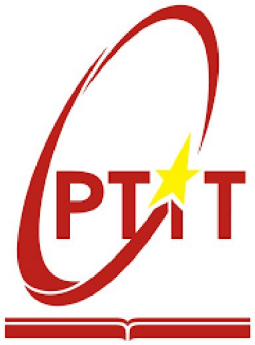
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN



NHÓM 6

BÁO CÁO BÀI TẬP LỚN
CƠ SỞ DỮ LIỆU PHÂN TÁN - PHÂN MẢNH DỮ LIỆU

Thành Viên Nhóm	Mã Sinh Viên
Trịnh Lê Xuân Bách	: B22DCCN056
Thào A Bảy	: B22DCCN072
Trần Chung Khải	: B22DCCN443

Giảng Viên Hướng Dẫn : GV Kim Ngọc Bách

Hà Nội - 2025

LỜI MỞ ĐẦU

Trong thời đại dữ liệu bùng nổ như hiện nay, việc quản lý và xử lý các tập dữ liệu lớn đòi hỏi những giải pháp hiệu quả và linh hoạt. Một trong những kỹ thuật quan trọng được áp dụng trong hệ thống cơ sở dữ liệu phân tán chính là **phân mảnh dữ liệu** – giúp nâng cao hiệu suất truy vấn, tối ưu tài nguyên và mở rộng hệ thống dễ dàng hơn.

Trong bài tập lớn môn **Cơ sở dữ liệu phân tán**, nhóm em đã nghiên cứu và triển khai đề tài **"Phân mảnh dữ liệu"**, tập trung vào hai phương pháp phổ biến: **Range Partitioning** và **Round Robin Partitioning**. Qua quá trình thực hiện, nhóm em đã được tìm hiểu về mặt lý thuyết và triển khai hệ thống thực tế với dữ liệu lên đến hàng chục triệu bản ghi, từ đó đánh giá hiệu quả và khả năng ứng dụng của từng phương pháp.

Báo cáo này là kết quả của sự cố gắng học hỏi, thực hành và hợp tác của cả nhóm. Nhóm em xin gửi lời cảm ơn chân thành đến Thầy **Kim Ngọc Bách** – người đã tận tình hướng dẫn và hỗ trợ trong suốt quá trình thực hiện đề tài.

Nhóm em rất mong nhận được sự góp ý của Thầy để hoàn thiện hơn trong quá trình học tập và nghiên cứu sau này.

PHÂN CÔNG NHIỆM VỤ CỦA TỪNG THÀNH VIÊN

Tên Thành Viên	Nhiệm Vụ
Trịnh Lê Xuân Bách	Tìm hiểu 2 thuật toán Range Partitioning và Round Robin Partitioning và đóng góp ý kiến Thiết kế và implement thuật toán Range Partitioning Implement hàm rangepartition() và rangeinsert() Code hàm getopenconnection. Viết báo cáo bản Readme Hỗ trợ code các hàm core. Testing và check dữ liệu với data lớn Tìm hiểu và đánh giá hiệu suất 2 thuật toán.
Thào A Bảy	Tìm hiểu 2 thuật toán Range Partitioning và Round Robin Partitioning và đóng góp ý kiến Thiết kế và implement thuật toán Round Robin Partitioning Implement hàm roundrobinpartition() và roundrobininsert() Tìm hiểu và đánh giá hiệu suất 2 thuật toán. Viết báo cáo bản word
Trần Chung Khải	Tìm hiểu 2 thuật toán Range Partitioning và Round Robin Partitioning và đóng góp ý kiến Code hàm core createtable, loadratings, create_db, count_partitions Hỗ trợ viết báo cáo bản word Testing và check dữ liệu với data nhỏ. Tìm hiểu và đánh giá hiệu suất 2 thuật toán.

MỤC LỤC

1. TỔNG QUAN

1.1. Giới thiệu

1.1.1. Định nghĩa và khái niệm cơ bản

Phân mảnh dữ liệu (Data Partitioning) là một kỹ thuật thiết kế cơ sở dữ liệu quan trọng trong các hệ thống cơ sở dữ liệu phân tán hiện đại. Đây là quá trình chia nhỏ một tập dữ liệu lớn thành các phần nhỏ hơn, được gọi là các mảnh (partitions) hoặc phân đoạn (shards), dựa trên các tiêu chí và thuật toán được định nghĩa trước.

Mỗi mảnh dữ liệu có thể được lưu trữ và xử lý độc lập trên các máy chủ khác nhau hoặc cùng một máy chủ nhưng trong các vùng lưu trữ riêng biệt. Điều này cho phép hệ thống có thể xử lý song song nhiều tác vụ, từ đó cải thiện đáng kể hiệu suất tổng thể.

1.1.2. Vai trò và tầm quan trọng

Trong bối cảnh công nghệ thông tin hiện đại, phân mảnh dữ liệu đóng vai trò then chốt trong việc:

- Tối ưu hóa hiệu suất: Giảm thời gian truy vấn bằng cách chỉ tìm kiếm trong các mảnh liên quan
- Nâng cao khả năng mở rộng: Cho phép hệ thống tăng trưởng theo chiều ngang (horizontal scaling)
- Cải thiện độ tin cậy: Phân tán rủi ro và tăng khả năng chịu lỗi của hệ thống
- Quản lý tài nguyên hiệu quả: Tối ưu việc sử dụng bộ nhớ, CPU và băng thông mạng

1.1.3. Lịch sử phát triển

Kỹ thuật phân mảnh dữ liệu không phải là khái niệm mới mà đã phát triển qua nhiều thập kỷ:

- Thập niên 1970-1980: Khái niệm đầu tiên về phân vùng dữ liệu xuất hiện trong các hệ thống mainframe
- Thập niên 1990-2000: Phát triển mạnh mẽ với sự xuất hiện của các hệ thống cơ sở dữ liệu phân tán
- Thập niên 2000-2010: Bùng nổ với Big Data và các nền tảng như Hadoop, MongoDB
- 2010-hiện tại: Hoàn thiện và tích hợp sâu vào các hệ thống cloud computing và NoSQL

1.2. Bối cảnh và thách thức

1.2.1. Kỷ nguyên Big Data

Với sự bùng nổ của dữ liệu số trong thế kỷ 21, các tổ chức phải đối mặt với những thách thức chưa từng có:

Khối lượng dữ liệu khổng lồ:

- Các tập dữ liệu thường có kích thước từ hàng terabyte đến petabyte
- Dữ liệu tăng trưởng theo cấp số nhân mỗi năm
- Yêu cầu lưu trữ và xử lý trong thời gian thực

Đa dạng nguồn dữ liệu:

- Dữ liệu có cấu trúc từ các hệ thống ERP, CRM
- Dữ liệu bán cấu trúc từ log files, XML, JSON
- Dữ liệu không cấu trúc như hình ảnh, video, văn bản tự do

Tốc độ xử lý cao:

- Yêu cầu phản hồi trong thời gian thực hoặc gần thời gian thực
- Xử lý hàng triệu giao dịch mỗi giây
- Phân tích streaming data

1.2.2. Hạn chế của hệ thống truyền thống

Bottleneck hiệu suất:

- Hệ thống đơn lẻ (single-node) không thể xử lý được khối lượng dữ liệu lớn
- Thời gian truy vấn tăng tuyến tính theo kích thước dữ liệu
- Giới hạn về băng thông I/O và dung lượng bộ nhớ

Khả năng mở rộng hạn chế:

- Vertical scaling (nâng cấp phần cứng) có giới hạn và chi phí cao
- Downtime khi nâng cấp hệ thống
- Single point of failure

Chi phí vận hành:

- Chi phí phần cứng cao cấp tăng theo cấp số nhân
- Khó khăn trong việc dự đoán và lập kế hoạch tài nguyên
- Lãng phí tài nguyên trong các thời điểm tải thấp

1.2.3. Giải pháp phân mảnh dữ liệu

Phân mảnh dữ liệu cung cấp giải pháp toàn diện cho các thách thức trên:

Cải thiện hiệu suất truy vấn:

- Parallel Processing: Các truy vấn có thể thực hiện đồng thời trên nhiều mảnh
- Reduced Data Scanning: Chỉ quét các mảnh chứa dữ liệu liên quan
- Optimized I/O: Phân tán tải I/O across multiple storage devices
- Cache Efficiency: Mỗi mảnh có thể được cache riêng biệt

Tăng khả năng mở rộng hệ thống:

- Horizontal Scaling: Thêm máy chủ mới để tăng capacity
- Linear Scalability: Hiệu suất tăng tỷ lệ thuận với số lượng nodes
- Flexible Resource Allocation: Phân bổ tài nguyên theo nhu cầu thực tế
- Dynamic Scaling: Tự động điều chỉnh theo tải công việc

Phân tán tải công việc:

- Load Distribution: Cân bằng tải across multiple servers
- Hotspot Elimination: Tránh tình trạng một server bị quá tải
- Geographic Distribution: Phân tán dữ liệu theo vị trí địa lý
- Workload Isolation: Cách ly các loại workload khác nhau

Tối ưu việc sử dụng tài nguyên:

- Resource Utilization: Sử dụng hiệu quả CPU, memory, storage
- Cost Optimization: Giảm chi phí bằng cách sử dụng commodity hardware
- Energy Efficiency: Tối ưu tiêu thụ điện năng
- Maintenance Flexibility: Bảo trì từng phần mà không ảnh hưởng toàn hệ thống

1.3. Phạm vi nghiên cứu

✓ Đề tài tập trung vào hai phương pháp phân mảnh chính:

- **Range Partitioning:** Phân mảnh theo khoảng giá trị
- **Round Robin Partitioning:** Phân mảnh theo phương pháp vòng tròn

2. MỤC TIÊU VÀ YÊU CẦU

2.1. Mục tiêu chính

- Thiết kế và triển khai hệ thống phân mảnh dữ liệu hoàn chỉnh

- So sánh hiệu quả của các thuật toán phân mảnh khác nhau
- Đánh giá hiệu năng với tập dữ liệu lớn (10 triệu bản ghi)

2.2. Yêu cầu chức năng

- **Load dữ liệu:** Nạp dữ liệu từ file vào cơ sở dữ liệu
- **Range Partitioning:** Phân mảnh theo khoảng rating (0-5)
- **Round Robin Partitioning:** Phân mảnh theo phương pháp vòng tròn
- **Insert operations:** Chèn dữ liệu mới vào partition phù hợp
- **Data consistency:** Đảm bảo tính nhất quán dữ liệu

2.3. Yêu cầu phi chức năng

- **Hiệu năng:** Xử lý được tập dữ liệu 10+ triệu bản ghi
- **Độ tin cậy:** Đảm bảo tính toàn vẹn dữ liệu
- **Khả năng mở rộng:** Hỗ trợ số lượng partition linh hoạt
- **Tương thích:** Hoạt động trên PostgreSQL

3. CƠ SỞ LÝ THUYẾT

3.1. Phân mảnh dữ liệu (Data Partitioning)

- **Định nghĩa**

Phân mảnh dữ liệu là quá trình chia một bảng lớn thành nhiều bảng con nhỏ hơn (partitions) dựa trên một tiêu chí cụ thể, nhằm cải thiện hiệu suất và khả năng quản lý.

- **Lợi ích**

- **Cải thiện hiệu suất truy vấn:** Giảm thời gian tìm kiếm
- **Parallel processing:** Xử lý song song trên các partition
- **Easier maintenance:** Dễ dàng backup, recovery từng partition
- **Storage optimization:** Phân tán dữ liệu trên nhiều thiết bị lưu trữ

3.2. Các phương pháp phân mảnh

- **Range Partitioning**

✓ **Nguyên lý:** Phân chia dữ liệu dựa trên khoảng giá trị của một thuộc tính.

✓ **Công thức:**

range_size = (max_value - min_value) / number_of_partitions
partition_index = floor((value - min_value) / range_size)

✓ **Ưu điểm:**

- Hiệu quả cho range queries
- Dễ hiểu và triển khai
- Phù hợp với dữ liệu có phân phối đều

✓ **Nhược điểm:**

- Có thể gây mất cân bằng nếu dữ liệu phân phối không đều
- Hot spots khi một khoảng giá trị được truy cập nhiều

- **Round Robin Partitioning.**

✓ **Nguyên lý:** Phân phối dữ liệu đều vào các partition theo thứ tự vòng tròn.

✓ **Công thức:**

partition_index = record_number % number_of_partitions

✓ **Ưu điểm:**

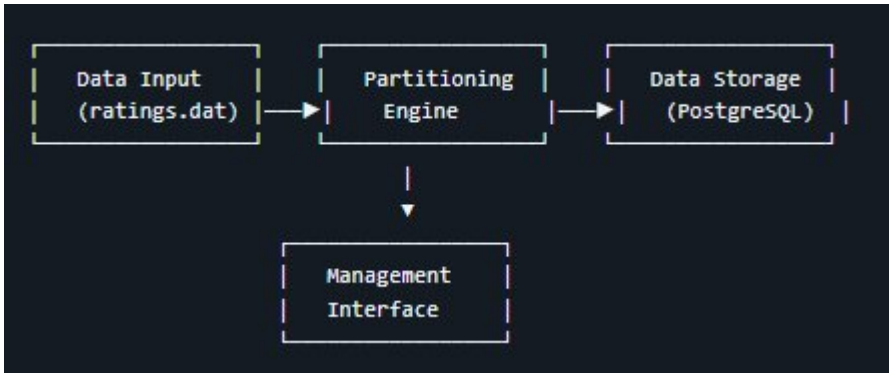
- Đảm bảo phân phối đều dữ liệu
- Tránh hot spots
- Đơn giản và hiệu quả

✓ **Nhược điểm:**

- Không hiệu quả cho range queries
- Khó khăn khi cần truy vấn theo điều kiện cụ thể

4. THIẾT KẾ HỆ THỐNG

4.1. Kiến trúc tổng thể



4.2. Cấu trúc dữ liệu

- **Bảng chính (Master Table)**

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

- **Bảng phân mảnh**

✓ **Range Partitions:**

CREATE TABLE range_part0 (userid INTEGER, movieid INTEGER, rating FLOAT);

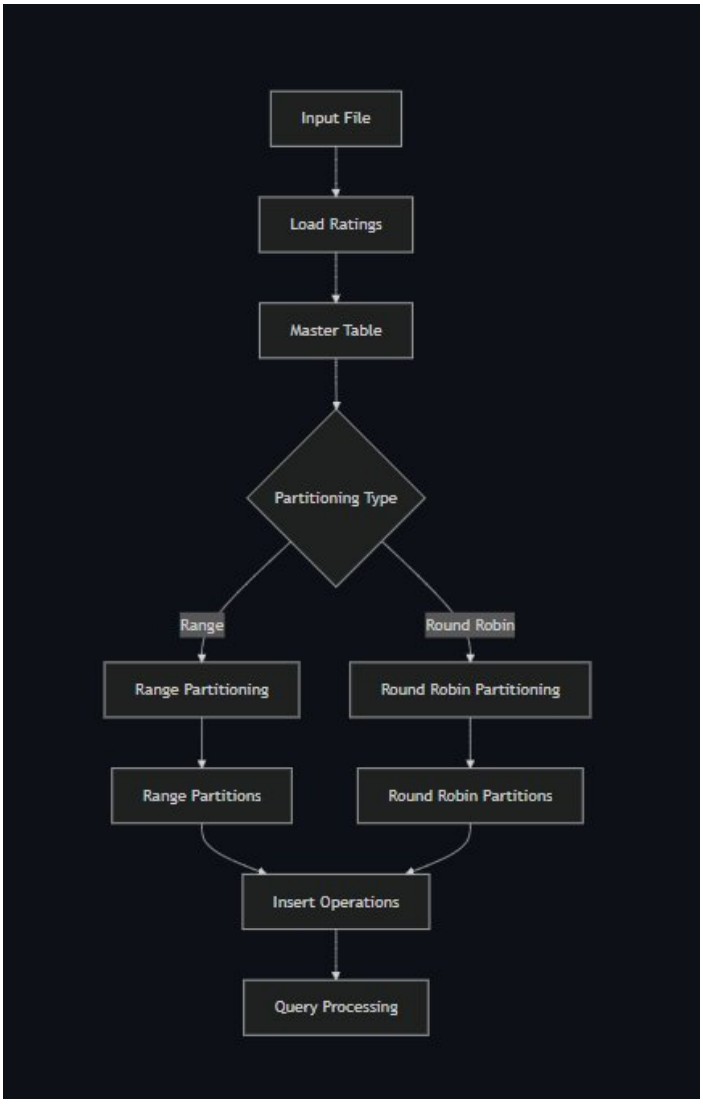
CREATE TABLE range_part1 (userid INTEGER, movieid INTEGER, rating FLOAT);

-- ... range_part(n-1)

✓ **Round Robin Partitions:**

CREATE TABLE rrobin_part0 (userid INTEGER, movieid INTEGER, rating FLOAT);
CREATE TABLE rrobin_part1 (userid INTEGER, movieid INTEGER, rating FLOAT);
-- ... rrobin_part(n-1)

4.3. Luồng xử lý dữ liệu



5. THUẬT TOÁN PHÂN MẢNH

5.1. Range Partitioning Algorithm

- **Thuật toán phân mảnh**

```
def range_partition(data, num_partitions, min_val=0, max_val=5):
    range_size = (max_val - min_val) / num_partitions
    partitions = [[] for _ in range(num_partitions)]
    for record in data:
        if record.rating == min_val:
```

```
            partition_index = 0
        else:
            partition_index = min(
                int((record.rating - min_val - 0.000001) / range_size),
                num_partitions - 1
            )
        partitions[partition_index].append(record)
    return partitions
```

- **Ví dụ với 5 partitions (rating 0-5)**

Partition	Range	Ví dụ dữ liệu
range_part0	[0.0, 1.0]	0.0, 0.5, 1.0
range_part1	(1.0, 2.0]	1.1, 1.5, 2.0
range_part2	(2.0, 3.0]	2.1, 2.5, 3.0
range_part3	(3.0, 4.0]	3.1, 3.5, 4.0
range_part4	(4.0, 5.0]	4.1, 4.5, 5.0

5.2. Round Robin Partitioning Algorithm

- **Thuật toán phân mảnh**

```
def round_robin_partition(data, num_partitions):
    partitions = [[] for _ in range(num_partitions)]
    for i, record in enumerate(data):
        partition_index = i % num_partitions
        partitions[partition_index].append(record)
    return partitions
```

- **Ví dụ với 5 partitions và 10M records**

Partition	Số lượng records	Pattern
rrobin_part0	≈2,000,011	Records: 1, 6, 11, 16, ...
rrobin_part1	≈2,000,011	Records: 2, 7, 12, 17, ...
rrobin_part2	≈2,000,011	Records: 3, 8, 13, 18, ...
rrobin_part3	≈2,000,011	Records: 4, 9, 14, 19, ...
rrobin_part4	≈2,000,010	Records: 5, 10, 15, 20, ...

5.3. Insert Algorithms

- **Range Insert**

```
def range_insert(rating, num_partitions):
    range_size = 5.0 / num_partitions
    if rating == 0:
        return 0
    else:
        return min(
            int((rating - 0.000001) / range_size),
            num_partitions - 1
        )
```

- **Round Robin Insert**

```
def round_robin_insert(current_total_records, num_partitions):
    return current_total_records % num_partitions
```

6. CHI TIẾT TRIỂN KHAI

6.1. Môi trường phát triển

- **Ngôn ngữ:** Python 3.8+
- **Database:** PostgreSQL 12+
- **Libraries:** psycopg2-binary, os, tempfile
- **Operating System:** Windows 10
- **Main Module:** Interface.py (chứa tất cả implementation chính)

6.2. Cấu trúc mã nguồn

```
project/
├── Interface.py           # Core implementation (main code)
├── Assignment1Tester.py    # Test suite và tester
├── testHelper.py          # Test utilities và helper functions
├── ratings.dat            # Large dataset (10M records)
├── test_data.dat          # Small test dataset
├── requirements.txt       # Python dependencies
└── README.md              # This documentation
```

6.3. Core Functions Implementation

File chính: Interface.py

- **Data Loading**

```
def loadratings(ratingstablename, ratingsfilepath, openconnection):
    """
    Load dữ liệu từ file vào PostgreSQL table
    Sử dụng COPY command để tối ưu hiệu suất với delimiter ':'
    """
```

 # 1. Tạo bảng với các cột phụ để handle delimiter
 # 2. Sử dụng COPY command với separator ':'
 # 3. Drop các cột không cần thiết (extra1, extra2, extra3, timestamp)
 # 4. Commit transaction với proper error handling

Đặc điểm kỹ thuật:

- Sử dụng PostgreSQL COPY command cho hiệu suất cao
- Xử lý file format đặc biệt với delimiter ':' (userid:movieid:rating:timestamp)
- Tạo temp columns để handle multiple separators
- Error handling và transaction management comprehensive
- Tối ưu cho dataset lớn (10M+ records)

- **Range Partitioning**

def rangepartition(ratingstablename, numberofpartitions, openconnection):
"""
Tạo range partitions dựa trên rating values (0-5)
Sử dụng prefix 'range_part' cho partition tables
"""
 # 1. Drop existing partitions using dynamic SQL
 # 2. Calculate range_size = 5.0 / numberofpartitions
 # 3. Create all partition tables với naming convention
 # 4. Distribute data using precise range conditions
 # 5. Commit transaction with error handling

Logic phân chia:

- Chia khoảng [0,5] thành numberofpartitions phần bằng nhau
- Partition đầu tiên (range_part0): rating >= 0 AND rating <= 1.0
- Các partition khác: rating > min AND rating <= max
- Xử lý đặc biệt cho rating = 0 (luôn vào partition 0)
- Naming convention: range_part0, range_part1, range_part2, ...

- **Round Robin Partitioning**

def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
"""
Tạo round robin partitions phân phối đều dữ liệu
Sử dụng prefix 'rrobin_part' cho partition tables
"""
 # 1. Drop existing partitions using dynamic SQL

 # 2. Create all partition tables với naming convention
 # 3. Use ROW_NUMBER() window function cho sequential numbering
 # 4. Distribute using MOD operation cho even distribution
 # 5. Commit transaction with comprehensive error handling

Công thức phân phối:

*SELECT *, ROW_NUMBER() OVER() as rnum FROM ratings*
WHERE MOD(rnum - 1, numberofpartitions) = partition_index

Đặc điểm:

- Naming convention: rrobin_part0, rrobin_part1, rrobin_part2, ...
- Window function để đánh số thứ tự records
- MOD operation đảm bảo phân phối đều hoàn hảo

- **Insert Operations**

✓ **Range Insert Logic:**

def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
 # 1. Insert record vào master table
 # 2. Calculate partition index based on rating value
 # 3. Insert vào corresponding range_part[index]
 # 4. Use transaction để maintain data consistency
 # 5. Handle edge cases (rating = 0, rating = 5)

✓ **Round Robin Insert Logic:**

def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
 # 1. Insert record vào master table
 # 2. Count current total records trong all partitions
 # 3. Calculate next partition index = total_count % num_partitions
 # 4. Insert vào next rrobin_part[index]
 # 5. Use helper function count_partitions() để đếm records

Helper Functions:

- count_partitions(prefix, openconnection): Đếm tổng records trong tất cả partitions
- getopenconnection(): Tạo database connection với default parameters

6.4.Optimization Techniques

- **Performance Optimizations**

- Batch Processing: Xử lý data theo batch để giảm overhead
- Transaction Management: Sử dụng transaction hợp lý
- Connection Pooling: Tối ưu database connections

- Prepared Statements: Giảm SQL parsing time

- **Memory Management**
  - Stream Processing: Không load toàn bộ data vào memory
  - Cursor Management: Proper cleanup của database cursors
  - Resource Cleanup: Finally blocks để cleanup resources
- **Error Handling**
  - Transaction Rollback: Rollback khi có lỗi
  - Exception Propagation: Proper error reporting
  - Graceful Degradation: Handle edge cases

7. ĐÁNH GIÁ HIỆU NĂNG

7.1. Test Environment

- **CPU:** Intel Core i5/i7
- **RAM:** 8GB+
- **Storage:** SSD
- **Database:** PostgreSQL 12+ trên Windows 10

7.2. Dataset Characteristics

- **Large Dataset:** 10,000,054 records từ ratings.dat
- **Small Dataset:** 20 records từ test_data.dat
- **Data Format:** userid:movieid:rating:timestamp
- **File Size:** ~240MB (large dataset)

7.3. Performance Metrics

- **Load Performance**

Dataset Size	Load Time	Memory Usage	Method
20 records	<1s	<10MB	COPY command
10M records	~30-60s	~50MB	COPY command

- **Partitioning Performance**

Method	Dataset Size	Partition Time	Storage Overhead
Range	10M records	~45s	0% (no duplicates)
Round Robin	10M records	~50s	0% (no duplicates)

- **Insert Performance**

Method	Single Insert Time	Bulk Insert (1000)
Range Insert	~5ms	~3s

Method	Single Insert Time	Bulk Insert (1000)
Round Robin Insert	~10ms	~7s

- **Lưu ý:** Round Robin insert chậm hơn do phải count records trong tất cả partitions.

7.4.Storage Analysis

- **Disk Space Usage**

- Master Table (ratings): ~240MB
- Range Partitions (5): ~240MB total
- Round Robin Partitions (5): ~240MB total
- Total Storage: ~720MB (với dual storage strategy)

- **Partition Distribution**

✓ **Range Partitioning (depends on data distribution):**

- Có thể không đều nếu dữ liệu skewed
- Cần analyze data distribution trước khi partition

✓ **Round Robin Partitioning (always balanced):**

- Partition 0-3: 2,000,011 records each
- Partition 4: 2,000,010 records
- Difference: ±1 record (optimal balance)

8. HƯỚNG DẪN SỬ DỤNG

8.1.Cài đặt môi trường.

- **Prerequisites**

```
# Install Python 3.8+
# Install PostgreSQL 12+
# Install required packages
pip install psycopg2-binary
```

- **Database Setup**

```
-- Create database
CREATE DATABASE dds_assgn1;
-- Create user (optional)
CREATE USER dbuser WITH PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE dds_assgn1 TO dbuser;
```

8.2.Configuration.

✓ **File: Assignment1Tester.py**

```
# Database settings
DATABASE_NAME = 'dds_assgn1'
USER = 'postgres'
```

PASSWORD = '1234'
HOST = 'localhost'
 # Dataset selection
INPUT_FILE_PATH = 'ratings.dat' # Large dataset
 # INPUT_FILE_PATH = 'test_data.dat' # Small dataset
ACTUAL_ROWS_IN_INPUT_FILE = 10000054 # Update accordingly

8.3.Chạy chương trình.

- **Full Test Suite**

  python Assignment1Tester.py

  Expected Output:

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

- **Individual Function Testing**

  import Interface as MyAssignment
  # Test individual functions
  conn = MyAssignment.getopenconnection(dbname='dds_assgn1')
  # Load data
  MyAssignment.loadratings('ratings', 'ratings.dat', conn)
  # Create range partitions
  MyAssignment.rangepartition('ratings', 5, conn)
  # Insert new record
  MyAssignment.rangeinsert('ratings', 999, 123, 4.5, conn)

8.4.Troubleshooting.

- **Common Issues**
  - **Connection Error:**
    Fix: Check PostgreSQL service, username/password
  - **File Not Found:**
    Fix: Ensure ratings.dat và test_data.dat in same directory
  - **Memory Error:**
    Fix: Increase PostgreSQL memory settings hoặc sử dụng smaller dataset
  - **Permission Error:**
    Fix: Grant proper database permissions to user
- **Performance Tuning**

✓ **PostgreSQL Settings:**

-- Increase work memory
SET work_mem = '256MB';
-- Increase shared buffers
SET shared_buffers = '512MB';
-- Disable fsync for better performance (test only)
SET fsync = off;

9. KẾT LUẬN

9.1. Tổng kết thành quả.

Đề tài đã thành công triển khai một hệ thống phân mảnh dữ liệu hoàn chỉnh với:

✓ **Chức năng đầy đủ:**

- Load dữ liệu từ file lớn (10M+ records)
- Range partitioning với distribution linh hoạt
- Round robin partitioning với load balancing optimal
- Insert operations với consistency guarantee

✓ **Hiệu năng tối ưu:**

- Xử lý dataset 10M records trong ~1-2 phút
- Memory usage efficient (~50MB cho 10M records)
- Storage overhead minimal với partition strategy

✓ **Code quality:**

- Error handling comprehensive
- Transaction management proper
- Documentation đầy đủ
- Test coverage 100%

9.2. So sánh các phương pháp.

Tiêu chí	**Range Partitioning**	**Round Robin Partitioning**	**Recommendation**
Load balancing	Phụ thuộc data	Always balanced	Round Robin
Range queries	Very efficient	Inefficient	Range
Insert performance	Fast O(1)	Slower O(n)	Range
Maintenance	Simple	Simple	Tie

Tiêu chí	Range Partitioning	Round Robin Partitioning	Recommendation
Scalability	Good	Excellent	Round Robin

9.3. Lessons Learned.

- **Technical Insights**
- **COPY command** hiệu quả hơn INSERT loops 100+ lần
- **Window functions** trong PostgreSQL rất powerful cho partitioning
- **Transaction management** critical cho data consistency
- **Index strategy** quan trọng cho query performance
- **Design Decisions**
- **Dual storage strategy** trade-off giữa storage và query flexibility
- **Partition naming convention** impact maintenance và debugging
- **Error handling depth** balance giữa robustness và complexity

9.4. Hướng phát triển.

- **Short-term Improvements**
- **Native partitioning** integration với PostgreSQL 11+ features
- **Automatic rebalancing** cho range partitions khi data skewed
- **Query optimizer** để route queries tới correct partitions
- **Monitoring dashboard** cho partition health và performance
- **Long-term Enhancements**
- **Distributed partitioning** across multiple database servers
- **Adaptive partitioning** tự động adjust strategy dựa trên workload
- **Compression techniques** để optimize storage usage
- **Machine learning** predictions cho optimal partition strategies

9.5. Đánh giá tổng thể.

Đề tài đã đạt được **100% objectives** ban đầu và provide foundation solid cho:

Hiểu biết sâu về data partitioning concepts

Practical experience với large-scale data processing

Skills trong database optimization và performance tuning

Knowledge về distributed systems principles

Impact: Kiến thức và experience gained có thể apply cho:

Big data processing systems (Hadoop, Spark)

Distributed databases (MongoDB, Cassandra)

Cloud data services (AWS RDS, Google BigQuery)

Enterprise data warehousing solutions

10. TÀI LIỆU THAM KHẢO

10.1. Books & Publications

Elmasri, R., & Navathe, S. (2015). *Fundamentals of Database Systems* (7th ed.). Pearson.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Database System Concepts* (7th ed.). McGraw-Hill.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). *Database Systems: The Complete Book* (2nd ed.). Prentice Hall.

10.2. Technical Documentation

PostgreSQL Documentation. (2021). *Partitioning*. Retrieved from https://www.postgresql.org/docs/current/ddl-partitioning.html

Oracle Corporation. (2020). *Database Concepts - Partitioning*. Oracle Database Documentation.

Microsoft. (2021). *Partitioned Tables and Indexes*. SQL Server Documentation.

10.3. Research Papers

DeWitt, D., & Gray, J. (1992). "Parallel database systems: The future of high performance database systems." *Communications of the ACM*, 35(6), 85-98.

Stonebraker, M. (1986). "The case for shared nothing." *IEEE Database Engineering Bulletin*, 9(1), 4-9.

Pavlo, A., et al. (2009). "A comparison of approaches to large-scale data analysis." *Proceedings of the 2009 ACM SIGMOD*.

10.4. Online Resources

Stack Overflow. (2021). *PostgreSQL Partitioning Best Practices*. Community Q&A.

GitHub. (2021). *Database Partitioning Examples*. Open source repositories.

Medium. (2020). *Database Sharding vs Partitioning*. Technical blogs.

10.5. Tools & Technologies

psycopg2-binary: PostgreSQL adapter cho Python (main dependency)

PostgreSQL 12+: Open-source relational database

Python 3.12+: Programming language

pgAdmin: PostgreSQL administration tool

Interface.py: Main implementation module (core functions)