DATA SATURDAYS #64 PARMA 2024

PostgreSQL: il Query Optimizer in dettaglio

Danilo Dominici

Sponsor & Org

iSolutions

engage LABS

DATA SKILLS
UNDERSTANDING THE WORLD

LIME ware

daniela mavisi COMMUNICATION

Lucient ITALIA

Bi Factory
DATA KNOWLEDGE ADVISOR

Chi è Danilo Dominici

- Consulente Senior @ Altitudo sulle architetture database
  - SQL Server
  - PostgreSQL
  - Redis
- Creatore di SQL Start! (sqlstart.it) – evento community (Ancona, giugno 2025)
- Community speaker, Certified Trainer, ex-MVP

Microsoft CERTIFIED TRAINER MCT

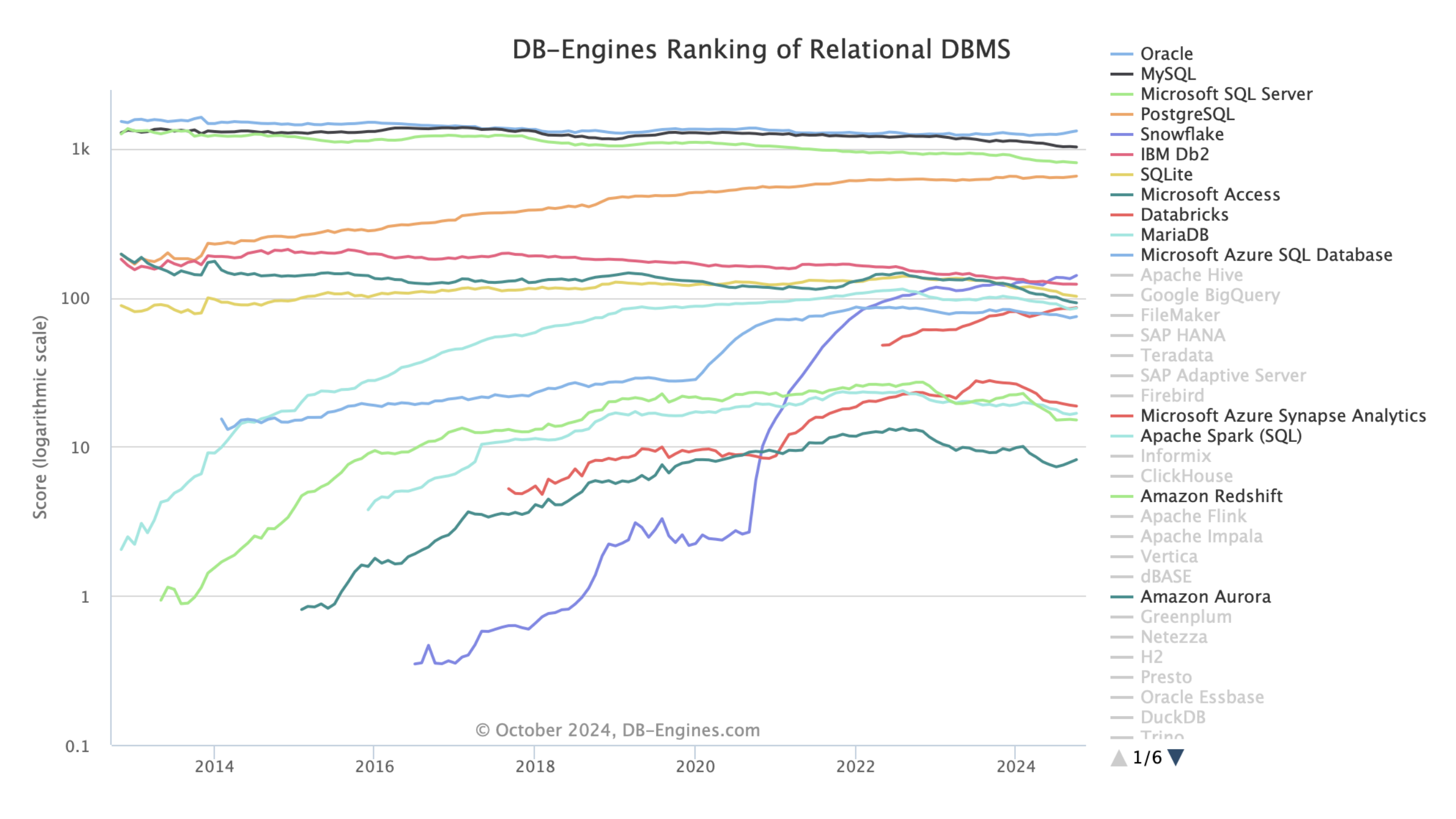
2000+

MVP Microsoft® Most Valuable Professional

2014-2020

Obiettivi di questa sessione

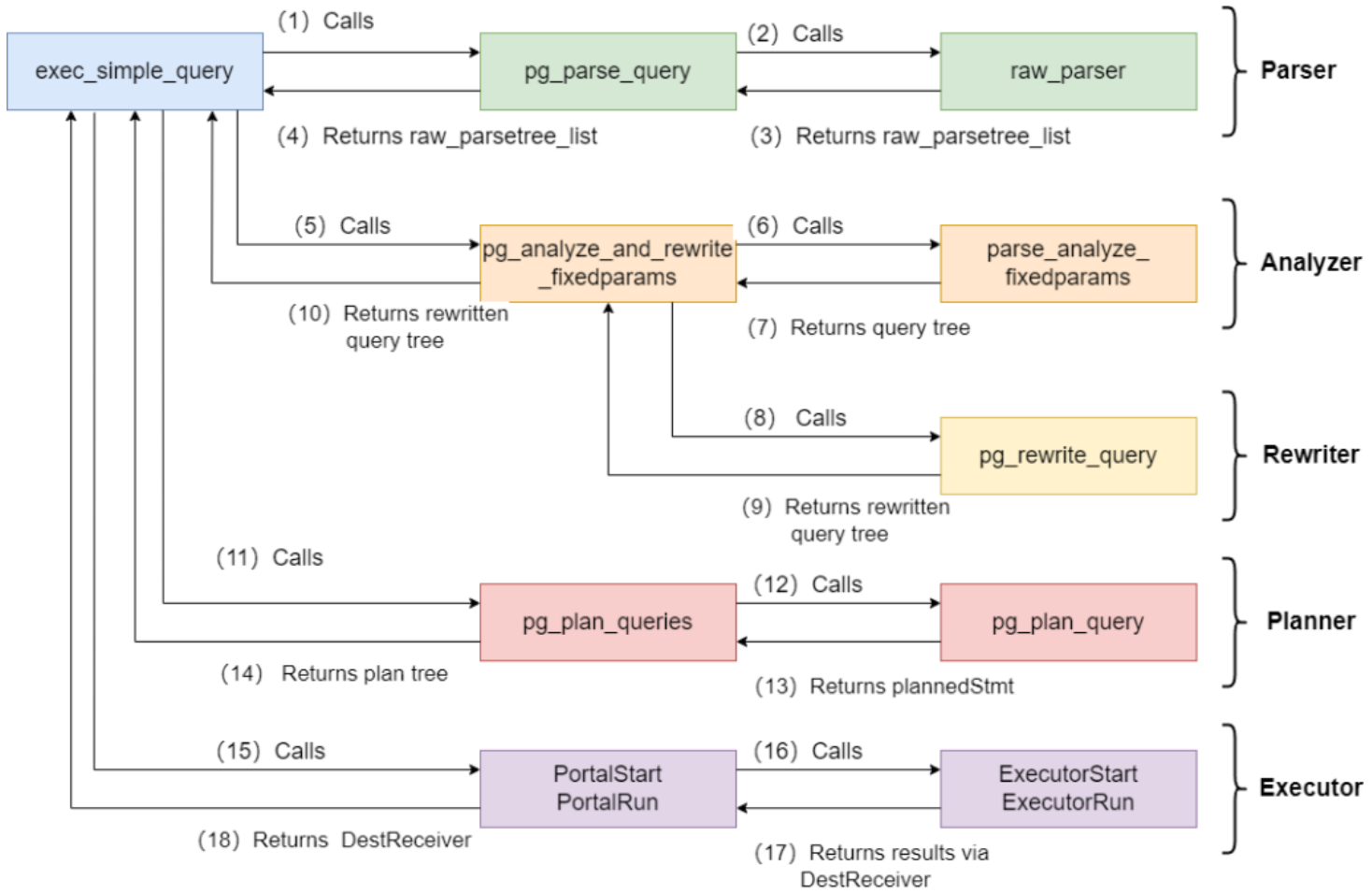
- Comprendere come PostgreSQL processa le query
- Comprendere quali tipologie di accesso ai dati usa Postgres
- Distinguere tra i diversi tipi di indici disponibili


DB-Engines Ranking of Relational DBMS
Score (logarithmic scale)
1k
100
10
1
0.1
2014
2016
2018
2020
2022
2024
© October 2024, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
Snowflake
IBM Db2
SQLite
Microsoft Access
Databricks
MariaDB
Microsoft Azure SQL Database
Apache Hive
Google BigQuery
FileMaker
SAP HANA
Teradata
SAP Adaptive Server
Firebird
Microsoft Azure Synapse Analytics
Apache Spark (SQL)
Informix
ClickHouse
Amazon Redshift
Apache Flink
Apache Impala
Vertica
dBASE
Amazon Aurora
Greenplum
Netezza
H2
Presto
Oracle Essbase
DuckDB
Trino
1/6

Il percorso di una query

Il percorso di una query

Il percorso di una query



Parser

- Responsabile del controllo della sintassi
- Usa i tools *flex* e *bison* per fare il parsing della query
- Genera il «parse tree»

```
SelectStmt
├─ targetList
│  └─ ResTarget
│     └─ ColumnRef
│        └─ A_Star
└─ fromClause
   └─ RangeVar
      └─ relname = mytable
```

SELECT * FROM mytable

```
DeleteStmt
├─ relation
│  └─ RangeVar
│     └─ relname = abc
└─ whereClause
   └─ A_Expr
      ├─ kind = AEXPR_OP
      └─ ...
```

DELETE FROM abc WHERE id = 1

The entry point is [parser/parser.c]

Analyzer

- Responsabile per l'analisi sintattica dettagliata
- Accede al database designato
- Controlla l'esistenza della tabella
- Controlla la correttezza dei formati dei dati
- Converte i nomi delle tabelle negli OID interni
- Genera il «query tree»

The entry point is in parser/analyze.c

Rewrite

- Responsabile della riscrittura della query, dove richiesto
- Se la query usa un oggetto VIEW o RULE si occupa di espandere (o riscrivere) la sua definizione

•The entry point is in rewrite/rewriteHandler.c

Planner

- Responsabile della generazione del piano di esecuzione
- Identifica tutti i possibili metodi (o paths) che possono essere utilizzati per ottenere il risultato richiesto
- Sceglie il metodo migliore per completare la query nel minor tempo possibile

•The entry point is in optimizer/plan/planner.c

Planner

- Deve prevedere:
  - Il costo iniziale
  - Il costo totale
  - Il numero di righe restituite
  - La lunghezza media delle righe

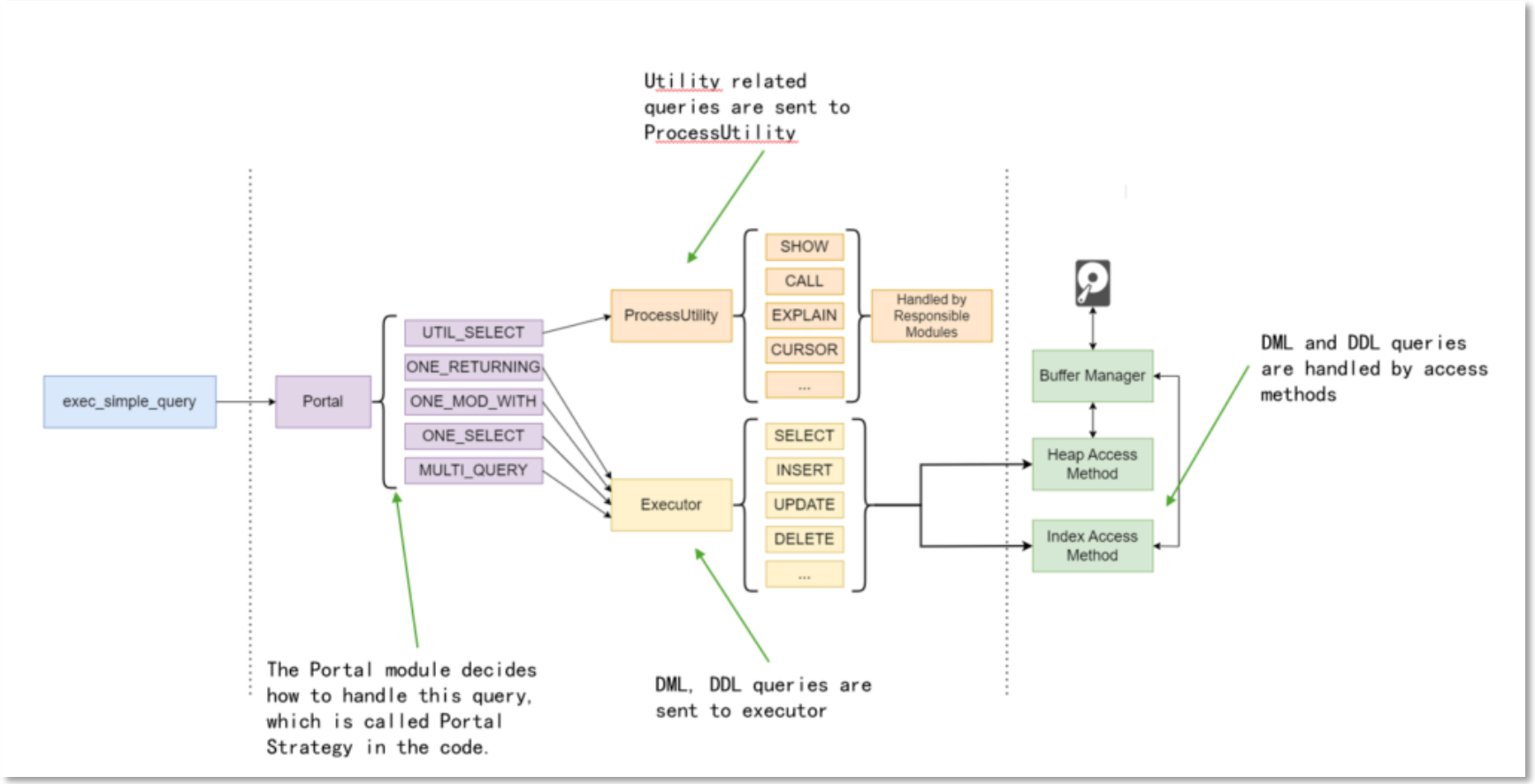
Seq Scan on bookings.aircrafts_data ml (**cost=0.00..3.36 rows=9 width=52**)

Executor

- Esegue la query in base al piano di esecuzione preparato
- Accede a diversi moduli di backend di PostgreSQL per completare la query

•The entry point is in executor/execMain.c

Executor


Utility related queries are sent to ProcessUtility
exec_simple_query
Portal
UTIL_SELECT
ONE_RETURNING
ONE_MOD_WITH
ONE_SELECT
MULTI_QUERY
ProcessUtility
Executor
SHOW
CALL
EXPLAIN
CURSOR
...
Handled by Responsible Modules
SELECT
INSERT
UPDATE
DELETE
...
Buffer Manager
Heap Access Method
Index Access Method
DML and DDL queries are handled by access methods
The Portal module decides how to handle this query, which is called Portal Strategy in the code.
DML, DDL queries are sent to executor

Quali decisioni deve prendere il QO?

- Decidere quale metodo di scansione usare
  - Sequential Scan
  - Bitmap Index Scan
  - Index Scan
- Decidere quale metodo di join usare
  - Nested loop
  - Hash join
  - Merge join
- Decidere l'ordine con cui applicare le join

Modalità di accesso ai dati

Heaps

- L'heap memorizza tutte le tuple in uno o più file.
- Ogni tupla ha un indirizzo, che chiamiamo TID.
- Il TID è la posizione fisica della tupla (numero di blocco, offset).
- Inizialmente, tutte le tuple vengono memorizzate nell'ordine in cui sono state inserite.
- Aggiornamenti/cancellazioni e inserimenti concorrenti possono riordinare le tuple in modo non deterministico.
- L'accesso ai dati è sequenziale.
    - È necessario scansionare l'intera tabella per trovare la/le riga/righe richieste.

Sequential scans

- Le letture sequenziali su disco sono molto più veloci rispetto alle letture casuali.
- Funziona bene con tabelle piccole o quando è necessario leggere la maggior parte della tabella.

Index scan

- La Scansione dell'Indice (Index Scan) attraversa l'albero (B-tree), attraversa i nodi foglia per trovare tutte le voci corrispondenti e recupera i dati corrispondenti dalla tabella.
- È simile a una scansione a intervalli dell'indice (Index Range Scan) seguita da un'operazione di accesso alla tabella tramite RowId dell'indice (Table access by Index RowId).
- I predicati di filtro dell'indice spesso causano problemi di prestazioni durante una Scansione dell'Indice (Index Scan).

Index Only Scan (covering indexes)

- La scansione dell'indice (Index Only Scan) esegue l'attraversamento dell'albero (B-tree) e attraversa i nodi foglia per trovare tutte le voci corrispondenti.
- Non è necessario accedere alla tabella perché l'indice contiene tutte le colonne necessarie per soddisfare la query (eccezione: informazioni sulla visibilità MVCC).

Bitmap Index Scan / Bitmap Heap Scan / Recheck Cond

- Una normale scansione dell'Indice (Index Scan) recupera un puntatore a tupla alla volta dall'indice e visita immediatamente quella tupla nella tabella.
- Una scansione con bitmap (bitmap scan), invece, recupera tutti i puntatori alle tuple dall'indice in una sola operazione, li ordina utilizzando una struttura dati "bitmap" in memoria e poi accede alle tuple della tabella nell'ordine fisico della loro posizione.
- Per evitare accessi ripetuti alla stessa pagina di dati, il motore utilizza una statistica speciale che mostra la correlazione tra l'ordine fisico e logico, per evitare di rileggere la stessa pagina quando i dati sono fisicamente ordinati nello stesso modo delle registrazioni dell'indice.

Modifica dei parametri di costo

- Il Planner utilizza delle costanti per il calcolo del costo, configurabili tramite il file di configurazione
- In casi particolari, si può intervenire cambiando temporaneamente il valore di questi parametri
- Ovviamente NON va fatto senza avere una profonda conoscenza del proprio sistema…

```
#seq_page_cost = 1.0
# cost of a sequentially-fetched disk page
#random_page_cost = 4.0
# cost of a non-sequentially-fetched disk page
#cpu_tuple_cost = 0.01
# cost of processing each row during a query
#cpu_index_tuple_cost = 0.005
# cost of processing each index entry
#cpu_operator_cost = 0.0025
# cost of processing each operator or function
```

https://www.postgresql.org/docs/current/runtime-config-query.html

Join operations

- Nested Loops
  - Unisce due tabelle recuperando il risultato da una tabella e interrogando l'altra tabella per ogni riga della prima.
- Hash Join /Hash
  - La Hash Join carica i record candidati da un lato del join in una tabella hash (indicata come Hash nel piano), che viene poi sondato per ogni record dall'altro lato del join
- Merge join
  - The (sort) merge join combina due liste ordinate (come una cerniera ☺).
    Entrambi i lati del join devono essere preordinati.

Il comando EXPLAIN

Il comando EXPLAIN

SELECT * FROM aircrafts

aircraft_code	model	range
773	Boeing 777-300	11.100
763	Boeing 767-300	7.900
SU9	Sukhoi Superjet-100	3.000
320	Airbus A320-200	5.700
321	Airbus A321-200	5.600
319	Airbus A319-100	6.700
733	Boeing 737-300	4.200
CN1	Cessna 208 Caravan	1.200
CR2	Bombardier CRJ-200	2.700

Il comando EXPLAIN

EXPLAIN (ANALYZE,VERBOSE,**COSTS**,BUFFERS,**TIMING**)

SELECT * FROM aircrafts

```
Seq Scan on bookings.aircrafts_data ml  (cost=0.00..3.36
rows=9 width=52) (actual time=0.029..0.045 rows=9
loops=1)
  Output: ml.aircraft_code, (ml.model ->> lang()), ml.range
  Buffers: shared hit=1
Query Identifier: 762113756556413256 7
Planning Time: 0.056 ms
Execution Time: 0.060 ms
```

Le opzioni disponibili in EXPLAIN

Option	Description	Default	Value
ANALYZE	Executes the SQL statement and discards the output information	FALSE	Boolean
VERBOSE	Allows you to show additional information regarding the plan	TRUE	Boolean
COSTS	Includes the estimated startup and total costs of each plan node, as well as the estimated number of rows and the estimated width of each row in the query plan.	TRUE	Boolean
BUFFERS	Adds information to the buffer usage. BUFFERS only can be used when ANALYZE is enabled.	FALSE	Boolean
TIMING	This parameter includes the actual startup time and time spent in each node in the output.	TRUE	Boolean
FORMAT	Specify the output format of the query plan	TEXT	TEXT | XML | JSON | YAML

DEMO

Indici

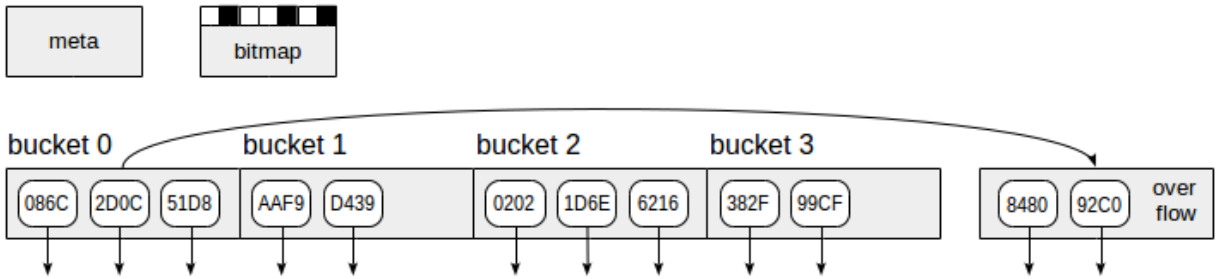
Tipologie di indici disponibili in PostgreSQL

- Hash indexes
- B-trees
- GiST
- SP-GiST
- GIN
- RUM
- BRIN

https://habr.com/ru/company/postgrespro/blog/441962/

Hash indexes

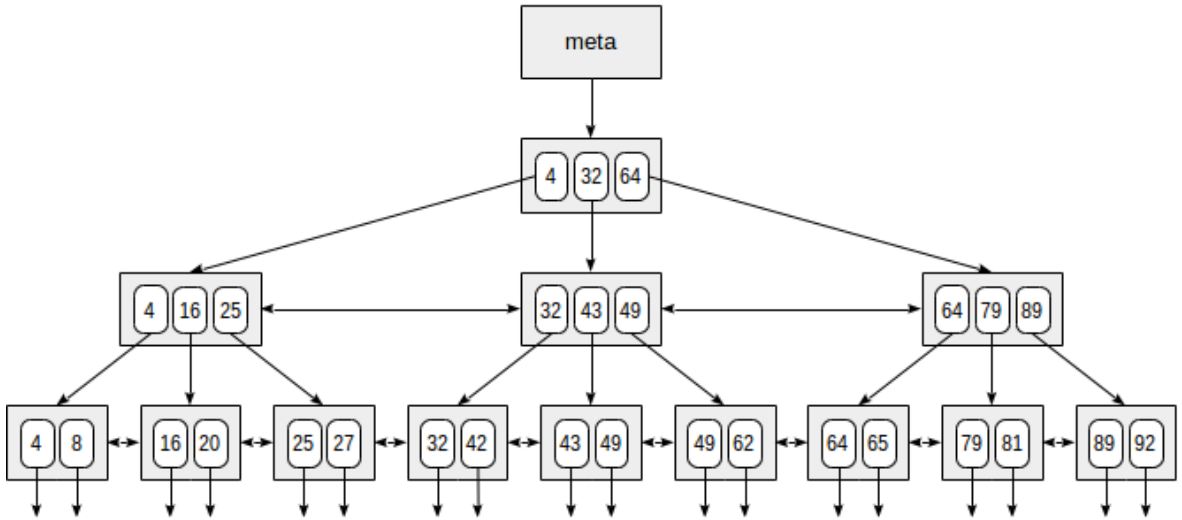
- Hash code and TID pairs are saved into buckets



https://habr.com/en/company/postgrespro/blog/442776/

B-trees

- Balanced: each leaf page is separated from the root by the same number of internal pages
- Multi-branched: each page contains a lot of TIDs
- Sorted in nondecreasing order and connected by bidirectional

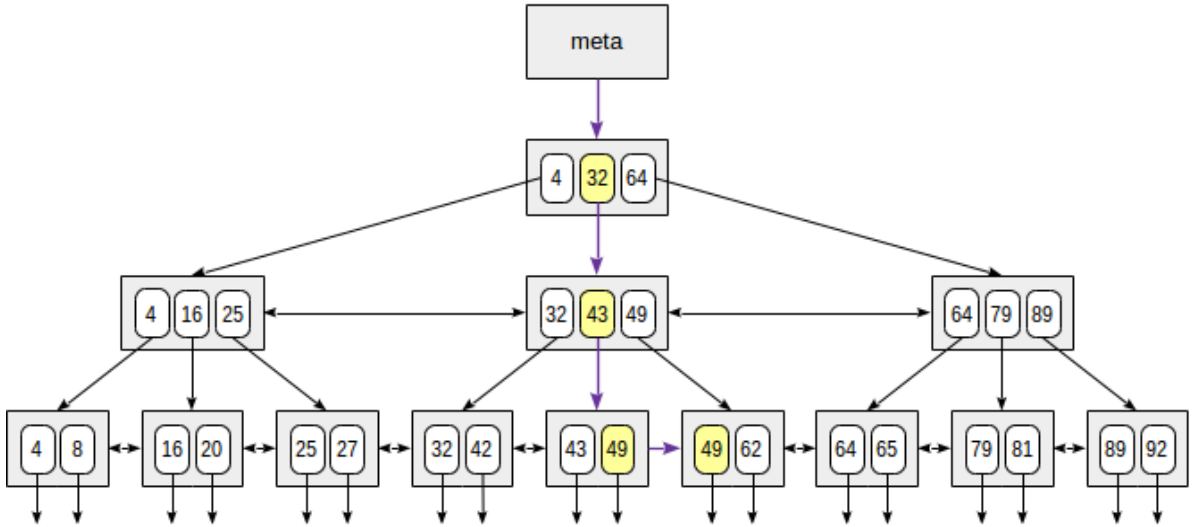


https://habr.com/ru/company/postgrespro/blog/443284/

B-trees

- Searching by equality

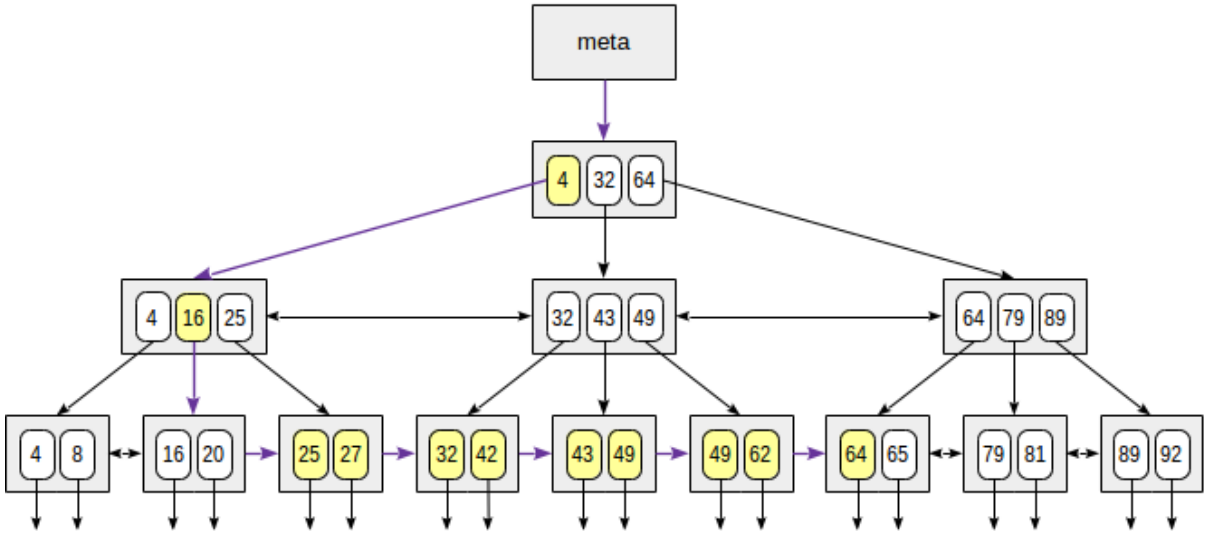
  SELECT id FROM table WITH id = 49



B-trees

- Searching by range

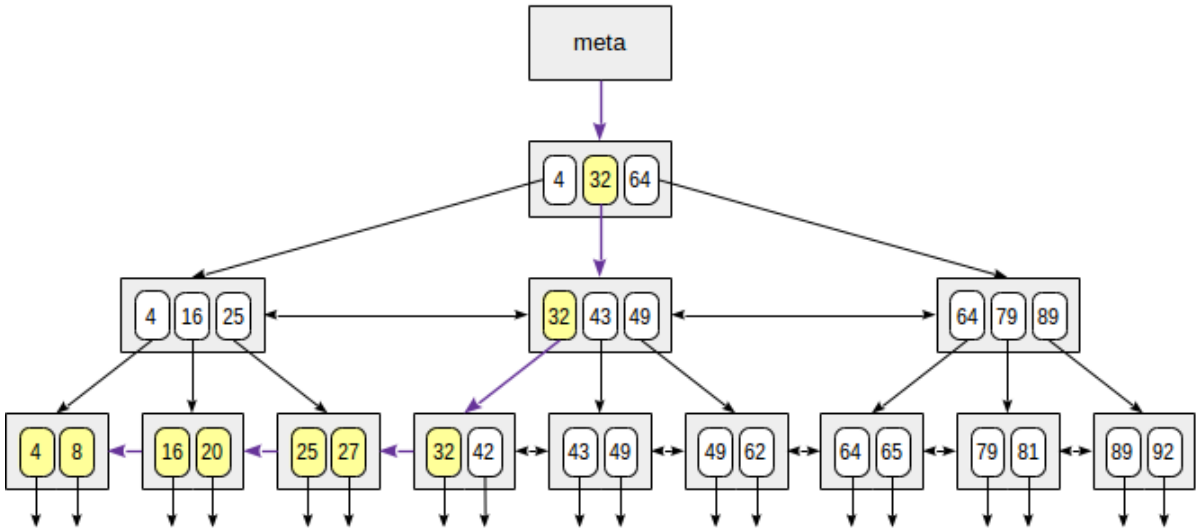
  SELECT id FROM table WITH id >= 23 AND id <= 64



B-trees

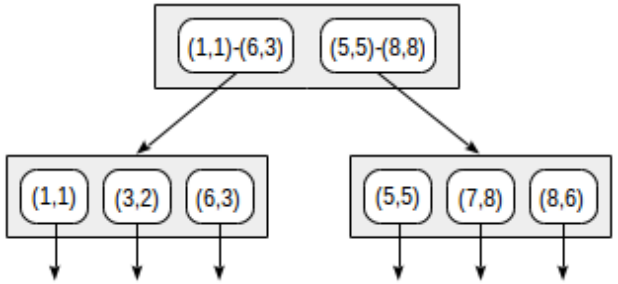
- Searching by inequality

  SELECT id FROM table WITH id <= 35



Generalized Search Tree (GiST)

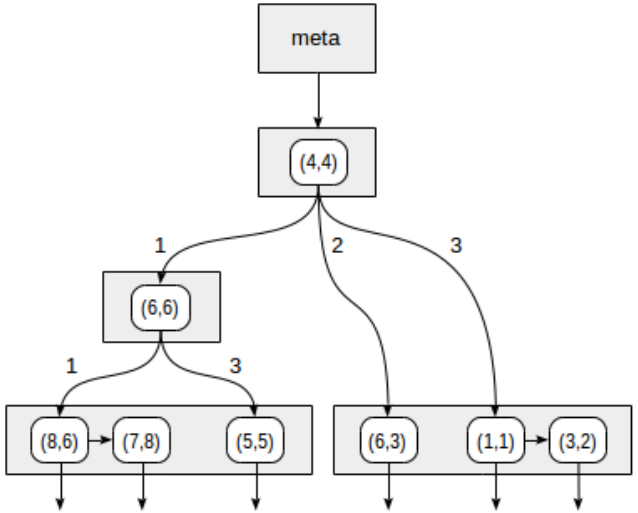
- The same idea of B-Tree, but without the strict rules connected with the comparison semantics (greater, less, equal)
- Can accomodate different data types like geodata, text documents, images

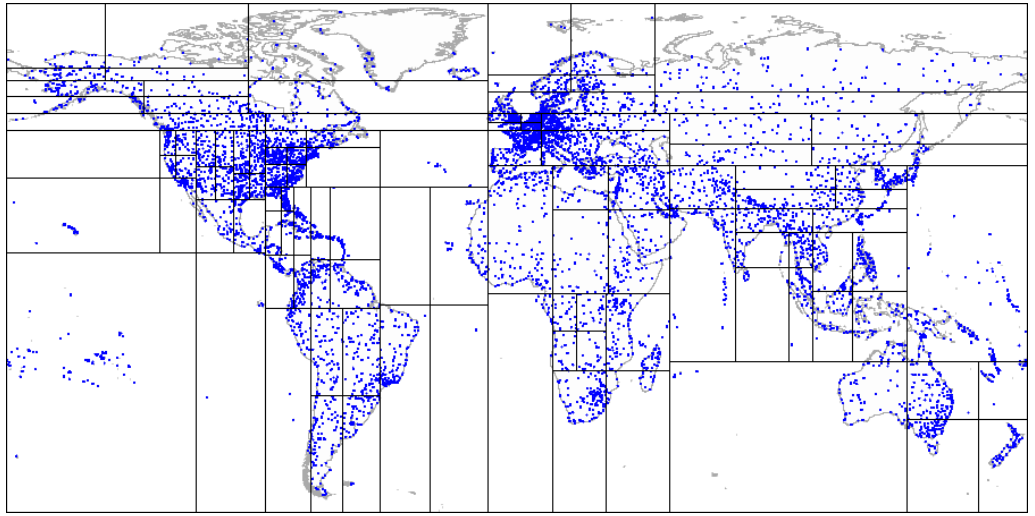


https://habr.com/ru/company/postgrespro/blog/444742/

SP-GiST

- Generalized Search Tree with Space partitioning
- Suitable for structures where the space can be recursively split into *non-intersecting* areas

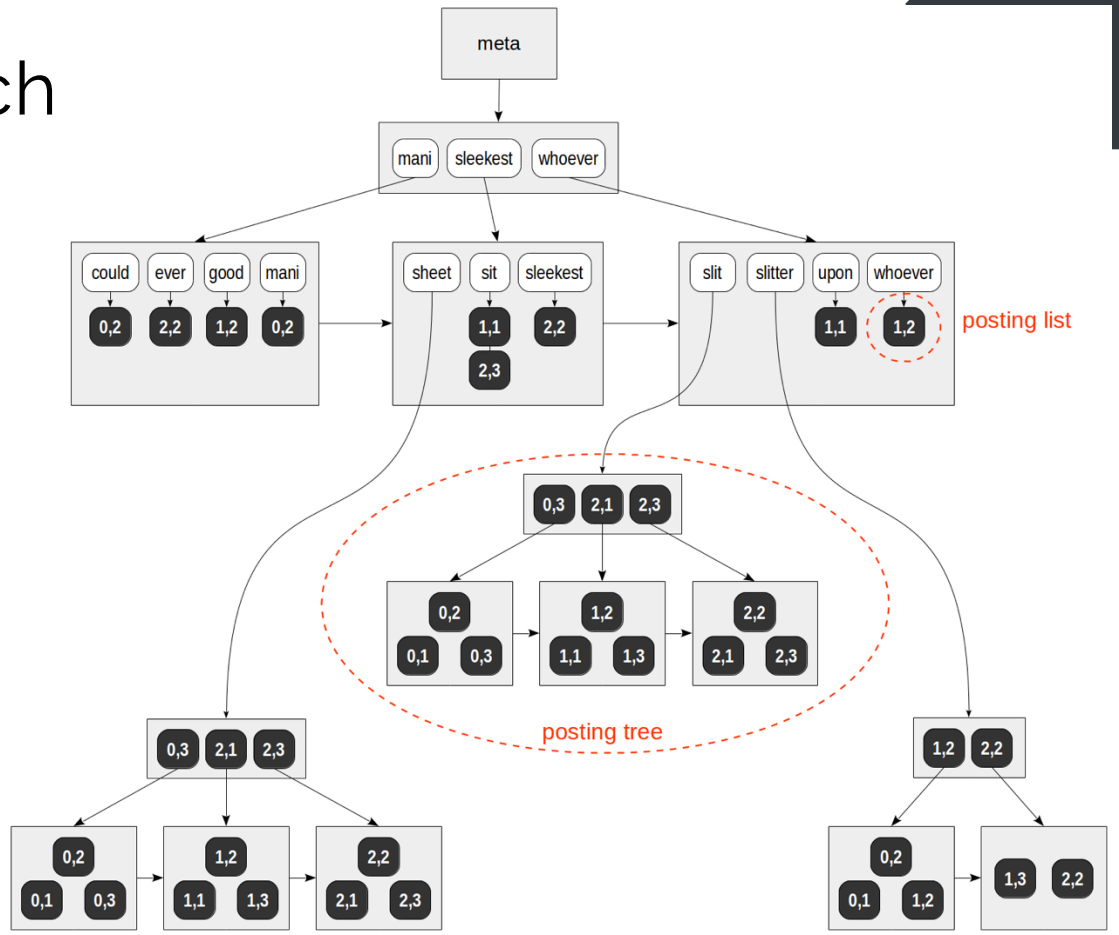




https://habr.com/ru/company/postgrespro/blog/446624/

Generalized Inverted Index (GIN)

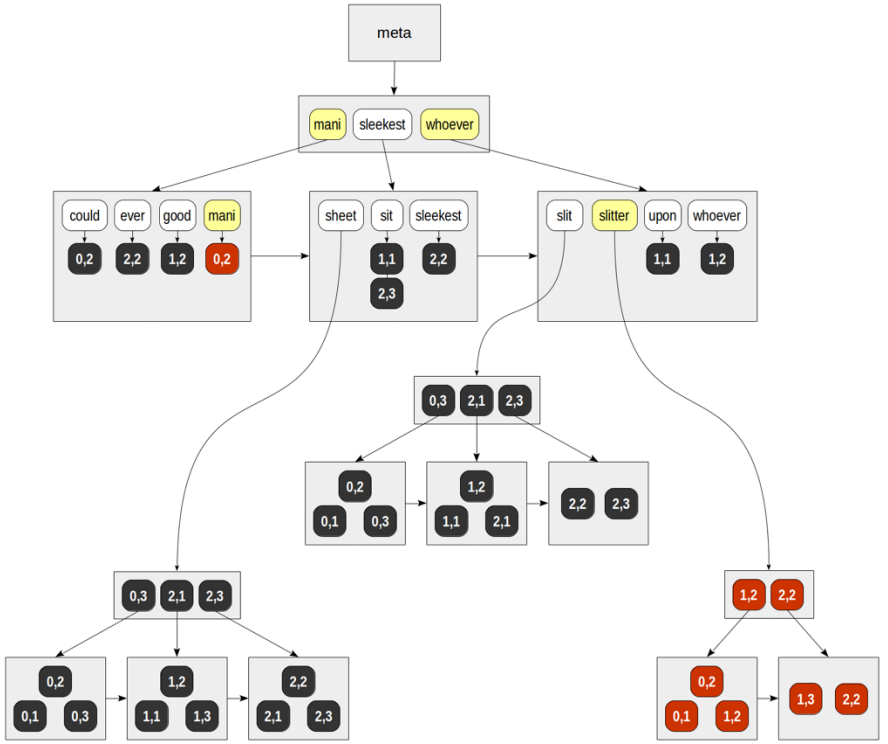
- Main application: full-text search



https://habr.com/ru/company/postgrespro/blog/448746/

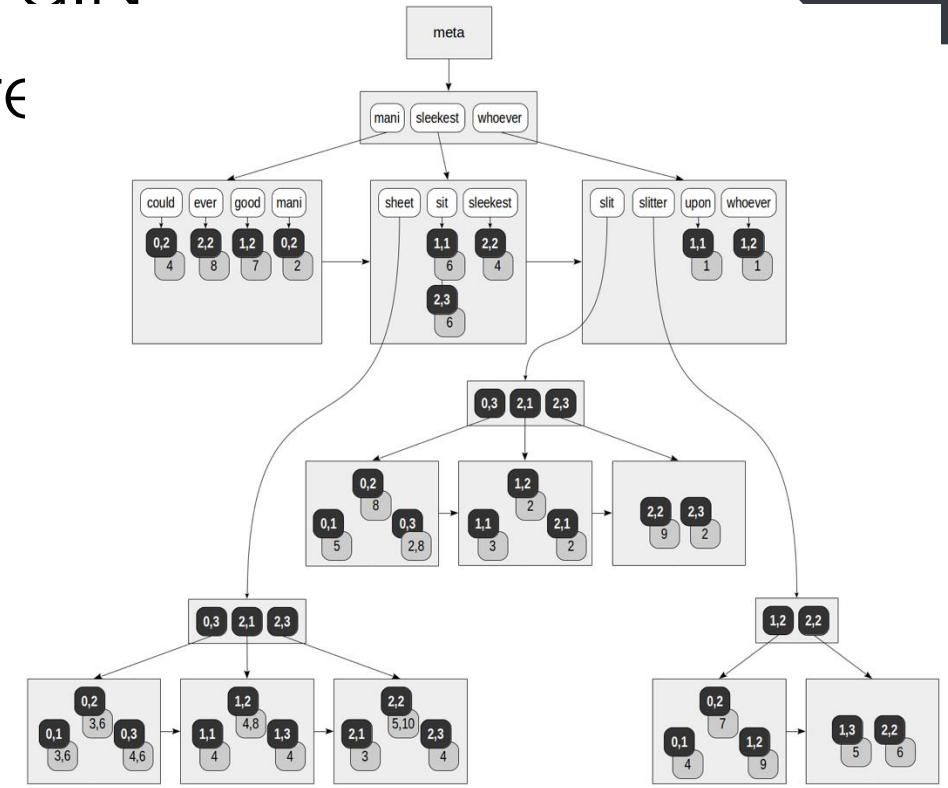
Generalized Inverted Index (GIN)

- Example:
  select doc from ts where doc_tsv @@ to_tsquery('many & slitter');



RUM

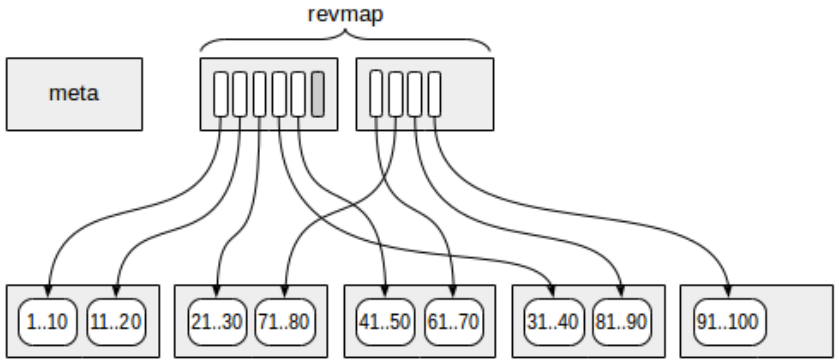
- Expands the concept that underlines GIN
- It is not included in a standard Postgre
- Better support for phrases search



https://habr.com/ru/company/postgrespro/blog/452116/

Block Range Index (BRIN)

- The idea of BRIN is to avoid looking through definitely unsuited rows rather than quickly find the matching ones
- A block range is a group of pages adjacent to each other, where summary information about all those pages is stored in Index.
- Keys on generated sequence numbers or created data are best candidates for E
  index.



https://habr.com/en/company/postgrespro/blog/452900/

Index analysis

- What to analyze?
  - Bloated indexes
  - Bloated tables
  - Duplicated indexes
  - Intersected indexes
  - Indexes with null values
  - Invalid indexes
  - Missing indexes
- Foreign keys without index
- Tables without primary keys
- Unused indexes

Ricapitolando…

- Il query optimizer di PostgreSQL funziona molto bene
  - A patto di effettuare la corretta manutenzione (VACUUM/ANALYZE)
  - Creare gli indici corretti a supporto delle query
- Monitorare postgresql per identificare le query problematiche
  - Slow query log
  - Sistemi di monitoring esterni (terze parti o open source, ce ne sono molti)
- In casi particolari, è possibile modificare in postgresql.conf i parametri di costo assegnati ai vari metodi di accesso

Grazie!!!