**Lab Exercise 11- Vacuuming & Autovacuum Configuration**

* Understand how PostgreSQL handles dead tuples
* Learn to use VACUUM, VACUUM FULL, and ANALYZE
* Configure and observe the autovacuum process

**Prerequisites**

* PostgreSQL installed on Windows
* SQL Shell (psql) or pgAdmin access
* Admin rights to edit postgresql.conf and restart the PostgreSQL service

**Step 1: Create Test Table and Insert Data**

CREATE TABLE cleanup\_test (

id SERIAL PRIMARY KEY,

data TEXT

);

-- Insert 10000 rows

INSERT INTO cleanup\_test (data)

SELECT 'row ' || i FROM generate\_series(1, 10000) AS i;

**Step 2: Delete Rows to Generate Dead Tuples**

DELETE FROM cleanup\_test WHERE id <= 9000;

Now, the table has 9000 dead rows, but the space is not reclaimed yet.

**Step 3: Check Table Statistics**

SELECT relname AS table, n\_tup\_ins, n\_tup\_upd, n\_tup\_del, n\_dead\_tup

FROM pg\_stat\_user\_tables

WHERE relname = 'cleanup\_test';

n\_dead\_tup shows the number of dead tuples.

**Step 4: Run Manual VACUUM**

**VACUUM cleanup\_test;**

Re-check the statistics:

**SELECT relname, n\_dead\_tup FROM pg\_stat\_user\_tables WHERE relname = 'cleanup\_test';**

**What Happened:**

* VACUUM cleaned the dead rows but didn’t shrink the table size (space is marked reusable).

**Step 5: VACUUM FULL to Reclaim Disk Space**

VACUUM FULL cleanup\_test;

VACUUM FULL rewrites the table and reclaims space. This **requires a lock** on the table.

**Step 6: Run ANALYZE**

ANALYZE cleanup\_test;

This updates statistics used by the planner for optimized queries.

**Step 7: View Autovacuum Settings**

SHOW autovacuum;

SHOW autovacuum\_vacuum\_threshold;

SHOW autovacuum\_vacuum\_scale\_factor;