Tìm hiểu, xử lý phân mảnh Round-Robin. Đưa ra hướng giải quyết chính. Viết báo cáo

Nguyễn Võ Hưng	B22DCCN417	Tìm hiểu và xử lý tối ưu loadratings. Tìm hiểu, xử lý phân mảnh Range. Cài đặt môi trường và kết nối với CSDL.

II. TỔNG QUAN VÀ CƠ SỞ LÝ THUYẾT

1. Vấn đề đặt ra

Trong thời đại số hiện tại, khối lượng dữ liệu được tạo ra hàng ngày đang tăng theo cấp số nhân. Theo báo cáo của IDC, khối lượng dữ liệu toàn cầu sẽ tăng từ 33 zettabytes vào năm 2018 lên 175 zettabytes vào năm 2025. Điều này đặt ra những thách thức lớn cho các hệ thống cơ sở dữ liệu truyền thống:

- Hiệu suất truy vấn giảm: Khi kích thước bảng tăng lên, thời gian thực hiện các truy vấn cũng tăng theo
- Khó khăn trong bảo trì: Việc backup, restore và maintain các bảng lớn trở nên phức tạp
- Giới hạn về tài nguyên: Bộ nhớ và CPU không đủ để xử lý các thao tác trên dữ liệu lớn
- Bottleneck về I/O: Đọc/ghi dữ liệu trở thành nút thắt cổ chai

Database Partitioning ra đời như một giải pháp tự nhiên cho các vấn đề trên. Thay vì lưu trữ tất cả dữ liệu trong một bảng duy nhất, kỹ thuật này chia dữ liệu thành nhiều phần nhỏ hơn (partitions), mỗi phần có thể được quản lý độc lập.

2. Tổng quan về Database Partitioning

Database Partitioning là kỹ thuật chia một bảng logic lớn thành nhiều phần vật lý nhỏ hơn (partitions). Từ góc độ người dùng, bảng vẫn được xem như một thực thể duy nhất, nhưng về mặt vật lý, dữ liệu được phân tán và lưu trữ trong các partition khác nhau.

Database Partitioning có thể được phân loại theo nhiều cách khác nhau:

- Theo cách phân chia:
  - Horizontal Partitioning (phân vùng ngang): Chia theo hàng
  - Vertical Partitioning (phân vùng dọc): Chia theo cột
- Theo thuật toán phân chia:

- Range Partitioning: Phân chia theo khoảng giá trị
- Hash Partitioning: Phân chia theo hàm băm
- List Partitioning: Phân chia theo danh sách giá trị
- Round Robin Partitioning: Phân chia tuần tự

Lợi ích mang lại:

- Cải thiện hiệu suất truy vấn lên đến 10-100 lần
- Giảm thời gian backup/restore từ giờ xuống phút
- Tăng khả năng xử lý song song
- Cải thiện khả năng mở rộng hệ thống

3. Range Partitioning (Phân vùng theo khoảng)

Range Partitioning (Phân vùng theo khoảng) là một kỹ thuật phân mảnh dữ liệu ngang, trong đó các bản ghi được phân chia vào các phân mảnh (bảng con) dựa trên giá trị thuộc một trường nhất định – thường là số hoặc ngày.

Nguyên lý hoạt động:

- Dữ liệu được chia thành nhiều khoảng giá trị không giao nhau.
- Mỗi khoảng tương ứng với một phân mảnh riêng.
- Khi chèn bản ghi mới, hệ thống sẽ kiểm tra giá trị trường phân mảnh để xác định bản ghi đó thuộc khoảng nào và đưa vào phân mảnh tương ứng.

Ưu điểm: hiệu quả khi truy vấn dữ liệu theo điều kiện lọc trên trường phân mảnh, dễ quản lý nếu dữ liệu có phân bố theo thứ bậc, mức độ, thời gian,…

Nhược điểm: nếu dữ liệu không phân bố đều, một số partition có thể chứa nhiều dữ liệu hơn, phải hiểu rõ đặc tính dữ liệu để chọn khoảng phù hợp, việc thay đổi khoảng partition sau khi đã tạo khá phức tạp,…

4. Round Robin Partitioning (Phân vùng vòng tròn)

Round Robin Partitioning (Phân vùng vòng tròn) là một kỹ thuật phân mảnh ngang, trong đó các bản ghi được phân chia lần lượt và đồng đều vào các phân mảnh (bảng con) theo thứ tự.

Nguyên lý hoạt động:

- Mỗi bản ghi mới được chèn vào một phân mảnh khác nhau theo vòng tròn.

- Ví dụ, nếu có 3 phân mảnh: bản ghi đầu tiên vào phân mảnh 1, bản ghi thứ hai vào phân mảnh 2, bản ghi thứ ba vào phân mảnh 3, bản ghi thứ tư quay lại phân mảnh 1… và cứ tiếp tục như vậy.

Ưu điểm: phân phối dữ liệu rất đều giữa các phân mảnh, không cần kiểm tra điều kiện giá trị khi phân mảnh → đơn giản và hiệu quả khi dữ liệu không có quy luật rõ ràng,…

Nhược điểm: khó tối ưu hóa truy vấn theo điều kiện lọc, vì không thể đoán bản ghi nằm ở phân mảnh nào nếu không quét tất cả, khi join với bảng khác, có thể cần scan nhiều partition,…

III. TRIỂN KHAI

1. Yêu cầu bài toán

Bài tập yêu cầu xây dựng một hệ thống quản lý dữ liệu đánh giá phim (movie ratings) với khả năng phân vùng dữ liệu. Các đánh giá được thực hiện trên thang điểm 5 sao, có thể chia nửa sao. Dấu thời gian(Timestamp) là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970.

2. Dữ liệu đầu vào

Tệp ratings.dat chứa 10 triệu đánh giá và 100.000 thẻ được áp dụng cho 10.000 bộ phim bởi 72.000 người dùng. Mỗi dòng trong tệp đại diện cho một đánh giá của một người dùng với một bộ phim, và có định dạng như sau:

UserID::MovieID::Rating::Timestamp

3. Giải quyết bài toán

Thực hiện phân mảnh bằng hai phương pháp

3.1 Phân mảnh theo khoảng giá trị (Range Partitioning):

Dữ liệu sẽ được chia thành nhiều phần nhỏ (phân mảnh) dựa trên giá trị cụ thể của trường rating. Mỗi mảnh dữ liệu sẽ tương ứng với một khoảng giá trị khác nhau, giúp cô lập và quản lý dữ liệu dễ dàng hơn.

Cách thực hiện: Ví dụ khi chọn chia thành 3 phân mảnh (N = 3), có thể phân chia theo các khoảng như sau:

- range_part0: rating $\in$ [0, 1.67]

- range_part1: rating ∈ (1.67, 3.34]
- range_part2: rating ∈ (3.34, 5]

Mục tiêu hướng đến:
Khi cần truy xuất những bản ghi với điều kiện cụ thể, như "rating > 4.5", hệ thống chỉ cần tìm trong đúng phân mảnh chứa khoảng đó, thay vì phải duyệt toàn bộ bảng dữ liệu → tối ưu hiệu suất truy vấn đáng kể.

3.2. Phân mảnh theo vòng tròn (Round Robin Partitioning):

Dữ liệu được phân bổ tuần tự và luân phiên vào các bảng phân mảnh. Cách này không phụ thuộc vào giá trị rating mà dựa vào vị trí xuất hiện của bản ghi trong tập dữ liệu.

Cách thực hiện: Giả sử có 3 bảng phân mảnh (part0, part1, part2):

- Bản ghi đầu tiên (chỉ số 0) được đưa vào part0
- Bản ghi thứ hai (chỉ số 1) đưa vào part1
- Bản ghi thứ ba (chỉ số 2) vào part2
- Sau đó quay lại từ đầu: bản ghi thứ tư (chỉ số 3) vào part0, v.v.

Mục tiêu hướng đến:
Giúp phân phối dữ liệu đồng đều giữa các phân mảnh → cân bằng tải khi xử lý song song hoặc truy cập đồng thời. Phương pháp này đặc biệt phù hợp nếu dữ liệu không có quy luật rõ ràng để phân loại theo giá trị.

4. Các bước thực hiện

4.1 Thiết lập và chuẩn bị môi trường

Tạo cơ sở dữ liệu nếu chưa có.
Kết nối CSDL thông qua getopenconnection() được dùng ở mọi hàm.

```python
def create_db(dbname):  1 usage
    postgres_conn = getopenconnection(dbname='postgres')
    postgres_conn.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    db_cursor = postgres_conn.cursor()

    check_db_query = "SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname='%s'" % (dbname,)
    db_cursor.execute(check_db_query)
    db_exists = db_cursor.fetchone()[0]

    if db_exists == 0:
        create_db_query = 'CREATE DATABASE %s' % (dbname,)
        db_cursor.execute(create_db_query)

    db_cursor.close()
    postgres_conn.close()
```

4.2. Tải dữ liệu vào bảng chính ratings

Gọi hàm loadratings(ratingstablename, ratingsfilepath, openconnection) để load dữ liệu từ ratings.dat vào bảng ratings.

```python
def getopenconnection(user='postgres', password='lohuyls123', dbname=DATABASE_NAME):  1 usage
    conn_string = f"dbname='{dbname}' user='{user}' host='localhost' password='{password}'"
    return psycopg2.connect(conn_string)
```

```python
def loadratings(ratingstablename, ratingsfilepath, openconnection):  2 usages (1 dynamic)
    create_db(DATABASE_NAME)
    connection = openconnection
    cursor = connection.cursor()
    drop_query = f"DROP TABLE IF EXISTS {ratingstablename};"
    cursor.execute(drop_query)
    create_query = f"CREATE TABLE {ratingstablename} (userid INTEGER, movieid INTEGER, rating FLOAT);"
    cursor.execute(create_query)
    connection.commit()
    cursor.close()
    processed_file = preprocess_file_to_temp(ratingsfilepath)
    try:
        with connection.cursor() as copy_cursor:
            with open(processed_file, 'r', encoding='utf-8') as data_file:
                copy_command = f"COPY {ratingstablename} (userid, movieid, rating) FROM STDIN WITH CSV DELIMITER ','"
                copy_cursor.copy_expert(copy_command, data_file)
                connection.commit()
    finally:
        if os.path.isfile(processed_file):
            os.unlink(processed_file)
```

- Hàm load ratings nhận vào 3 tham số
    - ratingstablename: tên của bảng rating trong cơ sở dữ liệu
    - ratingsfilepath: đường dẫn đến file data
    - openconnection: connection đến database

Các bước thực hiện:

- Đầu tiên tạo mới 1 database để chắc chắn rằng database đã tồn tại
- Drop table ratings trước đó nếu như đã tồn tại
- Tạo mới 1 table ratings
- Gọi đến hàm preprocess_file_to_temp(input_path, sep_char=',')
  Hàm preprocess_file_to_temp(input_path, sep_char=',') được gọi đến trong hàm loadratings là một bước tiền xử lý dữ liệu dùng để chuyển đổi dữ liệu từ định dạng gốc trong file ratings.dat sang định dạng chuẩn CSV để tương thích với CSDL.

```python
def preprocess_file_to_temp(input_path, sep_char=','):  1 usage
    temp_fd, temp_path = tempfile.mkstemp(prefix="ratings_processed_", suffix=".csv", text=True)

    try:
        with os.fdopen(temp_fd,  mode: 'w', encoding='utf-8') as temp_file:
            with open(input_path, 'r', encoding='utf-8', errors='ignore') as source_file:
                line_data = source_file.readlines()
                for raw_line in line_data:
                    cleaned_line = raw_line.strip()
                    if not cleaned_line:
                        continue

                    columns = cleaned_line.split('::')
                    if len(columns) >= 3:
                        output_row = sep_char.join(columns[:3])
                        temp_file.write(output_row + '\n')
    except:
        if os.path.exists(temp_path):
            os.unlink(temp_path)
        raise
    return temp_path
```

Cách thực hiện:

- Đọc toàn bộ nội dung từ file ratings.dat, với định dạng các trường được phân cách bởi dấu ::
- Tạo một file tạm thời, xử lý từng dòng trong file nguồn để chuyển định dạng phân cách từ :: thành dấu ,
- Trả về file tạm

Sử dụng câu lệnh COPY thông qua copy_expert() để nạp toàn bộ dữ liệu từ file CSV tạm đã xử lý vào bảng ratings trong PostgreSQL:

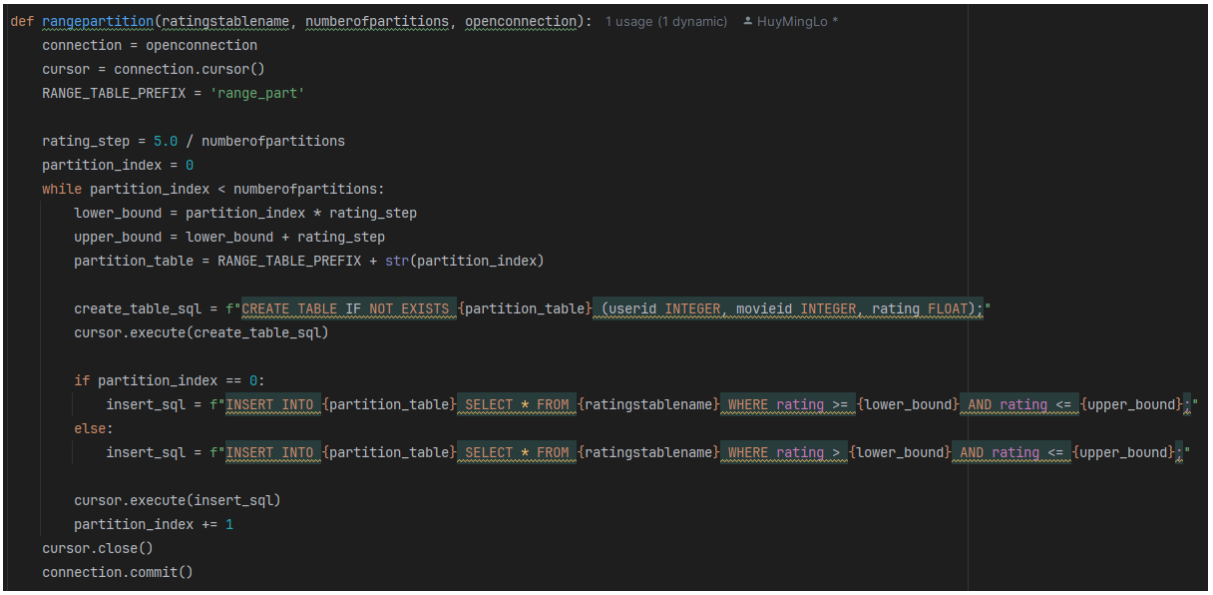
- Lệnh COPY trong PostgreSQL cho phép nạp dữ liệu hàng loạt vào bảng một cách rất nhanh chóng thay vì insert từng record theo cách nguyên thủy

Sau khi nạp xong, xóa file tạm khỏi hệ thống để giải phóng tài nguyên.

4.3. Phân mảnh theo khoảng (Range Partitioning)

Mục đích nhằm chia dữ liệu thành nhiều phân đoạn dựa theo giá trị của trường rating. Mỗi phân mảnh sẽ đại diện cho một khoảng giá trị cụ thể. Mục tiêu là hỗ trợ các truy vấn lọc theo khoảng giá trị hiệu quả hơn – ví dụ, truy vấn rating > 4.5 sẽ chỉ cần quét một phân mảnh duy nhất thay vì toàn bộ bảng.

4.3.1. Hàm rangepartition(ratingstablename, numberofpartitions, openconnection) chia bảng ratings thành N partitions theo khoảng rating



```
def rangepartition(ratingstablename, numberofpartitions, openconnection):  1 usage (1 dynamic)  HuyMingLo *
    connection = openconnection
    cursor = connection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    rating_step = 5.0 / numberofpartitions
    partition_index = 0
    while partition_index < numberofpartitions:
        lower_bound = partition_index * rating_step
        upper_bound = lower_bound + rating_step
        partition_table = RANGE_TABLE_PREFIX + str(partition_index)

        create_table_sql = f"CREATE TABLE IF NOT EXISTS {partition_table} (userid INTEGER, movieid INTEGER, rating FLOAT);"
        cursor.execute(create_table_sql)

        if partition_index == 0:
            insert_sql = f"INSERT INTO {partition_table} SELECT * FROM {ratingstablename} WHERE rating >= {lower_bound} AND rating <= {upper_bound};"
        else:
            insert_sql = f"INSERT INTO {partition_table} SELECT * FROM {ratingstablename} WHERE rating > {lower_bound} AND rating <= {upper_bound};"

        cursor.execute(insert_sql)
        partition_index += 1
    cursor.close()
    connection.commit()
```

- Giải thích các bước cài đặt hàm:
  - `rating_step = 5.0 / numberofpartitions`
  - Tính khoảng cách bước nhảy của các ratings ở đây nhằm phân chia các giá trị về ratings cho các bảng

```
 partition_index = 0
    while partition_index < numberofpartitions:
        lower_bound = partition_index * rating_step
        upper_bound = lower_bound + rating_step
        partition_table = RANGE_TABLE_PREFIX + str(partition_index)

        create_table_sql = f"CREATE TABLE IF NOT EXISTS {partition_table} (userid INTEGER, movieid INTEGER, rating FLOAT);"
        cursor.execute(create_table_sql)

        if partition_index == 0:
            insert_sql = f"INSERT INTO {partition_table} SELECT * FROM
```

```
{ratingstablename} WHERE rating >=
{lower_bound} AND rating <=
{upper_bound};"
        else:
            insert_sql = f"INSERT INTO
{partition_table} SELECT * FROM
{ratingstablename} WHERE rating >
{lower_bound} AND rating <=
{upper_bound};"

        cursor.execute(insert_sql)
        partition_index += 1
```

- Bước đầu khởi tạo index của partition = 0:
  - Tính cận trên và cận dưới của rating:
  vd: với index = 1, rating_step = 1 thì cận trên là 2, cận dưới là 1
  - Tạo bảng để chuẩn bị cho việc phân mảnh, tên bảng được đặt theo prefix + index của partition
  - Thực hiện insert các dữ liệu vào bảng thông qua việc quét bảng gốc và dùng điều kiện lọc:
    - Đối với trường hợp phân mảnh có index = 0, ta sẽ lấy ratings lớn hơn bằng (>=) cận dưới và nhỏ hơn bằng (<=) cận trên
    - Đối với các trường hợp còn lại ta sẽ lấy ratings có giá trị lớn hơn (>) cận dưới và nhỏ hơn bằng (<=) cận trên
  - Sau insert thì tăng index lên để tiếp tục insert tiếp vào các phân mảnh tiếp theo

4.3.2. Hàm rangeinsert(ratingstablename, userid, itemid, rating, openconnection)

Đầu vào hàm là 1 thông tin insert của 1 row

```python
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):  1 usage (1 dynamic)  v.huwng.204 *
    connection = openconnection
    cursor = connection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    total_partitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
    if total_partitions == 0:
        raise Exception("Partitions have not been created. Call rangepartition first.")
    rating_step = 5.0 / total_partitions
    partition_idx = int(rating / rating_step)

    if partition_idx >= total_partitions:
        partition_idx = total_partitions - 1
    elif rating % rating_step == 0 and rating != 0:
        partition_idx = max(partition_idx - 1, 0)

    target_table = RANGE_TABLE_PREFIX + str(partition_idx)
    insert_query = f"INSERT INTO {target_table}(userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});"
    cursor.execute(insert_query)

    cursor.close()
    connection.commit()
```

- Các bước cài đặt hàm:
    - Đầu tiên phải xác định được số phân mảnh hiện tại đang có dựa trên phương pháp range partition
    - Sau đó tính bước nhảy rating step tương tự như trên để xác định được khoảng cách bước nhảy
    - tiếp theo ta sẽ tính xem rating này sẽ thuộc phân mảnh nào bằng cách lấy phần nguyên của (rating / bước nhảy) khi đó sẽ có 2 trường hợp:
        - TH1: nếu chỉ số index lớn hơn bằng (>=) số lượng phân mảnh
          ví dụ khi rating = 5.0, phép tính ban đầu cho ra partition_idx = 5, một chỉ số không tồn tại. Mệnh đề if này sẽ phát hiện ra điều đó (5 >= 5 là đúng) và gán lại partition_idx về chỉ số của phân mảnh cuối cùng (5 - 1 = 4). Điều này đảm bảo giá trị rating cao nhất luôn nằm trong phân mảnh cuối cùng
        - TH2:
            - Mục đích: Xử lý các trường hợp rating là giá trị biên (ví dụ: 1.0, 2.0, 3.0, 4.0) mà phép tính ban đầu đã tính sai vì ta cần trừ chỉ số đi 1 để đúng phân mảnh.
            - partition_idx = max(partition_idx - 1, 0): Nếu điều kiện trên là đúng, nó sẽ lấy chỉ số đã tính ban đầu trừ đi 1

- Bước tiếp theo ta chỉ cần lấy ra insert vào đúng phân mảnh mà ta đã tìm ở bước trước bằng cách nối chuỗi prefix tên bảng + index

4.4. Phân mảnh theo vòng tròn (Round Robin Partitioning)

Mục đích : Dữ liệu được phân phối tuần tự và luân phiên vào các bảng con. Ví dụ: bản ghi đầu tiên vào rr_part0, bản ghi thứ hai vào rr_part1, bản ghi thứ ba vào rr_part2, sau đó quay lại rr_part0… Cách này đảm bảo rằng các bảng con có số lượng bản ghi tương đối đồng đều, hữu ích khi không có quy luật rõ ràng để chia theo giá trị.

4.4.1. Hàm roundrobinpartition(ratingstablename, numberofpartitions, openconnection) chia bảng ratings thành N bảng và phân phối theo Round-Robin.

```python
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):  1 usage (1 dynamic)  Hoa Ngo
    connection = openconnection
    cursor = connection.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    _create_rrobin_sequence(openconnection)

    partition_num = 0
    while partition_num < numberofpartitions:
        partition_name = RROBIN_TABLE_PREFIX + str(partition_num)

        table_creation_sql = f"CREATE TABLE IF NOT EXISTS {partition_name} (userid INTEGER, movieid INTEGER, rating FLOAT);"
        cursor.execute(table_creation_sql)

        data_insertion_sql = f"""INSERT INTO {partition_name}
                                 SELECT userid, movieid, rating
                                 FROM (SELECT userid, movieid, rating, ROW_NUMBER() OVER () AS row_seq
                                       FROM {ratingstablename}) AS numbered_data
                                 WHERE MOD(numbered_data.row_seq-1, {numberofpartitions}) = {partition_num};"""
        cursor.execute(data_insertion_sql)

        partition_num += 1

    cursor.close()
    connection.commit()
```

- Chi tiết các bước cài đặt hàm:
    - Đầu tiên ta sẽ thực hiện tạo sequence trong csdl nhằm mục đích để sử dụng sau này với roundrobininsert
    - Tương tự như range_partition, ta sẽ khởi tạo index ban đầu cho phân mảnh, bắt đầu từ 0
    - Chi tiết các bước trong vòng lặp:
        - Thực hiện nối chuỗi tạo tên table phân mảnh
        - Thực hiện tạo phân mảnh
        - Thực hiện chèn dữ liệu vào phân mảnh với câu lệnh truy vấn con và mệnh đề Where:

```
FROM (SELECT userid, movieid,
rating, ROW_NUMBER() OVER () AS
row_seq

FROM {ratingstablename}) AS
numbered_data

WHERE MOD(numbered_data.row_seq-1,
{numberofpartitions}) =
{partition_num};"""
```

+ ROW_NUMBER() OVER () AS row_seq: Đây là một hàm cửa sổ (window function). Nó duyệt qua tất cả các hàng trong bảng ratingstablename và gán cho mỗi hàng một số thứ tự tuần tự duy nhất,

bắt đầu từ 1. Kết quả là một bảng tạm thời có thêm một cột row_seq

```
WHERE MOD(numbered_data.row_seq-1, {numberofpartitions}) = {partition_num};"""
```

+ ta trừ row_seq đi 1 để đúng với index và mod nó với số lượng phân mảnh

+ thực hiện phép mod -> kết quả sẽ là chỉ số của phân mảnh - ví dụ ở đây là từ 0 -> 4

+ so sánh để lọc ra sao các record thuộc về phân mảnh hiện tại

- Sau đó ta tăng index lên để tiếp tục xây dựng phân mảnh tiếp theo

4.4.2. Hàm roundrobininsert(ratingstablename, userid, itemid, rating, openconnection) sẽ chèn bản ghi vào các bảng con theo tuần tự.

```
def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):  1 usage (1 dynamic)  Hoa Ngo
    connection = openconnection
    cursor = connection.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    _create_rrobin_sequence(openconnection)

    main_insert_sql = f"INSERT INTO {ratingstablename}(userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});"
    cursor.execute(main_insert_sql)

    total_partitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
    target_partition_index = _get_next_rrobin_index(total_partitions, openconnection)
    target_partition_name = RROBIN_TABLE_PREFIX + str(target_partition_index)

    partition_insert_sql = f"INSERT INTO {target_partition_name}(userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});"
    cursor.execute(partition_insert_sql)

    cursor.close()
    connection.commit()
```

- Chi tiết cách cài đặt hàm:
    - Đầu tiên để chắc chắn, ta sẽ tiến hành khởi tạo lại sequence nếu chưa tồn tại

```
def _create_rrobin_sequence(openconnection):  2 usages  Hoa Ngo
    connection = openconnection
    cursor = connection.cursor()
    sequence_sql = "CREATE SEQUENCE IF NOT EXISTS rrobin_seq START 1;"
    try:
        cursor.execute(sequence_sql)
        connection.commit()
    except Exception:
        connection.rollback()
    finally:
        cursor.close()
```

+Một SEQUENCE là một đối tượng trong database tự động tạo ra một chuỗi các số nguyên tuần tự (1, 2, 3, ...). Phục vụ cho việc tính toán xem record tiếp theo sẽ thuộc về phân mảnh nào

- Thực hiện tính toán xem hiện tại có bao nhiêu phân mảnh nhờ vào hàm count_partition
- Sau đó gọi đến hàm _get_next_rrobin_index(numberofpartitions, openconnection) để lấy giá trị index của phân mảnh cần insert

```
def _get_next_rrobin_index(numberofpartitions, openconnection):  1 usage  Hoa Ngo
    connection = openconnection
    cursor = connection.cursor()

    next_val_query = "SELECT nextval('rrobin_seq');"
    cursor.execute(next_val_query)
    sequence_value = cursor.fetchone()[0]
    cursor.close()

    partition_index = (sequence_value - 1) % numberofpartitions
    return partition_index
```

  + Hàm này có nhiệm vụ lấy ra value tiếp theo, 1 số điểm đặc biệt của hàm :
    + Nó lấy giá trị tiếp theo trong chuỗi số (lần đầu gọi trả về 1, lần hai trả về 2, ...).
    + Quan trọng: Nó tự động tăng bộ đếm của sequence lên 1. Đây là một hành động "nguyên tử" (atomic), nghĩa là dù nhiều người dùng gọi cùng lúc, mỗi người sẽ nhận được một số duy nhất.
    + Rất phù hợp vì độ phức tạp tính toán chỉ là O(1)
- Sau khi lấy được index của phân mảnh, ta chỉ cần insert đúng vào phân mảnh mục tiêu

4.5 Hàm count_partition:

```python
def count_partitions(prefix, openconnection):  2 usages  Hoa Ngo
    connection = openconnection
    cursor = connection.cursor()

    count_query = f"SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE '{prefix}%';"
    cursor.execute(count_query)
    partition_count = cursor.fetchone()[0]
    cursor.close()

    return partition_count
```

- Nhiệm vụ của hàm này là tính số lượng phân mảnh theo prefix truyền vào
- Tham số đầu vào là 1 prefix (tên bắt đầu của phân mảnh)
- Giá trị trả về là 1 số nguyên số lượng các phân mảnh đang có trong csdl

5. Bộ kiểm thử Assignment1Tester.py và TestHelper.py

Hàm testloadratings(MyAssignment, ratingstablename, filepath, openconnection, rowsininpfile) dùng để kiểm tra hàm loadratings tải đúng dữ liệu vào bảng ratings

```python
def testloadratings(MyAssignment, ratingstablename, filepath, openconnection, rowsininpfile):  1 usage
    """
    Tests the load ratings function
    :param ratingstablename: Argument for function to be tested
    :param filepath: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param rowsininpfile: Number of rows in the input file provided for assertion
    :return:Raises exception if any test fails
    """
    try:
        MyAssignment.loadratings(ratingstablename, filepath, openconnection)
        # Test 1: Count the number of rows inserted
        with openconnection.cursor() as cur:
            cur.execute('SELECT COUNT(*) from {0}'.format(ratingstablename))
            count = int(cur.fetchone()[0])
            if count != rowsininpfile:
                raise Exception(
                    'Expected {0} rows, but {1} rows in \'{2}\' table'.format( *args: rowsininpfile, count, ratingstablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

Cách hoạt động:

- Hàm testloadratings() gọi hàm loadratings() trong Interface1.py.

- Sau đó, nó thực thi SELECT COUNT(*) FROM ratings để đếm số hàng trong bảng ratings.
- Nếu số hàng khác ACTUAL_ROWS_IN_INPUT_FILE thì hàm ném ngoại lệ.
- Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

Hàm testrangepartition(MyAssignment, ratingstablename, n, openconnection, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE) kiểm tra việc phân mảnh theo khoảng giá trị rating có tuân thủ đúng logic và nguyên tắc không

```python
def testrangepartition(MyAssignment, ratingstablename, n, openconnection, partitionstartindex,  1 usage
                       ACTUAL_ROWS_IN_INPUT_FILE):
    """
    Tests the range partition function for Completness, Disjointness and Reconstruction
    :param ratingstablename: Argument for function to be tested
    :param n: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param partitionstartindex: Indicates how the table names are indexed. Do they start as rangepart1, 2 ... or rangepart0, 1, 2...
    :return:Raises exception if any test fails
    """

    try:
        MyAssignment.rangepartition(ratingstablename, n, openconnection)
        testrangeandrobinpartitioning(n, openconnection, RANGE_TABLE_PREFIX, partitionstartindex,
                                      ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRangePartition(ratingstablename, n, openconnection, RANGE_TABLE_PREFIX)
        return [True, None]
    except Exception as e:
        traceback.print_exc()
        return [False, e]
```

Cách thực hiện:

- Hàm testrangepartition() gọi hàm rangepartition() trong Interface1.py.
- Sau đó, nó thực thi: SELECT COUNT(table_name) FROM information_schema.tables WHERE table_name LIKE 'range_part%' để đếm số partition tables.
- SELECT COUNT(*) FROM (SELECT * FROM range_part0 UNION ALL SELECT * FROM range_part1...) để kiểm tra tổng số records.
- SELECT COUNT(*) FROM range_part{i} cho từng partition để verify distribution.
- Nếu số partition khác N, hoặc tổng records khác ACTUAL_ROWS_IN_INPUT_FILE, hoặc distribution không đúng theo range thì hàm ném ngoại lệ.
- Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

Hàm testrangeinsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex) kiểm tra việc chèn dữ liệu mới vào đúng range partition dựa trên giá trị rating.

```python
def testrangeinsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):  2 usages
    """
    Tests the range insert function by checking whether the tuple is inserted in he Expected table you provide
    :param ratingstablename: 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 = RANGE_TABLE_PREFIX + expectedtableindex
        MyAssignment.rangeinsert(ratingstablename, userid, itemid, rating, openconnection)
        if not testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
            raise Exception(
                'Range insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format( *args: userid, itemid, rating,
                                                                                               expectedtablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

Cách thực hiện:

- Hàm testrangeinsert() gọi hàm rangeinsert() trong Interface1.py.
- Sau đó, nó thực thi SELECT COUNT(*) FROM {expectedtablename} WHERE userid = {userid} AND movieid = {itemid} AND rating = {rating} để kiểm tra record đã được insert.
- Nếu count khác 1 (không tìm thấy record hoặc có duplicate) thì hàm ném ngoại lệ.
- Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

Hàm testroundrobinpartition(MyAssignment, ratingstablename, numberofpartitions, openconnection, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE) kiểm tra việc phân mảnh theo vòng tròn có phân phối đều dữ liệu không.

```python
def testroundrobinpartition(MyAssignment, ratingstablename, numberofpartitions, openconnection, 1 usage
                            partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    """
    Tests the round robin partitioning for Completness, Disjointness and Reconstruction
    :param ratingstablename: 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(ratingstablename, numberofpartitions, openconnection)
        testrangeandrobinpartitioning(numberofpartitions, openconnection, RROBIN_TABLE_PREFIX, partitionstartindex,
                                      ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRoundrobinPartition(ratingstablename, numberofpartitions, openconnection, RROBIN_TABLE_PREFIX)
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

Cách thực hiện:

- Hàm testroundrobinpartition() gọi hàm roundrobinpartition() trong Interface1.py.
- Sau đó, nó thực thi:
- SELECT COUNT(table_name) FROM information_schema.tables WHERE table_name LIKE 'rrobin_part%' để đếm số partition tables.
- SELECT COUNT(*) FROM (SELECT * FROM rrobin_part0 UNION ALL SELECT * FROM rrobin_part1...) để kiểm tra tổng số records.
- SELECT COUNT(*) FROM (SELECT *, ROW_NUMBER() OVER() FROM ratings) WHERE (row_number-1)%{N} = {i} để tính expected count cho partition i.
- SELECT COUNT(*) FROM rrobin_part{i} để đếm actual count trong partition i.
- Nếu số partition khác N, hoặc tổng records khác ACTUAL_ROWS_IN_INPUT_FILE, hoặc distribution không đúng theo round robin thì hàm ném ngoại lệ.
- Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

Hàm testroundrobininsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex) Kiểm tra việc chèn dữ liệu mới theo thứ tự round robin sequence có đúng không

```python
def testroundrobininsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):  3 usages
    """
    Tests the roundrobin insert function by checking whether the tuple is inserted in he Expected table you provide
    :param ratingstablename: Argument for function to be tested
    :param userid: Argument for function to be tested
    :param itemid: Argument for function to be tested
```

Cách thực hiện:

- Hàm testroundrobininsert() gọi hàm roundrobininsert() trong Interface1.py.
- Sau đó, nó thực thi SELECT COUNT(*) FROM {expectedtablename} WHERE userid = {userid} AND movieid = {itemid} AND rating = {rating} để kiểm tra record đã được insert vào đúng partition.
- Nếu count khác 1 (không tìm thấy record trong expected partition) thì hàm ném ngoại lệ.
- Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

IV. Thiết kế bảng cơ sở dữ liệu

1. Các bảng phân mảnh theo range (range_partX)

- Mục đích: Chia nhỏ bảng ratings thành nhiều bảng con dựa trên giá trị rating, mỗi bảng lưu các dòng có rating nằm trong một khoảng xác định.
- Cách tạo: Các bảng này có tên lần lượt là range_part0, range_part1, ..., range_partN (N là số phân mảnh).
- Cấu trúc:

```
CREATE TABLE range_partX (

    userid  INTEGER,
    movieid INTEGER,
   rating  FLOAT
);
```

- Ý nghĩa:
  - range_part0: Lưu các đánh giá có rating nằm trong khoảng nhỏ nhất (ví dụ: từ 0 đến 1 nếu chia 5 phân mảnh).
  - range_part1: Lưu các đánh giá có rating nằm trong khoảng tiếp theo, v.v.
- Các bảng này được tạo tự động bằng hàm rangepartition trong mã nguồn.

2. Các bảng phân mảnh theo round robin (rrobin_partX)

- Mục đích: Chia đều các dòng của bảng ratings vào nhiều bảng con theo thứ tự lượt vòng (round robin).
- Cách tạo: Các bảng này có tên lần lượt là rrobin_part0, rrobin_part1, ..., rrobin_partN.
- Cấu trúc:

```
CREATE TABLE rrobin_partX (
    userid  INTEGER,
   movieid INTEGER,
    rating  FLOAT
);
```

- Ý nghĩa:

    - Mỗi bảng lưu một phần dữ liệu, được phân bổ đều từ bảng ratings, giúp cân bằng tải khi truy vấn hoặc xử lý song song.
- Các bảng này được tạo tự động bằng hàm roundrobinpartition trong mã nguồn.

3. Sequence hỗ trợ phân mảnh round robin

- Mục đích: Hỗ trợ xác định thứ tự phân bổ dòng mới vào các bảng phân mảnh round robin.
- Được tạo tự động bằng lệnh:

  ```
  CREATE SEQUENCE IF NOT EXISTS rrobin_seq START 1;
  ```

- Không phải là bảng dữ liệu, nhưng là đối tượng quan trọng trong quản lý phân mảnh round robin.

Tóm tắt bảng và mục đích

Tên bảng/đối tượng	Mục đích	Cấu trúc/trường dữ liệu
ratings	Bảng chính lưu đánh giá	userid, movieid, rating
range_part0, 1, ... N	Phân mảnh theo khoảng rating	userid, movieid, rating
rrobin_part0, 1, ... N	Phân mảnh theo lượt vòng	userid, movieid, rating
rrobin_seq (sequence)	Hỗ trợ xác định thứ tự phân bổ round robin	-

V. KIỂM THỬ VÀ ĐÁNH GIÁ

Dưới đây là các trường hợp cần kiểm thử, kèm theo truy vấn kiểm tra và đánh giá về mặt thời gian thực thi dựa trên hệ thống phân mảnh cơ sở dữ liệu đã triển khai.

1. **Test Case 1: Load dữ liệu vào bảng (testloadratings)**

   Mô tả: Tiến hành tải dữ liệu từ file ratings.dat vào bảng ratings trong cơ sở dữ liệu PostgreSQL.

   Mục tiêu: Kiểm tra số dòng dữ liệu được nạp từ file vào bảng có đủ hay không, đảm bảo tính toàn vẹn dữ liệu.

   Tiêu chí Pass: Số dòng trong bảng sau khi tải dữ liệu bằng với số dòng của file dữ liệu (10,000,054 records).

   Truy vấn kiểm tra:

```
-- 1. Kiểm tra tổng số dòng được load
SELECT COUNT(*) as total_rows FROM ratings;

-- 2. Kiểm tra cấu trúc bảng được tạo đúng
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'ratings'
ORDER BY ordinal_position;

-- 3. Kiểm tra sample dữ liệu
SELECT userid, movieid, rating
FROM ratings
LIMIT 10;

-- 4. Kiểm tra range giá trị rating
SELECT
    MIN(rating) as min_rating,
    MAX(rating) as max_rating,
    AVG(rating) as avg_rating
FROM ratings;

-- 5. Kiểm tra không có NULL values
SELECT
```

```
    COUNT(*) - COUNT(userid) as null_userid,
    COUNT(*) - COUNT(movieid) as null_movieid,
    COUNT(*) - COUNT(rating) as null_rating
FROM ratings;
```

Thời gian thực thi:

```
loadratings function pass! Thời gian thực thi: 19.9856 giây
```

2. **Test Case 2: Kiểm tra phân mảnh theo khoảng (testrangepartition)**

Mô tả: Tiến hành phân mảnh ngang dữ liệu bảng đã tải dữ liệu bằng phương pháp phân mảnh theo khoảng (Range Partitioning) với 5 partitions.

Mục tiêu: Kiểm tra thuật toán phân mảnh theo khoảng hoạt động đúng theo logic:
Rating từ 0.0 đến 1.0: range_part0
Rating từ 1.0 đến 2.0: range_part1
Rating từ 2.0 đến 3.0: range_part2
Rating từ 3.0 đến 4.0: range_part3
Rating từ 4.0 đến 5.0: range_part4

Tiêu chí Pass: Đúng 5 bảng phân mảnh được tạo
Tổng số dòng trong các partitions = số dòng bảng gốc
Không có dữ liệu trùng lặp giữa các partitions
Dữ liệu được phân phối đúng theo range rating

Truy vấn kiểm tra:

```
-- 1. Kiểm tra số bảng phân mảnh được tạo
SELECT COUNT(*) as partition_count
FROM information_schema.tables
WHERE table_schema = 'public'
AND table_name LIKE 'range_part%';

-- 2. Kiểm tra tổng số dòng trong phân mảnh bằng số dòng trong bảng gốc
SELECT
```

```
    (SELECT COUNT(*) FROM range_part0) +
    (SELECT COUNT(*) FROM range_part1) +
    (SELECT COUNT(*) FROM range_part2) +
    (SELECT COUNT(*) FROM range_part3) +
    (SELECT COUNT(*) FROM range_part4) as
total_partition_rows,
    (SELECT COUNT(*) FROM ratings) as original_rows,
    ((SELECT COUNT(*) FROM range_part0) +
     (SELECT COUNT(*) FROM range_part1) +
     (SELECT COUNT(*) FROM range_part2) +
     (SELECT COUNT(*) FROM range_part3) +
     (SELECT COUNT(*) FROM range_part4)) = (SELECT
COUNT(*) FROM ratings) as is_complete;

-- 3. Kiểm tra phân phối dữ liệu theo từng partition
SELECT
    'range_part0' as partition_name,
    COUNT(*) as row_count,
    MIN(rating) as min_rating,
    MAX(rating) as max_rating
FROM range_part0
UNION ALL
SELECT
    'range_part1' as partition_name,
    COUNT(*) as row_count,
    MIN(rating) as min_rating,
    MAX(rating) as max_rating
FROM range_part1
UNION ALL
SELECT
    'range_part2' as partition_name,
    COUNT(*) as row_count,
    MIN(rating) as min_rating,
    MAX(rating) as max_rating
FROM range_part2
UNION ALL
SELECT
    'range_part3' as partition_name,
    COUNT(*) as row_count,
    MIN(rating) as min_rating,
```

```
    MAX(rating) as max_rating
FROM range_part3
UNION ALL
SELECT
    'range_part4' as partition_name,
    COUNT(*) as row_count,
    MIN(rating) as min_rating,
    MAX(rating) as max_rating
FROM range_part4;

-- 4. Kiểm tra không có bản ghi trùng giữa các partitions
-- Kiểm tra intersection giữa partition 0 và 1
SELECT COUNT(*) as common_records_0_1
FROM (
    SELECT userid, movieid, rating FROM range_part0
    INTERSECT
    SELECT userid, movieid, rating FROM range_part1
) AS common;

-- 5. Kiểm tra tính đúng đắn của range partitioning
SELECT
    COUNT(*) as invalid_records_part0
FROM range_part0
WHERE rating < 0.0 OR rating > 1.0;

SELECT
    COUNT(*) as invalid_records_part1
FROM range_part1
WHERE rating <= 1.0 OR rating > 2.0;

-- 6. Test reconstruction (khôi phục lại bảng gốc)
CREATE TEMP VIEW reconstructed_ratings AS (
    SELECT userid, movieid, rating FROM range_part0
    UNION ALL
    SELECT userid, movieid, rating FROM range_part1
    UNION ALL
    SELECT userid, movieid, rating FROM range_part2
    UNION ALL
    SELECT userid, movieid, rating FROM range_part3
    UNION ALL
```

```
    SELECT userid, movieid, rating FROM range_part4
);

SELECT COUNT(*) as reconstructed_count FROM
reconstructed_ratings;
```

Thời gian thực thi:

```
rangepartition function pass! Thời gian thực thi: 26.2462 giây
```

3. **Test Case 3: Kiểm tra thao tác chèn vào phân mảnh theo khoảng (testrangeinsert)**

Mô tả: Tiến hành chèn dữ liệu vào bảng chính và kiểm tra xem dữ liệu có được chèn vào đúng bảng phân mảnh theo range hay không.

Mục tiêu: Kiểm tra xem dữ liệu có được chèn vào bảng phân mảnh đúng như dự đoán không dựa trên giá trị rating.

Tiêu chí Pass: Dữ liệu được chèn vào đúng partition theo range rating
Ví dụ:
Record (100, 2, 3.0) phải vào range_part3 (rating=3.0 thuộc range 3.0-4.0)
Record (100, 2, 0.0) phải vào range_part0 (rating=0.0 thuộc range 0.0-1.0)

Truy vấn kiểm tra:

```
-- 1. Test case: Insert (userid=100, movieid=2, rating=3.0) ->
range_part3
-- Trước khi insert, đếm số records hiện tại
SELECT COUNT(*) as count_before_insert FROM range_part3
WHERE userid = 100 AND movieid = 2;

-- Sau khi gọi rangeinsert(100, 2, 3.0), kiểm tra
SELECT * FROM range_part3
WHERE userid = 100 AND movieid = 2 AND rating = 3.0;
```

```
-- 2. Test case: Insert (userid=100, movieid=2, rating=0.0) ->
range_part0
SELECT * FROM range_part0
WHERE userid = 100 AND movieid = 2 AND rating = 0.0;

-- 3. Kiểm tra dữ liệu KHÔNG xuất hiện ở partition sai
SELECT COUNT(*) as wrong_partition_count FROM range_part0
WHERE userid = 100 AND movieid = 2 AND rating = 3.0; -- Phải =
0

SELECT COUNT(*) as wrong_partition_count FROM range_part3
WHERE userid = 100 AND movieid = 2 AND rating = 0.0; -- Phải =
0

-- 4. Kiểm tra tổng số records sau insert
SELECT
   'range_part0' as partition,
   COUNT(*) as total_records
FROM range_part0
UNION ALL
SELECT
   'range_part1' as partition,
   COUNT(*) as total_records
FROM range_part1
UNION ALL
SELECT
   'range_part2' as partition,
   COUNT(*) as total_records
FROM range_part2
UNION ALL
SELECT
   'range_part3' as partition,
   COUNT(*) as total_records
FROM range_part3
UNION ALL
SELECT
   'range_part4' as partition,
   COUNT(*) as total_records
FROM range_part4;
```

-- 5. Verify function logic với các edge cases
-- Test rating = 1.0 (boundary case)
-- Test rating = 5.0 (max boundary)
-- Test rating = 2.5 (middle of range)

Thời gian thực thi:

```
rangeinsert function pass! Thời gian thực thi: 0.1510 giây
```

4. **Test Case 4: Kiểm tra phân mảnh vòng (testroundrobinpartition)**

Mô tả: Tiến hành phân mảnh ngang dữ liệu bảng đã tải dữ liệu bằng phương pháp phân mảnh vòng (Round Robin Partitioning) với 5 partitions.

Mục tiêu: Kiểm tra thuật toán phân mảnh vòng hoạt động đúng theo logic:
Records được phân phối tuần tự theo ROW_NUMBER() % 5
rrobin_part0: records có (row_number-1) % 5 = 0
rrobin_part1: records có (row_number-1) % 5 = 1
Và tiếp tục cho đến rrobin_part4

Tiêu chí Pass: Đúng 5 bảng phân mảnh được tạo
Tổng số dòng trong các partitions = số dòng bảng gốc
Phân phối tương đối đều giữa các partitions (chênh lệch không quá 1 record)
Sequence rrobin_seq được tạo và hoạt động đúng

Truy vấn kiểm tra:

```
-- 1. Kiểm tra số bảng phân mảnh được tạo
SELECT COUNT(*) as rrobin_partition_count
FROM information_schema.tables
WHERE table_schema = 'public'
AND table_name LIKE 'rrobin_part%';

-- 2. Kiểm tra sequence được tạo
SELECT sequence_name, increment, minimum_value,
maximum_value, start_value
FROM information_schema.sequences
WHERE sequence_name = 'rrobin_seq';
```

```
-- 3. Kiểm tra tổng số dòng các partitions = bảng gốc
SELECT
    (SELECT COUNT(*) FROM rrobin_part0) +
    (SELECT COUNT(*) FROM rrobin_part1) +
    (SELECT COUNT(*) FROM rrobin_part2) +
    (SELECT COUNT(*) FROM rrobin_part3) +
    (SELECT COUNT(*) FROM rrobin_part4) as total_rrobin_rows,
    (SELECT COUNT(*) FROM ratings) as original_rows,
    ((SELECT COUNT(*) FROM rrobin_part0) +
     (SELECT COUNT(*) FROM rrobin_part1) +
     (SELECT COUNT(*) FROM rrobin_part2) +
     (SELECT COUNT(*) FROM rrobin_part3) +
     (SELECT COUNT(*) FROM rrobin_part4)) = (SELECT
COUNT(*) FROM ratings) as is_complete;

-- 4. Kiểm tra phân phối đều giữa các partitions
WITH partition_counts AS (
    SELECT 'rrobin_part0' as partition_name, COUNT(*) as
row_count FROM rrobin_part0
    UNION ALL
    SELECT 'rrobin_part1' as partition_name, COUNT(*) as
row_count FROM rrobin_part1
    UNION ALL
    SELECT 'rrobin_part2' as partition_name, COUNT(*) as
row_count FROM rrobin_part2
    UNION ALL
    SELECT 'rrobin_part3' as partition_name, COUNT(*) as
row_count FROM rrobin_part3
    UNION ALL
    SELECT 'rrobin_part4' as partition_name, COUNT(*) as
row_count FROM rrobin_part4
)
SELECT
    partition_name,
    row_count,
    row_count - LAG(row_count) OVER (ORDER BY
partition_name) as difference_from_prev,
    MAX(row_count) OVER () - MIN(row_count) OVER () as
max_difference
```

```
FROM partition_counts;

-- 5. Verify round robin logic bằng cách tái tạo
WITH numbered_ratings AS (
   SELECT userid, movieid, rating, ROW_NUMBER() OVER () as
rn
   FROM ratings
),
expected_distribution AS (
   SELECT
      CASE (rn-1) % 5
         WHEN 0 THEN 'rrobin_part0'
         WHEN 1 THEN 'rrobin_part1'
         WHEN 2 THEN 'rrobin_part2'
         WHEN 3 THEN 'rrobin_part3'
         WHEN 4 THEN 'rrobin_part4'
      END as expected_partition,
      COUNT(*) as expected_count
   FROM numbered_ratings
   GROUP BY (rn-1) % 5
)
SELECT * FROM expected_distribution ORDER BY
expected_partition;

-- 6. Kiểm tra không có overlap giữa partitions
SELECT COUNT(*) as common_records_0_1
FROM (
   SELECT userid, movieid, rating FROM rrobin_part0
   INTERSECT
   SELECT userid, movieid, rating FROM rrobin_part1
) AS common;

-- 7. Sample data từ mỗi partition
SELECT 'rrobin_part0' as partition, userid, movieid, rating FROM
rrobin_part0 LIMIT 5
UNION ALL
SELECT 'rrobin_part1' as partition, userid, movieid, rating FROM
rrobin_part1 LIMIT 5
UNION ALL
```

```
SELECT 'rrobin_part2' as partition, userid, movieid, rating FROM
rrobin_part2 LIMIT 5;
```

Thời gian thực thi:

```
roundrobinpartition function pass! Thời gian thực thi: 50.9760 giây
```

5. **Test Case 5: Kiểm tra thao tác chèn vào phân mảnh vòng (testroundrobininsert)**

   Mô tả: Tiến hành chèn dữ liệu vào bảng chính và kiểm tra xem dữ liệu có được chèn vào đúng bảng phân mảnh theo sequence round robin hay không.

   Mục tiêu: Kiểm tra xem dữ liệu có được chèn vào bảng phân mảnh đúng như dự đoán không dựa trên giá trị sequence rrobin_seq.

   Tiêu chí Pass: Dữ liệu được chèn vào đúng partition theo sequence value
   Insert thứ 1: (100, 1, 3) vào rrobin_part0 (sequence % 5 = 0)
   Insert thứ 2: (100, 1, 3) vào rrobin_part1 (sequence % 5 = 1)
   Insert thứ 3: (100, 1, 3) vào rrobin_part2 (sequence % 5 = 2)

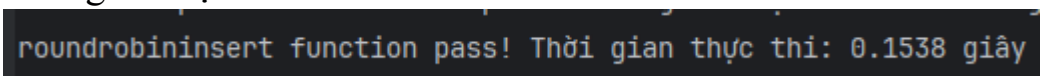
   Truy vấn kiểm tra:

```
-- 1. Kiểm tra current value của sequence trước khi insert
SELECT currval('rrobin_seq') as current_sequence_value;

-- 2. Test case: Insert lần 1 (userid=100, movieid=1, rating=3) -> rrobin_part0
SELECT * FROM rrobin_part0
WHERE userid = 100 AND movieid = 1 AND rating = 3.0;

-- 3. Test case: Insert lần 2 (userid=100, movieid=1, rating=3) -> rrobin_part1
SELECT * FROM rrobin_part1
WHERE userid = 100 AND movieid = 1 AND rating = 3.0;

-- 4. Test case: Insert lần 3 (userid=100, movieid=1, rating=3) -> rrobin_part2
SELECT * FROM rrobin_part2
WHERE userid = 100 AND movieid = 1 AND rating = 3.0;

-- 5. Kiểm tra sequence value sau mỗi lần insert
SELECT currval('rrobin_seq') as sequence_after_inserts;
```

```
-- 6. Verify không có dữ liệu ở partition sai
-- Insert 1 không được xuất hiện ở part1, part2
SELECT COUNT(*) as should_be_zero FROM rrobin_part1
WHERE userid = 100 AND movieid = 1 AND rating = 3.0
AND userid || movieid || rating = '100' || '1' || '3';

-- 7. Kiểm tra pattern sequence với multiple inserts
WITH insert_sequence AS (
   SELECT
      nextval('rrobin_seq') as seq_val,
      (currval('rrobin_seq') - 1) % 5 as target_partition
   FROM generate_series(1, 10)
)
SELECT seq_val, target_partition,
   CASE target_partition
      WHEN 0 THEN 'rrobin_part0'
      WHEN 1 THEN 'rrobin_part1'
      WHEN 2 THEN 'rrobin_part2'
      WHEN 3 THEN 'rrobin_part3'
      WHEN 4 THEN 'rrobin_part4'
   END as expected_partition_name
FROM insert_sequence;

-- 8. Kiểm tra tăng số lượng records sau insert
SELECT
   'rrobin_part0' as partition,
   COUNT(*) as total_records,
   COUNT(*) FILTER (WHERE userid = 100 AND movieid = 1) as
test_records
FROM rrobin_part0
UNION ALL
SELECT
   'rrobin_part1' as partition,
   COUNT(*) as total_records,
   COUNT(*) FILTER (WHERE userid = 100 AND movieid = 1) as
test_records
FROM rrobin_part1
UNION ALL
SELECT
```

```
    'rrobin_part2' as partition,
    COUNT(*) as total_records,
    COUNT(*) FILTER (WHERE userid = 100 AND movieid = 1) as
test_records
FROM rrobin_part2;

-- 9. Test concurrent insert behavior (nếu cần)
-- Kiểm tra sequence safety trong môi trường concurrent
BEGIN;
SELECT nextval('rrobin_seq') as seq1;
SELECT nextval('rrobin_seq') as seq2;
SELECT nextval('rrobin_seq') as seq3;
ROLLBACK;

-- 10. Reset sequence để test lại từ đầu (nếu cần)
-- SELECT setval('rrobin_seq', 1, false);
```

Thời gian thực thi:



V. KẾT LUẬN

Bài tập đã giúp tôi thực hành và hiểu rõ cách xử lý dữ liệu lớn với cơ sở dữ liệu quan hệ thông qua việc phân vùng (partitioning) bằng range và round-robin. Việc cài đặt các hàm loadratings, rangepartition, rangeinsert, roundrobinpartition, và roundrobininsert cho thấy tầm quan trọng của việc tổ chức dữ liệu hiệu quả để tối ưu hóa hiệu suất truy vấn và cập nhật trong hệ quản trị cơ sở dữ liệu.

1. Kết quả đạt được

- Hoàn thành đầy đủ tất cả yêu cầu bài toán.
- Xử lý thành công tập dữ liệu lớn (10 triệu bản ghi).
- Thiết kế và cài đặt hệ thống phân vùng đầy đủ logic.
- Xây dựng framework kiểm thử tự động và chính xác.
- Tối ưu hiệu suất hệ thống thông qua kỹ thuật xử lý theo lô.

2. Kiến thức và kỹ năng học được

- Hiểu rõ lý thuyết phân vùng dữ liệu (Completeness, Disjointness, Reconstruction) và kỹ thuật Range & Round Robin.
- Lập trình cơ sở dữ liệu với Python (psycopg2), xử lý kết nối, giao dịch, lỗi và tiền xử lý file lớn.
- Thiết kế kiểm thử toàn diện, tách biệt rõ logic xử lý và kiểm thử.
- Nắm vững kiến thức thiết kế hệ thống: phân vùng ngang, cân bằng tải, tối ưu hiệu năng và quản lý song song.
- Thực hành kỹ thuật phần mềm: thiết kế mô-đun rõ ràng, ghi chú code đầy đủ và sử dụng Git hiệu quả cho làm việc nhóm.

3. Khó khăn gặp phải & Bài học rút ra

- Lỗi logic tại ranh giới phân vùng Range; xử lý bằng điều kiện chính xác, tránh trùng lặp dữ liệu.
- Kiểm thử phức tạp do phải xác minh đầy đủ thuộc tính phân vùng; giải quyết bằng thiết kế kiểm thử mô-đun.

- Làm việc nhóm hiệu quả nhờ phân chia nhiệm vụ rõ, chuẩn hóa mã nguồn và kiểm thử liên tục.
- Biết cách gỡ lỗi: phân tích log, kiểm tra dữ liệu trung gian, phân rã vấn đề để tìm nguyên nhân.
- Đọc kỹ tài liệu chính thức, ghi lại quyết định kỹ thuật để phục vụ cải tiến về sau.

4. Tổng kết

Dự án giúp chúng em củng cố thêm về kiến thức về cơ sở dữ liệu phân tán, rèn luyện tư duy lập trình hướng giải pháp, xử lý dữ liệu lớn thực tế, triển khai phân mảnh hiệu quả và nâng cao kỹ năng làm việc nhóm, kiểm thử hệ thống, quản lý dự án.

V. TÀI LIỆU THAM KHẢO

- Tài liệu chính thức PostgreSQL: https://www.postgresql.org/docs/
- psycopg2 Documentation (Python - PostgreSQL connector): https://www.psycopg.org/docs/
- MovieLens dataset (gốc dữ liệu ratings.dat): https://grouplens.org/datasets/movielens/
- Hướng dẫn xử lý file lớn trong Python: https://realpython.com/read-write-files-python/
- Tài liệu về partitioning trong PostgreSQL: https://www.postgresql.org/docs/current/ddl-partitioning.html
- Các bài giảng và bài tập từ khóa học Data Management (CMU hoặc tương đương).