PostgreSQL

Partitioning and Full-Text Search

Surendra Panpaliya

Agenda

- Range and list partitioning
- tsvector, tsquery for full-text search
- Functional indexes and GIN

Hands-On

- Partition an orders table by year
- Build product search using full-text index
- **Assignment:**
- Build a full-text index and query using @@ operator

Partitioning

DIVIDING A LARGE TABLE INTO SMALLER,

MORE MANAGEABLE PIECES (PARTITIONS),

WHICH TOGETHER BEHAVE LIKE A SINGLE TABLE.

Partitioning

The technique of splitting a large table

into smaller, more manageable

child tables (partitions)

based on certain criteria

(date, region, ID range).

Why Partitioning Matters?

- Improves **query performance** on large datasets
- Speeds up **bulk inserts, deletes, and archiving**
- Enables **parallel query execution**
- Reduces **index size per partition**
- Minimizes table bloat and vacuum overhead

Types of Partitioning

Type	Description
Range	Partitions by a range (e.g., dates, IDs)
List	Partitions by list of values (e.g., regions)
Hash	Partitions by hash values (used for even spreading)

Telecom Use Cases for Partitioning

Use Case	Partitioning Type
Daily recharge logs	Range (by date)
Usage by telecom circle (Delhi, Mumbai)	List (by circle)
Customer ID sharding	Hash

RANGE PARTITIONING

Partitioning rows

based on a range of values

in a column

by date, amount, or ID ranges.

Why Range Partitioning Matters?

Efficient querying when filtering by ranges (e.g., date range)

Reduces table scan time

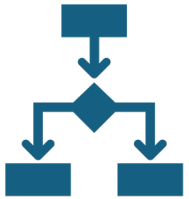
Speeds up **bulk deletes**, **archival**, and **index maintenance**

Telecom Use Cases for Range Partitioning

Use Case	Range Column
Recharge records per month	recharge_date
Daily call data (CDRs)	call_date
Usage data partitioned by billing date	usage_date

LIST PARTITIONING

Partitioning rows based on



a list of values,

such as city, region,

or network type.

Why List Partitioning Matters?

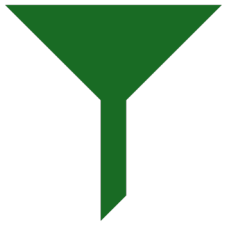
Efficient when your data

naturally belongs to a

small set of distinct categories

Why List Partitioning Matters?

Speeds up



filtering and reporting by category

Reduces index size per partition

Telecom Use Cases for List Partitioning

Use Case	List Column
Call logs by telecom circle	circle
SMS feedback grouped by region	region
Complaints by plan type	plan_type

Range vs List Partitioning

Feature	Range Partitioning	List Partitioning
Based On	Continuous values (e.g., dates)	Discrete values (e.g., regions)
Telecom Use Case	Monthly recharges, billing	Calls/SMS by circle
Example Column	recharge_date, usage_date	circle, region, plan_type
PostgreSQL Syntax	PARTITION BY RANGE (column)	PARTITION BY LIST (column)

What is Full-Text Search?

Allows you to perform complex text queries

(searching, ranking, stemming, etc.)

inside PostgreSQL using **natural language** terms.

What is Full-Text Search?

Unlike LIKE '%term%',

it uses **inverted indexes**,

lexical analysis, and

ranking algorithms.

Why FTS Matters?

Enables

searchable support tickets,

call logs, SMS content, and

feedback text

Why FTS Matters?

Highly efficient and fast

when used with

GIN indexes

Telecom Use Cases for Full-Text Search

Use Case	Benefit
Search SMS logs or complaints	Customer care analytics
Match call descriptions	Fraud detection or keyword tracking
Filter JSON logs (CDRs)	Query specific event tags or call reasons
Analyze survey feedback	Sentiment or keyword-based filtering

FTS Operators

Operator	Description	Example
@@	Text matches query	tsvector @@ tsquery
to_tsquery()	Converts query string	'network & poor'
plainto_tsquery()	Tokenizes plain text	'slow internet'
ts_rank()	Ranks results by relevance	ORDER BY ts_rank(...)

tsvector: Text Search Vector

A normalized and tokenized

form of text used for indexing.

Stores words (lexemes)

with positional info.

tsvector: Text Search Vector

USED FOR FULL-TEXT SEARCH (FTS)

A WAY TO SEARCH NATURAL LANGUAGE

DOCUMENTS EFFICIENTLY.

tsquery: Text Search Query

A query structure for

matching words against a tsvector.

Supports Boolean logic (AND, OR, NOT),

prefix matching, and phrases.

Purpose of tsvector and tsquery

Component	Description
tsvector	A normalized representation of a document — it stores lexemes (root words) for fast searching.
tsquery	A parsed search string/query — used to match against tsvector values.

Why Use tsvector and tsquery?

- Fast full-text searching
- Lexeme-based matching
- Tokenization, stop word removal, and stemming
- Advanced search

Fast full-text searching

- in large text fields
- like articles
- product descriptions
- logs

Lexeme-based matching

“running”,

“ran”, “runs”

→ all reduced to “run”.

Why Use tsvector and tsquery?

Tokenization,

stop word removal,

and stemming

happens automatically.

Why Use tsvector and tsquery?

Advanced search:

AND, OR, ! (NOT),

proximity.

Summary

tsvector:

Preprocessed searchable form of text.

tsquery:

Structured search query for text.

Summary

Use Cases: Search in blogs,

e-commerce product descriptions,logs, emails, etc.

Index: Use GIN index on tsvector column

to speed up search.

What is a GIN Index?

Generalized Inverted Index

Inverted Index = Maps keys to documents (rows)

What is a GIN Index?

- Used for indexing
- composite data types such as:
- tsvector (for Full-Text Search)
- Arrays, JSONB

What is a GIN Index?

GIN stores a separate entry

for **each component** in a composite field

(e.g., each word in a sentence).

What is a GIN Index?

Commonly used with:

tsvector (full-text search)

array columns

jsonb fields

hstore

Why GIN Index?

Useful for queries like:

```
WHERE column @@ to_tsquery('english', 'network & issue')
```

Why GIN Index?

Efficient for **multi-key containment** queries:

WHERE tags @> ARRAY['4G', 'coverage’]

WHERE metadata ? 'signal_strength'

Telecom Use Cases for GIN Indexes

Use Case	Operator / Column Type
Complaint search (full-text)	tsvector + @@ operator
JSON-based CDR or usage metadata	jsonb + @> operator
Find customers with specific data packs	ARRAY column

Summary

Feature	GIN Index
Use Case	Full-text, arrays, JSONB, multikey columns
Search Operators	@@, @>, ?, ?&, `?
Access Method	Inverted index: key → matching rows
Performance	Excellent for containment-style queries
Limitation	Slower to update than B-tree (write-heavy use cases)

Why it matters?

- Traditional search with LIKE
- '%text%' is slow and inaccurate.
- Full-text search with tsvector + tsquery:
- Is fast, especially with GIN indexes.

Why it matters?

Supports natural language search.

Is built-in

no external tools like

Elasticsearch needed.

Telecom Domain Use Cases

Use Case	Description
Complaint search	Customers complaining about “call drops”
SMS content analysis	Search messages containing “network issue”
Agent ticket resolution	Search across ticket summaries and actions
Fraud detection	Look for messages or calls with suspicious terms
Chatbot or IVR logs	Search chat/call transcripts for QA

tsvector vs tsquery

Concept	Description	Example
tsvector	Document representation for indexing	to_tsvector('english', text)
tsquery	Query structure for searching	to_tsquery('recharge & failed')
plainto_tsquery	Converts plain text into tsquery	plainto_tsquery('slow internet')
@@	Match operator	tsvector @@ tsquery
ts_rank()	Computes relevance of a match	ts_rank(tsvector, tsquery)
GIN Index	Fastest index for FTS in PostgreSQL	CREATE INDEX … USING GIN (tsvector)

FUNCTIONAL INDEXES

Built on the result of

an expression or function

applied to a column,

rather than on the raw column value itself.

Why Functional Indexes Matter?

BOOSTS PERFORMANCE FOR QUERIES

WITH EXPRESSIONS

LOWER(EMAIL), DATE(TIMESTAMP)

Why Functional Indexes Matter?

Avoids full scans

when using computed WHERE conditions

Useful for non-trivial WHERE clauses

Telecom Use Cases for Functional Indexes

Use Case	Function Used
Search by lowercase complaint email/text	LOWER(complaint_text)
Billing grouped by day (not timestamp)	DATE(billing_time)
Region code extraction from phone number	SUBSTRING(phone, 1, 4)

Functional vs GIN Index

Feature	Functional Index	GIN Index
Based On	Expression output (e.g. LOWER())	Composite types (text, jsonb, array)
Telecom Use Case	Normalize email, extract date	Search complaints, filter usage logs
PostgreSQL Syntax	CREATE INDEX ON table (FUNC(col))	USING GIN (col)
Use for FTS?	❌ No	✅ Yes
Use for JSONB?	❌ No	✅ Yes

Surendra Panpaliya

Founder and CEO

GKTCS Innovations

https://www.gktcs.com