HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: CƠ SỞ DỮ LIỆU PHÂN TÁN

Nhóm sinh viên thực hiện: Nhóm 21

Đỗ Xuân Bách	B22DCCN053
Nguyễn Văn Huân	B22DCCN353
Bùi Đăng Thịnh	B22DCCN828

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Hà Nội 2025

LỜI CẢM ƠN

Kính gửi: Thầy giáo Kim Ngọc Bách - giảng viên bộ môn Cơ sở dữ liệu Phân tán.

Trước hết, nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến **Thầy Kim Ngọc Bách**, giảng viên hướng dẫn môn **Cơ sở dữ liệu phân tán**, người đã tận tình giảng dạy và truyền đạt cho chúng em những kiến thức quý báu trong suốt học kỳ vừa qua.

Nhờ sự hướng dẫn tận tâm và phương pháp giảng dạy sinh động của thầy, chúng em đã có cơ hội tiếp cận và nắm bắt được những khái niệm quan trọng cũng như các ứng dụng thực tiễn của hệ thống cơ sở dữ liệu phân tán trong lĩnh vực công nghệ thông tin. Đây là nền tảng quan trọng giúp chúng em củng cố kiến thức chuyên môn và rèn luyện tư duy hệ thống trong quá trình học tập và nghiên cứu.

Một lần nữa, nhóm chúng em xin chân thành cảm ơn thầy và kính chúc thầy luôn dồi dào sức khỏe, hạnh phúc và thành công trong sự nghiệp giảng dạy cũng như nghiên cứu khoa học.

Trân trọng!

Nhóm sinh viên thực hiện

Nhóm 21

MỤC LỤC

1: Tổng quan

1.1. Giới thiệu chung

Trong hệ thống cơ sở dữ liệu phân tán, phân mảnh dữ liệu (data fragmentation) là một kỹ thuật quan trọng nhằm tối ưu hóa hiệu năng, độ tin cậy và khả năng mở rộng của hệ thống. Phân mảnh cho phép chia nhỏ một bảng dữ liệu (hoặc quan hệ) thành nhiều phần nhỏ hơn gọi là các mảnh (fragments), sau đó phân bố các mảnh này đến các vị trí khác nhau trong hệ thống phân tán.

Việc phân mảnh giúp đảm bảo rằng dữ liệu được lưu trữ gần với nơi phát sinh truy vấn hoặc xử lý, giảm độ trễ truy xuất, tối ưu băng thông mạng và tăng khả năng chịu lỗi. Đồng thời, nó cũng hỗ trợ việc quản lý dữ liệu phân tán một cách minh bạch, sao cho người dùng không cần quan tâm đến việc dữ liệu thật sự nằm ở đâu.

Bài tập này yêu cầu mô phỏng hai phương pháp phân mảnh ngang phổ biến: Range Partition và Round-Robin Partition trên cơ sở dữ liệu PostgreSQL.

Dữ liệu sử dụng là tập đánh giá phim từ MovieLens (tệp ratings.dat), bao gồm hơn 10 triệu dòng dữ liệu, mỗi dòng chứa thông tin: UserID:: MovieID:: Rating:: Timestamp.

1.2. Phạm vi tìm hiểu

- Phân mảnh theo dải giá trị (Range Partitioning): Range Partition là phương pháp phân mảnh dựa trên một khoảng giá trị của thuộc tính trong bảng. Mỗi mảnh sẽ chứa các bản ghi có giá trị của thuộc tính đó nằm trong một khoảng xác định.

- Phân mảnh theo vòng tròn (Round-Robin Partitioning): Round-Robin là phương pháp phân phối bản ghi theo thứ tự tuần tự, lần lượt đưa từng bản ghi vào các mảnh một cách luân phiên, bất kể nội dung của dữ liệu.

1.3. Mục đích và yêu cầu

- Cài đặt và sử dụng hệ quản trị cơ sở dữ liệu PostgreSQL.

- Cài đặt các hàm Python để:
  - Tải dữ liệu vào bảng Ratings.
  - Phân mảnh dữ liệu bằng phương pháp Range và Round-Robin.
  - Thêm dữ liệu mới vào đúng phân mảnh tương ứng.

Phần 2: Cơ sở lý thuyết

2.1. Cơ sở lý thuyết

- Phân mảnh dữ liệu (logic) là dữ liệu trong một hệ thống cơ sở dữ liệu lớn được chia nhỏ và phân tán trên nhiều máy chủ nhằm cải thiện hiệu suất, khả năng mở rộng và độ tin cậy.
- Mục đích:

- Cải thiện hiệu suất: truy vấn có thể xử lý song song trên nhiều máy chủ, giảm tải và tăng tốc độ phản hồi cho từng máy chủ.
- Khả năng mở rộng: Dễ dàng thêm các máy chủ mới khi lượng dữ liệu và người dùng tăng lên mà không cần nâng cấp một máy chủ duy nhất.
- Độ tin cậy và tính sẵn sàng: Nếu một máy chủ bị lỗi, chỉ một phần dữ liệu bị ảnh hưởng, hệ thống có thể tiếp tục hoạt động với phần còn lại của dữ liệu.
- Quản lý dữ liệu lớn: Giúp quản lý và xử lý hiệu quả lượng dữ liệu khổng lồ mà một máy chủ duy nhất khó có thể đảm đương.

2.2. Các phương pháp phân mảnh

- Range Partitioning:
  - Cách thức hoạt động: Khi một bản ghi mới được thêm vào, hệ thống sẽ kiểm tra giá trị của cột phân vùng trong bản ghi đó và đưa nó vào phân vùng tương ứng với dải giá trị mà nó thuộc về.

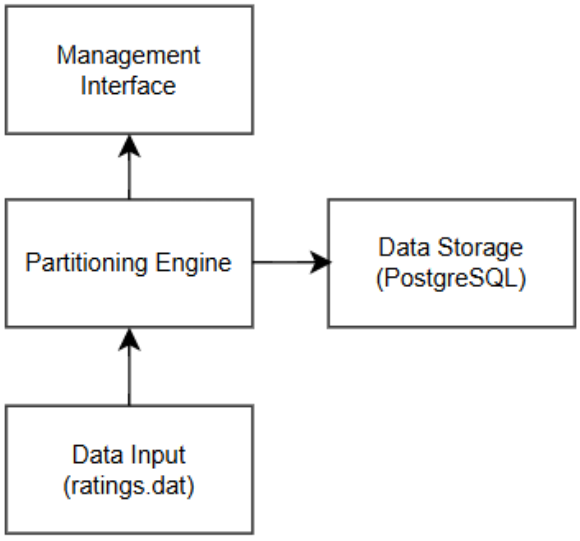
    → phân vùng dữ liệu trên khoảng giá trị của thuộc tính.
  - Ưu điểm: Hiệu quả đối với dải truy vấn, dễ quản lý các dữ liệu cũ và phù hợp dữ liệu được phân phối đều theo thời gian.

- Hạn chế: một dải giá trị có quá nhiều dữ liệu hoặc quá nhiều truy vấn, phân vùng đó có thể trở thành hotspot làm mất cân bằng giữa các phân vùng. Và nhất là cần xác định trước các khoảng giá trị cần điều chỉnh khi dữ liệu thay đổi.

- Round Robin Partitioning:
  - Cách thức hoạt động: Khi một bản ghi mới được chèn vào, nó sẽ được gán cho phân vùng tiếp theo trong danh sách các phân vùng có sẵn. Sau khi đạt đến phân vùng cuối cùng, nó sẽ quay trở lại phân vùng đầu tiên và lặp lại.

    → phân vùng dữ liệu một cách tuần tự, xoay vòng.
  - Ưu điểm: Là phương pháp đơn giản, dễ hiểu; dữ liệu luôn được phân tán đều trên các vùng, tránh hotspot bởi quá tải.
  - Hạn chế: truy vấn kém hiệu quả, khó mở rộng số lượng phân vùng và không tận dụng được locality (các nhóm có dữ liệu liên quan đến nhau).

2.3. Thiết kế hệ thống



Hình 1: Sơ đồ cấu trúc hệ thống

Phần 3: Cài đặt và triển khai

3.1. Cài đặt môi trường

Để thực hiện yêu cầu phân mảnh dữ liệu theo 2 cách (Range Partition và RoundRobin Partition), nhóm đã thực hiện cài đặt các môi trường cần thiết đảm bảo cho việc xử lý dữ liệu và tiến hành phân mảnh.

Các môi trường bao gồm:

- Hệ điều hành: Windows 11.
- Ngôn ngữ lập trình: Python 3.12.10.
- Phần mềm quản trị cơ sở dữ liệu: pgAdmin 4.
- Hệ thống quản lý cơ sở dữ liệu: PostgreSQL 14.18.

3.1.1. Cài đặt Python 3.12.10 và thiết lập môi trường

1. Cài đặt Python 3.12.10:

Truy cập trang *https://www.python.org/downloads/* và tải phiên bản Python 3.12.10. Tiến hành install tại thư mục tải về, lưu ý cần tick vào ô "Add python.exe to PATH" để sử dụng Python từ dòng lệnh rồi mới click "Install Now".

Hình 2: Cài đặt Python 3.12.10

2. Tải và cấu hình mã nguồn:

Thực hiện clone mã nguồn trên github:

git clone https://github.com/thinhote/Nhom21-BTL_CSDLPT.git

3. Cài đặt thư viện:

Sử dụng công cụ pip để cài đặt thư viện kết nối với PostgreSQL:

pip install psycopg2

3.1.2. Cài đặt pgAdmin 4 và PostgreSQL

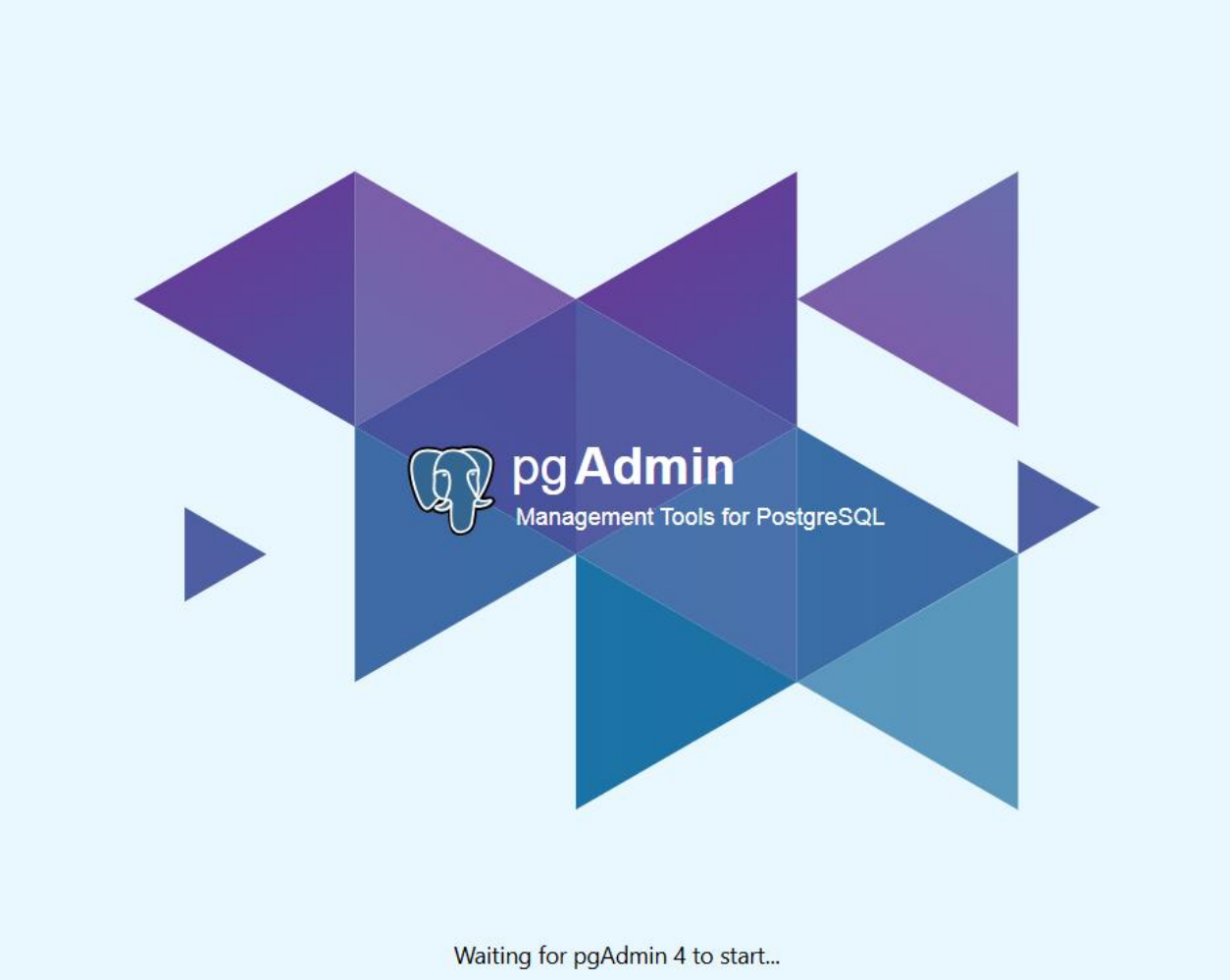
PostgreSQL là một hệ thống quản trị cơ sở dữ liệu quan hệ và đối tượng (object-relational database management system) miễn phí và nguồn mở (RDBMS) tiên tiến nhất hiện nay. khả năng mở rộng cao và tuân thủ các tiêu chuẩn kỹ thuật. Nó được thiết kế để xử lý một loạt các khối lượng công việc lớn, từ các máy tính cá nhân đến kho dữ liệu hoặc dịch vụ Web có nhiều người dùng đồng thời.

1. Tải và cài đặt PostgreSQL:

Truy cập trang *https://www.enterprisedb.com/downloads/postgres-postgresql-downloads/* và tải PostgreSQL Version 14.18 dành cho Windows x86-64. Bộ cài đặt đã bao gồm phần mềm pgAdmin 4.

2. Tạo database:

Sau khi pgAdmin 4 đã cài đặt xong, đăng nhập và tạo mật khẩu mới cho người dùng. Sau đó tiến hành tạo một database mới.



Hình 3: Cài đặt pgAdmin 4

3.2. Chuẩn bị dữ liệu

Dữ liệu đầu vào là một tập dữ liệu đánh giá phim được thu thập từ trang web MovieLens (http://movielens.org).

3.2.1. Download và xử lý file dữ liệu

- Truy cập trang *http://files.grouplens.org/datasets/movielens/ml-10m.zip* để bắt đầu tải file nén chứa dữ liệu.

- Giải nén file zip vừa tải về và tìm đến tệp dữ liệu '*ratings.dat*'.

- Copy file 'ratings.dat' và paste tại thư mục chứa mã nguồn.

3.2.2. Phân tích dữ liệu

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 trường dữ liệu:

+) UserID: mã người dùng (kiểu int).

+) MovieID: mã bộ phim (kiểu int).

+) Rating: điểm đánh giá (kiểu float với thang điểm 5, có thể chia nửa điểm).

+) Timestamp: dấu thời gian (số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970).

3.3. Chi tiết triển khai cài đặt các hàm

3.3.1. Cấu trúc dữ liệu

- Bảng chính (master):

```
CREATE TABLE ratings (
        userid INTERGER,
        movieid INTERGER,
        rating FLOAT
);
```

- Bảng phân mảnh:

- Range Partitions 0 - (n-1):

```
CREATE TABLE range_part0 (
        userid INTERGER,
        movieid INTERGER,
        rating FLOAT
);
```

CREATE TABLE range_part1 (

userid INTERGER,

movieid INTERGER,

rating FLOAT

);

… part (n-1)

- Round Robin Partitions 0 - (n-1):

CREATE TABLE rrobin_part0 (

userid INTERGER,

movieid INTERGER,

rating FLOAT

);

(tương tự range) … part (n-1)

- Phương thức kết nối:

Sử dụng thư viện:

```
import psycopg2

DATABASE_NAME = 'dds_assgn1'
#Kết nối với PostgreSQL database
```

Cài đặt csdl:

```
def getopenconnection(user='postgres', password='14102004', dbname='postgres'):
    return psycopg2.connect(f"dbname='{dbname}' user='{user}' host='localhost'
password='{password}'")
```

Các trường dữ liệu:

```
USER_ID_COLNAME = 'userid'
MOVIE_ID_COLNAME = 'movieid'
RATING_COLNAME = 'rating'
INPUT_FILE_PATH = 'ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 10000054  # Number of lines in the input file
```

Các bảng chứa dữ liệu:

```
RATINGS_TABLE = 'ratings'
RANGE_TABLE_PREFIX = 'range_part'
RROBIN_TABLE_PREFIX = 'rrobin_part'
```

File dữ liệu input:

```
INPUT_FILE_PATH = 'ratings.dat'
ACTUAL_ROWS_IN_INPUT_FILE = 10000054  # Number of lines in the input file
```

3.3.2. Các hàm chức năng chính

a) Hàm loadratings

```
def loadratings(tablename, filepath, conn):
    try:
        cur = conn.cursor()
    # Tạo bảng nếu chưa tồn tại
        cur.execute(f"""
            CREATE TABLE IF NOT EXISTS {tablename} (
                userid INTEGER,
                movieid INTEGER,
                rating FLOAT
            );
        """)
        batch_size = 50000
        batch = []
        with open(filepath, 'r') as file:
            for line in file:
                tokens = line.strip().split("::")
                if len(tokens) >= 3:
                    user = int(tokens[0])
                    movie = int(tokens[1])
                    rate = float(tokens[2])
                    batch.append((user, movie, rate))

                    # Khi đủ batch_size thì insert một lần
                    if len(batch) >= batch_size:
                        cur.executemany(
                            f"INSERT INTO {tablename} (userid, movieid, rating)
VALUES (%s, %s, %s);", batch)
                        batch = []

            # Chèn phần còn lại chưa đủ batch_size
            if batch:
                cur.executemany(
                    f"INSERT INTO {tablename} (userid, movieid, rating) VALUES
(%s, %s, %s);", batch)

        conn.commit()
        cur.close()
    except Exception as e:
        print(f'Error in loadratings:', e)
```

Hình 4: Hàm loadratings()

Mục đích: Tạo bảng gốc gồm các thuộc tính: userid, movieid và rating và nạp dữ liệu từ file ratings.dat vào bảng.

Thực thi:

- Tạo bảng ratings mới nếu chưa tồn tại trên database với 3 trường:
  - userid INT
  - movieid INT
  - rating FLOAT
- Đọc file dữ liệu đầu vào:
  - Tạo 1 danh sách để lưu tạm dữ liệu chèn vào với kích thước là 50.000 => Nạp mỗi lần 50.000 dòng để tăng tốc độ nạp.
  - Mở file và đọc từng dòng, mỗi dòng tách bằng "::" và lấy 3 phần tử đầu tiên.
  - Gán vào user, movie, rate rồi thêm vào batch.
  - Nếu batch đạt 50.000 dòng thì dùng executemany() để chèn hàng loạt vào CSDL rồi reset batch.
  - Nếu đọc hết file mà batch vẫn chưa đầy thì chèn phần còn lại trong batch vào CSDL.
  - Ghi lại các thay đổi bằng commit.

b) Hàm rangepartition

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    cursor = openconnection.cursor()
    # Xóa các phân mảnh cũ nếu tồn tại (tùy chọn an toàn khi chạy nhiều lần)
    for i in range(numberofpartitions):
        cursor.execute(f"DROP TABLE IF EXISTS range_part{i};")

    # Tính độ rộng mỗi phân mảnh
    min_rating = 0.0
    max_rating = 5.0
    partition_width = (max_rating - min_rating) / numberofpartitions

    # Tạo bảng và chèn dữ liệu cho từng phân mảnh
    for i in range(numberofpartitions):
        lower_bound = min_rating + i * partition_width
        upper_bound = lower_bound + partition_width
```

```
        # Tạo bảng phân mảnh mới
        cursor.execute(f"""
            CREATE TABLE range_part{i} (
                UserID INT,
                MovieID INT,
                Rating FLOAT
            );
        """)
        # Tùy điều kiện WHERE theo phân mảnh đầu tiên và các phân mảnh còn lại
        if i == 0:
            condition = f"Rating >= {lower_bound} AND Rating <= {upper_bound}"
        else:
            condition = f"Rating > {lower_bound} AND Rating <= {upper_bound}"
        # Chèn dữ liệu từ bảng gốc vào phân mảnh
        cursor.execute(f"""
            INSERT INTO range_part{i} (UserID, MovieID, Rating)
            SELECT UserID, MovieID, Rating
            FROM {ratingstablename}
            WHERE {condition};
        """)
    openconnection.commit()
    cursor.close()
```

Hình 5: Hàm rangepartition()

Mục đích: Chia bảng ratings thành n bảng phân mảnh ngang dựa trên giá trị Rating. Mỗi bảng chứa một khoảng rating riêng biệt, được chia đều từ 0.0 đến 5.0.

Thực thi:

- Xóa bảng phân mảnh cũ (nếu có) để khi tạo lại không bị trung lặp.
- Tính khoảng cách điểm số mỗi phân mảnh.
- Sử dụng vòng lặp for để tạo các bảng phân mảnh, lặp qua từng phân mảnh để tạo bảng mới.
- Phân biệt phân mảnh đầu tiên để đảm bảo không trùng lặp rating:
  - range_part0: lấy cả Rating = 0.0
  - Các bảng sau: lấy > ở dưới, <= ở trên.
- Sau đó, chèn dữ liệu vào các bảng phân mảnh vừa tạo với các điều kiện biên.
- Ghi lại các thay đổi bằng commit và đóng cursor.

c) Hàm roundrobinpartition

```python
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
    cur = openconnection.cursor()
    # Xóa các bảng partition cũ nếu có
    for i in range(numberofpartitions):
        cur.execute(f"DROP TABLE IF EXISTS rrobin_part{i};")
    for i in range(numberofpartitions):
        cur.execute(f"CREATE TABLE rrobin_part{i} (userid INT, movieid INT,
rating FLOAT);")
    # Lấy tất cả dữ liệu và phân phối theo thứ tự
    cur.execute(f"SELECT userid, movieid, rating FROM {ratingstablename};")
    rows = cur.fetchall()
    for idx, row in enumerate(rows):
        part_idx = idx % numberofpartitions
        cur.execute(f"INSERT INTO rrobin_part{part_idx} (userid, movieid, rating)
VALUES (%s, %s, %s);", row)
    openconnection.commit()
    cur.close()
```

Hình 6: Hàm roundrobinpartition()

Mục đích: Phân chia dữ liệu từ bảng đánh giá (ratings table) thành nhiều bảng con (partitions) theo phương pháp vòng tròn (round robin).

Thực thi:

- Tạo con trỏ truy vấn: cur = openconnection.cursor()

- Xóa các bảng partition cũ (nếu có): Lặp qua số lượng partition, xóa các bảng rrobin_part0, rrobin_part1, ..., rrobin_partN-1 nếu chúng đã tồn tại.

- Tạo các bảng partition mới: Tạo lại các bảng rrobin_part0, rrobin_part1, ..., rrobin_partN-1 với cấu trúc giống bảng gốc (userid, movieid, rating).

- Lấy toàn bộ dữ liệu từ bảng đánh giá: Truy vấn tất cả các dòng dữ liệu từ bảng ratingstablename.

- Phân phối dữ liệu theo vòng tròn:

Duyệt từng dòng dữ liệu, xác định bảng partition sẽ chèn dựa vào chỉ số dòng (idx % numberofpartitions).

Chèn dòng dữ liệu vào bảng partition tương ứng.

- Lưu thay đổi và đóng con trỏ.

- Ghi nhận các thay đổi vào cơ sở dữ liệu và đóng con trỏ.

d) Hàm rangeinsert

- Chèn dữ liệu mới vào phân vùng dựa trên khoảng giá trị

```
- def rangeinsert(tablename, userid, movieid, rating, conn):
-     try:
-         cur = conn.cursor()
-         cur.execute(f"INSERT INTO {tablename} VALUES (%s, %s, %s);",
  (userid, movieid, rating))
-
-         cur.execute("SELECT COUNT(*) FROM pg_tables WHERE tablename LIKE
  'range_part%';")
-         n = cur.fetchone()[0]
-         step = 5.0 / n
-
-         index = int(rating / step)
-         if rating % step == 0 and index != 0:
-             index -= 1
-
-         target_table = f"{RANGE_TABLE_PREFIX}{index}"
-         cur.execute(f"INSERT INTO {target_table} VALUES (%s, %s, %s);",
  (userid, movieid, rating))
-
-         conn.commit()
-         cur.close()
-
-     except Exception as e:
-         print(f'Error in rangeinsert:', e)
```

Hình 7: Hàm rangeinsert()

Mục đích: Thêm một dòng dữ liệu mới vào bảng ratings gốc và bảng phân mảnh tương ứng theo phương pháp range partition.

Thực thi:

- Chèn dòng mới vào bảng chính ratings.

- Lấy số lượng bảng phân mảnh range_part0, range_part1,... đang có trong cơ sở dữ liệu.

- Tính độ rộng của mỗi phân mảnh step, ví dụ nếu có 5 phân mảnh => Mỗi phân mảnh có độ rộng 1.0 đơn vị.

- Tính chỉ số phân mảnh mà dòng hiện tại cần được đưa vào:

  - rating = 2.9 → nằm trong khoảng [2.0, 3.0] (index = 2)
  - rating = 3.0 → cần xử lý đặc biệt nếu nằm ngay biên (% step == 0)

e) Hàm roundrobininsert

- Chèn dữ liệu mới vào phân vùng theo round-robin

```
- def roundrobininsert(ratingstablename, userid, itemid, rating,
  openconnection):
-     cur = openconnection.cursor()
-     # Thêm vào bảng chính
-     cur.execute(f"INSERT INTO {ratingstablename} (userid, movieid, rating)
  VALUES (%s, %s, %s);", (userid, itemid, rating))
-     # Xác định số partition
-     cur.execute("SELECT COUNT(*) FROM information_schema.tables WHERE
  table_name LIKE 'rrobin_part%';")
-     numberofpartitions = cur.fetchone()[0]
-     # Đếm tổng số bản ghi đã có trong bảng ratings
-     cur.execute(f"SELECT COUNT(*) FROM {ratingstablename};")
-     total_rows = cur.fetchone()[0]
-     index = (total_rows - 1) % numberofpartitions
-     cur.execute(f"INSERT INTO rrobin_part{index} (userid, movieid, rating)
  VALUES (%s, %s, %s);", (userid, itemid, rating))
-     openconnection.commit()
-     cur.close()
```

Hình 8: Hàm roundrobininsert()

Mục đích: Chèn một bản ghi mới (một đánh giá mới) vào bảng đánh giá chính và đồng thời phân phối bản ghi này vào bảng partition phù hợp theo phương pháp round robin.

Thực thi:

- Tạo con trỏ truy vấn: cur = openconnection.cursor()

- Chèn bản ghi vào bảng chính: Thêm dòng dữ liệu mới (userid, itemid, rating) vào bảng đánh giá gốc (ratingstablename).

- Xác định số lượng partition: Truy vấn số lượng bảng partition hiện có với tên bắt đầu bằng rrobin_part.

- Đếm tổng số bản ghi trong bảng chính: Lấy tổng số dòng hiện có trong bảng đánh giá sau khi đã thêm bản ghi mới.

- Tính toán partition sẽ chèn: Sử dụng công thức (total_rows - 1) % numberofpartitions để xác định partition phù hợp cho bản ghi mới (trừ 1 vì chỉ số bắt đầu từ 0).

- Chèn bản ghi vào partition tương ứng: Thêm dòng dữ liệu mới vào bảng partition đã xác định.
- Lưu thay đổi và đóng con trỏ: Ghi nhận các thay đổi vào cơ sở dữ liệu và đóng con trỏ.

Phần 4: Kiểm thử

4.1. LoadRatings()

Chạy file Assignment1Tester.py với hàm loadratings():

```
[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!")
```

Hình 9: Mã code kiểm thử hàm loadratings()

Sau khi chạy hàm trên, ta được:

```
A database named "dds_assgn1" already exists
loadratings function pass!
```

Hình 10: Kết quả trên console sau khi chạy hàm loadratings()

Kiểm tra trong pgAdmin 4, kiểm tra lại bảng ratings bằng câu lệnh SQL:

*SELECT * FROM ratings;*

Ta thu được bảng ratings có số dòng là 10.000.054 dòng bằng đúng với số lượng dòng trong file ratings.dat.

Query | Query History | Scratch Pad

```
1  SELECT * FROM ratings;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page No: 1 of 10001

	userid integer	movieid integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5

Total rows: 10000054 | Query complete 00:00:03.143

CRLF | Ln 1, Col 22

Hình 11: Kết quả bảng ratings

Kết luận: Hàm Python loadratings() đã được cài đặt đúng để load dữ liệu từ file input ratings.dat vào bảng ratings trong PostgreSQL.

4.2. Range_Partition()

Chạy file Assignment1Tester.py với hàm rangepartition():

```
[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!")
```

Hình 12: Mã code kiểm thử hàm rangepartition()

Sau khi chạy tiếp hàm trên, ta được:

```
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
```

Hình 13: Kết quả trên console sau khi chạy hàm rangepartition()

Vào pgAdmin 4 kiểm tra, ta thấy 5 bảng mới đã được tạo tương ứng với 5 phân mảnh từ range_part0, range_part1, …, range_part5.

Tổng số bản ghi của 5 bảng bằng đúng tổng số dòng của tệp dữ liệu ratings.dat (10.000.054 dòng).

Số lượng bản ghi chi tiết các bảng như sau:

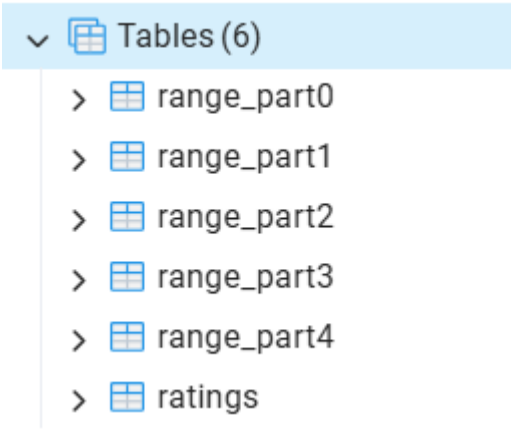
+) range_part0: 479.168 dòng.

+) range_part1: 908.584 dòng.

+) range_part2: 2.726.854 dòng.

+) range_part3: 3.755.614 dòng.

+) range_part4: 2.129.834 dòng.



Hình 14: Các bảng trong database sau khi chạy hàm rangepartiton()

Kết luận: Hàm Python rangepartition() đã được cài đặt đúng để tạo N phân mảnh ngang của bảng ratings và lưu vào PostgreSQL.

4.3. Range_Insert()

Chạy file Assignment1Tester.py với hàm rangeinsert():

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

Hình 15: Mã code kiểm thử hàm rangeinsert ()

Sau khi chạy tiếp hàm trên, ta được:

```
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
```

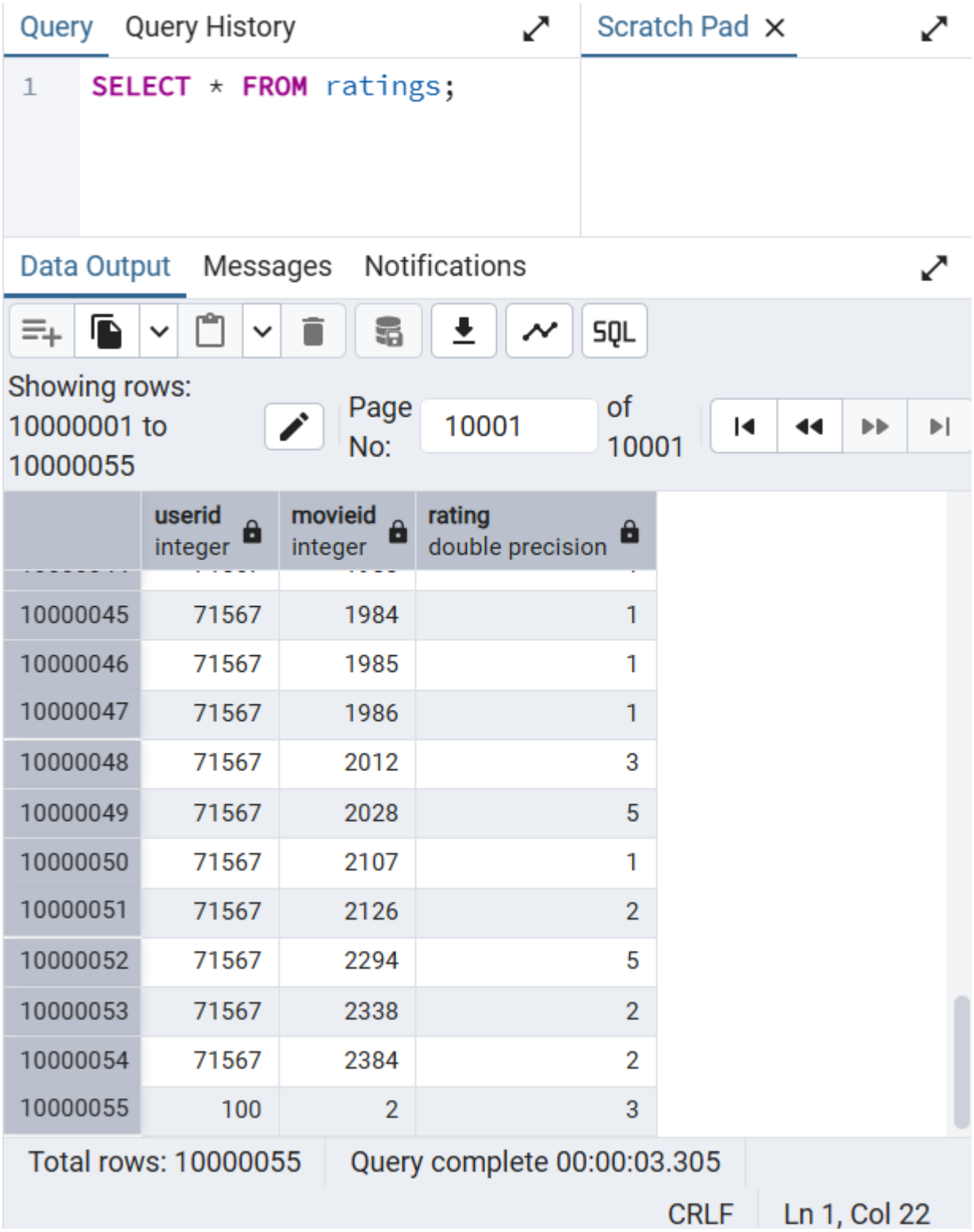
Hình 16: Kết quả trên console sau khi chạy hàm rangeinsert()

Vào pgAdmin 4 kiểm tra với 2 câu lệnh SQL:

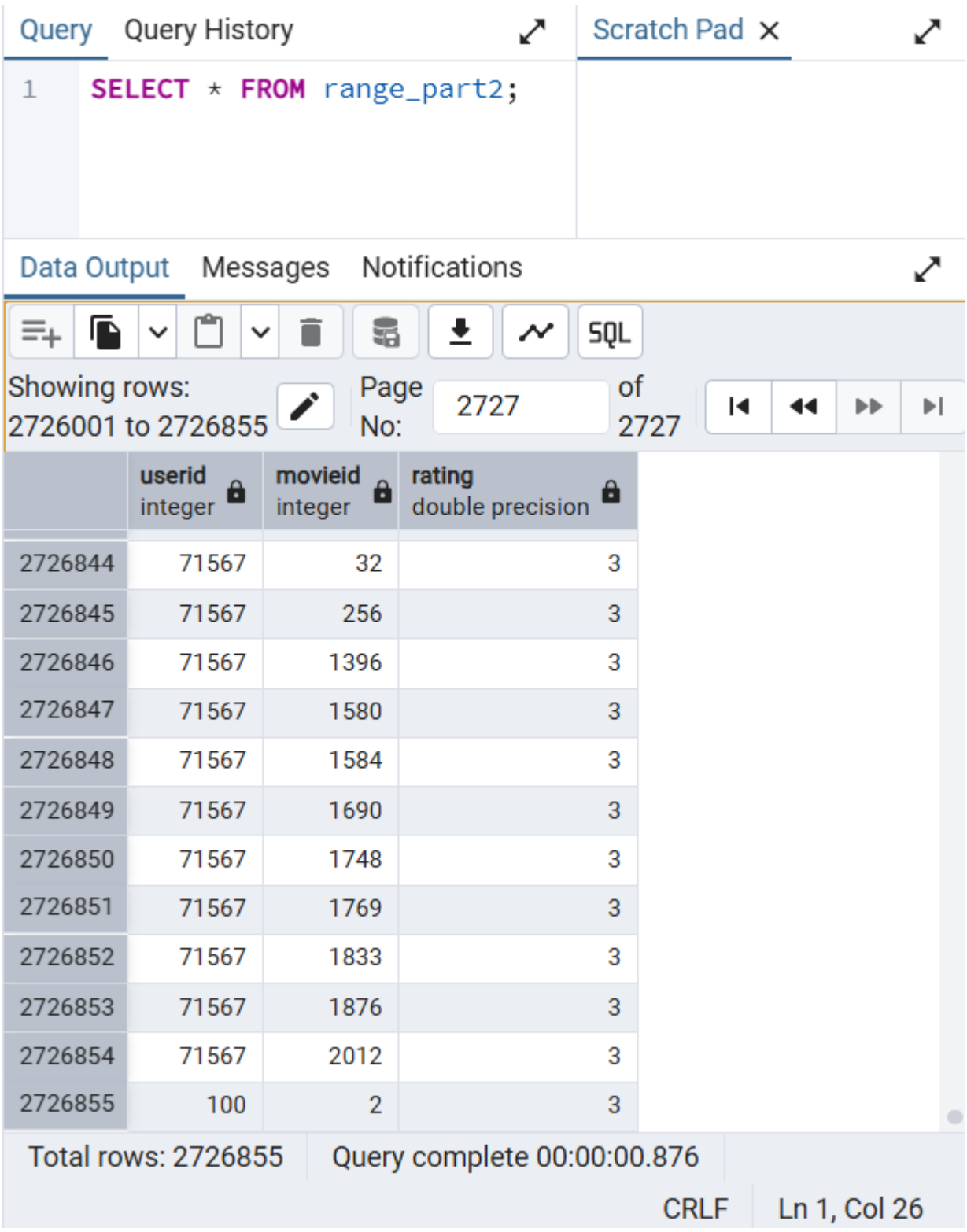
*SELECT * FROM ratings;*

*SELECT * FROM range_part2;*

Ta thấy số lượng bản ghi trong bảng ratings và range_part2 đều tăng thêm 1, chứng tỏ đã thêm thành công 1 bản ghi vào bảng ratings và range_part2.



Hình 17: Kết quả bảng ratings sau khi insert



Hình 18: Kết quả bảng range_part2 sau khi insert

Kết luận: Hàm Python rangeinsert() đã được cài đặt đúng để chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh dựa trên giá trị của Rating.

4.4. RoundRobin_Partition()

Chạy file Assignment1Tester.py với hàm roundrobinpartition():

```
[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")
```

Hình 19: Mã code kiểm thử hàm roundrobinpartition ()

Sau khi chạy tiếp hàm trên, ta được:

```
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
```

Hình 20: Kết quả trên console sau khi chạy hàm roundrobinpartition()

Vào pgAdmin 4 kiểm tra, ta thấy 5 bảng mới đã được tạo tương ứng với 5 phân mảnh từ rrobin_part0, rrobin _part1, …, rrobin _part5.

Tổng số bản ghi của 5 bảng bằng đúng tổng số dòng của tệp dữ liệu ratings.dat (10.000.054 dòng).

Số lượng bản ghi chi tiết các bảng như sau:

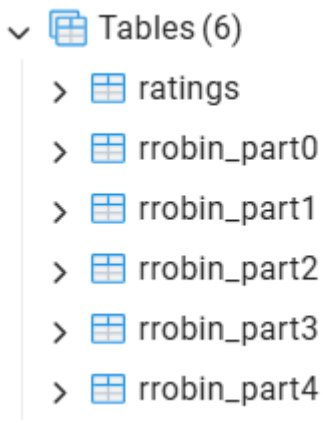
+) rrobin_part1: 2.000.011 dòng.

+) rrobin_part1: 2.000.011 dòng.

+) rrobin_part1: 2.000.011 dòng.

+) rrobin_part1: 2.000.011 dòng.

+) rrobin_part1: 2.000.010 dòng.



Hình 21: Các bảng trong database sau khi chạy hàm roundrobinpartiton()

Kết luận: Hàm Python roundrobinpartition() đã được cài đặt đúng để tạo N phân mảnh kiểu vòng tròn(round robin) từ bảng ratings và lưu vào PostgreSQL.

4.5. RoundRobin_Insert()

Chạy file Assignment1Tester.py với hàm roundrobininsert():

```
[result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100,
1, 3, conn, '4')
            if result :
                print("roundrobininsert function pass!")
            else:
                print("roundrobininsert function fail!")
```

Hình 22: Mã code kiểm thử hàm roundrobininsert ()

Sau khi chạy tiếp hàm trên, ta được:

```
A database named "dds_assgn1" already exists
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
roundrobinpartition function pass!
roundrobininsert function pass!
```

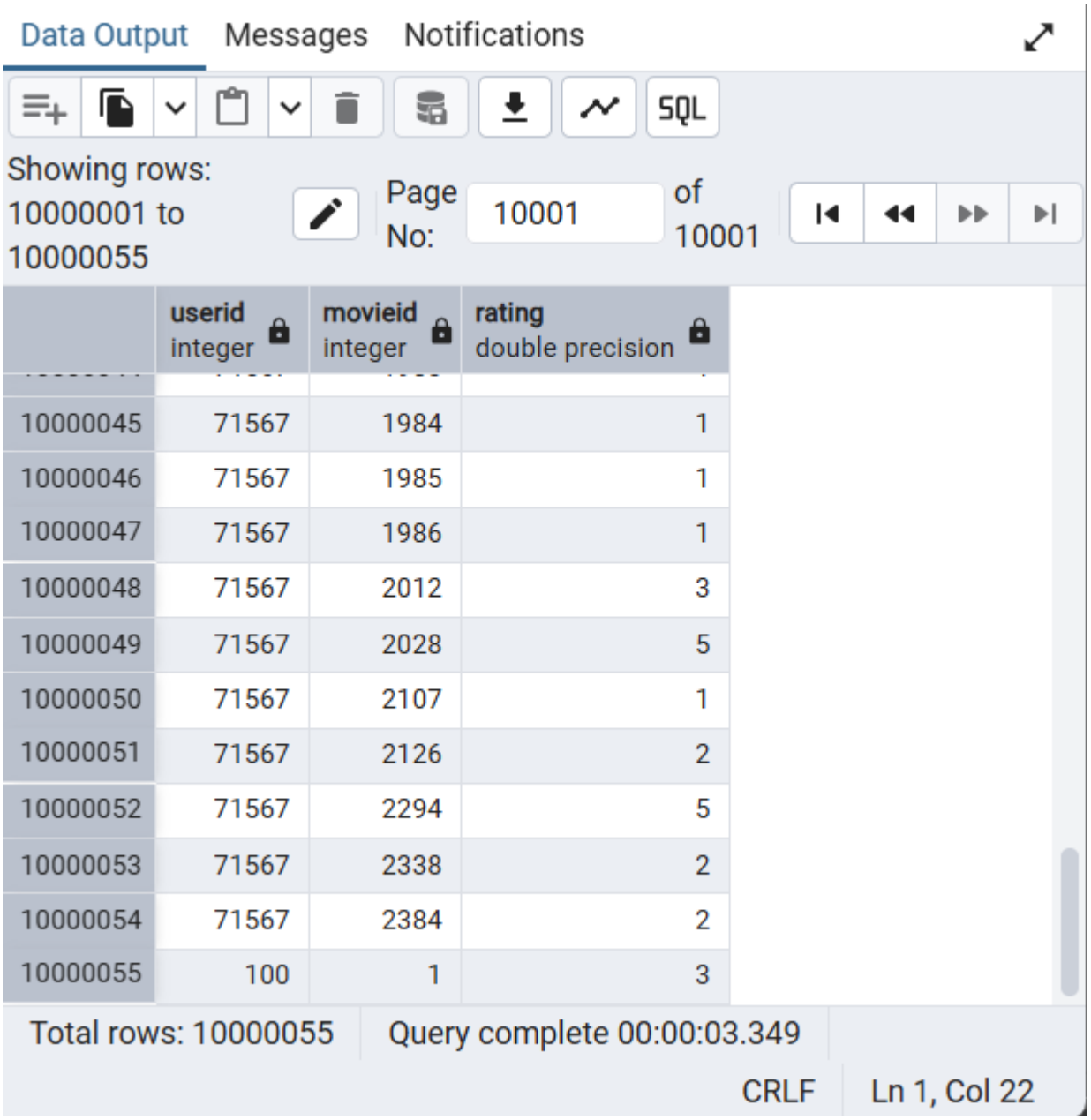
Hình 23: Kết quả trên console sau khi chạy hàm roundrobininsert()

Vào pgAdmin 4 kiểm tra với 2 câu lệnh SQL:

*SELECT * FROM ratings;*

*SELECT * FROM rrobin_part4;*

Ta thấy số lượng bản ghi trong bảng ratings và range_part2 đều tăng thêm 1, chứng tỏ đã thêm thành công 1 bản ghi vào bảng ratings và range_part2.



	userid integer	movieid integer	rating double precision
10000045	71567	1984	1
10000046	71567	1985	1
10000047	71567	1986	1
10000048	71567	2012	3
10000049	71567	2028	5
10000050	71567	2107	1
10000051	71567	2126	2
10000052	71567	2294	5
10000053	71567	2338	2
10000054	71567	2384	2
10000055	100	1	3

Hình 24: Kết quả bảng ratings sau khi insert

Data Output | Messages | Notifications

Showing rows: 2000001 to 2000011 | Page No: 2001 of 2001

	userid integer	movieid integer	rating double precision
2000001	71567	256	3
2000002	71567	589	4
2000003	71567	898	4
2000004	71567	1210	4
2000005	71567	1396	3
2000006	71567	1598	2
2000007	71567	1769	3
2000008	71567	1909	2
2000009	71567	1984	1
2000010	71567	2107	1
2000011	100	1	3

Total rows: 2000011 | Query complete 00:00:00.786

CRLF | Ln 1, Col 27

Hình 25: Kết quả bảng range_part2 sau khi insert

Kết luận: Hàm Python roundrobininsert() đã được cài đặt đúng để chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh theo cách round robin.

Phần 5: Đánh giá

5.1. Đánh giá

Hệ thống đã đạt được mục tiêu ban đầu: mô phỏng 2 phương pháp phân mảnh là Range Partition và RoundRobin Partition; xử lý được tệp dữ liệu ratings.dat với ≥ 10 triệu dòng dữ liệu, mỗi dòng chứa thông tin: UserID:: MovieID:: Rating:: Timestamp.

Một số ưu nhược điểm:

- Ưu điểm:
  - Đã kết hợp được cả hai phương pháp phân mảnh bổ trợ cho nhau.
  - Cơ chế transaction (commit) đảm bảo tính nhất quán.
  - Code được tổ chức tốt và dễ mở rộng.
- Nhược điểm:
  - Hiệu năng còn khá kém, 15-20p cho 10 triệu dòng dữ liệu.
  - Chất lượng code chưa đạt đúng kỳ vọng.

5.2. Phương hướng phát triển

Một số điểm có thể cải thiện để tối ưu:

- Với Range Partition: thêm tùy chọn cho khoảng phân mảnh không đều, cho phép điều chỉnh biên độ dựa trên phân phối dữ liệu thực tế.
- Với RoundRobin: cơ chế cache để tối ưu truy vấn(khắc phục nhược điểm của rrobin), hỗ trợ rebalancing khi thêm/bớt phân mảnh.
- Ngoài ra: Cơ chế monitoring hiệu năng của từng phân mảnh, backup và recovery theo từng phân mảnh, có thể cải thiện code để tối ưu hiệu năng.

Trên đây là đánh giá và phương hướng khách quan của chúng em tự đưa ra cho nên vẫn còn sai sót. Vậy nên nhóm chúng em mong thầy có thể đưa ra một số lời khuyên cũng như đánh giá về hệ thống trên đây.

Qua dự án này nhóm chúng em đã củng cố kiến thức cũng như hiểu thêm về cách hoạt động phân mảnh dữ liệu, được sử dụng và áp dụng nó vào 1 dự án thiết thực góp phần vào sự phát triển của 1 số dự án khác sau này.

Phần 6: Tài liệu tham khảo

Công cụ đã sử dụng:

PostgreSQL: https://www.postgresql.org/download/

Ngôn ngữ phát triển: Python

Quản trị: pgAdmin

Tài liệu tham khảo:

https://www.w3schools.com/sql/

https://www.w3schools.com/python/

https://github.com/bachkimn/bai_tap_lon_CSDL_phan_tan/

Source code của nhóm:

GitHub - thinhote/Nhom21-BTL_CSDLPT

BẢNG PHÂN CHIA CÔNG VIỆC

Thành viên	Lập trình	Viết báo cáo
Đỗ Xuân Bách	loadratings() count_partitions()	Phần 2, 5, 6
Nguyễn Văn Huân	round_robinpartition() round_robininsert()	Phần 3, 4
Bùi Đăng Thịnh	loadratings() range_partition() range_insert()	Phần 1, 3, 4