

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  
**KHOA CÔNG NGHỆ THÔNG TIN I**



**BÁO CÁO BÀI TẬP LỚN MÔN**  
**CƠ SỞ DỮ LIỆU PHÂN TÁN**

Các sinh viên thực hiện

B22DCCN519	Bùi Đức Mạnh
B22DCCN807	Chu Ngọc Thắng
B22DCCN154	Nguyễn Văn Duy

Tên nhóm: Nhóm 4

Tên lớp: D22-010

Giảng viên hướng dẫn: Kim Ngọc Bách

*Hà Nội – 6/2025*

## Mục lục

I. Chuẩn bị cấu hình .....	4
1. Môi trường thực thi:.....	4
2. Dữ liệu đầu vào:.....	4
3. Cấu trúc bảng Ratings:.....	4
II. Thực hiện.....	5
1. Hàm loadratings .....	5
1.1. Phân tích .....	5
1.2 Thực hiện .....	6
2. Hàm rangepartition .....	7
2.1 Phân tích .....	7
2.2 Thực hiện .....	8
3. Hàm rangeinsert .....	11
3.1. Phân tích .....	11
3.2. Thực hiện .....	12
4. Hàm roundrobinpartition .....	14
4.1. Phân tích .....	14
4.2. Thực hiện .....	15
5. Hàm roundrobininsert .....	18
5.1. Phân tích .....	18
5.2. Thực hiện .....	19
III. Kiểm thử.....	20
1. Cấu hình chung môi trường: .....	20
2. Kết quả kiểm thử.....	21
2.1. Kết quả chung.....	21
2.2. Kiểm thử Hàm LoadRatings.....	21
2.3. Kiểm thử Hàm Range_Partition .....	23
2.4. Kiểm thử Hàm Range_Insert.....	24
2.5. Kiểm thử Hàm RoundRobin_Partition .....	24

2.6. Kiểm thử Hàm RoundRobin_Insert.....	27
IV. Kết luận .....	28

#### **PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN**

<b>TT</b>	<b>Công việc / Nhiệm vụ</b>	<b>SV thực hiện</b>
1	Code hàm loadratings, kiểm thử phần mềm và viết báo cáo.	Bùi Đức Mạnh
2	Code hàm rangepartition và hàm rangeinsert, viết báo cáo về hàm đã thực hiện	Chu Ngọc Thắng
3	Code hàm roundrobinpartition và hàm roundrobininsert, viết báo cáo về hàm đã thực hiện	Nguyễn Văn Duy

# I. Chuẩn bị cấu hình

## 1. Môi trường thực thi:

- Hệ điều hành: Chuẩn bị một hệ điều hành Ubuntu hoặc Windows 10.
- Phiên bản Python: Cài đặt Python phiên bản 3.12.x hoặc cao hơn.
- Hệ quản trị cơ sở dữ liệu: Cài đặt và cấu hình một trong các hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở được hỗ trợ: PostgreSQL hoặc MySQL. Đảm bảo rằng cơ sở dữ liệu có thể được truy cập và quản lý thông qua các thư viện Python tương ứng.

## 2. Dữ liệu đầu vào:

- Nguồn dữ liệu: Tải tệp ratings.dat từ trang web MovieLens thông qua liên kết sau: <http://files.grouplens.org/datasets/movielens/ml-10m.zip>.
- Mô tả dữ liệu: Tệp ratings.dat chứa 10 triệu đánh giá và 100.000 thẻ áp dụng cho 10.000 bộ phim bởi 72.000 người dùng.
- Định dạng mỗi dòng: Mỗi dòng trong tệp đại diện cho một đánh giá và tuân theo định dạng sau:
  - UserID :: MovieID :: Rating :: Timestamp
    - + UserID: Định danh người dùng (số nguyên).
    - + MovieID: Định danh bộ phim (số nguyên).
    - + Rating: Điểm đánh giá (số thực, thang điểm 5 sao, có thể nửa sao).
    - + Timestamp: Thời điểm đánh giá (tính từ ngày 1 tháng 1 năm 1970).

## 3. Cấu trúc bảng Ratings:

- Để lưu trữ dữ liệu đánh giá, một bảng có tên Ratings sẽ được tạo trong cơ sở dữ liệu (PostgreSQL hoặc MySQL) với lược đồ (schema) sau:
  - + UserID (kiểu dữ liệu: INT)
  - + MovieID (kiểu dữ liệu: INT)
  - + Rating (kiểu dữ liệu: FLOAT hoặc REAL)

## II. Thực hiện

### 1. Hàm loadratings

#### 1.1. Phân tích

- **Vấn đề:** Hàm loadratings được tạo ra để đưa dữ liệu đánh giá phim từ một tệp tin vào một bảng trong cơ sở dữ liệu. Tệp tin này có các thông tin được ngăn cách bởi dấu hai chấm (:) và trong đó có vài thứ không cần thiết (như các cột extra1, extra2, extra3 và timestamp). Chúng ta chỉ cần lấy đúng ba cột chính là userid, movieid, và rating để lưu vào bảng cuối cùng.

- **Phương pháp:**

+ Kết nối và Chuẩn bị: Đầu tiên, hàm sẽ nhận một kết nối đến cơ sở dữ liệu (openconnection) và tạo một "con trỏ" để ra lệnh cho cơ sở dữ liệu.

+ Xóa và Tạo Lại Bảng Chính: Để đảm bảo dữ liệu mới được đưa vào một cách sạch sẽ, hàm sẽ kiểm tra và xóa bảng ratingtablename nếu nó đã có sẵn, sau đó tạo lại bảng này với ba cột cần thiết: userid (số nguyên), movieid (số nguyên), và rating (số thập phân).

+ Dùng Bảng Tạm: Để xử lý các thông tin không cần thiết, hàm sẽ tạo một bảng tạm thời (temp\_ratings) chứa tất cả các cột có trong tệp tin, bao gồm cả các cột "linh tinh" (extra) và thời gian (timestamp).

+ Chép Dữ liệu từ Tệp: Hàm dùng lệnh “cur.copy\_from” để chép toàn bộ dữ liệu từ tệp vào bảng tạm. Một cách rất nhanh và hiệu quả khi có lượng lớn dữ liệu.

+ Lọc và Chèn Dữ liệu: Sau khi dữ liệu thô đã được đưa vào bảng tạm, một câu lệnh SQL INSERT INTO ... SELECT sẽ được sử dụng để chọn lọc chỉ các cột userid, movieid, và rating từ bảng tạm (temp\_ratings) và chèn vào bảng chính ratingtablename.

+ Dọn dẹp Bảng Tạm: Cuối cùng, bảng tạm temp\_ratings sẽ được xóa đi để giải phóng bộ nhớ.

+ Đảm bảo Dữ liệu An Toàn: Hàm được đặt trong một khối try-except-finally để đảm bảo dữ liệu luôn được an toàn. Nếu có bất kỳ lỗi nào xảy ra khi tải dữ liệu, mọi thay đổi sẽ được hủy bỏ (rollback), và lỗi sẽ được báo

lại. Nếu mọi thứ diễn ra suôn sẻ, các thay đổi sẽ được lưu lại vĩnh viễn (commit) vào cơ sở dữ liệu.

### - Đánh giá:

+ Rất nhanh: Việc dùng copy\_from để đưa dữ liệu vào bảng tạm rồi lọc ra là cách siêu nhanh để xử lý các tệp dữ liệu khổng lồ, hiệu quả hơn nhiều so với việc đọc và chèn từng dòng một.

+ Dễ hiểu: Với việc dùng bảng tạm, hàm tách biệt rõ ràng việc đọc dữ liệu gốc với việc lọc và lưu trữ dữ liệu cuối cùng, làm cho đoạn mã dễ hiểu và dễ sửa chữa hơn.

+ Xử lý lỗi đáng tin cậy: Khối try-except-finally giúp đảm bảo rằng cơ sở dữ liệu luôn ở trạng thái đúng đắn, ngay cả khi có sự cố xảy ra trong quá trình tải dữ liệu.

+ Linh hoạt: Tên bảng có thể thay đổi tùy ý, giúp hàm có thể dùng lại để tải dữ liệu vào nhiều bảng khác nhau.

## 1.2 Thực hiện

### - Kết nối và Chuẩn bị

```
def loadratings(ratingtablename, ratingsfilepath, openconnection):  
    """  
    Nạp dữ liệu từ file ratingsfilepath vào bảng ratingtablename.  
    """  
  
    con = openconnection  
    cur = con.cursor()
```

### - Xóa và Tạo Lại Bảng

```
# Xóa bảng nếu đã tồn tại  
cur.execute("DROP TABLE IF EXISTS " + ratingtablename + ";")  
  
# Tạo bảng với 3 cột chính  
cur.execute("CREATE TABLE " + ratingtablename + " (userid INTEGER, movieid INTEGER, rating FLOAT);")
```

### - Dùng Bảng Tạm: Để xử lý các thông tin không cần thiết

```
try:  
    with open(ratingsfilepath, 'r') as f:  
  
        # Tạo bảng tạm chứa tất cả các trường  
        cur.execute("DROP TABLE IF EXISTS temp_ratings;")  
        cur.execute("CREATE TABLE temp_ratings (userid INTEGER, extra1 CHAR, movieid INTEGER, extra2 CHAR, rating FLOAT, extra3 CHAR, timestamp BIGINT);")
```

### - Chép Dữ liệu từ Tệp

```
f.seek(0) # Đưa con trỏ file về đầu  
cur.copy_from(f, 'temp_ratings', sep=';')
```

## - Lọc và Chèn Dữ liệu

```
# Chèn dữ liệu đã lọc vào bảng chính
cur.execute("INSERT INTO " + ratingtablename + " (userid, movieid, rating) SELECT userid, movieid, rating FROM temp_ratings;")
```

## - Dọn dẹp Bảng Tạm

```
# Xóa bảng tạm
cur.execute("DROP TABLE temp_ratings;")
```

## - Đảm bảo Dữ liệu An Toàn: Hàm được đặt trong một khối try-except-finally

```
try:
    with open(ratingsfilepath, 'r') as f:
        # Tạo bảng tạm chứa tất cả các trường
        cur.execute("DROP TABLE IF EXISTS temp_ratings;")
        cur.execute("CREATE TABLE temp_ratings (userid INTEGER, extra1 CHAR, movieid INTEGER, extra2 CHAR, rating FLOAT, extra3 CHAR, timestamp BIGINT);")

        f.seek(0) # Đưa con trỏ file về đầu
        cur.copy_from(f, 'temp_ratings', sep=';')

        # Chèn dữ liệu đã lọc vào bảng chính
        cur.execute("INSERT INTO " + ratingtablename + " (userid, movieid, rating) SELECT userid, movieid, rating FROM temp_ratings;")

        # Xóa bảng tạm
        cur.execute("DROP TABLE temp_ratings;")

    con.commit()
except Exception as e:
    con.rollback()
    raise e
finally:
    cur.close()
```

## 2. Hàm rangepartition

### 2.1 Phân tích

#### - Vấn đề:

- + Cần chia bảng Ratings thành N phân mảnh ngang dựa trên khoảng đồng đều của Rating ( $[0, 5]$ ), lưu vào các bảng range\_part

- + Phải đảm bảo tính hoàn chỉnh (mọi hàng đều được phân phối), bất giao (không có dữ liệu trùng lặp giữa các phân mảnh), và tái tạo (kết hợp phân mảnh cho ra bảng gốc).

- + Cần đồng bộ ranh giới khoảng với rangeinsert để chèn dữ liệu mới chính xác.

- + Không được mã hóa cứng tên cơ sở dữ liệu, không đóng kết nối, và phải xử lý các bảng phân mảnh cũ từ lần chạy trước.

#### - Phương pháp:

- + Chia khoảng đồng đều: Chia  $[0, 5]$  thành N khoảng bằng nhau, tính ranh giới và lưu vào bảng metadata để đồng bộ với rangeinsert.

+ Sử dụng metadata: Lưu số phân mảnh (num\_partitions), loại phân mảnh (partition\_type) và ranh giới (range\_boundaries) dưới dạng chuỗi TEXT phân tách bằng dấu phẩy.

+ Xóa bảng cũ: Xóa các bảng range\_part cũ để tránh dữ liệu thừa.

+ Phân phối dữ liệu: Dùng câu lệnh SQL để chèn dữ liệu từ bảng Ratings gốc vào range\_part dựa trên điều kiện khoảng ranh giới với  $\geq$  cho khoảng đầu,  $>$  cho các khoảng sau).

+ Không xóa bảng Ratings gốc: Giữ dữ liệu bảng Ratings gốc để phù hợp yêu cầu trong hàm rangeinsert khi cần chèn vào bảng dữ liệu gốc.

+ Quản lý lỗi: Sử dụng try-except để rollback nếu có lỗi, đóng cursor đúng cách.

## - Đánh giá

+ Lược đồ, tên bảng đúng.

+ Phân mảnh đồng đều.

+ Đảm bảo hoàn chỉnh, bất giao, tái tạo.

+ Sử dụng metadata.

+ Không mã hóa cứng, không đóng kết nối.

## 2.2 Thực hiện

- **Kiểm tra đầu vào:** Thoát chương trình nếu số phân mảnh không hợp lệ (numberofpartitions  $\leq 0$ ) để đảm bảo chương trình không hoạt động với dữ liệu không hợp lệ.

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):  
    if numberofpartitions <= 0:  
        return
```

- **Khai báo con trỏ:** Sử dụng openconnection.cursor() để khởi tạo con trỏ truy vấn

```
connection = None  
cursor = None  
try:  
    connection = openconnection  
    cursor = connection.cursor()
```



- **Xóa bảng cũ:** Truy vấn từ pg\_tables để tìm các bảng có tên bắt đầu với tiền tố range\_part, xóa bằng lệnh DROP TABLE IF EXISTS để đảm bảo không có dữ liệu thừa từ lần chạy trước.

```
RANGE_TABLE_PREFIX = 'range_part'

# Xóa các bảng phân vùng cũ
cursor.execute("""
    SELECT tablename
    FROM pg_tables
    WHERE tablename LIKE 'range_part%';
""")
old_tables = cursor.fetchall()
for table_tuple in old_tables:
    table_name = table_tuple[0]
    cursor.execute(f"DROP TABLE IF EXISTS {table_name};")
```

## - Tạo bảng metadata

```
# Tạo bảng metadata để lưu thông tin phân vùng
cursor.execute("""
    CREATE TABLE IF NOT EXISTS metadata (
        partition_type TEXT,
        num_partitions INTEGER,
        range_boundaries TEXT
    );
""")

cursor.execute("DELETE FROM metadata WHERE partition_type = 'range';")
```

+ Tạo bảng metadata với các thuộc tính partition\_type, num\_partitions, range\_boundaries với partition\_type là chuỗi lưu loại phân mảnh, num\_partitions là một số để nhận dạng số phân mảnh có trong CSDL và range\_boundaries là chuỗi số đại diện cho ranh giới của các phân mảnh.

+ Xóa dữ liệu cũ trong bảng metadata để tránh trùng lặp với dữ liệu mới chèn vào.

## - Tính ranh giới và chèn dữ liệu vào metadata

```
# Tính toán ranh giới cho các phân vùng
max_rating = 5.0
min_rating = 0.0
delta = (max_rating - min_rating) / numberofpartitions
boundaries = [min_rating + i * delta for i in range(numberofpartitions + 1)]

# Làm tròn lên 2 chữ số thập phân
boundaries = [round(b * 100 + 0.5) / 100 for b in boundaries]
boundaries_str = ",".join(str(b) for b in boundaries)

# Lưu thông tin phân vùng vào metadata
cursor.execute("""
    INSERT INTO metadata (partition_type, num_partitions, range_boundaries)
    VALUES (%s, %s, %s);
""", ('range', numberofpartitions, boundaries_str))
```

+ Chia  $[0, 5]$  thành  $N$  khoảng với  $\delta$  sẽ là khoảng cách giá trị rating giữa các mảnh.  $\delta$  được tính toán bằng khoảng giá trị của rating và chia cho số phân mảnh cần chia.

+ Tạo boundaries: là một mảng chứa các giới hạn của từng phân mảnh, mỗi cặp số liền kề sẽ là giới hạn trên và dưới của một phân mảnh. Các cặp giá trị giới hạn ứng với từng phân mảnh tăng dần. Mảng boundaries sẽ có dạng  $[0.0, \delta, 2*\delta, \dots, 5.0]$ .

+ Ví dụ:  $N=2 \rightarrow \text{boundaries} = [0.0, 2.5, 5.0]$ ;  $N=3 \rightarrow [0.0, 1.6667, 3.3333, 5.0]$ .

+ Lưu vào boundaries\_str là chuỗi phân tách bởi dấu phẩy, làm tròn 2 chữ số thập phân, ví dụ: "0.0,1.67,3.34,5.0".

+ Lưu thông tin số phân mảnh, loại phân mảnh là 'range' và chuỗi ranh giới vào bảng metadata bằng lệnh insert để đồng bộ cho việc chèn dữ liệu.

## - Phân mảnh

```
# Tạo và phân phối dữ liệu cho các phân vùng
for i in range(numberofpartitions):
    table_name = f"{RANGE_TABLE_PREFIX}{i}"
    cursor.execute(f"""
        CREATE TABLE IF NOT EXISTS {table_name} (
            userid INTEGER,
            movieid INTEGER,
            rating FLOAT
        );
    """)
    cursor.execute(f"DELETE FROM {table_name};")

    if i == 0:
        cursor.execute(f"""
            INSERT INTO {table_name} (userid, movieid, rating)
            SELECT userid, movieid, rating FROM {ratingtablename}
            WHERE rating >= %s AND rating <= %s;
        """, (boundaries[i], boundaries[i + 1]))
    else:
        cursor.execute(f"""
            INSERT INTO {table_name} (userid, movieid, rating)
            SELECT userid, movieid, rating FROM {ratingtablename}
            WHERE rating > %s AND rating <= %s;
        """, (boundaries[i], boundaries[i + 1]))
```

+ Tạo bảng phân mảnh với các thuộc tính giống với bảng rating gốc với bảng phân mảnh có tiền tố RANGE\_TABLE\_PREFIX= 'range\_part'.

+ Xóa dữ liệu cũ trong các phân mảnh nếu có tồn tại dữ liệu cũ để tránh trùng lặp.

+ Phân phối dữ liệu:

Khoảng đầu tiên ( $i == 0$ ):  $\text{rating} \geq \text{boundaries}[0]$  AND  $\text{rating} \leq \text{boundaries}[1]$  bao gồm ranh giới dưới với phân mảnh đầu tiên.

Các khoảng sau ( $i > 0$ ):  $\text{rating} > \text{boundaries}[i]$  AND  $\text{rating} \leq \text{boundaries}[i + 1]$  không bao gồm ranh giới dưới với các phân mảnh còn lại.

Ví dụ: Với  $N=2$ ,  $\text{rating}=2.5$  chỉ thuộc `range_part0`, đảm bảo bất giao.

+ Sử dụng `INSERT INTO ... SELECT ...` để phân phối dữ liệu từ `Ratings`, đảm bảo hoàn chỉnh tất cả hàng đều được phân phối.

- **Không đóng kết nối:** Không đóng kết nối, chỉ đóng cursor và rollback lại khi gặp một ngoại lệ nào đó.

```
connection.commit()
except Exception as e:
    if connection:
        connection.rollback()
    raise e
finally:
    if cursor:
        cursor.close()
```

### 3. Hàm `rangeinsert`

#### 3.1. Phân tích

##### - Vấn đề

+ Chèn bộ (`userid`, `itemid`, `rating`) vào bảng `Ratings` và đúng phân mảnh `range_part` dựa trên `rating`.

+ Phải đồng bộ với ranh giới trong `rangepartition` để đảm bảo dữ liệu mới được chèn đúng phân mảnh.

+ Hỗ trợ chèn nhiều lần bất kỳ lúc nào.

+ Đảm bảo đầu vào hợp lệ, đúng tính chất phân mảnh và xử lý lỗi an toàn.

##### - Phương pháp

+ Chèn vào cả `Ratings` và phân mảnh: Chèn vào `Ratings` để giữ dữ liệu gốc, đồng thời chèn vào `range_part` dựa trên `rating`.

+ Đọc metadata: Lấy số phân mảnh hiện tại trong bảng metadata để xác định chỉ số phân mảnh cần chèn vào.

+ Kiểm tra đầu vào: Xác minh rating trong  $[0, 5]$ , userid và itemid là số nguyên dương. Dữ liệu đó đã tồn tại trong bảng Ratings gốc chưa.

+ Quản lý lỗi: Rollback nếu có lỗi, đóng cursor, không đóng kết nối.

### -Đánh giá

+ Hỗ trợ chèn nhiều lần do hàm đọc metadata động, không phụ thuộc vào bất kỳ thông tin tĩnh hoặc trạng thái được lưu trữ từ các lần gọi trước đó.

+ Hàm xử lý độc lập từng bản ghi và sử dụng connection.commit() để xác nhận giao dịch sau khi chèn thành công, cho phép hàm được gọi lại ngay cả khi lần gọi trước thất bại.

+ Chèn đúng vào bảng Ratings gốc và đúng phân mảnh.

+ Không đóng kết nối.

+ Kiểm tra giá trị đầu vào để đảm bảo chương trình không gặp nhiều lỗi.

## 3.2. Thực hiện

### - Kiểm tra đầu vào:

```
def rangeinsert(ratingtablename, userid, itemid, rating, openconnection):  
    if not (0 <= rating <= 5):  
        raise Exception("Rating must be between 0 and 5")  
    if not (isinstance(userid, int) and isinstance(itemid, int) and userid > 0 and itemid > 0):  
        raise Exception("UserID and MovieID must be positive integers.")
```

+ Kiểm tra rating nằm trong khoảng  $[0, 5]$ .

+ Kiểm tra userid và itemid là số nguyên dương:

### - Khai báo con trỏ: Sử dụng openconnection.cursor() để khởi tạo con

trỏ truy vấn

```
connection = None  
cursor = None  
try:  
    connection = openconnection  
    cursor = connection.cursor()
```

```
# Kiểm tra xem dữ liệu đã tồn tại trong bảng gốc chưa
cursor.execute(f"""
    SELECT 1 FROM {ratingtablename}
    WHERE userid = %s AND movieid = %s AND rating = %s;
""", (userid, itemid, rating))

if cursor.fetchone():
    print("Dữ liệu đã tồn tại trong Ratings")
    return
```

+ Kiểm tra xem dữ liệu cả 3 trường userId, movieId và rating đã tồn tại trong bảng Ratings gốc chưa.

+ Nếu đã tồn tại in ra thông báo và dừng chương trình.

+ Đảm bảo đúng tính chất phân mảnh mỗi dữ liệu chỉ được xuất hiện tại 1 phân mảnh duy nhất.

### - Chèn vào bảng Ratings gốc:

+ Sử dụng câu lệnh SQL:

```
RANGE_TABLE_PREFIX = 'range_part'

# Chèn dữ liệu vào bảng chính
cursor.execute(f"""
    INSERT INTO {ratingtablename} (userid, movieid, rating)
    VALUES (%s, %s, %s);
""", (userid, itemid, rating))
```

+ Đáp ứng yêu cầu đề bài rangeinsert phải chèn vào bảng Ratings gốc.

### - Đọc thông tin từ metadata:

```
# Lấy thông tin phân vùng từ metadata
cursor.execute("SELECT num_partitions FROM metadata WHERE partition_type = 'range';")
num_partitions = cursor.fetchone()[0]

# Tính trực tiếp phân mảnh dựa trên rating
partition_size = round(5.0 / num_partitions, 2)
partition_index = int(rating / partition_size)
if rating % partition_size == 0 and rating != 0:
    partition_index -= 1
```

- Truy vấn bảng metadata để lấy num\_partitions.

- Tính chỉ số phân mảnh cần chèn vào:

+ Chia giá trị rating của dữ liệu cần chèn cho partition\_size và lấy giá trị nguyên để xác định phân mảnh

+ Ví dụ: N=5, rating=3, partition\_size=1.0 → partition\_index = int(3 / 1.0) =

+ Nếu rating nằm trên ranh giới và không phải 0, giảm partition\_index để khớp với các ranh giới khi phân mảnh đầu lấy giá trị cận dưới, các mảnh sau không lấy giá trị cận dưới.

+ Ví dụ:  $N=2$ ,  $partition\_size=2.5$ ,  $rating=2.5 \rightarrow 2.5 \% 2.5 = 0$ ,  $partition\_index = \text{int}(2.5 / 2.5) = 1$ , sau điều chỉnh  $partition\_index = 0 \rightarrow$  chèn vào range\_part0

#### - Chèn vào bảng phân mảnh đúng:

```
# Chèn vào phân mảnh tương ứng
table_name = f"{RANGE_TABLE_PREFIX}{partition_index}"
cursor.execute(f"""
    INSERT INTO {table_name} (userid, movieid, rating)
    VALUES (%s, %s, %s);
""", (userid, itemid, rating))
```

+ Tạo tên bảng bằng tiền tố RANGE\_TABLE\_PREFIX = "range\_part" và chỉ số phân mảnh tìm được ở phần trên.

+ Chèn vào bảng bằng lệnh INSERT INTO

+ Ví dụ:  $rating=3$ ,  $N=5 \rightarrow$  chèn vào range\_part2.

#### - Quản lý kết nối và lỗi:

```
connection.commit()
except Exception as e:
    if connection:
        connection.rollback()
    raise e
finally:
    if cursor:
        cursor.close()
```

+ Không đóng connection, chỉ đóng cursor, đáp ứng yêu cầu.

+ Sử dụng try-except để rollback nếu có lỗi, đảm bảo tính toàn vẹn dữ liệu.

## 4. Hàm roundrobinpartition

### 4.1. Phân tích

Hàm roundrobinpartition thực hiện phân mảnh dữ liệu từ một bảng đánh giá (ratingtablename) thành nhiều bảng con theo phương pháp Round-Robin, dựa trên số lượng phân mảnh (numberofpartitions) được chỉ định. Mục tiêu là phân phối đều các bản ghi trên các phân mảnh để đảm bảo cân bằng tải và tối ưu hóa truy vấn.

#### - Vấn đề:

- + Phân chia dữ liệu từ bảng chính thành `numberofpartitions` bảng con (`rrobin_part0`, `rrobin_part1`, ...).
- + Đảm bảo mỗi bản ghi từ bảng chính được phân bổ tuần tự vào các bảng con theo thứ tự vòng tròn (Round-Robin).
- + Lưu trữ thông tin metadata (số lượng phân mảnh và chỉ số tiếp theo) để hỗ trợ việc chèn dữ liệu sau này.
- + Khi dữ liệu lớn, việc truy xuất và quản lý hiệu quả trở nên quan trọng, cần phải tối ưu hiệu suất truy vấn.

#### **- Phương pháp:**

- + Kiểm tra số lượng phân mảnh hợp lệ (`numberofpartitions > 0`).
- + Xóa các bảng phân mảnh cũ (nếu có) và tạo lại các bảng mới với cấu trúc tương tự bảng chính (`userid`, `movieid`, `rating`).
- + Tạo bảng `rrobin_metadata` để lưu số lượng phân mảnh và chỉ số tiếp theo cho việc chèn.
- + Sử dụng buffer ảo (`io.StringIO`) để tối ưu hóa việc chèn dữ liệu vào từng phân mảnh, giảm số lần truy vấn cơ sở dữ liệu.
- + Phân phối dữ liệu bằng cách sử dụng chỉ số bản ghi (`i % numberofpartitions`) để xác định bảng con đích cần chèn.
- + Sử dụng `copy_from` để chèn dữ liệu hiệu quả từ buffer vào các bảng con.

#### **- Đánh giá:**

- + Phân phối dữ liệu đều, đảm bảo mỗi bảng con có số lượng bản ghi gần bằng nhau.
- + Tối ưu hóa hiệu suất bằng cách sử dụng buffer ảo và `copy_from` thay vì chèn từng dòng.
- + Quản lý metadata giúp đồng bộ hóa quá trình chèn dữ liệu sau này.
- + Không hỗ trợ phân mảnh động (thêm hoặc xóa phân mảnh sau khi tạo).

## **4.2. Thực hiện**

**Kiểm tra điều kiện đầu vào:** Nếu `numberofpartitions <= 0`, hàm thoát ngay.

```
def roundrobinpartition(ratingstablename, numberofpartitions,
openconnection):
    if numberofpartitions <= 0:
        return
```

**Khởi tạo cursor:** Sử dụng `openconnection.cursor()` để thực hiện các truy vấn SQL.

```
try:
    cur = openconnection.cursor()
```

### Xóa và tạo bảng phân mảnh

- + Xóa các bảng cũ có tiền tố `rrobin_part`.
- + Tạo các bảng mới (`rrobin_part0`, `rrobin_part1`, ...) với các cột `userid` (INT), `movieid` (INT), `rating` (FLOAT).

```
# Xóa và tạo lại các bảng phân mảnh
for i in range(numberofpartitions):
    cur.execute(f"DROP TABLE IF EXISTS {RROBIN_TABLE_PREFIX}{i};")
    cur.execute(f"""
        CREATE TABLE {RROBIN_TABLE_PREFIX}{i} (
            userid INT,
            movieid INT,
            rating FLOAT
        );
    """)
```

### Quản lý metadata

- + Xóa bảng `rrobin_metadata` cũ (nếu có).
- + Tạo bảng `rrobin_metadata` với các cột `partition_count` và `next_index`.
- + Khởi tạo giá trị `partition_count` = `numberofpartitions`, `next_index` = 0.



```
# Tạo lại bảng metadata lưu tổng số phân mảnh và chỉ số tiếp theo
của bảng cần chèn
cur.execute("DROP TABLE IF EXISTS rrobin_metadata;")
cur.execute("CREATE TABLE rrobin_metadata (partition_count INT,
next_index INT);")
cur.execute("INSERT INTO rrobin_metadata VALUES (%s, 0);",
(numberofpartitions,))
```

## Phân phối dữ liệu

- + Lấy toàn bộ dữ liệu từ bảng chính (ratingtablename) bằng truy vấn SELECT.
- + Sử dụng danh sách buffer (io.StringIO) để lưu dữ liệu tạm thời cho mỗi phân mảnh.
- + Ghi từng bản ghi vào buffer tương ứng với phân mảnh được tính bằng  $i \% \text{numberofpartitions}$ .
- + Sử dụng copy\_from để chèn dữ liệu từ buffer vào bảng phân mảnh tương ứng.

```
# Lấy dữ liệu từ bảng chính
cur.execute(f"SELECT userid, movieid, rating FROM {ratingtablename};")
rows = cur.fetchall()

# Chuẩn bị buffer dạng file ảo theo từng phân mảnh, mỗi buffer sẽ chứa dữ liệu cho
một phân mảnh
buffers = [io.StringIO() for _ in range(numberofpartitions)]

for i, row in enumerate(rows):
    partition_index = i % numberOfpartitions
    line = f"{row[0]}\t{row[1]}\t{row[2]}\n" #lấy dữ liệu cột userid, movieid,
    rating và phân tách bằng tab
    buffers[partition_index].write(line)

for i in range(numberofpartitions):
    # Đặt lại vị trí con trỏ đầu mỗi buffer
    buffers[i].seek(0)
    cur.copy_from(buffers[i], f"{RROBIN_TABLE_PREFIX}{i}", columns=("userid",
    "movieid", "rating"))
```

## Quản lý giao dịch

- + Commit giao dịch nếu thành công (openconnection.commit()).

- + Rollback nếu có lỗi (openconnection.rollback()).
- + Đóng cursor trong khối finally.

```
openconnection.commit()

except Exception as e:
    openconnection.rollback()
    raise e
finally:
    if cur:
        cur.close()
```

## 5. Hàm roundrobininsert

### 5.1. Phân tích

Hàm roundrobininsert chèn một bản ghi đánh giá mới (userid, itemid, rating) vào bảng chính (ratingstablename) và vào đúng phân mảnh theo cách round robin.

#### - Vấn đề:

- + Chèn bản ghi mới vào bảng chính và bảng phân mảnh phù hợp theo cơ chế Round-Robin.

- + Cần kiểm tra dữ liệu chèn đã tồn tại hay thỏa mãn chưa.

#### - Phương pháp:

- + Chèn bản ghi vào bảng chính (ratingstablename) trước.

- + Lấy thông tin partition\_count và next\_index từ bảng rrobin\_metadata.

- + Tính phân mảnh đích bằng next\_index % partition\_count.

- + Chèn bản ghi vào bảng phân mảnh tương ứng (rrobin\_partX).

- + Cập nhật next\_index trong rrobin\_metadata để chuẩn bị cho lần chèn tiếp theo.

#### - Đánh giá:

- + Đảm bảo tính nhất quán giữa bảng chính và bảng phân mảnh.
- + Tự động chọn phân mảnh đích dựa trên next\_index, duy trì phân phối đều.
- + Quản lý lỗi tốt với cơ chế commit/rollback.

## 5.2. Thực hiện

### Xác thực đầu vào

```
def roundrobininsert(ratingtablename, userid, itemid, rating,
openconnection):
    if not (0 <= rating <= 5):
        raise Exception("Rating must be between 0 and 5.")
    if not (isinstance(userid, int) and isinstance(itemid, int) and
userid > 0 and itemid > 0):
        raise Exception("UserID and MovieID must be positive integers.")
```

### Khởi tạo cursor

- + Sử dụng openconnection.cursor() để thực hiện các truy vấn SQL.

```
def roundrobininsert(ratingtablename, userid, itemid, rating, openconnection):
    try:
        cur = openconnection.cursor()
```

### Chèn vào bảng chính

- + Thực hiện truy vấn INSERT để thêm bản ghi (userid, itemid, rating) vào ratingtablename.

```
# Chèn vào bảng Ratings
cur.execute(f"INSERT INTO {ratingtablename} (userid, movieid, rating) VALUES (%s,
%s, %s);", (userid, itemid, rating))
```

### Lấy thông tin metadata

- + Truy vấn bảng rrobin\_metadata để lấy partition\_count và next\_index.

```
# Lấy thông tin metadata
cur.execute("SELECT partition_count, next_index FROM rrobin_metadata;")
partition_count, next_index = cur.fetchone()
```

### Chèn vào bảng phân mảnh

- + Tính chỉ số phân mảnh đích bằng `next_index % partition_count`.
- + Chèn bản ghi vào bảng `rrobin_partX` tương ứng.

```
# Tính phân mảnh tiếp theo cần chèn
target_partition = next_index % partition_count
cur.execute(f"INSERT INTO {RROBIN_TABLE_PREFIX}{target_partition} (userid, movieid,
rating) VALUES (%s, %s, %s);", (userid, itemid, rating))
```

### Cập nhật metadata:

- + Tăng `next_index` lên 1 và cập nhật vào bảng `rrobin_metadata`.

```
# Cập nhật chỉ số vòng tròn
cur.execute("UPDATE rrobin_metadata SET next_index = %s;", (next_index + 1,))
```

### Quản lý giao dịch:

- + Commit giao dịch nếu thành công (`openconnection.commit()`).
- + Rollback nếu có lỗi (`openconnection.rollback()`).
- + Đóng cursor trong khối `finally`.

```
openconnection.commit()
except Exception as e:
    openconnection.rollback()
    raise e
finally:
    if cur:
        cur.close()
```

## III. Kiểm thử

### 1. Cấu hình chung môi trường:

- + Hệ điều hành: Windows 10.
- + Phiên bản Python: 3.12.11.
- + Hệ quản trị cơ sở dữ liệu: PostgreSQL.

### - Dữ liệu kiểm thử:

+ Sử dụng tệp ratings.dat đầy đủ (được cung cấp trong repository) để kiểm tra các hàm trong quá trình phát triển

+ Đảm bảo cơ sở dữ liệu đã được khởi tạo và không chứa dữ liệu cũ trước mỗi lần kiểm thử quan trọng.

### - Công cụ kiểm thử:

+ Sử dụng VisualStudioCode để chạy file Assignment1Tester.py

+ Sử dụng postgresSQL để kiểm tra xem dữ liệu có đúng mong muốn hay không.

## 2. Kết quả kiểm thử

### 2.1. Kết quả chung

Đánh giá tổng quan về hiệu suất và tính đúng đắn của toàn bộ hệ thống phân mảnh dữ liệu. Các tiêu chí đánh giá bao gồm:

+ Đảm bảo tất cả các test case được chạy qua mà không có lỗi biên dịch.

+ Kiểm tra tính toàn vẹn dữ liệu trong các bảng phân mảnh.

+ Đảm bảo dữ liệu được phân bố đúng cách vào các phân mảnh theo thuật toán đã chọn.

+ Đánh giá hiệu suất của các hàm khi xử lý tập dữ liệu lớn.

```
PS D:\Năm 3\Kỳ 2\Cơ sở dữ liệu phân tán\BTL-CSDLPT\src> python Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
Press enter to Delete all tables? [
```

### 2.2. Kiểm thử Hàm LoadRatings

**Mô tả:** Kiểm tra khả năng tải dữ liệu từ tệp ratings.dat vào bảng Ratings chính.

**Kịch bản:**

+ Thực thi hàm LoadRatings() với đường dẫn tuyệt đối đến tệp ratings.dat.

+ Sau khi thực thi, kiểm tra số lượng bản ghi trong bảng Ratings và một vài bản ghi ngẫu nhiên để xác nhận dữ liệu được tải chính xác (đúng schema, đúng giá trị).

### Kết quả:

+ Hàm loadratings vượt qua test, không bị lỗi biên dịch

```
PS D:\Năm 3\Kỳ 2\Cơ sở dữ liệu phân tán\BTL-CSDLPT\src> python Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
loadratings function pass!
```

+ Số lượng data được thêm vào đúng.

> Procedures				
> 1.3 Sequences				
✓ Tables (1)				
✓ ratings				
> Columns				
> Constraints				
> Indexes				
> RLS Policies				
> Rules				
> Triggers				
> Trigger Functions				
> Types				
> Views				
Subscriptions				
postgres				
test				
gin/Group Roles				
blespaces				
	userid	movieid	rating	
	integer	integer	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	490	5	
Total rows: 10000054			Query complete 00:00:02.714	

+ Thời gian chạy ổn định

## 2.3. Kiểm thử Hàm Range\_Partition

**Mô tả:** Kiểm tra khả năng phân mảnh bảng Ratings theo phương pháp phân mảnh dựa trên dải giá trị (range partition).

**Kịch bản:**

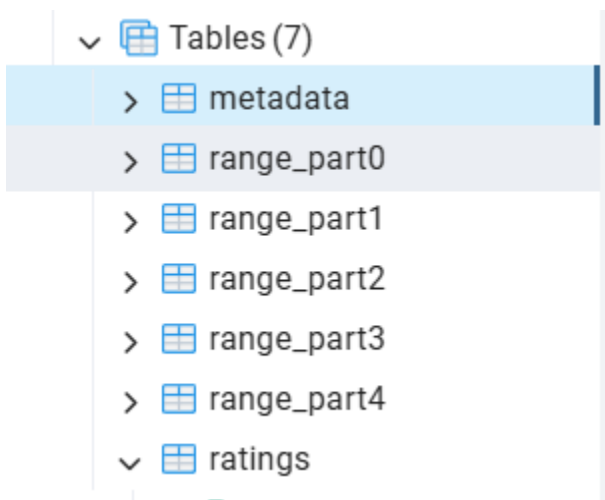
- + Thực thi hàm Range\_Partition() với bảng Ratings và các giá trị N khác nhau (ví dụ: N=1, N=2, N=3).
- + Với mỗi giá trị N, kiểm tra số lượng bảng phân mảnh được tạo (range\_part0, range\_part1,...).
- + Kiểm tra nội dung của từng bảng phân mảnh để đảm bảo các giá trị Rating nằm đúng trong dải được quy định (ví dụ: [0, 2.5] cho Partition 0 khi N=2).

**Kết quả:**

- + Hàm rangepartition vượt qua test, không bị lỗi biên dịch

```
PS D:\Năm 3\Kỳ 2\Cơ sở dữ liệu phân tán\BTL-CSDLPT\src> python Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
rangepartition function pass!
```

- + Với N = 5 đã tạo ra 5 phân mảnh



- + Các dải phân mảnh đúng những gì mong muốn

> { } Procedures									
> 1.3 Sequences									
✓ Tables (7)									
✓ metadata									
> Columns									
> Constraints									

	partition_type	num_partitions	range_boundaries
	text	integer	text
1	range	5	0.0,1.0,2.0,3.0,4.0,5.0

+ Dữ liệu đc thêm đúng vào các phân mảnh

## 2.4. Kiểm thử Hàm Range\_Insert

**Mô tả:** Kiểm tra khả năng chèn một bộ dữ liệu mới vào đúng phân mảnh theo phương pháp range partition.

### Kịch bản:

+ Sau khi bảng đã được phân mảnh bằng Range\_Partition, thực thi hàm Range\_Insert() với các giá trị UserID, MovieID, Rating để kiểm tra.

+ Thực hiện việc kiểm thử trong các trường hợp: Chèn 1 bản ghi chuẩn, chèn 1 bản ghi đã tồn tại trong bảng ratings gốc và chèn 2 bản ghi khác nhau cùng lúc.

+ Kiểm tra số lượng bản ghi trong bảng Ratings tổng và trong từng bảng phân mảnh để xác nhận bộ dữ liệu mới được chèn vào đúng vị trí.

### Kết quả:

- TH1:

+ Thực hiện việc chèn 1 bản ghi chuẩn: Hàm rangeInsert vượt qua test, không bị lỗi biên dịch

```
# 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:

rangeinsert function pass!
PS D:\Download\CSDLPT\BTL-CSDLPT\src>
```

+ Bản ghi được chèn vào bảng ratings gốc và chèn vào đúng phân mảnh range\_part2.



Procedures			
1.3 Sequences			
Tables (7)			
metadata			
range_part0			
range_part1			
range_part2			
range_part3			
range_part4			
ratings			
Trigger Functions			
Types			
Views			
Subscriptions			
postgres			
gin/Group Roles			
rolespaces			

	userid integer	movieid integer	rating double precision
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	3
Total rows: 10000055		Query complete 00:00:03.4	

Procedures			
1.3 Sequences			
Tables (7)			
metadata			
range_part0			
range_part1			
range_part2			
range_part3			
range_part4			
ratings			
Trigger Functions			
Types			
Views			
Subscriptions			
postgres			
gin/Group Roles			
rolespaces			

	userid integer	movieid integer	rating double
2726843	71566	434	3
2726844	71567	32	3
2726845	71567	256	3
2726846	71567	1396	3
2726847	71567	1580	3
2726848	71567	1584	3
2726849	71567	1690	3
2726850	71567	1748	3
2726851	71567	1769	3
2726852	71567	1833	3
2726853	71567	1876	3
2726854	71567	2012	3
2726855	100	2	3
Total rows: 2726855		Query complete 00:00:00.779	

- TH2:

+ Thực hiện chèn 1 bản ghi đã tồn tại trong bảng ratings gốc, sử dụng luôn bản ghi vừa chèn để kiểm thử: Hàm rangeInsert vượt qua test, không bị lỗi biên dịch nhưng thông báo dữ liệu đã tồn tại và chèn không thành công

```
# 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:
```

Dữ liệu đã tồn tại trong Ratings  
rangeinsert function pass!  
PS D:\Download\CSDLPT\BTL-CSDLPT\src> |

+ Bảng ratings gốc và các phân mảnh khác vẫn giữ nguyên như trường hợp 1, không tăng số lượng bản ghi lên.

- TH3:

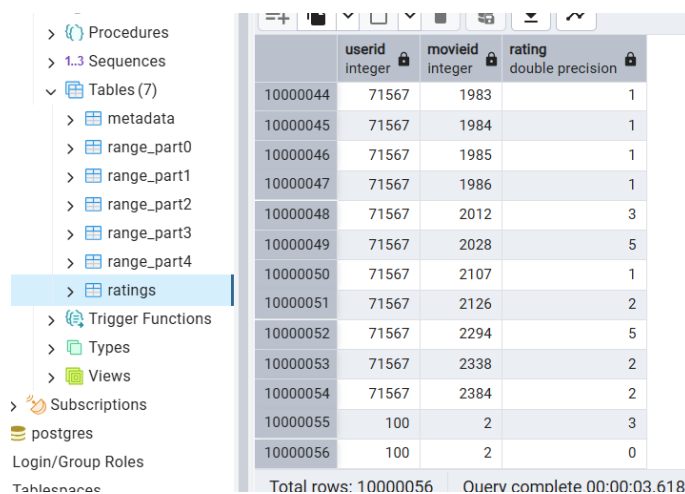
+ Chèn 2 bản ghi khác nhau cùng lúc: Hàm rangeInsert vượt qua test, không bị lỗi biên dịch.

```
# 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')
```

```
PS D:\Download\CSDLPT\BTL-CSDLPT\src> python .\Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assign1" already exists
rangeinsert function pass!
PS D:\Download\CSDLPT\BTL-CSDLPT\src>
```

+ Dữ liệu được chèn vào bảng ratings gốc và vào 2 phân mảnh range\_part0 và range\_part2.

### Bảng ratings



The screenshot shows the PostgreSQL Enterprise Manager interface. On the left, a tree view displays the database structure, including 'Tables (7)' with partitions 'range\_part0' through 'range\_part4', and the main 'ratings' table. The 'ratings' table is selected, and its data is displayed in a table view on the right. The table has three columns: 'userid' (integer), 'movieid' (integer), and 'rating' (double precision). The data shows 100,000 rows, with the last row (100,000,056) having a rating of 0. The status bar at the bottom indicates 'Total rows: 10000056' and 'Query complete 00:00:03.618'.

userid	movieid	rating
10000044	71567	1983
10000045	71567	1984
10000046	71567	1985
10000047	71567	1986
10000048	71567	2012
10000049	71567	2028
10000050	71567	2107
10000051	71567	2126
10000052	71567	2294
10000053	71567	2338
10000054	71567	2384
10000055	100	2
10000056	100	2

### Phân mảnh range\_part0

> Procedures				
> 1.3 Sequences				
> Tables (7)				
> metadata				
> range_part0				
> range_part1				
> range_part2				
> range_part3				
> range_part4				
> ratings				
> Trigger Functions				
> Types				
> Views				
> Subscriptions				
postgres				
Login/Group Roles				
Tablespaces				

	userid integer	movieid integer	rating double precision	
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 00:00:00.245

## Phân mảnh range\_part2

> Procedures				
> 1.3 Sequences				
> Tables (7)				
> metadata				
> range_part0				
> range_part1				
> range_part2				
> range_part3				
> range_part4				
> ratings				
> Trigger Functions				
> Types				
> Views				
> Subscriptions				
postgres				
Login/Group Roles				
Tablespaces				

	userid integer	movieid integer	rating double precision	
2726843	71566	434		3
2726844	71567	32		3
2726845	71567	256		3
2726846	71567	1396		3
2726847	71567	1580		3
2726848	71567	1584		3
2726849	71567	1690		3
2726850	71567	1748		3
2726851	71567	1769		3
2726852	71567	1833		3
2726853	71567	1876		3
2726854	71567	2012		3
2726855	100	2		3

Total rows: 2726855    Query complete 00:00:00.829

## 2.5. Kiểm thử Hàm RoundRobin\_Partition

**Mô tả:** Kiểm tra khả năng phân mảnh bảng Ratings theo phương pháp phân mảnh vòng tròn (round robin partition).

### Kịch bản:

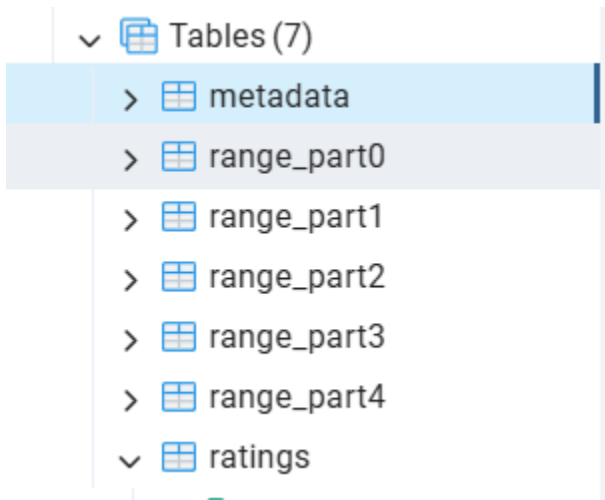
- + Thực thi hàm Range\_Partition() với bảng Ratings và các giá trị N khác nhau (ví dụ: N=1, N=2, N=3).
- + Với mỗi giá trị N, kiểm tra số lượng bảng phân mảnh được tạo (range\_part0, range\_part1,...).
- + Kiểm tra nội dung của từng bảng phân mảnh để đảm bảo các giá trị Rating nằm đúng trong dải được quy định (ví dụ: [0, 2.5] cho Partition 0 khi N=2).

### Kết quả:

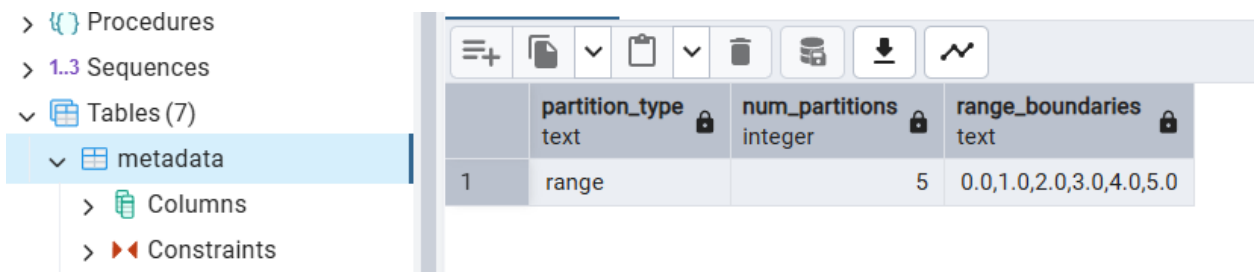
+ Hàm rangepartition vượt qua test, không bị lỗi biên dịch

```
PS D:\Năm 3\Kỳ 2\Cơ sở dữ liệu phân tán\BTL-CSDLPT\src> python Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
rangepartition function pass!
```

+ Với  $N = 5$  đã tạo ra 5 phân mảnh



+ Các dải phân mảnh đúng những gì mong muốn



+ Dữ liệu đc thêm đúng vào các phân mảnh

## 2.6. Kiểm thử Hàm RoundRobin\_Insert

**Mô tả:** Kiểm tra khả năng chèn một bộ dữ liệu mới vào đúng phân mảnh theo phương pháp round robin partition.

**Kịch bản:**

+ Sau khi bảng đã được phân mảnh bằng RoundRobin\_Partition, thực thi hàm RoundRobin\_Insert() với các giá trị UserID, MovieID, Rating khác nhau.

+ Kiểm tra số lượng bản ghi trong bảng Ratings tổng và trong từng bảng phân mảnh để xác nhận bộ dữ liệu mới được chèn vào đúng vị trí theo quy tắc vòng tròn.

## Kết quả:

- TH1:

+ Thực hiện việc chèn 1 bản ghi chuẩn vào : Hàm RoundRobin\_Insert vượt qua test, không bị lỗi biên dịch

```
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, '0')

$ py Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
roundrobininsert function pass!
```

+ Kết quả trong cơ sở dữ liệu:

Query Query History

1 select \* from ratings where userid = 100 and movieid = 1 and rating = 3

Data Output Messages Notifications

Showing rows: 1 to 1 Page No:

	userid integer	movieid integer	rating double precision
1	100	1	3

-TH2:

+ Thực hiện chèn 2 bản ghi giống nhau vào bảng Ratings và phân mảnh thứ nhất: kết quả báo lỗi “Dữ liệu đã tồn tại trong bảng Ratings”

```
testing sequence.
[result, e] = testHelper.testroundrobininsert
(MyAssignment, RATINGS_TABLE, 100, 2, 5, conn, '2')
[result, e] = testHelper.testroundrobininsert
(MyAssignment, RATINGS_TABLE, 100, 2, 5, conn, '2')
# [result, e] = testHelper.testroundrobininsert

$ py Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
Traceback (most recent call last):
  File "D:\Tài liệu Ptit\CSDLPT\BTL-CSDLPT\src\testHelper.py", line 264, in testroundrobininsert
    raise Exception(
        'Round robin insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format(userid, itemid, rating,
                                                    expectedtablename))
Exception: Round robin insert failed! Couldnt find (100, 2, 5) tuple in rrobin_part0 table
Dữ liệu đã tồn tại trong Ratings
Traceback (most recent call last):
  File "D:\Tài liệu Ptit\CSDLPT\BTL-CSDLPT\src\testHelper.py", line 264, in testroundrobininsert
    raise Exception(
        'Round robin insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format(userid, itemid, rating,
                                                    expectedtablename))
Exception: Round robin insert failed! Couldnt find (100, 2, 5) tuple in rrobin_part0 table
roundrobininsert function fail!
```

+ Kết quả kiểm tra trong cơ sở dữ liệu: chỉ có 1 bản ghi được chèn vào

Bảng Ratings:

Query

Query History

1

```
select * from ratings where userid = 100 and movieid = 2 and rating = 5
```

Data Output

Messages

Notifications

SQL

Showing rows: 1 to 1

Pag

	userid integer	movieid integer	rating double precision
1	100	2	5

Bảng rrobin\_part2:

Query

Query History

1

select \* from rrobin\_part2 where userid = 100 and movieid = 2 and rating = 5

Data Output

Messages

Notifications

Showing rows: 1 to 1

Page

	userid integer	movieid integer	rating double precision
1	100	2	5

-TH3:

+ Thực hiện chèn 2 bản ghi khác nhau cùng lúc: kết quả báo thành công

```
[result, e] = testHelper.testroundrobininsert
(MyAssignment, RATINGS_TABLE, 100, 3, 3, conn, '3')
[result, e] = testHelper.testroundrobininsert
(MyAssignment, RATINGS_TABLE, 100, 4, 4, conn, '4')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
```

```
Admin@NguyenDuy MINGW64 /d/Tài liệu Ptit/CSDLPT/BTL-CSDLPT/src (main)
$ py Assignment1Tester.py
Hàm tạo bảng ở testHelper A database named "dds_assgn1" already exists
roundrobininsert function pass!
```

+ Kết quả trong cơ sở dữ liệu:

Bảng rrobin\_part3:

Query Query History

1 select \* from rrobin\_part3 where userid = 100 and movieid = 3 and rating = 3

Data Output Messages Notifications

SQL

Showing rows: 1 to 1

Page

	userid integer	movieid integer	rating double precision
1	100	3	3

Bảng rrobin\_part4:

Query

Query History

1

select \* from rrobin\_part4 where userid = 100 and movieid = 4 and rating = 4

Data Output

Messages

Notifications

Showing rows: 1 to 1

Page

	userid integer	movieid integer	rating double precision
1	100	4	4

## IV. Kết luận

Bài tập lớn đã mô phỏng thành công các phương pháp phân mảnh dữ liệu ngang, cụ thể là phân mảnh theo dải giá trị (range partitioning) và phân mảnh vòng



tròn (round robin partitioning), trên một hệ quản trị cơ sở dữ liệu quan hệ (PostgreSQL). Các hàm Python đã được triển khai để tải dữ liệu, thực hiện phân mảnh và chèn dữ liệu mới vào các phân mảnh tương ứng.

Thông qua quá trình kiểm thử, các chức năng chính của hệ thống đã được xác nhận hoạt động đúng đắn:

- + Hàm LoadRatings đảm bảo việc tải dữ liệu lớn từ tệp ratings.dat vào bảng cơ sở dữ liệu chính xác và hiệu quả.

- + Các hàm phân mảnh (Range\_Partition và RoundRobin\_Partition) đã tạo ra các bảng con (phân mảnh) đúng theo số lượng và quy tắc phân chia được yêu cầu, với sự phân bố dữ liệu hợp lý.

- + Các hàm chèn (Range\_Insert và RoundRobin\_Insert) đã cho phép thêm các bộ dữ liệu mới vào đúng phân mảnh theo logic phân mảnh đã được định nghĩa.

Tuy nhiên, hiệu suất và khả năng mở rộng của hệ thống trên tập dữ liệu cực lớn cần được tối ưu hóa thêm. Các ràng buộc về việc không sử dụng biến toàn cục và không sửa đổi tệp dữ liệu đã được tuân thủ, đồng thời việc sử dụng bảng meta-data đã hỗ trợ quản lý thông tin phân mảnh.

### **Hướng phát triển tiếp theo:**

- + **Tối ưu hóa hiệu suất:** Cải thiện tốc độ thực thi của các hàm, đặc biệt là các hàm phân mảnh và chèn, khi làm việc với khối lượng dữ liệu khổng lồ.

- + **Quản lý lỗi:** Xử lý các trường hợp ngoại lệ và lỗi một cách mạnh mẽ hơn (ví dụ: lỗi kết nối cơ sở dữ liệu, lỗi định dạng dữ liệu).