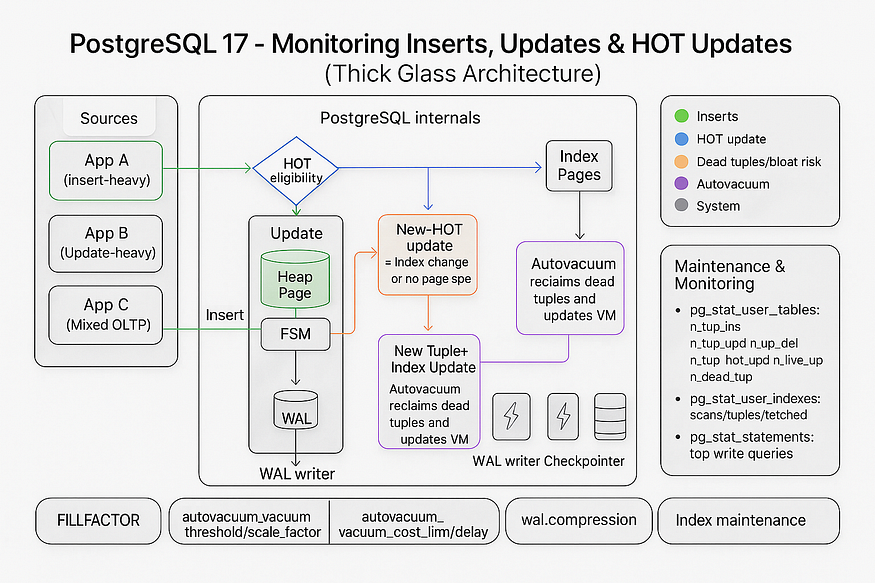
# **23 - PostgreSQL 17 Performance Tuning: Monitoring Inserts, Updates, and HOT Updates**



When tuning PostgreSQL, it is very important to understand the ****INSERT, UPDATE, and DELETE patterns**** of your tables. Different workloads create different storage challenges — for example, some tables may only ever receive inserts, while others may be heavily updated.

By analyzing these patterns, you can decide whether you need to run ****reindexing****, ****CLUSTER****, or whether you are benefiting from ****HOT (Heap-Only Tuple) updates****, which significantly improve update performance.

## **Step 1: Create a Large Products Table**

For this demo, let’s work with a products table containing 10 million rows:

CREATE TABLE products (  
 product\_id BIGSERIAL PRIMARY KEY,  
 product\_name TEXT,  
 category TEXT,  
 price NUMERIC(10,2),  
 stock\_qty INT  
);

postgres=# CREATE TABLE products (  
 product\_id BIGSERIAL PRIMARY KEY,  
 product\_name TEXT,  
 category TEXT,  
 price NUMERIC(10,2),  
 stock\_qty INT  
);  
CREATE TABLE  
postgres=#

-- Insert 10 million rows  
INSERT INTO products (product\_name, category, price, stock\_qty)  
SELECT  
 'Product\_' || g,  
 'Category\_' || (g % 50),  
 (random()\*500)::NUMERIC(10,2),  
 (random()\*100)::INT  
FROM generate\_series(1, 10000000) g;  
ANALYZE products;

postgres=# -- Insert 10 million rows  
INSERT INTO products (product\_name, category, price, stock\_qty)  
SELECT  
 'Product\_' || g,  
 'Category\_' || (g % 50),  
 (random()\*500)::NUMERIC(10,2),  
 (random()\*100)::INT  
FROM generate\_series(1, 10000000) g;  
ANALYZE products;  
INSERT 0 10000000  
ANALYZE  
postgres=#

## **Step 2: Run Inserts and Updates**

We’ll simulate different workloads:

-- Insert new products (append-only workload)  
INSERT INTO products (product\_name, category, price, stock\_qty)  
SELECT  
 'New\_Product\_' || g,  
 'Category\_' || (g % 50),  
 (random()\*500)::NUMERIC(10,2),  
 (random()\*100)::INT  
FROM generate\_series(1, 100000) g;

postgres=# -- Insert new products (append-only workload)  
INSERT INTO products (product\_name, category, price, stock\_qty)  
SELECT  
 'New\_Product\_' || g,  
 'Category\_' || (g % 50),  
 (random()\*500)::NUMERIC(10,2),  
 (random()\*100)::INT  
FROM generate\_series(1, 100000) g;  
INSERT 0 100000  
postgres=#

-- Update stock quantity (update-heavy workload)  
UPDATE products  
SET stock\_qty = stock\_qty + 5  
WHERE category = 'Category\_10';

postgres=# -- Update stock quantity (update-heavy workload)  
UPDATE products  
SET stock\_qty = stock\_qty + 5  
WHERE category = 'Category\_10';  
UPDATE 202000  
postgres=#

-- Delete some rows (sparse blocks may remain)  
DELETE FROM products WHERE category = 'Category\_20';

postgres=# DELETE FROM products WHERE category = 'Category\_20';  
DELETE 202000  
postgres=#

## **Step 3: Inspect Table Modification Stats**

Now, let’s check how many inserts, updates, and deletes have been performed on products.

SELECT relname,  
 n\_tup\_ins AS inserts,  
 n\_tup\_upd AS updates,  
 n\_tup\_del AS deletes,  
 n\_tup\_hot\_upd AS hot\_updates  
FROM pg\_stat\_user\_tables  
WHERE relname = 'products';

📊 ****Sample Output:****

postgres=# SELECT relname,  
 n\_tup\_ins AS inserts,  
 n\_tup\_upd AS updates,  
 n\_tup\_del AS deletes,  
 n\_tup\_hot\_upd AS hot\_updates  
FROM pg\_stat\_user\_tables  
WHERE relname = 'products';  
 relname | inserts | updates | deletes | hot\_updates  
----------+----------+---------+---------+-------------  
 products | 10100000 | 404000 | 202000 | 2  
(1 row)  
  
postgres=#

## **Step 4: Interpreting the Results**

* ****Inserts**** → show how many rows were added. Tables that only grow (append-only) are generally simpler to maintain.
* ****Updates**** → indicate how often rows are modified. Heavy updates increase the chance of ****dead tuples****, requiring VACUUM and possibly reindexing.
* ****Deletes**** → reveal potential for ****sparse blocks****, where space is wasted. In such cases, you may consider running CLUSTER to fully reorganize the table.
* ****HOT Updates**** → these are the most interesting metric. HOT (Heap-Only Tuple) updates mean that PostgreSQL was able to update a row ****without moving it to a new page****, keeping it in the same block.

👉 A ****high percentage of HOT updates**** is a very good sign because:

* Updates stay local to the page.
* Fewer index changes are needed.
* Performance is better, especially for update-heavy workloads.

## **Why HOT Updates Matter**

Normally, PostgreSQL creates a new row version for each update, even if only one column changes. This means more dead tuples, more index churn, and more VACUUM overhead.

With HOT updates:

* The row version stays ****inside the same page****.
* Index entries don’t need to be updated.
* VACUUM has less work to do.

The general rule is:

* ****If your workload is update-heavy, aim for a high HOT update ratio.****
* This shows PostgreSQL is efficiently handling updates without bloating indexes.

✅ By monitoring ****inserts, updates, deletes, and HOT updates**** in pg\_stat\_user\_tables, you can clearly see how your workload behaves and adjust your ****maintenance strategy**** (VACUUM, REINDEX, CLUSTER) accordingly.