

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



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

Giảng viên: Kim Ngọc Bách

Sinh viên : Tạ Quang Chiến - B22DCCN109

Nguyễn Mạnh Cường - B22DCCN100

Nguyễn Mậu Phi Hùng - B22DCCN366

Nhóm lớp : 9

## Mục lục

I) Yêu cầu bài tập .....	2
1) Đề bài .....	2
2) Giải pháp .....	2
II) Thực hiện yêu cầu .....	2
1) Tải tệp rating.dat .....	2
2) Kết nối và tạo database .....	4
3) Cài đặt hàm LoadRatings() .....	6
4) Cài đặt hàm Range_Partition() .....	13
4.1) Thu thập thông tin test .....	13
4.2) Cài đặt hàm .....	16
4.3) Kiểm thử .....	19
5) Cài đặt hàm Range_Insert() .....	20
5.1) Thu thập thông tin test .....	20
5.2) Cài đặt hàm .....	21
5.3) Kiểm thử .....	23
6) Cài đặt hàm RoundRobin_Partition() .....	25
6.1) Thu thập thông tin test .....	25
6.2) Cài đặt hàm .....	26
6.3) Kiểm thử .....	29
7) Cài đặt hàm RoundRobin_Insert() .....	30
7.1) Thu thập thông tin test .....	30
7.2) Cài đặt hàm .....	32
7.3) Kiểm thử .....	33
III) Tổng kết .....	35

## **I) Yêu cầu bài tập**

### **1) 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 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.

### **2) Giải pháp**

a) Hệ quản trị cơ sở dữ liệu sử dụng

- Chọn PostgreSQL cho bài toán

b) Phương pháp

- Các bước cài đặt hàm sẽ được phân tích đánh giá qua các quá trình:

+ Đọc và hiểu cách thức mà code test hoạt động từ đó đưa ra giải pháp và ghi nhớ những điều cần chú ý

+ Cài đặt hàm phù hợp với yêu cầu ở bước đầu

+ Kết hợp hàm kiểm tra test và kiểm tra thủ công trên cơ sở dữ liệu để đảm bảo tính đúng đắn của hàm cài đặt

⇒ Các hàm sắp cài đặt dưới đây hầu hết sẽ trải qua các bước đã nêu trên

## **II) Thực hiện yêu cầu**

### **1) Tải tệp rating.dat**

- Xem xét file Assignment1Tester.py,

- File sau khi tải về có dạng như sau

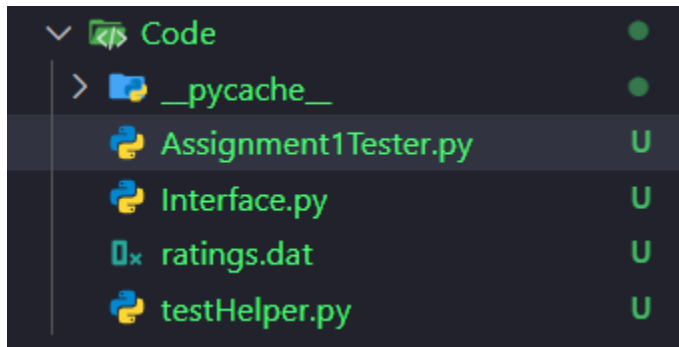
```
BTLCSDLPT > Code > ratings.dat
1 1::122::5:838985046
2 1::185::5:838983525
3 1::231::5:838983392
4 1::292::5:838983421
5 1::316::5:838983392
6 1::329::5:838983392
7 1::355::5:838984474
8 1::356::5:838983653
9 1::362::5:838984885
10 1::364::5:838983707
11 1::370::5:838984596
12 1::377::5:838983834
13 1::420::5:838983834
14 1::466::5:838984679
15 1::480::5:838983653
16 1::520::5:838984679
17 1::539::5:838984068
18 1::586::5:838984068
19 1::588::5:838983339
20 1::589::5:838983778
21 1::594::5:838984679
22 1::616::5:838984941
23 2::110::5:868245777
24 2::151::3:868246450
25 2::260::5:868244562
26 2::376::3:868245920
27 2::539::3:868246262
28 2::590::5:868245608
```

```
BTLCSDLPT > Code > ratings.dat
10000038 71567::1833::3::912649171
10000039 71567::1876::3::912580722
10000040 71567::1909::2::912580688
10000041 71567::1917::4::912580787
10000042 71567::1920::4::912578247
10000043 71567::1982::1::912580553
10000044 71567::1983::1::912580553
10000045 71567::1984::1::912580553
10000046 71567::1985::1::912580553
10000047 71567::1986::1::912580553
10000048 71567::2012::3::912580722
10000049 71567::2028::5::912580344
10000050 71567::2107::1::912580553
10000051 71567::2126::2::912649143
10000052 71567::2294::5::912577968
10000053 71567::2338::2::912578016
10000054 71567::2384::2::912578173
```

Số bản ghi lên tới con số 10000054 (10 triệu +54 bản)

## 2) Kết nối và tạo database

- Nhìn vào cấu trúc thư mục ta thấy được cấu trúc thư mục như sau



Thư mục chia làm 3 file python, trong đó file **Assignment1Tester.py** là file thực thi chính, file **testHelper.py** chứa code dùng để test cơ sở dữ liệu với các hàm được viết sẵn

Xem xét code file **Assignment1Tester.py**, cần điều chỉnh lại **INPUT\_FILE\_PATH**, và **ACTUAL\_ROWS\_IN\_INPUT\_FILE=1000054** cho phù hợp với số bản ghi trong file ratings

```
BTLCSDLPT > Code > Assignment1Tester.py > ...
1 #
2 # Tester for the assignment1
3 #
4 DATABASE_NAME = 'dds_assgn1'
5
6 # TODO: Change these as per your code
7 RATINGS_TABLE = 'ratings'
8 RANGE_TABLE_PREFIX = 'range_part'
9 RROBIN_TABLE_PREFIX = 'rrobin_part'
10 USER_ID_COLNAME = 'userid'
11 MOVIE_ID_COLNAME = 'movieid'
12 RATING_COLNAME = 'rating'
13 INPUT_FILE_PATH = r"E:\BTLCSDLPT\Code\ratings.dat"
14 ACTUAL_ROWS_IN_INPUT_FILE = 1000054 # Number of lines in the input file
15
16 import psycopg2
17 import traceback
18 import testHelper
19 import Interface as MyAssignment
20
21 if __name__ == '__main__':
22     try:
23         testHelper.createdb(DATABASE_NAME)
24
25         with testHelper.getopenconnection(dbname=DATABASE_NAME) as conn:
26             conn.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
27
28         testHelper.deleteAllPublicTables(conn)
```

- Các tham số được khởi tạo mặc định, main gọi **testHelper.createdb(DATABASE\_NAME)**

- Đi vào hàm **createdb** của **testHelper**:

```
def createdb(dbname):
    """
    We create a DB by connecting to the default user and database of Postgres
    The function first checks if an existing database exists for a given name, else creates it.
    :return:None
    """
    # 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()
```

➔ File test đã viết sẵn hàm kết nối với database, tạo database nếu chưa có

```
def getopenconnection(user='postgres', password='1234', dbname='postgres'):
    return psycopg2.connect("dbname='" + dbname + "' user='" + user + "' host='localhost' password='" + password + "'")
```

Hàm **getopenconnection()** kết nối mặc định đến database postgres với user postgres và password="1234"

```
DATABASE_NAME = 'dds_assgn1'

# TODO: Change these as per your code
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 = 'ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 1000000 # Number 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)

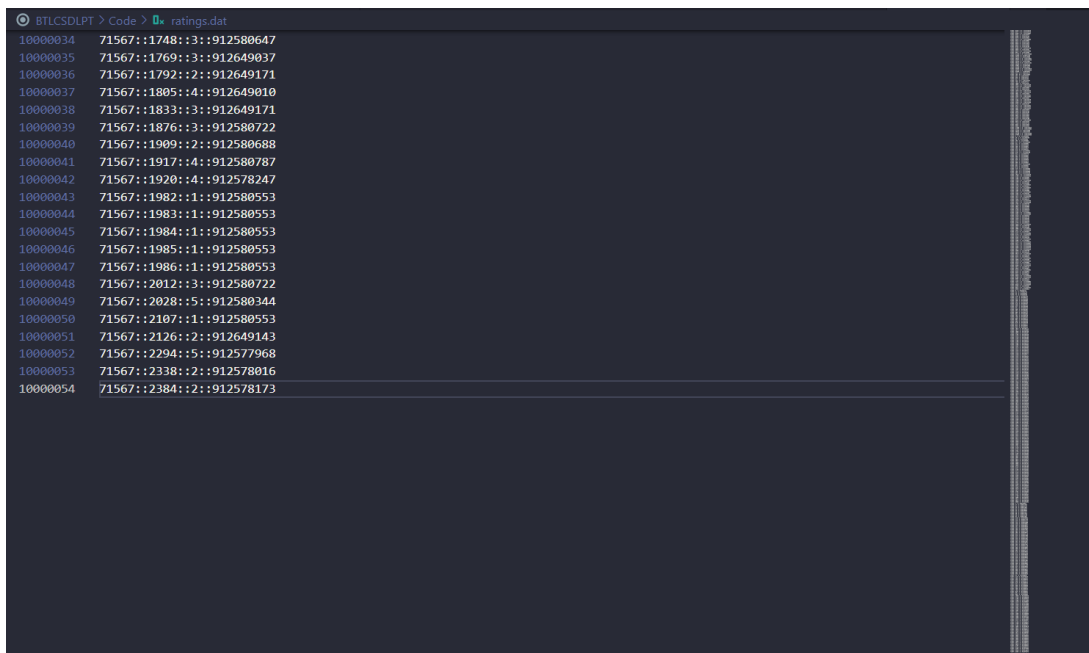
            testHelper.deleteAllPublicTables(conn)
```

- Sau khi kết nối với database thì thực hiện xóa các bảng 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()
```

### 3) Cài đặt hàm LoadRatings()

File **ratings.dat** sau khi được tải về có khoảng 10000054 (10 triệu +54 bản ghi)



```
8TLCSDLPT > Code > ratings.dat
10000034 71567::1748::3::912580647
10000035 71567::1769::3::912649037
10000036 71567::1792::2::912649171
10000037 71567::1805::4::912649010
10000038 71567::1833::3::912649171
10000039 71567::1876::3::912580722
10000040 71567::1909::2::912580688
10000041 71567::1917::4::912580787
10000042 71567::1920::4::912578247
10000043 71567::1982::1::912580553
10000044 71567::1983::1::912580553
10000045 71567::1984::1::912580553
10000046 71567::1985::1::912580553
10000047 71567::1986::1::912580553
10000048 71567::2012::3::912580722
10000049 71567::2028::5::912580344
10000050 71567::2107::1::912580553
10000051 71567::2126::2::912649143
10000052 71567::2294::5::912577968
10000053 71567::2338::2::912578016
10000054 71567::2384::2::912578173
```

- Từ file **Assignment1Tester.py**, ta thấy được để test hàm **LoadRatings()** cần thông qua câu lệnh **testHelper.testloadratings()**

```

def testloadratings(MyAssignment, ratingstablename, filepath, openconnection, rowsininpfile):
    """
    Tests the load ratings function
    :param ratingstablename: Argument for function to be tested
    :param filepath: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param rowsininpfile: Number of rows in the input file provided for assertion
    :return: Raises exception if any test fails
    """
    try:
        MyAssignment.loadratings(ratingstablename, filepath, openconnection)
        # Test 1: Count the number of rows inserted
        with openconnection.cursor() as cur:
            cur.execute('SELECT COUNT(*) from {}'.format(ratingstablename))
            count = int(cur.fetchone()[0])
            if count != rowsininpfile:
                raise Exception(
                    'Expected {} rows, but {} rows in \'{}\'' table'.format(rowsininpfile, count, ratingstablename))
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]

```

- Hàm **testloadratings()** nhận vào tham số là file bài tập(**MyAssignment**), tên bảng(**ratingstablename**), đường dẫn file dat(**filepath**), kết nối(**openconnection**) và số lượng bản ghi trong file dat(**rowsininpfile**)

- Hàm test sẽ chạy hàm **LoadRatings()** sau đó kiểm tra số lượng bản ghi trong bảng có bằng với số lượng bản ghi trong file dat(**rowsininpfile**) hay không, nếu không thì trả về kết quả false cùng với thông báo lỗi

=> Cần 1 hàm **LoadRatings()** phù hợp cho bài toán, ta có thể đề xuất 3 giải pháp

a) Chèn từng dòng, đọc từng dòng trong file ratings.dat rồi insert lần lượt vào trong bảng cơ sở dữ liệu



```

def loadratings(ratingtablename, ratingsfilepath, openconnection):
    cur = openconnection.cursor()
    # Tạo bảng nếu chưa có
    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {ratingtablename} (
            userid INTEGER,
            movieid INTEGER,
            rating FLOAT
        )
    """)
    openconnection.commit()
    inserted_rows = 0

    with open(ratingsfilepath, 'r') as file:
        for line_num, line in enumerate(file, 1):
            parts = line.strip().split(':')
            if len(parts) >= 3:
                try:
                    userid = int(parts[0])
                    movieid = int(parts[1])
                    rating = float(parts[2])
                    cur.execute(f"""
                        INSERT INTO {ratingtablename} (userid, movieid, rating)
                        VALUES (%s, %s, %s)
                    """, (userid, movieid, rating))
                    inserted_rows += 1
                    if inserted_rows % 1000 == 0:
                        openconnection.commit()
                except Exception as e:
                    print(f"⚠️ Error at line {line_num}: {e}")

    openconnection.commit()
    cur.close()
    print(f"✅ Inserted {inserted_rows} rows into '{ratingtablename}'")

```

- Đầu tiên phải tạo bảng trong database nếu chưa có

- Tiếp theo, trong đoạn code trên, từng dòng được xử lý rồi thực hiện insert vào cơ sở dữ liệu, cứ 1000 bản ghi thì sẽ commit 1 lần

=> Dễ gây lỗi khi chèn, có thể lỗi giữa chừng, đặc biệt là chạy cực lâu khi thực hiện từng lệnh sql

=> Không sử dụng

b) Lưu vào file tạm rồi dùng copy\_from của psycopg2

```
def loadratings(ratingtablename, ratingsfilepath, openconnection):
    start_time = time.time()
    cur = openconnection.cursor()
    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {ratingtablename} (
            userid INTEGER,
            movieid INTEGER,
            rating FLOAT
        )
    """)

    import tempfile

    with tempfile.NamedTemporaryFile(mode='w+', delete=True) as tmpfile:
        with open(ratingsfilepath, 'r') as f:
            for line in f:
                parts = line.strip().split('::')
                if len(parts) >= 3:
                    userid, movieid, rating = parts[0], parts[1], parts[2]
                    tmpfile.write(f"{userid}:{movieid}:{rating}\n")
            tmpfile.flush()
            tmpfile.seek(0)

            cur.copy_from(tmpfile, ratingtablename, sep=':', columns=('userid', 'movieid', 'rating'))

    openconnection.commit()
    cur.close()

    end_time = time.time()
    print(f"Loaded data into '{ratingtablename}' in {end_time - start_time:.2f} seconds.")
```

- Đầu tiên phải tạo bảng trong database nếu chưa có
  - Tiếp theo, trong đoạn code trên, từng dòng được xử lý (loại bỏ bớt 1 ký tự "::") rồi ghi vào file tạm
- => Nhanh nhưng vẫn phải xử lý từng dòng trong file

Loaded data into 'ratings' in 13.80 seconds.

=> Có thể sử dụng

c) Lợi dụng **copy\_from** của **psycopg2** để tạo bảng với các cột không liên quan rồi xóa cột

- Trong hàm **copy\_from** của **psycopg2** nhận đầu vào chỉ là 1 ký tự, nhưng phần input đầu vào lại phân tách các trường bằng 2 dấu "::"

=> Giải pháp: tạo các cột phụ bên cạnh các cột chính, sau khi thêm tất cả các bản ghi vào cơ sở dữ liệu thì drop những cột thừa ra đó/

Ví dụ: đầu vào là 1::1::3::912580553

Khi dùng **copy\_from** với tham số **sep="::"** sẽ được phân tách thành [1,"1","3","912580553]

=> Tạo bảng với 7 cột với tên lần lượt là [userid,extra1,movieid,extra2,rating,extra3,timestamp] như hình dưới

```
cur.execute(f"""
    CREATE TABLE IF NOT EXISTS {ratingtablename} (
        userid INTEGER,
        extra1 CHAR,
        movieid INTEGER,
        extra2 CHAR,
        rating FLOAT,
        extra3 CHAR,
        timestamp BIGINT
    );
""")
```

Đọc file rating và dùng hàm **copy\_from**

Sau khi đọc xong sẽ drop các cột thừa

```
with open(ratingsfilepath, 'r') as infile:
    cur.copy_from(infile, ratingtablename, sep=':')

cur.execute(f"""
    ALTER TABLE {ratingtablename}
    DROP COLUMN extra1,
    DROP COLUMN extra2,
    DROP COLUMN extra3,
    DROP COLUMN timestamp;
""")
```

```

def loadratings(ratingtablename, ratingsfilepath, openconnection):
    start_time = time.time()
    con = openconnection
    cur = con.cursor()
    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {ratingtablename} (
            userid INTEGER,
            extra1 CHAR,
            movieid INTEGER,
            extra2 CHAR,
            rating FLOAT,
            extra3 CHAR,
            timestamp BIGINT
        );
    """)
    with open(ratingsfilepath, 'r') as infile:
        cur.copy_from(infile, ratingtablename, sep=':')

    cur.execute(f"""
        ALTER TABLE {ratingtablename}
        DROP COLUMN extra1,
        DROP COLUMN extra2,
        DROP COLUMN extra3,
        DROP COLUMN timestamp;
    """)

    con.commit()
    cur.close()

    end_time = time.time()
    print(f"Loaded data into '{ratingtablename}' in {end_time - start_time:.2f} seconds.")

```

=> Nhanh

Loaded data into 'ratings' in 7.07 seconds.

=> Sử dụng chiến lược này để load data vào cơ sở dữ liệu

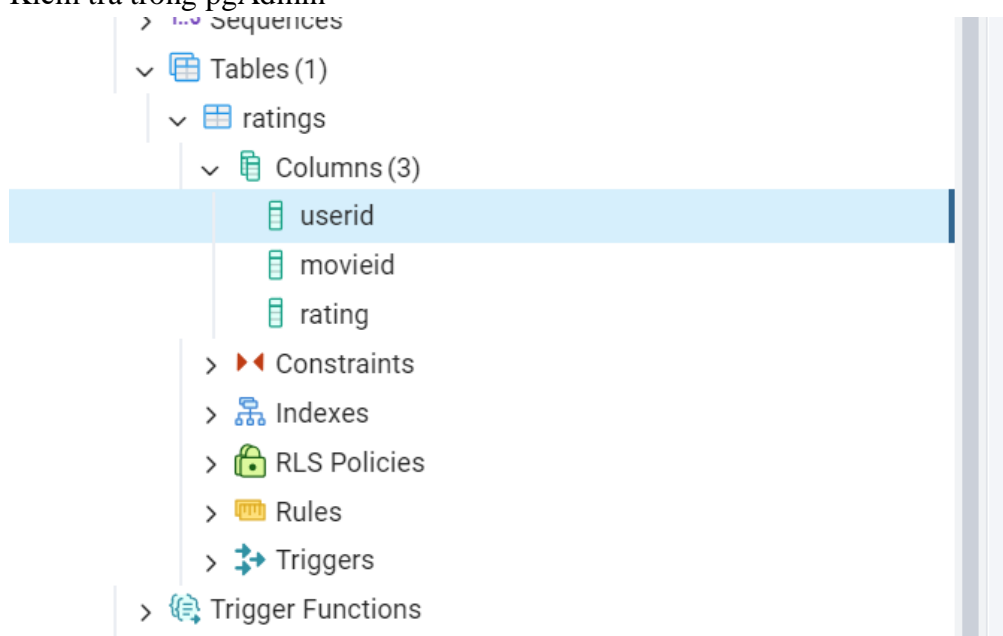
- Sau khi chạy test đã thì đã có thể thấy thông báo thành công

```

PS E:\BTLCSDLPT> & C:/Users/chien/AppData/Local/Programs/Python/Python312/python.exe e:/BTLCSDLPT/Code/Assignment1Tester.py
A database named "dds_assgn1" already exists
Loaded data into 'ratings' in 7.07 seconds.
loadratings function pass!

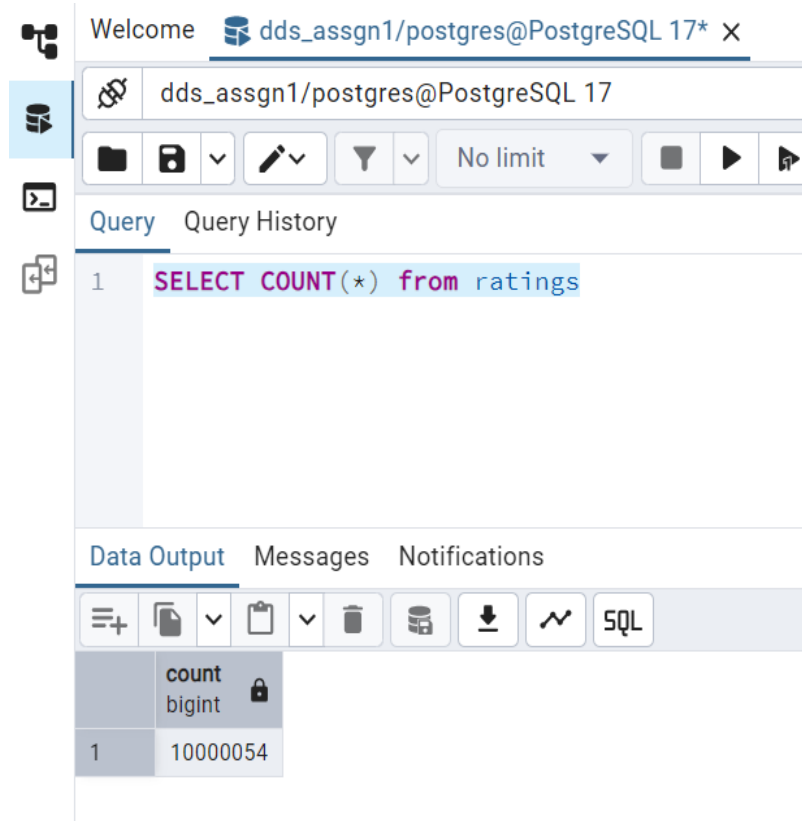
```

Kiểm tra trong pgAdmin



=> Đã có bảng ratings với đầy đủ các trường yêu cầu

Thực hiện truy vấn cũng ra kết quả tương tự



=> Số lượng bản ghi hoàn toàn trùng khớp với file ban đầu là 10 triệu +54 bản ghi

## 4) Cài đặt hàm Range\_Partition()

### 4.1) Thu thập thông tin test

- Sơ lược về **Range partition** (phân vùng theo khoảng giá trị) là phương pháp chia một bảng lớn thành nhiều bảng (hoặc phân vùng) con dựa trên **giá trị của một cột** sao cho mỗi phân vùng chứa các bản ghi có giá trị nằm trong một **khoảng liên tiếp** (range) đã định trước. Trong bài tập này, bảng con được phân vùng dựa trên thuộc tính rating trải dài trên miền giá trị 0 đến 5

- Trong file **Assignment1Tester.py**, hàm **testHelper.testrangeinsert()** được gọi để test **Range\_Partition**.

```
[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!")
```

Đi

vào file **testHelper.py** ta có hàm **testrangepartition()** nhận các tham số:

- *MyAssignment*: file bài tập
- *ratingtablename*: bảng ratings
- *n*: số phân mảnh
- *openconnection*: kết nối
- *partitionstartindex*: giá trị đầu tiên của phân chia bảng
- *ACTUAL\_ROWS\_IN\_INPUT\_FILE*: số dòng trong file input

```
def testrangepartition(MyAssignment, ratingtablename, n, openconnection, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    """
    Tests the range partition function for Completeness, Disjointness and Reconstruction
    :param ratingtablename: Argument for function to be tested
    :param n: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param partitionstartindex: Indicates how the table names are indexed. Do they start as rangepart1, 2 ... or rangepart0, 1, 2...
    :return: Raises exception if any test fails
    """

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

Hàm này gọi hàm **rangepartition** của file lời giải, sau đó test bằng 2 hàm **testrangeandrobinpartitioning()** và **testEachRangePartition**, ta cần đọc và kiểm tra những điều kiện cần để code đúng yêu cầu cho hàm **rangepartition()** trong file lời giải

a) Hàm **testrangeandrobinpartitioning()**

```

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. Expected {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(
                "Disjointness property of Partitioning failed. Expected {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(
                "Reconstruction property of Partitioning failed. Expected {0} rows after merging all tables, but found {1} rows".format(
                    ACTUAL_ROWS_IN_INPUT_FILE, count))

```

Hàm này gọi 2 hàm con:

- **checkpartitioncount()** : hàm này đếm số bảng có prefix được quy định, nếu số lượng bảng thực tế không khớp với số lượng bảng phân chia sẽ raise exception

```

# Helpers for Tester functions
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. Expected {0} table(s) but found {1} table(s)'.format(
            expectedpartitions,
            count))

```

- **totalrowsinallpartitions()**: kiểm tra và trả về tổng số lượng bản ghi trong từng bảng con sau khi phân chia (các bảng có prefix được quy định khi truyền vào hàm)

```

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

```

Trong hàm **testrangeandrobinpartitioning()** kiểm tra tổng số lượng bản ghi mà **totalrowsinallpartitions()** trả về có khớp với **ACTUAL\_ROWS\_IN\_INPUT\_FILE** hay không, nếu không thì sẽ raise exception tùy thuộc vào nó lớn hơn hay nhỏ hơn.

b) Hàm **testEachRangePartition()**

```
def testEachRangePartition(ratingtablename, n, openconnection, rangepartitiontableprefix):
    countList = getCounterangepartition(ratingtablename, 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]
            ))
```

Hàm này gọi hàm con **getCounterangepartition()**

```
def getCounterangepartition(ratingtablename, numberofpartitions, openconnection):
    """
    Get number of rows for each partition
    :param ratingtablename:
    :param numberofpartitions:
    :param openconnection:
    :return:
    """
    cur = openconnection.cursor()
    countList = []
    interval = 5.0 / numberofpartitions
    cur.execute("select count(*) from {0} where rating >= {1} and rating <= {2}".format(ratingtablename, 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(ratingtablename,
                                                                                          lowerbound,
                                                                                          lowerbound + interval))
        lowerbound += interval
        countList.append(int(cur.fetchone()[0]))

    cur.close()
    return countList
```

Khi xem xét ta thấy được điểm cần lưu ý và suy ra được từ phần test:

+ Bảng con đầu tiên sẽ bắt đầu được chia theo điểm từ 0 đến nhỏ hơn hoặc bằng interval (hiệu lớn nhất giữa 2 điểm trong 1 bảng con). Biến interval sẽ là điểm xuất phát cho phần điểm nhỏ nhất của bảng con sau

+ Bắt đầu từ bảng con thứ 2, điểm sẽ giao động từ điểm lớn hơn điểm cao nhất của bảng con trước đến nhỏ hơn hoặc bằng điểm cao nhất của bảng con trước cộng với interval

⇒ Cần lưu ý phần trên khi cài đặt hàm **Range\_Patition()**

Hàm **getCounterangepartition()** trả về 1 list lần lượt số bản ghi nên có trong các bảng con sau khi chia bảng, sau đó so sánh với số bản ghi có trong các bảng con (có prefix đúng yêu cầu) chỉ cần lệch 1 bảng thì sẽ raise exception

Kết luận: Hàm test chỉ đúng khi các hàm test không raise exception nào, lưu ý khi chia bảng sẽ được áp dụng để cài đặt hàm **Range\_Patition()**



## 4.2) Cài đặt hàm

- Bắt đầu với các giá trị **RANGE\_TABLE\_PREFIX** = 'range\_part' theo yêu cầu đề bài
- **Range\_size** sẽ được tính bằng cách lấy 5 chia số số bảng muốn chia sẽ ra được khoảng cách lớn nhất giữa điểm lớn nhất và nhỏ nhất trong 1 bảng con

```
def rangepartition(ratingtablename, numberofpartitions, openconnection):  
    cur = openconnection.cursor()  
    RANGE_TABLE_PREFIX = 'range_part'  
  
    range_size = 5.0 / numberofpartitions  
  
    create_metadata_table(openconnection)
```

- Đề bài cho phép sử dụng bảng meta-data, nên ta có tạo bảng tên là **partition\_metadata** để thuận tiện cho các phần sau này sẽ giải thích sau

```
def create_metadata_table(openconnection):  
    cur = openconnection.cursor()  
    cur.execute("""  
        CREATE TABLE IF NOT EXISTS partition_metadata (  
            partition_type VARCHAR(20),  
            partition_count INTEGER,  
            current_partition INTEGER,  
            range_size FLOAT  
        )  
    """)  
    openconnection.commit()  
    cur.close()
```

- Hàm **rangepartition()** gọi hàm tạo bảng **partition\_metadata** đồng thời chèn vào 1 bản ghi với thông tin như sau

- partition\_type: kiểu chia bảng hiện tại là 'range'
- partition\_count: số bảng muốn chia
- range\_size: kích thước range

- Trong hàm sẽ không chèn giá trị cho hàm **current\_partition** vì nó được thiết kế để dành riêng cho phần round robin partition của phần sau

```
def rangepartition(ratingtablename, numberofpartitions, openconnection):  
    cur = openconnection.cursor()  
    RANGE_TABLE_PREFIX = 'range_part'  
  
    range_size = 5.0 / numberofpartitions  
  
    create_metadata_table(openconnection)  
  
    cur.execute("""  
        INSERT INTO partition_metadata (partition_type, partition_count, range_size)  
        VALUES ('range', %s, %s)  
    """, (numberofpartitions, range_size))
```

- Xử lý từng bảng con :

- Chạy vòng lặp từ bảng con đầu tiên bắt đầu là 0 đến (số bảng muốn chia -1)
- Tìm điểm thấp nhất và cao nhất có thể chèn vào trong bảng,
  - điểm thấp nhất là  $\text{min\_rating} = \text{range\_size} * \text{số thứ tự bảng con}$
  - điểm cao nhất là  $\text{max\_rating} = \text{min\_rating} + \text{range\_size}$
- Đối với bảng con cuối cùng, điểm cao nhất có thể cho vào trong bảng phải là 5 vì phép chia và cộng số thực trong python có thể có sai số khiến cho phần  $\text{max\_rating}$  nhỏ hơn 5
- Tạo bảng con nếu chưa có: bằng Prefix theo yêu cầu đề bài + số thứ tự bảng con
- Tạo câu truy vấn insert các bản ghi thỏa mãn điều kiện từ bảng ratings (Bảng gốc) sang bảng con
- Riêng với bảng con đầu tiên, các bản ghi được lọc  $\text{min\_rating}$  (ở bảng con đầu có giá trị bằng 0)  $\leq \text{rating} \leq \text{max\_rating}$ , các bảng con sau đó sẽ được lọc theo  $\text{min\_rating} < \text{rating} \leq \text{max\_rating}$
- commit và kết thúc hàm

```
for i in range(numberofpartitions):
    min_rating = i * range_size
    max_rating = min_rating + range_size
    if i==numberofpartitions-1: max_rating=5

    table_name = f"{RANGE_TABLE_PREFIX}{i}"
    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {table_name} (
            userid INTEGER,
            movieid INTEGER,
            rating FLOAT
        )
    """)

    if i == 0:
        cur.execute(f"""
            INSERT INTO {table_name}
            SELECT userid, movieid, rating
            FROM {ratingtablename}
            WHERE rating >= {min_rating} AND rating <= {max_rating}
        """)
    else:
        cur.execute(f"""
            INSERT INTO {table_name}
            SELECT userid, movieid, rating
            FROM {ratingtablename}
            WHERE rating > {min_rating} AND rating <= {max_rating}
        """)
```

- **Nhưng:** dễ thấy, bảng con và bảng cha có cấu trúc bảng giống nhau chỉ khác nhau tên bảng

=> có thể gộp bước tạo bảng con và insert thành 1 câu truy vấn, sử dụng kỹ thuật CTAS(Create Table As Select). Code sẽ trở thành như sau

```
for i in range(numberofpartitions):
    min_rating = i * range_size
    max_rating = 5.0 if i == numberOfpartitions - 1 else min_rating + range_size

    table_name = f"{RANGE_TABLE_PREFIX}{i}"

    condition = f"rating >= {min_rating} AND rating <= {max_rating}" if i == 0 \
        else f"rating > {min_rating} AND rating <= {max_rating}"

    cur.execute(f"""
        CREATE TABLE {table_name} AS
        SELECT userid, movieid, rating
        FROM {ratingtablename}
        WHERE {condition}
    """)

openconnection.commit()
cur.close()
```

Toàn bộ hàm

```
def rangepartition(ratingtablename, numberOfpartitions, openconnection):
    import time
    start_time = time.time()
    cur = openconnection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    range_size = 5.0 / numberOfpartitions

    create_metadata_table(openconnection)
    cur.execute("""
        INSERT INTO partition_metadata (partition_type, partition_count, range_size)
        VALUES ('range', %s, %s)
    """, (numberOfpartitions, range_size))

    for i in range(numberofpartitions):
        min_rating = i * range_size
        max_rating = 5.0 if i == numberOfpartitions - 1 else min_rating + range_size

        table_name = f"{RANGE_TABLE_PREFIX}{i}"

        condition = f"rating >= {min_rating} AND rating <= {max_rating}" if i == 0 \
            else f"rating > {min_rating} AND rating <= {max_rating}"

        cur.execute(f"""
            CREATE TABLE {table_name} AS
            SELECT userid, movieid, rating
            FROM {ratingtablename}
            WHERE {condition}
        """)

    openconnection.commit()
    cur.close()
    elapsed_time = time.time() - start_time
    print(f"rangepartition executed in {elapsed_time:.2f} seconds")
```

### 4.3) Kiểm thử

- Từ bảng ratings gốc và số phân mảnh là 5 các bảng đã được phân chia như hình dưới

Tables (7)

- > partition\_metadata
- > range\_part0
- > range\_part1
- > range\_part2
- > range\_part3
- > range\_part4
- > ratings

Query Query History

```
26
27 SELECT COUNT(*)
28 FROM range_part0
29 WHERE rating > 1
30
```

Data Output Messages Notifications

	count bigint
1	0

Query Query History

```
26
27 SELECT COUNT(*)
28 FROM range_part1
29 WHERE rating <=1 OR rating > 2
30
```

Data Output Messages Notifications

	count bigint
1	0

Query Query History

```
26
27 SELECT COUNT(*)
28 FROM range_part2
29 WHERE rating <=2 OR rating > 3
30
```

Data Output Messages Notifications

	count bigint
1	0

Query Query History

```
26
27 SELECT COUNT(*)
28 FROM range_part3
29 WHERE rating <=3 OR rating > 4
30
```

Data Output Messages Notifications

	count bigint
1	0

Query Query History

```
26
27 SELECT COUNT(*)
28 FROM range_part4
29 WHERE rating <=4 OR rating > 5
30
```

Data Output Messages Notifications

	count bigint
1	0

Query Query History

```
31 SELECT * FROM partition_metadata
32
33
```

Data Output Messages Notifications

	partition_type character varying (20)	partition_count integer	current_partition integer	range_size double precision
1	range	5	[null]	1

- Hoặc thử với số phân mảnh là 2

The left screenshot shows a database interface with a 'Tables (4)' list containing 'partition\_metadata', 'range\_part0', 'range\_part1', and 'ratings'. Below, a query window shows a SQL query: `SELECT COUNT(*) FROM range_part0 WHERE rating > 2.5`. The 'Data Output' tab shows a single row with 'count' 0.

The right screenshot shows a similar interface but with a query for 'range\_part1': `SELECT COUNT(*) FROM range_part1 WHERE rating <= 2.5`. The 'Data Output' tab also shows a single row with 'count' 0.

- Các truy vấn trên hình kết hợp với các test có trong phần hàm test có thể cho thấy được thuật toán phân mảnh trên là đúng đắn

## 5) Cài đặt hàm Range\_Insert()

### 5.1) Thu thập thông tin test

- Hàm `testrangeinsert()` là hàm kiểm tra cho phần cài đặt **Range\_Insert()**

```
def testrangeinsert(MyAssignment, ratingtablename, userid, itemid, rating, openconnection, expectedtableindex):
    """
    Tests the range insert function by checking whether the tuple is inserted in he Expected table you provide
    :param ratingtablename: Argument for function to be tested
    :param userid: Argument for function to be tested
    :param itemid: Argument for function to be tested
    :param rating: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param expectedtableindex: The expected table to which the record has to be saved
    :return:Raises exception if any test fails
    """
    try:
        expectedtablename = RANGE_TABLE_PREFIX + expectedtableindex
        MyAssignment.rangeinsert(ratingtablename, 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]
```

- Trong hàm này **testrangeandrobinpartitioning()** được dùng để kiểm tra xem bản ghi mới được thêm vào có tồn tại trong bảng con hay không

```
def testrangeandrobininsert(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
```

- Nếu không tồn tại trong bảng con thì raise exception

=> Phải cài đặt hàm với đầu vào là 1 bản ghi mới và nó phải được điền vào đúng bảng con

## 5.2) Cài đặt hàm

- Trong phần interface có sẵn có thể thấy được số bảng được phân chia được đếm thông

```
def rangeinsert(ratingtablename, userid, itemid, rating, openconnection):
    """
    Function to insert a new row into the main table and specific partition based on range rating.
    """
    con = openconnection
    cur = con.cursor()
    RANGE_TABLE_PREFIX = 'range_part'
    numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
```

=> Có thể thấy được số bảng được phân chia được đếm thông qua hàm **count\_partitions()**

```
def count_partitions(prefix, openconnection):
    """
    Function to count the number of tables which have the @prefix in their name somewhere.
    """
    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
```

Nhưng ở phần **rangepartition()** trước đó, chúng ta đã tạo 1 bảng **partition\_metadata** để lưu trữ thông tin của các phiên bản phân chia cơ sở dữ liệu, từ đó ta có thể dễ dàng lấy được số bảng cùng với đó là **range\_size** của nó mà không cần tính toán lại

Vì vậy hàm **rangeinsert()** của chúng ta sẽ như sau:

```
def rangeinsert(ratingtablename, userid, itemid, rating, openconnection):
    cur = openconnection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    cur.execute("""
        SELECT partition_count, range_size
        FROM partition_metadata
        WHERE partition_type = 'range'
    """)
    partition_info = cur.fetchone()
    if not partition_info:
        raise Exception("No range partitioning information found")

    numberofpartitions, range_size = partition_info

    partition_num = int(rating / range_size)

    if rating % range_size == 0 and partition_num > 0:
        partition_num -= 1
```

- Lấy **range\_size** từ metadata, tính toán bảng con mà bản ghi sẽ được chèn vào bằng cách lấy điểm cho bộ phim của bản ghi đó chia cho **range\_size** và chỉ lấy phần nguyên
- Lưu ý với phép tính chia hết, ta phải giảm 1 bậc số thứ tự của bảng con sau khi chia vì ở phần **rangepartition** trước đó đã để điều kiện **min\_rating < rating <= max\_rating** (trong đó rating là điểm của bản ghi mới định cho vào)
- Sau khi tính toán bảng mà bản ghi sẽ được thêm vào thì tất nhiên phải thêm bản ghi đó vào bảng gốc trước rồi mới thêm nó vào bảng con sau, cuối cùng là commit, kết thúc hàm

```
cur.execute(f"""
    INSERT INTO {ratingtablename} (userid, movieid, rating)
    VALUES (%s, %s, %s)
""", (userid, itemid, rating))

partition_table = f"{RANGE_TABLE_PREFIX}{partition_num}"
cur.execute(f"""
    INSERT INTO {partition_table} (userid, movieid, rating)
    VALUES (%s, %s, %s)
""", (userid, itemid, rating))

openconnection.commit()
cur.close()
```

=> Toàn bộ hàm như sau

```
def rangeinsert(ratingtablename, userid, itemid, rating, openconnection):
    cur = openconnection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'

    cur.execute("""
        SELECT partition_count, range_size
        FROM partition_metadata
        WHERE partition_type = 'range'
    """)
    partition_info = cur.fetchone()
    if not partition_info:
        raise Exception("No range partitioning information found")

    numberofpartitions, range_size = partition_info

    partition_num = int(rating / range_size)

    if rating % range_size == 0 and partition_num > 0:
        partition_num -= 1

    cur.execute(f"""
        INSERT INTO {ratingtablename} (userid, movieid, rating)
        VALUES (%s, %s, %s)
    """, (userid, itemid, rating))

    partition_table = f"{RANGE_TABLE_PREFIX}{partition_num}"
    cur.execute(f"""
        INSERT INTO {partition_table} (userid, movieid, rating)
        VALUES (%s, %s, %s)
    """, (userid, itemid, rating))

    openconnection.commit()
    cur.close()
```

### 5.3) Kiểm thử

- Với số lượng bảng phân chia là 5 vậy range\_size là 1:

```
[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!")

# 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, 2.5, conn, '2')
#[result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 3, conn, '2')
#[result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 0, conn, '0')
[result, e] = testHelper.testrangeinsert(MyAssignment, RATINGS_TABLE, 100, 2, 5, conn, '4')
if result:
    print("rangeinsert function pass!")
else:
    print("rangeinsert function fail!")
```

+ range\_part0: từ  $0 \leq \text{rating} \leq 1$

+ range\_part1: từ  $1 < \text{rating} \leq 2$

+ range\_part2: từ  $2 < \text{rating} \leq 3$

+ range\_part3: từ  $3 < \text{rating} \leq 4$



+ range\_part4: từ  $4 < \text{rating} \leq 5$

- Vậy với rating 2.5, 3, 0, 5 bảng mong muốn sẽ vào các bảng con có thứ tự 2, 2, 0, 4

+Rating=2.5 tìm trong bảng range\_part2 đã ra đúng kết quả

```
Query Query History
1 SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2 AND rating = 2.5
```

Data Output Messages Notifications

	userid integer	movieid integer	rating double precision
1	100	2	2.5

+Rating=3 tìm trong bảng range\_part2 đã ra đúng kết quả

```
Query Query History
1 SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2 AND rating = 3
```

Data Output Messages Notifications

Icons: SQL

	userid	movieid	rating
	integer	integer	double precision
1	100	2	3

+Rating=0 tìm trong bảng range\_part0 đã ra đúng kết quả

Query

Query History

1SELECT \* FROM range\_part0 WHERE userid = 100 AND movieid = 2 AND rating = 0

Data Output

Messages

Notifications

SQL

	userid integer	movieid integer	rating double precision
1	100	2	0

+Rating=5 tìm trong bảng range\_part4 đã ra đúng kết quả

Query

Query History

1

SELECT \* FROM range\_part4 WHERE userid = 100 AND movieid = 2 AND rating = 5;

Data Output

Messages

Notifications

SQL

	userid Integer	movieid integer	rating double precision
1	100	2	5

Và điều chắc chắn rằng bảng con khác (các bảng từ 0 đến 3) sẽ không chứa bản ghi này

The screenshot shows a SQL query editor with a 'Query' tab selected. The query is as follows:

```
1 SELECT * FROM range_part0 WHERE userid = 100 AND movieid = 2 AND rating = 5
2 UNION ALL
3 SELECT * FROM range_part1 WHERE userid = 100 AND movieid = 2 AND rating = 5
4 UNION ALL
5 SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2 AND rating = 5
6 UNION ALL
7 SELECT * FROM range_part3 WHERE userid = 100 AND movieid = 2 AND rating = 5
```

Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with three columns: 'userid' (integer), 'movieid' (integer), and 'rating' (double precision). The table is currently empty.

Tương tự với các giá trị còn lại

## 6) Cài đặt hàm RoundRobin\_Partition()

### 6.1) Thu thập thông tin test

- Sơ lược về Round Robin Partition (phân vùng vòng tròn): là một kỹ thuật chia bảng lớn thành nhiều bảng con (partition) bằng cách phân phối các bản ghi theo thứ tự luân phiên, không dựa vào giá trị cụ thể của bất kỳ cột nào.

- Hàm **testroundrobinpartition()** có logic như sau

```
def testroundrobinpartition(MyAssignment, ratingtablename, numberofpartitions, openconnection,
                           partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
    """
    Tests the round robin partitioning for Completeness, Disjointness and Reconstruction
    :param ratingtablename: Argument for function to be tested
    :param numberofpartitions: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param robinpartitiontableprefix: This function assumes that you tables are named in an order. Eg: robinpart1, robinpart2...
    :return: Raises exception if any test fails
    """
    try:
        MyAssignment.roundrobinpartition(ratingtablename, numberofpartitions, openconnection)
        testrangeandrobinpartitioning(numberofpartitions, openconnection, RROBIN_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
        testEachRoundrobinPartition(ratingtablename, numberofpartitions, openconnection, RROBIN_TABLE_PREFIX)
    except Exception as e:
        traceback.print_exc()
        return [False, e]
    return [True, None]
```

- Phần **testrangeandrobinpartitioning()** đã được nói ở phần a) của phần 4.1), cần xem xét xử lý, kiểm tra phần test của hàm **testEachRoundrobinPartition()**

```
def testEachRoundRobinPartition(ratingtablename, n, openconnection, roundrobinpartitiontableprefix):
    countList = getCountRoundRobinPartition(ratingtablename, 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]
            ))
```

- Hàm **testEachRoundRobinPartition()** gọi hàm **getCountRoundRobinPartition()**

```
def getCountRoundRobinPartition(ratingtablename, numberofpartitions, openconnection):
    """
    Get number of rows for each partition
    :param ratingtablename:
    :param numberofpartitions:
    :param openconnection:
    :return:
    """
    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(
                ratingtablename, numberofpartitions, i))
        countList.append(int(cur.fetchone()[0]))
    cur.close()
    return countList
```

- Hàm trả về 1 danh sách số dòng mà các bảng con nên có, số thứ tự dòng phải trừ đi 1 vì số thứ tự của bảng con bắt đầu từ 0, nếu chỉ xét về mặt số lượng bản ghi thực chất chỉ là phân chia đồng đều số lượng bản ghi vào các bảng con sao cho đồng đều hết mức có thể (chênh lệch số bản ghi lớn nhất giữa các bảng là 1 hoặc 0).

- Sau khi có danh sách số lượng bản ghi mong muốn, hàm **testEachRoundRobinPartition()** so sánh với số lượng bản ghi thực tế có trong bảng con, nếu có sai khác thì raise exception

=> Cài đặt hàm chỉ đúng khi hàm test không raise exception

## 6.2) Cài đặt hàm

- Định nghĩa RROBIN\_TABLE\_PREFIX = 'rrobin\_part'

```
def roundrobinpartition(ratingtablename, numberofpartitions, openconnection):
    cur = openconnection.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    create_metadata_table(openconnection)

    cur.execute("""
        INSERT INTO partition_metadata (partition_type, partition_count, current_partition)
        VALUES ('roundrobin', %s, 0)
    """, (numberofpartitions,))

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

- Gọi hàm **create\_metadata\_table()** (đã nói ở phần rangepartition) để tạo bảng partition\_metadata
- Trước tiên phải thêm metadata cho phần chia bảng theo round robin:

- partition\_type: là 'roundrobin'
- partition\_count: số lượng phân mảnh tùy thuộc vào đầu vào
- current\_partition: đặt là 0, sẽ được thay đổi sau này mỗi lần insert

Tương tự với kiểu **partition\_type** là 'range' không init trường **current\_partition**, với bản ghi có **partition\_type** là 'roundrobin' trường **range\_size** sẽ không được init vì đó là trường dành riêng cho kiểu phân chia bảng bằng 'range'

- Tạo các bảng con có prefix là rrobin\_part + số thứ tự của phân mảnh bắt đầu từ 0

```
for i in range(numberofpartitions):
    table_name = f"{RROBIN_TABLE_PREFIX}{i}"
    cur.execute(f"""
        INSERT INTO {table_name}
        SELECT userid, movieid, rating
        FROM (
            SELECT userid, movieid, rating,
                   ROW_NUMBER() OVER (ORDER BY userid, movieid) - 1 as row_num
            FROM {ratingtablename}
        ) numbered_rows
        WHERE row_num % {numberofpartitions} = {i}
    """)

    openconnection.commit()
    cur.close()
```

- Duyệt qua từng phân mảnh con, thực hiện truy vấn bảng gốc và đánh số từng dòng trong bản ghi
- ROW\_NUMBER() OVER (ORDER BY userid, movieid) đảm bảo mỗi lần thực hiện đánh số dòng được đồng nhất vì đã được sắp xếp theo userid và movieid, nếu không sắp xếp (không ORDER), mỗi bản ghi sẽ có số thứ tự dòng khác nhau trong mỗi lần truy xuất (vì truy vấn con

SELECT userid, movieid, rating,

ROW\_NUMBER() OVER () - 1 as row\_num

FROM {ratingtablename}

được gọi lại nhiều lần bằng với số bảng con )

- Chỉ chèn những bản ghi có row\_num chia lấy dư cho số bảng con bằng với số thứ tự bảng con(bắt đầu từ 0)

**Nhưng:** mỗi lần 1 bảng con lại phải sắp đánh số cho bảng gốc theo thứ tự sắp xếp, nếu có 5 phân mảnh thì phải sắp xếp + đánh số 5 lần, nếu phân mảnh sẽ làm bước này lặp lại nhiều lần vì việc sắp xếp và đánh số đảm bảo việc 1 bản ghi không có 2 số thứ tự khác nhau trong các lần đánh số ở mỗi vòng lặp gây ra việc chèn thiếu.

=> Nên tạo bảng đã đánh số từ bảng gốc rồi chèn các bản ghi vào bảng con từ bảng đã đánh số đó (tránh được việc sắp xếp và đánh số nhiều lần) kết hợp với kỹ thuật CTAS. Như vậy code sẽ như sau

```
def roundrobinpartition(ratingtablename, numberofpartitions, openconnection):
    import time
    start_time = time.time()
    cur = openconnection.cursor()
    RROBIN_TABLE_PREFIX = 'rrobin_part'

    create_metadata_table(openconnection)

    cur.execute(f"""
        CREATE TEMP TABLE temp_numbered AS
        SELECT
            userid,
            movieid,
            rating,
            ROW_NUMBER() OVER () - 1 AS row_num
        FROM {ratingtablename};
    """)

    for i in range(numberofpartitions):
        table_name = f"{RROBIN_TABLE_PREFIX}{i}"
        cur.execute(f"""
            CREATE TABLE {table_name} AS
            SELECT userid, movieid, rating
            FROM temp_numbered
            WHERE row_num % {numberofpartitions} = {i};
        """)

    cur.execute(f"""
        INSERT INTO partition_metadata (partition_type, partition_count, current_partition)
        SELECT 'roundrobin', %s, COUNT(*) %% %s FROM temp_numbered
        """, (numberofpartitions, numberofpartitions))

    openconnection.commit()
    cur.close()
```

- Bảng **temp\_numbered** sẽ chứa toàn bộ bản ghi của bảng gốc cùng với số thứ tự hàng với mỗi bản ghi
  - Lặp n lần (số phân mảnh cần tạo) tạo bảng và chèn vào theo đúng yêu cầu
  - Thêm bản ghi vào bảng **partition\_metadata** trong 1 câu truy vấn thay vì tạo 1 bản ghi rồi cập nhật như code cũ
- => Giải quyết được vấn đề đặt ra đồng thời tối ưu việc chia bảng khi mà không phải sắp xếp đánh số nhiều lần trên bảng gốc

### 6.3) Kiểm thử

The screenshot shows a SQL query editor with a query window and a data output window. The query is as follows:

```

1 SELECT * FROM ratings
2 EXCEPT
3 (SELECT * FROM rrobin_part0
4 UNION ALL
5 SELECT * FROM rrobin_part1
6 UNION ALL
7 SELECT * FROM rrobin_part2
8 UNION ALL
9 SELECT * FROM rrobin_part3
10 UNION ALL
11 SELECT * FROM rrobin_part4)
12

```

The data output window shows the following columns:

userid	movieid	rating
integer	integer	double precision

The screenshot shows a SQL query editor with a query window and a data output window. The query is as follows:

```

5 (SELECT * FROM rrobin_part0
6 UNION ALL
7 SELECT * FROM rrobin_part1
8 UNION ALL
9 SELECT * FROM rrobin_part2
10 UNION ALL
11 SELECT * FROM rrobin_part3
12 UNION ALL
13 SELECT * FROM rrobin_part4)
14 EXCEPT
15 SELECT * FROM ratings
16

```

The data output window shows the following columns:

userid	movieid	rating
integer	integer	double precision

- Thực hiện kiểm tra những bản ghi có trong bảng gốc nhưng không có trong những phân mảnh gộp lại và ngược lại đều cho ra kết quả rỗng (không có dòng hay bản ghi nào thỏa mãn)

No limit

Query

Query History

14

15

16

17

18

19

20

21

22

23

24

SELECT 'part0',COUNT(\*) FROM rrobin\_part0

UNION ALL

SELECT 'part1',COUNT(\*) FROM rrobin\_part1

UNION ALL

SELECT 'part2',COUNT(\*) FROM rrobin\_part2

UNION ALL

SELECT 'part3',COUNT(\*) FROM rrobin\_part3

UNION ALL

SELECT 'part4',COUNT(\*) FROM rrobin\_part4

Data Output

Messages

Notifications

SQL

?column?

text

count

bigint

1

part4

2000010

2

part0

2000011

3

part1

2000011

4

part3

2000011

5

part2

2000011

- Các phân mảnh con cũng có đúng với bản ghi mà nó nên có

=> Phân mảnh có thể coi như thành công

## 7) Cài đặt hàm RoundRobin\_Insert()

### 7.1) Thu thập thông tin test

- Hàm **testroundrobininsert()** được sử dụng để test phân chèn bản ghi vào phân mảnh round robin

```
def testroundrobininsert(MyAssignment, ratingtablename, userid, itemid, rating, openconnection, expectedtableindex):
    """
    Tests the roundrobin insert function by checking whether the tuple is inserted in the Expected table you provide
    :param ratingtablename: Argument for function to be tested
    :param userid: Argument for function to be tested
    :param itemid: Argument for function to be tested
    :param rating: Argument for function to be tested
    :param openconnection: Argument for function to be tested
    :param expectedtableindex: The expected table to which the record has to be saved
    :return: Raises exception if any test fails
    """
    try:
        expectedtablename = RROBIN_TABLE_PREFIX + expectedtableindex
        MyAssignment.roundrobininsert(ratingtablename, userid, itemid, rating, openconnection)
        if not teststrangerrobininsert(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 trên lại gọi teststrangerrobininsert() chỉ đơn giản là kiểm tra xem bản ghi mới chèn có nằm trong bảng mong muốn không, nếu không sẽ raise exception

```
def testroundrobininsert(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
```

- Yêu cầu test không có công thức chuẩn để nhận biết được rằng nên chèn bản ghi này vào bảng nào tiếp theo, đầu vào ví dụ chỉ là

```
# ALERT:: Change the partition index according to your 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, '1')
# [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
if result:
```

với số phân mảnh là 5 khi đã có sẵn 20 bản ghi trong bảng gốc

- Thì ra thuật toán chạy sẽ thêm lần lượt vào các bảng con theo số thứ tự bảng tăng dần, nhưng phải bắt đầu từ bảng sau bảng trước đó mới có bản ghi thêm vào. Nghĩa là nếu bản ghi trước đó được chèn vào phân mảnh 2 thì bản ghi tiếp theo sẽ phải được chèn vào phân mảnh 3

=> Cần có phương án để tìm phân mảnh tiếp theo, đề xuất 3 giải pháp:

+ Đếm số lượng bản ghi gốc trong bảng chính để tính phân mảnh tiếp theo được chèn vào

- Nguy cơ khi mà xóa 1 bản ghi trong bảng gốc, khi có bản ghi mới, hệ thống tính tổng bản ghi ở bảng gốc rồi chia modul cho số bảng thì bảng con được chọn để chèn vào lại trùng với bảng con được chọn của bản ghi trước đó => không đúng logic.
- Nguy cơ khi việc chèn là liên tục, xảy ra cùng 1 thời điểm, 2 lần chèn xảy ra cùng 1 thời điểm sẽ đọc bảng gốc cùng 1 thời điểm cho ra cùng 1 kết quả số lượng bản ghi trong bảng gốc, khi đó, 2 bản ghi này sẽ cùng 1 lúc được chèn vào cùng 1 bảng con => không hợp lệ

+ Tính số lượng bản ghi trong từng bảng con, chọn bảng con có số lượng bản ghi thấp nhất và có số thứ tự bảng thấp nhất để chèn: Nguy cơ tương tự với phương án đầu tiên khi xóa 1 bản ghi ở 1 bảng con nào đó hoặc đồng thời đọc số lượng bản ghi ở các bảng con ở cùng 1 thời điểm khi có 2 yêu cầu chèn cùng lúc

+ Dùng metadata: như đã trình bày ở các bước cài đặt hàm trước đó, bảng partition\_metadata lưu trữ bảng ghi số lượng phân mảnh, riêng với kiểu round robin đã có 1 trường current\_partition dùng để lưu trữ bảng con sẽ được chèn tiếp theo nếu có bản ghi mới

=> Sử dụng phương án 3 để giải quyết cho phần cài đặt hàm roundrobininsert



## 7.2) Cài đặt hàm

- Hàm thiết lập những tham số đầu vào RROBIN\_TABLE\_PREFIX = 'rrobin\_part' theo yêu cầu
- Tất nhiên cần phải thêm bản ghi vào trong bảng gốc trước

```
def roundrobininsert(ratingtablename, userid, itemid, rating, openconnection):  
    cur = openconnection.cursor()  
    RROBIN_TABLE_PREFIX = 'rrobin_part'  
  
    cur.execute(f"""  
        INSERT INTO {ratingtablename} (userid, movieid, rating)  
        VALUES (%s, %s, %s)  
        """, (userid, itemid, rating))
```

- Như đã trình bày ở bước phân tích, việc chèn bản ghi theo chiến lược round robin cần sử dụng bảng partition\_metadata để lấy thông tin index của bảng sẽ được chọn để chèn thêm bản ghi như mới

```
cur.execute("""  
    UPDATE partition_metadata  
    SET current_partition = (current_partition + 1) % partition_count  
    WHERE partition_type = 'roundrobin'  
    RETURNING ((current_partition - 1 + partition_count) % partition_count) AS old_partition,  
              partition_count  
    """)  
old_partition, numberofpartitions = cur.fetchone()
```

Để đảm bảo không có lần gọi hàm chèn cùng 1 lúc, ta sẽ không select rồi update mà thay vào đó sẽ Update và Returning, điều này đảm bảo mỗi lần gọi hàm chèn sẽ lấy được current\_partition khác nhau

- current\_partition chính là index của bảng được chọn để chèn bản ghi hiện tại vào, thực hiện insert vào bảng phân mảnh tương ứng qua prefix+index

```
)  
old_partition, numberofpartitions = cur.fetchone()  
  
partition_table = f"{RROBIN_TABLE_PREFIX}{old_partition}"  
cur.execute(f"""  
    INSERT INTO {partition_table} (userid, movieid, rating)  
    VALUES (%s, %s, %s)  
    """, (userid, itemid, rating))
```

- Toàn bộ hàm

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

    cur.execute(f"""
        INSERT INTO {ratingtablename} (userid, movieid, rating)
        VALUES (%s, %s, %s)
        """, (userid, itemid, rating))

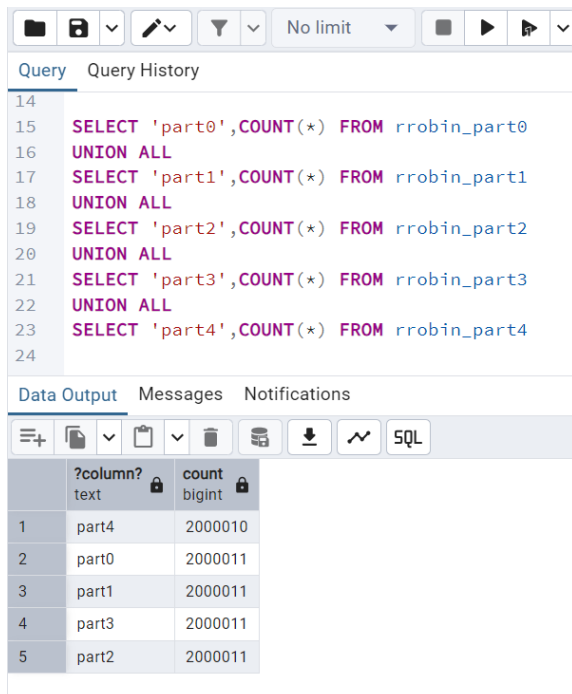
    cur.execute("""
        UPDATE partition metadata
        SET current_partition = (current_partition + 1) % partition_count
        WHERE partition_type = 'roundrobin'
        RETURNING ((current_partition - 1 + partition_count) % partition_count) AS old_partition,
        partition_count
        """)
    old_partition, numberofpartitions = cur.fetchone()

    partition_table = f"{RROBIN_TABLE_PREFIX}{old_partition}"
    cur.execute(f"""
        INSERT INTO {partition_table} (userid, movieid, rating)
        VALUES (%s, %s, %s)
        """, (userid, itemid, rating))

    openconnection.commit()
    cur.close()
```

### 7.3) Kiểm thử

- Với phần phân mảnh con của phần trước (phân mảnh thành 5 phần) đã có kết quả như sau



The screenshot shows a database query interface. At the top, there are icons for file operations and a dropdown menu. Below this is a 'Query' tab with a 'Query History' button. The main area displays a SQL query that counts records in five different partitions. Below the query is a 'Data Output' tab with a table of results. The table has two columns: 'part' and 'count'. The results show the count for each partition: part4 (2000010), part0 (2000011), part1 (2000011), part3 (2000011), and part2 (2000011).

```
14
15 SELECT 'part0',COUNT(*) FROM rrobin_part0
16 UNION ALL
17 SELECT 'part1',COUNT(*) FROM rrobin_part1
18 UNION ALL
19 SELECT 'part2',COUNT(*) FROM rrobin_part2
20 UNION ALL
21 SELECT 'part3',COUNT(*) FROM rrobin_part3
22 UNION ALL
23 SELECT 'part4',COUNT(*) FROM rrobin_part4
24
```

	?column? text	count bigint
1	part4	2000010
2	part0	2000011
3	part1	2000011
4	part3	2000011
5	part2	2000011

- Dễ dàng thấy nếu có bản ghi mới, nó sẽ được chèn vào part4, sau khi chèn vào part4 thì quay trở về chèn bắt đầu từ part0 -> part1 -> part2 -> part3 ...

```
[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")

# ALERT:: Change the partition index according to your testing sequence.
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 71568, 1, 3, conn, '4')
if result :
    print("roundrobininsert 1 function pass!")
else:
    print("roundrobininsert 1 function fail!")
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 71568, 2, 3, conn, '0')
if result :
    print("roundrobininsert 2 function pass!")
else:
    print("roundrobininsert 2 function fail!")
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 71568, 3, 3, conn, '1')
if result :
    print("roundrobininsert 3 function pass!")
else:
    print("roundrobininsert 3 function fail!")
```

Thử chèn 3 bản ghi, vậy thứ tự chèn phải tuân thủ chèn từ part4 rồi part0 rồi part1

Query

Query History

```

7  select 0 AS part,count(*) from rrobin_part0
8  UNION ALL
9  select 1 AS part,count(*) from rrobin_part1
10 UNION ALL
11 select 2 AS part,count(*) from rrobin_part2
12 UNION ALL
13 select 3 AS part,count(*) from rrobin_part3
14 UNION ALL
15 select 4 AS part,count(*) from rrobin_part4
16 ORDER BY part
17

```

Data Output

Messages

Notifications

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	part integer	count bigint
1	0	2000012
2	1	2000012
3	2	2000011
4	3	2000011
5	4	2000011

Vậy thuật toán đã tuân thủ đúng quy tắc,chèn vào bảng con số 4 rồi bắt đầu chèn đến bảng 0 và 1, phần kiểm tra bản ghi có tồn tại trong bảng con không thì hàm test đã thực hiện điều đó

### III) Tổng kết

Thực hiện bài test tổng

- Vì số lượng bản ghi thực sự trong file ratings khác với file ví dụ nên cần điều chỉnh 1 chút ở file Assignment1Tester.py

```
[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!")

# ALERT:: Change the partition index according to your testing sequence.
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '4')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '0')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '1')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
if result :
    print("roundrobininsert function pass!")
else:
    print("roundrobininsert function fail!")
```

- Phần test đầu vào sẽ được thay đổi theo vùng màu đỏ, kết quả cho ra được phần giống hình ảnh bên dưới

```
PS E:\BTLCSDLPT> & C:/Users/chien/AppData/Local/Programs/Python/Python312/python.exe e:/BTLCSDLPT/Code/Assignment1Tester.py
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
roundrobininsert function pass!
roundrobininsert function pass!
roundrobininsert function pass!
```

- Nhìn chung phần cài đặt thuật toán đã thành công.

Trong đề án này, nhóm em đã triển khai hai phương pháp phân mảnh ngang trên PostgreSQL:

Range Partition và Round Robin Partition, cùng với hai hàm chèn động tương ứng (Range\_Insert và RoundRobin\_Insert). Đầu tiên, dữ liệu MovieLens (10 triệu+ 54 đánh giá) được nạp vào bảng gốc Ratings qua hàm LoadRatings().

Với Range\_Partition(N), thực hiện chia khoảng giá trị rating thành N vùng đồng đều, tạo N bảng con range\_part0 ... range\_partN-1. Mỗi bản ghi được phân vào vùng phụ thuộc giá trị rating. Hàm Range\_Insert() sau đó đảm bảo chèn bản ghi mới vào đúng phân vùng bằng cách tính  $partition\_num = \text{floor}((rating - MIN) / range\_size)$ .

Với RoundRobin\_Partition(N), dùng ROW\_NUMBER() OVER () để đánh số các bản ghi hiện có và phân luồng vào N bảng con rrobin\_part0 ... rrobin\_partN-1 bằng công thức  $(rn - 1) \% N$ . Đối với dữ liệu mới, RoundRobin\_Insert() dùng câu UPDATE ... RETURNING để lấy chỉ số partition hiện tại và tăng mod N, sau đó chèn đánh giá vào bảng con tương ứng. Thử nghiệm với bộ dữ liệu mẫu và tập test do giảng viên cung cấp đều cho kết quả đúng, các testcase tự động pass 100%.

Trong quá trình thực hiện, nhóm nhận thấy metadata (current\_partition) trong Round Robin trở thành điểm “cổ chai” khi có nhiều client chen song song, dù đã dùng UPDATE ... RETURNING. Đối với Range Partition, khi dữ liệu phân phối không đồng đều (ví dụ rating tập trung ở 4–5 sao), một số partition chứa quá nhiều dữ liệu (skew), do đó cần cân nhắc nâng cấp thành Hash Partition hoặc giải pháp tự cân bằng.

Tóm lại, nhóm đã hoàn thành đầy đủ yêu cầu:

1. Load dữ liệu từ ratings.dat.
2. Tạo Range/ Round Robin Partition với N phân vùng.
3. Cài đặt hàm chèn động.
4. Kiểm thử toàn bộ test cases do giảng viên cung cấp.

- Phân chia công việc:

- Nguyễn Mậu Phi Hùng: làm Load Rating
- Nguyễn Mạnh Cường: làm Range Partition và Range Insert
- Tạ Quang Chiến: làm Round Robin Partition và Round Robin Insert