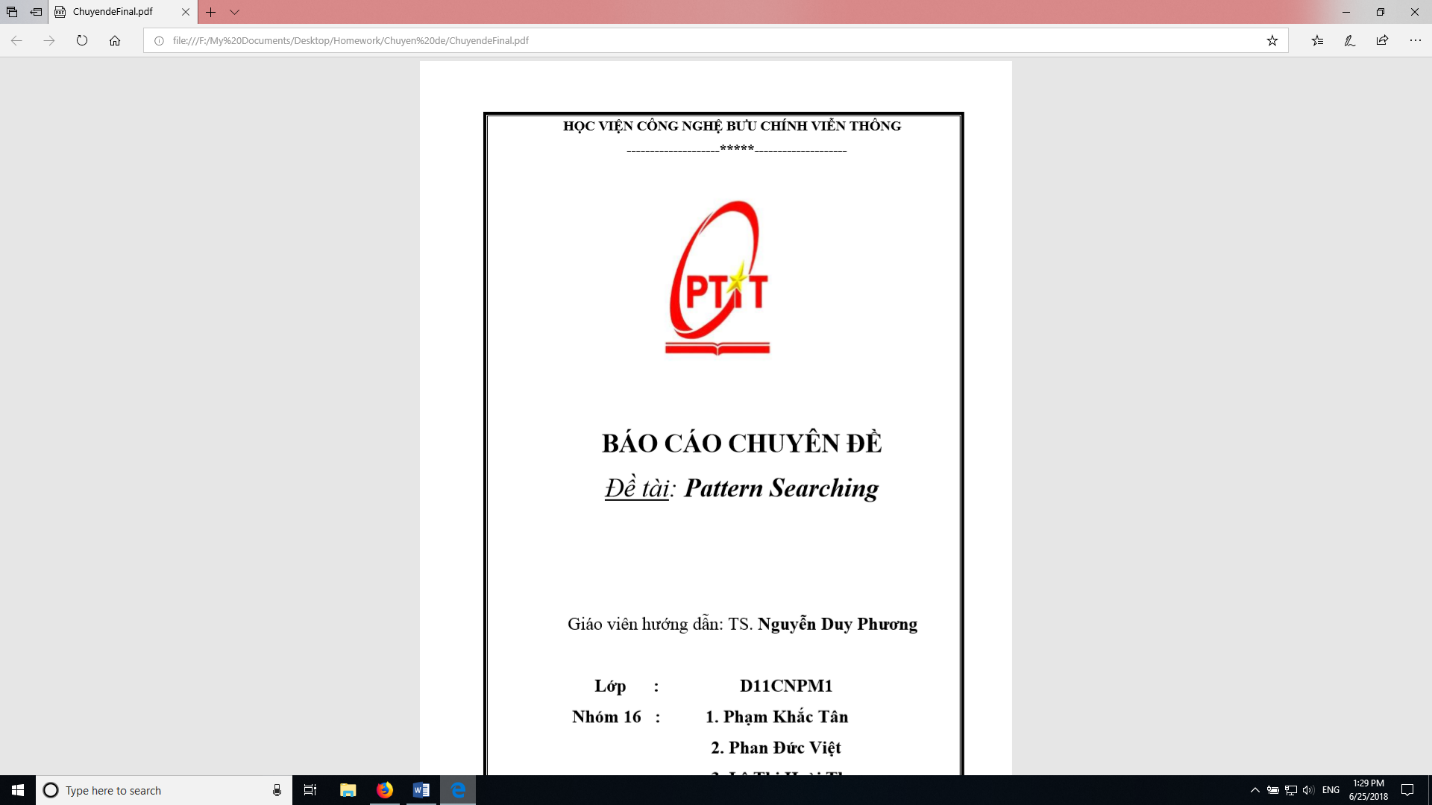


**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**



**BÁO CÁO**

**MÔN: CƠ SỞ DỮ LIỆU PHÂN TÁN**

**Giảng viên:** Kim Ngọc Bách

**Sinh viên:** Trần Đức Đạt

**Mã sinh viên**: B22DCCN203

**Lớp**: D22CNPM06

**Số điện thoại:** 0385550840

***Hà Nội, 10/06/2025***

1. **Mô tả bài toán:**
2. **Giới thiệu:**

Bài tập lớn yêu cầu mô phỏng các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở như PostgreSQL hoặc MySQL. Nhiệm vụ bao gồm viết các hàm Python để tải dữ liệu từ tập tin ratings.dat của MovieLens, thực hiện phân mảnh ngang theo hai phương pháp (phân mảnh theo khoảng và phân mảnh vòng tròn), đồng thời hỗ trợ chèn dữ liệu mới vào đúng phân mảnh

* 1. Phân mảnh theo khoảng

**Phân mảnh theo khoảng** là kỹ thuật chia một quan hệ thành các mảnh dựa trên việc xác định các khoảng giá trị liên tục hoặc rời rạc của một thuộc tính (thường là khóa phân mảnh). Mỗi mảnh sẽ chứa các hàng mà giá trị của thuộc tính được chọn nằm trong một khoảng được định nghĩa trước. Các khoảng này được xác định sao cho:

* **Tính đầy đủ (Completeness)**: Tất cả các hàng trong quan hệ gốc đều được phân bổ vào một mảnh nào đó, không có hàng nào bị bỏ sót.
* **Tính tách biệt (Disjointness)**: Các mảnh không chồng lấn, tức là một hàng chỉ thuộc về một mảnh duy nhất (trừ trường hợp đặc biệt khi cần sao chép dữ liệu).
* **Tính tái tạo (Reconstruction)**: Quan hệ gốc có thể được tái tạo bằng cách hợp (union) tất cả các mảnh.

Phân mảnh theo khoảng thường được sử dụng khi dữ liệu có thể được phân chia dựa trên các giá trị thuộc tính có tính chất phân bố tự nhiên, ví dụ: theo giá trị số (tuổi, thu nhập, mã ID) hoặc theo thứ tự (ngày tháng, vùng địa lý).

* 1. Phân mảnh Round Robin
* **Định nghĩa**: Phân mảnh Round-Robin chia các hàng của một quan hệ thành các mảnh bằng cách phân bổ chúng tuần tự (theo thứ tự) vào các nút hoặc mảnh khác nhau trong hệ thống cơ sở dữ liệu phân tán. Mỗi hàng được gán vào một mảnh theo một chu kỳ lặp lại, đảm bảo phân phối đều dữ liệu.
* **Đặc điểm**:
  + Không sử dụng điều kiện hoặc giá trị thuộc tính để phân chia.
  + Phân bổ dữ liệu theo kiểu "vòng tròn" (round-robin), tức là hàng thứ nhất vào mảnh 1, hàng thứ hai vào mảnh 2, hàng thứ ba vào mảnh 3, rồi quay lại mảnh 1, v.v.
  + Thích hợp khi không có thuộc tính nào tự nhiên để phân mảnh (như khoảng giá trị hoặc giá trị băm).
* **Mục đích**:
  + Phân tải đều dữ liệu giữa các nút để tối ưu hóa hiệu suất truy vấn song song.
  + Đơn giản hóa việc quản lý dữ liệu khi không cần xác định các vị từ (predicates) phức tạp.

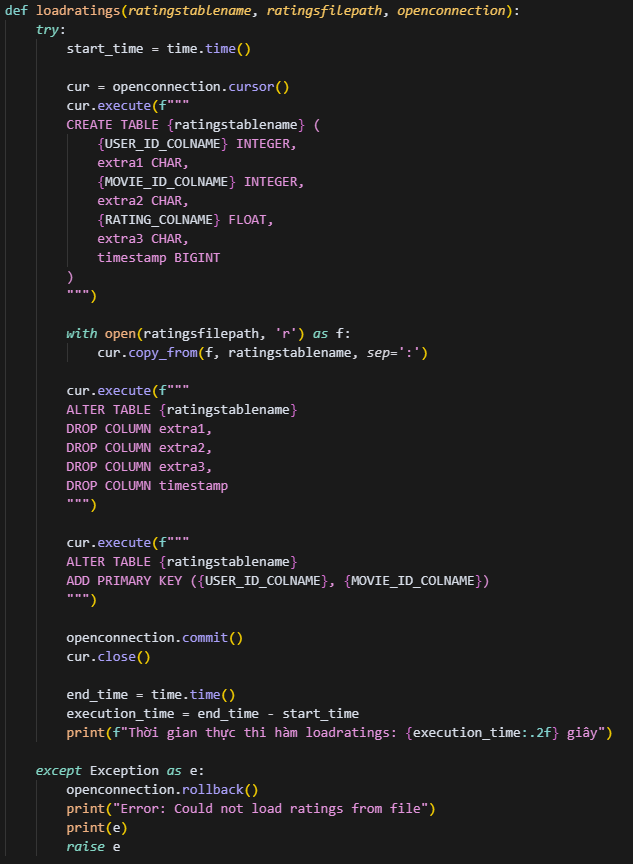
1. **Phạm vi**  
    Báo cáo trình bày cách triển khai các hàm sau:

* LoadRatings(): Tải dữ liệu từ tệp rating.dat vào bảng Ratings trong PostgreSQL.
* Range\_Partition(): Phân mảnh bảng Ratings thành N mảnh dựa trên các khoảng giá trị đồng đều của thuộc tính Rating.
* RoundRobin\_Partition(): Phân mảnh bảng Ratings thành N mảnh theo phương pháp vòng tròn.
* Range\_Insert(): Chèn một bộ dữ liệu mới vào bảng Ratings và đúng mảnh dựa trên giá trị Rating.
* RoundRobin\_Insert(): Chèn một bộ dữ liệu mới vào bảng Ratings và đúng mảnh theo phương pháp vòng tròn.

1. **Tập dữ liệu**  
    Tập dữ liệu MovieLens (rating.dat) chứa thông tin đánh giá phim với định dạng: UserID::MovieID::Rating::Timestamp. Lược đồ bảng Ratings trong PostgreSQL gồm:

* UserID (int): ID người dùng.
* MovieID (int): ID phim.
* Rating (float): Điểm đánh giá (từ 0 đến 5, bước nhảy 0.5).

1. **Triển khai:**
2. **Hàm loadsrating(ratingstablename, ratingsfilepath, openconnection):**



***1.1. Tạo Bảng Tạm Thời***

Đầu tiên, hàm sẽ tạo một bảng tạm thời trong cơ sở dữ liệu với cấu trúc phù hợp với định dạng của tệp dữ liệu ratings. Tệp dữ liệu có định dạng:

**userid:extra1:movieid:extra2:rating:extra3:timestamp**

Do đó, bảng tạm thời cần có các cột để chứa các giá trị này, bao gồm userid, extra1, movieid, extra2, rating, extra3, và timestamp. Câu lệnh SQL tạo bảng tạm thời như sau:

**CREATE TABLE {ratingstablename} (**

**{USER\_ID\_COLNAME} INTEGER,**

**extra1 CHAR,**

**{MOVIE\_ID\_COLNAME} INTEGER,**

**extra2 CHAR,**

**{RATING\_COLNAME} FLOAT,**

**extra3 CHAR,**

**timestamp BIGINT**

**)**

Trong đó, {USER\_ID\_COLNAME} và {MOVIE\_ID\_COLNAME} là các cột chính của bảng, và các cột extra1, extra2, extra3, timestamp là các cột phụ không cần thiết sau khi tải dữ liệu.

***1.2. Copy Dữ Liệu Từ File vào Bảng***

Sau khi tạo bảng tạm thời, chúng ta sử dụng phương thức copy\_from của thư viện psycopg2 để tải dữ liệu từ tệp vào bảng trong cơ sở dữ liệu. Phương thức này rất hiệu quả và nhanh chóng cho việc tải lượng lớn dữ liệu.

Câu lệnh để thực hiện thao tác copy là:

**with open(ratingsfilepath, 'r') as f:**

**cur.copy\_from(f, ratingstablename, sep=':')**

*ratingsfilepath: Đường dẫn đến tệp dữ liệu.*

*ratingstablename: Tên bảng trong cơ sở dữ liệu mà chúng ta muốn tải dữ liệu vào.*

*sep=':': Tệp dữ liệu có các trường được phân tách bởi dấu :.*

***1.3. Dọn Dẹp Bảng***

Sau khi dữ liệu đã được tải vào bảng tạm thời, các cột phụ không cần thiết (extra1, extra2, extra3, timestamp) sẽ được loại bỏ để bảng chỉ còn các cột cần thiết (userid, movieid, rating).

Câu lệnh SQL để xóa các cột phụ này là:

**ALTER TABLE {ratingstablename}**

**DROP COLUMN extra1,**

**DROP COLUMN extra2,**

**DROP COLUMN extra3,**

**DROP COLUMN timestamp**

***1.4. Thêm Primary Key***

Để đảm bảo tính duy nhất của các bản ghi trong bảng, chúng ta thêm một khóa chính composite cho cặp (userid, movieid). Điều này giúp tránh việc có nhiều bản ghi với cùng một cặp (userid, movieid) trong bảng.

Câu lệnh SQL để thêm khóa chính là:

**ALTER TABLE {ratingstablename}**

**ADD PRIMARY KEY ({USER\_ID\_COLNAME}, {MOVIE\_ID\_COLNAME})**

***1.5. Đảm Bảo Tính Toàn Vẹn Dữ Liệu với Transaction***

Để đảm bảo tính toàn vẹn của dữ liệu trong cơ sở dữ liệu, toàn bộ quá trình tải dữ liệu và xử lý bảng được thực hiện trong một giao dịch (transaction). Nếu có bất kỳ lỗi nào xảy ra trong quá trình thực thi, toàn bộ giao dịch sẽ được rollback, và dữ liệu sẽ không bị thay đổi.

**except Exception as e:**

**openconnection.rollback() # Rollback nếu có lỗi**

**print("Error: Could not load ratings from file")**

**print(e)**

**raise e**

***1.6. Tính Thời Gian Thực Thi***

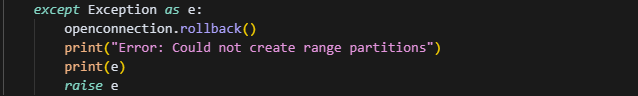
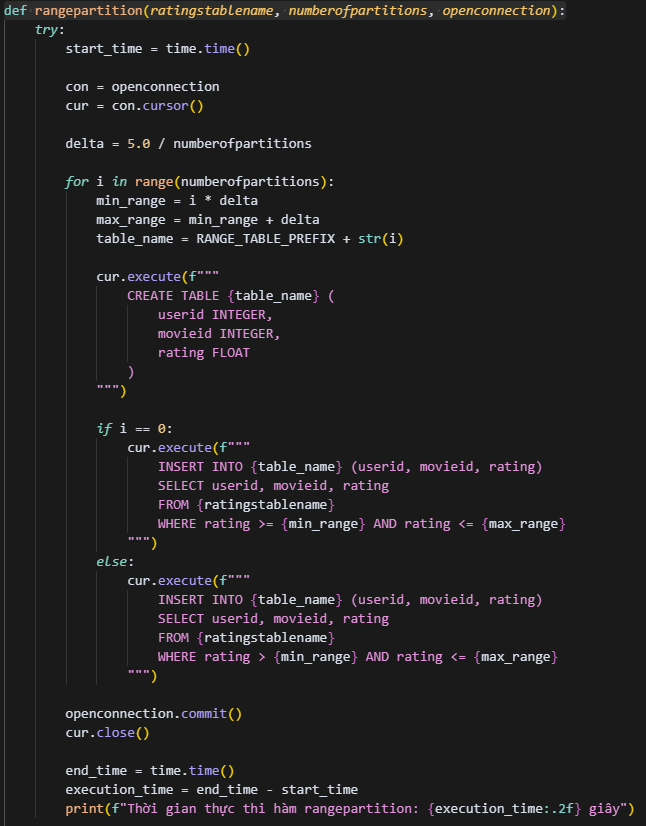
Cuối cùng, chúng ta tính toán và in ra thời gian thực thi của hàm để đánh giá hiệu quả của quá trình tải dữ liệu.

**end\_time = time.time()**

**execution\_time = end\_time - start\_time**

**print(f"Thời gian thực thi hàm loadratings: {execution\_time:.2f} giây")**

1. **Hàm Rangepartition(ratingstablename, numberofpartitions, openconnection):**



***2.1. Tính toán Khoảng Giá Trị Mỗi Phân Vùng***

Đầu tiên, chúng ta cần tính toán kích thước của mỗi phân vùng dựa trên số phân vùng được yêu cầu. Điều này được thực hiện bằng cách chia khoảng giá trị [0, 5] cho số lượng phân vùng (numberofpartitions).

Ví dụ: Nếu số phân vùng là 5, mỗi phân vùng sẽ có kích thước là:

delta = 5.0 / 5 = 1.0

Từ đó, ta có thể xác định các khoảng giá trị cho từng phân vùng:

* range\_part0: [0.0, 1.0]
* range\_part1: (1.0, 2.0]
* range\_part2: (2.0, 3.0]
* range\_part3: (3.0, 4.0]
* range\_part4: (4.0, 5.0]

***2.2. Tạo Các Bảng Phân Vùng***

Mỗi phân vùng sẽ được lưu trữ trong một bảng riêng biệt, với tên định dạng range\_part{index}. Các bảng này có cấu trúc giống như bảng ratings, bao gồm ba cột: userid, movieid, và rating.

Việc tạo bảng cho mỗi phân vùng được thực hiện thông qua câu lệnh SQL CREATE TABLE. Cụ thể, mỗi bảng phân vùng có cấu trúc như sau:

**CREATE TABLE range\_part0 (**

**userid INTEGER,**

**movieid INTEGER,**

**rating FLOAT)**

***2.3. Phân Phối Dữ Liệu Vào Các Bảng Phân Vùng***

Sau khi bảng phân vùng được tạo ra, chúng ta thực hiện phân phối dữ liệu từ bảng ratings vào các bảng phân vùng tương ứng.

Đối với phân vùng đầu tiên (i=0), chúng ta sẽ bao gồm cả giá trị min\_range và max\_range. Ví dụ, đối với phân vùng range\_part0, câu lệnh SQL sẽ lấy tất cả các bản ghi có rating nằm trong khoảng [0.0, 1.0].

Đối với các phân vùng còn lại (i > 0), chúng ta sẽ sử dụng điều kiện (min\_range, max\_range], tức là không bao gồm giá trị min\_range nhưng bao gồm max\_range.

Ví dụ:

Đối với phân vùng range\_part0 (i=0), câu lệnh SQL sẽ là:

**INSERT INTO range\_part0 (userid, movieid, rating)**

**SELECT userid, movieid, rating**

**FROM ratings**

**WHERE rating >= 0.0 AND rating <= 1.0**

Đối với phân vùng range\_part1 (i=1), câu lệnh SQL sẽ là:

**INSERT INTO range\_part1 (userid, movieid, rating)**

**SELECT userid, movieid, rating**

**FROM ratings**

**WHERE rating > 1.0 AND rating <= 2.0**

***2.4. Xử Lý Lỗi và Đảm Bảo Tính Toàn Vẹn Dữ Liệu***

Để đảm bảo tính toàn vẹn dữ liệu, chúng ta sử dụng transaction để thực hiện toàn bộ quá trình phân vùng. Nếu có bất kỳ lỗi nào xảy ra trong quá trình thực thi (tạo bảng hoặc phân phối dữ liệu), toàn bộ giao dịch sẽ được rollback, đảm bảo rằng không có thay đổi nào được thực hiện trong cơ sở dữ liệu nếu có lỗi.

**try:**

**openconnection.commit()**

**except Exception as e:**

**openconnection.rollback()**

**print("Error: Could not create range partitions")**

**print(e)**

**raise e**

***2.5. Tính Thời Gian Thực Thi***

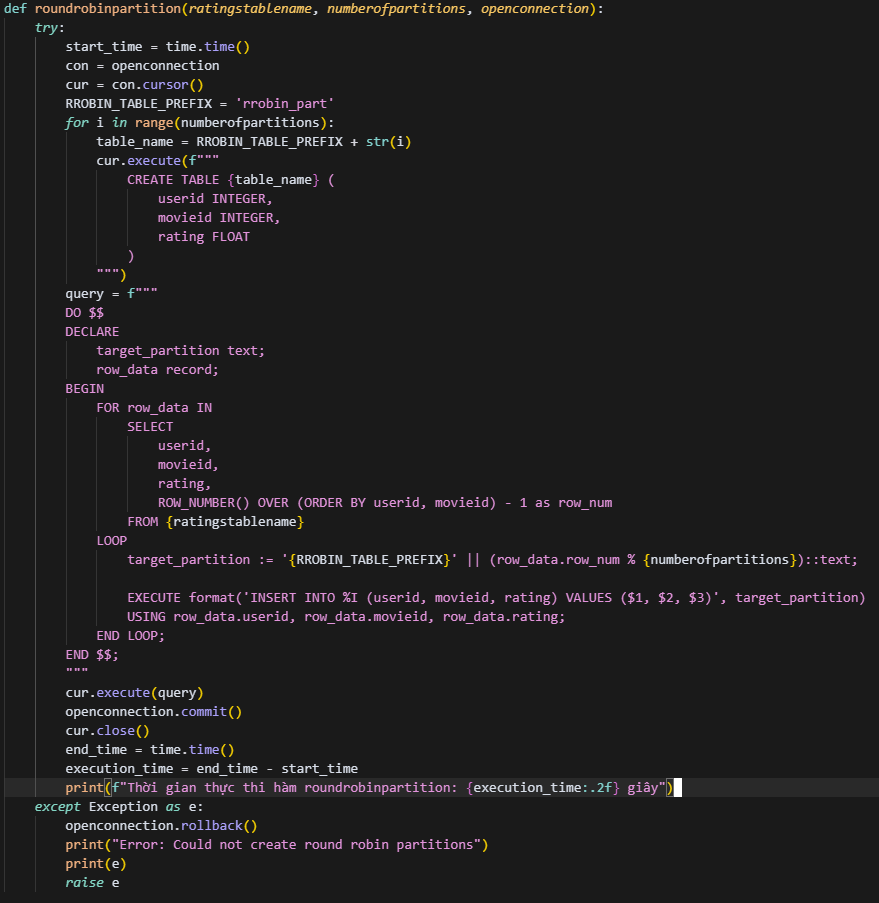
Cuối cùng, chúng ta tính toán và in ra thời gian thực thi của hàm để đánh giá hiệu quả của quá trình phân vùng. Thời gian thực thi được đo bằng cách sử dụng time.time().

**end\_time = time.time()**

**execution\_time = end\_time - start\_time**

**print(f"Thời gian thực thi hàm rangepartition: {execution\_time:.2f} giây")**

1. **Hàm Roundrobinpartition()**



***3.1. Tạo Bảng Phân Mảnh***

Đầu tiên, hàm tạo ra numberofpartitions bảng phân mảnh với tên bảng theo định dạng rrobin\_part0, rrobin\_part1, ..., rrobin\_part{N-1}, trong đó N là số phân mảnh cần tạo.

Câu lệnh SQL để tạo bảng phân mảnh:

CREATE TABLE rrobin\_part0 (

userid INTEGER,

movieid INTEGER,

rating FLOAT

)

Cấu trúc của các bảng phân mảnh là giống nhau, gồm ba cột: userid, movieid, và rating.

***3.2. Phân Phối Dữ Liệu vào Các Bảng Phân Mảnh***

Dữ liệu trong bảng ratings sẽ được phân phối vào các bảng phân mảnh theo phương pháp vòng tròn (Round-Robin). Cụ thể, dữ liệu sẽ được chèn vào các bảng phân mảnh theo thứ tự, sau đó quay lại bảng đầu tiên sau khi đã hết số bảng.

Ví dụ: Với 3 bảng phân mảnh, dữ liệu sẽ được phân phối như sau:

Dòng 1 → rrobin\_part0

Dòng 2 → rrobin\_part1

Dòng 3 → rrobin\_part2

Dòng 4 → rrobin\_part0

Dòng 5 → rrobin\_part1

…

Để thực hiện điều này, hàm sử dụng cấu trúc PL/pgSQL trong PostgreSQL, với một vòng lặp qua các dòng trong bảng ratings:

DO $$

DECLARE

target\_partition text;

row\_data record;

BEGIN

FOR row\_data IN

SELECT userid, movieid, rating, ROW\_NUMBER() OVER (ORDER BY userid,movieid) - 1 as row\_num

FROM ratings

LOOP

target\_partition := 'rrobin\_part' || (row\_data.row\_num % {numberofpartitions})::text;

EXECUTE format('INSERT INTO %I (userid, movieid, rating) VALUES ($1, $2, $3)', target\_partition)

USING row\_data.userid, row\_data.movieid, row\_data.rating;

END LOOP;

END $$;

Trong đoạn mã trên:

ROW\_NUMBER() OVER (ORDER BY userid, movieid) - 1 tạo ra một chỉ số hàng (row number) để phân phối dữ liệu vào các bảng phân mảnh.

row\_num % {numberofpartitions} sẽ tính toán chỉ số phân mảnh mà dữ liệu sẽ được chèn vào. Cách này giúp phân phối đều dữ liệu vào các phân mảnh theo kiểu vòng tròn.

EXECUTE format(...) cho phép thực hiện câu lệnh SQL động để chèn dữ liệu vào bảng phân mảnh tương ứng.

***3.3. Quản Lý Transaction và Thời Gian Thực Thi***

Hàm sử dụng transaction để đảm bảo tính toàn vẹn của dữ liệu. Nếu có bất kỳ lỗi nào xảy ra trong quá trình thực thi, toàn bộ giao dịch sẽ bị rollback và các thay đổi sẽ không được lưu vào cơ sở dữ liệu.

openconnection.commit() # Commit các thay đổi vào database

Ngoài ra, thời gian thực thi của hàm được tính và in ra để giúp đánh giá hiệu quả của quá trình phân mảnh:

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"Thời gian thực thi hàm roundrobinpartition: {execution\_time:.2f} giây")

***3.4. Xử Lý Lỗi***

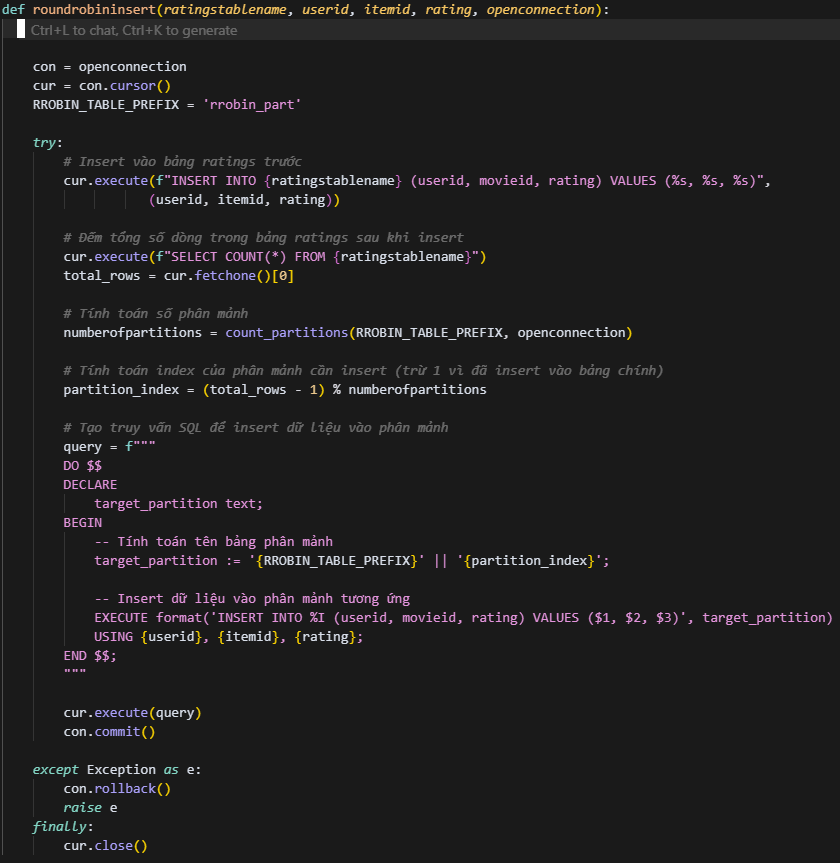
Trong trường hợp có lỗi xảy ra, hàm sẽ rollback toàn bộ giao dịch và thông báo lỗi cho người sử dụng:

openconnection.rollback() # Rollback nếu có lỗi

print("Error: Could not create round robin partitions")

print(e)

raise e

1. **Hàm Roundrobininsert()** 

***4.1. Khởi tạo Kết nối và Tiền tố Bảng***

Hàm nhận vào một đối tượng kết nối cơ sở dữ liệu (openconnection) và khởi tạo một con trỏ (cur) để thực hiện các truy vấn SQL. Tiền tố cho các bảng phân mảnh được định nghĩa là RROBIN\_TABLE\_PREFIX = 'rrobin\_part'.

***4.2. Chèn Dữ liệu vào Bảng Gốc (ratings)***

Đầu tiên, bản ghi dữ liệu mới (userid, itemid, rating) được chèn vào bảng ratings chính:

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

**(userid, itemid, rating))**

Việc chèn vào bảng chính trước tiên là cần thiết để có cơ sở dữ định danh cho bản ghi mới trước khi phân phối.

***4.3. Đếm Tổng số Dòng và Xác định Phân mảnh***

Sau khi chèn vào bảng chính, hàm đếm tổng số dòng hiện có trong bảng ratings để xác định vị trí của bản ghi vừa chèn:

**cur.execute(f"SELECT COUNT(\*) FROM {ratingstablename}")**

**total\_rows = cur.fetchone()[0]**

**numberofpartitions = count\_partitions(RROBIN\_TABLE\_PREFIX, openconnection)**

**partition\_index = (total\_rows - 1) % numberofpartitions**

total\_rows: Lấy tổng số dòng trong bảng ratings sau khi bản ghi mới được thêm vào.

numberofpartitions: Được xác định bằng cách gọi hàm count\_partitions (giả định đã được định nghĩa) để biết số lượng bảng phân mảnh hiện có.

partition\_index: Tính toán chỉ số của bảng phân mảnh mà bản ghi hiện tại cần được chèn vào. Công thức (total\_rows - 1) % numberofpartitions đảm bảo phân phối d ữ liệu theo kiểu vòng tròn, với total\_rows - 1 chuyển đổi số đếm từ 1 sang chỉ số bắt đầu từ 0.

***4.4. Chèn Dữ liệu vào Bảng Phân mảnh Tương ứng***

Sử dụng chỉ số phân mảnh đã tính toán, hàm xây dựng và thực thi một khối lệnh PL/pgSQL để chèn dữ liệu vào bảng phân mảnh đích:

**query = f"""**

**DO $$**

**DECLARE**

**target\_partition text;**

**BEGIN**

**target\_partition := '{RROBIN\_TABLE\_PREFIX}' || '{partition\_index}';**

**EXECUTE format('INSERT INTO %I (userid, movieid, rating) VALUES ($1, $2, $3)', target\_partition)**

**USING {userid}, {itemid}, {rating};**

**END $$;**

**"""**

**cur.execute(query)**

**con.commit()**

target\_partition: Tên bảng phân mảnh được tạo động dựa trên tiền tố và partition\_index.

EXECUTE format(...): Thực thi câu lệnh INSERT động vào bảng phân mảnh đã xác định.

***4.5. Quản lý Transaction và Xử lý Lỗi***

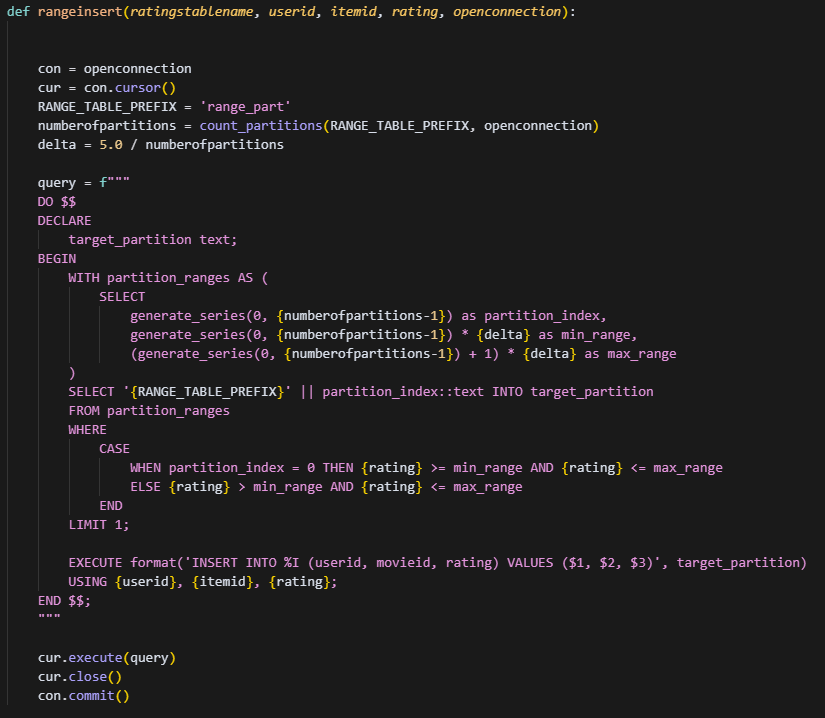
Hàm sử dụng cơ chế transaction để đảm bảo tính toàn vẹn của dữ liệu:

Nếu tất cả các thao tác chèn (vào bảng gốc và bảng phân mảnh) thành công, con.commit() sẽ lưu các thay đổi vào cơ sở dữ liệu.

Trong trường hợp có bất kỳ lỗi nào xảy ra trong quá trình thực thi, con.rollback() sẽ hủy bỏ toàn bộ giao dịch, đảm bảo không có dữ liệu bị thay đổi một cách không nhất quán. Lỗi sẽ được thông báo và được raise ra ngoài.

Con trỏ cơ sở dữ liệu (cur) luôn được đóng trong khối finally để giải phóng tài nguyên.

1. **Hàm rangeinsert()**



***5.1. Khởi tạo Kết nối và Xác định Tham số Phân mảnh***

Hàm nhận vào một đối tượng kết nối cơ sở dữ liệu (openconnection) và khởi tạo một con trỏ (cur) để thực hiện các truy vấn SQL. Tiền tố cho các bảng phân mảnh được định nghĩa là RANGE\_TABLE\_PREFIX = 'range\_part'.

Tính toán số lượng phân mảnh: Hàm gọi count\_partitions để xác định tổng số bảng phân mảnh hiện có.

Tính toán delta: Biến delta được tính bằng cách chia tổng phạm vi điểm rating cho số lượng phân mảnh. delta này sẽ định nghĩa kích thước của mỗi phạm vi điểm rating.

**numberofpartitions = count\_partitions(RANGE\_TABLE\_PREFIX, openconnection)**

**delta = 5.0 / numberofpartitions**

***5.2. Xác định Bảng Phân mảnh Đích và Chèn Dữ liệu (Sử dụng PL/pgSQL)***

Hàm sử dụng một khối lệnh PL/pgSQL để thực hiện logic xác định phân mảnh và chèn dữ liệu:

**DO $$**

**DECLARE**

**target\_partition text;**

**BEGIN**

**WITH partition\_ranges AS (**

**SELECT**

**generate\_series(0, {numberofpartitions-1}) as partition\_index,**

**generate\_series(0, {numberofpartitions-1}) \* {delta} as min\_range,**

**(generate\_series(0, {numberofpartitions-1}) + 1) \* {delta} as max\_range**

**)**

**SELECT '{RANGE\_TABLE\_PREFIX}' || partition\_index::text INTO target\_partition**

**FROM partition\_ranges**

**WHERE**

**CASE**

**WHEN partition\_index = 0 THEN {rating} >= min\_range AND {rating} <= max\_range**

**ELSE {rating} > min\_range AND {rating} <= max\_range**

**END**

**LIMIT 1;**

**EXECUTE format('INSERT INTO %I (userid, movieid, rating) VALUES ($1, $2, $3)', target\_partition)**

**USING {userid}, {itemid}, {rating};**

**END $$;**

WITH partition\_ranges AS (...): Một Common Table Expression (CTE) tạm thời được tạo ra để sinh ra các phạm vi điểm rating cho mỗi phân mảnh. Mỗi hàng trong CTE sẽ có partition\_index, min\_range, và max\_range.

Xác định target\_partition: Câu lệnh SELECT sau đó chọn ra partition\_index và xây dựng target\_partition (ví dụ: range\_part0, range\_part1, v.v.) dựa trên việc giá trị rating nằm trong phạm vi nào.

Đối với phân mảnh đầu tiên (partition\_index = 0), phạm vi được định nghĩa là rating >= min\_range AND rating <= max\_range (bao gồm cả điểm 0.0).

Đối với các phân mảnh tiếp theo, phạm vi là rating > min\_range AND rating <= max\_range để tránh trùng lặp điểm ranh giới.

EXECUTE format(...): Sau khi target\_partition được xác định, một câu lệnh INSERT động được thực thi để chèn userid, itemid, và rating vào bảng phân mảnh tương ứng.

***5.3. Quản lý Transaction và Đóng Con trỏ***

Sau khi thực hiện truy vấn PL/pgSQL, hàm sẽ:

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

Thực hiện con.commit() để lưu các thay đổi vào cơ sở dữ liệu.

1. **Kết luận**

Thông qua bài tập lớn này, em đã có cơ hội tìm hiểu và triển khai các kỹ thuật phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu phân tán, bao gồm phân mảnh theo khoảng (Range Partitioning) và phân mảnh vòng tròn (Round Robin Partitioning). Bài tập không chỉ giúp hiểu rõ về mặt lý thuyết mà còn rèn luyện kỹ năng lập trình thao tác với cơ sở dữ liệu sử dụng PostgreSQL và thư viện psycopg2 trong Python.

Cụ thể, em đã:

Xây dựng và quản lý hệ thống kết nối đến cơ sở dữ liệu PostgreSQL.

Cài đặt và thực hiện nạp dữ liệu từ file vào bảng chính.

Thiết kế và triển khai hai cơ chế phân mảnh dữ liệu:

* Range Partitioning: chia dữ liệu theo khoảng giá trị của rating.
* Round Robin Partitioning: chia đều dữ liệu theo thứ tự xuất hiện.
* Thực hiện chèn dữ liệu mới vào đúng phân vùng tương ứng.

Sau khi hoàn tất các chức năng và tiến hành test thử, em nhận thấy thời gian truy vấn ở một số hàm vẫn còn cao, đặc biệt với các bảng có kích thước lớn. Trong bài tập lớn của em vẫn còn nhiều thiếu sót, em sẽ rút kinh nghiệm và hoàn thành tốt hơn vào các bài sau.