

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



Báo cáo bài tập lớn
CƠ SỞ DỮ LIỆU PHÂN TÁN
Nhóm 13

Giảng viên:	Kim Ngọc Bách
Sinh Viên:	Nguyễn Đức Phúc - B22DCCN632
	Trương Chiến Nguyên - B22DCCN596
	Hoàng Đình Nhật Văn - B22DCCN888

Hà Nội, 6/2025

MỤC LỤC

PHÂN CÔNG NHIỆM VỤ.....	3
A. Phần làm việc chung.....	3
B. Phần làm cá nhân.....	3
Báo cáo.....	4
A. Phần làm việc chung.....	4
1. Thiết lập môi trường.....	4
2. Hệ quản trị cơ sở dữ liệu.....	4
3. Download dữ liệu.....	4
4. Cài đặt các hàm dùng chung.....	4
5. Cài đặt hàm LoadRatings().....	5
B. Phần làm việc cá nhân.....	9
1. Hàm Range Partition.....	9
2. Hàm RoundRobin Partition.....	12
3. Hàm roundrobininsert().....	15
4. Hàm range_insert().....	18
C. Kết quả kiểm thử chương trình.....	22

PHÂN CÔNG NHIỆM VỤ

A. Phần làm việc chung

- Các yêu cầu từ 1 - 4 từ đề của thầy giao.

B. Phần làm cá nhân

1. Hoàng Đình Nhật Văn
 - Cài đặt hàm Python Range_Partition().
 - Cài đặt hàm Python RoundRobin_Partition().
2. Trương Chiến Nguyên.
 - Cài đặt hàm Python RoundRobin_Partition().
 - Cài đặt hàm Python RoundRobin_Insert().
3. Nguyễn Đức Phúc
 - Cài đặt hàm Python RoundRobin_Insert().
 - Cài đặt hàm Python Range_Insert().

Báo cáo

A. Phần làm việc chung

Trong phạm vi phần báo cáo của phần A và phần B các kết quả được đề cập là kết quả khi thực thi với file *test_data.dat*. Trong phần C kết quả kiểm thử chương trình được thực thi với file *ratings.dat*

1. Thiết lập môi trường.

Để hoàn thành bài tập nhóm về môn học lần này nhóm quyết định sử dụng máy tính cá nhân và ngôn ngữ lập trình Python để giải quyết vấn đề.

- Hệ điều hành: Windows 11.
- Python Version: 3.12.x
- Thư viện: psycopg2

2. Hệ quản trị cơ sở dữ liệu.

- Nhóm quyết định cài đặt và sử dụng **PostgreSQL**.

3. Download dữ liệu.

- Tải tệp rating.dat từ trang MovieLens
(<http://files.grouplens.org/datasets/movielens/ml-10m.zip>)

4. Cài đặt các hàm dùng chung

a. `getopenconnection(user='postgres', password='root', dbname='db_assign1')`:

- Mục đích: Khởi tạo và trả về 1 kết nối với PostgreSQL thông qua hàm `psycopg2.connect()`.

b. `create_db(db_name)`:

- Mục đích: Tạo 1 database theo yêu cầu nếu chưa có sẵn database đó.
- Các bước triển khai:

Bước 1: Kết nối tới database mặc định thông qua `getconnection`. Bên cạnh đó, sử dụng câu lệnh

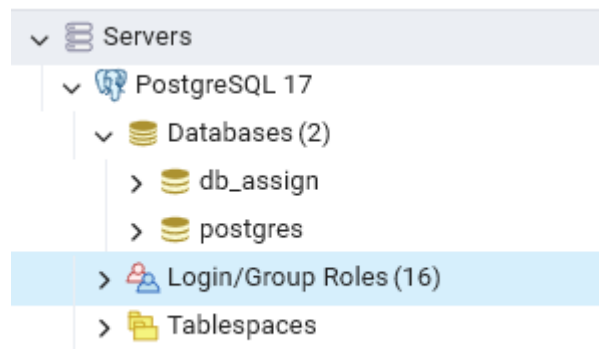
“`con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)`” để tự động commit.

```
# Connect to the default database
con = getopenconnection(dbname='postgres')
con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
cur = con.cursor()
```

Bước 2: Kiểm tra bảng tồn tại hay chưa? Sử dụng cú pháp truy vấn “`SELECT COUNT(*) FROM...`” trong Python để kiểm tra xem đã tồn tại bảng “`db_name`” hay chưa từ bảng hệ thống “`pg_catalog.pg_database`”.

```
# Check if an existing database with the same name exists
cur.execute('SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname=\'%s\' % (dbname,))
count = cur.fetchone()[0]
if count == 0:
    cur.execute('CREATE DATABASE %s' % (dbname,)) # Create the database
else:
    print('A database named {0} already exists'.format(dbname))
```

- Kết quả: Trong trường hợp database *db_assign* chưa được tạo thì sẽ được tạo. Trong đó, database *postgres* là database mặc định khi cài đặt PostgreSQL



5. Cài đặt hàm *LoadRatings()*.

- Mục đích:** Import file *rating.dat* vào 1 bảng trong PostgreSQL có tên *Ratings* với các thuộc tính:

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

- Phân tích tệp *rating.dat*:** Trong tệp có 4 trường dữ liệu (*userid*, *movieid*, *rating*, *timestamp*) tuy nhiên chỉ sử dụng 3 trường dữ liệu đầu tiên:

- Cài đặt hàm:**

- Khởi tạo:
 - Các tham số được truyền vào trong hàm:
 - + *ratingstable*(str): Tên của bảng để tạo và tải dữ liệu vào.
 - + *ratingsfilepath*: Đường dẫn đến nơi lưu trữ file *rating.dat*.
 - + *openconnection*: Đối tượng kết nối PostgreSQL.
 - Tạo database thông qua hàm *create_db(database_name)* trong trường hợp chưa có database.
 - Tạo một con trỏ để giao tiếp với cơ sở dữ liệu thông qua *openconnection.cursor()*.

```
# Ensure database exists before proceeding
create_db('db_assign') # Create database if it doesn't exist

# Get cursor from the connection
cur = openconnection.cursor()
```

- Các bước để import dữ liệu vào cơ sở dữ liệu:

- + Bước 1: Tạo bảng ratings với các thuộc tính yêu cầu (userid, movieid, rating) thông qua lệnh “CREATE TABLE IF NOT EXISTS ...”

```
# Step 1: Create the ratings table with correct schema
# Schema: UserID (int), MovieID (int), Rating (float)
create_table_query = f"""
CREATE TABLE IF NOT EXISTS {ratingtablename} (
    userid INTEGER,
    movieid INTEGER,
    rating REAL
);
"""

cur.execute(create_table_query)
```

- + Bước 2: Để đảm bảo tính đúng đắn của dữ liệu do đó cần tiến hành xóa các dữ liệu đang có sẵn ở trong bảng.

```
# Step 2: Clear existing data if any
cur.execute(f"DELETE FROM {ratingtablename};")
```

- + Bước 3: Đọc, xử lý file, chèn dữ liệu vào database và commit để lưu dữ liệu
 - Tiến hành mở file rating.dat ở chế độ đọc.
 - Sử dụng vòng for để đọc từng dòng dữ liệu:
 - Tiến hành cắt xâu dựa trên ký tự “:” thông qua hàm split() thành 1 chuỗi.
 - Đảm bảo chiều dài hợp lệ của chuỗi (ít nhất 4 phần tử).
 - Ép kiểu dữ liệu phù hợp với các thuộc tính cần insert vào database (userid, movieid, rating).
 - Chèn dữ liệu vào bảng: Sử dụng cú pháp truy vấn “INSERT INTO ..” trong Python để tiến hành chèn dữ

liệu vào bảng. Ngoài ra còn tham số hóa dữ liệu (%s) để tránh SQL injection

```
# Step 3: Read and parse the data file
with open(ratingsfilepath, 'r') as file:
    for line in file:
        # Parse each line: UserID::MovieID::Rating::Timestamp
        parts = line.strip().split('::')
        if len(parts) >= 4: # Ensure we have at least 4 parts
            userid = int(parts[0])
            movieid = int(parts[1])
            rating = float(parts[2])
            # We ignore timestamp (parts[3]) as per schema requirement

            # Insert the record
            insert_query = f"""
            INSERT INTO {ratingtablename} (userid, movieid, rating)
            VALUES (%s, %s, %s);
            """
            cur.execute(insert_query, (userid, movieid, rating))

# Commit the transaction
openconnection.commit()
print(f"Successfully loaded data into {ratingtablename}")
```

- Xử lý ngoại lệ: Trong trường hợp có lỗi khi chèn dữ liệu vào bảng thì sẽ tiến hành rollback lại và hiển thị thông báo lỗi.

```
except Exception as e:
    # Rollback in case of error
    openconnection.rollback()
    print(f"Error loading data: {str(e)}")
    raise e

finally:
    # Close cursor (but not connection as per requirement)
    cur.close()
```

d. Kết quả thực thi:

```
1 SELECT * FROM public.test_data
2
```

Data Output Messages Notifications



	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	4.5
3	1	231	4
4	1	292	3.5
5	1	316	3
6	1	329	2.5
7	1	355	2
8	1	356	1.5
9	1	362	1
10	1	364	0.5
11	1	370	0
12	1	377	3.5
13	1	420	5
14	1	466	4
15	1	480	5
16	1	520	2.5
17	1	539	5
18	1	586	3.5
19	1	588	5
20	1	589	1.5

B. Phần làm việc cá nhân

1. Hàm *Range Partition*

1.1. Mục đích của hàm *Range_Partition*

Hàm `rangepartition()` thực hiện phân mảnh ngang bảng `Ratings` dựa trên giá trị cột `rating` theo phương pháp chia khoảng đều (*range partitioning*).

1.2. Giải thích chi tiết từng phần

Bước 1:

- Bắt đầu bộ đếm thời gian.
- Đặt tiền tố tên bảng phân mảnh là `range_part`.
- Tạo một con trỏ để thao tác với cơ sở dữ liệu.

```
start_time = time.time()
RANGE_TABLE_PREFIX = 'range_part'
cur = openconnection.cursor()
```

Bước 2: Xóa bảng cũ nếu đã tồn tại

- Đảm bảo không có bảng phân mảnh cũ gây xung đột.
- Xóa `range_part0`, `range_part1`, ..., nếu tồn tại.

```
for i in range(numberofpartitions):
    cur.execute(f"DROP TABLE IF EXISTS {RANGE_TABLE_PREFIX}{i};")
```

Bước 3:

- Chia đoạn `rating` `[0.0, 5.0]` thành các khoảng đều nhau.
- Ví dụ: nếu `numberofpartitions = 5`, mỗi khoảng rộng `1.0`.

```
interval = 5.0 / numberOfpartitions
```

Bước 4:

- Tạo `numberofpartitions` bảng mới để lưu các phần dữ liệu.
- Các bảng có cùng cấu trúc: `userid`, `movieid`, `rating`.

```
for i in range(numberofpartitions):
    table_name = f"{RANGE_TABLE_PREFIX}{i}"
    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {table_name} (
            userid INTEGER,
            movieid INTEGER,
            rating REAL
        );
    """)
```

Bước 5: Phân chia dữ liệu

- Với partition đầu tiên ($i=0$):
 - Bao gồm giá trị rating \geq min và rating \leq max.
 - Đảm bảo giá trị thấp nhất (0.0) được đưa vào đúng bảng.

```
if i == 0:
    cur.execute(f"""
        INSERT INTO {table_name}
        SELECT * FROM {ratingtablename}
        WHERE rating >= {i * interval} AND rating <= {(i + 1) * interval};
    """)
```

- Với các partition còn lại ($i > 0$):
 - Bao gồm giá trị rating $>$ min và rating \leq max.
 - Đảm bảo không có giá trị nào bị trùng giữa hai bảng liên tiếp.

```
else:
    cur.execute(f"""
        INSERT INTO {table_name}
        SELECT * FROM {ratingtablename}
        WHERE rating > {i * interval} AND rating <= {(i + 1) * interval};
    """)
```

Bước 6:

- Ghi lại thay đổi vào cơ sở dữ liệu.
- Tính toán thời gian chạy, log ra số lượng phân mảnh và thời gian thực thi.

```
openconnection.commit()
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Successfully created {numberofpartitions} range partitions in {elapsed_time:.4f} seconds.")
```

Bước 7: Kiểm tra ngoại lệ

- Nếu có lỗi trong quá trình tạo bảng hoặc chèn dữ liệu, thực hiện rollback để tránh lỗi nửa vời.
- Cuối cùng, đóng cursor (nhưng không đóng kết nối, vì bài yêu cầu giữ connection mở).

```
except Exception as e:
    openconnection.rollback()
    print(f"Error in rangepartition: {str(e)}")
    raise

finally:
    cur.close()
```

1.2. Kết quả kiểm thử với file testdata.dat

- Kết quả thực thi:

```
=== Testing Range Partition Functions ===  
Successfully created 3 range partitions in 0.5619 seconds.
```

- range_part0 với giá trị rating [0.0, 1.67]:

	userid integer	movieid integer	rating real
1	1	356	1.5
2	1	362	1
3	1	364	0.5
4	1	370	0
5	1	589	1.5

Total rows: 5 Query complete 00:00

- range_part1 với giá trị rating (1.67, 3.33]:

	userid integer	movieid integer	rating real
1	1	316	3
2	1	329	2.5
3	1	355	2
4	1	520	2.5

Total rows: 4 Query complete 00:00

- range_part2 với giá trị rating (3.33, 5]:

	userid integer	movieid integer	rating real
1	1	122	5
2	1	185	4.5
3	1	231	4
4	1	292	3.5
5	1	377	3.5
6	1	420	5
7	1	466	4

Total rows: 11 Query complete 00:00

2. Hàm RoundRobin Partition

2.1. Mục đích của hàm RoundRobin_Partition

Hàm roundrobinpartition() thực hiện phân mảnh ngang bảng Ratings theo phương pháp vòng tròn (Round-Robin Partitioning).

2.2. Mục đích của hàm RoundRobin_Partition

Bước 1:

- Bắt đầu bộ đếm thời gian
- Đặt tiền tố tên bảng là rrobin_part, ví dụ: rrobin_part0, rrobin_part1,...
- Dùng con trỏ cur để thao tác với PostgreSQL.

```
start_time = time.time()
cur = openconnection.cursor()
RROBIN_TABLE_PREFIX = 'rrobin_part'
```

Bước 2: Tạo các bảng phân mảnh

- Tạo numberofpartitions bảng với cùng schema như bảng chính.
- Nếu bảng đã có, xóa toàn bộ dữ liệu cũ bằng DELETE.

```
# Step 1: Create partition tables
for i in range(numberofpartitions):
    table_name = RROBIN_TABLE_PREFIX + str(i)
    create_table_query = f"""
CREATE TABLE IF NOT EXISTS {table_name} (
    userid INTEGER,
    movieid INTEGER,
    rating REAL
);
"""
    cur.execute(create_table_query)

# Clear existing data if any
cur.execute(f"DELETE FROM {table_name};")
```

Bước 3: Chia dữ liệu theo vòng tròn

- Sử dụng ROW_NUMBER() để đánh số thứ tự cho từng dòng dữ liệu từ bảng Ratings.
- Dùng phép chia dư (modulo) để phân phối từng dòng vào các bảng theo vòng tròn:
 - Dòng đầu tiên → bảng 0
 - Dòng thứ 2 → bảng 1
 - Dòng thứ N → bảng (N-1) % numberofpartitions

```

# Step 2: Distribute data using round robin approach
# Use ROW_NUMBER() to assign sequential numbers to rows
# Then use modulo operation to distribute to partitions
for i in range(numberofpartitions):
    table_name = RROBIN_TABLE_PREFIX + str(i)

    insert_query = f"""
    INSERT INTO {table_name} (userid, movieid, rating)
    SELECT userid, movieid, rating
    FROM (
        SELECT userid, movieid, rating,
               ROW_NUMBER() OVER() as row_num
        FROM {ratingtablename}
    ) as numbered_rows
    WHERE (row_num - 1) % {numberofpartitions} = {i};
    """

    cur.execute(insert_query)

```

Bước 4: Kết thúc và thông báo thời gian thực thi

- Ghi lại thay đổi vào cơ sở dữ liệu.
- Tính toán thời gian chạy, log ra số lượng phân mảnh và thời gian thực thi.

```

# Commit the transaction
openconnection.commit()
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Successfully created {numberofpartitions} round robin partitions in {elapsed_time:.4f} seconds.")

```

Bước 5: Kiểm tra ngoại lệ

- Nếu có lỗi trong quá trình tạo bảng hoặc chèn dữ liệu, thực hiện rollback để tránh lỗi nửa vời.
- Cuối cùng, đóng cursor (nhưng không đóng kết nối, vì bài yêu cầu giữ connection mở).

```

except Exception as e:
    # Rollback in case of error
    openconnection.rollback()
    print(f"Error creating round robin partitions: {str(e)}")
    raise e

finally:
    # Close cursor (but not connection as per requirement)
    cur.close()

```

2.3. Kết quả kiểm thử với file testdata.dat

- Kết quả thực thi:

```
=== Testing Round Robin Functions ===  
Successfully created 3 round robin partitions in 1.5585 seconds.
```

- rrobin_part0 với giá trị $(\text{row_num}-1)\%3 == 0$:

	userid integer	movieid integer	rating real
1	1	122	5
2	1	292	3.5
3	1	355	2
4	1	364	0.5
5	1	420	5
6	1	520	2.5
7	1	588	5
Total rows: 7		Query complete 00:00:00.000	

- rrobin_part1 với giá trị $(\text{row_num}-1)\%3 == 1$:

	userid integer	movieid integer	rating real
1	1	185	4.5
2	1	316	3
3	1	356	1.5
4	1	370	0
5	1	466	4
6	1	539	5
7	1	588	1.5
Total rows: 7		Query complete 00:00:00.000	

- rrobin_part2 với giá trị $(\text{row_num}-1)\%3 == 2$:

	userid integer	movieid integer	rating real
1	1	231	4
2	1	329	2.5
3	1	362	1
4	1	377	3.5
5	1	480	5
6	1	586	3.5
Total rows: 6		Query complete 00:00:00.000	

3. Hàm `roundrobininsert()`

3.1. Mục đích của hàm `roundrobininsert()`

Hàm `roundrobininsert()` thực hiện chèn dữ liệu vào bảng chính **ratings** và các phân mảnh phù hợp được chia dựa theo phương pháp vòng tròn Round Robin trước đó.

3.2. Các bước thực hiện

Bước 1: Khởi tạo

```
def roundrobininsert(ratingtablename, userid, itemid, rating, openconnection):
    start_time = time.time()

    cur = openconnection.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'
```

- Tạo con trỏ giao tiếp với cơ sở dữ liệu thông qua `cursor()`.
- Đặt tiền tố cho các bảng theo yêu cầu của đề.
- Đặt biến tính thời gian thực thi hàm.

Bước 2: Thêm vào bảng chính

```
# Step 1: Insert into main table
insert_main_query = f"""
INSERT INTO {ratingtablename} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cur.execute(insert_main_query, (userid, itemid, rating))
```

- Tiến hành insert dữ liệu vào bảng bằng việc chèn các giá trị: `userId`, `movieId`, `rating`.

Bước 3: Tính toán tổng số bản ghi

```
# Step 2: Get total number of rows in main table after insertion
cur.execute(f"SELECT COUNT(*) FROM {ratingtablename};")
total_rows = cur.fetchone()[0]
```

- Đếm tổng số bản ghi trong bảng chính sau khi chèn.
- Thông tin này được sử dụng để xác định partition đích.

Bước 4: Xác định số lượng partition

```
# Step 3: Count number of existing partitions
numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
```

- Gọi hàm `count_partition()` để đếm số phân mảnh hiện có với prefix `rrobin_part` để xác định.

Bước 5: Tính toán partition đích

```
# Step 4: Calculate which partition this new row should go to
# Since we use 0-based indexing and round robin distribution
partition_index = (total_rows - 1) % numberofpartitions
partition_table_name = RROBIN_TABLE_PREFIX + str(partition_index)
```

- Áp dụng công thức round robin để tính toán phân mảnh mà hàng dữ liệu mới sẽ được thêm vào.

Bước 6: Chèn dữ liệu vào partition tương ứng

```
# Step 5: Insert into the appropriate partition table
insert_partition_query = f"""
INSERT INTO {partition_table_name} (userid, movieid, rating)
VALUES (%s, %s, %s);
"""
cur.execute(insert_partition_query, (userid, itemid, rating))

# Commit the transaction
openconnection.commit()
print(f"Successfully inserted record into {ratingtablename} and {partition_table_name}")
```

- Xác định tên partition table = `rrobin_part` + `partition_index`
- Thực hiện câu lệnh vào partition table tương ứng.
- Sử dụng cùng dữ liệu đã chèn vào bảng chính.

Bước 7: Hoàn tất transaction

```
# Commit the transaction
openconnection.commit()
print(f"Successfully inserted record into {ratingtablename} and {partition_table_name}")
```

- Commit transactions nếu câu lệnh thành công, hiển thị thời gian thực hiện

Xử lý ngoại lệ: Trong trường hợp có ngoại lệ xảy ra thì log lỗi đồng thời rollback lại kết quả và cuối cùng thì đóng cursor nhưng không đóng kết nối.


```

except Exception as e:
    # Rollback in case of error
    openconnection.rollback()
    print(f"Error inserting record: {str(e)}")
    raise e

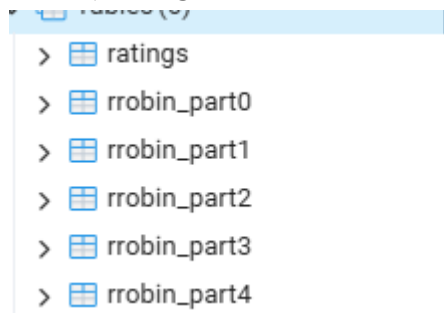
finally:
    # Close cursor (but not connection as per requirement)
    cur.close()

end_time = time.time()
total_time = end_time - start_time
print(f"roundrobininsert completed in {total_time:.4f} seconds")

```

3.3. Kết quả thực hiện

- Chia thành 5 phân mảnh với rangepartition, thực hiện insert với hàm roundrobininsert('ratings', 9999, 1001, 4.5, conn)



- Giá trị được thêm vào bảng chính nằm ở dòng thứ 22 trong bảng ratings

17	1	539	5
18	1	586	3.5
19	1	588	5
20	1	589	1.5
21	100	1	3
22	9999	1001	4.5

- Phân mảnh tương ứng với dữ liệu mới được thêm vào là :

$$(row_count - 1) \% 5 = (22-1)\%5 = 1$$
Tương ứng phân mảng rrobin_part1 :

Query

Query History

1

2


SELECT * FROM public.rrobin_part1

Data Output


Messages

Notifications


≡





▼




▼









	userid integer	movieid integer	rating real
1	1	185	4.5
2	1	355	2
3	1	377	3.5
4	1	539	5
5	9999	1001	4.5

4. Hàm `range_insert()`

4.1. Mục đích của hàm `range_insert()`

Hàm `range_insert()` thực hiện chèn dữ liệu vào bảng chính **ratings** đồng thời dựa trên giá trị *rating* để chèn vào phân mảnh ngang tương ứng.

Ví dụ: Có 3 phân mảnh ngang thì các khoảng giá trị lần lượt là $[0, 1.67]$, $(1.67, 3.34]$, $(3.34, 5]$. Với 1 bộ mới có giá trị *rating* = 3 thì sẽ được chèn vào phân mảnh số 2.

4.2. Các bước triển khai.

Khởi tạo:

- Tạo 1 con trỏ giao tiếp với PostgreSQL thông qua `cursor()`.
- Đặt tên cho tiền tố bảng theo yêu cầu của đề.
- Xác định tính hợp lệ của giá trị *rating*.

```
cur = openconnection.cursor()
RANGE_TABLE_PREFIX = "range_part"

if rating > 5 or rating < 0:
    return f"Rating value is invalid"
```

Bước 1:

- Tiến hành insert dữ liệu vào bảng chính.

```
# Step1: Insert main table
insert_table_query = f"""
    INSERT INTO {ratingtablename} (userid, movieid, rating)
    VALUES (%s, %s, %s);
    """
cur.execute(insert_table_query, (userid, itemid, rating))
```

Bước 2:

- Đếm số lượng phân mảnh thông qua hàm *count_partitions()*.

```
#Step 2: Count numbers of partitions
numbers = count_partitions(RANGE_TABLE_PREFIX, openconnection)
```

Bước 3:

- Tính khoảng giá trị *delta* của từng phân mảnh
Vd: Có 5 phân mảnh thì sẽ có 5 khoảng [0, 1], (1, 2], (2, 3], (3, 4], (4, 5].

Ngoài ra, trong công thức sử dụng 5 chia cho số khoảng. Lý do chọn số 5 bởi vì giá trị *rating* tối đa là 5

```
#Step 3: Calculate range value of each partition
delta = 5 / numbers # max rating = 5
```

Bước 4: Xác định phân mảnh cần chèn dữ liệu.

- Giá trị *index* sẽ là phân mảnh thứ mấy cần chèn vào khi ép kiểu *int* với phép chia *rating/delta*. Tuy nhiên, khi phép chia này chia hết và *index != 0* thì ta cần trừ giá trị *index* đi 1 đơn vị để về đúng phân mảnh cần chèn

```
#Step 4: Define partition that will insert
index = int(rating/delta)
if rating % delta == 0 and index != 0:
    index -=1
```

Bước 5: Chèn dữ liệu vào đúng phân mảnh.

- Đặt biến cho tên bảng để dễ dàng tái sử dụng
- Sử dụng câu lệnh “INSERT INTO ...” thông thường để chèn dữ liệu vào bảng phân mảnh
- Thực hiện commit câu lệnh và in thông báo.

```
# Step 5: Insert data into partition
table_name = RANGE_TABLE_PREFIX + str(index)
insert_query = (f"""
    INSERT INTO {table_name} (userid, movieid, rating)
    VALUES (%s, %s, %s)
    """)
cur.execute(insert_query, (userid, itemid, rating))

openconnection.commit()
print(f"Successfully inserted record into {ratingtablename} and {table_name}")
```

Xử lý ngoại lệ: Trong trường hợp có ngoại lệ xảy ra thì log lỗi đồng thời rollback lại kết quả và cuối cùng thì đóng cursor nhưng không đóng kết nối.

```
except Exception as e:
    # Rollback in case of error
    openconnection.rollback()
    print(f"Error inserting record: {str(e)}")
    raise e

finally:
    # Close cursor (but not connection as per requirement)
    cur.close()
```

4.3. Kết quả

Kết quả phân mảnh sẽ dựa vào kết quả phần 1.2.3 phần làm cá nhân ở trên. Gọi hàm rangeinsert('ratings', 145, 8, 4, conn)

Giá trị được chèn vào bảng chính:

17	1	585	5
18	1	586	3.5
19	1	588	5
20	1	589	1.5
21	145	8	4
ratings	rows: 21	Query complete 00:00:00.217	

$rating = 4 \Rightarrow$ Bộ mới này sẽ nằm ở phân mảnh số 3

9	1	539	5
10	1	586	3.5
11	1	588	5
12	145	8	4

C. Kết quả kiểm thử chương trình

Kiểm thử với file rating.dat.

1. Log chương trình chạy.

- Kết quả log với hàm `rangepartition()` và `rangeinsert()`.

```
D:\python\python.exe G:\CSDLPT\btl-csdlpt1\Assignment1Tester.py
A database named "db_assign1" already exists
A database named db_assign1 already exists
create_db completed in 0.1420 seconds
Data loading successful: 10000054 records imported to ratings
Operation completed in 21.4370 seconds
loadratings function pass!
Successfully created 5 range partitions
rangepartition completed in 34.2107 seconds
rangepartition function pass!
Successfully inserted record into ratings and range_part2
rangeinsert completed in 0.0111 seconds
rangeinsert function pass!
Press enter to continue? If you enter, result will be deleted
```

- Kết quả log với hàm `roundrobinpartition()` và `roundrobininsert()`.

```
Press enter to continue? If you enter, result will be deleted
A database named db_assign1 already exists
create_db completed in 0.1038 seconds
Data loading successful: 10000054 records imported to ratings
Operation completed in 21.6307 seconds
Successfully created 5 round robin partitions
roundrobinpartition completed in 44.1336 seconds
roundrobinpartition function pass!
Successfully inserted record into ratings and rrobin_part4
roundrobininsert completed in 0.2980 seconds
roundrobininsert function pass!
Press enter to Delete all tables?
```

2. Cấu trúc thư mục các bảng trong pgAdmin4.

- Cấu trúc thư mục sau khi thực hiện xong hàm `rangepartition()` và `rangeinsert()`.

Tables (6)
> range_part0
> range_part1
> range_part2
> range_part3
> range_part4
> ratings

- Cấu trúc thư mục sau khi thực hiện xong hàm `roundrobinpartition()` và `roundrobininsert()`.

Tables (6)
> ratings
> rrobin_part0
> rrobin_part1
> rrobin_part2
> rrobin_part3
> rrobin_part4

3. Kiểm thử hàm `loadratings()`.

- Kết quả thực thi

```
Data loading successful: 10000054 records imported to ratings
Operation completed in 22.0460 seconds
loadratings function pass!
```

- Kết quả ở trong pgAdmin4:

	userid integer	movieid integer	rating real
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
Total rows: 10000054		Query complete 00:00:04.115	

4. Kết quả kiểm thử hàm rangepartition().

- Kết quả thực thi:

```
=== Testing Range Partition Functions ===  
Successfully created 3 range partitions in 18.2195 seconds.
```

- range_part0 với giá trị rating [0.0, 1.67]:

	userid integer	movieid integer	rating real
1	4	231	1
2	5	1	1
3	5	708	1
4	5	736	1
5	5	780	1
6	5	1391	1
7	6	2086	1
Total rows: 597446		Query complete 00:00:00.559	

- range_part1 với giá trị rating (1.67, 3.33]:

	userid integer	movieid integer	rating real
1	2	151	3
2	2	376	3
3	2	539	3
4	2	648	2
5	2	719	3
6	2	733	3
7	2	736	3
Total rows: 3517160		Query complete 00:00:01.450	

- range_part2 với giá trị rating (3.33, 5]:

	userid integer	movieid integer	rating real
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
Total rows: 5885448		Query complete 00:00:04.445	

5. Kết quả kiểm thử hàm roundrobinpartition()

Sau khi gọi hàm `rangepartition('ratings', 3, conn)`, sẽ có kết quả thời gian thực thi ở terminal và các phân mảnh sẽ được tạo:

- Kết quả thực thi:

```
=== Testing Round Robin Functions ===  
Successfully created 3 round robin partitions in 40.5826 seconds.
```

- `rrobin_part0` với giá trị $(\text{row_num}-1)\%3 == 0$:

	userid integer	movieid integer	rating real
1	1	122	5
2	1	292	5
3	1	355	5
4	1	364	5
5	1	420	5
6	1	520	5
7	1	588	5
Total rows: 3333352		Query complete 00:00:05.467	

- `rrobin_part1` với giá trị $(\text{row_num}-1)\%3 == 1$:

	userid integer	movieid integer	rating real
1	1	185	5
2	1	316	5
3	1	356	5
4	1	370	5
5	1	466	5
6	1	539	5
7	1	580	5
Total rows: 3333351		Query complete 00:00:04.162	

- `rrobin_part2` với giá trị $(\text{row_num}-1)\%3 == 2$:

	userid integer	movieid integer	rating real
1	1	231	5
2	1	329	5
3	1	362	5
4	1	377	5
5	1	480	5
6	1	586	5
7	1	584	5
Total rows: 3333351		Query complete 00:00:03.001	

6. Kết quả kiểm thử hàm roundrobininsert().

- Dữ liệu đầu vào:

userid: 100

itemid: 1

rating: 3

- Kết quả thực thi: Dữ liệu được chèn vào bảng ratings và rrobin_part1.

Successfully inserted record into ratings and rrobin_part1

roundrobininsert completed in 0.2770 seconds

roundrobininsert function pass!

- Kết quả ở pgAdmin4: Bộ dữ liệu mới xuất hiện ở trong bảng ratings.



26	100	4501	3
27	100	4508	3
28	100	7171	3
29	100	31770	3
30	100	1	3

7. Kết quả kiểm thử hàm rangeinsert().

- Dữ liệu đầu vào:

userid: 1

itemid: 2

rating: 3

- Kết quả thực thi: Bộ mới được chèn vào bảng ratings và range_part1.

Successfully inserted record into ratings and range_part1

rangeinsert completed in 0.0090 seconds

rangeinsert function pass!

- Kết quả ở trong pgAdmin4:

- Bộ mới được chèn vào dữ liệu.

```
1 SELECT * FROM public.range_part1
2 WHERE userid = 1 and rating = 3
3
```

Data Output Messages Notifications

	userid integer	movieid integer	rating real
1	1	2	3