HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

_____o0o_____

BÁO CÁO BÀI TẬP LỚN

Môn học: Cơ sở dữ liệu phân tán

Nhóm lớp: 09

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

Họ và tên	**Mã sinh viên**
Trần Cảnh Hưng	B22DCCN421
Đặng Hữu Nghĩa	B22DCCN601
Vũ Ngọc Hùng	B22DCCN373

Hà Nội, Tháng 6/2025

MỤC LỤC

ĐỀ BÀI

Nhiệm vụ yêu cầu là mô phỏng các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở (ví dụ: PostgreSQL hoặc MySQL). Mỗi nhóm sinh viên phải tạo một tập các hàm Python để tải dữ liệu đầu vào vào một bảng quan hệ, phân mảnh bảng này bằng các phương pháp phân mảnh ngang khác nhau, và chèn các bộ dữ liệu mới vào đúng phân mảnh.

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.

Nhiệm vụ yêu cầu

1. Cấu hình máy ảo: Python 3.12.x, OS: Ubuntu hoặc Windows 10.
2. Cài đặt PostgreSQL hoặc MySQL.
3. Tải tệp rating.dat từ trang MovieLens:
http://files.grouplens.org/datasets/movielens/ml-10m.zip
4. 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)
5. Cài đặt hàm Python Range_Partition() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL) và (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.
6. Cài đặt hàm Python RoundRobin_Partition() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL) và (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 (hoặc MySQL), sử dụng phương pháp phân mảnh kiểu **vòng tròn (round robin).**

7. 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.

8. Cài đặt hàm Python Range_Insert() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL), (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

I. YÊU CẦU CÀI ĐẶT

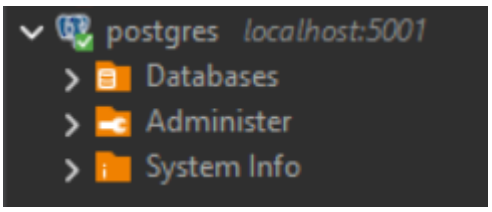
1. Cấu hình máy

- Nhóm đã sử dụng máy hệ điều hành Windows 10 và Python bản 3.12.6

```
PS D:\KÌ 2 NĂM 3\CSDLPT\major_assignment_1> python --version
Python 3.12.6
```

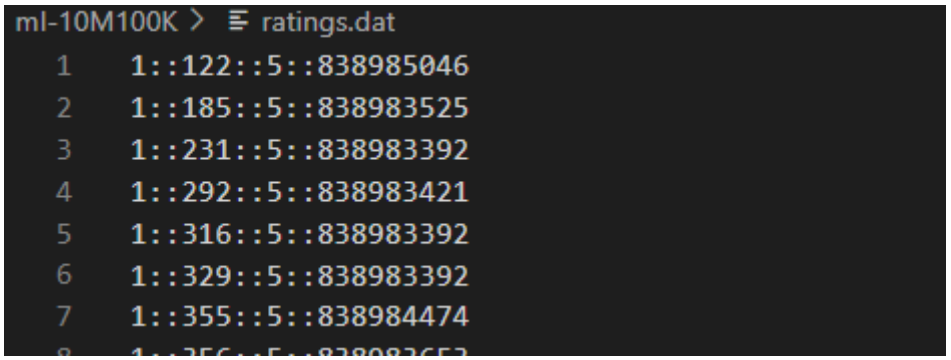
2. Cơ sở dữ liệu

- Nhóm đã cài đặt hệ quản trị cơ sở dữ liệu PostgreSQL.



3. Dữ liệu

- Tải tệp rating.dat từ trang MovieLens:



- 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. Timestamp là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970

II. PHÂN TÍCH BÀI TOÁN

1. Tổng quan

- Bài toán yêu cầu phân mảnh ngang (horizontal partitioning) một bảng dữ liệu chứa thông tin đánh giá phim
- Dữ liệu đầu vào là file ratings.dat chứa các cột: userid, movieid, rating
- Cần thực hiện 2 phương pháp phân mảnh: Range Partitioning và Round Robin Partitioning

2. Phương pháp phân mảnh cần thực hiện

a. Range Partitioning:

- Dữ liệu được phân chia dựa trên giá trị của cột rating
- Khoảng giá trị (delta) = 5/numberofpartitions (với numberofpartitions là số phân mảnh mong muốn)
- Mỗi phân mảnh sẽ chứa dữ liệu trong một khoảng rating cụ thể
- Ví dụ với 5 phân mảnh:
    + Delta = 5/5 = 1
    + Phân mảnh 0: rating từ 0 đến 1 ($0 \leq rating \leq 1$)
    + Phân mảnh 1: rating từ 1 đến 2 ($1 < rating \leq 2$)
    + Phân mảnh 2: rating từ 2 đến 3 ($2 < rating \leq 3$)
    + Phân mảnh 3: rating từ 3 đến 4 ($3 < rating \leq 4$)
    + Phân mảnh 4: rating từ 4 đến 5 ($4 < rating \leq 5$)

b. Round Robin Partitioning:

- Dữ liệu được phân phối luân phiên vào các phân mảnh
- Không phụ thuộc vào giá trị của dữ liệu
- Sử dụng ROW_NUMBER() để đánh số thứ tự các dòng
- Ví dụ cụ thể với 5 phân mảnh:
    + Dòng 1 (ROW_NUMBER = 1) -> Phân mảnh 0 (1 % 5 = 1)
    + Dòng 2 (ROW_NUMBER = 2) -> Phân mảnh 1 (2 % 5 = 2)
    + Dòng 3 (ROW_NUMBER = 3) -> Phân mảnh 2 (3 % 5 = 3)
    + Dòng 4 (ROW_NUMBER = 4) -> Phân mảnh 3 (4 % 5 = 4)
    + Dòng 5 (ROW_NUMBER = 5) -> Phân mảnh 4 (5 % 5 = 0)
    + Dòng 6 (ROW_NUMBER = 6) -> Phân mảnh 0 (6 % 5 = 1)
    + Và cứ tiếp tục…

3. Yêu cầu của các phân mảnh

- Tính đầy đủ (***Completeness***): Tất cả dữ liệu phải được phân mảnh
- Tính rời rạc (***Disjointness***): Không có dữ liệu trùng lặp giữa các phân mảnh
- Tính tái tạo (**Reconstruction**): Có thể tái tạo lại bảng gốc từ các phân mảnh

III. TRIỂN KHAI BÀI TOÁN (Interface.py)

1. Thiết lập và kết nối cơ sở dữ liệu

a. Hàm createdb(...):

```
def createdb(dbname):

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

    # 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))

    # Clean up
    cur.close()
    con.close()
```

- Ta định nghĩa hàm **createdb(dbname)** nhận vào tên của cơ sở dữ liệu ta muốn tạo.
- Gọi hàm **getopenconnection()** để kết nối đến cơ sở dữ liệu PostgreSQL hiện tại.
- Vì PostgreSQL không cho phép lệnh CREATE DATABASE chạy trong một transaction, nên cần tắt chế độ transaction bằng cách đặt isolation level thành AUTOCOMMIT.
- Sau đó tạo 1 con trỏ cur = con.cursor() , gọi cur.execute() để thực hiện lệnh SQL kiểm tra xem có cơ sở dữ liệu nào có tên là giống dbname hay chưa.

```
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))
```

- Tạo biến count = cur.fetchone()[0] để lấy kết quả trả về từ truy vấn. Lệnh fetchone() lấy một dòng kết quả từ truy vấn, và vì kết quả trả về chỉ có 1 phần tử là số lượng cơ sở dữ liệu đếm được, nên chỉ cần lấy phần từ [0].
- Kiểm tra xem nếu count = 0 thì chưa có cơ sở dữ liệu nào, thực hiện truy vấn tạo mới cơ sở dữ liệu. Còn nếu đã có thì print ra thông báo.

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

b. Hàm getopenconnection()

```python
def getopenconnection(user='postgres', password='newpassword', port=5001, host='localhost', dbname='postgres'):
    conn_params = {
        'dbname': dbname,
        'user': user,
        'password': password,
        'host': host,
        'port': port
    }
    print(f"Connecting to the database {conn_params}")
    conn = psycopg2.connect(**conn_params)
    print(f"Connected to the database {conn_params}")
    return conn
```

- Ta định nghĩa hàm **getopenconnection(...)** nhận vào các tham số với giá trị mặc định:
    + user='postgres': tên người dùng
    + password='newpassword': mật khẩu của user
    + port=5001: cổng kết nối đến PostgreSQL
    + host='localhost': địa chỉ máy chủ PostgreSQL
    + dbname='postgres': tên cơ sở dữ liệu
- Tạo một từ điển conn_params chứa tất cả thông tin cần thiết để kết nối đến PostgreSQL.
- Sau đó mở kết nối đến cơ sở dữ liệu qua lệnh conn = psycopg2.connect(**conn_params).
- In ra thông báo kết nối thành công và trả về kết nối: return con.

2. Xây dựng hàm LoadRatings() để tải nội dung vào bảng trong database

```python
def loadratings(ratingstablename, ratingsfilepath, openconnection):

    con = openconnection
    cur = con.cursor()
    print(f"Loading data from {ratingsfilepath} to {ratingstablename}")

    cur.execute("create table " + ratingstablename + "(userid integer, extra1 char, movieid integer, extra2 char, rating float, extra3 char, timesta
    cur.copy_from(open(ratingsfilepath),ratingstablename,sep=':')

    cur.execute("alter table " + ratingstablename + " drop column extra1, drop column extra2, drop column extra3, drop column timestamp;")
    cur.execute("SELECT * FROM " + ratingstablename + " LIMIT 5")
    rows = cur.fetchall()
    for row in rows:
        print(f"Row: {row}")
    cur.close()
    con.commit()
```

- Ta định nghĩa hàm **loadratings()** nhận vào các tham số sau:
    + ratingstablename: tên bảng trong PostgreSQL để lưu dữ liệu.
    + ratingsfilepath: đường dẫn đến file dữ liệu ratings.dat

+ openconnection: kết nối PostgreSQL (đã mở).

- Tạo biến con lấy kết nối từ openconnection, con trỏ cur = con.cursor() để thực hiện các truy vấn SQL.
- Thực hiện truy vấn để tạo bảng chứa dữ liệu từ file ratings.dat.

```
cur.execute("create table " + ratingstablename + "(userid integer, extra1 char, movieid integer, extra2 char, rating float, extra3 char, timestamp bigint);")
cur.copy_from(open(ratingsfilepath),ratingstablename,sep=':')
```

Phải thực hiện truy vấn như vậy thay vì truy vấn tạo trực tiếp các cột yêu cầu bởi vì:

+ Dữ liệu trong file có 4 trường chính là userid, movieid, rating, timestamp, được phân cách bởi dấu "::"

userid::movieid::rating::timestamp

+ Tuy nhiên psycopg2.copy_from() không hỗ trợ trực tiếp :: là separator.

  ⇒ Giải pháp: coi :: là nhiều dấu : rồi tạo bảng có thêm các cột "extra" để bỏ qua phần separator phụ.
+ Dòng trên sẽ được chia thành 7 cột: userid, extra1, movieid, extra2, rating, extra3, timestamp khi dùng sep=':' với copy_from.

- Sử dụng copy_from() để tải dữ liệu hàng loạt từ file vào bảng. Dấu phân cách là ':', do đó mỗi dòng bị tách thành 7 phần.
- Sau đó sử dụng truy vấn có ALTER TABLE để loại các cột extra1,2,3 và timestamp, chỉ giữ lại 3 cột chính theo yêu cầu đề bài.

```
cur.execute("alter table " + ratingstablename + " drop column extra1, drop column extra2, drop column extra3, drop column timestamp;")
```

- Cuối cùng, dùng truy vấn lấy ra 5 hàng đầu tiên từ bảng sau khi nạp dữ liệu, sử dụng fetchall() để lấy dữ liệu và in ra từng dòng để kiểm tra kết quả.

```
cur.execute("SELECT * FROM " + ratingstablename + " LIMIT 5")
rows = cur.fetchall()
for row in rows:
    print(f"Row: {row}")
cur.close()
con.commit()
```

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

3. Xây dựng hàm Range_Partitions() để tạo phân mảnh ngang dựa trên Rating

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):

    con = openconnection
    cur = con.cursor()
    delta = 5 / numberofpartitions
    print(f"Delta: {delta}")
    RANGE_TABLE_PREFIX = 'range_part'
    for i in range(0, numberofpartitions):
        minRange = i * delta
        maxRange = minRange + delta
        table_name = RANGE_TABLE_PREFIX + str(i)

        cur.execute("create table " + table_name + " (userid integer, movieid integer, rating float);")
        if i == 0:
            cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablenam
        else:
            cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablenam


        cur.execute(f"SELECT COUNT(*) FROM {table_name};")
        count = cur.fetchone()[0]
        print(f"[DEBUG] Inserted {count} rows into {table_name}")
    cur.close()
    con.commit()
```

- Ta định nghĩa hàm **rangepartition(...)** nhận vào các tham số sau:
    + ratingstablename: tên bảng gốc chứa ratings.
    + numberofpartitions: số lượng phân mảnh muốn tạo.
    + openconnection: kết nối đến CSDL PostgreSQL (đã mở).
- Tạo biến con lấy kết nối từ openconnection, con trỏ cur = con.cursor() để thực hiện các truy vấn SQL.
- Vì ratings chấm theo thang từ 0 đến 5 và chia đều thành n đoạn nên ta sẽ xử lý thông qua biến delta:

$$delta = 5 / numberofpartitions$$

Sau đó lặp để phân mảnh: Với mỗi i, tính khoảng giá trị ratings thuột [minRange, maxRange] mà bảng con thứ i sẽ lưu. Cùng với đó gắn tên bảng phân mảnh thứ i tương ứng

```
delta = 5 / numberofpartitions
print(f"Delta: {delta}")
RANGE_TABLE_PREFIX = 'range_part'
for i in range(0, numberofpartitions):
    minRange = i * delta
    maxRange = minRange + delta
    table_name = RANGE_TABLE_PREFIX + str(i)
```

- Gọi truy vấn để tạo bảng phân mảnh, có 3 cột giống bảng Ratings: userid, movieid, rating.
- Sau đó, xử lý truy vấn để chèn dữ liệu vào.

```
cur.execute("create table " + table_name + " (userid integer, movieid integer, rating float);")
if i == 0:
    cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablename +
else:
    cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from " + ratingstablename +
```

```
" + ratingstablename + " where rating >= " + str(minRange) + " and rating <= " + str(maxRange) + ";")

" + ratingstablename + " where rating > " + str(minRange) + " and rating <= " + str(maxRange) + ";")
```

- Tuy nhiên cần phải xử lý bảng đầu tiên, với i = 0 thì ta dùng “>= minRange” vì cần phải bao gồm giá trị 0.0, còn các bảng còn lại dùng “>” để tránh chồng chéo dữ liệu tại biên do cận trên đã lấy “<= maxRange”.
- Để kiểm tra, ta sẽ thử in ra số dòng dữ liệu được chèn vào mỗi phân mảnh:

```
cur.execute(f"SELECT COUNT(*) FROM {table_name};")
count = cur.fetchone()[0]
print(f"[DEBUG] Inserted {count} rows into {table_name}")
```

Gọi truy vấn để đếm xem có bao nhiêu dòng trong phân mảnh vừa tạo, kết quả trả về 1 giá trị duy nhất là số bản ghi có trong cơ sở dữ liệu. Sau đó dùng fetchone()[0] để lấy giá trị đó lưu vào biến count. In ra để biết thực sự đã có bao nhiêu dòng được thêm vào bảng phân mảnh.

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

4. Xây dựng hàm RoundRobin_Partitions() để tạo phân mảnh ngang dựa trên phương pháp phân mảnh vòng tròn

```
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
    """
    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'
    for i in range(0, numberofpartitions):
        table_name = RROBIN_TABLE_PREFIX + str(i)
        cur.execute("create table " + table_name + " (userid integer, movieid integer, rating float);")
        cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from (select userid, movieid, r
    cur.close()
    con.commit()
```

- Ta định nghĩa hàm **roundrobinpartition(...)** nhận vào các tham số sau:
    + ratingstablename: tên bảng gốc chứa ratings.
    + numberofpartitions: số lượng phân mảnh muốn tạo.
    + openconnection: kết nối đến CSDL PostgreSQL (đã mở)
- Tạo biến con lấy kết nối từ openconnection, con trỏ cur = con.cursor() để thực hiện các truy vấn SQL.
- Ta thực hiện tương tự để tạo các bảng phân mảnh, gọi biến i chạy vòng lặp chạy từ 0 đến numberofpartitions - 1. Sau đó gọi truy vấn để tạo các phân mảnh i tương ứng. Mỗi bảng con sẽ có 3 cột giống với bảng Ratings: userid, movieid, rating.

```
RROBIN_TABLE_PREFIX = 'rrobin_part'
for i in range(0, numberofpartitions):
    table_name = RROBIN_TABLE_PREFIX + str(i)
    cur.execute("create table " + table_name + " (userid integer, movieid integer, rating float);")
```

- Sau khi tạo xong, ta thực hiện truy vấn tiếp theo để phân dữ liệu vào các bảng:

```
    cur.execute("insert into " + table_name + " (userid, movieid, rating) select userid, movieid, rating from (select userid, movieid, rating, R
cur.close()
```

```
ating, ROW_NUMBER() over() as rnum from " + ratingstablename + ") as temp where mod(temp.rnum-1, "+ str(numberofpartitions) + ") = " + str(i) + ";")
```

Đầu tiên truy vấn xác định phân mảnh muốn thêm dữ liệu vào, sau đó lấy ra các thuộc tính userid, movieid, rating, cùng với đó là sử dụng hàm ROW_NUMBER() OVER() để lấy số dòng (rnum) trong bảng tạm thời temp. Sau đó, lấy số dòng (rnum - 1 do tính từ 0) từ bảng temp chia dư cho số phân mảnh mong muốn (numberofpartitions). Ví dụ mod(rnum-1, 5) = x thì ta sẽ gắn dữ liệu vào phân mảnh x tương ứng. Cứ như vậy cho tới khi tất cả dữ liệu được chia vào các phân mảnh.

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

5. Xây dựng hàm Range_Insert() để chèn một bộ dữ liệu mới vào bảng Ratings và vào đúng phân mảnh dựa trên giá trị Rating

```
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):

    con = openconnection
    cur = con.cursor()
    RANGE_TABLE_PREFIX = 'range_part'
    cur.execute("insert into " + ratingstablename + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
    numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
    delta = 5 / numberofpartitions
    index = int(rating / delta)
    if rating % delta == 0 and index != 0:
        index = index - 1
    table_name = RANGE_TABLE_PREFIX + str(index)
    cur.execute("insert into " + table_name + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
    cur.close()
    con.commit()
```

- Ta định nghĩa hàm **rangeinsert(...)** nhận vào các tham số sau:
    + ratingstablename: tên bảng gốc chứa ratings.
    + userid, itemid, rating: tương ứng các giá trị bản ghi mới.
    + openconnection: kết nối đến CSDL PostgreSQL (đã mở).
- Tạo biến con lấy kết nối từ openconnection, con trỏ cur = con.cursor() để thực hiện các truy vấn SQL.
- Gọi truy vấn để chèn bản ghi vào bảng chính Ratings.
- Gắn tên cho bảng phân mảnh qua biến RANGE_TABLE_PREFIX, sau đó đếm số phân mảnh qua hàm count_partitions() .

```
cur.execute("insert into " + ratingstablename + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
```

- Hàm **count_partitions(...)**: gọi truy vấn để đếm trong cơ sở dữ liệu các bảng có tên giống với tham số prefix (ở đây là tên các bảng phân mảnh được truyền vào qua RANGE_TABLE_PREFIX).

```
def count_partitions(prefix, openconnection):

    con = openconnection
    cur = con.cursor()
    cur.execute("select count(*) from pg_stat_user_tables where relname like " + "'" + prefix + "%';")
    count = cur.fetchone()[0]
    cur.close()

    return count
```

- Sau đó tính ngược lại delta = 5 / numberofpartitions
- Dựa vào rating muốn thêm vào và delta để tìm ra phân mảnh thích hợp cho bản ghi muốn thêm:

```
delta = 5 / numberofpartitions
index = int(rating / delta)
if rating % delta == 0 and index != 0:
    index = index - 1
table_name = RANGE_TABLE_PREFIX + str(index)
```

Có index = int(rating / delta) để tìm vị trí phân . VD: rating = 3.5, delta = 1.0 → 3.5 / 1.0 = 3.5 → int(3.5) = 3 → range_part3

- Tuy nhiên, nếu rating nằm tại biên của 1 phân mảnh, cần xem xét để tránh rơi nhầm sang phân mảnh khác.

  ⇒ Giải pháp: Nếu rating chia hết cho delta (VD: 3.0 % 1.0 == 0) và không phải phân mảnh đầu tiên (index ≠ 0) thì index -= 1. (do các phân mảnh lấy cận trên không lấy cận dưới)

- Sau khi chọn đúng phân mảnh, ta lấy tên bảng trong cơ sở dữ liệu dựa trên index. Sau đó thực hiện truy vấn đề chèn dữ liệu vào.

```
cur.execute("insert into " + table_name + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
cur.close()
con.commit()
```

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

6. Xây dựng hàm RoundRobin_Insert() để chèn một bộ dữ liệu mới vào bảng Ratings và vào đúng phân mảnh theo cách round robin

```
def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):

    con = openconnection
    cur = con.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'
    cur.execute("insert into " + ratingstablename + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
    cur.execute("select count(*) from " + ratingstablename + ";")
    total_rows = (cur.fetchall())[0][0]
    numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
    index = (total_rows-1) % numberofpartitions
    table_name = RROBIN_TABLE_PREFIX + str(index)
    cur.execute("insert into " + table_name + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
    cur.close()
    con.commit()
```

-
- Ta định nghĩa hàm **roundrobininsert(...)** nhận vào các tham số sau:
  + ratingstablename: tên bảng gốc chứa ratings.
  + userid, itemid, rating: tương ứng các giá trị bản ghi mới.
  + openconnection: kết nối đến CSDL PostgreSQL (đã mở).
- Tạo biến con lấy kết nối từ openconnection, con trỏ cur = con.cursor() để thực hiện các truy vấn SQL.
- Đặt tiền tố cho phân mảnh bảng: RROBIN_TABLE_PREFIX = 'rrobin_part'.

```
con = openconnection
cur = con.cursor()
RROBIN_TABLE_PREFIX = 'rrobin_part'
cur.execute("insert into " + ratingstablename + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
```

- Gọi truy vấn để chèn bản ghi vào bảng chính Ratings.
- Sau khi chèn xong, ta gọi truy vấn để đếm tổng số bản ghi sau chèn. Dùng cur.fetchall()[0][0] để lấy dữ liệu sau truy vấn và gắn vào biến total_rows. Do đã chèn bản ghi mới ở bước trước, nên dòng vừa thêm là dòng thứ total_rows.

```
cur.execute("select count(*) from " + ratingstablename + ";")
total_rows = (cur.fetchall())[0][0]
numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
index = (total_rows-1) % numberofpartitions
table_name = RROBIN_TABLE_PREFIX + str(index)
```

Ta gọi hàm **count_partitions()** để đếm có bao nhiêu bảng phân mảnh. Hàm này ta đã phân tích ở mục 5. Gắn dữ liệu cho biến numberofpartitions.
Tiếp theo ta tìm phân mảnh thích hợp cho bản ghi:

index = (total_rows - 1) % numberofpartitions

Vì đã thêm dòng mới, nó nằm ở vị trí total_rows, nhưng đánh chỉ số từ 0 nên phải dùng total_rows - 1. Sau đó chia dư cho số phân mảnh để tìm ra.

- Sau khi chọn đúng phân mảnh, ta lấy tên bảng trong cơ sở dữ liệu dựa trên index. Sau đó thực hiện truy vấn đề chèn dữ liệu vào.

```
cur.execute("insert into " + table_name + "(userid, movieid, rating) values (" + str(userid) + "," + str(itemid) + "," + str(rating) + ");")
cur.close()
con.commit()
```

- Cuối cùng đóng cursor và kết nối sau khi hoàn tất.

IV. THỰC HIỆN PHÂN MẢNH TRÊN TẬP DỮ LIỆU

1. Hàm Tester để kiểm tra phân mảnh (Assignment1Tester.py)

- Ta sẽ cấu hình tên cơ sở dữ liệu.
- Sau đó sử dụng Path(__file__).parent lấy thư mục hiện tại chứa file Python đang chạy. Nối thêm "ml-10M100K/ratings.dat" để tạo đường dẫn đầy đủ tới file dữ liệu.
- Sau đó định nghĩa cho tên các bảng, cột trong cơ sở dữ liệu. Định nghĩa cả đường dẫn đến file dữ liệu cùng số dòng trong file.
- Cuối cùng Import các thư viện cần thiết (psycopg2, traceback) và các file (testHelper: chứa các file hỗ trợ test, Interface: chứa các hàm phân mảnh theo yêu cầu đề bài).

```
DATABASE_NAME = 'ddb_assignment1'
from pathlib import Path
current_dir = Path(__file__).parent
print(f"Current directory: {current_dir}")
file_path = current_dir /"ml-10M100K"/"ratings.dat"
print(f"File path: {file_path}")

RATINGS_TABLE = 'ratings'
RANGE_TABLE_PREFIX = 'range_part'
RROBIN_TABLE_PREFIX = 'rrobin_part'
USER_ID_COLNAME = 'userid'
MOVIE_ID_COLNAME = 'movieid'
RATING_COLNAME = 'rating'
INPUT_FILE_PATH = file_path
ACTUAL_ROWS_IN_INPUT_FILE = 10000054 #nmber of lines in the input file

import psycopg2
import traceback
import testHelper
import Interface as MyAssignment
```

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

        with testHelper.getopenconnection(dbname=DATABASE_NAME) as conn:

            conn.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)

            testHelper.deleteAllPublicTables(conn)
```

- Khởi tạo main cho file kiểm thử. Bắt đầu gọi hàm **createdb()** từ module testHelper để tạo CSDL ddb_assignment1 (nếu chưa có).
- Dùng context manager (with) để tự đóng kết nối sau khi xong.
- AUTOCOMMIT giúp chạy các câu CREATE, DROP, ... mà không cần gọi commit().
- Xóa sạch toàn bộ các bảng trước khi test, để tránh trùng tên, dữ liệu cũ...

- Sau khi thực hiện xong ta bắt đầu chuyển qua test tính đúng đắn của các thuật toán phân mảnh ta triển khai thông qua việc gọi các hàm:
    + **Test loadratings**: Gọi đến hàm trong testHelper để kiểm tra xem dữ liệu có được load đến bảng Ratings một cách đúng đắn hay không

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

    + **Test rangepartition**: Gọi đến hàm trong testHelper kiểm tra xem liệu đã phân mảnh cơ sở dữ liệu theo Ratings thành 5 bảng. Kiểm tra tính đúng đắn của các phân mảnh.

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

    + **Test rangeinsert**: Gọi đến hàm trong testHelper kiểm tra xem chèn một bản ghi mới đã đúng phân mảnh range hay chưa.

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

testHelper.deleteAllPublicTables(conn)
```

    + **Test roundrobinpartition**: Gọi đến hàm trong testHelper kiểm tra xem liệu đã phân mảnh cơ sở dữ liệu theo Ratings thành 5 bảng. Kiểm tra tính đúng đắn của các phân mảnh.

```
print("\nTesting roundrobinpartition functions")
MyAssignment.loadratings(RATINGS_TABLE, INPUT_FILE_PATH, conn)

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

+ **Test roundrobininsert**: Gọi đến hàm trong testHelper kiểm tra xem chèn một bản ghi mới đã đúng phân mảnh theo phương pháp roundrobin hay chưa.

```
# ALERT:: Change the partition index according to your testing sequence.
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '4')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '1')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
```

- Cuối cùng cho lựa chọn xóa tất cả các bảng sau khi kiểm thử xong. Và exception để bắt lỗi nếu có

```
            choice = input('Press "y" to Delete all tables? ')
            if choice == 'y':
                testHelper.deleteAllPublicTables(conn)
            if not conn.close:
                conn.close()

    except Exception as detail:
        traceback.print_exc()
```

2. Các hàm TestHelper (testHelper.py)

a. Các hàm setup database

- Hàm **createdb(dbname)**: Tạo cơ sở dữ liệu nếu chưa tồn tại

```
def createdb(dbname):

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

    # 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))

    # Clean up
    cur.close()
    con.close()
```

  + Kết nối tới DB mặc định (postgres) và bật chế độ AUTOCOMMIT vì lệnh CREATE DATABASE không thể thực hiện trong transaction.
  + Sau đó thực hiện truy vấn kiểm tra xem database có tồn tại chưa bằng cách đếm tất cả các database có tên giống dbname. Gắn dữ liệu lấy được vào count và kiểm tra.
  + Nếu count = 0 tức chưa tồn tại thì tạo database mới bằng truy vấn, ngược lại in thông báo.

- Hàm **delete_db(dbname)**: Xóa một cơ sở dữ liệu

```
def delete_db(dbname):
    con = getopenconnection(dbname = 'postgres')
    con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    cur = con.cursor()
    cur.execute('drop database ' + dbname)
    cur.close()
    con.close()
```

  + Kết nối đến database thông qua hàm **getopenconnection**()
  + Bật AUTOCOMMIT tương tự như trên.
  + Thực hiện lệnh truy vấn DROP DATABASE thông qua con trỏ cursor().
  + Sau khi thực hiện xong thì ngắt kết nối.

- Hàm **deleteAllPublicTables(openconnection)**: Xóa tất cả các bảng trong schema public.

```
def deleteAllPublicTables(openconnection):
    cur = openconnection.cursor()
    cur.execute("SELECT table_name FROM information_schema.tables WHERE table_schema = 'public'")
    l = []
    for row in cur:
        l.append(row[0])
    for tablename in l:
        cur.execute("drop table if exists {0} CASCADE".format(tablename))

    cur.close()
```

  + Thực hiện truy vấn để lấy tên bảng trong schema public thông qua con trỏ cursor.
  + Gán dữ liệu lấy được vào mảng l.
  + Duyệt từng tên bảng trong mảng l để đưa vào truy vấn xóa bảng đó. Sau khi thực hiện xong đóng con trỏ cursor.
- Hàm **getopenconnection(...)**: Mở một kết nối PostgreSQL với cấu hình tùy chỉnh (port, host, user, password, dbname).

```
def getopenconnection(user='postgres', password='newpassword', port=5001, host='localhost', dbname='postgres'):
    conn_params = {
        'dbname': dbname,
        'user': user,
        'password': password,
        'host': host,
        'port': port
    }
    print(f"Connecting to the database {conn_params}")
    conn = psycopg2.connect(**conn_params)
    print(f"Connected to the database {conn_params}")
    return conn
```

  + Tạo một từ điển conn_params để cấu hình mặc định cho database.
  + Sau đó mở kết nối đến cơ sở dữ liệu qua lệnh conn = psycopg2.connect(**conndatabase.
  + In ra thông báo kết nối thành công và trả về kết nối: return conn.

b. Các hàm TesterSupport

- Hàm **getCountrangepartition(...)**: Đếm số bản ghi theo mỗi khoảng rating trong bảng chính
    + Tạo mảng countList[]. Phần tử countList[i] sẽ lưu số bản ghi nằm trong phân mảnh thứ i
    + Thực hiện tương tự như ý tưởng lúc phân mảnh. Lặp và dựa vào interval = 5/numberofpartitions để thực hiện truy vấn đếm các bản ghi có rating thuộc khoảng đã chia trong bảng ratings (Lưu ý cần lấy cận dưới = 0.0)
    + Dùng fetchone()[0] để lấy kết quả là số bản ghi đếm được của khoảng đánh giá và append vào countList[].

```
def getCountrangepartition(ratingstablename, numberofpartitions, openconnection):

    cur = openconnection.cursor()
    countList = []
    interval = 5.0 / numberofpartitions
    cur.execute("select count(*) from {0} where rating >= {1} and rating <= {2}".format(ratingstablename,0, interval))
    countList.append(int(cur.fetchone()[0]))

    lowerbound = interval
    for i in range(1, numberofpartitions):
        cur.execute("select count(*) from {0} where rating > {1} and rating <= {2}".format(ratingstablename,
                                                                                           lowerbound,
                                                                                           lowerbound + interval))
        lowerbound += interval
        countList.append(int(cur.fetchone()[0]))

    cur.close()
    return countList
```

- Hàm **getCountroundrobinpartition(...)**: Đếm số bản ghi mỗi loại theo phương pháp roundrobin trong bảng chính.
    + Tạo mảng countList[]. Phần tử countList[i] sẽ lưu số bản ghi nằm trong phân mảnh thứ i
    + Lặp qua từng giá trị i: Tạo một bảng tạm temp với số thứ tự dòng row_number(). Với mỗi dòng, tính (row_number - 1) % numberofpartitions == i để xác định xem dòng đó thuộc phân mảnh nào. Đếm số dòng thuộc phân mảnh i và thêm vào countList[] thông qua fetchone()[0]

```
def getCountroundrobinpartition(ratingstablename, numberofpartitions, openconnection):

    cur = openconnection.cursor()
    countList = []
    for i in range(0, numberofpartitions):
        cur.execute(
            "select count(*) from (select *, row_number() over () from {0}) as temp where (row_number-1)%{1}= {2}".format(
                ratingstablename, numberofpartitions, i))
        countList.append(int(cur.fetchone()[0]))

    cur.close()
    return countList
```

- Hàm **checkpartitioncount(...)**: Kiểm tra số lượng bảng có tiền tố prefix, ví dụ: range_part0, range_part1,... xem có đúng bằng expectedpartitions hay không.
    + Thực hiện truy vấn để đếm số bảng trong schema public có tên bắt đầu bằng prefix.
    + Lấy số lượng bảng thực tế từ kết quả truy vấn qua fetchone()[0] và gán vào biến count. Dùng count để so sánh với giá trị mong muốn. Nếu số bảng thực tế khác với số bảng mong đợi, ném ra ngoại lệ (Exception) kèm thông báo lỗi.

```
def checkpartitioncount(cursor, expectedpartitions, prefix):
    cursor.execute(
        "SELECT COUNT(table_name) FROM information_schema.tables WHERE table_schema = 'public' AND table_name LIKE '{0}%';".format(
            prefix))
    count = int(cursor.fetchone()[0])
    if count != expectedpartitions:  raise Exception(
        'Range partitioning not done properly. Excepted {0} table(s) but found {1} table(s)'.format(
            expectedpartitions,
            count))
```

- Hàm **totalrowsinallpartitions(...)**: Kiểm tra tổng số lượng bản ghi dữ liệu có trong tất cả các phân mảnh.
    + Tạo mảng selects[] để lưu tập các câu lệnh truy vấn
    + Lặp để tạo danh sách các câu lệnh SELECT * FROM range_part0, range_part1, ..., range_part(n-1). Lưu vào mảng selects[]
    + Thực hiện truy vấn tổng để đếm tổng số dòng có trong tất cả các phân mảnh: Ghép các truy vấn lại bằng UNION ALL để tạo một bảng tạm chứa toàn bộ các dòng từ mọi phân mảnh, rồi đếm tổng số dòng.

```
def totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex):
    selects = []
    for i in range(partitionstartindex, n + partitionstartindex):
        selects.append('SELECT * FROM {0}{1}'.format(rangepartitiontableprefix, i))
    cur.execute('SELECT COUNT(*) FROM ({0}) AS T'.format(' UNION ALL '.join(selects)))
    count = int(cur.fetchone()[0])
    return count
```

- Hàm **testrangeandrobinpartitioning(...)**: Hàm tổng hợp để kiểm tra tính đúng đắn của phân mảnh với các thuộc tính kiểm tra: Số bảng, Tính đầy đủ (completeness) , Tính rời rạc (disjointness), Tính tái tạo (reconstruction).
    + ***Test 1***: Đầu vào không hợp lệ. Nếu n không phải số nguyên hoặc nhỏ hơn 0 → không có bảng nào được tạo.
      → Gọi **checkpartitioncount**() để kiểm tra có 0 bảng.

+ ***Test 2***: Đầu vào hợp lệ → Đảm bảo có đúng n bảng phân mảnh được tạo.
+ ***Test 3***: Kiểm tra tính đầy đủ (Completeness):
  Gọi hàm **totalrowsinallpartitions(...)** đếm tổng số dòng trong các phân mảnh
  Sau phân mảnh, tổng số dòng trong các bảng phân mảnh mà < số dòng gốc → dữ liệu bị mất → lỗi.
+ ***Test 4***: Kiểm tra tính rời rạc (Disjointness):
  Không bảng nào chứa bản ghi trùng lặp. Nếu tổng số dòng sau khi gộp > ban đầu → có trùng lặp → lỗi.
+ ***Test 5***: Kiểm tra tính tái tạo (Reconstruction):
  Sau khi gộp lại từ các phân mảnh, phải khôi phục đúng số dòng ban đầu. Nếu khác → lỗi.

```python
def testrangeandrobinpartitioning(n, openconnection, rangepartitiontableprefix, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    with openconnection.cursor() as cur:
        if not isinstance(n, int) or n < 0:
            # Test 1: Check the number of tables created, if 'n' is invalid
            checkpartitioncount(cur, 0, rangepartitiontableprefix)
        else:
            # Test 2: Check the number of tables created, if all args are correct
            checkpartitioncount(cur, n, rangepartitiontableprefix)

            # Test 3: Test Completeness by SQL UNION ALL Magic
            count = totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex)
            if count < ACTUAL_ROWS_IN_INPUT_FILE: raise Exception(
                "Completeness property of Partitioning failed. Excpected {0} rows after merging all tables, but found {1} rows".format(
                    ACTUAL_ROWS_IN_INPUT_FILE, count))

            # Test 4: Test Disjointness by SQL UNION Magic
            count = totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex)
            if count > ACTUAL_ROWS_IN_INPUT_FILE: raise Exception(
                "Dijointness property of Partitioning failed. Excpected {0} rows after merging all tables, but found {1} rows".format(
                    ACTUAL_ROWS_IN_INPUT_FILE, count))

            # Test 5: Test Reconstruction by SQL UNION Magic
            count = totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex)
            if count != ACTUAL_ROWS_IN_INPUT_FILE: raise Exception(
                "Rescontruction property of Partitioning failed. Excpected {0} rows after merging all tables, but found {1} rows".format(
                    ACTUAL_ROWS_IN_INPUT_FILE, count))
```

- Hàm **testrangerobininsert(...)**: Kiểm tra xem bản ghi (userid, itemid, rating) có đúng trong expectedtablename không.
  + Mở con trỏ để thực hiện truy vấn SQL. Truy vấn để tìm xem có bản ghi nào trong bảng expectedtablename có đúng: user_id = userid, movie_id = itemid, rating = rating.
  + Dùng fetchone()[0] lấy kết quả truy vấn lưu vào count. Nếu không tìm thấy bản ghi khớp (hoặc có nhiều hơn 1) → trả về False (sai), ngược lại True.

```
def testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
    with openconnection.cursor() as cur:
        cur.execute(
            'SELECT COUNT(*) FROM {0} WHERE {4} = {1} AND {5} = {2} AND {6} = {3}'.format(expectedtablename, userid,
                                                                                          itemid, rating,
                                                                                          USER_ID_COLNAME,
                                                                                          MOVIE_ID_COLNAME,
                                                                                          RATING_COLNAME))
        count = int(cur.fetchone()[0])
        if count != 1:  return False
        return True
```

- Hàm **testEachRangePartition(...)** và **testEachRoundrobinPartition(...)**: So sánh số bản ghi trong từng bảng phân mảnh theo 2 phương pháp xem có đúng với dự đoán mong đợi.
    + Gọi hàm getCount…() để lấy số bản ghi tương ứng của từng phân mảnh theo mỗi thuật toán từ bảng Ratings chính. Sau đó lưu vào countList.
    + Duyệt qua từng bảng đã phân mảnh và đếm số dòng tương ứng thông qua truy vấn. Lưu kết quả truy vấn vào biến count và so sánh luôn với countList[i]. Nếu không khớp → báo lỗi.

```
def testEachRangePartition(ratingstablename, n, openconnection, rangepartitiontableprefix):
    countList = getCountrangepartition(ratingstablename, n, openconnection)
    cur = openconnection.cursor()
    for i in range(0, n):
        cur.execute("select count(*) from {0}{1}".format(rangepartitiontableprefix, i))
        count = int(cur.fetchone()[0])
        if count != countList[i]:
            raise Exception("{0}{1} has {2} of rows while the correct number should be {3}".format(
                rangepartitiontableprefix, i, count, countList[i]
            ))

def testEachRoundrobinPartition(ratingstablename, n, openconnection, roundrobinpartitiontableprefix):
    countList = getCountroundrobinpartition(ratingstablename, n, openconnection)
    cur = openconnection.cursor()
    for i in range(0, n):
        cur.execute("select count(*) from {0}{1}".format(roundrobinpartitiontableprefix, i))
        count = cur.fetchone()[0]
        if count != countList[i]:
            raise Exception("{0}{1} has {2} of rows while the correct number should be {3}".format(
                roundrobinpartitiontableprefix, i, count, countList[i]
            ))
```

c. Các hàm Test chính

- Hàm **testloadratings(...)**: Kiểm tra xem hàm **loadratings(...)** có tải đúng số lượng bản ghi từ file dữ liệu vào bảng chính Ratings hay không.
    + Gọi hàm **loadratings(...)** từ module MyAssignment (chính là file Interface).
    + Tạo con trỏ thực hiện truy vấn đếm tất cả các bản ghi trong bảng chính. Dùng fetchone() lấy kết quả truy vấn và lưu vào count để so sánh.
    + So sánh với số dòng kỳ vọng, nếu không khớp → raise Exception và trả về False. Ngược lại trả về True.

```
def testloadratings(MyAssignment, ratingstablename, filepath, openconnection, rowsininpfile):
    try:
        MyAssignment.loadratings(ratingstablename,filepath,openconnection)
        # Test 1: Count the number of rows inserted
        with openconnection.cursor() as cur:
            cur.execute('SELECT COUNT(*) from {0}'.format(ratingstablename))
            count = int(cur.fetchone()[0])
            if count != rowsininpfile:
                raise Exception(
                    'Expected {0} rows, but {1} rows in \'{2}\' table'.format(rowsininpfile, count, ratingstablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

- Hàm **testrangepartition(...)**: Kiểm tra hàm **rangepartition(...)** có phân mảnh theo range chính xác không.
    + Gọi hàm **rangepartition(...)** với các tham số đầu vào.
    + Gọi hàm **testrangeandrobinpartitioning(...)** để kiểm tra các tính đúng đắn của các phân mảnh: số phân mảnh tạo ra, Tính đầy đủ (Completeness), Tính rời rạc (Disjointness), Tính tái tạo (Reconstruction).
    + Gọi hàm **testEachRangePartition(...)** để kiểm tra: Mỗi bảng con chứa đúng số dòng được tính toán dựa theo khoảng giá trị.
    + Nếu có lỗi → raise Exception và trả về False. Ngược lại trả về True.

```
def testrangepartition(MyAssignment, ratingstablename, n, openconnection, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    try:
        MyAssignment.rangepartition(ratingstablename, n, openconnection)
        testrangeandrobinpartitioning(n, openconnection, RANGE_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRangePartition(ratingstablename, n, openconnection, RANGE_TABLE_PREFIX)
        return [True, None]
    except Exception as e:
        traceback.print_exc()
        return [False, e]
```

- Hàm **testroundrobinpartition(...)**: Kiểm tra **roundrobinpartition(...)** có phân mảnh theo phương pháp roundrobin chính xác hay không.
    + Gọi hàm **roundrobinpartition(...)** với các tham số đầu vào.
    + Gọi hàm **testrangeandrobinpartitioning(...)** để kiểm tra các tính đúng đắn của các phân mảnh: số phân mảnh tạo ra, Tính đầy đủ (Completeness), Tính rời rạc (Disjointness), Tính tái tạo (Reconstruction).
    + Gọi hàm **testEachRoundrobinPartition(...)** để kiểm tra: Mỗi bảng con chứa đúng số dòng được phân mảnh bằng roundrobin.
    + Nếu có lỗi → raise Exception và trả về False. Ngược lại trả về True.

```
def testroundrobinpartition(MyAssignment, ratingstablename, numberofpartitions, openconnection,
                            partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):

    try:
        MyAssignment.roundrobinpartition(ratingstablename, numberofpartitions, openconnection)
        testrangeandrobinpartitioning(numberofpartitions, openconnection, RROBIN_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRoundrobinPartition(ratingstablename, numberofpartitions, openconnection, RROBIN_TABLE_PREFIX)
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

- Hàm **testroundrobininsert(...)**: Kiểm tra hàm **roundrobininsert(...)** có chèn đúng vào phân mảnh hay không.
    + Ghép tên bảng con mong đợi.
    + Gọi hàm **roundrobininsert(...)** từ module MyAssignment (chính là file Interface) để chèn bản ghi mong muốn theo phương pháp roundrobin.
    + Gọi hàm phụ **testrangerobininsert(...)** để kiểm tra xem bản ghi có tồn tại đúng trong bảng con mong đợi không.
    + Nếu bản ghi không tồn tại → raise Exception và trả về False. Còn lại trả về True.

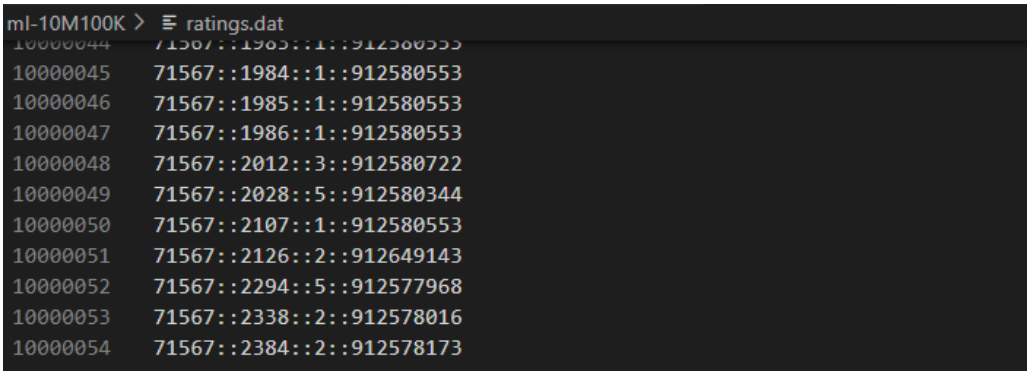
```
def testroundrobininsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):

    try:
        expectedtablename = RROBIN_TABLE_PREFIX + expectedtableindex
        MyAssignment.roundrobininsert(ratingstablename, userid, itemid, rating, openconnection)
        if not testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
            raise Exception(
                'Round robin insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format(userid, itemid, rating,
                                                                                                    expectedtablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

- Hàm **testrangeinsert(...)**: Kiểm tra hàm **rangeinsert(...)** có chèn đúng vào phân mảnh hay không.
    + Ghép tên bảng con mong đợi.
    + Gọi hàm **rangeinsert(...)** từ module MyAssignment (chính là file Interface) để chèn bản ghi mong muốn theo range của ratings.
    + Gọi hàm phụ **testrangerobininsert(...)** để kiểm tra xem bản ghi có tồn tại đúng trong bảng con mong đợi không.
    + Nếu bản ghi không tồn tại → raise Exception và trả về False. Còn lại trả về True.

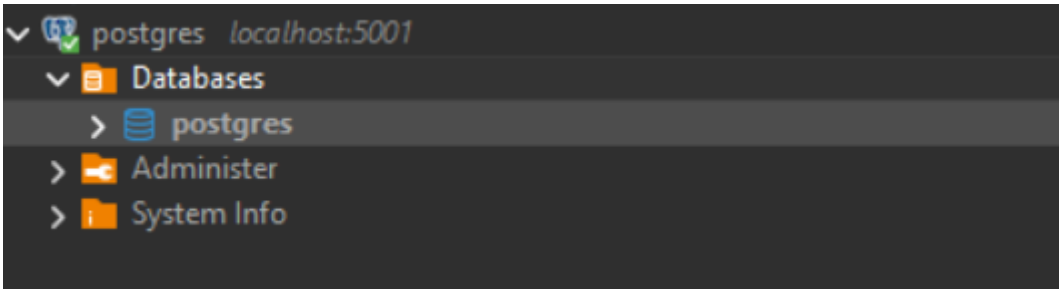
```
def testrangeinsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):

    try:
        expectedtablename = RANGE_TABLE_PREFIX + expectedtableindex
        MyAssignment.rangeinsert(ratingstablename, userid, itemid, rating, openconnection)
        if not testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
            raise Exception(
                'Range insert failed! Couldnt find ({0}, {1}, {2}) tuple in {3} table'.format(userid, itemid, rating,
                                                                                              expectedtablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

3. Kết quả khi thực hiện phân mảnh

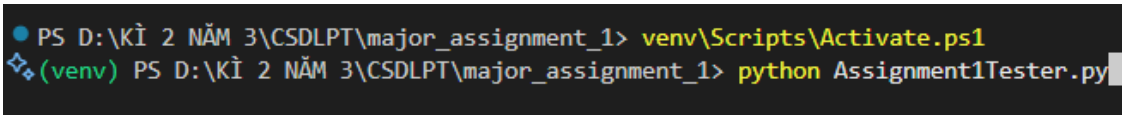
- Dữ liệu là file **Ratings.dat** chứa 10000054 bản ghi về thông tin đánh giá phim:



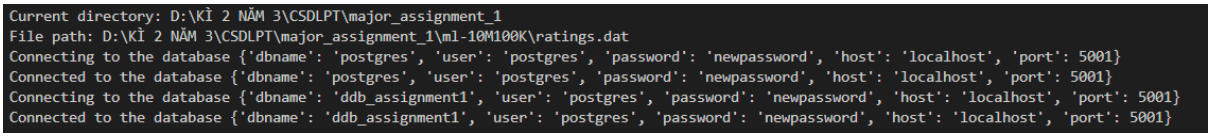
- Cơ sở dữ liệu ban đầu: gồm 1 cơ sở dữ liệu mặc định



- Thực hiện chạy file test: **Assignment1Tester.py**



- Trong terminal hiển thị các thông tin về vị trí hiện tại cũng như vị trí của file dữ liệu trong project. Sau đó hiển thị kết nối đến cơ sở dữ liệu mặc định cũng như cơ sở dữ liệu riêng "**ddb_assignment1**" để bắt đầu thực hiện các test.

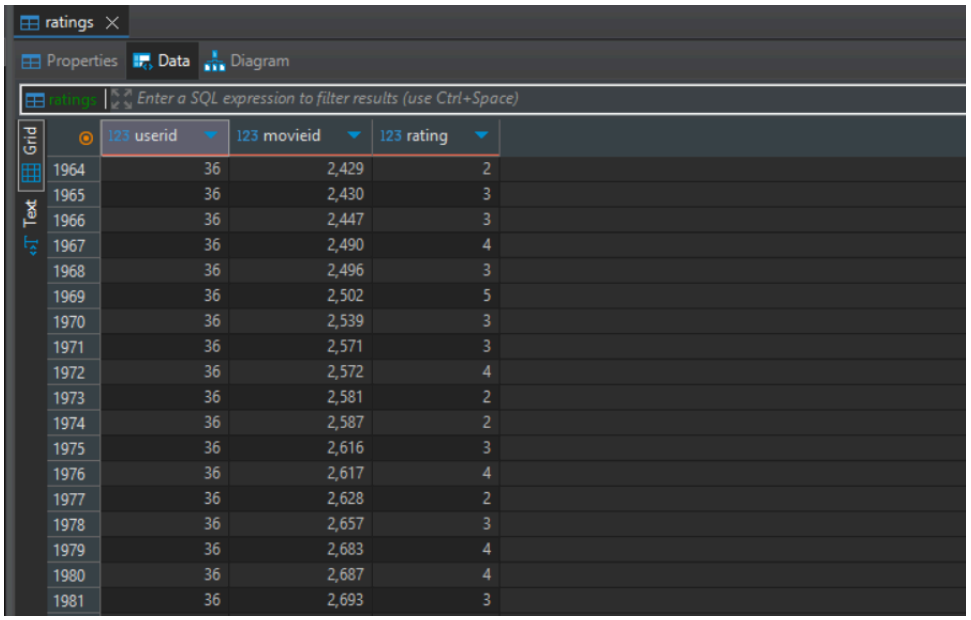


- Trong Assignment1Tester.py có 5 test case. Ta sẽ đi lần lượt đi vào chi tiết từng test.
- **Test loadratings**: Gọi đến hàm trong testHelper để kiểm tra xem dữ liệu có được load đến bảng Ratings một cách đúng đắn hay không.

  ⇒ Ta nhận thấy đã hiển thị 5 dòng đầu tiên của phân mảnh cũng như thông báo "loadratings function pass!".

```
Testing loadratings function
Loading data from D:\KÌ 2 NĂM 3\CSDLPT\major_assignment_1\ml-10M100K\ratings.dat to ratings
Row: (1, 122, 5.0)
Row: (1, 185, 5.0)
Row: (1, 231, 5.0)
Row: (1, 292, 5.0)
Row: (1, 316, 5.0)
loadratings function pass!
```

+ Kiểm tra lại trong cơ sở dữ liệu ddb_assignment1, toàn bộ dữ liệu đã được tải vào bảng Ratings



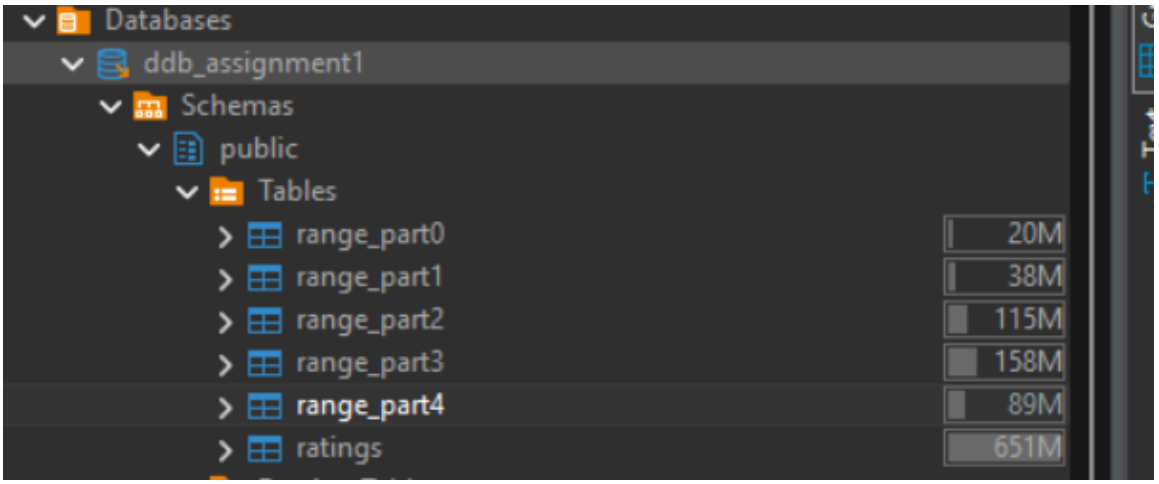
	123 userid	123 movieid	123 rating
1964	36	2,429	2
1965	36	2,430	3
1966	36	2,447	3
1967	36	2,490	4
1968	36	2,496	3
1969	36	2,502	5
1970	36	2,539	3
1971	36	2,571	3
1972	36	2,572	4
1973	36	2,581	2
1974	36	2,587	2
1975	36	2,616	3
1976	36	2,617	4
1977	36	2,628	2
1978	36	2,657	3
1979	36	2,683	4
1980	36	2,687	4
1981	36	2,693	3

- **Test rangepartition**: Gọi đến hàm trong testHelper kiểm tra xem liệu đã phân mảnh cơ sở dữ liệu theo Ratings thành 5 bảng. Kiểm tra tính đúng đắn của các phân mảnh.
- **Test rangeinsert**: Gọi đến hàm trong testHelper kiểm tra xem chèn một bản ghi mới đã đúng phân mảnh range hay chưa.

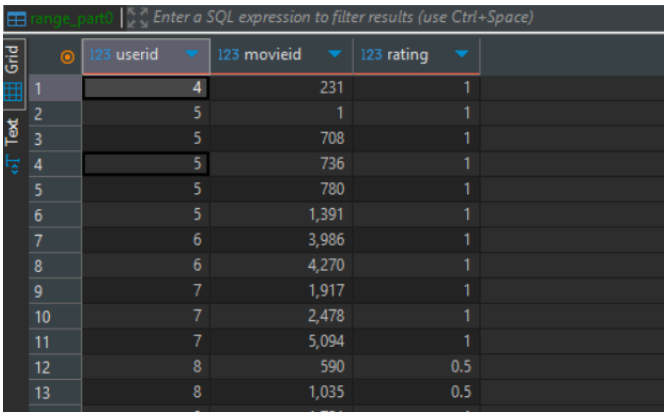
  ⇒ Terminal hiển thị các giá trị: delta dùng để chi range cho giá trị ratings. Hiện thông báo đã chèn bao nhiêu dòng vào các bảng phân mảnh dựa trên ratings. Cuối cùng là thông báo pass các test case.

```
Testing rangepartition functions
Delta: 1.0
[DEBUG] Inserted 479168 rows into range_part0
[DEBUG] Inserted 908584 rows into range_part1
[DEBUG] Inserted 2726854 rows into range_part2
[DEBUG] Inserted 3755614 rows into range_part3
[DEBUG] Inserted 2129834 rows into range_part4
rangepartition function pass!
rangeinsert function pass!
```

+ Để kiểm tra trong cơ sở dữ liệu, ta sẽ **tạm bỏ qua** bước xóa các bảng cũ bằng **testHelper.deleteAllPublicTables(conn)**. Mở cơ sở dữ liệu ta thấy có các bảng phân mảnh theo range thỏa mãn.

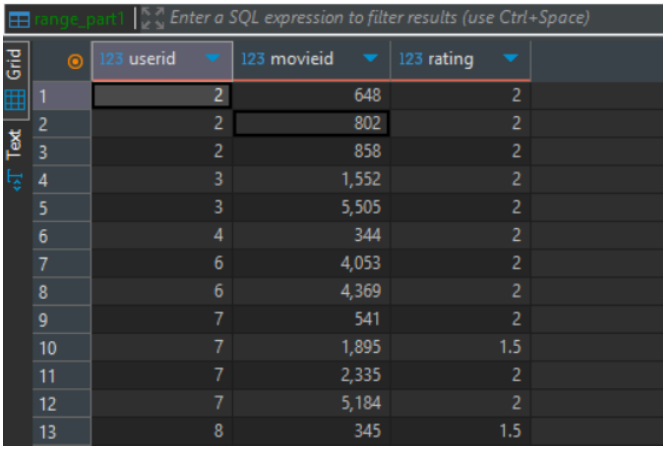


+ Các phân mảnh được chia theo khoảng ratings ⇒ có thể thấy dữ liệu tập trung nhiều hơn vào các range_part2,3,4; tức phim có đánh giá khoảng từ 3-5 sao.
+ range_part0 chứa các phim có ratings từ 0 đến 1 sao:



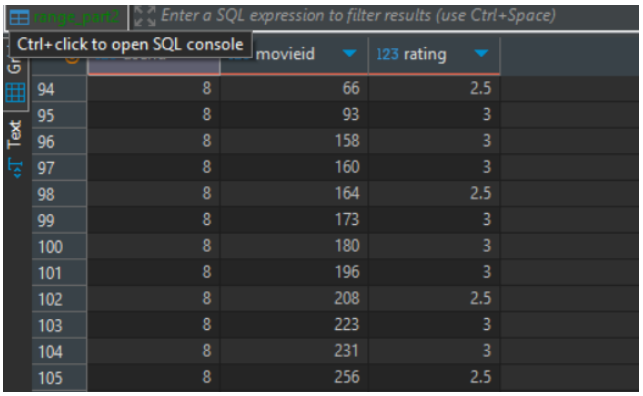
	userid	movieid	rating
1	4	231	1
2	5	1	1
3	5	708	1
4	5	736	1
5	5	780	1
6	5	1,391	1
7	6	3,986	1
8	6	4,270	1
9	7	1,917	1
10	7	2,478	1
11	7	5,094	1
12	8	590	0.5
13	8	1,035	0.5

+ range_part1 chứa các phim có ratings từ 1.5 đến 2 sao:



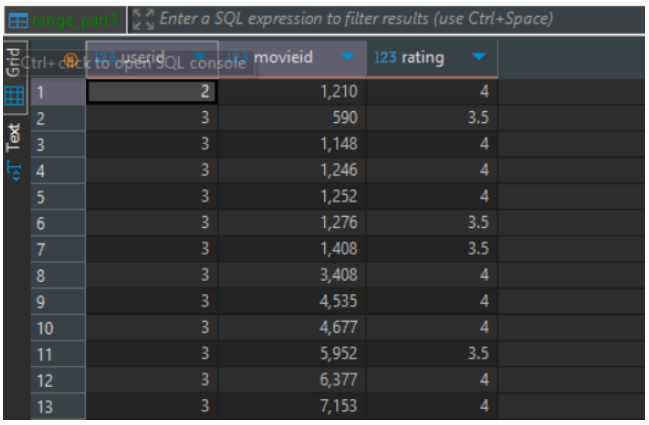
	userid	movieid	rating
1	2	648	2
2	2	802	2
3	2	858	2
4	3	1,552	2
5	3	5,505	2
6	4	344	2
7	6	4,053	2
8	6	4,369	2
9	7	541	2
10	7	1,895	1.5
11	7	2,335	2
12	7	5,184	2
13	8	345	1.5

+ range_part2 chứa các phim có ratings từ 2.5 đến 3 sao:



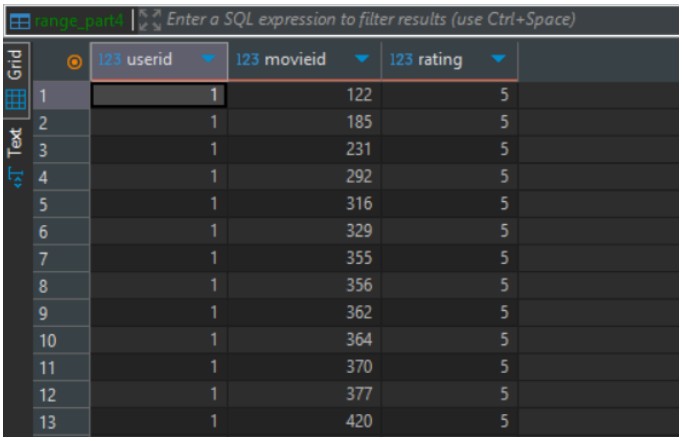
		movieid	rating
94	8	66	2.5
95	8	93	3
96	8	158	3
97	8	160	3
98	8	164	2.5
99	8	173	3
100	8	180	3
101	8	196	3
102	8	208	2.5
103	8	223	3
104	8	231	3
105	8	256	2.5

+ range_part3 chứa các phim có ratings từ 3.5 đến 4 sao:



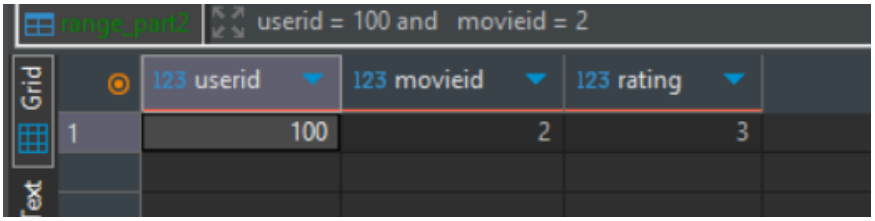
	userid	movieid	rating
1	2	1,210	4
2	3	590	3.5
3	3	1,148	4
4	3	1,246	4
5	3	1,252	4
6	3	1,276	3.5
7	3	1,408	3.5
8	3	3,408	4
9	3	4,535	4
10	3	4,677	4
11	3	5,952	3.5
12	3	6,377	4
13	3	7,153	4

+ range_part4 chứa các phim có ratings từ 4.5 đến 5 sao:



	userid	movieid	rating
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

+ Cùng với đó bộ test (userid, movieid, ratings) = (100, 2, 3) cũng đã được chèn thành công vào phân mảnh 2.



	userid	movieid	rating
1	100	2	3

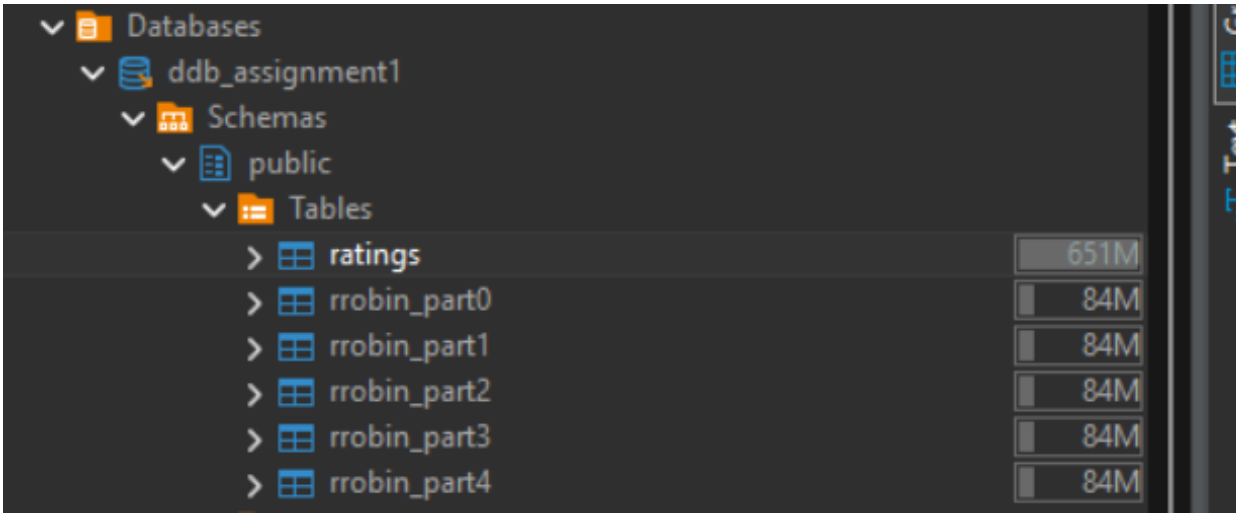
- **Test roundrobinpartition**: Gọi đến hàm trong testHelper kiểm tra xem liệu đã phân mảnh cơ sở dữ liệu theo Ratings thành 5 bảng. Kiểm tra tính đúng đắn của các phân mảnh.
- **Test roundrobininsert**: Gọi đến hàm trong testHelper kiểm tra xem chèn một bản ghi mới đã đúng phân mảnh theo phương pháp roundrobin hay chưa.

  ⇒ Sau khi phân mảnh theo range thì có gọi hàm **testHelper.deleteAllPublicTables(conn)** để xóa hết các bảng cũ. Sau đó gọi hàm để load lại dữ liệu vào bảng chính ratings.

  ⇒ Terminal hiển thị đã pass 2 test về roundrobinpartition và roundrobininsert.

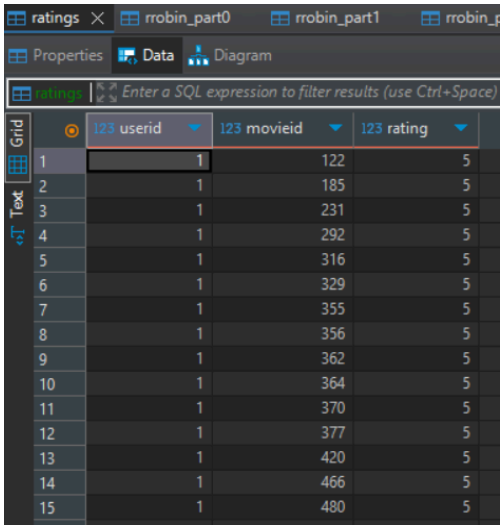
```
Testing roundrobinpartition functions
Loading data from D:\KÌ 2 NĂM 3\CSDLPT\major_assignment_1\ml-10M100K\ratings.dat to ratings
Row: (1, 122, 5.0)
Row: (1, 185, 5.0)
Row: (1, 231, 5.0)
Row: (1, 292, 5.0)
Row: (1, 316, 5.0)
roundrobinpartition function pass!
roundrobininsert function pass!
Press "y" to Delete all tables?
```

+ Ta kiểm tra trong cơ sở dữ liệu, nhận thấy đã có 5 bảng phân mảnh theo roundrobin

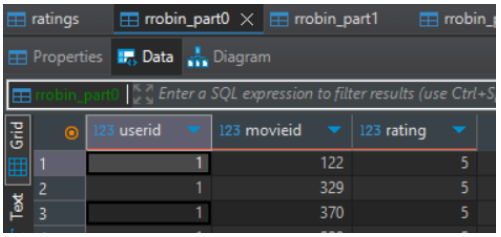


+ Các phân mảnh đều đã chứa dữ liệu được xoay vòng lần lượt. Ta có thể so sánh một cách trực quan.

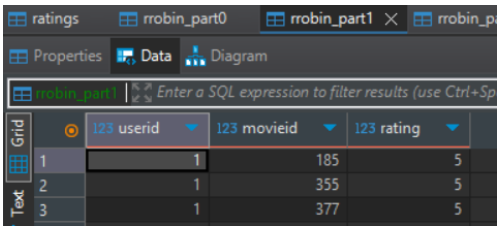
+ Có thể lấy ví dụ 1 phần nhỏ: 15 bản ghi đầu tiên của ratings đã được phân mảnh đều vào 5 bảng phân mảnh:



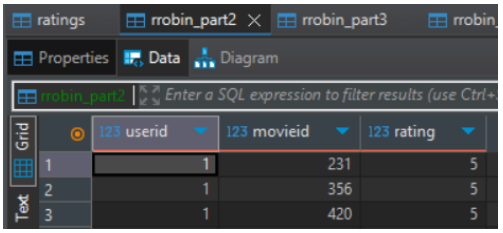
	userid	movieid	rating
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5
13	1	420	5
14	1	466	5
15	1	480	5



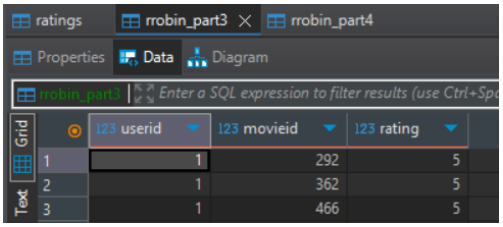
	userid	movieid	rating
1	1	122	5
2	1	329	5
3	1	370	5



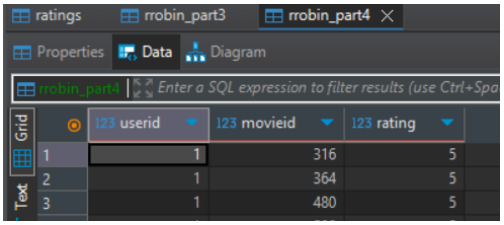
	userid	movieid	rating
1	1	185	5
2	1	355	5
3	1	377	5



	userid	movieid	rating
1	1	231	5
2	1	356	5
3	1	420	5

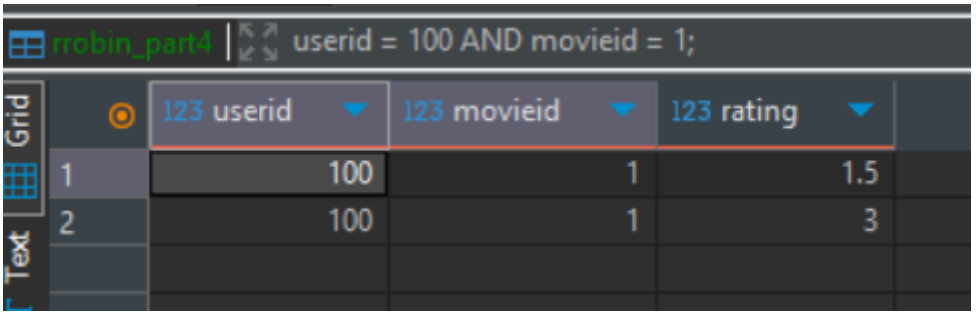


	userid	movieid	rating
1	1	292	5
2	1	362	5
3	1	466	5



	userid	movieid	rating
1	1	316	5
2	1	364	5
3	1	480	5

+ Cùng với đó bộ test (userid, movieid, ratings) = (100, 1, 3) cũng đã được chèn thành công vào phân mảnh 4



rrobin_part4 | userid = 100 AND movieid = 1;

	userid	movieid	rating
1	100	1	1.5
2	100	1	3

PHÂN CHIA CÔNG VIỆC TRONG NHÓM

Họ và tên	Mã sinh viên	Công việc
Đặng Hữu Nghĩa	B22DCCN601	Viết báo cáo và cài đặt các hàm liên quan đến Range Partitions: phân mảnh, insert, test…
Trần Cảnh Hưng	B22DCCN421	Báo cáo cài đặt môi trường , thư viện và thực hiện xử lý các hàm liên quan đến load dữ liệu . Hoàn thiện báo cáo tổng.
Vũ Ngọc Hùng	B22DCCN373	Viết báo cáo và cài đặt các hàm liên quan đến Round Robin Partitions: phân mảnh, insert, test…