PostgreSQL

Indexing and Performance Tuning

Surendra Panpaliya

Agenda

PostgreSQL indexing strategies

EXPLAIN ANALYZE vs Oracle EXPLAIN PLAN

VACUUM, ANALYZE, statistics

Agenda

Hands-On:

Identify and tune slow queries using EXPLAIN ANALYZE

Assignment:

Create 2 indexes and show performance improvement

Indexing

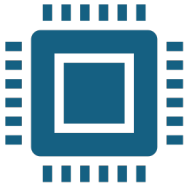
Process of creating data structures

that speed up data retrieval

by minimizing full table scans.

Why Use Indexing?

Improve **query performance**



Reduce **CPU, I/O, and memory usage**

Ensure **scalability** of telecom systems

Handling millions of rows

Indexing Strategies

Strategy	Index Type	Use Case in Telecom
Single-column index	B-Tree	Mobile number lookup
Multi-column index	B-Tree	Recharge by customer/date
Partial index	B-Tree	Active customers only
Expression index	B-Tree	Case-insensitive or computed values

Indexing Strategies

Strategy	Index Type	Use Case in Telecom
BRIN index	BRIN	Billing/usage logs (time-series data)
GIN index	GIN	JSONB search in call logs
Covering index	B-Tree	Index-only scan for performance

Index Types

Index Type	Description	Use Case Example
B-Tree	Default, for equality and range queries	Search by customer_id, recharge_date
Hash	For equality comparisons only	WHERE mobile_number = '9999999999'
GIN	For full-text search and JSONB	Search call logs stored in JSONB format
GiST	For geometric/spatial indexing	Location-based telecom towers (PostGIS)

Index Types

Index Type	Description	Use Case Example
BRIN	For very large, sequential datasets (billing logs)	Time-series usage or billing data
Partial	Only for filtered rows (e.g., active customers)	WHERE status = 'active'
Expression	Indexes on computed columns	LOWER(email) or DATE_TRUNC('month', bill_date)

B-Tree vs. Hash Index

Feature	B-Tree Index	Hash Index
Equality lookups	✅ Good	✅ Very Fast
Range queries	✅ Yes	❌ No
Ordering support	✅ Yes (ORDER BY)	❌ No
Index size	Medium	Small
Write performance	Slightly slower	Slightly faster
Best use case	Equality + range	Equality only, very frequent lookups

EXPLAIN ANALYZE vs EXPLAIN PLAN

Feature	PostgreSQL EXPLAIN ANALYZE	Oracle EXPLAIN PLAN FOR
Purpose	Show the actual query execution plan	Show the estimated execution plan
Actual Timing	✅ Yes (real execution with timings)	❌ No (estimated only, unless traced)
Buffers/Costs	✅ Shows cost, rows, buffers, time	✅ Shows cost and estimated rows
Execution	✅ Executes the query	❌ Only parses/estimates, doesn't execute

WHY Use Execution Plans?

Diagnose

optimize slow queries,

especially in large

telecom datasets.

Benefits

Understand if indexes are used

Identify bottlenecks (e.g., sequential scans, nested loops)

Reduce resource consumption and improve performance

EXPLAIN ANALYZE vs EXPLAIN PLAN

Feature	PostgreSQL (EXPLAIN ANALYZE)	Oracle (EXPLAIN PLAN FOR)
Executes the query	✅ Yes	❌ No
Shows actual time taken	✅ Yes	❌ No
Uses ANALYZE keyword	✅ Required	❌ Not applicable
Real vs estimated rows comparison	✅ Yes	❌ No
Buffers & Disk I/O stats	✅ Yes (with BUFFERS option)	❌ Not shown by default

Use Cases for Execution Plan Analysis

Use Case	Analysis Goal	Plan Utility
Recharge summary by date & circle	Detect full table scan vs index scan	Use EXPLAIN ANALYZE/PLAN
Top N data users per month	Ensure proper sort/order method used	Look for Sort, Index usage
Daily billing reports	Avoid unnecessary joins	Check Nested Loop, Merge Join
JSON-based CDR filtering	Confirm GIN index usage	GIN/Seq Scan identification
Active user filter	Validate Partial Index usage	See WHERE filter in plan

Summary

Feature/Capability	PostgreSQL (EXPLAIN ANALYZE)	Oracle (EXPLAIN PLAN)
Real execution time	✅ Yes	❌ No
Optimizer estimation only	✅ with EXPLAIN only	✅ Default
Tool to display plan	Native (psql)	DBMS_XPLAN.DISPLAY
Common in telecom queries	Lookup, filters, joins, analytics	Same

WHAT is VACUUM, ANALYZE, and Statistics?

Tool	Description
VACUUM	Reclaims storage from dead tuples (deleted/updated rows)
ANALYZE	Updates statistics for query planner to make better decisions
Statistics	Metadata about table data (row count, distinct values, value distribution)

WHY Are They Important?

PostgreSQL uses

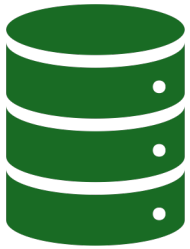
MVCC (Multi-Version Concurrency Control):

updates don’t overwrite rows

they mark old ones as dead.

WHY Are They Important?

Without VACUUM,



disk bloat and

performance degradation occur.

WHY Are They Important?

Without ANALYZE,

the query planner

might **choose bad plans**

(e.g., sequential scan instead of index scan).

TYPES of VACUUM

Command	Description
VACUUM	Cleans dead tuples (but doesn’t update stats)
ANALYZE	Updates planner statistics only
VACUUM ANALYZE	Cleans dead tuples and updates stats
VACUUM FULL	Rewrites table to shrink disk usage (exclusive lock)

Summary Table

Command	When to Use	Impact
VACUUM	Regular cleanup (minimal locking)	Reclaims dead tuples
ANALYZE	After large inserts/updates	Updates planner stats
VACUUM ANALYZE	Routine table maintenance	Reclaims + updates stats
VACUUM FULL	After mass deletions	Shrinks table size on disk

PostgreSQL Indexing Strategies

Index Type	Use Case
B-Tree (Default)	For equality and range queries
Hash Index	For simple equality comparisons (rarely used, not WAL-logged by default)
GIN Index (Generalized Inverted Index)	For full-text search, JSONB
GiST Index (Generalized Search Tree)	For geometric data, range types
BRIN Index (Block Range Index)	For very large tables where data is naturally ordered

Difference

PostgreSQL EXPLAIN ANALYZE	Oracle EXPLAIN PLAN
Shows actual execution with timing	Only shows predicted plan (unless AUTOTRACE is used)
Shows buffer reads, loops, and execution time	Displays cost, cardinality, and rows

VACUUM, ANALYZE, and Statistics

PostgreSQL uses MVCC

(Multi-Version Concurrency Control),

so it needs regular cleanup

to avoid **table bloat** and

keep stats updated.

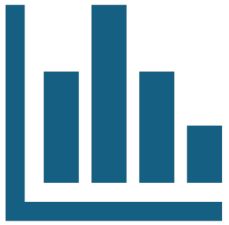
VACUUM

Removes dead tuples

Frees up space

Prevents table bloating

ANALYZE



Updates **statistics**

used by the query planner

to make decisions.

VACUUM ANALYZE

Combines both:

```
VACUUM ANALYZE customers;
```

Performance Tuning Workflow

Step	Action	Command
1	Create Index	CREATE INDEX
2	Run Query with EXPLAIN ANALYZE	EXPLAIN ANALYZE SELECT ...
3	Tune query or schema	Modify indexes or query
4	Maintain DB	VACUUM ANALYZE

Summary

Concept	PostgreSQL Command	Oracle Equivalent
Index Creation	CREATE INDEX	CREATE INDEX
Query Plan	EXPLAIN ANALYZE	EXPLAIN PLAN + DBMS_XPLAN.DISPLAY
Statistics Update	ANALYZE	DBMS_STATS.GATHER_TABLE_STATS
Cleanup	VACUUM	Auto-managed in Oracle

PostgreSQL Term	Meaning (Oracle Context)
Seq Scan	Full Table Scan
Index Scan	Index Range Scan
Bitmap Index Scan	Similar to Oracle Bitmap Index Access
cost=0.00..10000.00	Estimated cost (Oracle shows cost too)
actual time=0.050..10.000	Real execution time (Oracle needs AUTOTRACE or SQL TRACE for this)
rows=50	Number of rows returned
loops=1	Number of times operation repeated

Performance Tuning

Process of optimizing

database performance

by analyzing and improving:

Query execution time

Performance Tuning

Disk I/O

CPU usage

Memory usage

Index efficiency

WHY Use Indexing and Tuning?

Benefit	Impact
Faster query response	Especially with large telecom datasets
Reduced I/O cost	Avoids full table scans
Efficient use of CPU/memory	Better planning and caching
Better user experience	Essential for dashboards, billing systems
Cost savings	Lowers compute usage on cloud/datacenter

TYPES of Indexes

Index Type	Description	Use Case Example
B-Tree	Default, for equality and range queries	Search by customer_id, recharge_date
Hash	For equality comparisons only	WHERE mobile_number = '9999999999'
GIN	For full-text search and JSONB	Search call logs stored in JSONB format
GiST	For geometric/spatial indexing	Location-based telecom towers (PostGIS)

TYPES of Indexes

Index Type	Description	Use Case Example
BRIN	For very large, sequential datasets (billing logs)	Time-series usage or billing data
Partial	Only for filtered rows (e.g., active customers)	WHERE status = 'active'
Expression	Indexes on computed columns	LOWER(email) or DATE_TRUNC('month', bill_date)

Telecom Use Cases

Surendra Panpaliya

Customer Search by ID or Mobile Number

CREATE INDEX idx_customers_mobile ON telecom.customers(mobile_number);

Why: Frequently used in login, support queries, fraud checks.

Filter Recharge Logs by Date

```
CREATE INDEX idx_recharge_date ON
telecom.recharges(recharge_date);
```

Why: Improves filtering by date in reports and dashboards.

Top Users by Data Usage

CREATE INDEX idx_usage_customer_date ON telecom.usage(customer_id, usage_date);

Why: Used in analytical queries for customer data consumption patterns.

Partial Index for Active Customers Only

```
CREATE INDEX idx_active_customers ON
telecom.customers(customer_id)
WHERE status = 'active’;
```

Why: Saves space and speeds up queries on active users only.

Index for Call Data Stored as JSONB

CREATE INDEX idx_call_logs_json ON telecom.call_logs USING GIN (details_jsonb);

Why: Required for efficient filtering inside JSON-based CDR (Call Detail Records).

Range Queries on Billing Amounts

```
CREATE INDEX idx_billing_amount ON telecom.billing(bill_amount);
```

Why: Speeds up filtering users with high billing or discounts.

Performance Tuning Tips in PostgreSQL

Technique	Description / Use Case
EXPLAIN ANALYZE	Analyze query plans and detect slow operations
VACUUM / ANALYZE	Reclaim storage, update planner statistics
pg_stat_statements	Extension to track slow and frequent queries
Connection pooling	Use PgBouncer or built-in pooler for high concurrency

Performance Tuning Tips in PostgreSQL

Technique	Description / Use Case
Tune work_mem, shared_buffers	Allocate enough memory for sorting, joins
Partitioning large tables	Split usage/billing tables by month or circle
Index-only scans	Ensure queries only need indexed columns (no table access)

Real Example: Query Optimization

Surendra Panpaliya

Slow Query:

SELECT * FROM telecom.usage WHERE customer_id = 1001 AND usage_date >= '2025-07-01’;

Improvement

CREATE INDEX idx_usage_cust_date ON telecom.usage(customer_id, usage_date);

Slow Query:

Re-run:

EXPLAIN ANALYZE

SELECT * FROM telecom.usage WHERE customer_id = 1001 AND usage_date >= '2025-07-01’;

You should see Index Scan instead of Seq Scan.

Summary Table

Concept	PostgreSQL Feature	Telecom Example
Point lookup	B-tree index	WHERE customer_id = 101
Time-series billing	BRIN/Partitioning	Monthly usage and billing data
Status filters	Partial Index	status = 'active'
Geospatial search	GiST with PostGIS	Cell tower proximity
JSON search	GIN index on JSONB	Call detail records stored in JSON
Analytics performance	Materialized views + indexes	Recharge summary by region/date

Surendra Panpaliya

Founder and CEO

GKTCS Innovations

https://www.gktcs.com