Gaia Postgres Service / EPAS Training

January 2026

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

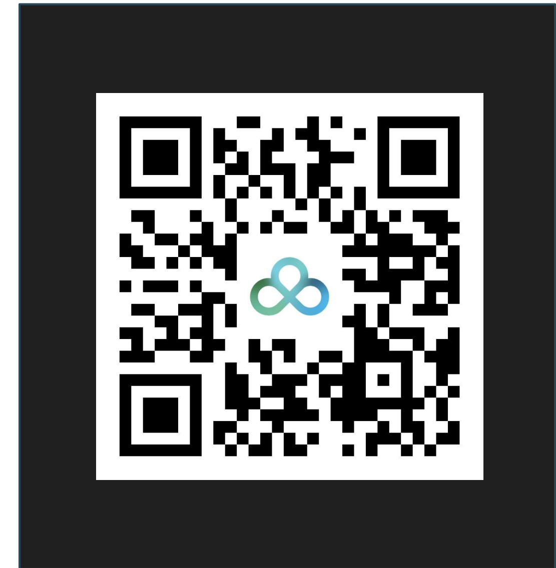
Speaker Introduction

Nick Ivanov

Resident Architect

Nick is a seasoned solutions architect at EDB. Since April 2022 he has brought extensive expertise in database architecture and analytics from a notable tenure at IBM from May 2010 to March 2022.

He holds a Dipl.-Ing. degree in computing systems and networks from Bauman Moscow State Technical University.



Connect on **Linkedin**

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Speaker Introduction

Matthew Chaddock

Snr Lead Software Engineer

Matt is a Senior Lead Software Engineer on the Infrastructure Platforms – Cloud Relational Databases team. He currently leads the Gaia Postgres Service software engineering team, based in Bournemouth, UK.

Matt joined JPMC in 2006 through the Graduate Programme after graduating from Bournemouth University. Over his tenure, he has worked extensively with a variety of database technologies, including Sybase, Oracle, MS SQL, and—most notably—Postgres, which remains his favorite.

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Speaker Introduction

Sunita Behera

Lead Software Engineer

Sunita is a Lead Software Engineer with nearly 10 years of experience, specializing in Python, Django, and database platforms. She leads the Gaia Postgres Service within JPMorgan Chase’s Infrastructure Platforms – Cloud Relational Databases team in Bournemouth, UK.

Since joining JPMC in 2019, Sunita has worked across the bank’s database products—including Dbaasnow, Moonraker, DBPortal, and now Gaia Postgres—delivering scalable, secure, high-performance services that enable teams across the firm.

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Agenda

- PostgreSQL concepts
- PostgreSQL vs EDB Postgres Advanced Server
- Gaia Postgres Service implementation details
- Provisioning Postgres instances
- Managing Postgres instances
- Data import and export
- Monitoring and Troubleshooting
- Performance tuning

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Concepts

PostgreSQL Concepts

- Lingo
- MVCC
- WAL
- Replication
- Connection management

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Lingo

Instance	A running PostgreSQL server process
Cluster	A collection of databases managed by one instance
Also cluster	A group of PostgreSQL servers (nodes) in an HA configuration
WAL	Write-Ahead Log (transaction log)
Relation	Table or view
Tuple	Row
Checkpoint	Writing “dirty” data and index pages from the shared memory buffer to disk

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Concepts: MVCC

- Readers don't block writers; writers don't block readers
- Each tuple carries the transaction ID (txid) that modified it; multiple versions of a tuple exist
- Transactions only consider tuples with txid older than their own
- UPDATE = DELETE + INSERT
- VACUUM
  - frees "dead" (obsolete deleted) tuples
  - updates visibility maps
  - updates statistics
  - txid freeze
- Side effects
  - Bloat (too many dead tuples)
  - txid wraparound risk

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Concepts: Write-Ahead Log

- Ensures durability and consistency of changes
- `min_wal_size` preallocated at startup; can grow to `max_wal_size` for large transactions
- Reaching `max_wal_size` triggers a checkpoint
  - Blocks incoming transactions
- Inactive WAL files are recycled or removed
- WAL files also drive replication
  - Inactive replicas will cause WALs to accumulate
- Running out of WAL disk space stops the database instance

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Concepts: Replication

- Physical
  - Transmits complete WAL records for replicas to replay
  - All databases in the instance are replicated
  - Useful for HA
- Logical
  - Transmits logical operations (insert/update/delete)
  - Can filter by database, table, row
  - Useful for CDC, maintaining a DW, integration
  - Not mutually exclusive with physical replication
- Replication slots on the upstream server track each replica (subscriber) progress

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Concepts: Connection management

- Each client connection (session) starts a separate background process
  - Typically ~2.5+ MB overhead even if idle
- Consider connection pooling when reaching 300+ concurrent connections
- “Autocommit” is implicit, i.e. each statement runs in its own transaction
  - Unless `BEGIN` is issued
  - Then until `COMMIT` or `ROLLBACK`
- Many performance configuration parameters can be set at the session level
  - E.g. `SET work_mem = 2GB; SELECT ...`

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

EDB Postgres Advanced Server vs. PostgreSQL

Enterprise Postgres

- **Oracle Compatibility** - Compatibility for schemas, data types, indexes, users, roles, partitioning, packages, views, PL/SQL triggers, stored procedures, functions, and utilities
- **Enterprise Security** - Transparent Data encryption, Virtual Private Databases, password policy management, built-in auditing, data redaction, SQL injection protection, and procedural language code obfuscation
- **Developer Productivity** - Over 200 pre-packaged utility functions, user-defined object types, autonomous transactions, SPL debugger
- **DBA Productivity** - Resource Manager (throttle CPU and I/O at the session level), instance and session wait event history, declarative policy management, automatic table partitioning
- **Performance** - Query optimizer hints, SQL Profiler, Query Advisor, dynamic resource tuning
- **Replication Enhancements** - Enables EDB Postgres Distributed functionality such as Group Commit, Commit at Most Once and Eager all-node synchronous replication etc.

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Enterprise Postgres vs. PostgreSQL

- EPAS is PostgreSQL, but note:
  - EPAS extends supported SQL commands
    - Some names accepted by PostgreSQL are reserved in EPAS (e.g. “connect”, “constructor”)
  - EPAS extends some catalog tables to enable extra functionality (e.g. pg_namespace)
  - Databases not binary compatible (cannot backup/restore across editions)
  - Some 3d party extensions may not work with EPAS

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Gaia Postgres Service

Data Import and Export

Importing data

- Postgres command line tools
  - `psql \COPY` command
  - `pg_restore` (from `pg_dump` data)
- Third party tools
  - DBeaver
  - SQL Developer
- Logical replication
  - From other Postgres instances

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Exporting data

- Postgres command line tools
  - `psql \COPY` command
  - `pg_dump`
- Third party tools
  - DBeaver
  - SQL Developer
- Logical replication
  - To other Postgres instances

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Troubleshooting Postgres Databases

Troubleshooting Postgres

- Built-in monitoring tools
- Grafana (metrics), Splunk (logs)
- Link to engineering knowledge base article

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Postgres Monitoring Interfaces

- Cumulative statistics views (**pg_stat_activity**, **pg_stat_statements**, etc.)
- Configurable to track (or not) row and block counts, I/O timing, activities etc.
- Minimal overhead
- Requires **pg_read_all_stats** role

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Monitoring Views

View name	Contents
`pg_stat_database`	Database-wide metrics
`pg_stat_bgwriter`, `pg_stat_checkpointer`	Server background process statistics
`pg_stat_user_tables`, `pg_stat_user_indexes`	Activities on tables and indexes, like inserts, updates, deletes, vacuum, autovacuum etc.
`pg_stat_user_indexes`	Shows information about index usage for all user tables
`pg_stat_progress_vacuum`, `pg_stat_progress_create_index`, `pg_stat_progress_cluster`	Progress of background operations
`pg_stat_activity`	Connected session states and queries in progress
`pg_locks`	Currently held locks
`pg_stat_statements`	Past statement execution statistics

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_activity

```
postgres=# select * from pg_stat_activity\gx
-[ RECORD 1 ]----+------------------------------------------------------------
datid            | 5
datname          | postgres
pid              | 119179
leader_pid       |
usesysid         | 16388
usename          | efm
application_name | efm-5.1
client_addr      | 10.33.16.147
client_hostname  |
client_port      | 37050
backend_start    | 2026-01-19 15:32:17.690251+00
xact_start       |
query_start      | 2026-01-19 15:32:17.701672+00
state_change     | 2026-01-19 15:32:17.701711+00
wait_event_type  | Client
wait_event       | ClientRead
state            | idle
backend_xid      |
backend_xmin     |
query_id         | 735318608620182175
query            | show server_version
backend_type     | client backend
```


©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_activity

```
postgres=# select datname, application_name, state, count(1)
postgres-# from pg_stat_activity where backend_type = 'client backend'
postgres-# group by 1,2,3 order by 4 desc;
 datname  |                 application_name                 |        state        | count
----------+--------------------------------------------------+---------------------+-------
 bench    | pgbench                                          | active              |     5
 bench    | pgbench                                          | idle in transaction |     5
 postgres | efm-5.2                                          | idle                |     1
 postgres | Postgres Enterprise Manager - Agent Monitoring   | idle                |     1
 postgres | psql                                             | active              |     1
(5 rows)
```


©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_statements

```
-[ RECORD 3 ]----------+-------------------------------------------------------------------------------
userid                 | 10
dbid                   | 5
toplevel               | t
queryid                | 328522144530266311
query                  | SELECT  s.nspname AS schema_name, $1 AS package_name, f.proname AS function_name,
plans                  | 0
total_plan_time        | 0
min_plan_time          | 0
max_plan_time          | 0
mean_plan_time         | 0
stddev_plan_time       | 0
calls                  | 48
total_exec_time        | 49.72249000000001
min_exec_time          | 0.701626
max_exec_time          | 2.86297
mean_exec_time         | 1.0358852083333336
stddev_exec_time       | 0.42440258757541355
rows                   | 0
shared_blks_hit        | 12536
shared_blks_read       | 40
shared_blks_dirtied    | 0
shared_blks_written    | 0
```


©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_statements

```
SELECT
 left(query, 60) AS qry, calls,
 round(total_exec_time::numeric,2) AS total,
 round(max_exec_time::numeric,2) AS max,
 round(mean_exec_time::numeric,2) AS mean
FROM pg_stat_statements
ORDER BY 2 DESC LIMIT 10;
                            qry                              | calls  |   total    |   max   | mean
-------------------------------------------------------------+--------+------------+---------+-------
 begin                                                       | 442922 |     694.50 |   39.50 |  0.00
 commit                                                      | 442918 |     667.50 |   14.89 |  0.00
 UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE a | 442918 | 7197290.98 |  787.33 | 16.25
 UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE ti | 442917 |   21465.83 |  205.34 |  0.05
 INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VA | 442917 |   29962.22 | 1554.47 |  0.07
 UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE b | 442917 |   40545.55 | 1408.07 |  0.09
 SELECT abalance FROM pgbench_accounts WHERE aid = $1        | 442917 |   14346.73 |  154.34 |  0.03
 SELECT * FROM pg_catalog.pg_rewrite WHERE ev_class = $1 AND |   3381 |     281.41 |  140.42 |  0.08
 SELECT * FROM pg_catalog.pg_rewrite WHERE ev_class = $1 AND |   1738 |     173.90 |   81.12 |  0.10
 SELECT version()
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_user_tables

```
-[ RECORD 8 ]-------+-----------------------------------
relid               | 18220
schemaname          | pemdata
relname             | index_statistics
seq_scan            | 98
last_seq_scan       | 2026-01-19 15:38:29.355055+00
seq_tup_read        | 149283
idx_scan            | 18162
last_idx_scan       | 2026-01-19 15:38:29.355055+00
idx_tup_fetch       | 34216
n_tup_ins           | 1539
n_tup_upd           | 18084
n_tup_del           | 0
n_tup_hot_upd       | 17869
n_tup_newpage_upd   | 215
n_live_tup          | 1539
n_dead_tup          | 276
n_mod_since_analyze | 106
n_ins_since_vacuum  | 436
last_vacuum         |
last_autovacuum     | 2026-01-15 10:46:41.501455+00
last_analyze        |
last_autoanalyze    | 2026-01-19 15:32:54.513019+00
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_stat_user_tables

```
SELECT schemaname, relname, seq_scan, seq_tup_read, idx_scan, idx_tup_fetch
FROM pg_stat_user_tables
ORDER BY 3 DESC;

 schemaname |     relname      | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch
------------+------------------+----------+--------------+----------+---------------
 public     | pgbench_accounts |        2 |    100000000 |   877601 |        877601
 public     | pgbench_branches |        2 |         2000 |   438801 |        438800
 public     | pgbench_tellers  |        1 |        10000 |   438800 |        438800
 public     | pgbench_history  |        0 |            0 |          |
(4 rows)

schemaname |     relname      | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch
------------+------------------+----------+--------------+----------+---------------
 public     | pgbench_accounts |        2 |    100000000 |   885835 |        885835
 public     | pgbench_branches |        2 |         2000 |   442918 |        442917
 public     | pgbench_tellers  |        1 |        10000 |   442917 |        442917
 public     | pgbench_history  |        0 |            0 |          |
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_locks

```
postgres=# select * from pg_locks\gx
-[ RECORD 1 ]------+----------------------------------
locktype           | relation
database           | 16863
relation           | 16903
page               |
tuple              |
virtualxid         |
transactionid      |
classid            |
objid              |
objsubid           |
virtualtransaction | 0/3686
pid                | 12988
mode               | RowExclusiveLock
granted            | t
fastpath           | t
waitstart          |
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_locks

```
SELECT
  bl.pid AS blocked_pid,
  ba.query AS blocked_query,
  bl.mode AS blocked_mode,
  bl.relation::regclass AS blocked_table,
  wl.pid AS blocking_pid,
  wa.query AS blocking_query,
  wl.mode AS blocking_mode,
  wl.relation::regclass AS blocking_table
FROM pg_locks bl
JOIN pg_stat_activity ba ON ba.pid = bl.pid
JOIN pg_locks wl
  ON bl.locktype = wl.locktype
  AND bl.database IS NOT DISTINCT FROM wl.database
  AND bl.relation IS NOT DISTINCT FROM wl.relation
  AND bl.pid <> wl.pid
JOIN pg_stat_activity wa ON wa.pid = wl.pid
WHERE NOT bl.granted AND wl.granted
AND ba.wait_event_type = 'Lock' AND wa.wait_event_type <> 'Lock';
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Example: pg_locks

```
-[ RECORD 1 ]--+------------------------------------------------------------------
blocked_pid    | 74226
blocked_query  | UPDATE pgbench_accounts SET abalance = abalance + -1692 WHERE aid = 4019;
blocked_mode   | ShareLock
blocked_table  |
blocking_pid   | 74520
blocking_query | update pgbench_accounts set abalance = abalance + 1;
blocking_mode  | ExclusiveLock
blocking_table |
-[ RECORD 2 ]--+------------------------------------------------------------------
blocked_pid    | 74204
blocked_query  | UPDATE pgbench_accounts SET abalance = abalance + 1815 WHERE aid = 61281;
blocked_mode   | ShareLock
blocked_table  |
blocking_pid   | 74520
blocking_query | update pgbench_accounts set abalance = abalance + 1;
blocking_mode  | ExclusiveLock
blocking_table |
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

EDB Wait States Extension

- Samples session waits
- Configurable sampling interval and retention period

```
edb=# select * from edb_wait_states_sql_statements();
-[ RECORD 1 ]-+----------------------------------------------------
query_id      | -1697985474390439145
dbtime        | 188
waittime      | 188
cputime       | 0
top_waitevent | DataFileRead
query         | vacuum analyze pgbench_accounts
-[ RECORD 2 ]-+----------------------------------------------------
query_id      | 2577670717561330585
dbtime        | 143
waittime      | 52
cputime       | 91
top_waitevent | WALSync
query         | copy pgbench_accounts from stdin with (freeze on)
-[ RECORD 3 ]-+----------------------------------------------------
query_id      | -7684589253409855891
dbtime        | 250
waittime      | 204
cputime       | 46
top_waitevent | WALWrite
query         | alter table pgbench_accounts add primary key (aid)
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

EDB Wait States - Example

```
SELECT left(query, 60) AS qry, dbtime, waittime, cputime, top_waitevent
FROM edb_wait_states_sql_statements()
ORDER BY 3 DESC;
                             qry                              | dbtime | waittime | cputime | top_waitevent
--------------------------------------------------------------+--------+----------+---------+---------------
 UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE a |   1636 |     1587 |      49 | DataFileRead
 alter table pgbench_accounts add primary key (aid)           |    191 |      165 |      26 | DataFileRead
 SELECT n.nspname AS schema_name, c.relname AS table_name, pg |     66 |       66 |       0 | relation
 copy pgbench_accounts from stdin with (freeze on)            |    155 |       64 |      91 | WalSync
 vacuum analyze pgbench_accounts                              |      9 |        8 |       1 | DataFileRead
 UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE b |     33 |        5 |      28 | transactionid
 UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE ti |     31 |        2 |      29 | BufferContent
 INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VA |     12 |        1 |      11 | WALInsert
 SELECT                                                      +|      1 |        0 |       1 |
         schemaname AS schema_name, tablename AS table_name, +|        |          |         |
                                                              |        |          |         |
 SELECT abalance FROM pgbench_accounts WHERE aid = $1         |     21 |        0 |      21 |
 SELECT c.relname AS index_name, r.relname AS table_name, i.i |      1 |        0 |       1 |
 SELECT sw.backend_id, psa.datname AS dbname, psa.usename, sw |      1 |        0 |       1 |
 END                                                          |      2 |        0 |       2 |
 SELECT c.relname AS view_name, c.relkind AS view_type, c.rel |      1 |        0 |       1 |
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

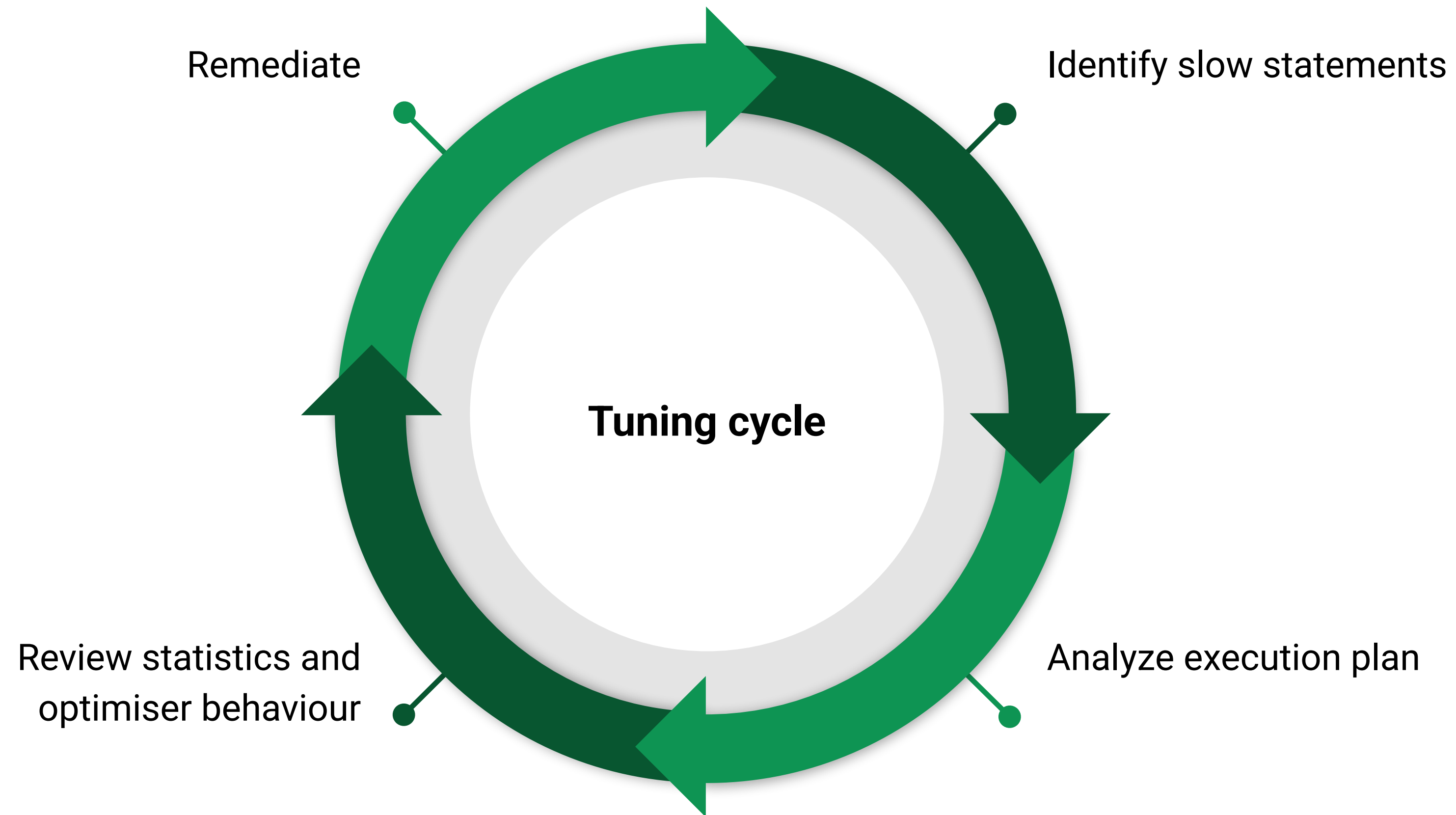
Performance tuning with Postgres

Statement Processing by SQL Engine

- Parse & generate syntax tree
- Rewrite (apply rules, inject view definitions etc.)
- Generate plan alternatives
- Retrieve statistics and calculate cost
- Choose the least expensive plan
- Execute plan nodes

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Performance Tuning Basics



©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Identify Slow Statements

- Logging: **log_min_duration_statement**
- **pg_stat_activity**
- EDB Wait States
- Application-side measurements may be misleading

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Review Execution Plan

- EXPLAIN shows optimiser decision
- EXPLAIN ANALYZE shows actual runtime metrics
  - Can include memory usage, I/O timing, etc.
- Difference between expected and actual row counts can mean “stale” statistics
- Watch for disk-based sorts, full table scans
  - But: table scans aren’t *always* bad
- Plan cost is a relative metric
  - High cost != slow query

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Execution plan example: Table

```
CREATE TABLE sales_mdam_paper
(
  dept INT4,
  sdate DATE,
  item_class SERIAL,
  store INT4,
  item INT4,
  total_sales NUMERIC
);

CREATE INDEX mdam_idx ON sales_mdam_paper(
  dept, sdate, item_class, store
);
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Execution plan example: Query

```
EXPLAIN (ANALYZE, WAL, VERBOSE, BUFFERS)
select
  dept,
  sdate,
  item_class,
  store,
  sum(total_sales)
from
  sales_mdam_paper
where
  sdate between '1995-06-01' and '1995-06-30'
  and item_class in (20, 35, 50)
  and store in (200, 250)
group by dept, sdate, item_class, store
order by dept, sdate, item_class, store;
```


©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Execution plan example: Postgres 18 index skip scan

```
GroupAggregate  (cost=0.43..119.67 rows=1 width=48) (actual time=41.308..41.308 rows=0.00 loops=1)
   Output: dept, sdate, item_class, store, sum(total_sales)
   Group Key: sales_mdam_paper.dept, sales_mdam_paper.sdate, sales_mdam_paper.item_class, sales_mdam_paper.store
   Buffers: shared hit=31 read=53
   ->  Index Scan using mdam_idx on public.sales_mdam_paper  (cost=0.43..119.64 rows=1 width=27) (actual time=41.306..41.306 rows=0.00 loops=1)
         Output: dept, sdate, item_class, store, item, total_sales
         Index Cond: ((sales_mdam_paper.sdate >= '1995-06-01'::date) AND (sales_mdam_paper.sdate <= '1995-06-30'::date) AND (sales_mdam_paper.item_class = ANY ('{20,35,50}'::integer[])) AND (sales_mdam_paper.store = ANY ('{200,250}'::integer[])))
         Index Searches: 26
         Buffers: shared hit=31 read=53
 Planning:
   Buffers: shared hit=6
 Planning Time: 0.181 ms
 Execution Time: 42.099 ms
```

EDB POSTGRES AI

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

Remediation

- ANALYZE tables with outdated statistics
  - Autovacuum may not always keep up with bulk updates
- CREATE STATISTICS for expressions and correlated columns
- Try new indexes
  - Can help avoid table scans and sorts
  - Low cardinality columns are poor candidates
- Review parallelism
- Larger **work_mem** can address disk-based sorts and aggregation
- Consider rewriting the statement
- Use optimiser hints (sparingly) – EPAS only

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

PostgreSQL Indexes

Type	Description
B-tree	Default index type, works for equality and range predicates
Hash	Only equality predicates
GiST	Generalized Search Tree, supports complex data types (e.g. geometric, full-text) and custom operators (e.g. containment)
SP-GiST	Space-Partitioned Generalized Search Tree; appropriate for unbalanced tree structures (hierarchical data), spatial data
GIN	Generalized Inverted Index, suitable for multi-component values (arrays, JSONB)
BRIN	Block Range Index. Suitable for naturally ordered data (e.g. time series), space-efficient.

©EDB 2025, 2026 — ALL RIGHTS RESERVED.

BOURNEMOUTH TECHNOLOGY CENTRE

Gaia Postgres Day Feedback

Thursday, February 26 | 9:00am GMT
The Hub D1S

Scan the QR code or go/BMTHPGD
to provide feedback.

Thank You

Please complete EDB Day A.M.Survey

Gaia Postgres Day Feedback

EDB POSTGRES AI

©EDB 2026 — ALL RIGHTS RESERVED.

JPMorganChase

Glasgow
Technology Centre

Gaia Postgres Day Feedback

Tuesday, March 3 | 9:00am – 5:00pm GMT

L1 Ideation and L10 Event Space
315 Argyle Street, Glasgow

Scan the **QR code** to provide feedback.

Q&A