HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

CSDL PHÂN TÁN

Đề tài : Bài tập áp dụng phân mảnh ngang

Giảng viên hướng dẫn : TS. Kim Ngọc Bách

Thành viên **Nguyễn Mai Thanh - B22DCCN786**

Vũ Minh Dương - B22DCCN174

Trần Thanh Thảo - B22DCCN804

Nhóm bài tập : 15

Lớp : 9

Hà Nội – 2025

MỤC LỤC

MỞ ĐẦU

Trước tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến các thầy cô giáo trong khoa Công nghệ thông tin, Học viện Công nghệ Bưu chính Viễn thông (PTIT) – những người đã truyền đạt cho chúng em kiến thức nền tảng và định hướng vững chắc trong suốt quá trình học tập. Đặc biệt, nhóm xin được bày tỏ lòng biết ơn sâu sắc đến thầy Bách – giảng viên phụ trách môn học, người đã tận tình giảng dạy, hỗ trợ và đưa ra các hướng dẫn rõ ràng để chúng em có thể hoàn thành bài tập lớn này.

Bài tập lớn môn Cơ sở dữ liệu phân tán không chỉ giúp chúng em vận dụng kiến thức lý thuyết vào thực tế, mà còn tạo cơ hội để chúng em tiếp cận và thực hành các kỹ thuật quan trọng trong quản lý dữ liệu – cụ thể là mô phỏng các phương pháp phân mảnh ngang dữ liệu trên hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở như PostgreSQL hoặc MySQL.

Thông qua việc xử lý bộ dữ liệu đánh giá phim từ hệ thống MovieLens, chúng em đã thực hành thiết kế hàm xử lý dữ liệu, phân mảnh dữ liệu theo hai phương pháp Range Partitioning và Round Robin Partitioning, cũng như triển khai các hàm chèn dữ liệu vào đúng phân mảnh tương ứng. Việc xây dựng và kiểm thử toàn bộ hệ thống bằng ngôn ngữ Python giúp chúng em bổ sung thêm kỹ năng lập trình, tư duy thiết kế hệ thống và hiểu rõ hơn về cách tổ chức, lưu trữ dữ liệu hiệu quả trong cơ sở dữ liệu.

Chúng em hy vọng rằng bài làm này không chỉ đáp ứng đúng yêu cầu của môn học, mà còn phản ánh được tinh thần học tập nghiêm túc và sự nỗ lực của cả nhóm trong suốt quá trình thực hiện.

I. GIỚI THIỆU

1. Tổng quan về bài toán

Trong thời đại bùng nổ dữ liệu hiện nay, việc quản lý và xử lý các tập dữ liệu lớn đã trở thành một thách thức quan trọng đối với các hệ thống cơ sở dữ liệu. Khi dữ liệu tăng trưởng theo cấp số nhân, các phương pháp truyền thống lưu trữ dữ liệu trong một bảng duy nhất có thể dẫn đến các vấn đề về hiệu suất, khả năng mở rộng và tính khả dụng của hệ thống.

Phân mảnh dữ liệu (Data Partitioning) là một kỹ thuật quan trọng trong thiết kế cơ sở dữ liệu phân tán, cho phép chia nhỏ một bảng lớn thành nhiều phần nhỏ hơn được gọi là các phân mảnh (partitions hoặc fragments). Mỗi phân mảnh có thể được lưu trữ và xử lý độc lập, từ đó cải thiện đáng kể hiệu suất truy vấn, giảm thời gian phản hồi và tăng khả năng mở rộng của hệ thống.

Việc nghiên cứu và thực hành các phương pháp phân mảnh dữ liệu không chỉ giúp hiểu rõ hơn về nguyên lý hoạt động của các hệ cơ sở dữ liệu phân tán mà còn cung cấp nền tảng kiến thức quan trọng cho việc thiết kế các hệ thống xử lý dữ liệu lớn trong thực tế.

2. Mục tiêu của bài tập

Bài tập lớn này nhằm mục đích cung cấp trải nghiệm thực hành về việc triển khai các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ thực tế. Cụ thể, các mục tiêu chính bao gồm:

a. Mục tiêu học thuật

- Hiểu sâu về các khái niệm và nguyên lý của phân mảnh dữ liệu ngang (horizontal partitioning)
- Nắm vững hai phương pháp phân mảnh chính: Range Partitioning và Round Robin Partitioning
- Học cách triển khai các thuật toán phân mảnh dữ liệu bằng ngôn ngữ Python
- Thực hành tương tác với hệ quản trị cơ sở dữ liệu PostgreSQL/MySQL thông qua Python

b. Mục tiêu kỹ thuật

- Xây dựng hệ thống hoàn chỉnh bao gồm các chức năng: tải dữ liệu, phân mảnh và chèn dữ liệu mới
- Cài đặt thuật toán Range Partitioning với khả năng chia dữ liệu thành các khoảng giá trị đồng đều
- Cài đặt thuật toán Round Robin Partitioning để phân phối dữ liệu đều trên các phân mảnh

- Đảm bảo tính nhất quán và chính xác của dữ liệu sau khi phân mảnh và chèn

c. Mục tiêu thực tiễn

- Làm việc với tập dữ liệu thực tế từ MovieLens với quy mô lớn (10 triệu đánh giá)
- Phát triển kỹ năng làm việc nhóm và quản lý dự án phần mềm
- Rèn luyện khả năng phân tích, thiết kế và triển khai giải pháp kỹ thuật

3. **Phạm vi và giới hạn của đề tài**

a. Phạm vi nghiên cứu

Đề tài tập trung vào việc mô phỏng và triển khai hai phương pháp phân mảnh dữ liệu ngang cơ bản:

- Range Partitioning: Phân mảnh dựa trên khoảng giá trị của thuộc tính Rating, với các khoảng được chia đều từ giá trị tối thiểu đến tối đa
- Round Robin Partitioning: Phân mảnh theo cách phân phối vòng tròn, đảm bảo dữ liệu được phân bổ đều trên các phân mảnh

Hệ thống được xây dựng trên nền tảng:

- Ngôn ngữ lập trình: Python 3.12
- Hệ quản trị cơ sở dữ liệu: PostgreSQL.
- Dữ liệu thử nghiệm: Tập dữ liệu đánh giá phim từ MovieLens

b. Giới hạn của đề tài

- Phạm vi kỹ thuật: Chỉ tập trung vào phân mảnh ngang, không bao gồm phân mảnh dọc hay phân mảnh hỗn hợp
- Môi trường triển khai: Hệ thống được thiết kế để chạy trên môi trường đơn máy, chưa xem xét đến việc phân tán trên nhiều máy chủ
- Tối ưu hóa hiệu suất: Tập trung vào việc tối ưu hóa hiệu suất truy vấn hay các chỉ mục cơ sở dữ liệu
- Xử lý lỗi: Chỉ xử lý các trường hợp lỗi cơ bản, chưa có cơ chế xử lý lỗi phức tạp cho môi trường sản xuất
- Bảo mật: Không xem xét các vấn đề về bảo mật dữ liệu và kiểm soát truy cập

c. Ý nghĩa thực tiễn

Mặc dù có những giới hạn nhất định, đề tài này cung cấp nền tảng kiến thức quan trọng về phân mảnh dữ liệu - một kỹ thuật được sử dụng rộng rãi trong các hệ thống cơ sở dữ liệu lớn như Amazon DynamoDB, Google Bigtable, hay Apache Cassandra. Kiến thức và kỹ năng thu được từ bài tập này có thể được áp dụng và mở rộng cho các dự án thực tế trong tương lai.

II. CƠ SỞ LÝ THUYẾT

1. Tổng quan về phân mảnh ngang (horizontal partitioning)

- Khái niệm cơ bản

Phân mảnh ngang là kỹ thuật chia một bảng lớn thành nhiều bảng con (partition) nhỏ hơn, trong đó mỗi partition chứa một tập con các hàng (rows) của bảng gốc. Tất cả các partition có cùng cấu trúc schema nhưng chứa dữ liệu khác nhau.

- Lợi ích của phân mảnh ngang:
  - Cải thiện hiệu suất: Giảm thời gian truy vấn bằng cách chỉ scan các partition liên quan
  - Tăng khả năng mở rộng: Có thể lưu trữ dữ liệu trên nhiều server khác nhau
  - Quản lý dễ dàng: Có thể backup, restore từng partition độc lập
  - Cân bằng tải: Phân tán tải truy cập trên nhiều partition
- Các loại phân mảnh ngang chính:
  - Round Robin Partition: Phân phối tuần tự
  - Range Partition: Phân chia theo khoảng giá trị
  - Hash Partition: Phân chia theo hash function
  - List Partition: Phân chia theo danh sách giá trị cụ thể

2. Roundrobin partition

- Khái niệm

Round Robin Partition là một phương pháp phân mảnh dữ liệu theo kiểu xoay vòng, trong đó các bản ghi được phân phối tuần tự và đều đặn qua các partition theo thứ tự vòng tròn.

- Cách thức hoạt động:
  - Dữ liệu được phân phối theo thứ tự: Partition 1 → Partition 2 → ... → Partition n → Partition 1 (lặp lại)
  - Mỗi bản ghi mới sẽ được đặt vào partition tiếp theo trong chu kỳ
  - Không dựa vào giá trị của bất kỳ cột nào để quyết định partition
- Ưu điểm:

  - Phân phối đều: Đảm bảo dữ liệu được phân bố đồng đều giữa các partition
  - Đơn giản: Không cần định nghĩa key hoặc range
  - Cân bằng tải: Mỗi partition có số lượng bản ghi gần như nhau
  - Hiệu suất INSERT cao: Không cần tính toán phức tạp khi chèn dữ liệu
- Nhược điểm:
  - Hiệu suất truy vấn thấp: Cần scan tất cả partition khi tìm kiếm
  - Không tối ưu cho JOIN: Khó khăn khi thực hiện các phép nối
  - Không phù hợp với truy vấn có điều kiện: Không thể tận dụng partition elimination
- Công thức phân mảnh

Partition_ID = (Row_Number % Number_of_Partitions) + 1

3. **Range partition**

- Khái niệm

Range Partition là phương pháp phân mảnh dữ liệu dựa trên khoảng giá trị của một hoặc nhiều cột (partition key). Mỗi partition chứa dữ liệu có giá trị trong một khoảng xác định.

Cách thức hoạt động:

- Định nghĩa partition key (thường là cột có thứ tự như ngày tháng, ID)
- Chia khoảng giá trị thành các range không trùng lặp
- Mỗi bản ghi được đặt vào partition tương ứng với giá trị của partition key

Ưu điểm:

- Hiệu suất truy vấn cao: Partition elimination giúp chỉ truy cập partition cần thiết
- Tối ưu cho truy vấn có điều kiện: Đặc biệt hiệu quả với điều kiện WHERE trên partition key
- Dễ bảo trì: Có thể dễ dàng thêm/xóa partition theo thời gian
- Phù hợp với dữ liệu thời gian: Lý tưởng cho dữ liệu có tính chất thời gian

Nhược điểm:

- Phân phối không đều: Có thể gây ra data skew nếu phân chia range không hợp lý
- Phức tạp trong thiết kế: Cần phân tích kỹ để chọn partition key và range phù hợp
- Hotspot: Partition chứa dữ liệu mới nhất có thể bị overload

Chiến lược phân chia Range

- Theo thời gian: Theo ngày, tháng, năm
- Theo ID: Chia theo khoảng ID
- Theo giá trị: Chia theo khoảng giá trị của cột

4. **Ví dụ chi tiết**

Bảng gốc: ORDERS

```
CREATE TABLE ORDERS (
    order_id INT,
    customer_id INT,
    order_date DATE,
    amount DECIMAL(10,2),
    status VARCHAR(20)
);
```

Dữ liệu mẫu:

order_id	customer_id	order_date	amount	status
1001	501	2024-01-15	150.00	COMPLETED
1002	502	2024-01-20	200.50	PENDING
1003	503	2024-02-05	75.25	COMPLETED
1004	504	2024-02-10	300.00	SHIPPED
1005	505	2024-03-01	125.75	COMPLETED
1006	506	2024-03-15	180.00	PENDING
1007	507	2024-04-02	220.50	COMPLETED
1008	508	2024-04-20	95.00	SHIPPED

A. ROUND ROBIN PARTITION (3 partitions)

Cấu trúc partition:

```
-- Partition 1
CREATE TABLE ORDERS_P1 AS SELECT * FROM ORDERS WHERE 1=0;
-- Partition 2
CREATE TABLE ORDERS_P2 AS SELECT * FROM ORDERS WHERE 1=0;
-- Partition 3
CREATE TABLE ORDERS_P3 AS SELECT * FROM ORDERS WHERE 1=0;
```

Phân phối dữ liệu:

ORDERS_P1:

order_id	customer_id	order_date	amount	status
1001	501	2024-01-15	150.00	COMPLETED
1004	504	2024-02-10	300.00	SHIPPED
1007	507	2024-04-02	220.50	COMPLETED

ORDERS_P2:

order_id	customer_id	order_date	amount	status
1002	502	2024-01-20	200.50	PENDING
1005	505	2024-03-01	125.75	COMPLETED
1008	508	2024-04-20	95.00	SHIPPED

ORDERS_P3:

order_id	customer_id	order_date	amount	status
1003	503	2024-02-05	75.25	COMPLETED
1006	506	2024-03-15	180.00	PENDING

Ví dụ truy vấn:

-- Tìm order với order_id = 1005

-- Phải scan cả 3 partition vì không biết nó ở đâu

```
SELECT * FROM ORDERS_P1 WHERE order_id = 1005
UNION ALL
SELECT * FROM ORDERS_P2 WHERE order_id = 1005
UNION ALL
SELECT * FROM ORDERS_P3 WHERE order_id = 1005;
```

B. RANGE PARTITION (theo order_date)

Cấu trúc partition:

-- Partition Q1 2024 (Jan-Mar)

```
CREATE TABLE ORDERS_Q1_2024 AS
SELECT * FROM ORDERS
WHERE order_date >= '2024-01-01' AND order_date < '2024-04-01';
```

-- Partition Q2 2024 (Apr-Jun)

```
CREATE TABLE ORDERS_Q2_2024 AS
SELECT * FROM ORDERS
WHERE order_date >= '2024-04-01' AND order_date < '2024-07-01';
```

-- Partition Q3 2024 (Jul-Sep) - rỗng trong ví dụ này

```
CREATE TABLE ORDERS_Q3_2024 AS
SELECT * FROM ORDERS
WHERE order_date >= '2024-07-01' AND order_date < '2024-10-01';
```

Phân phối dữ liệu:

ORDERS_Q1_2024:

order_id	customer_id	order_date	amount	status
1001	501	2024-01-15	150.00	COMPLETED
1002	502	2024-01-20	200.50	PENDING
1003	503	2024-02-05	75.25	COMPLETED
1004	504	2024-02-10	300.00	SHIPPED
1005	505	2024-03-01	125.75	COMPLETED
1006	506	2024-03-15	180.00	PENDING

ORDERS_Q2_2024:

order_id	customer_id	order_date	amount	status
1007	507	2024-04-02	220.50	COMPLETED
1008	508	2024-04-20	95.00	SHIPPED

ORDERS_Q3_2024: (Rỗng)

Ví dụ truy vấn tối ưu:

-- Tìm orders trong tháng 2/2024

-- Chỉ cần truy cập ORDERS_Q1_2024

```
SELECT * FROM ORDERS_Q1_2024
WHERE order_date >= '2024-02-01' AND order_date < '2024-03-01';
```

-- Tìm orders có amount > 200 trong Q2 2024

-- Chỉ cần truy cập ORDERS_Q2_2024

```
SELECT * FROM ORDERS_Q2_2024

WHERE amount > 200;
```

5. **So sánh hiệu suất**

Round Robin Partition:

- INSERT: Rất nhanh - O(1)
- SELECT với điều kiện: Chậm - phải scan tất cả partition
- Phân phối: Hoàn toàn đều (3-3-2 records)
- Bảo trì: Đơn giản

Range Partition:

- INSERT: Nhanh - O(1) với partition key
- SELECT với điều kiện trên partition key: Rất nhanh - partition elimination
- Phân phối: Không đều (6-2-0 records)
- Bảo trì: Phức tạp hơn, cần quản lý range

6. **Kết luận**

Sử dụng Round Robin khi:

- Cần phân phối dữ liệu đều
- Chủ yếu thực hiện INSERT/UPDATE
- Không có truy vấn phức tạp
- Muốn thiết kế đơn giản

Sử dụng Range Partition khi:

- Có truy vấn thường xuyên dựa trên một cột cụ thể
- Dữ liệu có tính chất thời gian hoặc tuần tự
- Cần tối ưu hiệu suất SELECT
- Có thể chấp nhận phân phối không đều

III. PHÂN TÍCH ĐỀ BÀI

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). Dữ liệu thô có trong tệp 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 gian (Timestamp) là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970. Ví dụ nội dung tệp:

1::122::5::838985046

1::185::5::838983525

1::231::5::838983392

2. Yêu cầu cụ thể

a. Cài đặt *hàm Python LoadRatings()* nhận vào một đường dẫn tuyệt đối đến tệp rating.dat. LoadRatings() sẽ tải nội dung tệp vào một bảng trong PostgreSQL có tên Ratings với schema sau:

- UserID (int)
- MovieID (int)
- Rating (float)

b. Cài đặ*t hàm Python Range_Partition()* nhận vào:

(1) bảng Ratings trong PostgreSQL

(2) một số nguyên N là số phân mảnh cần tạo.

Range_Partition() sẽ tạo N phân mảnh ngang của bảng Ratings và lưu vào PostgreSQL. Thuật toán sẽ phân chia dựa trên N khoảng giá trị đồng đều của thuộc tính Rating.

c. Cài đặt *hàm Python RoundRobin_Partition()* nhận vào:

(1) bảng Ratings trong PostgreSQL

(2) một số nguyên N là số phân mảnh cần tạo.

Hàm sẽ tạo N phân mảnh ngang của bảng Ratings và lưu chúng trong PostgreSQL sử dụng phương pháp phân mảnh kiểu vòng tròn (round robin)

d. Cài đặ*t hàm Python RoundRobin_Insert()* nhận vào:

(1) bảng Ratings trong PostgreSQL,

(2) UserID,

(3) ItemID,

(4) Rating.

RoundRobin_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh theo cách round robin.

e. Cài đặt *hàm Python Range_Insert()* nhận vào:

(1) bảng Ratings trong PostgreSQL

(2) UserID,

(3) ItemID,

(4) Rating.

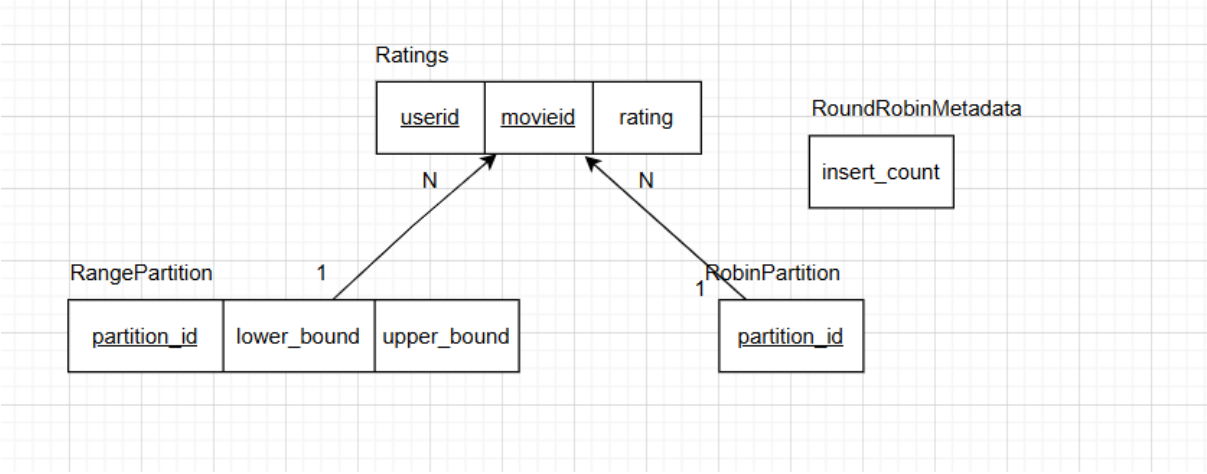
Range_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh dựa trên giá trị của Rating.

3. Một số lưu ý trong quá trình triển khai

- Số phân mảnh bắt đầu từ 0, nếu có 3 phân mảnh thì tên các bảng sẽ là range_part0, range_part1, range_part2 cho phân mảnh theo khoảng, và tương tự cho phân mảnh vòng tròn.
- Không được thay đổi tiền tố tên bảng phân mảnh đã được cung cấp trong assignment_tester.py.
- Không được mã hóa cứng tên tệp đầu vào.
- Không được đóng kết nối bên trong các hàm đã triển khai.
- Không được mã hóa cứng tên cơ sở dữ liệu.
- Lược đồ bảng phải giống với mô tả ở hàm LoadRatings().
- Không dùng biến toàn cục trong quá trình triển khai. Cho phép sử dụng bảng meta-data.
- Không được sửa đổi tệp dữ liệu.
- Việc vượt qua tất cả các testcase kiểm thử không đảm bảo kết quả đúng hoàn toàn. Nó chỉ có nghĩa là mã của bạn không có lỗi biên dịch.
- Để kiểm tra đầy đủ, bạn cần kiểm tra nội dung các bảng trong cơ sở dữ liệu.
- Hai hàm chèn có thể được gọi nhiều lần bất kỳ lúc nào. Chúng được thiết kế để duy trì các bảng trong cơ sở dữ liệu khi có thao tác chèn.

IV. THIẾT KẾ HỆ THỐNG

1. **Mục tiêu hệ thống**
   - Mô phỏng quá trình phân mảnh ngang dữ liệu trong hệ quản trị cơ sở dữ liệu.
   - Làm việc với bộ dữ liệu thực tế (MovieLens).
   - Áp dụng kỹ thuật "Range Partitioning" và "Round Robin Partitioning".
   - Chèn dữ liệu mới vào đúng phân mảnh, đảm bảo tính chính xác và hiệu quả.
2. **Mô hình hệ thống**
   - Dữ liệu gốc: ratings.dat (từ MovieLens)
   - Mô hình vật lý:
     + Mỗi table (gốc hay partition) là một thực thể vật lý riêng biệt trong PostgreSQL, có file lưu trữ và chỉ mục mặc định.
     + Indexes tự động trên các khóa (userid, movieid, rating) để cải thiện tốc độ SELECT.
   - Mô hình logic:
     + Bảng chính ratings(userid INT, movieid INT, rating FLOAT)
     + Partitions
     + Range partitions: range_part0 … range_partN−1 có cùng schema với ratings.
     + Round-robin partitions: rrobin_part0 … rrobin_partN−1 cũng tương tự.
     + Metadata
     + Bảng roundrobin_metadata(insert_count INT) lưu trạng thái chèn (thay thế biến toàn cục).
   - Trạng thái hệ thống: Được lưu vào bảng roundrobin_metadata, không dùng biến toàn cục
3. **Lược đồ phục vụ cho phân mảnh**



Các quan hệ:
- Ratings → RangePartition: N : 1(Mỗi bản ghi ở Ratings thuộc một RangePartition dựa trên giá trị rating)
- Ratings → RobinPartition: N : 1(Mỗi bản ghi ở Ratings thuộc một RobinPartition)

- RoundRobinMetadata là nơi lưu trữ trạng thái (số lần chèn) để hàm roundrobininsert() biết phân mảnh tiếp theo, chứ không chứa dữ liệu kết hợp với ratings.

Mô tả chi tiết:

- Thực thể Ratings:
  + userid (INT, PK): Mã định danh duy nhất của người dùng đã đánh giá phim.
  + movieid (INT, PK): Mã định danh duy nhất của bộ phim được đánh giá.
  + rating (FLOAT): Điểm đánh giá của người dùng cho phim, nằm trong khoảng 0.5 đến 5.0.
  + Hai cột userid + movieid cùng làm khóa chính, đảm bảo mỗi người dùng chỉ có một đánh giá cho mỗi phim.
- Thực thể RangePartition (khái niệm, không phải bảng vật lý):
  + partition_id (INT, PK): Số thứ tự phân mảnh, từ 0 đến N−1.
  + lower_bound (FLOAT): Giá trị rating tối thiểu (inclusive) mà partition này chứa.
  + upper_bound (FLOAT): Giá trị rating tối đa mà partition này chứa.

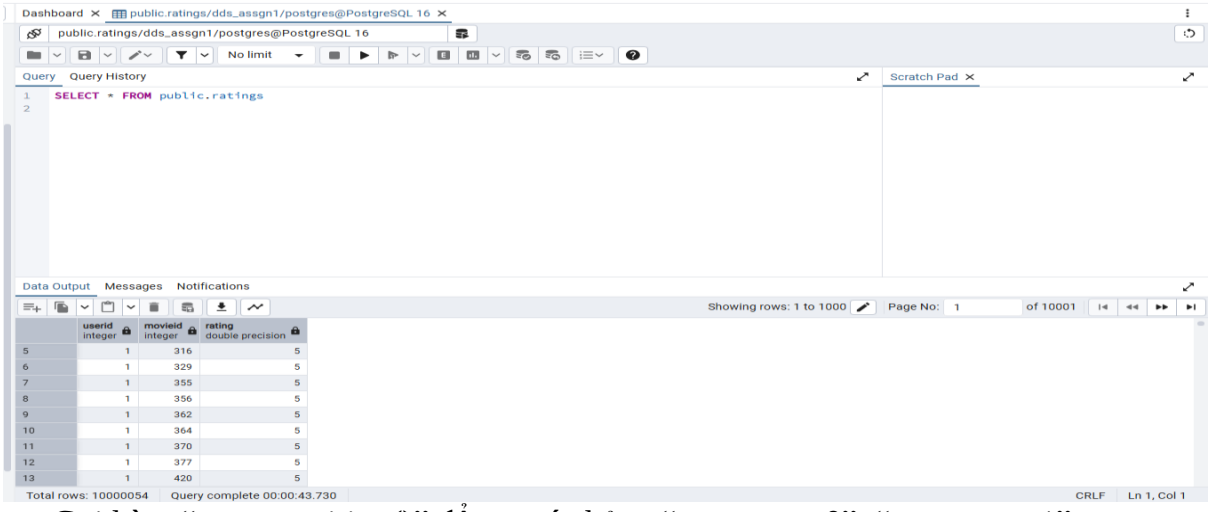
  Ví dụ:

  Đối với giá trị Rating trong khoảng [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5], với N=2:
    + Partition 0: lower_bound = 0.0, upper_bound = 2.5
    + Partition 1: lower_bound = 2.5, upper_bound = 5.0
- Thực thể RobinPartition (khái niệm)
  + partition_id (INT, PK): Số thứ tự phân mảnh vòng tròn, từ 0 đến N−1. Bản ghi thứ i sẽ vào partition i % N.
- Thực thể RoundRobinMetadata:
  + insert_count (INT): Đếm tổng số lần roundrobininsert() đã được gọi (số bản ghi đã chèn).Dùng để tính partition kế tiếp:
  + next_partition = insert_count % N
- Các bảng vật lý range_parti / rrobin_parti:
  + Mỗi bảng con đều kế thừa schema (userid, movieid, rating) từ Ratings.
  + Không có trường lower_bound/upper_bound trong bảng vật lý — thông tin này được giữ ngầm qua tên bảng range_part0, range_part1,…

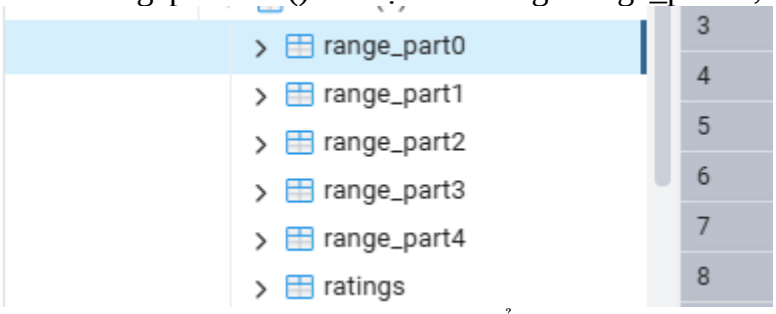
4. Cách hoạt động tổng quát

Hệ thống hoạt động theo các bước:

- Tải dữ liệu từ “ratings.dat” vào bảng “ratings” (bảng chính).

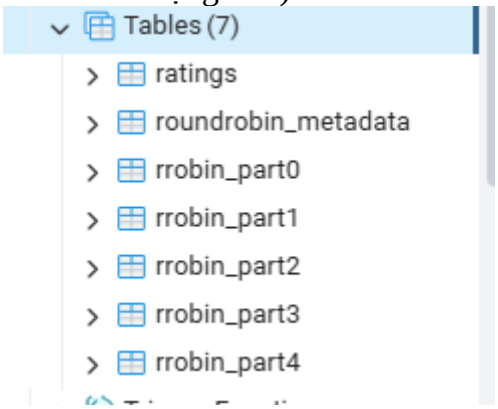


- Gọi hàm "rangepartition()" để tạo các bảng "range_part0", "range_part1",...



- Gọi hàm "roundrobinpartition()" để tạo các bảng "rrobin_part0", "rrobin_part1",...

- Khi có bản ghi mới: Dùng "rangeinsert()" hoặc "roundrobininsert()" để chèn đúng bảng -phân mảnh (sử dụng bảng "roundrobin_metadata" để theo dõi trạng thái).



V. CÀI ĐẶT VÀ TRIỂN KHAI

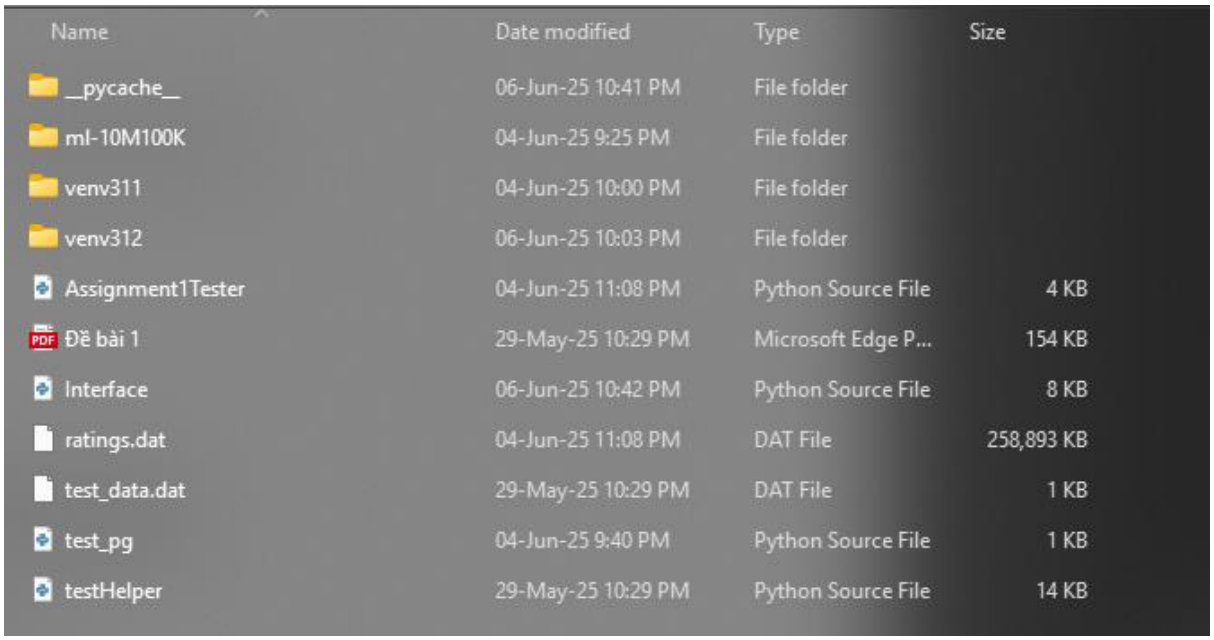
1. Cài đặt môi trường

- **Hệ điều hành:** Windows 10 64bit
- **CSDL:** PostgreSQL
- **Ngôn ngữ:** Python 3.12

- **Thư viện:** psycopg2, io

2. **Các file hệ thống**

File	Mô tả
Interface.py	Chứa toàn bộ hàm cài đặt cho phân mảnh và chèn
Assignment1Tester.py	File test tự động tất cả hàm
test_data.dat / ratings.dat	File dữ liệu nguồn
testHelper.py	Cung cấp API hỗ trợ tạo DB, xóa bảng...



3. **Giải thích chi tiết hàm trong Interface.py**

a. **loadratings() - Load dữ liệu từ file vào bảng**

Thuật toán:

- Đọc từng dòng trong file ratings.dat
- Chuẩn hóa định dạng từ :: thành tab (\t)
- Dùng copy_from của PostgreSQL để nạp dữ liệu rất nhanh

Code chính:

```
    # Dùng COPY để insert nhanh
    cur.copy_from(buffer, ratingstablename, sep='\t', columns=('userid', 'movieid', 'rating'))
    openconnection.commit()
    cur.close()
```

Giải thích:

- Hàm copy_from() cho phép chèn hàng loạt từ file (hoặc buffer) trực tiếp vào CSDL PostgreSQL mà không cần chạy từng câu lệnh INSERT.

- Định dạng ratings.dat có dạng UserID::MovieID::Rating::Timestamp → ta bỏ cột cuối và chỉ giữ 3 cột đầu để phù hợp bảng schema.

b. rangepartition() - Phân mảnh theo khoảng rating

Thuật toán:

- Xét dải giá trị rating từ 0 đến 5.0
- Chia đều thành N đoạn, mỗi đoạn có độ rộng delta = 5.0 / N
- Với mỗi đoạn [min, max], tạo bảng range_part{i}
- Chèn các dòng phù hợp vào bảng tương ứng:
- range_part0: rating >= min AND rating <= max
- Các bảng còn lại: rating > min AND rating <= max

Code ví dụ:

```
cur.execute("INSERT INTO range_part0 SELECT ... WHERE rating >= 0 AND rating <= 1")
```

Giải thích:

- Bảng range_part0 bao gồm ranh giới dưới để tránh mất dòng rating = 0
- Dùng nhiều bảng con giúp phân mảnh dễ truy vấn theo phân vùng.

c. roundrobinpartition() - Phân mảnh luân phiên

Thuật toán:

- Dùng ROW_NUMBER() để đánh số từng dòng của bảng ratings
- Dùng công thức (rnum - 1) % numberofpartitions để xác định phân mảnh tương ứng

Code mẫu:

```
SELECT userid, movieid, rating, ROW_NUMBER() OVER() as rnum FROM ratings
INSERT INTO rrobin_part0 SELECT ... WHERE mod(rnum - 1, N) = 0
```

Giải thích:

- Bằng cách đánh số dòng và chia theo số bảng, mỗi dòng được gán vào 1 bảng con theo vòng tròn
- Ưu điểm: phân phối đồng đều, không bị lệch như range partition nếu dữ liệu lệch phân bố

d. rangeinsert() và roundrobininsert()

rangeinsert()

Logic:

- Tính delta = 5.0 / N
- Với một rating mới → tính chỉ số index = int(rating / delta)
- Nếu rating % delta == 0, điều chỉnh để tránh dư thừa ở đầu vùng kế tiếp
- Chèn vào bảng ratings và range_part{index}

Code chính:

```
if rating % delta == 0 and index != 0:
   index = index - 1
```

Giải thích:

- Đảm bảo rating = 1.0 không bị vào cả range_part1 lẫn range_part0
- Tránh lỗi chỉ số vượt nếu rating = 5.0

roundrobininsert()

Logic:

- Dùng bảng roundrobin_metadata để lưu số lượt insert (insert_count)
- Tính chỉ số bảng cần insert: index = insert_count % numberofpartitions
- Cập nhật insert_count sau mỗi insert

Code chính:

```
index = insert_count % numberofpartitions
cur.execute("UPDATE roundrobin_metadata SET insert_count = insert_count + 1;")
```

Giải thích:

- Cách làm này giúp **duy trì thứ tự insert đúng** như đã phân chia bằng roundrobinpartition()
- Không bị ảnh hưởng bởi số dòng có sẵn trong bảng ratings

VI. KẾT QUẢ THỬ NGHIỆM

- Cấu hình database
  + Database: dds_assgn1
  + User: postgres
  + Password: admin123
  + Host: localhost
  + Port: 5432
- Dữ liệu Test
  + **Dataset**: MovieLens ratings.dat
  + **Format**: UserID::MovieID::Rating::Timestamp
  + **Rating range**: [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
  + **Kích thước test**: 1,000, 10,000, 10,000,054 records

1. **Kết quả hàm Loadratings()**

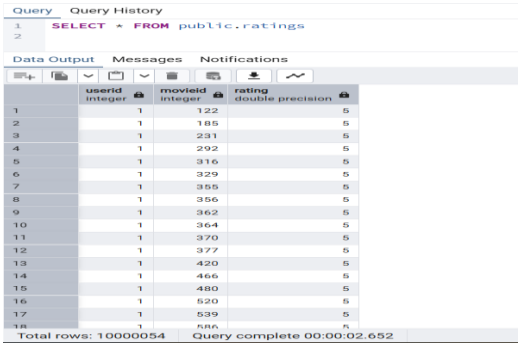
Điểm mạnh của code:

- Sử dụng COPY FROM để load nhanh
- Xử lý encoding UTF-8 chính xác
- Parsing format '::' thành tab-separated values
- Buffer in-memory tối ưu cho I/O

```
# Schema được tạo:
CREATE TABLE ratings (
    userid INT,
    movieid INT,
    rating FLOAT
);
```

- Đọc chính xác format UserID::MovieID::Rating::Timestamp
- Bỏ qua trường Timestamp theo yêu cầu
- Xử lý được các rating thập phân (0.5, 1.5, 2.5, ...)
- Không có data loss trong quá trình load

Bảng trong CSDL



2. **Kết quả Range Partitioning**

Kết quả Test với các giá trị N

Test với 10,000,054 records:

N	Partition	Range	Số records	Phần trăm
2	range_part0	[0, 2.5]	1757930	17.58%
	range_part1	(2.5, 5]	8242124	82.42%
3	range_part0	[0, 1.67]	597446	5.98%
	range_part1	(1.67, 3.34]	3517160	35.17%
	range_part2	(3.34, 5]	5885448	58.85%
5	range_part0	[0, 1]	479169	4.79%
	range_part1	(1, 2]	908584	9.08%
	range_part2	(2, 3]	2726854	27.27%
	range_part3	(3, 4]	3755614	37.56%
	range_part4	(4, 5]	2129834	21.3%

Nhận xét:

- Dữ liệu MovieLens có xu hướng rating cao (3-5 sao)

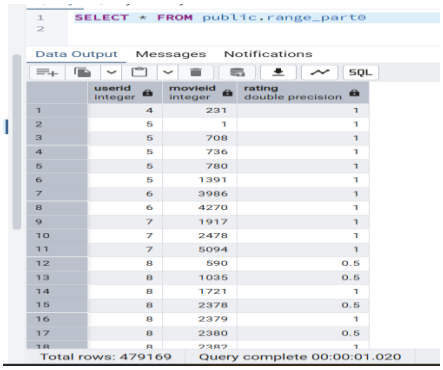
- Partition chứa rating thấp có ít dữ liệu hơn

- Độ mất cân bằng tăng theo số partition

Bảng trong CSDL

- *Với N = 5, có 5 phân mảnh*

Bảng range_part0



Bảng range_part1

```
SELECT * FROM public.range_part1
```

Data Output | Messages | Notifications

	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
16	8	420	1.5
17	8	587	1.5
18	8	737	2

Total rows: 908584 Query complete 00:00:00.451

Bảng range_part2

```
SELECT * FROM public.range_part2
```

Data Output | Messages | Notifications

	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
16	3	6287	3
17	4	21	3
18	4	39	3

Total rows: 2726854 Query complete 00:00:01.194

Bảng range_part3

Data Output | Messages | Notifications

	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
16	3	33750	3.5

Total rows: 3755614 Query complete 00:00:01.460

Bảng range_part4

```
SELECT * FROM public.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
16	1	520	5

Total rows: 2129834 Query complete 00:00:00.946

- *Với N = 3, có 3 phân mảnh*

Bảng range_part0

```
SELECT * FROM public.range_part0
```

	userid integer	movieid integer	rating double precision
1	69035	410	0.5
2	69035	466	0.5
3	69035	788	1.5
4	69035	1257	1.5
5	69035	3390	0.5
6	69035	3868	0.5
7	69035	4975	0.5
8	69035	7980	1
9	69036	585	1.5
10	69037	160	1
11	69037	344	1
12	69039	345	1
13	69039	432	1
14	69039	539	1
15	69039	1028	1.5
16	69039	1089	1

Total rows: 597446 Query complete 00:00:00.603

Bảng range_part1

```
SELECT * FROM public.range_part1
```

	userid integer	movieid integer	rating double precision
1	68725	3994	3
2	68725	3996	3
3	68725	3999	3
4	68725	4020	3
5	68725	4022	3
6	68725	4023	3
7	68725	4034	3
8	68725	4056	3
9	68725	4159	3
10	68725	4167	3
11	68725	4210	3
12	68725	4234	3
13	68725	4270	2
14	68725	4306	3
15	68725	4308	3
16	68725	4344	3

Total rows: 3517160 Query complete 00:00:01.434

Bảng range_part2

```
SELECT * FROM public.range_part2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
1	69035	288	5
2	69035	293	5
3	69035	302	3.5
4	69035	318	5
5	69035	332	3.5
6	69035	356	3.5
7	69035	434	5
8	69035	441	4
9	69035	445	4
10	69035	457	3.5
11	69035	497	3.5
12	69035	514	4
13	69035	527	5
14	69035	539	4
15	69035	541	5
16	69035	551	5

Total rows: 5885448 Query complete 00:00:03.193

- *Với N = 2, có 2 phân mảnh*

Bảng range_part0

```
SELECT * FROM public.range_part0
```

Data Output | Messages | Notifications

	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	231	1
7	4	344	2
8	5	1	1
9	5	708	1
10	5	736	1
11	5	780	1
12	5	1391	1
13	6	3986	1
14	6	4053	2
15	6	4270	1
16	6	4369	2

Total rows: 1757930 Query complete 00:00:01.451

Bảng range_part1

```
SELECT * FROM public.range_part1
```

Data Output | Messages | Notifications

	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
16	1	520	5

Total rows: 8242124 Query complete 00:00:03.471

3. **Kết quả Round Robin Partitioning**

Test với 10,000,054 records:

N	Partition	Số records	Phần trăm
2	rrobin_part0	5000027	50.00%
	rrobin_part1	5000027	50.00%
3	rrobin_part0	3333352	33.33%
	rrobin_part1	3333351	33.33%
	rrobin_part2	3333351	33.33%
5	rrobin_part0	2,000,011	20%
	rrobin_part1	2,000,011	20%
	rrobin_part2	2,000,011	20%
	rrobin_part3	2,000,011	20%
	rrobin_part4	2,000,010	20%

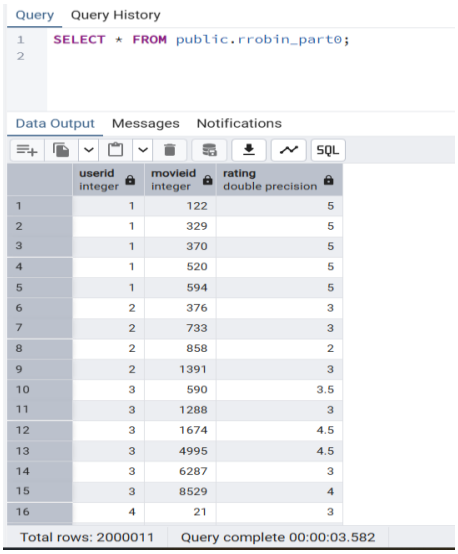
Nhận xét:

- Phân bố hoàn toàn đều (perfect load balancing)
- Hiệu suất tạo partition tốt hơn range partition
- Không phụ thuộc vào phân bố dữ liệu gốc
- Không tối ưu cho range queries
- Cần scan nhiều partition cho analytical queries

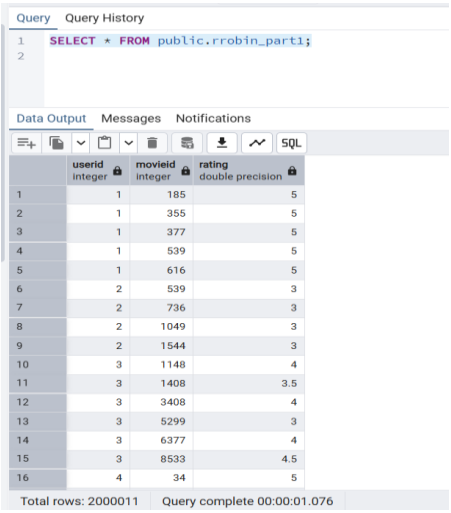
Bảng trong CSDL

- *Với N = 5*

Bảng rrobin_part0



Bảng rrobin_part1



Query Query History

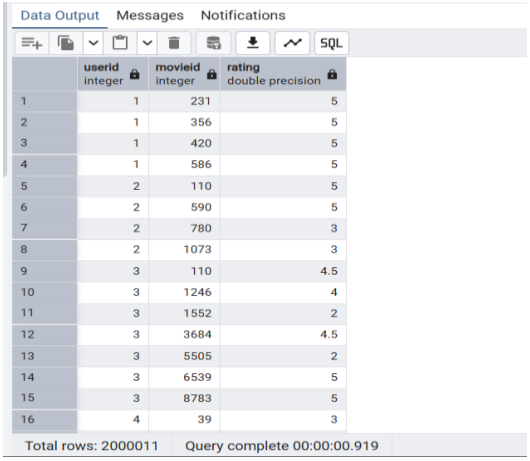
```
SELECT * FROM public.rrobin_part1;
```

Data Output Messages Notifications

	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
16	4	34	5

Total rows: 2000011 Query complete 00:00:01.076

Bảng rrobin_part2

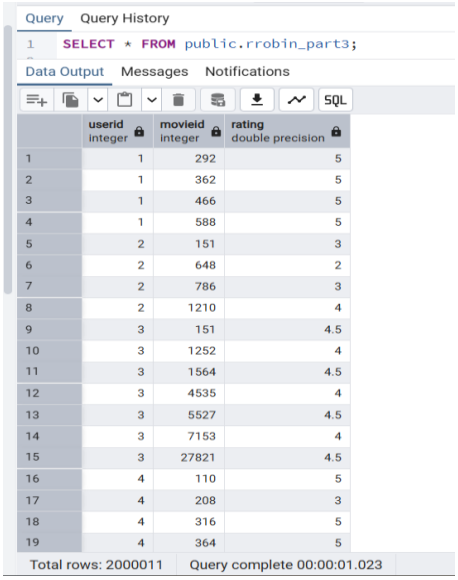


Data Output Messages Notifications

	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
16	4	39	3

Total rows: 2000011 Query complete 00:00:00.919

Bảng rrobin_part3



Query Query History

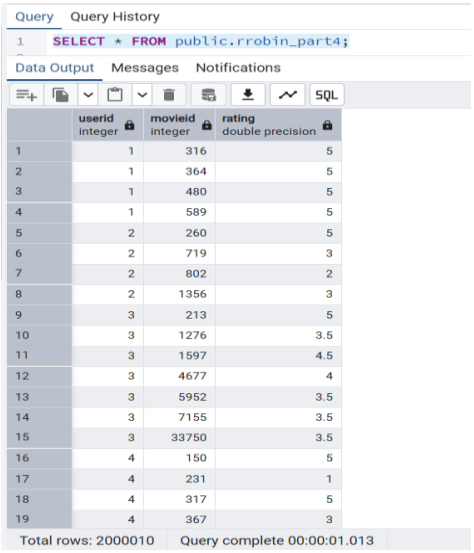
```
SELECT * FROM public.rrobin_part3;
```

Data Output Messages Notifications

	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
16	4	110	5
17	4	208	3
18	4	316	5
19	4	364	5

Total rows: 2000011 Query complete 00:00:01.023

Bảng rrobin_part4



```
Query  Query History
1  SELECT * FROM public.rrobin_part4;
Data Output  Messages  Notifications
```

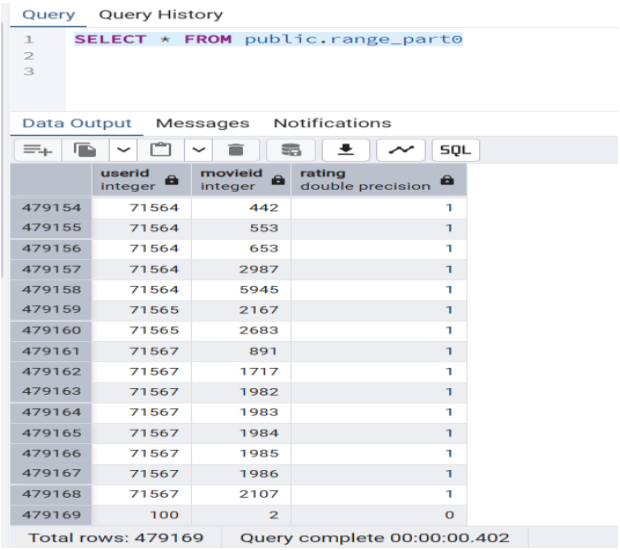
	userid integer	movieid integer	rating double precision
1	1	316	5
2	1	364	5
3	1	480	5
4	1	589	5
5	2	260	5
6	2	719	3
7	2	802	2
8	2	1356	3
9	3	213	5
10	3	1276	3.5
11	3	1597	4.5
12	3	4677	4
13	3	5952	3.5
14	3	7155	3.5
15	3	33750	3.5
16	4	150	5
17	4	231	1
18	4	317	5
19	4	367	3

Total rows: 2000010 Query complete 00:00:01.013

4. **Kết quả hàm rangeinsert()**

Test cases với N = 5, UserID = 100, MovieID = 2, Rating = 0

Bảng phân mảnh

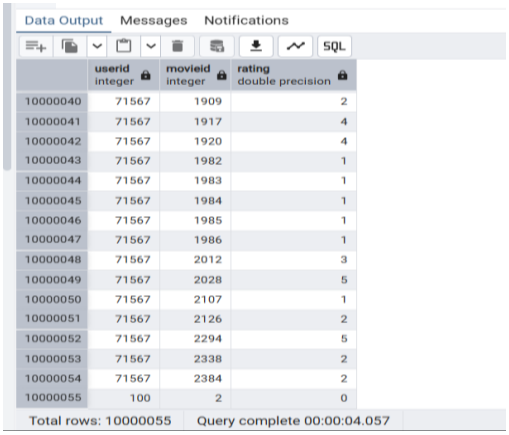


```
Query  Query History
1  SELECT * FROM public.range_part0
2
3
Data Output  Messages  Notifications
```

	userid integer	movieid integer	rating double precision
479154	71564	442	1
479155	71564	553	1
479156	71564	653	1
479157	71564	2987	1
479158	71564	5945	1
479159	71565	2167	1
479160	71565	2683	1
479161	71567	891	1
479162	71567	1717	1
479163	71567	1982	1
479164	71567	1983	1
479165	71567	1984	1
479166	71567	1985	1
479167	71567	1986	1
479168	71567	2107	1
479169	100	2	0

Total rows: 479169 Query complete 00:00:00.402

Bảng chính ratings



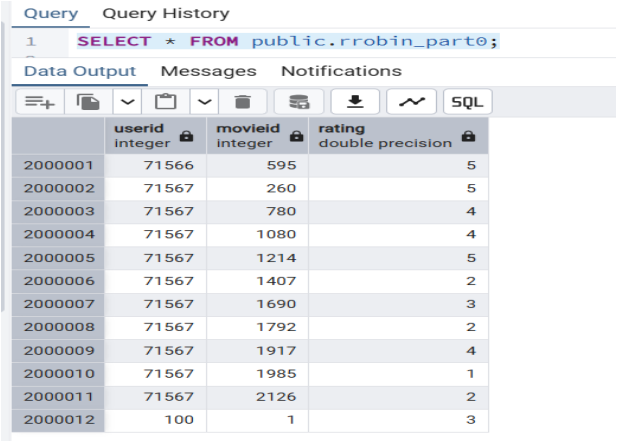
```
Data Output  Messages  Notifications
```

	userid integer	movieid integer	rating double precision
10000040	71567	1909	2
10000041	71567	1917	4
10000042	71567	1920	4
10000043	71567	1982	1
10000044	71567	1983	1
10000045	71567	1984	1
10000046	71567	1985	1
10000047	71567	1986	1
10000048	71567	2012	3
10000049	71567	2028	5
10000050	71567	2107	1
10000051	71567	2126	2
10000052	71567	2294	5
10000053	71567	2338	2
10000054	71567	2384	2
10000055	100	2	0

Total rows: 10000055 Query complete 00:00:04.057

5. **Kết quả Hàm roundrobininsert()**

Test case với N = 5, UserID = 100, MovieID = 1, rating =3, và phân mảnh part0



Query | Query History

```
1  SELECT * FROM public.rrobin_part0;
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
2000001	71566	595	5
2000002	71567	260	5
2000003	71567	780	4
2000004	71567	1080	4
2000005	71567	1214	5
2000006	71567	1407	2
2000007	71567	1690	3
2000008	71567	1792	2
2000009	71567	1917	4
2000010	71567	1985	1
2000011	71567	2126	2
2000012	100	1	3

PHÂN CHIA CÔNG VIỆC

Họ tên	Nhiệm vụ	Mức độ hoàn thành
Nguyễn Mai Thanh	Tìm hiểu lý thuyết, code các hàm phân mảnh, viết báo cáo chính	Tốt
Vũ Minh Dương	Tìm hiểu lý thuyết, code các hàm insert, bổ sung báo cáo	Tốt
Trần Thanh Thảo	Tìm hiểu lý thuyết, code hàm load, bổ sung báo cáo	Tốt

KẾT LUẬN

Qua quá trình thực hiện bài tập lớn môn Cơ sở dữ liệu phân tán, nhóm chúng em đã có cơ hội tiếp cận và thực hành các kiến thức cốt lõi liên quan đến kỹ thuật phân mảnh dữ liệu ngang, một trong những thành phần quan trọng trong thiết kế hệ thống cơ sở dữ liệu hiện đại.

Cụ thể, nhóm đã thành công trong việc xây dựng hệ thống mô phỏng phân mảnh dữ liệu trên PostgreSQL bằng Python, bao gồm:

- Tải dữ liệu đánh giá phim từ tập tin ratings.dat vào cơ sở dữ liệu thông qua hàm loadratings().
- Phân mảnh dữ liệu theo khoảng giá trị (range) và phân phối vòng tròn (round robin) với các hàm rangepartition() và roundrobinpartition().
- Chèn dữ liệu mới đúng phân mảnh tương ứng bằng các hàm rangeinsert() và roundrobininsert()..
- Lưu trạng thái hệ thống thông qua bảng metadata thay vì dùng biến toàn cục.

Quá trình thực hiện đã củng cố kiến thức về cơ sở dữ liệu phân tán, kỹ thuật phân mảnh dữ liệu, đồng thời rèn luyện kỹ năng lập trình Python, thiết kế hệ thống, thao tác với PostgreSQL và làm việc nhóm.

Mặc dù có những giới hạn như chỉ áp dụng trên môi trường đơn máy và chưa tối ưu hóa cho hiệu suất truy vấn lớn, kinh nghiệm này là nền tảng quan trọng cho các dự án thực tế liên quan đến xử lý dữ liệu lớn và phân tán trong tương lai.

TÀI LIỆU THAM KHẢO

[1] Özsu, M. T., & Valduriez, P. (2020). *Principles of Distributed Database Systems* (4th ed.). Springer International Publishing.

[2] Elmasri, R., & Navathe, S. B. (2019). *Fundamentals of Database Systems* (7th ed.). Pearson Education.

[3] Harper, F. M., & Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 1-19.

[4] GroupLens Research. (2023). *MovieLens Dataset*. Retrieved from https://grouplens.org/datasets/movielens/

[5] Consistent Hashing and Random Trees. (1997). *Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*. MIT Laboratory for Computer Science.

[6] Rendezvous Hashing. (1996). *A Method for Distributed Caching*. USENIX Summer Technical Conference.