

**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**

Giáo viên hướng dẫn:	Kim Ngọc Bách
Lớp:	9
Nhóm thực hiện:	20
Thành viên:	Lê Văn Đô - B22DCCN214 Bùi Mạnh Dũng - B22DCCN121 Thái Đoàn Trường - B22DCCN886

*Hà Nội, 2025*

## MỤC LỤC

MỞ ĐẦU .....	2
I. GIỚI THIỆU.....	3
1. Tổng quan về bài toán.....	3
2. Mục tiêu của bài tập.....	3
3. Phạm vi và giới hạn của đề tài .....	3
II. CƠ SỞ LÝ THUYẾT .....	3
1. Tổng quan về phân mảnh ngang (horizontal partitioning).....	3
2. Round Robin Partition .....	4
3. Range Partition.....	4
III. PHÂN TÍCH ĐỀ BÀI .....	5
1. Dữ liệu đầu vào .....	5
2. Yêu cầu cụ thể.....	5
3. Một số lưu ý trong quá trình triển khai .....	6
IV. CÀI ĐẶT VÀ TRIỂN KHAI CODE.....	7
1. Cài đặt môi trường.....	7
2. Các file cần thiết.....	7
3. Giải thích các hàm trong Interface.py .....	8
• Loadratings() .....	8
• Rangepartition().....	9
• Roundrobinpartition().....	10
• Rangeinsert() .....	11
• Roundrobininsert() .....	12
V. THỰC NGHIỆM.....	13
1. Kết quả hàm rangepartition() .....	14
2. Kết quả hàm rangeinsert() .....	15
3. Kết quả hàm roundrobinpartition() .....	16
4. Kết quả hàm roundrobininsert.....	17
5. Kết luận .....	18
Bảng phân công nhiệm vụ của thành viên .....	19
TÀI LIỆU THAM KHẢO .....	20

## MỞ ĐẦU

Đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến quý thầy cô khoa Công nghệ Thông tin – Học viện Công nghệ Bưu chính Viễn thông (PTIT) đã truyền đạt những kiến thức nền tảng quý báu. Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc tới thầy Bách, người đã tận tình hướng dẫn và chỉ bảo để chúng em hoàn thành tốt bài tập lớn này.

Bài tập lớn môn "Cơ sở dữ liệu phân tán" là cơ hội để chúng em vận dụng kiến thức vào thực tiễn, cụ thể là mô phỏng hai kỹ thuật phân mảnh ngang (Range Partitioning và Round Robin Partitioning) trên PostgreSQL hoặc MySQL.

Trong quá trình thực hiện, nhóm đã sử dụng ngôn ngữ Python và tập dữ liệu MovieLens để xây dựng các hàm xử lý, phân mảnh và chèn dữ liệu. Qua đó, chúng em không chỉ rèn luyện kỹ năng lập trình, thiết kế hệ thống mà còn hiểu sâu hơn về các phương pháp tổ chức dữ liệu hiệu quả.

Chúng em hy vọng bài làm này đáp ứng tốt yêu cầu của môn học, đồng thời phản ánh được tinh thần làm việc nghiêm túc và sự nỗ lực của cả nhóm.

## I. GIỚI THIỆU

### 1. Tổng quan về bài toán

Trong bối cảnh dữ liệu tăng mạnh, việc lưu trữ trong một bảng duy nhất gây nhiều vấn đề về hiệu suất và mở rộng. Phân mảnh dữ liệu là kỹ thuật chia nhỏ bảng thành các phần riêng biệt, giúp cải thiện hiệu suất truy vấn và khả năng mở rộng trong hệ cơ sở dữ liệu phân tán.

### 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:

- **Học thuật:** Nắm vững phân mảnh dữ liệu ngang, cụ thể là Range Partitioning và Round Robin Partitioning; triển khai bằng Python và kết nối với PostgreSQL/MySQL.
- **Kỹ thuật:** Xây dựng hệ thống thực hiện tải, phân mảnh và chèn dữ liệu; đảm bảo tính chính xác và nhất quán.
- **Thực tiễn:** Làm việc với dữ liệu thực tế từ MovieLens (10 triệu đánh giá), rèn kỹ năng nhóm và triển khai kỹ thuật.

### 3. Phạm vi và giới hạn của đề tài

- **Phạm vi:** Tập trung vào hai kỹ thuật phân mảnh ngang: Range và Round Robin, sử dụng Python và PostgreSQL.
- **Giới hạn:** Không xử lý phân mảnh dọc, không triển khai trên hệ phân tán thực sự, chưa tối ưu hóa chuyên sâu, chưa có cơ chế bảo mật hoặc xử lý lỗi nâng cao.
- **Ý nghĩa:** Dù còn hạn chế, đề tài cung cấp kiến thức nền tảng về phân mảnh dữ liệu, có thể áp dụng vào các hệ thống lớn như DynamoDB hay Bigtable.

## 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**
  - Roundrobin 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. Round Robin 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.

- **Ưu điểm**

Phân phối dữ liệu đều, đơn giản triển khai, chèn dữ liệu nhanh và cân bằng tải

- **Nhược điểm**

Truy vấn kém hiệu quả do phải quét toàn bộ partition, không tối ưu cho JOIN hoặc truy vấn có điều kiện.

- **Công thức phân mảnh**

$$\text{Partition\_ID} = (\text{Row\_Number} \% \text{Number\_of\_Partitions}) + 1$$

## 3. Range Partition

- **Khái niệm**

**Range Partitioning** là kỹ thuật phân mảnh dữ liệu theo khoảng giá trị của một cột khóa (partition key). Dữ liệu được chia thành các partition không trùng lặp, mỗi bản ghi được gán vào partition phù hợp với giá trị của khóa.

- **Ưu điểm**

Truy vấn nhanh nhờ chỉ truy cập partition liên quan, dễ bảo trì, phù hợp với dữ liệu theo thời gian

- **Nhược điểm**

Dễ gây mất cân bằng dữ liệu nếu chia không hợp lý, thiết kế phức tạp, có thể gây quá tải ở partition chứa dữ liệu mới nhất

- **Chiến lược phân chia theo Range**

Theo thời gian (ngày/tháng), theo ID hoặc theo giá trị cột

### 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 đặt 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 đặt 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. CÀI ĐẶT VÀ TRIỂN KHAI CODE

1. Cài đặt môi trường
  - Hệ điều hành: Window 10
  - CSDL: PostgreSQL
  - Ngôn ngữ: Python 3.12
  - Thư viện: psycopg2
2. Các file cần thiết

ratingdata.dat	File chứa dữ liệu thử nghiệm
test_data.dat	File chứa toàn bộ dữ liệu
Interface.py	File chứa các hàm cần thiết
testHelper.py	File chứa các hàm dùng để thử nghiệm
Assignment1Tester	File chứa chương trình kiểm thử

Name	Date modified	Type	Size
 Assignment1Tester	6/6/2025 3:02 PM	Python File	4 KB
 Interface	6/8/2025 12:37 PM	Python File	17 KB
 ratings.dat	6/6/2025 1:20 AM	DAT File	258,893 KB
 README	6/8/2025 12:43 PM	Markdown Source ...	1 KB
 test_data.dat	6/3/2025 10:17 PM	DAT File	1 KB
 testHelper	6/6/2025 2:16 AM	Python File	14 KB



### 3. Giải thích các hàm trong Interface.py

#### • Loadratings()

- Hàm tải dữ liệu từ file ratings vào bảng ratings trong cơ sở dữ liệu PostgreSQL
  - Mục đích: Tạo bảng ratings và nhập dữ liệu từ file test\_data.dat (định dạng UserID::MovieID::Rating::Timestamp)
  - Tác dụng: Chuẩn bị dữ liệu gốc cho các hàm phân mảnh (rangepartition, roundrobinpartition) và chèn dữ liệu (rangeinsert, roundrobininsert)
  - Schema bảng ratings: userid (INTEGER), movieid (INTEGER), rating (FLOAT)
  - Quy trình: Xóa bảng cũ nếu tồn tại, tạo bảng tạm với cột bổ sung để xử lý dấu ::, nhập dữ liệu từ file, xóa cột không cần thiết
- Tham số
  - ratingtablename (str): Tên bảng ratings
  - ratingsfilepath (str): Đường dẫn tuyệt đối đến file dữ liệu (test\_data.dat)
  - openconnection: Đối tượng kết nối đến cơ sở dữ liệu PostgreSQL (từ testHelper.getopenconnection)
- Biến chính
  - con: Đối tượng kết nối đến cơ sở dữ liệu
  - cur: Con trỏ (cursor) để thực thi các lệnh SQL
- Thuật toán

```
# Tạo bảng tạm thời với các cột bổ sung để xử lý định dạng file (UserID::MovieID::Rating::Timestamp)
# Cột tạm: extra1, extra2, extra3 lưu dấu ::, timestamp lưu giá trị không cần thiết
cur.execute(
    f"CREATE TABLE {ratingtablename} (userid INTEGER, extra1 CHAR, movieid INTEGER, extra2 CHAR, rating FLOAT, extra3 CHAR, timestamp BIGINT);")

# Nhập dữ liệu từ file vào bảng bằng phương thức copy_from
# Mở file ratingsfilepath, dùng dấu phân tách ':' để tách các cột
cur.copy_from(open(ratingsfilepath), ratingtablename, sep=':')
```

- Rangepartition()

- Hàm phân mảnh bảng ratings theo phương pháp Range Partitioning dựa trên cột rating (khoảng giá trị từ 0 đến 5).
  - Mục đích: Chia dữ liệu trong bảng ratings thành N bảng phân mảnh (range\_part0, range\_part1, ..., range\_part{N-1}),
- Mỗi bảng chứa các bản ghi có rating nằm trong một khoảng xác định.
  - Tác dụng: Tối ưu hóa truy vấn theo khoảng giá trị rating (ví dụ: tìm tất cả bản ghi có rating từ 3 đến 4).
  - Quy trình: Chia khoảng rating (0-5) thành N phần đều, mỗi phần có độ rộng  $\text{delta} = 5/N$ .
- Tạo bảng phân mảnh và chèn dữ liệu từ ratings vào bảng tương ứng dựa trên rating.
- Tham số:
  - ratingtablename (str): Tên bảng ratings.
  - numberofpartitions (int): Số lượng phân mảnh cần tạo.
  - openconnection: Đối tượng kết nối đến cơ sở dữ liệu PostgreSQL.
- Biến chính:
  - con: Đối tượng kết nối đến cơ sở dữ liệu.
  - cur: Con trỏ để thực thi lệnh SQL.
  - delta (float): Độ rộng của mỗi khoảng rating ( $5.0 / \text{numberofpartitions}$ ).
  - RANGE\_TABLE\_PREFIX (str): Tiền tố tên bảng phân mảnh, mặc định là 'range\_part'.
  - minRange (float): Giới hạn dưới của khoảng rating cho mỗi phân mảnh.
  - maxRange (float): Giới hạn trên của khoảng rating cho mỗi phân mảnh.
  - table\_name (str): Tên bảng phân mảnh (range\_part0, range\_part1, ...).
- Thuật toán:

```
# Tính độ rộng của mỗi khoảng rating
# delta: Khoảng cách giữa các khoảng rating, đảm bảo chia đều từ 0 đến 5
delta = 5.0 / numberofpartitions

# RANGE_TABLE_PREFIX: Tiền tố để đặt tên bảng phân mảnh
RANGE_TABLE_PREFIX = 'range_part'

# Lập qua N phân mảnh để tạo bảng và chèn dữ liệu
for i in range(numberofpartitions):
    # Tính giới hạn của khoảng rating cho phân mảnh i
    minRange = i * delta # Giới hạn dưới (ví dụ: 0, 1, 2, ...)
    maxRange = minRange + delta # Giới hạn trên (ví dụ: 1, 2, 3, ...)
    table_name = RANGE_TABLE_PREFIX + str(i) # Tên bảng: range_part0, range_part1, ...

    # Tạo bảng phân mảnh với schema giống bảng ratings
    cur.execute(f"CREATE TABLE {table_name} (userid INTEGER, movieid INTEGER, rating FLOAT);")

    # Chèn dữ liệu từ ratings vào bảng phân mảnh
    # Với i=0: Bao gồm cả giá trị minRange (rating >= minRange)
    # Với i>0: Không bao gồm minRange (rating > minRange) để tránh trùng lặp
    if i == 0:
        cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) SELECT userid, movieid, rating FROM {ratingtablename} WHERE rating >= {minRange} AND rating <= {maxRange};")
    else:
        cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) SELECT userid, movieid, rating FROM {ratingtablename} WHERE rating > {minRange} AND rating <= {maxRange};")
```

- Roundrobinpartition()

- Hàm phân mảnh bảng ratings theo phương pháp Round-Robin Partitioning.
  - Mục đích: Chia dữ liệu trong bảng ratings thành N bảng phân mảnh (rrobin\_part0, rrobin\_part1, ..., rrobin\_part{N-1}),
- Phân phối bản ghi đều đặn theo thứ tự vòng tròn.
  - Tác dụng: Đảm bảo phân bố đều dữ liệu, phù hợp cho các truy vấn không yêu cầu chọn lọc theo giá trị cụ thể.
  - Quy trình: Gán số thứ tự cho mỗi bản ghi bằng ROW\_NUMBER(), sau đó phân phối bản ghi vào các bảng dựa trên công thức  $(row\_number - 1) \% N$ .
- Tham số:
  - ratingtablename (str): Tên bảng ratings.
  - numberofpartitions (int): Số lượng phân mảnh cần tạo (N).
  - openconnection: Đối tượng kết nối đến cơ sở dữ liệu PostgreSQL.
- Biến chính:
  - con: Đối tượng kết nối đến cơ sở dữ liệu.
  - cur: Con trỏ để thực thi lệnh SQL.
  - RROBIN\_TABLE\_PREFIX (str): Tiền tố tên bảng phân mảnh, mặc định là 'rrobin\_part'.
  - table\_name (str): Tên bảng phân mảnh (rrobin\_part0, rrobin\_part1, ...).
- Thuật toán:

```
# RROBIN_TABLE_PREFIX: Tiền tố để đặt tên bảng phân mảnh
RROBIN_TABLE_PREFIX = 'rrobin_part'

# Lặp qua N phân mảnh để tạo bảng và chèn dữ liệu
for i in range(numberofpartitions):
    table_name = RROBIN_TABLE_PREFIX + str(i) # Tên bảng: rrobin_part0, rrobin_part1, ...

    # Tạo bảng phân mảnh với schema giống bảng ratings
    cur.execute(f"CREATE TABLE {table_name} (userid INTEGER, movieid INTEGER, rating FLOAT);")

    # Chèn dữ liệu vào bảng phân mảnh theo phương pháp round-robin
    # ROW_NUMBER() OVER(): gán số thứ tự (1, 2, 3, ...) cho mỗi bản ghi trong ratings
    # (row-1) % numberofpartitions = i: Chọn các bản ghi thuộc phân mảnh i
    cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) SELECT userid, movieid, rating FROM (SELECT userid, movieid, rating, ROW_NUMBER() OVER() AS rownum FROM {ratingtablename}) AS temp WHERE (rownum-1) % {numberofpartitions} = {i};")
```

- Yếu tố phụ:

```
# Đếm số bản ghi hiện tại trong bảng ratings
# total_rows: Số bản ghi sau khi chèn bản ghi mới
cur.execute(f"SELECT COUNT(*) FROM {ratingtablename};")
total_rows = cur.fetchone()[0]

cur.execute("CREATE TABLE roundrobin_metadata(next_use_partition int);")
# Lưu lại vị trí tiếp theo để chèn
next_use = int(total_rows % numberofpartitions)
cur.execute(f"INSERT INTO roundrobin_metadata (next_use_partition) VALUES ({next_use});")
```

Tạo bảng roundrobin\_metadata dùng để lưu trữ ngay vị trí tiếp theo sẽ được chèn nếu gọi hàm insert

- Rangeinsert()

- Hàm chèn một bản ghi mới vào bảng ratings và bảng phân mảnh range tương ứng.
  - Mục đích: Thêm bản ghi vào bảng chính (ratings) và một bảng range\_partX dựa trên giá trị rating.
  - Tác dụng: Duy trì tính toàn vẹn dữ liệu, đảm bảo bản ghi xuất hiện ở cả bảng chính và đúng phân mảnh.
  - Quy trình: Chèn bản ghi vào ratings, tính toán phân mảnh dựa trên rating / delta, chèn vào bảng range\_partX.
- Tham số:
  - ratingtablename (str): Tên bảng ratings.
  - userid (int): ID của người dùng.
  - itemid (int): ID của phim (movieid).
  - rating (float): Điểm đánh giá, từ 0 đến 5.
  - openconnection: Đối tượng kết nối đến cơ sở dữ liệu PostgreSQL.
- Biến chính:
  - con: Đối tượng kết nối đến cơ sở dữ liệu.
  - cur: Con trỏ để thực thi lệnh SQL.
  - RANGE\_TABLE\_PREFIX (str): Tiền tố tên bảng phân mảnh ('range\_part').
  - numberofpartitions (int): Số phân mảnh hiện có, lấy từ count\_partitions.
  - delta (float): Độ rộng của mỗi khoảng rating ( $5.0 / \text{numberofpartitions}$ ).
  - index (int): Chỉ số của bảng phân mảnh (X trong range\_partX).
  - table\_name (str): Tên bảng phân mảnh để chèn bản ghi.
- Thuật toán:

```
# Chèn bản ghi mới vào bảng ratings
# userid, itemid, rating: Giá trị của bản ghi mới
cur.execute(f"INSERT INTO {ratingtablename} (userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});")

# Tìm bảng phân mảnh range phù hợp
RANGE_TABLE_PREFIX = 'range_part'
# numberofpartitions: Số bảng phân mảnh hiện có, lấy từ hàm count_partitions
numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
if numberofpartitions <= 0:
    return # Thoát nếu không có bảng phân mảnh

# Tính độ rộng khoảng rating và chỉ số phân mảnh
delta = 5.0 / numberofpartitions
index = int(rating / delta) # Xác định phân mảnh dựa trên rating / delta
# Điều chỉnh chỉ số nếu rating nằm ở ranh giới (ví dụ: rating=5 vào range_part{N-1})
if rating % delta == 0 and index != 0:
    index -= 1
table_name = RANGE_TABLE_PREFIX + str(index) # Tên bảng: range_partX

# Chèn bản ghi vào bảng phân mảnh tương ứng
cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});")
```

- Roundrobininsert()
  - Hàm chèn một bản ghi mới vào bảng ratings và bảng phân mảnh round-robin tương ứng.
    - Mục đích: Thêm bản ghi vào bảng chính (ratings) và một bảng rrobin\_partX dựa trên thứ tự chèn.
    - Tác dụng: Duy trì phân bố đều dữ liệu trong các phân mảnh round-robin.
    - Quy trình: Chèn bản ghi vào ratings, đếm số bản ghi hiện tại để xác định phân mảnh, chèn vào rrobin\_partX.
  - Tham số:
    - ratingtablename (str): Tên bảng ratings.
    - userid (int): ID của người dùng.
    - itemid (int): ID của phim (movieid).
    - rating (float): Điểm đánh giá, từ 0 đến 5.
    - openconnection: Đối tượng kết nối đến cơ sở dữ liệu PostgreSQL.
  - Biến chính:
    - con: Đối tượng kết nối đến cơ sở dữ liệu.
    - cur: Con trỏ để thực thi lệnh SQL.
    - RROBIN\_TABLE\_PREFIX (str): Tiền tố tên bảng phân mảnh ('rrobin\_part').
    - numberofpartitions (int): Số phân mảnh hiện có, lấy từ count\_partitions.
    - next\_use: Chỉ số chứa thông tin của lần chèn tiếp theo.
    - index (int): Chỉ số của bảng phân mảnh (X trong rrobin\_partX).
    - table\_name (str): Tên bảng phân mảnh để chèn bản ghi.
  - Thuật toán

```
# Chèn bản ghi mới vào bảng ratings
cur.execute(f"INSERT INTO {ratingtablename} (userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});")

# Tìm bảng phân mảnh round-robin phù hợp
RROBIN_TABLE_PREFIX = 'rrobin_part'
numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
if numberofpartitions <= 0:
    return # Thoát nếu không có bảng phân mảnh

# Lấy vị trí sẽ được chèn tiếp theo được lưu trong bảng roundrobin_metadata
cur.execute("SELECT next_use_partition FROM roundrobin_metadata FOR UPDATE;")
next_use = cur.fetchone()[0]

# Tăng vị trí thêm 1
cur.execute("UPDATE roundrobin_metadata SET next_use_partition=next_use_partition+1;")

# Tính chỉ số phân mảnh dựa trên vị trí chèn tiếp theo
index = (next_use) % numberofpartitions
table_name = RROBIN_TABLE_PREFIX + str(index) # Tên bảng: rrobin_partX

# Chèn bản ghi vào bảng phân mảnh
cur.execute(f"INSERT INTO {table_name} (userid, movieid, rating) VALUES ({userid}, {itemid}, {rating});")
```

## V. THỰC NGHIỆM

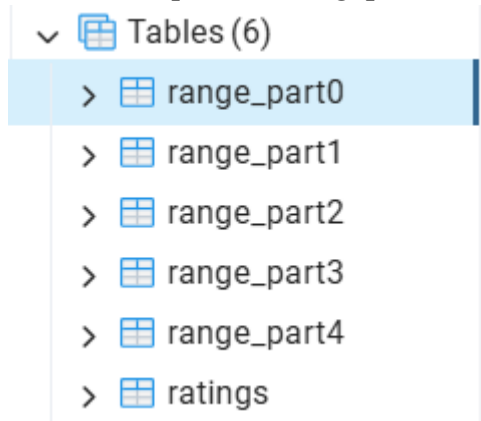
Cấu hình database:

- Database: dds\_assgn1
- User: postgres
- Password: 1234
- Host: localhost
- Port: 5432

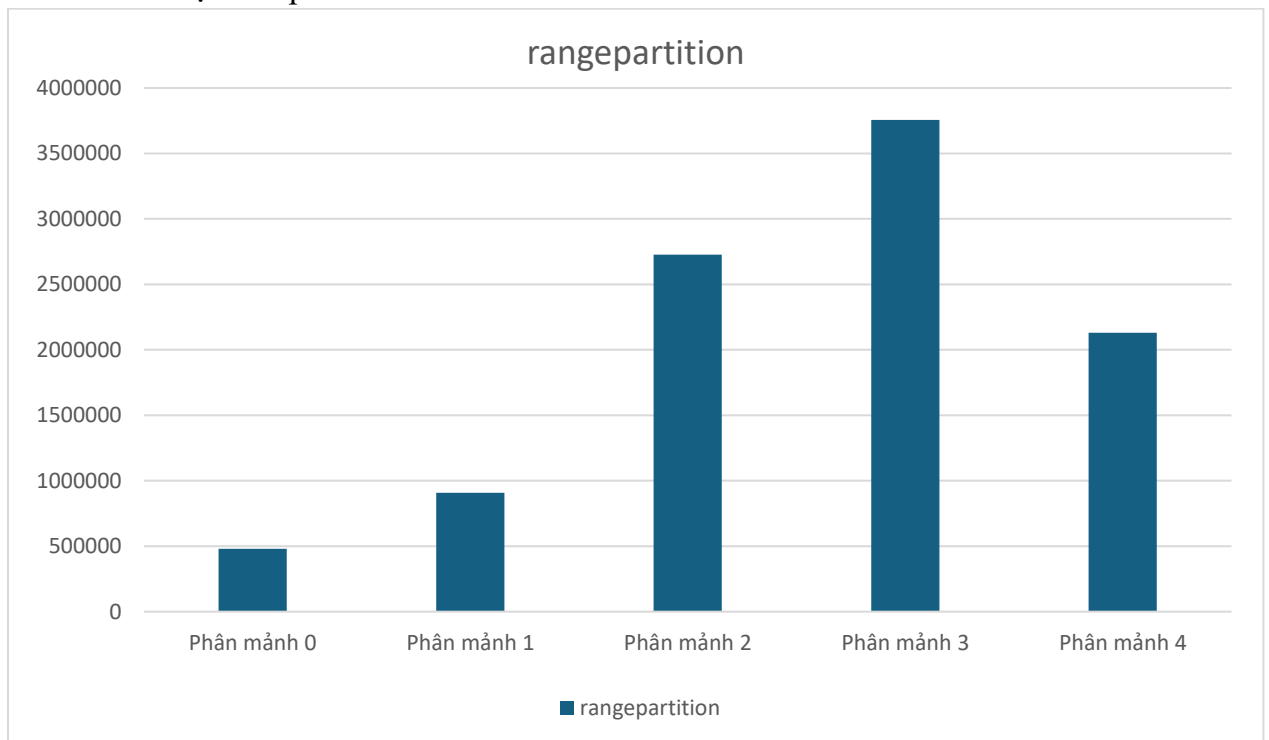
Dữ liệu test:

- Dataset: MovieLens ratings.dat gồm 10000054 dòng dữ liệu
- 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]

## 1. Kết quả hàm rangepartition()



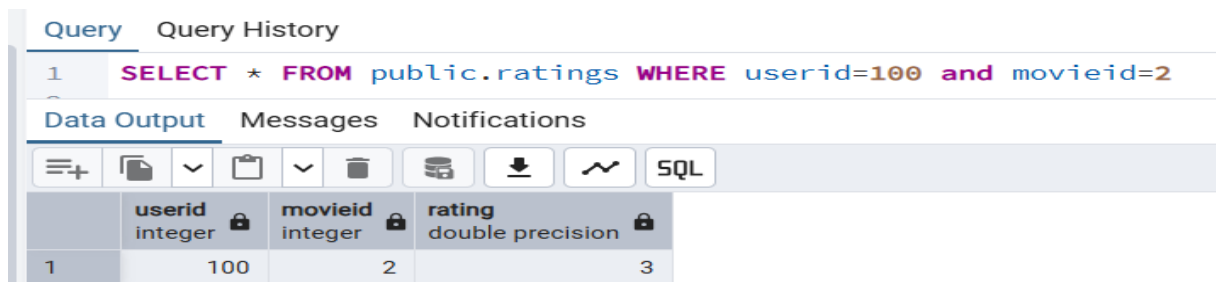
Phân bố dữ liệu sau phân mảnh:



## 2. Kết quả hàm rangeinsert()

Ta sẽ chèn 1 hàng với dữ liệu userid=100, movieid=2, rating=3

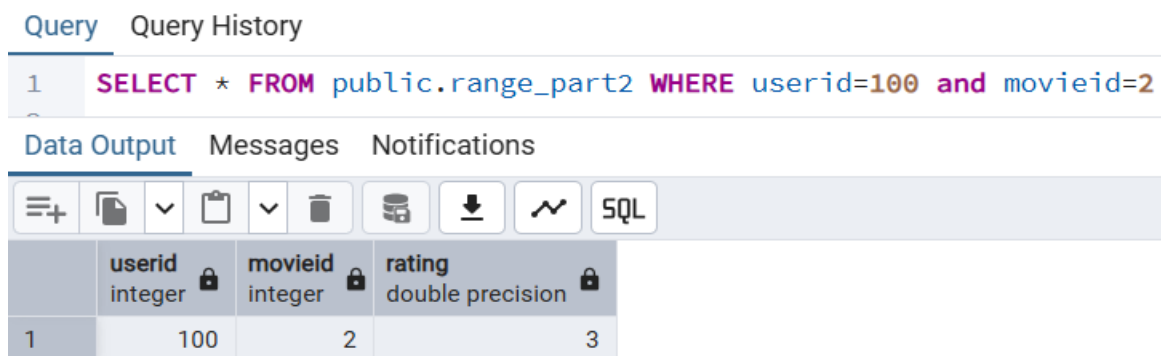
Kết quả trong bảng ratings



The screenshot shows a database query tool interface. The 'Query' tab is active, displaying the SQL query: `SELECT * FROM public.ratings WHERE userid=100 and movieid=2`. Below the query, the 'Data Output' tab is selected, showing a table with 3 columns: 'userid' (integer), 'movieid' (integer), and 'rating' (double precision). The table contains one row with values 100, 2, and 3 respectively.

	userid integer	movieid integer	rating double precision
1	100	2	3

Kết quả trong bảng range\_part2 nơi chứa các đánh giá 2.5 và 3.0



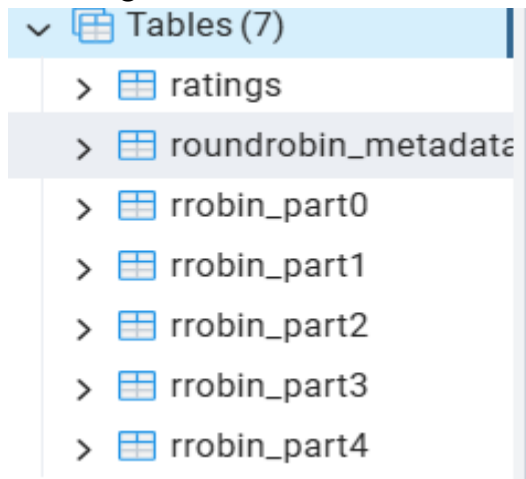
The screenshot shows a database query tool interface. The 'Query' tab is active, displaying the SQL query: `SELECT * FROM public.range_part2 WHERE userid=100 and movieid=2`. Below the query, the 'Data Output' tab is selected, showing a table with 3 columns: 'userid' (integer), 'movieid' (integer), and 'rating' (double precision). The table contains one row with values 100, 2, and 3 respectively.

	userid integer	movieid integer	rating double precision
1	100	2	3

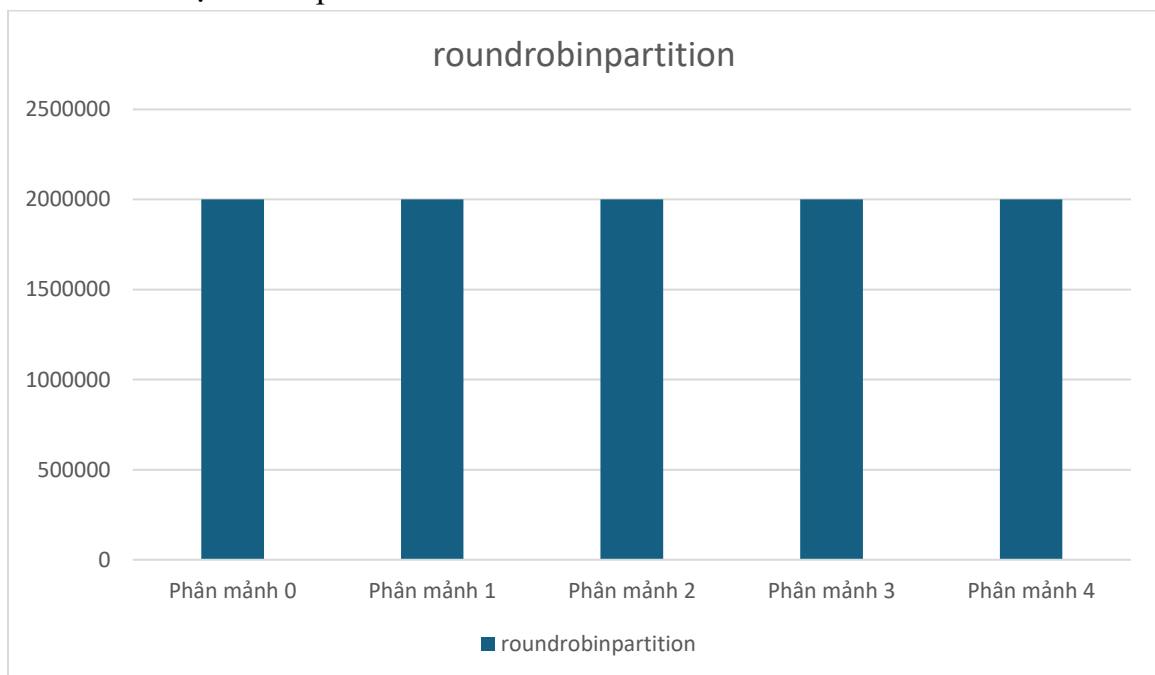


### 3. Kết quả hàm roundrobinpartition()

Các bảng được tạo ra:



Phân bố dữ liệu ở các phân mảnh:



#### 4. Kết quả hàm roundrobininsert

Ta sẽ chèn 1 hàng với dữ liệu userid=100, movieid=1, rating=3

Dữ liệu đã chèn thành công vào bảng ratings:

Query

Query History

1

SELECT \* FROM public.ratings WHERE userid=100 and movieid=1 and rating=3

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	userid integer	movieid integer	rating double precision
1	100	1	3

Dữ liệu đã chèn thành công vào phân mảnh số 4:

Query

Query History

1

SELECT \* FROM public.rrobin\_part4 WHERE userid=100 and movieid=1 and rating=3

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	userid integer	movieid integer	rating double precision
1	100	1	3

5. Kết luận

- Tất cả các hàm đều hoạt động đúng chức năng
- Range Partition sẽ cho ra dữ liệu phân bố trực quan theo tỉ lệ đánh giá của người xem
- RoundRobin Partition sẽ phân bố dữ liệu đều trong các phân mảnh
- Các hàm chèn đều hoạt động tốt

**Bảng phân công nhiệm vụ của thành viên**

Tên thành viên	Nhiệm vụ	Đánh giá
Lê Văn Đô	Tìm hiểu các thuật toán, viết các hàm load và insert và hỗ trợ làm báo cáo	Tốt
Bùi Mạnh Dũng	Tìm hiểu các thuật toán, viết các hàm insert và làm báo cáo chính	Tốt
Thái Đoàn Trường	Tìm hiểu các thuật toán, hỗ trợ quá trình tối ưu code và hỗ trợ làm báo cáo	Tốt

## TÀI LIỆU THAM KHẢO

- [1] Özsu, M. T., & Valduriez, P. (2020). *Principles of Distributed Database Systems* (4th ed.). Springer International Publishing.
- [2] Harper, F. M., & Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 1-19.
- [3] GroupLens Research. (2023). *MovieLens Dataset*. Retrieved from <https://grouplens.org/datasets/movielens/>
- [4] [bachkimn/bai\\_tap\\_lon\\_CSDL\\_phan\\_tan](#)