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

MÃ HỌC PHẦN: INT14148

Các sinh viên thực hiện:

B22DCCN584	Tạ Kim Ngân
B22DCCN104	Lê Minh Châu
B22DCCN773	Phạm Văn Tuyến

Số thứ tự nhóm: 13

Tên lớp: CSDLPT – N10

Giảng viên hướng dẫn: Thầy Kim Ngọc Bách

Hà Nội – 2025

PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN

TT	Công việc / Nhiệm vụ	SV thực hiện	Thời hạn hoàn thành
1	Setup, cài đặt công cụ cần thiết Hàm loadratings()	Tạ Kim Ngân	22/5/2025
2	Hàm range_partition() Hàm range_insert()	Lê Minh Châu	1/6/2025
3	Hàm roundrobin_partition () Hàm roundrobin_insert()	Phạm Văn Tuyến	1/6/2025
4	Phân công, tổng hợp, viết báo cáo	Tạ Kim Ngân	8/6/2025

MỤC LỤC

DANH MỤC CÁC HÌNH ẢNH

MỞ ĐẦU

Trong bối cảnh dữ liệu lớn ngày càng phổ biến, việc tổ chức và quản lý dữ liệu hiệu quả trở thành một trong những yếu tố then chốt giúp tối ưu hiệu năng hệ quản trị cơ sở dữ liệu. Một trong các chiến lược quan trọng là phân mảnh dữ liệu – đặc biệt là phân mảnh ngang – cho phép chia nhỏ bảng dữ liệu thành nhiều phân vùng, giúp tăng tốc độ truy xuất, giảm độ trễ và dễ dàng mở rộng hệ thống.

Bài tập lớn này được thực hiện nhằm mục tiêu mô phỏng các kỹ thuật phân mảnh dữ liệu ngang trên hệ quản trị cơ sở dữ liệu mã nguồn mở PostgreSQL. Cụ thể, nhóm chúng tôi triển khai một tập các hàm Python để:

- Tải dữ liệu đánh giá phim từ tập tin ratings.dat (MovieLens),
- Thực hiện phân mảnh theo hai phương pháp: **phân mảnh theo khoảng giá trị (Range Partitioning)** và **phân mảnh theo vòng tròn (Round Robin Partitioning)**,
- Đồng thời phát triển các hàm chèn dữ liệu mới vào đúng phân mảnh tương ứng theo từng phương pháp.

Việc xây dựng các hàm này không chỉ giúp sinh viên nắm vững bản chất của phân mảnh dữ liệu mà còn rèn luyện khả năng thao tác với PostgreSQL bằng ngôn ngữ lập trình Python, từ đó tăng cường kỹ năng triển khai các kỹ thuật xử lý dữ liệu trong thực tiễn.

1. Cấu hình máy ảo

- Bản Ubuntu 22.04 LTS: https://releases.ubuntu.com/jammy/ubuntu-22.04.5-desktop-amd64.iso
- Mở VMware Workstation / Player → "Create a New Virtual Machine".
- Chọn file .iso của Ubuntu vừa tải.
- Cấu hình máy ảo:
  RAM: tối thiểu **4GB** (nên để 8GB nếu máy bạn đủ).
  CPU: 2 cores.
  Ổ cứng: ≥ 20GB (dùng SSD thì nhanh hơn).
- Chạy lệnh cập nhật hệ thống:

```
sudo apt upadate && sudo apt upgrade -y
```

Hình 1-1 Chạy lệnh cập nhật hệ thống

- Cài python 3.12
  - Cài thư viện phụ thuộc:

```
sudo apt install -y wget build-essential libssl-dev zlib1g-dev \libbz2-dev libreadline-dev libsqlite3-dev curl libncursesw5-dev \xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev
```

```
Setting up libxmlsec1-dev (1.2.33-1build2) ...
Setting up build-essential (12.9ubuntu3) ...
Setting up libfontconfig-dev:amd64 (2.13.1-4.2ubuntu5) ...
Setting up libfreetype6-dev:amd64 (2.11.1+dfsg-1ubuntu0.3) ...
Setting up libfontconfig1-dev:amd64 (2.13.1-4.2ubuntu5) ...
Processing triggers for libc-bin (2.35-0ubuntu3.9) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for sgml-base (1.30) ...
Processing triggers for install-info (6.8-4build1) ...
Setting up x11proto-dev (2021.5-1) ...
Setting up libxau-dev:amd64 (1:1.0.9-1build5) ...
Setting up libxdmcp-dev:amd64 (1:1.1.3-0ubuntu5) ...
Setting up x11proto-core-dev (2021.5-1) ...
Setting up libxcb1-dev:amd64 (1.14-3ubuntu3) ...
Setting up libx11-dev:amd64 (2:1.7.5-1ubuntu0.3) ...
Setting up libxext-dev:amd64 (2:1.3.4-1build1) ...
Setting up libxrender-dev:amd64 (1:0.9.10-1build4) ...
Setting up libxft-dev:amd64 (2.3.4-1) ...
Setting up libxss-dev:amd64 (1:1.2.3-1build2) ...
Setting up tk8.6-dev:amd64 (8.6.12-1build1) ...
Setting up tk-dev:amd64 (8.6.11+1build2) ...
```

Hình 1-2 Cài thư viện phụ thuộc cho python 3.12

- Tải mã nguồn Python 3.12.3

```
cd /usr/src

sudo wget https://www.python.org/ftp/python/3.12.3/Python-3.12.3.tgz

sudo tar xzf Python-3.12.3.tgz

cd Python-3.12.3
```

- Biên dịch và cài đặt

```
sudo ./configure --enable-optimizations

sudo make -j$(nproc)

sudo make altinstall
```

- Kết quả:

```
python3.12 –version
```

```
ngan@ngan-virtual-machine:~$ python3.12 --version
Python 3.12.3
```

Hình 1-3 Python 3.12 Version

2. Cài đặt PostgreSQL

- Cài đặt PostgreSQL

```
sudo apt install postgresql postgresql-contrib -y
```

- Khởi động dịch vụ PostgreSQL

```
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

```
fixing permissions on existing directory /var/lib/postgresql/14/main ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Asia/Ho_Chi_Minh
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok
update-alternatives: using /usr/share/postgresql/14/man/man1/postmaster.1.gz to
 provide /usr/share/man/man1/postmaster.1.gz (postmaster.1.gz) in auto mode
Setting up postgresql-contrib (14+238) ...
Setting up postgresql (14+238) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.9) ...
ngan@ngan-virtual-machine:/usr/src/Python-3.12.3$ sudo systemctl start postgres
ql
ngan@ngan-virtual-machine:/usr/src/Python-3.12.3$ sudo systemctl enable postgre
sql
Synchronizing state of postgresql.service with SysV service script with /lib/sy
stemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable postgresql
```

Hình 2-1 Khởi động dịch vụ PostgreSQL

- Tạo user và password cho dự án:

```
sudo -u postgres psql
```

Trong giao diện PostgreSQL

```
ALTER USER postgres WITH PASSWORD '1234';

\q
```

```
ngan@ngan-virtual-machine:~/movielens_project$ sudo -u postgres psql
[sudo] password for ngan:
could not change directory to "/home/ngan/movielens_project": Permission deni
ed
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# ALTER USER postgres with PASSWORD '1234';
ALTER ROLE
postgres=# \q
```

Hình 2-2 Tạo user và password cho dự án

3. Tệp rating.dat

- Tải file zip

```
wget http://files.grouplens.org/datasets/movielens/ml-10m.zip
```

```
ngan@ngan-virtual-machine:~$ wget http://files.grouplens.org/datasets/movielens
/ml-10m.zip
--2025-05-20 17:43:47--  http://files.grouplens.org/datasets/movielens/ml-10m.z
ip
Resolving files.grouplens.org (files.grouplens.org)... 128.101.65.152
Connecting to files.grouplens.org (files.grouplens.org)|128.101.65.152|:80... c
onnected.
HTTP request sent, awaiting response... 200 OK
Length: 65566137 (63M) [application/zip]
Saving to: 'ml-10m.zip'

ml-10m.zip          100%[===================>]  62,53M  88,3KB/s    in 14m 33s

2025-05-20 17:58:20 (73,4 KB/s) - 'ml-10m.zip' saved [65566137/65566137]
```

Hình 3-1 Tải file zip chứa tệp ratings.dat

- Giải nén

```
unzip ml-10m.zip
```

- Kiểm tra dữ liệu

```
head ml-10M100K/ratings.dat
```

```
ngan@ngan-virtual-machine:~$ head ml-10M100K/ratings.dat
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
1::316::5::838983392
1::329::5::838983392
1::355::5::838984474
1::356::5::838983653
1::362::5::838984885
1::364::5::838983707
```

Hình 3-2 Kiểm tra 10 dòng đầu của tệp ratings.dat

4. Cài đặt hàm LoadRatings()

4.1 Mục đích

Hàm loadratings() có nhiệm vụ đọc dữ liệu từ file ratings.dat và nạp vào bảng ratings trong PostgreSQL với cấu trúc gồm ba cột: userid, movieid, và rating. Đây là bước đầu tiên quan trọng nhằm chuẩn bị dữ liệu cho các phương pháp phân mảnh tiếp theo.

4.2 Ý tưởng

- Sử dụng kỹ thuật COPY của PostgreSQL để nạp dữ liệu hàng loạt thay vì thực hiện nhiều lệnh INSERT đơn lẻ giúp tối ưu hiệu suất nhập dữ liệu
- Tiền xử lý dữ liệu bằng StringIO: Dùng bộ đệm bộ nhớ (StringIO) để biến đổi định dạng file từ :: sang định dạng CSV nội bộ (userid,movieid,rating) giúp tương thích với COPY.
- Tăng tốc độ ghi: Vô hiệu hóa ghi đồng bộ (synchronous_commit = OFF) và tăng bộ đệm tạm (temp_buffers, work_mem) để tăng tốc độ import.

4.3 Tiến hành

- Tạo thư mục làm project:

```
cd ~
mkdir movielens_project
cd movielens_project
```

- Tạo file Interface.py. Đảm bảo trong thư mục có chứa đủ 3 file test và file data

```
ngan@ngan-virtual-machine:~/movielens_project$ ls
Assignment1Tester.py  __pycache__  test_data.dat
Interface.py          ratings.dat  testHelper.py
```

Hình 4-1 thư mục có chứa đủ các file test và file data

- Cài thư viện psycopg2

```
sudo apt install libpq-dev python3-psycopg2 -y

pip install psycopg2
```

- Mở file Interface.py và dán đoạn code chứa hàm load_ratings() vào:

```
nano Interface.py
```

- Dán đoạn code sau vào file Interface.py

```
from io import StringIO
import  psycopg2
from psycopg2 import sql, extensions
DATABASE_NAME        = 'dds_assgn1'
RANGE_TABLE_PREFIX   = 'range_part'
RROBIN_TABLE_PREFIX  = 'rrobin_part'
USER_ID_COLNAME      = 'userid'
MOVIE_ID_COLNAME     = 'movieid'
RATING_COLNAME       = 'rating'

def getopenconnection(user='postgres', password='1234', dbname='dds_assgn1'):
    return psycopg2.connect(
        dbname=dbname,
        user=user,
        password=password,
        host='localhost'
    )

def create_db(dbname):
    """
    Tạo cơ sở dữ liệu nếu chưa tồn tại, kết nối đến mặc định 'postgres'
    """
    con = getopenconnection(dbname='postgres')  # Kết nối DB gốc
    con.set_isolation_level(extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    cur = con.cursor()

    # Kiểm tra database đã tồn tại chưa
    cur.execute(
        "SELECT 1 FROM pg_catalog.pg_database WHERE datname = %s;",
        (dbname,)
    )
    exists = cur.fetchone()
    if not exists:
        cur.execute(sql.SQL("CREATE DATABASE
{}").format(sql.Identifier(dbname)))
        print(f"Database '{dbname}' created.")
    else:
        print(f"Database '{dbname}' already exists.")
```

```
    cur.close()
    con.close()

def loadratings(ratingstablename, filepath, openconnection):
    create_db(DATABASE_NAME)
    cur = openconnection.cursor()

    cur.execute("SET synchronous_commit TO OFF;")
    cur.execute("SET temp_buffers TO '64MB';")
    cur.execute("SET work_mem TO '64MB';")

    cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {ratingstablename} (
            userid INT,
            movieid INT,
            rating FLOAT
        );
    """)

    buffer = StringIO()
    with open(filepath, 'r') as file:
        buffer.writelines(
            f"{line.split('::', 3)[0]},{line.split('::',
3)[1]},{line.split('::', 3)[2]}\n"
            for line in file
        )

    buffer.seek(0)

    cur.copy_expert(f"""
        COPY {ratingstablename}(userid, movieid, rating)
        FROM STDIN WITH (FORMAT csv)
    """, buffer)

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

4.4 Thuật toán và giải thích dòng lệnh

- **Khai báo thư viện và hằng số cấu hình**

```
from io import StringIO
import  psycopg2
from psycopg2 import sql, extensions
DATABASE_NAME         = 'dds_assgn1'
RANGE_TABLE_PREFIX    = 'range_part'
RROBIN_TABLE_PREFIX   = 'rrobin_part'
USER_ID_COLNAME       = 'userid'
```

```
MOVIE_ID_COLNAME     = 'movieid'
RATING_COLNAME       = 'rating'
```

- **Hàm getopenconnection(...)**

```
def getopenconnection(user='postgres', password='1234', dbname='dds_assgn1'):
    return psycopg2.connect(
        dbname=dbname,
        user=user,
        password=password,
        host='localhost'
    )
```

Tạo và trả về một kết nối tới cơ sở dữ liệu PostgreSQL bằng thư viện psycopg2. Dùng mặc định:

- Tên người dùng (postgres)
- Mật khẩu (1234)
- Tên cơ sở dữ liệu (dds_assgn1)
- Host (cục bộ – localhost)

- **Hàm create_db(...)**

```
def create_db(dbname):
    """
    Tạo cơ sở dữ liệu nếu chưa tồn tại, kết nối đến mặc định 'postgres'
    """
    con = getopenconnection(dbname='postgres')  # Kết nối DB gốc
    con.set_isolation_level(extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    cur = con.cursor()

    # Kiểm tra database đã tồn tại chưa
    cur.execute(
        "SELECT 1 FROM pg_catalog.pg_database WHERE datname = %s;",
        (dbname,)
    )
    exists = cur.fetchone()
    if not exists:
        cur.execute(sql.SQL("CREATE DATABASE
{}").format(sql.Identifier(dbname)))
        print(f"Database '{dbname}' created.")
    else:
        print(f"Database '{dbname}' already exists.")

    cur.close()
    con.close()
```

Chức năng:

- Tạo cơ sở dữ liệu mới nếu chưa tồn tại.
- Kết nối tới CSDL mặc định postgres, dùng để thao tác meta-level (tạo DB).

Bảo mật:

- Sử dụng sql.Identifier(...) đảm bảo chống lỗi cú pháp hoặc SQL injection.

Tăng hiệu năng:

- ISOLATION_LEVEL_AUTOCOMMIT giúp chạy lệnh CREATE DATABASE ngoài transaction block (bắt buộc với PostgreSQL).

- **Hàm loadratings(...)**

```
def loadratings(ratingstablename, filepath, openconnection):
```

Tham số:

- ratingstablename: tên bảng sẽ được tạo ra hoặc chèn dữ liệu vào.
- filepath: đường dẫn tới file ratings.dat.
- openconnection: đối tượng kết nối đang mở tới PostgreSQL.

```
create_db(DATABASE_NAME)
```

Gọi hàm create_db() để tạo database nếu cần.

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

Tạo con trỏ để tương tác CSDL: con trỏ (cursor) thực thi các câu lệnh SQL trên kết nối cơ sở dữ liệu.

```
    cur.execute("SET synchronous_commit TO OFF;")
    cur.execute("SET temp_buffers TO '64MB';")
    cur.execute("SET work_mem TO '64MB';")
```

Tối ưu hóa hiệu năng nhập dữ liệu:

- `synchronous_commit TO OFF`: tắt ghi đồng bộ để tăng tốc độ insert (chấp nhận mất an toàn nếu mất điện).
- `temp_buffers` và `work_mem`: tăng kích thước bộ nhớ tạm thời để xử lý dữ liệu lớn nhanh hơn khi import hàng loạt.

```
cur.execute(f"""
        CREATE TABLE IF NOT EXISTS {ratingstablename} (
            userid INT,
            movieid INT,
            rating FLOAT
        );
    """)
```

Tạo bảng ratings nếu chưa có với 3 cột chính, bỏ qua timestamp

```
    buffer = StringIO()
    with open(filepath, 'r') as file:
        buffer.writelines(
            f"{line.split('::', 3)[0]},{line.split('::',
3)[1]},{line.split('::', 3)[2]}\n"
            for line in file
        )

    buffer.seek(0)
```

Đọc dữ liệu từ file và chuẩn hóa:

- Đọc từng dòng từ ratings.dat
- Bỏ cột thứ 4 (timestamp)
- Chuyển thành CSV: userid, movieid, rating
- Ghi vào buffer trong RAM

Sử dụng StringIO thay vì tạo file tạm trên ổ đĩa → hiệu năng cao hơn.

```
    cur.copy_expert(f"""
        COPY {ratingstablename}(userid, movieid, rating)
        FROM STDIN WITH (FORMAT csv)
    """, buffer)
```

Chèn dữ liệu vào PostgreSQL. Dùng lệnh COPY ... FROM STDIN để chèn dữ liệu cực nhanh từ buffer.

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

Ghi dữ liệu xuống đĩa. Đóng con trỏ để giải phóng tài nguyên.

4.5 Kết quả thực thi

- Test:

```
Database 'dds_assgn1' already exists.
loadratings function pass! - Time: 20.9810s
```

Hình 4-2 loadratings fuction pass

- Kiểm tra trong CSDL:

```
sudo -u postgres psql dds_assgn1

\dt

SELECT * FROM ratings LIMIT 5;

SELECT COUNT(*) FROM ratings;

\q
```

```
ngan@ngan-virtual-machine:~/movielens_project$ sudo -u postgres psql dds_assgn1
[sudo] password for ngan:
could not change directory to "/home/ngan/movielens_project": Permission denied
psql (14.18 (Ubuntu 14.18-0ubuntu0.22.04.1))
Type "help" for help.

dds_assgn1=# \dt
          List of relations
 Schema |  Name   | Type  |  Owner
--------+---------+-------+----------
 public | ratings | table | postgres
(1 row)

dds_assgn1=# SELECT * FROM ratings LIMIT 5;
 userid | movieid | rating
--------+---------+--------
      1 |     122 |      5
      1 |     185 |      5
      1 |     231 |      5
      1 |     292 |      5
      1 |     316 |      5
(5 rows)

dds_assgn1=# SELECT COUNT(*) FROM ratings;
  count
----------
 10000054
(1 row)

dds_assgn1=# \q
```

Hình 4-3 Kiểm tra cơ sở dữ liệu thực sau hàm loadratings

5. Cài đặt hàm Range_Partition()

5.1 Mục đích

Mục đích của hàm Range_Partition() là phân mảnh ngang bảng Ratings trong cơ sở dữ liệu PostgreSQL thành N phân mảnh dựa trên thuộc tính rating. Mỗi phân mảnh sẽ lưu một đoạn

giá trị rating đều nhau trong khoảng từ 0.0 đến 5.0. Điều này giúp tăng hiệu năng truy vấn và xử lý dữ liệu.

5.2 Ý tưởng

Hàm Range_Partition() thực hiện phân mảnh ngang bảng Ratings dựa trên thuộc tính rating, chia toàn bộ miền giá trị [0.0, 5.0] thành N đoạn đều nhau. Mỗi đoạn tương ứng với một bảng phân mảnh riêng biệt trong cơ sở dữ liệu, có tên lần lượt là range_part0, range_part1, ..., range_part(N-1).

1. Quy tắc phân chia miền giá trị

- Toàn bộ miền rating được chia thành N khoảng liên tiếp, mỗi khoảng có độ rộng là:

$$\delta = \frac{5.0 - 0.0}{N}$$

- Các khoảng này lần lượt có dạng:
    - Phân mảnh đầu tiên (range_part0) chứa khoảng **[lower, upper]** — bao gồm cả biên trái và phải.
    - Các phân mảnh tiếp theo (range_part1 đến range_part(N-1)) chứa khoảng **(lower, upper]** — loại trừ biên trái để tránh trùng lặp dữ liệu giữa các phân mảnh.

2. Các bước thực hiện

3. Xóa các bảng phân mảnh cũ (nếu có):
   Truy vấn tất cả các bảng có tên bắt đầu bằng range_part, sau đó xóa toàn bộ các bảng này để đảm bảo quá trình phân mảnh là sạch và chính xác.

4. Tính toán độ rộng mỗi phân mảnh (delta):
   Xác định độ rộng mỗi khoảng delta dựa trên số phân mảnh được yêu cầu, đảm bảo phân chia đều trên miền [0.0, 5.0].

5. Duyệt qua từng phân mảnh từ 0 đến N-1:
    - Tính toán biên dưới lower và biên trên upper cho từng khoảng.
    - Tạo câu truy vấn SELECT để lọc các bản ghi trong bảng Ratings có rating thuộc khoảng [lower, upper] hoặc (lower, upper] tùy vào vị trí phân mảnh.
    - Tạo bảng range_part<i> tương ứng và chèn vào đó các dòng dữ liệu thỏa mãn điều kiện rating.

Việc phân mảnh theo khoảng giúp tổ chức dữ liệu hợp lý và hiệu quả hơn, đặc biệt trong các truy vấn có điều kiện theo rating. Điều này cải thiện hiệu năng hệ thống và là bước chuẩn bị cho các thao tác chèn và truy vấn về sau.

5.3 Tiến hành

- Mở file Interface.py và dán đoạn code chứa hàm rangepartition(...) vào:

```
nano Interface.py
```

- Dán đoạn code sau vào file Interface.py

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    cur = openconnection.cursor()
    RANGE_TABLE_PREFIX = 'range_part'
    cur.execute(f"SELECT tablename FROM pg_tables WHERE tablename LIKE
'{RANGE_TABLE_PREFIX}%'")
    drop_tables = [row[0] for row in cur.fetchall()]
    if drop_tables:
        cur.execute("DROP TABLE IF EXISTS " + ", ".join(drop_tables))
    min_rating = 0.0
    max_rating = 5.0
    delta = float(max_rating - min_rating) / numberofpartitions
    for i in range(numberofpartitions):
        lower = min_rating + i * delta
        upper = lower + delta
        part_table = f"{RANGE_TABLE_PREFIX}{i}"
        if i == 0:
            where_clause = f"rating >= {lower} AND rating <= {upper}"
        else:
            where_clause = f"rating > {lower} AND rating <= {upper}"
        cur.execute(
            f"CREATE TABLE {part_table} AS "
            f"SELECT userid, movieid, rating FROM {ratingstablename} WHERE
{where_clause}"
        )
    cur.close()
```

5.4 Thuật toán và giải thích dòng lệnh

```
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    cur = openconnection.cursor()
```

Mở con trỏ để thao tác với cơ sở dữ liệu.

```
    RANGE_TABLE_PREFIX = 'range_part'
    cur.execute(f"SELECT tablename FROM pg_tables WHERE tablename LIKE
'{RANGE_TABLE_PREFIX}%'")
```

Tìm tất cả bảng có tiền tố range_part để xóa đi.

```
    drop_tables = [row[0] for row in cur.fetchall()]
    if drop_tables:
```

```
        cur.execute("DROP TABLE IF EXISTS " + ", ".join(drop_tables))
```
Tạo danh sách các bảng cần xóa và thực hiện xóa nếu có.

```
    min_rating = 0.0
    max_rating = 5.0
    delta = float(max_rating - min_rating) / numberofpartitions
```
Đặt miền phân mảnh và tính độ rộng mỗi phân mảnh.

```
    for i in range(numberofpartitions):
        lower = min_rating + i * delta
        upper = lower + delta
        part_table = f"{RANGE_TABLE_PREFIX}{i}"
```
Lặp qua từng phân mảnh, xác định ranh giới và tên bảng.

```
        if i == 0:
            where_clause = f"rating >= {lower} AND rating <= {upper}"
        else:
            where_clause = f"rating > {lower} AND rating <= {upper}"
```
Phân biệt điều kiện chèn dòng giữa phân mảnh đầu tiên và các phân mảnh còn lại.

```
        cur.execute(
            f"CREATE TABLE {part_table} AS "
            f"SELECT userid, movieid, rating FROM {ratingstablename} WHERE
{where_clause}"
        )
```
Tạo bảng phân mảnh chứa dữ liệu thỏa mãn điều kiện rating

```
    cur.close()
```
Đóng con trỏ để giải phóng tài nguyên.

5.5 Kết quả thực thi

- Test:

```
rangepartition function pass! - Time: 20.0614s
```

Hình 5-1 rangepartition fuction pass

- Kiểm tra trong cơ sở dữ liệu:

```
sudo -u postgres psql dds_assgn1
```

```
\dt
SELECT MIN(rating), MAX(rating) FROM range_part0;
SELECT MIN(rating), MAX(rating) FROM range_part1;
\q
```

```
dds_assgn1=# \dt
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | range_part0 | table | postgres
 public | range_part1 | table | postgres
 public | range_part2 | table | postgres
 public | range_part3 | table | postgres
 public | range_part4 | table | postgres
 public | ratings      | table | postgres
(6 rows)

dds_assgn1=# SELECT MIN(rating), MAX(rating) FROM range_part0;
 min | max
-----+-----
 0.5 |   1
(1 row)

dds_assgn1=# SELECT MIN(rating), MAX(rating) FROM range_part1;
 min | max
-----+-----
 1.5 |   2
(1 row)

dds_assgn1=# \q
```

Hình 5-2 Kiểm tra cơ sở dữ liệu thực sau hàm range_partition

6. Cài đặt hàm RoundRobin_Partition()

6.1 Mục đích

Hàm RoundRobin_Partition() thực hiện phân mảnh ngang bảng Ratings bằng phương pháp **vòng tròn (round-robin)**. Mục tiêu là chia đều các bản ghi trong bảng gốc thành N phân mảnh có tên rrobin_part0, rrobin_part1, ..., rrobin_part(N-1), sao cho thứ tự phân phối tuân theo nguyên tắc luân phiên từng dòng, đảm bảo cân bằng số lượng dòng giữa các phân mảnh.

6.2 Ý tưởng

Hàm RoundRobin_Partition() thực hiện phân mảnh dữ liệu theo phương pháp vòng tròn (round-robin) — một chiến lược phân phối dữ liệu đều đặn và tuần tự qua nhiều phân mảnh mà không phụ thuộc vào giá trị của bất kỳ thuộc tính nào (như rating).

Toàn bộ bản ghi từ bảng gốc Ratings sẽ được gán thứ tự duy nhất, bắt đầu từ 0. Mỗi bản ghi thứ i sẽ được đưa vào phân mảnh có chỉ số:

$$partition\_index = i \bmod N$$

Trong đó:

- i là chỉ số dòng (từ 0).
- N là tổng số phân mảnh cần tạo.
- rrobin_part<partition_index> là tên bảng phân mảnh tương ứng.

Nhờ đó, dữ liệu được phân phối theo kiểu "vòng quay", lần lượt luân chuyển giữa các phân mảnh, đảm bảo sự cân bằng số lượng dòng giữa các bảng phân mảnh mà không xét đến nội dung cụ thể của từng bản ghi.

1. Tạo N bảng phân mảnh mới:

- Các bảng có tên dạng rrobin_part0, rrobin_part1, ..., rrobin_part(N-1).
- Mỗi bảng có cấu trúc giống với bảng Ratings (gồm userid, movieid, rating).
- Dùng UNLOGGED TABLE để giảm chi phí ghi log và tăng tốc độ xử lý tạm thời (phù hợp trong môi trường kiểm thử).

2. Đánh số thứ tự cho các bản ghi trong bảng Ratings:

- Dùng hàm cửa sổ ROW_NUMBER() trong SQL để tạo bảng tạm numbered.
- Mỗi bản ghi được gán một số thứ tự rn, bắt đầu từ 0.

3. Phân phối bản ghi vào phân mảnh phù hợp:

- Với mỗi phân mảnh i ∈ [0, N-1], lựa chọn những bản ghi có rn % N = i từ bảng tạm numbered.
- Chèn các bản ghi này vào bảng rrobin_part<i> tương ứng.

Phương pháp này đảm bảo tính phân phối đồng đều dữ liệu giữa các phân mảnh và không bị phụ thuộc vào giá trị của thuộc tính nào, đặc biệt hữu ích trong các hệ thống cần xử lý tải đồng đều hoặc khi không có thuộc tính rõ ràng để phân vùng.

6.3 Tiến hành

- Mở file Interface.py và dán đoạn code chứa hàm roundrobinpartition (...) vào:

```
nano Interface.py
```

- Dán đoạn code sau vào file Interface.py

```
def roundrobinpartition(ratingstablename: str, numberofpartitions: int,
openconnection):
    if numberofpartitions <= 0:
        raise ValueError('numberofpartitions phải > 0')
    cur = openconnection.cursor()
```

```
    cur.execute(
        "SELECT table_name FROM information_schema.tables "
        "WHERE table_schema='public' AND table_name LIKE %s;",
        (f'{RROBIN_TABLE_PREFIX}%',)
    )
    for tbl, in cur.fetchall():
        cur.execute(f'DROP TABLE IF EXISTS "{tbl}" CASCADE;')
    cur.execute("SET synchronous_commit TO OFF;")
    for i in range(numberofpartitions):
        cur.execute(f"""
            CREATE UNLOGGED TABLE {RROBIN_TABLE_PREFIX}{i} (
                {USER_ID_COLNAME}  INT,
                {MOVIE_ID_COLNAME} INT,
                {RATING_COLNAME}   FLOAT
            );
        """)
    cur.execute(f"""
        CREATE TEMPORARY TABLE numbered AS
        SELECT {USER_ID_COLNAME}, {MOVIE_ID_COLNAME}, {RATING_COLNAME},
               ROW_NUMBER() OVER () - 1 AS rn
        FROM {ratingstablename};
    """)
    for i in range(numberofpartitions):
        cur.execute(f"""
            INSERT INTO {RROBIN_TABLE_PREFIX}{i}
            SELECT {USER_ID_COLNAME}, {MOVIE_ID_COLNAME}, {RATING_COLNAME}
            FROM numbered
            WHERE (rn % {numberofpartitions}::BIGINT) = {i};
        """)
    cur.execute("DROP TABLE numbered;")
    openconnection.commit()
    cur.close()
```

6.4 Thuật toán và giải thích dòng lệnh

```
def roundrobinpartition(ratingstablename: str, numberofpartitions: int,
openconnection):
    if numberofpartitions <= 0:
        raise ValueError('numberofpartitions phải > 0')
```

Ràng buộc đầu vào: đảm bảo số phân mảnh là hợp lệ.

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

Mở con trỏ thao tác cơ sở dữ liệu.

```
    cur.execute(
        "SELECT table_name FROM information_schema.tables "
        "WHERE table_schema='public' AND table_name LIKE %s;",
        (f'{RROBIN_TABLE_PREFIX}%',)
    )
```
Lấy danh sách các bảng hiện tại có tên bắt đầu bằng rrobin_part.

```
    for tbl, in cur.fetchall():
        cur.execute(f'DROP TABLE IF EXISTS "{tbl}" CASCADE;')
```
Xóa toàn bộ các bảng phân mảnh cũ nếu tồn tại.

```
    cur.execute("SET synchronous_commit TO OFF;")
```
Tắt đồng bộ hóa để tăng tốc độ ghi (không ảnh hưởng đến tính chính xác trong môi trường kiểm thử).

```
    for i in range(numberofpartitions):
        cur.execute(f"""
            CREATE UNLOGGED TABLE {RROBIN_TABLE_PREFIX}{i} (
                {USER_ID_COLNAME}  INT,
                {MOVIE_ID_COLNAME} INT,
                {RATING_COLNAME}   FLOAT
            );
        """)
```
Tạo các bảng phân mảnh mới, dùng UNLOGGED để tăng tốc độ (vì không cần ghi log WAL).

```
    cur.execute(f"""
        CREATE TEMPORARY TABLE numbered AS
        SELECT {USER_ID_COLNAME}, {MOVIE_ID_COLNAME}, {RATING_COLNAME},
               ROW_NUMBER() OVER () - 1 AS rn
        FROM {ratingstablename};
    """)
```
Tạo bảng tạm numbered, trong đó mỗi bản ghi được gán chỉ số rn dùng để xác định thứ tự phân mảnh.

```
    for i in range(numberofpartitions):
        cur.execute(f"""
            INSERT INTO {RROBIN_TABLE_PREFIX}{i}
            SELECT {USER_ID_COLNAME}, {MOVIE_ID_COLNAME}, {RATING_COLNAME}
            FROM numbered
            WHERE (rn % {numberofpartitions}::BIGINT) = {i};
        """)
```

Chèn dữ liệu vào từng phân mảnh dựa trên điều kiện chia theo số dòng (mod).

```
    cur.execute("DROP TABLE numbered;")
    openconnection.commit()
    cur.close()
```

Dọn dẹp bảng tạm và ghi thay đổi vào CSDL, sau đó đóng con trỏ.

6.5 Kết quả thực thi

- Test:

```
roundrobinpartition function pass! - Time: 38.2448s
```

Hình 6-1 roundrobinpartition fuction pass

- Kiểm tra trong cơ sở dữ liệu

```
sudo -u postgres psql dds_assgn1

\dt rrobin_part*

\q
```

```
dds_assgn1=# \dt rrobin_part*
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | rrobin_part0 | table | postgres
 public | rrobin_part1 | table | postgres
 public | rrobin_part2 | table | postgres
 public | rrobin_part3 | table | postgres
 public | rrobin_part4 | table | postgres
(5 rows)
```

Hình 6-2 Kiểm tra cơ sở dữ liệu thực sau hàm roundrobin_partition

7. Cài đặt hàm RoundRobin_Insert()

7.1 Mục đích

Hàm RoundRobin_Insert() có mục tiêu đảm bảo rằng mỗi bản ghi mới (gồm userid, movieid, rating) được:

- Chèn vào bảng gốc Ratings để lưu toàn bộ dữ liệu đầu vào.
- Chèn vào đúng bảng phân mảnh rrobin_part<i> theo nguyên tắc vòng tròn (round-robin), kế thừa trạng thái phân phối từ lần chèn trước đó.

Hàm này đảm bảo tính nhất quán với cách phân mảnh đã được thiết lập bởi RoundRobin_Partition(), đồng thời tuân thủ yêu cầu không sử dụng biến toàn cục bằng cách lưu trữ thông tin phân mảnh tiếp theo trong bảng meta-data rrobin_meta.

7.2 Ý tưởng

Hàm RoundRobin_Insert() được thiết kế để chèn một bản ghi mới vào hệ thống đã được phân mảnh theo phương pháp vòng tròn (round-robin). Để đảm bảo sự phân phối tuần tự và chính xác của các bản ghi giữa các phân mảnh rrobin_part<i>, hàm sử dụng một **bảng meta-data** có tên rrobin_meta để lưu trữ **trạng thái lần chèn gần nhất**.

Thay vì đếm tổng số dòng hiện có trong các bảng phân mảnh (vốn có thể tốn kém và không chính xác trong môi trường đa luồng), hệ thống sử dụng một biến quản lý duy nhất là next_part, lưu trong bảng rrobin_meta. Biến này chỉ ra bảng phân mảnh tiếp theo sẽ nhận bản ghi mới.

- Cơ chế phân phối bản ghi

- Trường next_part trong bảng rrobin_meta là một con trỏ chỉ đến phân mảnh kế tiếp sẽ được sử dụng.

- Sau mỗi lần chèn, next_part được cập nhật theo công thức:

$$next\_part_{mới} = (next_{part_{cũ}} + 1) \ mod \ N$$

Trong đó N là tổng số phân mảnh hiện có.

Cách tiếp cận này mang lại các lợi ích sau:

- Tối ưu hiệu năng: Không cần quét dữ liệu để tính toán phân mảnh.

- Đảm bảo chính xác: Tránh việc phân mảnh sai khi có nhiều thao tác chèn liên tiếp hoặc song song.

- Dễ mở rộng: Việc duy trì trạng thái trong bảng rrobin_meta giúp dễ quản lý và không phụ thuộc vào biến toàn cục.

Các bước thực hiện cụ thể

1. Chèn bản ghi mới vào bảng Ratings để lưu trữ toàn bộ dữ liệu đầu vào một cách đầy đủ.
2. Xác định số lượng phân mảnh N bằng cách đếm các bảng có tên bắt đầu bằng rrobin_part.
3. Tạo bảng rrobin_meta nếu chưa tồn tại. Bảng này chỉ chứa một dòng duy nhất với khóa chính mặc định.
4. Truy vấn giá trị next_part hiện tại từ rrobin_meta để xác định bảng phân mảnh cần chèn.
5. Chèn bản ghi vào bảng rrobin_part<next_part> tương ứng.

6. Cập nhật lại giá trị next_part trong rrobin_meta bằng cách tăng lên 1 (và lấy modulo với N), để đảm bảo phân phối vòng tròn tiếp theo.

7.3 Tiến hành

- Mở file Interface.py và dán đoạn code chứa hàm roundrobininsert (...) vào:

```
nano Interface.py
```

- Dán đoạn code sau vào file Interface.py

```
def roundrobininsert(ratingstablename: str,
                     userid: int, movieid: int, rating: float,
                     openconnection):

    cur = openconnection.cursor()

    cur.execute(f"""
        INSERT INTO {ratingstablename} ({USER_ID_COLNAME},
                                        {MOVIE_ID_COLNAME},
                                        {RATING_COLNAME})
        VALUES (%s, %s, %s);
    """, (userid, movieid, rating))
    cur.execute("""
        SELECT COUNT(*)
        FROM   information_schema.tables
        WHERE  table_schema='public'
          AND  table_name LIKE %s;
    """, (f'{RROBIN_TABLE_PREFIX}%',))
    n = cur.fetchone()[0]
    if n == 0:
        raise RuntimeError('Phải gọi roundrobinpartition trước!')
    cur.execute("""
        CREATE TABLE IF NOT EXISTS rrobin_meta (
            dummy     INT PRIMARY KEY DEFAULT 1,
            next_part INT
        );
    """)
    cur.execute("SELECT next_part FROM rrobin_meta FOR UPDATE;")
    row = cur.fetchone()
    next_part = 0 if row is None else row[0]
    partition_tbl = f'{RROBIN_TABLE_PREFIX}{next_part}'
    cur.execute(f"""
        INSERT INTO {partition_tbl} ({USER_ID_COLNAME},
                                     {MOVIE_ID_COLNAME},
                                     {RATING_COLNAME})
        VALUES (%s, %s, %s);
    """, (userid, movieid, rating))
    cur.execute("""
```

```
        INSERT INTO rrobin_meta (next_part)
        VALUES (%s)
        ON CONFLICT (dummy)
        DO UPDATE SET next_part = (%s + 1) %% %s;
    """, ( (next_part + 1) % n, next_part, n ))

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

7.4 Thuật toán và giải thích dòng lệnh

```
def roundrobininsert(ratingstablename: str,
                     userid: int, movieid: int, rating: float,
                     openconnection):
    cur = openconnection.cursor()
```

Mở con trỏ thao tác cơ sở dữ liệu.

```
    cur.execute(f"""
        INSERT INTO {ratingstablename} ({USER_ID_COLNAME},
                                        {MOVIE_ID_COLNAME},
                                        {RATING_COLNAME})
        VALUES (%s, %s, %s);
    """, (userid, movieid, rating))
```

Chèn bản ghi vào bảng chính Ratings.

```
    cur.execute("""
        SELECT COUNT(*)
        FROM   information_schema.tables
        WHERE  table_schema='public'
          AND  table_name LIKE %s;
    """, (f'{RROBIN_TABLE_PREFIX}%',))
    n = cur.fetchone()[0]
    if n == 0:
        raise RuntimeError('Phải gọi roundrobinpartition trước!')
```

Lấy số phân mảnh n. Nếu chưa có phân mảnh, báo lỗi.

```
    cur.execute("""
        CREATE TABLE IF NOT EXISTS rrobin_meta (
            dummy     INT PRIMARY KEY DEFAULT 1,
            next_part INT
        );
    """)
```

Tạo bảng meta để lưu vị trí phân mảnh kế tiếp.

```
    cur.execute("SELECT next_part FROM rrobin_meta FOR UPDATE;")
    row = cur.fetchone()
```

Khóa truy cập và đọc next_part hiện tại để xác định phân mảnh sẽ chèn.

```
    next_part = 0 if row is None else row[0]
    partition_tbl = f'{RROBIN_TABLE_PREFIX}{next_part}'
    cur.execute(f"""
        INSERT INTO {partition_tbl} ({USER_ID_COLNAME},
                                     {MOVIE_ID_COLNAME},
                                     {RATING_COLNAME})
        VALUES (%s, %s, %s);
    """, (userid, movieid, rating))
```

Chèn bản ghi vào bảng phân mảnh đúng theo vòng.

```
    cur.execute("""
        INSERT INTO rrobin_meta (next_part)
        VALUES (%s)
        ON CONFLICT (dummy)
        DO UPDATE SET next_part = (%s + 1) %% %s;
    """, ( (next_part + 1) % n, next_part, n ))
```

Cập nhật giá trị next_part mới vào bảng meta, đảm bảo quay vòng qua n phân mảnh.

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

Ghi thay đổi vào cơ sở dữ liệu và giải phóng tài nguyên.

7.5 Kết quả thực thi

- Test:

```
roundrobininsert function pass! - Time: 0.1856s
```

Hình 7-1 roundrobininsert fuction pass

- Kiểm tra trong cơ sở dữ liệu:

```
sudo -u postgres psql dds_assgn1

\dt

SELECT * FROM rrobin_part0 WHERE userid = 100 AND movieid = 1 AND rating = 3;

\q
```

```
dds_assgn1=# \dt rrobin_part*
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | rrobin_part0 | table | postgres
 public | rrobin_part1 | table | postgres
 public | rrobin_part2 | table | postgres
 public | rrobin_part3 | table | postgres
 public | rrobin_part4 | table | postgres
(5 rows)

dds_assgn1=# SELECT * FROM rrobin_part0 WHERE userid = 100 AND movieid = 1 AND ra
ting = 3;
 userid | movieid | rating
--------+---------+--------
    100 |       1 |      3
(1 row)

dds_assgn1=# \q
```

Hình 7-2 Kiểm tra cơ sở dữ liệu thực sau hàm roundrobin_insert

8. Cài đặt hàm Range_Insert()

8.1 Mục đích

Hàm Range_Insert() có mục đích chèn một bộ dữ liệu mới (gồm userid, movieid, rating) vào:

- Bảng chính Ratings trong cơ sở dữ liệu.
- Đồng thời, chèn bản ghi này vào **đúng phân mảnh** range_part<i> dựa trên giá trị rating, tuân theo phương pháp phân mảnh theo khoảng đã được thiết lập bởi Range_Partition().

8.2 Ý tưởng

Hàm Range_Insert() được xây dựng dựa trên nguyên lý phân mảnh theo khoảng (range partitioning) đã được thiết lập trong hàm Range_Partition(). Ý tưởng cốt lõi là đảm bảo rằng mỗi bản ghi mới được chèn vào không chỉ lưu trong bảng chính Ratings, mà còn được phân phối chính xác vào một trong các bảng phân mảnh range_part<i>, tương ứng với khoảng giá trị rating của bản ghi đó.

Quá trình thực hiện bao gồm các bước sau:

1. Chèn bản ghi vào bảng gốc **Ratings**:
   Thao tác này nhằm duy trì tính toàn vẹn và đầy đủ dữ liệu trong bảng chính, nơi chứa toàn bộ dữ liệu trước khi phân mảnh.
2. Xác định số lượng phân mảnh hiện có:
   Truy vấn hệ thống để đếm số bảng có tên bắt đầu bằng range_part. Đây là cơ sở để tính toán độ rộng mỗi phân mảnh (delta).

3. Tính toán độ rộng mỗi phân mảnh (**delta**):
   Vì miền giá trị của rating được xác định trong đoạn [0.0, 5.0], nên delta được tính bằng công thức:

$$\delta = \frac{5.0 - 0.0}{N}$$

   với N là số lượng phân mảnh đã được tạo trước đó.

4. Xác định phân mảnh phù hợp với giá trị **rating**:
   - Nếu rating nằm trong khoảng đầu tiên [0.0, delta], bản ghi sẽ được chèn vào bảng range_part0.
   - Với các giá trị còn lại, chỉ số phân mảnh được tính bằng công thức:

$$index = \left\lceil \frac{rating - \epsilon}{\delta} \right\rceil$$

     Trong đó, $\epsilon = 10^{-8}$ là một giá trị rất nhỏ dùng để xử lý sai số số thực, đảm bảo rằng biên trái không bị rơi vào phân mảnh sau.

   - Nếu rating = 5.0 (trường hợp biên phải cuối cùng), chỉ số phân mảnh được điều chỉnh về N - 1 để tránh vượt giới hạn mảng.
5. Chèn bản ghi vào bảng phân mảnh tương ứng:
   Bản ghi sẽ được chèn vào bảng range_part<index>, nơi chứa các giá trị rating nằm trong khoảng tương ứng.

Việc sử dụng lại logic phân mảnh trong hàm này giúp duy trì tính đồng nhất dữ liệu, đảm bảo mọi thao tác chèn mới đều tuân theo quy tắc phân mảnh đã định nghĩa từ đầu, hỗ trợ hiệu quả cho việc truy vấn, tối ưu hiệu năng và kiểm thử dễ dàng hơn.

8.3 Tiến hành

- Mở file Interface.py và dán đoạn code chứa hàm rangepartition(...) vào:

```
nano Interface.py
```

- Dán đoạn code sau vào file Interface.py

```
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    cur = openconnection.cursor()
    cur.execute(
        f"INSERT INTO {ratingstablename} (userid, movieid, rating) VALUES
(%s, %s, %s)",
        (userid, itemid, rating)
    )
    cur.execute("SELECT COUNT(*) FROM pg_tables WHERE tablename LIKE
'range_part%'")
```

```
    numberofpartitions = cur.fetchone()[0]
    min_rating = 0.0
    max_rating = 5.0
    delta = (max_rating - min_rating) / numberofpartitions
    if rating <= min_rating + delta:
        index = 0
    else:
        index = int((rating - min_rating - 1e-8) // delta)
        if index >= numberofpartitions:
            index = numberofpartitions - 1
    part_table = f"range_part{index}"
    cur.execute(
        f"INSERT INTO {part_table} (userid, movieid, rating) VALUES (%s, %s,
%s)",
        (userid, itemid, rating)
    )
    cur.close()
```

8.4 Thuật toán và giải thích dòng lệnh

```
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    cur = openconnection.cursor()
```

Mở con trỏ để thao tác cơ sở dữ liệu.

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

Chèn bản ghi mới vào bảng chính Ratings.

```
    cur.execute("SELECT COUNT(*) FROM pg_tables WHERE tablename LIKE
'range_part%'")
    numberofpartitions = cur.fetchone()[0]
```

Đếm số lượng bảng phân mảnh range_part, từ đó xác định N

```
    min_rating = 0.0
    max_rating = 5.0
    delta = (max_rating - min_rating) / numberofpartitions
```

Thiết lập miền phân mảnh và độ rộng mỗi khoảng.

```
    if rating <= min_rating + delta:
        index = 0
```
Nếu rating thuộc khoảng đầu tiên [0.0, delta], đưa vào range_part0.

```
    else:
        index = int((rating - min_rating - 1e-8) // delta)
```
Nếu không, dùng công thức int((rating - ε) // delta) để tính chỉ số phân mảnh, ε = 1e-8 giúp tránh sai số làm rơi vào phân mảnh sai do giới hạn số thực.

```
        if index >= numberofpartitions:
            index = numberofpartitions - 1
```
Bảo vệ: không để index vượt giới hạn (trong trường hợp rating = 5.0).

```
    part_table = f"range_part{index}"
    cur.execute(
        f"INSERT INTO {part_table} (userid, movieid, rating) VALUES (%s, %s, %s)",
        (userid, itemid, rating)
    )
```
Bảo vệ: không để index vượt giới hạn (trong trường hợp rating = 5.0).

```
    cur.close()
```
Đóng con trỏ sau khi hoàn tất thao tác.

8.5 Kết quả thực thi

- Test

```
rangeinsert function pass! - Time: 0.2349s
```

Hình 8-1 rangeinsert fuction pass

- Kiểm tra CSDL:

```
sudo -u postgres psql dds_assgn1

\dt

SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2 AND rating = 3;

\q
```

```
dds_assgn1=# SELECT * FROM range_part2 WHERE userid = 100 AND movieid = 2 AND rat
ing = 3;
 userid | movieid | rating
--------+---------+--------
    100 |       2 |      3
(1 row)

dds_assgn1=# \q
```

Hình 8-2 Kiểm tra cơ sở dữ liệu thực sau hàm range_insert

9. Kết luận

Thông qua bài tập lớn này, nhóm chúng tôi đã tiếp cận và triển khai thành công các kỹ thuật phân mảnh dữ liệu ngang trên hệ quản trị cơ sở dữ liệu PostgreSQL. Cụ thể, chúng tôi đã xây dựng và thử nghiệm hai phương pháp phân mảnh phổ biến: **phân mảnh theo khoảng giá trị (Range Partitioning)** và **phân mảnh vòng tròn (Round Robin Partitioning)**, cùng với các hàm chèn dữ liệu mới sao cho đảm bảo đúng nguyên tắc phân phối đã thiết lập.

Việc kết hợp giữa Python và PostgreSQL trong quá trình lập trình giúp chúng tôi hiểu rõ hơn về:

- Cơ chế tổ chức và chia tách dữ liệu trong các hệ thống lớn,
- Ảnh hưởng của phương pháp phân mảnh đến hiệu năng truy vấn và quản lý,
- Cách sử dụng hiệu quả các công cụ như câu lệnh SQL, bảng metadata và hàm cửa sổ (ROW_NUMBER()) để xử lý dữ liệu linh hoạt.

Ngoài ra, nhóm cũng rút ra được những bài học thực tiễn quan trọng như: đảm bảo tính đồng nhất trong thiết kế dữ liệu, hạn chế việc phụ thuộc vào biến toàn cục, và sử dụng bảng trung gian để duy trì trạng thái giữa các thao tác chèn.

Qua quá trình thực hiện, nhóm không chỉ củng cố kiến thức về hệ quản trị cơ sở dữ liệu mà còn nâng cao kỹ năng lập trình, tư duy phân tích và khả năng làm việc nhóm – những yếu tố quan trọng cho các dự án thực tế trong ngành công nghệ thông tin.

```
ngan@ngan-virtual-machine:~/movielens_project$ python3 Assignment1Tester.py
A database named "dds_assgn1" already exists
Database 'dds_assgn1' already exists.
loadratings function pass!
rangepartition function pass!
rangeinsert function pass!
Database 'dds_assgn1' already exists.
roundrobinpartition function pass!
roundrobininsert function pass!
Press enter to Delete all tables?
```

Hình 9-1 Kết quả pass tất cả các test