

Query Processing and Optimization Report

Introduction:

The purpose of this report is to showcase various examples of query optimizations implemented in the `part3.py` file. The examples are based on a supply chain database schema.

Contents:

1. Python Scripts: Used for creating the database, connecting to it, executing SQL schema scripts, inserting data, performing database fragmentation, and database replication.
2. SQL Scripts: Contains the SQL code for creating tables (`main_schema.sql`) and inserting sample data (`insert_data.sql`). These tables include Products, Warehouses, Suppliers, Orders, OrderDetails, Shipments, and InventoryLevels.

Setup and Execution:

Prerequisites

Python 3.x

PostgreSQL Database Server

psycopg2 library (Python PostgreSQL adapter)

Installation Steps

Install psycopg2: This Python library is essential for connecting to and interacting with PostgreSQL databases. Install it using pip:

```
pip install psycopg2
```

PostgreSQL Server: Ensure you have PostgreSQL server installed and running on your system.

How to Use

1. Clone the Repository: Clone this repository to your local machine.
2. Database Creation: Run the `create_database()` function from the Python script to create a new database for the supply chain management.
3. Schema Execution: Execute the `execute_schema_script()` function to create the necessary tables defined in the `main_schema.sql`.

4. Data Insertion: Use the `execute_insert_script()` function to insert sample data into the database from `insert_data.sql`.
5. Example Query: Use the `example_query()` function to run an example query and results along with the time taken for the query to run.

Database Structure:

Tables:

Products: Stores information about the products.

Warehouses: Information about warehouses.

Suppliers: Details of suppliers.

Orders: Customer orders.

OrderDetails: Details of each order.

Shipments: Information about product shipments.

InventoryLevels: Inventory levels in each warehouse.

Database Schema:



PostgreSQL Connection:

```
jaytwm@Jays-MacBook-Pro ISP_Part3 % psql -U postgres
psql (14.9 (Homebrew))
Type "help" for help.

postgres=# \c supply_chain
You are now connected to database "supply_chain" as user "postgres".
supply_chain=# \dt
               List of relations
Schema |      Name      | Type  | Owner
-----+-----+-----+-----
public | inventorylevels | table | postgres
public | orderdetails   | table | postgres
public | orders         | table | postgres
public | products       | table | postgres
public | shipments      | table | postgres
public | suppliers      | table | postgres
public | warehouses     | table | postgres
(7 rows)

supply_chain=#
```

Examples and Optimizations:

1. Creating Index on ProductID:

- **Objective:** Improve retrieval speed for queries involving the `ProductID` column in the `InventoryLevels` table.
- **Implementation:** Created an index named `idx_inventory_product_id` on the `ProductID` column.

```
Index 'idx_inventory_product_id' created successfully!
```

2. INNER JOIN Optimization:

- **Objective:** Retrieve inventory levels along with product details.
- **Implementation:** Utilized INNER JOIN between the `InventoryLevels` and `Products` tables.

```

Example 2 - INNER JOIN:
('00000000-0000-0000-0004-000000000004', '00000000-0000-0000-0007-000000000007', 50, 20, 100, 30, 'Smartphone X', 'Latest smartphone model', 'Electronics', Decimal('799.99'))
('00000000-0000-0000-0004-000000000004', '00000000-0000-0000-0008-000000000008', 30, 15, 80, 25, 'Laptop Pro', 'High-performance laptop', 'Electronics', Decimal('1299.99'))
('00000000-0000-0000-0005-000000000005', '00000000-0000-0000-0009-000000000009', 15, 5, 50, 10, 'Fashionable Jacket', 'Warm and stylish jacket', 'Clothing and Fashion', Decimal('79.99'))
('00000000-0000-0000-0006-000000000006', '00000000-0000-0000-0010-000000000010', 100, 30, 150, 50, 'Bestseller Novel', 'Compelling story', 'Books', Decimal('19.99'))
Time taken: 0.0003781318664550781

```

3. LEFT JOIN Optimization:

- **Objective:** Retrieve inventory levels with or without corresponding product details.
- **Implementation:** Utilized LEFT JOIN between the `InventoryLevels` and `Products` tables.

```

Example 3 - LEFT JOIN:
('00000000-0000-0000-0004-000000000004', '00000000-0000-0000-0007-000000000007', 50, 20, 100, 30, 'Smartphone X', 'Latest smartphone model', 'Electronics', Decimal('799.99'))
('00000000-0000-0000-0004-000000000004', '00000000-0000-0000-0008-000000000008', 30, 15, 80, 25, 'Laptop Pro', 'High-performance laptop', 'Electronics', Decimal('1299.99'))
('00000000-0000-0000-0005-000000000005', '00000000-0000-0000-0009-000000000009', 15, 5, 50, 10, 'Fashionable Jacket', 'Warm and stylish jacket', 'Clothing and Fashion', Decimal('79.99'))
('00000000-0000-0000-0006-000000000006', '00000000-0000-0000-0010-000000000010', 100, 30, 150, 50, 'Bestseller Novel', 'Compelling story', 'Books', Decimal('19.99'))
Time taken: 0.00024175643920898438

```

4. WHERE Clause with Index:

- **Objective:** Optimize queries filtering by a specific category in the `Products` table.
- **Implementation:** Used a WHERE clause with an index on the `Category` column.

```

Example 4 - WHERE clause with Index:
('Smartphone X', Decimal('799.99'))
('Laptop Pro', Decimal('1299.99'))
Time taken: 0.00023293495178222656

```

5. Aggregation Optimization:

- **Objective:** Count the total number of products in each warehouse.
- **Implementation:** Utilized the GROUP BY clause for aggregation.

```

Example 5 - Aggregation:
('00000000-0000-0000-0006-000000000006', 1)
('00000000-0000-0000-0004-000000000004', 2)
('00000000-0000-0000-0005-000000000005', 1)
Time taken: 0.0004067420959472656

```

6. LIMIT Clause Optimization:

- **Objective:** Retrieve a limited number of recent orders.
- **Implementation:** Used the LIMIT clause in combination with ORDER BY to limit query results.

Example 6 - LIMIT:

```
('00000000-0000-0000-0014-000000000014', datetime.date(2023, 11, 4), 'Sophia Davis')  
( '00000000-0000-0000-0013-000000000013', datetime.date(2023, 11, 3), 'Michael Smith')  
( '00000000-0000-0000-0012-000000000012', datetime.date(2023, 11, 2), 'Emily Johnson')  
( '00000000-0000-0000-0011-000000000011', datetime.date(2023, 11, 1), 'John Gomez')  
Time taken: 0.0003299713134765625
```

7. EXISTS Clause Optimization:

- **Objective:** Retrieve product names for products existing in the `InventoryLevels` table.
- **Implementation:** Used the EXISTS clause to check for the existence of related records.

Example 7 - EXISTS:

```
('Smartphone X',)  
( 'Laptop Pro',)  
( 'Fashionable Jacket',)  
( 'Bestseller Novel',)  
Time taken: 0.0003421306610107422
```

Conclusion:

The provided examples demonstrate various query optimization techniques, including indexing, joining tables, using WHERE clauses with indexes, aggregation, limiting results, and checking for existence. These optimizations aim to enhance the speed and efficiency of real-time inventory tracking and supply chain analysis queries.

Future Considerations:

Continued optimization efforts may involve periodic analysis of query execution plans using the `EXPLAIN` statement to identify potential areas for improvement. Additionally, regular monitoring and profiling of database performance can guide further optimization strategies based on evolving data patterns and query requirements.

This report provides a foundation for optimizing queries in the supply chain database, and further enhancements can be tailored based on specific use cases and evolving business needs.