HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
CƠ SỞ DỮ LIỆU PHÂN TÁN

Giảng viên hướng dẫn : Kim Ngọc Bách
Lớp : 10
Nhóm : 08

Các thành viên

Dương Thị Hồng Hạnh	**B22DCCN276**
Nguyễn Thị Thu Phương	**B22DCCN637**
Ngô Thanh Vân	**B22DCCN891**

Hà Nội – 2025

I. Phân công công việc nhóm

Công việc	Người thực hiện
Code	Phương, Hạnh, Vân
Báo cáo phần I+II	Vân
Báo cáo phần III	Hạnh
Báo cáo phần IV	Phương
Báo cáo phần V	Hạnh

II. Giới thiệu chung

1.Giới thiệu chung về phân mảnh dữ liệu và phương pháp phân mảnh ngang, phân mảnh tròn trong hệ quản trị csdl phân tán.

1.1. Giới thiệu chung về phân mảnh dữ liệu

Phân mảnh dữ liệu (Data Fragmentation) là một kỹ thuật trong hệ quản trị cơ sở dữ liệu phân tán (Distributed Database Management System - DDBMS) nhằm chia nhỏ một cơ sở dữ liệu lớn thành các phần nhỏ hơn, gọi là mảnh (fragments), để lưu trữ và xử lý trên nhiều nút (nodes) hoặc vị trí địa lý khác nhau. Mục tiêu chính của phân mảnh dữ liệu là:

a. Tăng hiệu suất: Phân mảnh giúp giảm thời gian truy vấn bằng cách chỉ truy cập vào các mảnh dữ liệu liên quan thay vì toàn bộ cơ sở dữ liệu.

b. Tăng tính khả dụng: Dữ liệu được phân phối trên nhiều nút, nếu một nút gặp sự cố, các nút khác vẫn có thể hoạt động.

c. Tối ưu hóa tài nguyên: Phân mảnh cho phép phân bổ dữ liệu gần với nơi nó được sử dụng nhiều nhất, giảm chi phí truyền tải dữ liệu qua mạng.

d. Hỗ trợ mở rộng: Dễ dàng mở rộng hệ thống bằng cách thêm các nút mới mà không ảnh hưởng đến toàn bộ cơ sở dữ liệu.

Phân mảnh dữ liệu thường được thực hiện dựa trên ba phương pháp chính: phân mảnh ngang, phân mảnh dọc và phân mảnh hỗn hợp. Trong đó, phân mảnh ngang là một trong những phương pháp phổ biến nhất.

1.2.Phương pháp phân mảnh ngang

1.2.1. Khái niệm

Phân mảnh ngang (Horizontal Fragmentation) là kỹ thuật chia một bảng (table) trong cơ sở dữ liệu thành các tập hợp con (fragments) dựa trên các hàng (rows) của bảng, theo một điều kiện hoặc tiêu chí nhất định. Mỗi mảnh chứa một tập hợp con các hàng của bảng gốc, nhưng vẫn giữ nguyên cấu trúc lược đồ (schema) của bảng.

1.2.2.Đặc điểm của phân mảnh ngang:

a.Tiêu chí phân mảnh: Các hàng của bảng được chia dựa trên giá trị của một hoặc nhiều cột.

b.Tính độc lập: Mỗi mảnh là một tập hợp con độc lập, nhưng khi gộp tất cả các mảnh lại, ta sẽ tái tạo được bảng gốc (tính đầy đủ).

c.Ứng dụng: Thường được sử dụng khi dữ liệu cần được phân phối theo các đặc điểm địa lý, tổ chức, hoặc nhu cầu truy cập. Ví dụ, dữ liệu của các chi nhánh công ty được lưu trữ tại các địa điểm tương ứng.

1.2.3.Các loại phân mảnh ngang:

a. Phân mảnh ngang sơ cấp (Primary Horizontal Fragmentation):

- Dựa trên một điều kiện đơn giản (predicate) trên một hoặc nhiều cột.

b. Phân mảnh ngang dẫn xuất (Derived Horizontal Fragmentation):

- Dựa trên mối quan hệ giữa các bảng, thường sử dụng khóa ngoại (foreign key).

1.2.4.Quy trình thực hiện phân mảnh ngang:

a.Xác định tiêu chí phân mảnh: Dựa trên các yêu cầu ứng dụng, chọn cột và điều kiện để chia dữ liệu.

b.Xây dựng các mảnh: Tạo các mảnh bằng cách áp dụng các điều kiện lọc (ví dụ: `WHERE` clause trong SQL).

c.Phân bổ mảnh: Gán các mảnh đến các nút trong hệ thống phân tán, dựa trên nhu cầu truy cập hoặc vị trí địa lý.

d.Đảm bảo tính đầy đủ và bất trùng: Mỗi hàng trong bảng gốc phải thuộc đúng một mảnh (bất trùng) và tất cả các hàng phải được bao phủ bởi các mảnh (đầy đủ).

1.2.5. Ưu điểm của phân mảnh ngang:

- Tăng hiệu suất truy vấn cục bộ, đặc biệt khi các truy vấn chỉ cần truy cập một mảnh cụ thể.

- Giảm chi phí truyền dữ liệu qua mạng vì dữ liệu được lưu trữ gần nơi sử dụng.

- Dễ dàng quản lý dữ liệu theo các tiêu chí cụ thể (ví dụ: theo khu vực, phòng ban).

1.2.6. Nhược điểm:

- Phức tạp trong việc quản lý các truy vấn toàn cục (global queries) vì cần phải gộp dữ liệu từ nhiều mảnh.

- Nếu tiêu chí phân mảnh không được chọn cẩn thận, có thể dẫn đến phân phối dữ liệu không đồng đều, gây mất cân bằng tải.

1.3. Phân mảnh tròn

1.3.1. Khái niệm

Trong ngữ cảnh cơ sở dữ liệu phân tán, phân mảnh tròn (round-robin partitioning) là một phương pháp phân vùng dữ liệu, thường được sử dụng trong các hệ thống như cơ sở dữ liệu NoSQL (ví dụ: Cassandra) để phân phối dữ liệu đều trên các nút (nodes) trong cụm (cluster). Phương pháp này không dựa trên giá trị của dữ liệu mà phân bổ các bản ghi (records) tuần tự theo vòng tròn giữa các nút có sẵn.

1.3.2.Đặc điểm của phân mảnh tròn:

a.Cách hoạt động:

- Dữ liệu được ghi vào các nút theo thứ tự tuần hoàn (round-robin). Mỗi bản ghi mới được phân bổ lần lượt cho từng nút trong cụm, ví dụ: bản ghi 1 vào nút 1, bản ghi 2 vào nút 2, bản ghi 3 vào nút 3, rồi quay lại nút 1 cho bản ghi 4, và cứ tiếp tục như vậy.

- Không phụ thuộc vào giá trị của dữ liệu hay khóa (key), mà chỉ dựa trên thứ tự ghi.

b.Ưu điểm:

- Phân phối đều dữ liệu: Đảm bảo dữ liệu được phân bổ đồng đều trên các nút, tránh tình trạng một nút bị quá tải.

- Đơn giản: Thuật toán dễ triển khai, không cần tính toán phức tạp như hàm băm (hash function).

- Hiệu quả cho ghi dữ liệu: Phù hợp với các hệ thống có tần suất ghi cao và cần phân phối tải nhanh chóng.

c.Nhược điểm:

- Khó khăn trong truy vấn: Do dữ liệu được phân bổ tuần tự mà không dựa trên giá trị khóa, việc xác định vị trí của một bản ghi cụ thể để đọc hoặc truy vấn trở nên khó khăn hơn. Cần thêm cơ chế ánh xạ hoặc metadata để theo dõi vị trí dữ liệu.

- Tái cân bằng phức tạp: Khi thêm hoặc xóa một nút khỏi cụm, cần tái phân phối dữ liệu, có thể gây xáo trộn lớn và tốn tài nguyên.

- Không tối ưu cho truy vấn theo khóa: Nếu ứng dụng cần truy xuất dữ liệu dựa trên một giá trị khóa cụ thể, phân mảnh tròn không hiệu quả bằng các phương pháp như phân mảnh dựa trên khóa (key-based partitioning).

d.Ứng dụng thực tế:

- Phân mảnh tròn thường được sử dụng trong các hệ thống yêu cầu phân phối tải đồng đều, chẳng hạn như cơ sở dữ liệu hướng cột (column-oriented) như Cassandra. Tuy nhiên, trong Cassandra, phân mảnh tròn thường được kết hợp với Consistent Hashing để cải thiện khả năng xác định vị trí dữ liệu và xử lý tái cân bằng khi mở rộng hoặc thu hẹp cụm.

2. Ý nghĩa của việc phân mảnh dữ liệu trong hệ quản trị csdl phân tán

Phân mảnh dữ liệu (Data Fragmentation) trong hệ quản trị cơ sở dữ liệu phân tán (Distributed Database Management System - DDBMS) đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất, tăng tính khả dụng và hỗ trợ quản lý dữ liệu hiệu quả. Dưới đây là các ý nghĩa chính của phân mảnh dữ liệu:

2.1.Tăng hiệu suất truy vấn:

- Phân mảnh dữ liệu cho phép các truy vấn chỉ truy cập vào một tập hợp con (fragment) của dữ liệu thay vì toàn bộ cơ sở dữ liệu. Điều này giảm thời gian xử lý và tài nguyên cần thiết.

2.2.Giảm chi phí truyền tải dữ liệu:

- Dữ liệu được lưu trữ gần nơi nó được sử dụng nhiều nhất (ví dụ: dữ liệu của khách hàng tại Hà Nội được lưu trên máy chủ tại Hà Nội). Điều này giảm thiểu lượng dữ liệu cần truyền qua mạng, tiết kiệm băng thông và thời gian.

2.3.Tăng tính khả dụng và độ tin cậy:

- Trong hệ phân tán, dữ liệu được phân phối trên nhiều nút. Nếu một nút gặp sự cố, các nút khác vẫn có thể cung cấp dữ liệu, đảm bảo hệ thống tiếp tục hoạt động.

- Phân mảnh cũng hỗ trợ sao chép dữ liệu (replication), giúp tăng khả năng chịu lỗi.

2.4.Hỗ trợ mở rộng hệ thống (Scalability):

- Phân mảnh cho phép dễ dàng mở rộng hệ thống bằng cách thêm các nút mới để lưu trữ các mảnh dữ liệu bổ sung. Điều này đặc biệt quan trọng trong các ứng dụng có lượng dữ liệu tăng trưởng nhanh.

2.5.Tối ưu hóa tài nguyên:

- Phân mảnh giúp phân bổ dữ liệu dựa trên nhu cầu sử dụng, tránh lãng phí tài nguyên trên các nút không cần thiết.

2.6.Hỗ trợ quản lý dữ liệu cục bộ:

- Phân mảnh cho phép các địa điểm hoặc phòng ban quản lý dữ liệu của riêng họ, tăng tính độc lập và giảm sự phụ thuộc vào một hệ thống tập trung.

- Điều này đặc biệt hữu ích trong các tổ chức lớn với nhiều chi nhánh hoặc khu vực địa lý.

2.7.Tăng tính linh hoạt trong thiết kế hệ thống:

- Phân mảnh dữ liệu (ngang, dọc, hoặc hỗn hợp) có thể được thiết kế để phù hợp với đặc điểm của ứng dụng, như truy vấn thường xuyên, phân bố địa lý, hoặc yêu cầu bảo mật.

Lưu ý về thách thức:

Mặc dù có nhiều lợi ích, phân mảnh dữ liệu cũng đi kèm với các thách thức như:

- Phức tạp trong quản lý: Việc duy trì tính nhất quán, đồng bộ hóa dữ liệu giữa các mảnh, và xử lý các truy vấn toàn cục (global queries) đòi hỏi cơ chế phức tạp.
- Chi phí thiết kế: Xác định tiêu chí phân mảnh phù hợp và phân bổ mảnh đến các nút đòi hỏi phân tích kỹ lưỡng.
- Tái cân bằng dữ liệu: Khi thêm hoặc xóa nút, dữ liệu cần được tái phân phối, có thể gây tốn tài nguyên.

Kết luận:

Phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu phân tán mang lại nhiều lợi ích quan trọng như cải thiện hiệu suất, giảm chi phí mạng, tăng khả năng mở rộng và tính khả dụng. Tuy nhiên, cần cân nhắc kỹ lưỡng trong việc thiết kế và quản lý để đảm bảo hiệu quả tối ưu. Phương pháp này đặc biệt phù hợp với các ứng dụng lớn, phân tán về mặt địa lý hoặc có yêu cầu truy cập dữ liệu cục bộ cao

3. Mục tiêu bài tập lớn

Mục tiêu của bài tập lớn là hiểu và thực hành các kỹ thuật phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu phân tán, thông qua việc mô phỏng và triển khai các phương pháp phân mảnh ngang bằng ngôn ngữ lập trình Python và hệ quản trị cơ sở dữ liệu mã nguồn mở như PostgreSQL hoặc MySQL.

Cụ thể, bài tập hướng đến các mục tiêu sau:

- Hiểu rõ lý thuyết về các phương pháp phân mảnh dữ liệu, đặc biệt là phân mảnh ngang theo phạm vi (Range Partitioning) và vòng tròn (Round-Robin Partitioning).
- Cài đặt được các hàm xử lý dữ liệu bao gồm:
    - Tải dữ liệu vào cơ sở dữ liệu.
    - Phân mảnh dữ liệu theo hai phương pháp đã nêu.
    - Chèn dữ liệu mới vào các phân mảnh đúng cách.

- Làm quen với thao tác thực tế trên dữ liệu lớn thông qua bộ dữ liệu MovieLens có quy mô hàng triệu dòng.

- Rèn luyện kỹ năng lập trình, làm việc nhóm, và sử dụng công cụ quản lý mã nguồn (GitHub).

- Trình bày và đánh giá được kết quả thông qua báo cáo mô tả cách triển khai, phân tích kết quả và phân công công việc nhóm.

Thông qua bài tập này sẽ củng cố kiến thức lý thuyết về hệ cơ sở dữ liệu phân tán, đồng thời áp dụng được vào một tình huống thực tiễn để thấy được hiệu quả và thách thức của việc phân mảnh dữ liệu trong môi trường hệ thống phân tán.

4. Tổng quan về dữ liệu MovieLen (ratings.dat)

MovieLens là một hệ thống đề xuất phim nổi tiếng được phát triển và duy trì bởi GroupLens Research – một nhóm nghiên cứu về hệ thống đề xuất tại Đại học Minnesota, Hoa Kỳ. MovieLens không chỉ phục vụ người dùng với những gợi ý phim cá nhân hóa mà còn đóng vai trò là một nguồn dữ liệu nghiên cứu tiêu chuẩn trong các lĩnh vực như khai phá dữ liệu, học máy, hệ thống gợi ý, và quản lý cơ sở dữ liệu.

4.1. Nội dung dữ liệu

Dữ liệu MovieLens bao gồm:

- Thông tin đánh giá phim: người dùng đánh giá các bộ phim theo thang điểm từ 0.5 đến 5.0 sao, với bước chia là 0.5.

- Thông tin về phim: bao gồm tên phim, năm phát hành, và thể loại.

- Thông tin người dùng (ẩn danh): đôi khi bao gồm giới tính, độ tuổi, nghề nghiệp hoặc vị trí địa lý, tùy bộ dữ liệu.

4.2. Các bộ dữ liệu phổ biến

GroupLens phát hành nhiều bộ dữ liệu MovieLens với quy mô khác nhau, phù hợp cho các mục đích nghiên cứu và thử nghiệm:

- MovieLens 100K: chứa 100.000 đánh giá từ 943 người dùng đối với 1.682 bộ phim.

- MovieLens 1M: 1 triệu đánh giá, 6.000 phim, hơn 6.000 người dùng.

- MovieLens 10M: 10 triệu đánh giá, 10.000 bộ phim, 72.000 người dùng – đây là bộ dữ liệu được sử dụng trong bài tập lớn này.

4.3. Định dạng dữ liệu

Trong bài tập lớn, sinh viên sử dụng tệp ratings.dat từ bộ dữ liệu MovieLens 10M. Tệp này chứa các dòng dữ liệu theo định dạng.

Ví dụ:

- UserID: ID người dùng.

- MovieID: ID bộ phim.

- Rating: Điểm đánh giá (float, từ 0.5 đến 5.0).

- Timestamp: Thời điểm đánh giá (tính bằng giây từ ngày 01/01/1970 theo UTC).

4.4. Ý nghĩa và ứng dụng

MovieLens là bộ dữ liệu thực tế với quy mô lớn, mang tính chất đại diện cao, thường được sử dụng để:

- Xây dựng và thử nghiệm hệ thống gợi ý phim.

- Đánh giá thuật toán học máy và xử lý dữ liệu lớn.

- Nghiên cứu các phương pháp phân mảnh, phân phối và tối ưu hóa truy vấn trong hệ cơ sở dữ liệu phân tán.

III. Mô tả đề bài

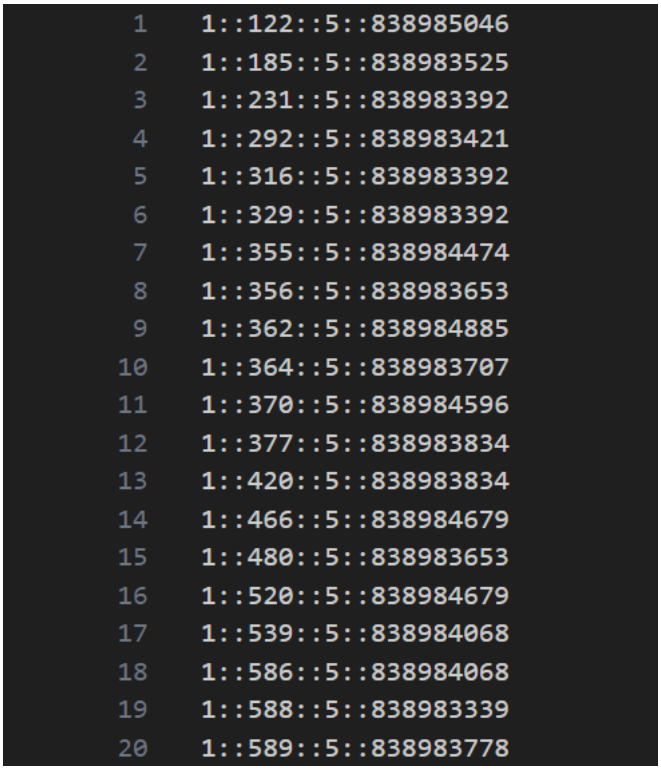
3.1.Dữ liệu đầu vào

Dữ liệu đầu vào là một tập dữ liệu đánh giá phim được thu thập từ trang web MovieLens (http://movielens.org): *ratings.dat*

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*

Các đánh giá được thực hiện trên thang điểm 5 sao, có thể chia nửa sao. Dấu thời (Timestamp) là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970.

Hai mươi dòng đầu tiên của tệp *ratings.dat*



3.2.Mục tiêu đầu ra

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

Tạo một tập các hàm Python để tải dữ liệu đầu vào vào một bảng quan hệ, phân mảnh bảng này bằng phương pháp phân mảnh ngang theo khoảng và phân mảnh nganh kiểu vòng tròn và chèn các bộ dữ liệu mới vào đúng phân mảnh đã tạo.

IV. Thiết kế và cài đặt phân mảnh

1. Ngôn ngữ và công nghệ sử dụng

-Ngôn ngữ lập trình: Python 3.12x

-Thư viện cơ sở dữ liệu: psycopg2 (kết nối PostgreSQL)

-Hệ quản trị CSDL: PostgreSQL

-Môi trường phát triển: Visual Studio Code

-Các công nghệ hỗ trợ:

- Git/GitHub: Quản lý phiên bản

2. Các hàm cài đặt chính

2.1.Hàm getopenconnection():

```
def getopenconnection(user='postgres', password='123456', dbname='postgres'):
    try:
        return psycopg2.connect(
            dbname=dbname,
            user=user,
            host='localhost',
            password=password,
            port=5432,
        )
    except Exception as e:
        print(f"Error connecting to DB: {e}")
        raise
```

-Ý nghĩa:Kết nối đến một cơ sở dữ liệu PostgreSQL.

-Cách hoạt động:

+Kết nối tới database postgres mặc định.

+Kiểm tra xem dbname đã tồn tại chưa bằng cách truy vấn bảng pg_database.

+Nếu chưa tồn tại → CREATE DATABASE.

+Đóng kết nối sau khi xong.

2.2.Hàm create_db()

```
def create_db(dbname, user='postgres', password='123456'):
    con = None
    cur = None
    try:
        con = getopenconnection(user=user, password=password, dbname='postgres')
        con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
        cur = con.cursor()
        cur.execute('SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname=%s', (dbname,))
        count = cur.fetchone()[0]
        if count == 0:
            cur.execute(f'CREATE DATABASE {dbname}')
            print(f"Database {dbname} created.")
        else:
            print(f"Database {dbname} already exists.")
    except Exception as e:
        print(f"Error creating database: {e}")
        raise
    finally:
        if cur:
            cur.close()
        if con:
            con.close()
```

-Ý nghĩa:Kết nối đến một cơ sở dữ liệu PostgreSQL.

-Cách hoạt động:

+Sử dụng psycopg2.connect để kết nối tới database.

+Trả về đối tượng connection.

2.3.Hàm loadratings()

```python
def loadratings(ratingstablename, ratingsfilepath, openconnection):
    start = time.time()
    cur = None
    try:
        cur = openconnection.cursor()
        cur.execute(f"DROP TABLE IF EXISTS {ratingstablename} CASCADE;")
        cur.execute(f"""
            CREATE TABLE {ratingstablename} (
                userid INTEGER,
                extra1 CHAR,
                movieid INTEGER,
                extra2 CHAR,
                rating FLOAT,
                extra3 CHAR,
                timestamp BIGINT
            );
        """)
        with open(ratingsfilepath, 'r') as f:
            cur.copy_from(f, ratingstablename, sep=':')
        cur.execute(f"""
            ALTER TABLE {ratingstablename}
            DROP COLUMN extra1,
            DROP COLUMN extra2,
            DROP COLUMN extra3,
            DROP COLUMN timestamp;
        """)
        openconnection.commit()
        print(f"[TIME]Data loaded into {ratingstablename} in {time.time() - start:.2f} seconds.")
    except Exception as e:
        if openconnection:
            openconnection.rollback()
        print(f"Error loading ratings: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Tạo bảng lưu đánh giá và nạp dữ liệu từ file.

-Cách hoạt động:

+Tạo bảng có thêm các cột tạm (extra1, extra2, extra3) do định dạng file là userID:extra1:movieID:extra2:rating:extra3:timestamp.

+Dùng copy_from để nạp dữ liệu rất nhanh.

+Xóa các cột không cần thiết sau khi nạp.

*Lưu ý:Có thể sử dụng lại hàm create_db() ở trên.

2.4.Hàm create_range_partition_metadata_table():

```python
def create_range_partition_metadata_table(openconnection):
    cur = None
    try:
        cur = openconnection.cursor()
        cur.execute("""
            CREATE TABLE IF NOT EXISTS range_metadata (
                partition_id SERIAL PRIMARY KEY,
                partition_table_name VARCHAR(50) NOT NULL UNIQUE,
                range_start FLOAT NOT NULL,
                range_end FLOAT NOT NULL
            );
        """)
        openconnection.commit()
    except Exception as e:
        print(f"Error creating range_metadata table: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Tạo bảng range_metadata để lưu thông tin phân vùng dạng range.

-Cách hoạt động:

+Tạo bảng range_metadata gồm:

- partition_id: ID tự động tăng.
- partition_table_name: tên bảng con.
- range_start, range_end: khoảng giá trị đánh giá.

2.5.Hàm rangepartition()

```python
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    start = time.time()
    cur = None
    try:
        cur = openconnection.cursor()
        create_range_partition_metadata_table(openconnection)
        cur.execute("DELETE FROM range_metadata;")
        for i in range(numberofpartitions):
            cur.execute(f"DROP TABLE IF EXISTS range_part{i} CASCADE;")
        delta = 5.0 / numberofpartitions
        for i in range(numberofpartitions):
            minRange = round(i * delta, 6)
            maxRange = round(minRange + delta, 6)
            table_name = f"range_part{i}"
            cur.execute(f"""
                CREATE TABLE {table_name} (
                    userid INTEGER,
                    movieid INTEGER,
                    rating FLOAT
                );
            """)
            if i == 0:
```

```python
            """)
            if i == 0:
                cur.execute(f"""
                    INSERT INTO {table_name} (userid, movieid, rating)
                    SELECT userid, movieid, rating FROM {ratingstablename}
                    WHERE rating >= %s AND rating <= %s;
                """, (minRange, maxRange))
            else:
                cur.execute(f"""
                    INSERT INTO {table_name} (userid, movieid, rating)
                    SELECT userid, movieid, rating FROM {ratingstablename}
                    WHERE rating > %s AND rating <= %s;
                """, (minRange, maxRange))
            cur.execute("""
                INSERT INTO range_metadata (partition_table_name, range_start, range_end)
                VALUES (%s, %s, %s);
            """, (table_name, minRange, maxRange))
        openconnection.commit()
        print(f"[TIME]Range partition completed in {time.time() - start:.2f} seconds.")
    except Exception as e:
        if openconnection:
            openconnection.rollback()
        print(f"Error in range partition: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Phân vùng dữ liệu của bảng ratings theo giá trị đánh giá thành nhiều bảng con.

-Cách hoạt động:

+Xóa metadata cũ, tạo lại bảng range_part0 đến range_partN.

+Chia thang rating (0–5.0) thành các khoảng đều nhau: mỗi khoảng dài delta = 5.0 / numberofpartitions.

+Dữ liệu được chèn vào các partition theo điều kiện:

- Partition 0: rating ≥ min và ≤ max
- Các partition khác: rating > min và ≤ max

+Ghi lại thông tin từng partition vào range_metadata.

2.6.Hàm count_partitions():

```python
def count_partitions(prefix, openconnection):
    cur = None
    try:
        cur = openconnection.cursor()
        if prefix == 'range_part':
            cur.execute("SELECT COUNT(*) FROM range_metadata;")
        elif prefix == 'rrobin_part':
            cur.execute("SELECT COUNT(*) FROM roundrobin_metadata;")
        else:
            return 0
        row = cur.fetchone()
        return row[0] if row else 0
    except Exception as e:
        print(f"Error counting partitions: {e}")
        return 0
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Đếm số partition hiện có (range hoặc round-robin) dựa trên prefix.

-Cách hoạt động : Truy vấn bảng metadata tương ứng (range_metadata hoặc roundrobin_metadata) để đếm số bảng con.

2.6.Hàm rangeinsert()

```python
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    start = time.time()
    cur = None
    try:
        cur = openconnection.cursor()
        numberofpartitions = count_partitions('range_part', openconnection)
        if numberofpartitions == 0:
            raise Exception("No range partitions found in metadata")
        delta = 5.0 / numberofpartitions
        index = int(rating / delta)
        if rating % delta == 0 and index != 0:
            index -= 1
        if index >= numberofpartitions:
            index = numberofpartitions - 1
        table_name = f"range_part{index}"
        cur.execute(
            f"INSERT INTO {ratingstablename} (userid, movieid, rating) VALUES (%s, %s, %s);",
            (userid, itemid, rating)
        )
        cur.execute(
            f"INSERT INTO {table_name} (userid, movieid, rating) VALUES (%s, %s, %s);",
            (userid, itemid, rating)
        )
        openconnection.commit()
        print(f"[TIME]Range insert done in {time.time() - start:.4f} seconds.")
    except Exception as e:
        if openconnection:
            openconnection.rollback()
        print(f"Error in rangeinsert: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Chèn một rating mới vào bảng chính và bảng partition theo range.

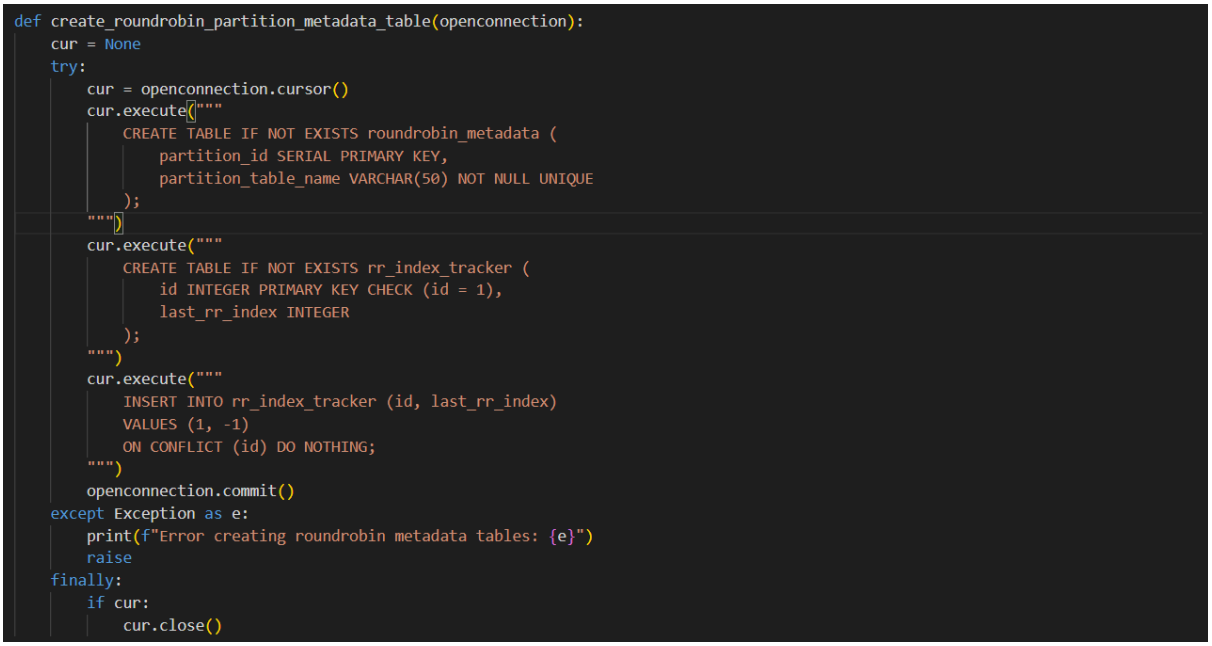
-Cách hoạt động:

+Tính khoảng (range) mà rating thuộc vào, dựa vào delta.

+Tính chỉ số partition (index).

+Chèn dữ liệu vào bảng chính và vào partition tương ứng.

2.7.Hàm create_roundrobin_partition_metadata_table():



```
def create_roundrobin_partition_metadata_table(openconnection):
    cur = None
    try:
        cur = openconnection.cursor()
        cur.execute("""
            CREATE TABLE IF NOT EXISTS roundrobin_metadata (
                partition_id SERIAL PRIMARY KEY,
                partition_table_name VARCHAR(50) NOT NULL UNIQUE
            );
        """)
        cur.execute("""
            CREATE TABLE IF NOT EXISTS rr_index_tracker (
                id INTEGER PRIMARY KEY CHECK (id = 1),
                last_rr_index INTEGER
            );
        """)
        cur.execute("""
            INSERT INTO rr_index_tracker (id, last_rr_index)
            VALUES (1, -1)
            ON CONFLICT (id) DO NOTHING;
        """)
        openconnection.commit()
    except Exception as e:
        print(f"Error creating roundrobin metadata tables: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Tạo bảng metadata cho round-robin partition và bảng theo dõi vòng chèn tiếp theo.

-Cách hoạt động:

+Tạo bảng roundrobin_metadata(partition_id, partition_table_name)

+Tạo bảng rr_index_tracker(last_rr_index), ban đầu gán 0.

2.8.Hàm roundrobinpartition():

```python
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
    import time
    start = time.time()
    cur = None
    try:
        cur = openconnection.cursor()

        create_roundrobin_partition_metadata_table(openconnection)
        cur.execute("DELETE FROM roundrobin_metadata;")

        # Drop và tạo lại các partition
        for i in range(numberofpartitions):
            cur.execute(f"DROP TABLE IF EXISTS rrobin_part{i} CASCADE;")
            cur.execute(f"CREATE TABLE rrobin_part{i} (userid INTEGER, movieid INTEGER, rating FLOAT);")

        # Tạo bảng tạm chứa dữ liệu đã đánh số thứ tự
        cur.execute("DROP TABLE IF EXISTS temp_rr_table;")
        cur.execute(f"""
            CREATE TEMP TABLE temp_rr_table AS
            SELECT userid, movieid, rating,
                   ROW_NUMBER() OVER (ORDER BY userid, movieid) AS rnum
            FROM {ratingstablename};
        """)

        # Chèn dữ liệu vào từng partition
        for i in range(numberofpartitions):
            cur.execute(f"""
                INSERT INTO rrobin_part{i} (userid, movieid, rating)
                SELECT userid, movieid, rating
                FROM temp_rr_table
                WHERE MOD(rnum - 1, %s) = %s;
            """, (numberofpartitions, i))
            cur.execute("INSERT INTO roundrobin_metadata (partition_table_name) VALUES (%s);", (f"rrobin_part{i}",))
```

```python
    # Tính last_rr_index
    cur.execute("SELECT COUNT(*) FROM temp_rr_table;")
    total_rows = cur.fetchone()[0]
    last_index = (total_rows - 1) % numberofpartitions if total_rows > 0 else -1

    # Cập nhật tracker
    cur.execute("""
        INSERT INTO rr_index_tracker (id, last_rr_index)
        VALUES (1, %s)
        ON CONFLICT (id) DO UPDATE SET last_rr_index = EXCLUDED.last_rr_index;
    """, (last_index,))

    openconnection.commit()
    print(f"[TIME] Round robin partition completed in {time.time() - start:.2f} seconds.")
except Exception as e:
    if openconnection:
        openconnection.rollback()
    print(f"Error in roundrobin partition: {e}")
    raise
finally:
    if cur:
        cur.close()
```

-Ý nghĩa:Phân vùng dữ liệu theo vòng tròn (round-robin).

-Cách hoạt động:

+ Chuẩn bị metadata và bảng phân mảnh

+ Chia dữ liệu theo Round Robin

+ Lưu thông tin metadata

+Cập nhật chỉ số vòng quay cuối cùng

+ Commit và đóng kết nối

2.9.Hàm roundrobininsert():

```python
def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
    start = time.time()
    cur = None
    try:
        cur = openconnection.cursor()
        cur.execute(f"INSERT INTO {ratingstablename} (userid, movieid, rating) VALUES (%s, %s, %s)", (userid, itemid, rating))
        numberofpartitions = count_partitions('rrobin_part', openconnection)
        if numberofpartitions == 0:
            raise Exception("No roundrobin partitions found in metadata")
        cur.execute("SELECT last_rr_index FROM rr_index_tracker WHERE id = 1")
        row = cur.fetchone()
        last_rr_index = row[0] if row else -1
        index = (last_rr_index + 1) % numberofpartitions
        table_name = f"rrobin_part{index}"
        cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) VALUES (%s, %s, %s)", (userid, itemid, rating))
        cur.execute("UPDATE rr_index_tracker SET last_rr_index = %s WHERE id = 1", (index,))
        openconnection.commit()
        print(f"[TIME]Round robin insert done in {time.time() - start:.4f} seconds.")
    except Exception as e:
        if openconnection:
            openconnection.rollback()
        print(f"Error in roundrobininsert: {e}")
        raise
    finally:
        if cur:
            cur.close()
```

-Ý nghĩa:Chèn một dòng dữ liệu vào bảng chính và một partition theo vòng tròn.

-Cách hoạt động:

+Lấy last_rr_index trong bảng rr_index_tracker.

+Tính chỉ số tiếp theo: (last_rr_index + 1) % numberofpartitions.

+Chèn vào bảng chính và partition tương ứng.

+Cập nhật last_rr_index.

3. Thiết kế bảng meta-data

-Bảng meta-data để lưu thông tin về các phân mảnh của bảng ratings

3.1.Cấu trúc bảng meta-data cho phân mảnh theo khoảng

-Tạo bảng range_metadata :

+Lưu thông tin về các bảng con (partition tables) mà dữ liệu được phân chia theo khoảng giá trị (range).

CREATE TABLE IF NOT EXISTS range_metadata (

partition_id SERIAL PRIMARY KEY,

partition_table_name VARCHAR(50) NOT NULL UNIQUE,

range_start FLOAT NOT NULL,

range_end FLOAT NOT NULL);

3.2.Cấu trúc bảng meta-data cho phân mảnh vòng tròn

-Tạo 2 bảng meta-data dùng cho phân mảnh kiểu Round Robin, cụ thể để:

+Lưu tên các bảng phân mảnh đã tạo.

+Theo dõi thứ tự hiện tại để biết dòng tiếp theo nên được chèn vào bảng phân mảnh nào.

3.2.1.Bảng roundrobin_metadata:

```
CREATE TABLE IF NOT EXISTS roundrobin_metadata (
   partition_id SERIAL PRIMARY KEY,
   partition_table_name VARCHAR(50) NOT NULL UNIQUE
);
```

- Chức năng: Khi bạn tạo bảng như rrobin_part0, rrobin_part1,,... thì mỗi bảng sẽ có một bản ghi trong bảng roundrobin_metadata.

3.2.2.Bảng rr_index_tracker:

```
CREATE TABLE IF NOT EXISTS rr_index_tracker (
   id INTEGER PRIMARY KEY CHECK (id = 1),
   last_rr_index INTEGER
);
```

-Chức năng: Cho biết index phân mảnh cuối cùng đã được sử dụng. Điều này giúp vòng tròn được duy trì liên tục.

V. Kiểm thử và đánh giá phân mảnh

1. Bộ dữ liệu kiểm thử

Test thử với bộ dữ liệu nhỏ trong file test_data.dat

Chạy mô phỏng phân mảnh file dữ liệu ratings.dat

2. Kết quả kiểm thử và đánh giá.

VI.Kết luận

Thông qua bài tập lớn này, chúng em đã có cơ hội tìm hiểu và triển khai các kỹ thuật phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu phân tán, bao gồm phân mảnh theo khoảng (Range Partitioning) và phân mảnh vòng tròn (Round Robin Partitioning). Bài tập không chỉ giúp hiểu rõ về mặt lý thuyết mà còn rèn luyện kỹ năng lập trình thao tác với cơ sở dữ liệu sử dụng PostgreSQL và thư viện psycopg2 trong Python.

Cụ thể, chúng em đã:

- Xây dựng và quản lý hệ thống kết nối đến cơ sở dữ liệu PostgreSQL.

- Cài đặt và thực hiện nạp dữ liệu từ file vào bảng chính.

- Thiết kế và triển khai hai cơ chế phân mảnh dữ liệu:

  - Range Partitioning: chia dữ liệu theo khoảng giá trị của rating.

  - Round Robin Partitioning: chia đều dữ liệu theo thứ tự xuất hiện.

- Thực hiện chèn dữ liệu mới vào đúng phân vùng tương ứng.

- Quản lý thông tin metadata của các phân vùng để dễ dàng truy xuất và bảo trì.

V.Tạo và đánh giá kết quả phân mảnh ngang (HẠNH)

1.Tiến hành tạo phân mảnh ngang

1.1.Tạo phân mảnh ngang với tệp dữ liệu test:

Sử dụng tệp dữ liệu test: file *test_data.dat*

***Tạo phân mảnh ngang theo khoảng**

Trong file *Assignment1Tester.py* comment đoạn code liên quan đến phân mảnh ngang kiểu vòng tròn:

```
# testHelper.deleteAllPublicTables(conn)
# MyAssignment.loadratings(RATINGS_TABLE, INPUT_FILE_PATH, conn)

# [result, e] = testHelper.testroundrobinpartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
# 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, '0')
# [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!")
```

Chạy file *Assignment1Tester.py*, có thể bỏ comment một trong hai dòng *[result, e] = testHelper.testrangeinsert(...)* để vừa tạo phân mảnh và vừa chèn dữ liệu mới vào.

```python
if __name__ == '__main__':
    try:
        testHelper.createdb(DATABASE_NAME)

        with testHelper.getopenconnection(dbname=DATABASE_NAME) as conn:
            conn.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)

            testHelper.deleteAllPublicTables(conn)

            [result, e] = testHelper.testloadratings(MyAssignment, RATINGS_TABLE, INPUT_FILE_PATH, conn, ACTUAL_ROWS_IN_INPUT_FILE)
            if result :
                print("loadratings function pass!")
            else:
                print("loadratings function fail!")

            [result, e] = testHelper.testrangepartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
            if result :
                print("rangepartition function pass!")
            else:
                print("rangepartition function fail!")

            # ALERT:: Use only one at a time i.e. uncomment only one line at a time and run the script
            [result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 3, conn, '2')
            # [result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 0, conn, '0')
            if result:
                print("rangeinsert function pass!")
            else:
                print("rangeinsert function fail!")
```

Màn hình *terminal* sau khi chạy code :

```
024_25/Dong Doi/Assignment1Tester.py"
Database dds_assgn1 already exists.
[TIME]Data loaded into ratings in 0.07 seconds.
loadratings function pass!
[TIME]Range partition completed in 0.02 seconds.
rangepartition function pass!
[TIME]Range insert done in 0.0062 seconds.
rangeinsert function pass!
Press enter to Delete all tables?
```

***Tạo phân mảnh ngang kiểu vòng tròn**

Trong file *Assignment1Tester.py* comment đoạn code liên quan đến phân mảnh ngang theo khoảng:

```python
# testHelper.deleteAllPublicTables(conn)

# [result, e] = testHelper.testloadratings(MyAssignment, RATINGS_TABLE, INPUT_FILE_PATH, conn, ACTUAL_ROWS_IN_INPUT_FILE)
# if result :
#     print("loadratings function pass!")
# else:
#     print("loadratings function fail!")

# [result, e] = testHelper.testrangepartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
# if result :
#     print("rangepartition function pass!")
# else:
#     print("rangepartition function fail!")

# # ALERT:: Use only one at a time i.e. uncomment only one line at a time and run the script
# # [result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 3, conn, '2')
# [result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 0, conn, '0')
# if result:
#     print("rangeinsert function pass!")
# else:
#     print("rangeinsert function fail!")
```

Chạy file *Assignment1Tester.py*, bỏ comment cho các dòng *[result, e] = testHelper.testroundrobininsert(...)* để vừa tạo phân mảnh và vừa chèn dữ liệu mới vào.

```
testHelper.deleteAllPublicTables(conn)
MyAssignment.loadratings(RATINGS_TABLE, INPUT_FILE_PATH, conn)

[result, e] = testHelper.testroundrobinpartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
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, '0')
[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!")
```

Màn hình *terminal* sau khi chạy code :

```
A database named "dds_assgn1" already exists
Database dds_assgn1 already exists.
[TIME]Data loaded into ratings in 0.07 seconds.
[TIME]Round robin partition completed in 0.04 seconds.
roundrobinpartition function pass!
[TIME]Round robin insert done in 0.0072 seconds.
[TIME]Round robin insert done in 0.0020 seconds.
[TIME]Round robin insert done in 0.0015 seconds.
roundrobininsert function pass!
Press enter to Delete all tables?
```

1.2.Tạo phân mảnh ngang với tệp dữ liệu thực

Sử dụng tệp dữ liệu thực: file *ratings.dat*

Các bước giống với tệp dữ liệu test, chỉ thay đoạn sau

```
INPUT_FILE_PATH = 'ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 10000054  # Number of lines in the input file
```

***Tạo phân mảnh ngang theo khoảng**

Các bước giống với tệp dữ liệu test.

Màn hình *terminal* sau khi chạy code :

```
PS D:\PTIT_HN\Sem2_2024_25\Dong Doi> & "D:/Progran
024_25/Dong Doi/Assignment1Tester.py"
A database named "dds_assgn1" already exists
Database dds_assgn1 already exists.
[TIME]Data loaded into ratings in 10.86 seconds.
loadratings function pass!
[TIME]Range partition completed in 28.00 seconds.
rangepartition function pass!
[TIME]Range insert done in 0.0140 seconds.
rangeinsert function pass!
Press enter to Delete all tables?
```

***Tạo phân mảnh ngang kiểu vòng tròn**

Các bước giống với tệp dữ liệu test.

Màn hình *terminal* sau khi chạy code :

```
PS D:\PTIT_HN\Sem2_2024_25\Dong Doi> & "D:/Program File
024_25/Dong Doi/Assignment1Tester.py"
[TIME]Data loaded into ratings in 10.64 seconds.
[TIME]Round robin partition completed in 76.89 seconds.
roundrobinpartition function pass!
[TIME]Round robin insert done in 0.0000 seconds.
[TIME]Round robin insert done in 0.0045 seconds.
[TIME]Round robin insert done in 0.0000 seconds.
roundrobininsert function pass!
Press enter to Delete all tables?
```

2.Kết quả phân mảnh ngang

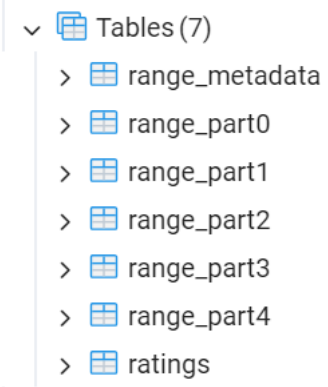
2.1. Với tệp dữ liệu test

Bảng *ratings* trong cơ sở dữ liệu: 15 dòng đầu tiên

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	4.5
3	1	231	4
4	1	292	3.5
5	1	316	3
6	1	329	2.5
7	1	355	2
8	1	356	1.5
9	1	362	1
10	1	364	0.5
11	1	370	0
12	1	377	3.5
13	1	420	5
14	1	466	4
15	1	480	5

***Tạo phân mảnh ngang theo khoảng**

>>Các phân mảnh tạo được



Bảng *range_metadata* lưu các thông tin gồm: id của phân mảnh *(partition_id)*, tên phân mảnh *(partition_table_name)*, khoảng giá trị bắt đầu, khoảng giá trị kết thúc *([range_start, range_end])*.

	partition_id [PK] integer	partition_table_name character varying (50)	range_start double precision	range_end double precision
1	1	range_part0	0	1
2	2	range_part1	1	2
3	3	range_part2	2	3
4	4	range_part3	3	4
5	5	range_part4	4	5

Phân mảnh 1: bảng *range_part0*

	userid integer	movieid integer	rating double precision
1	1	362	1
2	1	364	0.5
3	1	370	0

Phân mảnh 2: bảng *range_part1*

	userid integer	movieid integer	rating double precision
1	1	355	2
2	1	356	1.5
3	1	589	1.5

Phân mảnh 3: bảng *range_part2*

	userid integer	movieid integer	rating double precision
1	1	316	3
2	1	329	2.5
3	1	520	2.5
4	100	2	3

Phân mảnh 4: bảng *range_part3*

	userid integer	movieid integer	rating double precision
1	1	231	4
2	1	292	3.5
3	1	377	3.5
4	1	466	4
5	1	586	3.5

Phân mảnh 5: bảng *range_part4*

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	4.5
3	1	420	5
4	1	480	5
5	1	539	5
6	1	588	5

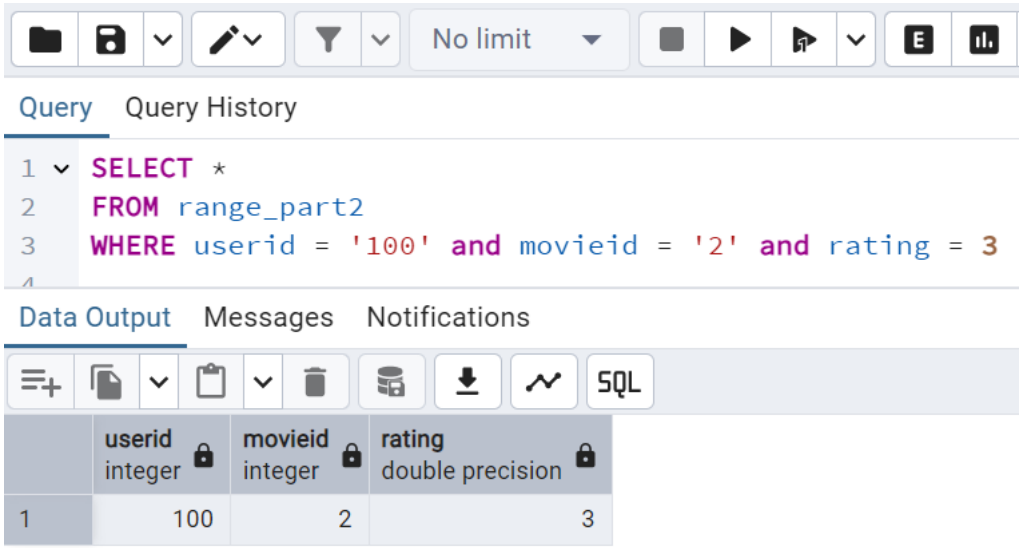
>> Chèn dữ liệu vào phân mảnh:

Bộ dữ liệu 1: chèn (useid, movieid, rating) = (100, 2, 3) vào phân mảnh.

Kết quả mong muốn: được chèn vào bảng *range_part2*.

Kết quả thực tế: chèn đúng.

Kiểm tra bộ dữ liệu có tồn tại trong bảng *range_part2* sử dụng câu lệnh truy vấn như hình

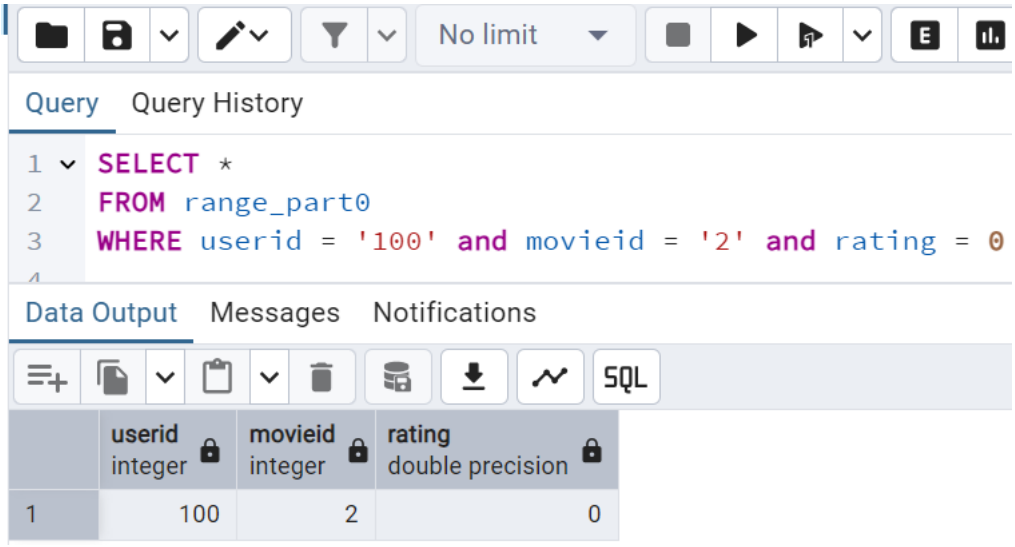


Bộ dữ liệu 2: chèn (useid, movieid, rating) = (100, 2, 0) vào phân mảnh.

Kết quả mong muốn: được chèn vào bảng *range_part0*.

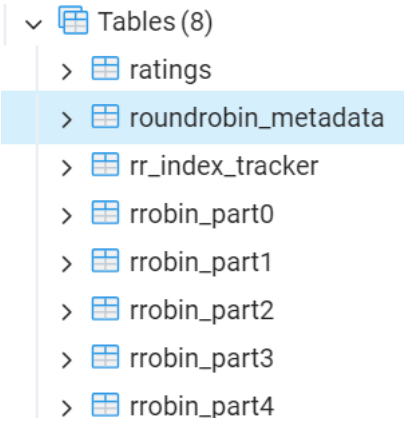
Kết quả thực tế: chèn đúng.

Kiểm tra bộ dữ liệu có tồn tại trong bảng *range_part0* sử dụng câu lệnh truy vấn như hình



***Tạo phân mảnh ngang kiểu vòng tròn**

>>Các phân mảnh tạo được:



Bảng *roundrobin_meta* lưu các thông tin gồm: id của phân mảnh *(partition_id)* và tên phân mảnh *(partition_table_name)*

	partition_id [PK] integer	partition_table_name character varying (50)
1	1	rrobin_part0
2	2	rrobin_part1
3	3	rrobin_part2
4	4	rrobin_part3
5	5	rrobin_part4

Bảng *rr_index_tracker* lưu thông tin gồm: *id* và chỉ số bảng cuối cùng được chèn dữ liệu *(last_rr_index)*

	id [PK] integer	last_rr_index integer
1	1	2

Phân mảnh 1: bảng *rrobin_part0*

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	329	2.5
3	1	370	0
4	1	520	2.5
5	100	1	3

Phân mảnh 2: bảng *rrobin_part1*

	userid integer	movieid integer	rating double precision
1	1	185	4.5
2	1	355	2
3	1	377	3.5
4	1	539	5
5	100	1	3

Phân mảnh 3: bảng *rrobin_part2*

	userid integer	movieid integer	rating double precision
1	1	231	4
2	1	356	1.5
3	1	420	5
4	1	586	3.5
5	100	1	3

Phân mảnh 4: bảng *rrobin_part3*

	userid integer	movieid integer	rating double precision
1	1	292	3.5
2	1	362	1
3	1	466	4
4	1	588	5

Phân mảnh 5: bảng *rrobin_part4*

	userid integer	movieid integer	rating double precision
1	1	316	3
2	1	364	0.5
3	1	480	5
4	1	589	1.5

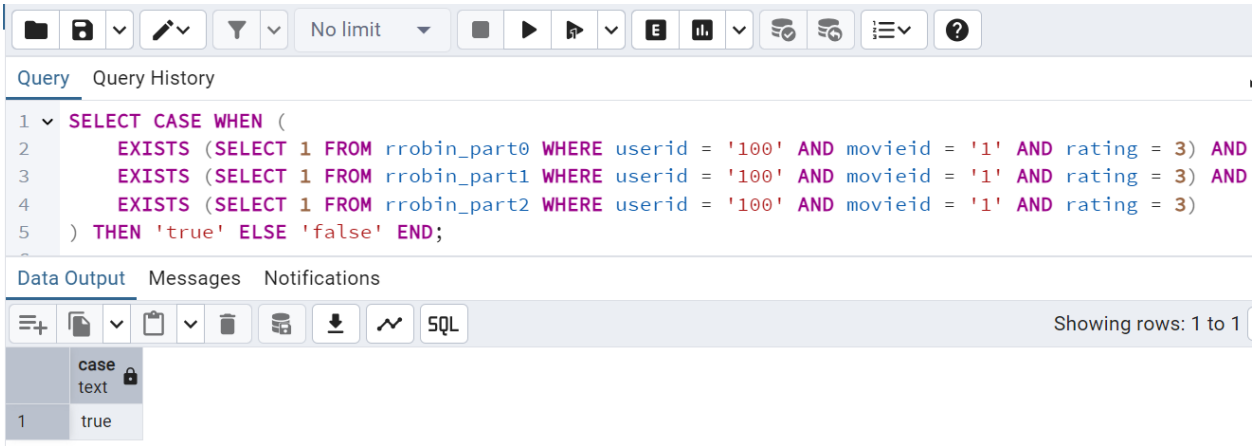
>>Chèn dữ liệu vào phân mảnh

Bộ dữ liệu 1: chèn (useid, movieid, rating) = (100, 1, 3) vào phân mảnh 3 lần.

Kết quả mong muốn: được chèn vào bảng *rrobin_part0, rrobin_part1* và *rrobin_part3* .

Kết quả thực tế: chèn đúng.

Kiểm tra bộ dữ liệu có tồn tại trong bảng *rrobin_part0, rrobin_part1* và *rrobin_part2* sử dụng câu lệnh truy vấn như hình



2.2. Với tệp dữ liệu thực

Bảng *ratings* trong cơ sở dữ liệu: 15 dòng cuối cùng

	userid integer	movieid integer	rating double precision
10000039	38940	3081	4
10000040	38940	3269	3.5
10000041	38940	3471	3
10000042	38940	3527	4.5
10000043	38940	3638	3
10000044	38940	3697	4
10000045	38940	3703	3.5
10000046	38940	3704	3
10000047	38940	3753	4
10000048	38940	3793	4
10000049	38940	3826	3.5
10000050	38940	3827	3.5
10000051	38940	3984	3.5
10000052	38940	3994	3.5
10000053	38940	3996	4
10000054	38940	4092	3.5

***Tạo phân mảnh ngang theo khoảng**

>>Các phân mảnh tạo được giống với tạo phân mảnh trên dữ liệu test.

Bảng *range_metadata* giống với tạo phân mảnh trên dữ liệu test.

Phân mảnh 1: bảng *range_part0*, 15 dòng đầu tiên

	userid integer	movieid integer	rating double precision
1	4	231	1
2	5	1	1
3	5	708	1
4	5	736	1
5	5	780	1
6	5	1391	1
7	6	3986	1
8	6	4270	1
9	7	1917	1
10	7	2478	1
11	7	5094	1
12	8	590	0.5
13	8	1035	0.5
14	8	1721	1
15	8	2378	0.5

Phân mảnh 2: bảng *range_part1*

	userid integer	movieid integer	rating double precision
1	2	648	2
2	2	802	2
3	2	858	2
4	3	1552	2
5	3	5505	2
6	4	344	2
7	6	4053	2
8	6	4369	2
9	7	541	2
10	7	1895	1.5
11	7	2335	2
12	7	5184	2
13	8	345	1.5
14	8	370	1.5
15	8	393	2

Phân mảnh 3: bảng *range_part2*

	userid integer	movieid integer	rating double precision
1	2	151	3
2	2	376	3
3	2	539	3
4	2	719	3
5	2	733	3
6	2	736	3
7	2	780	3
8	2	786	3
9	2	1049	3
10	2	1073	3
11	2	1356	3
12	2	1391	3
13	2	1544	3
14	3	1288	3
15	3	5299	3

Phân mảnh 4: bảng *range_part3*

	userid integer	movieid integer	rating double precision
1	2	1210	4
2	3	590	3.5
3	3	1148	4
4	3	1246	4
5	3	1252	4
6	3	1276	3.5
7	3	1408	3.5
8	3	3408	4
9	3	4535	4
10	3	4677	4
11	3	5952	3.5
12	3	6377	4
13	3	7153	4
14	3	7155	3.5
15	3	8529	4

Phân mảnh 5: bảng *range_part4*

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5
13	1	420	5
14	1	466	5
15	1	480	5

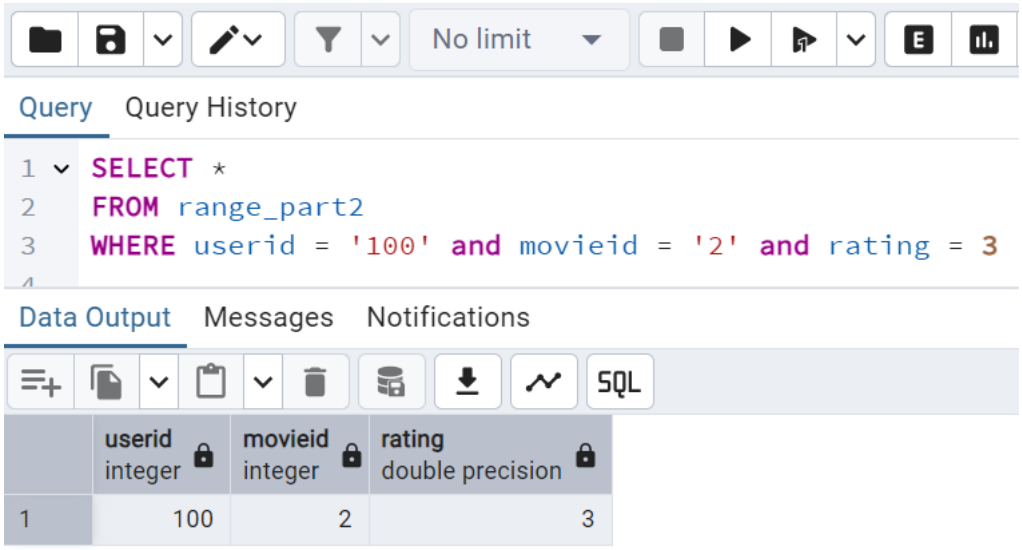
>> Chèn dữ liệu vào phân mảnh:

Bộ dữ liệu 1: chèn (useid, movieid, rating) = (100, 2, 3) vào phân mảnh.

Kết quả mong muốn: được chèn vào bảng *range_part2*.

Kết quả thực tế: chèn đúng.

Kiểm tra bộ dữ liệu có tồn tại trong bảng *range_part2* sử dụng câu lệnh truy vấn như hình

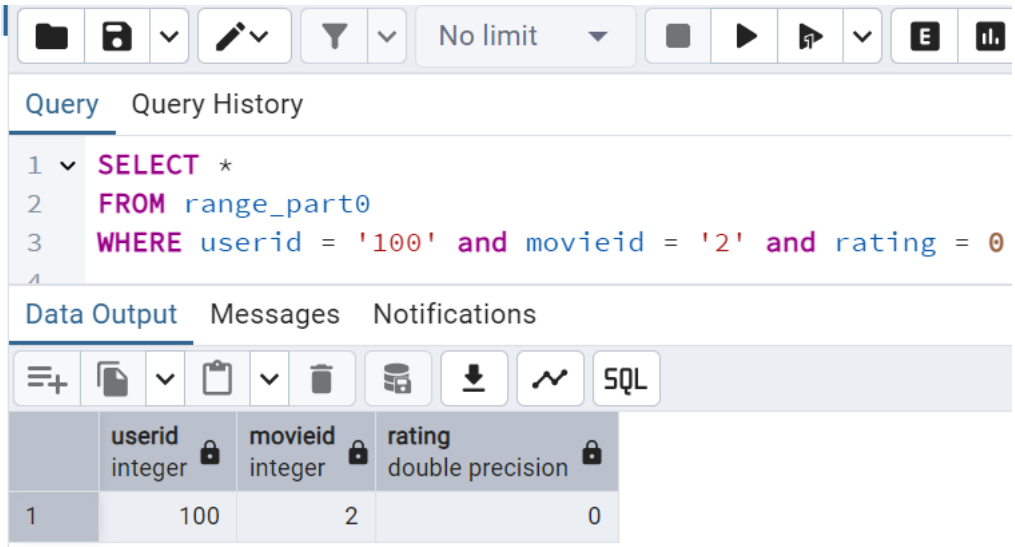


Bộ dữ liệu 2: chèn (useid, movieid, rating) = (100, 2, 0) vào phân mảnh.

Kết quả mong muốn: được chèn vào bảng *range_part0*.

Kết quả thực tế: chèn đúng.

Kiểm tra bộ dữ liệu có tồn tại trong bảng *range_part0* sử dụng câu lệnh truy vấn như hình



	userid integer	movieid integer	rating double precision
1	100	2	0

***Tạo phân mảnh ngang kiểu vòng tròn**

>>Các phân mảnh tạo được giống với tạo phân mảnh trên dữ liệu test.

Bảng *roundrobin_meta* giống với tạo phân mảnh trên dữ liệu test.

Bảng *rr_index_tracker:* chưa chèn dữ liệu

Phân mảnh 1: bảng *rrobin_part0*, 12 dòng cuối cùng

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	329	5
3	1	370	5
4	1	520	5
5	1	594	5
6	2	376	3
7	2	733	3
8	2	858	2
9	2	1391	3
10	3	590	3.5
11	3	1288	3
12	3	1674	4.5
13	3	4995	4.5
14	3	6287	3
15	3	8529	4

Phân mảnh 2: bảng *rrobin_part1*

	userid integer	movieid integer	rating double precision
1	1	185	5
2	1	355	5
3	1	377	5
4	1	539	5
5	1	616	5
6	2	539	3
7	2	736	3
8	2	1049	3
9	2	1544	3
10	3	1148	4
11	3	1408	3.5
12	3	3408	4
13	3	5299	3
14	3	6377	4
15	3	8533	4.5

Phân mảnh 3: bảng *rrobin_part2*

	userid integer	movieid integer	rating double precision
1	1	231	5
2	1	356	5
3	1	420	5
4	1	586	5
5	2	110	5
6	2	590	5
7	2	780	3
8	2	1073	3
9	3	110	4.5
10	3	1246	4
11	3	1552	2
12	3	3684	4.5
13	3	5505	2
14	3	6539	5
15	3	8783	5

Phân mảnh 4: bảng *rrobin_part3*

	userid integer	movieid integer	rating double precision
1	1	292	5
2	1	362	5
3	1	466	5
4	1	588	5
5	2	151	3
6	2	648	2
7	2	786	3
8	2	1210	4
9	3	151	4.5
10	3	1252	4
11	3	1564	4.5
12	3	4535	4
13	3	5527	4.5
14	3	7153	4
15	3	27821	4.5

Phân mảnh 5: bảng *rrobin_part4*

	userid integer	movieid integer	rating double precision
1	1	316	5
2	1	364	5
3	1	480	5
4	1	589	5
5	2	260	5
6	2	719	3
7	2	802	2
8	2	1356	3
9	3	213	5
10	3	1276	3.5
11	3	1597	4.5
12	3	4677	4
13	3	5952	3.5
14	3	7155	3.5
15	3	33750	3.5

>>Chèn dữ liệu vào phân mảnh

Bộ dữ liệu 1: chèn (useid, movieid, rating) = (100, 1, 3) vào phân mảnh 3 lần.

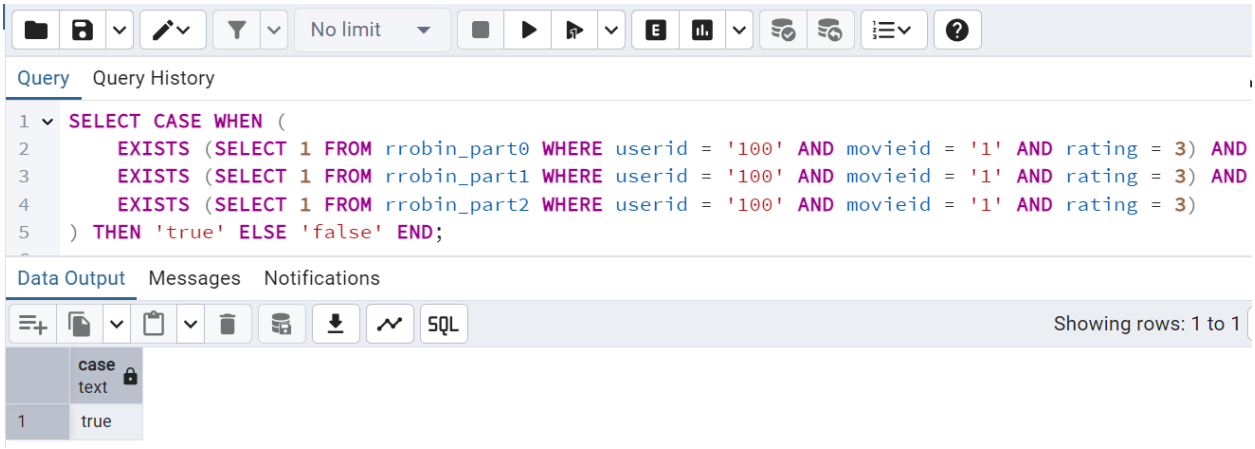
Kết quả mong muốn: được chèn vào bảng *rrobin_part0*, *rrobin_part1* và *rrobin_part3* .

Kết quả thực tế: chèn đúng.

Bảng *rr_index_tracker* khi đã chèn dữ liệu

	id [PK] integer	last_rr_index integer
1	1	1

Kiểm tra bộ dữ liệu có tồn tại trong bảng *rrobin_part0, rrobin_part1* và *rrobin_part2* sử dụng câu lệnh truy vấn như hình



Sau khi hoàn tất các chức năng và tiến hành test thử, chúng em nhận thấy thời gian truy vấn ở một số hàm vẫn còn cao, đặc biệt với các bảng có kích thước lớn. Trong bài tập lớn của chúng em vẫn còn nhiều thiếu sót, chúng em sẽ rút kinh nghiệm và hoàn thành tốt hơn vào các bài sau.

Tài liệu tham khảo