**Lab Exercise 22- Performance Tuning Best Practices in PostgreSQL**

**Objective**

* Understand and apply key PostgreSQL performance tuning best practices
* Analyze query performance using tools like EXPLAIN ANALYZE
* Optimize PostgreSQL configuration parameters for workload-specific performance

**Prerequisites**

* PostgreSQL installed and running
* Access to psql CLI or pgAdmin
* A test database with sample data (e.g., employees table with thousands of rows)

**Step 1: Baseline Query Performance**

1. Create a sample table with data:

CREATE TABLE employees (

id SERIAL PRIMARY KEY,

name TEXT,

department TEXT,

salary NUMERIC

);

-- Insert sample data (adjust number as needed)

INSERT INTO employees (name, department, salary)

SELECT

'Employee\_' || g,

CASE WHEN g % 3 = 0 THEN 'HR'

WHEN g % 3 = 1 THEN 'Engineering'

ELSE 'Finance' END,

round(random() \* 100000)

FROM generate\_series(1, 100000) AS g;

1. Run a query and measure performance:

EXPLAIN ANALYZE

SELECT \* FROM employees WHERE department = 'Engineering';

Take note of the execution time.

**Step 2: Add Indexes to Improve Query Time**

1. Create an index on the department column:

CREATE INDEX idx\_department ON employees(department);

1. Run the same query again with EXPLAIN ANALYZE:

EXPLAIN ANALYZE

SELECT \* FROM employees WHERE department = 'Engineering';

Compare execution time with the previous result.

**Step 3: Analyze and Vacuum**

1. Check if statistics are up to date:

ANALYZE VERBOSE employees;

1. Run VACUUM to reclaim storage:

VACUUM VERBOSE employees;

**Step 4: Tune PostgreSQL Configuration Parameters**

Edit postgresql.conf (requires restart):

Adjust these based on your system memory and workload:

shared\_buffers = 25% of total RAM

work\_mem = 4MB (adjust per connection)

maintenance\_work\_mem = 64MB

effective\_cache\_size = 50%-75% of total RAM

random\_page\_cost = 1.1 (on SSDs)

Restart PostgreSQL:

systemctl restart postgresql

**Step 5: Enable and Use pg\_stat\_statements**

1. Add to postgresql.conf:

shared\_preload\_libraries = 'pg\_stat\_statements'

1. Restart PostgreSQL, then create the extension:

CREATE EXTENSION pg\_stat\_statements;

1. Query slowest queries:

SELECT query, calls, total\_time, mean\_time

FROM pg\_stat\_statements

ORDER BY total\_time DESC

LIMIT 5;

**Step 6: Parallel Query Test**

Run a query with aggregation:

EXPLAIN ANALYZE

SELECT department, AVG(salary) FROM employees GROUP BY department;

Check if it’s using parallelism (Parallel Seq Scan). If not, tune:

max\_parallel\_workers\_per\_gather = 4

parallel\_setup\_cost = 1000

parallel\_tuple\_cost = 0.1

**Summary**

In this lab, you:

* Measured and improved query performance using indexes
* Tuned PostgreSQL memory and parallelism settings
* Used tools like EXPLAIN ANALYZE and pg\_stat\_statements
* Applied vacuum and analyze to keep statistics fresh