Performance e Escalabilidade em bancos SQL

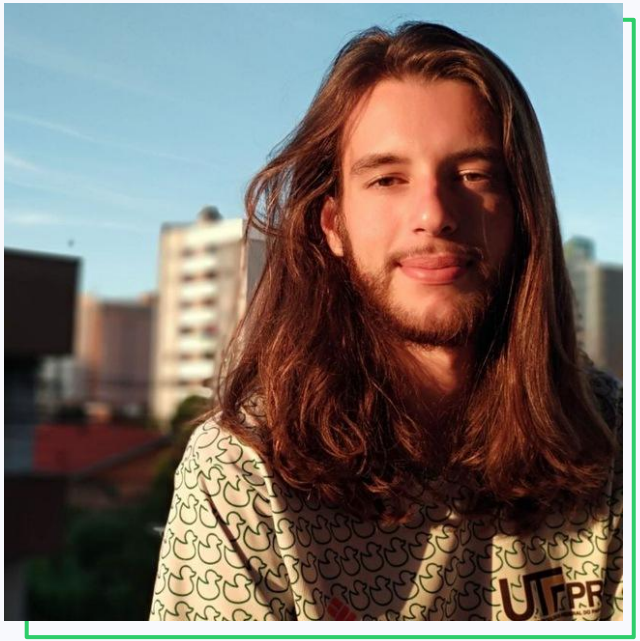
Gabriel Prando

Engenheiro de Software na Conta Simples

Gabriel Prando

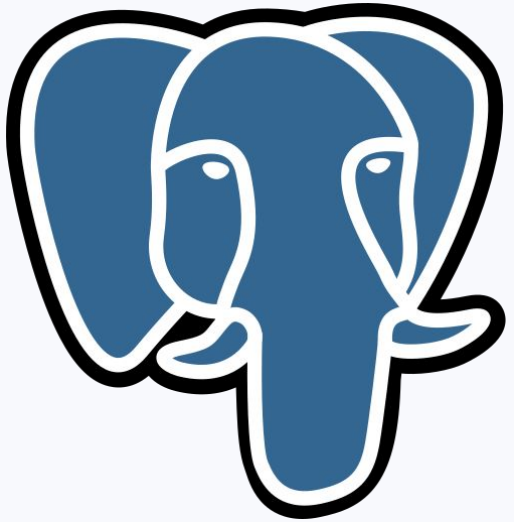
Engenheiro de software na **Conta Simples**

Backend no time de plataforma, engenheiro de computação formado pela UTFPR-PR e entusiasta em IoT.



O que veremos

- Critérios para escolha de um banco
- Funcionamento e arquitetura interna
- Formas de otimização
- Indexes
- Exemplos
- Dúvidas



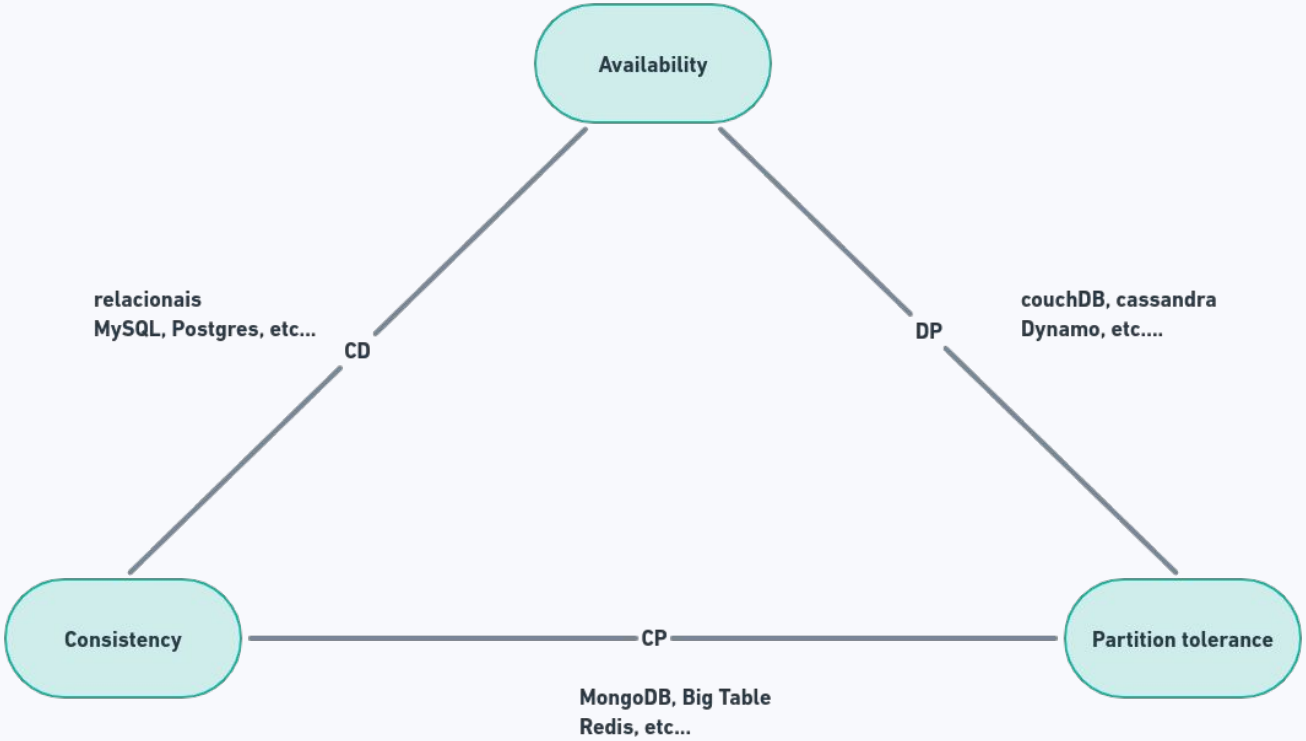
Ecossistema atual banco de dados

Ecossistema

- Dezenas de bancos de dados no mercado;
- Diversos casos de uso;
- Diversas classificações;
- Várias formas de manter.

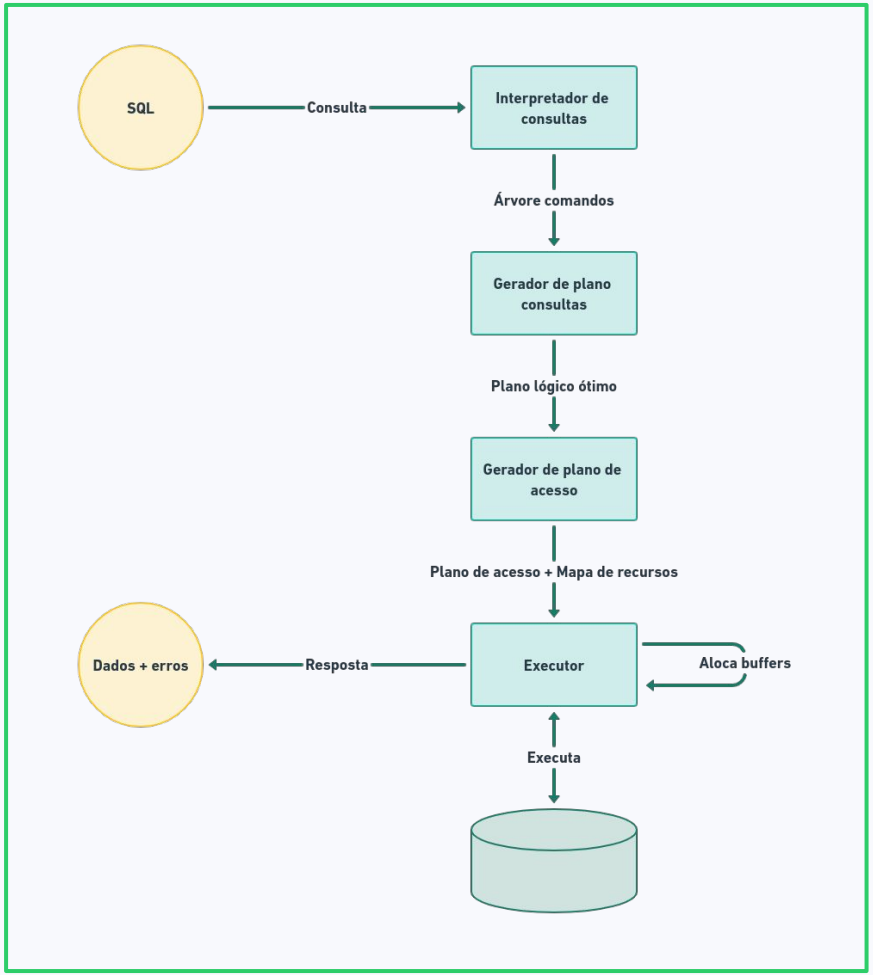
E como escolher?

Teorema CAP


Availability
relacionais
MySQL, Postgres, etc...
CD
DP
couchDB, cassandra
Dynamo, etc....
Consistency
CP
Partition tolerance
MongoDB, Big Table
Redis, etc...

Antes de falar de otimização, vamos entender um pouco como as coisas funcionam

Arquitetura interna



Agora que entendemos
um pouco

Quais as formas de otimizar um banco?

Formas

- Arquiteturas e padrões de projetos
- Particionamento / fragmentação (sharding)
- Escalabilidade vertical
- Escalabilidade horizontal
- Otimização de consultas

Arquiteturas e padrões de projetos para alta carga de dados

- CQRS
- Segregação de dados por micro serviços
- Cache em aplicações
- Projetar aplicações para interações assíncronas
- Estratégias de expurgo de dados

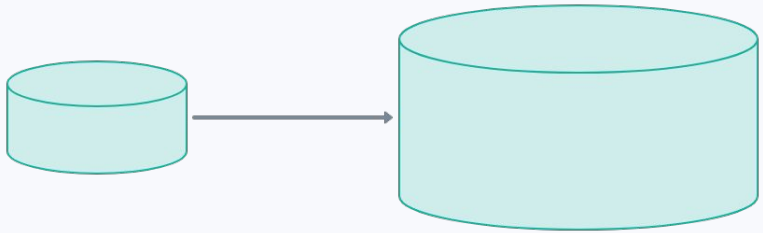
Particionamento vertical e horizontal

Vertical

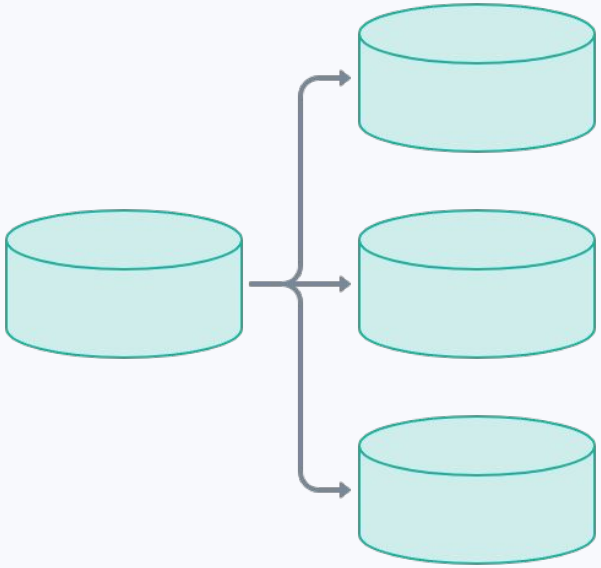
Quebrar / fragmentar colunas menos acessas das mais acessadas. Cada set do fragmento terá estrutura diferente

Horizontal

Fracionamos por conjunto e todas as frações têm a mesma estrutura

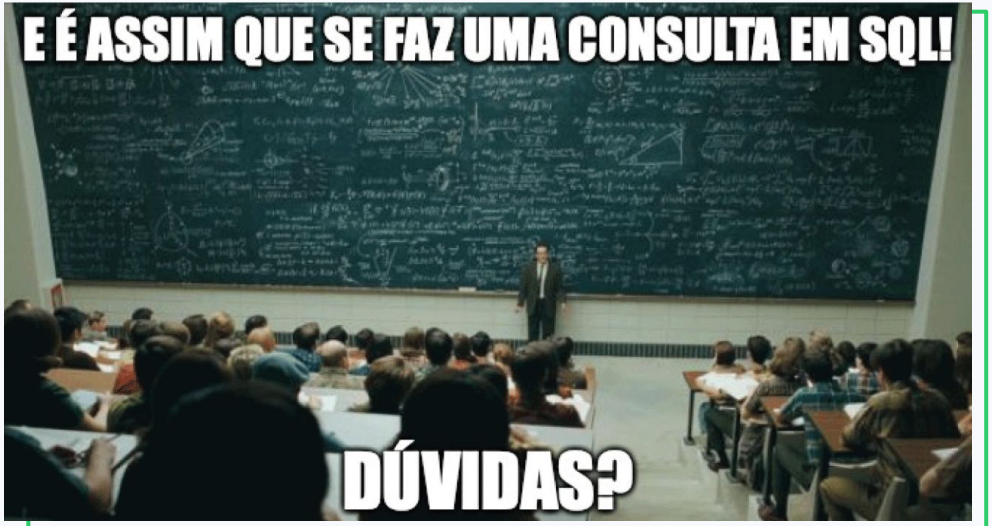


Escalar verticalmente



Escalar horizontalmente

Otimização de consultas



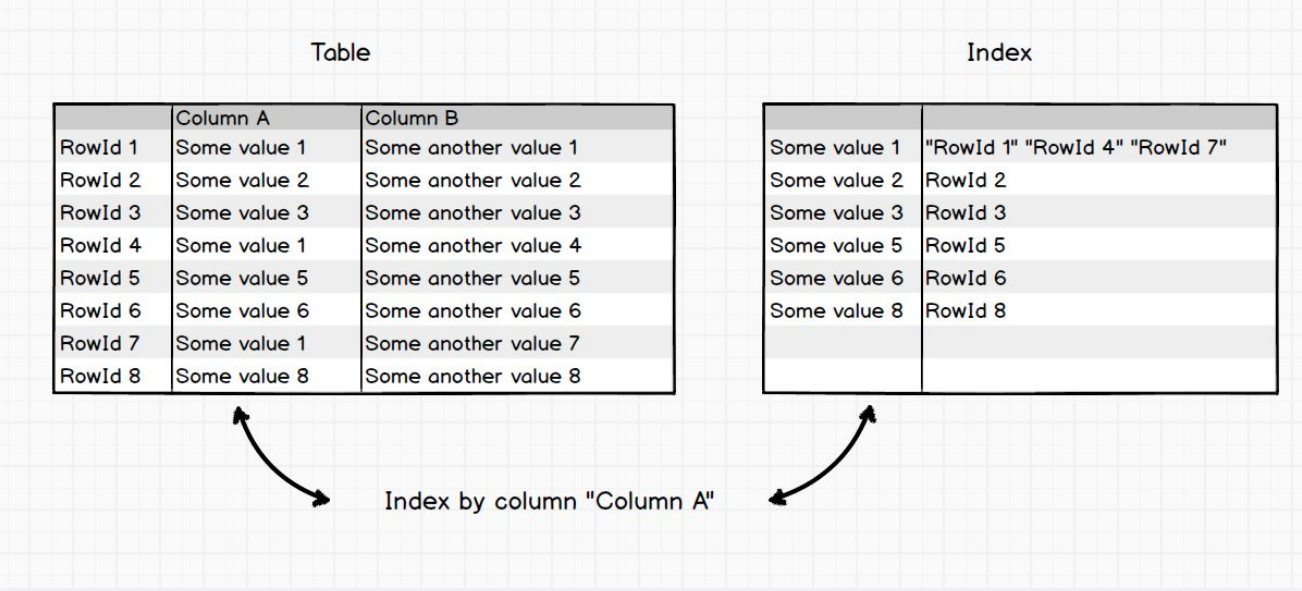
Não adiante ter arquitetura modelo se as queries no DB são um gargalo

Indexes

Assim como em livros, quando queremos achar uma seção específica, recorremos ao índice do livro. Vantagens:

- Performance;
- Pesquisa de dados;
- Otimização.

Relação inversa Index vs Tabela

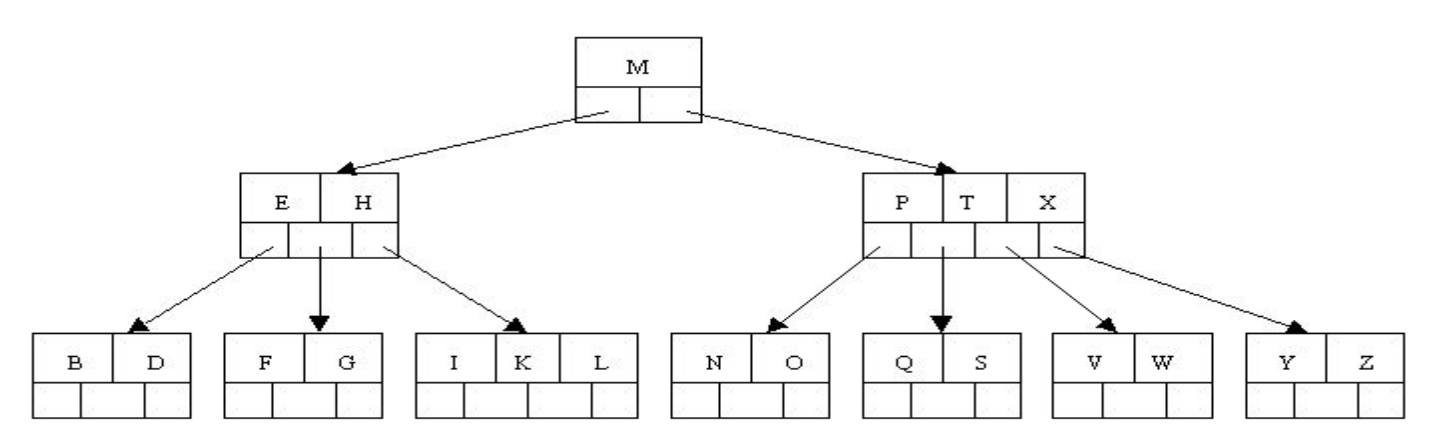


	Column A	Column B
RowId 1	Some value 1	Some another value 1
RowId 2	Some value 2	Some another value 2
RowId 3	Some value 3	Some another value 3
RowId 4	Some value 1	Some another value 4
RowId 5	Some value 5	Some another value 5
RowId 6	Some value 6	Some another value 6
RowId 7	Some value 1	Some another value 7
RowId 8	Some value 8	Some another value 8

Some value 1	"RowId 1" "RowId 4" "RowId 7"
Some value 2	RowId 2
Some value 3	RowId 3
Some value 5	RowId 5
Some value 6	RowId 6
Some value 8	RowId 8

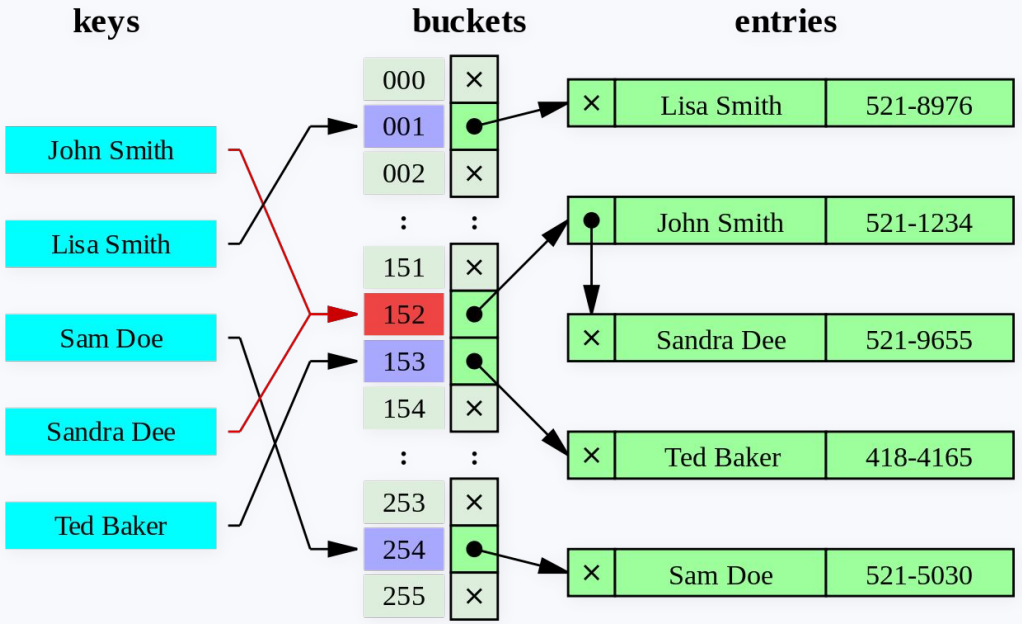
B-Tree

- Padrão default “create index”
- Funciona com base em árvores B (permanecem equilibradas em cada ramo da árvore)
- Utilizados para consultas de igualdade e intervalo de forma eficiente



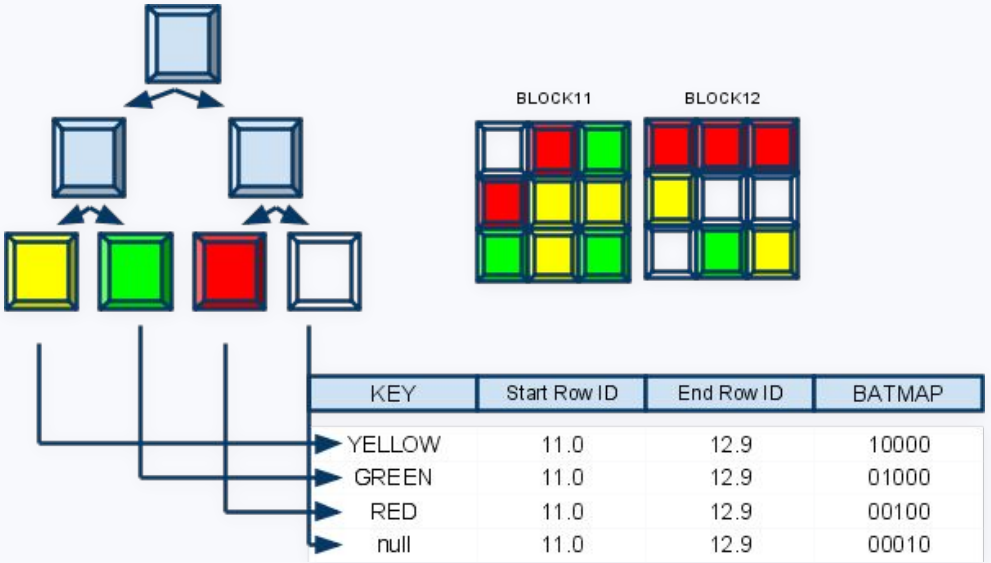
Hash

- Comparações de igualdade;
- Pesquisa muito rápida O(1);
- Índice não precisa ser reconstruído.
- Sensível a colisões se os dados forem mal distribuído



Índex Bitmap

- Cria um bitmap separado (uma sequência de 0 e 1) para cada valor possível da coluna;
- Representação compacta;
- Leitura rápida para "is".

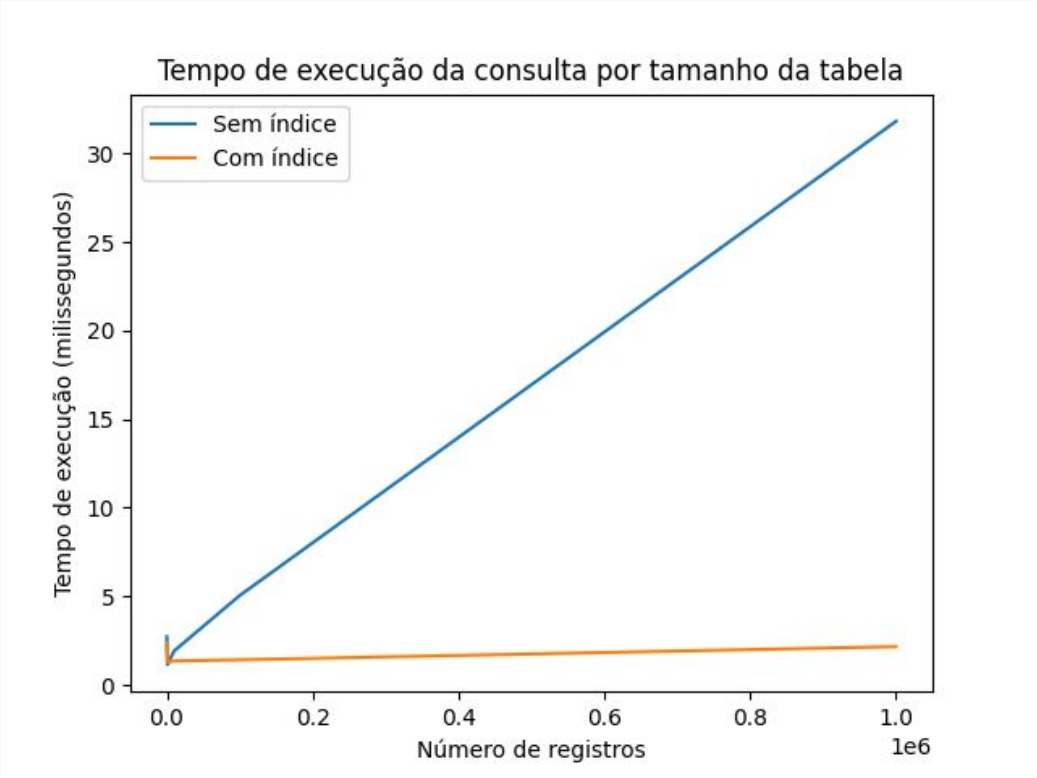


GIN e GIST

- **Índices invertido generalizado (GIN)**
  - Úteis quando um índice deve mapear muitos valores para uma linha
- **Índex de pesquisa generalizado (GIST)**
  - Permitem que construa estruturas de árvore balanceadas gerais e podem ser usados para operações além de comparações de igualdade e intervalo.
  - Usados para indexar dados geométricos, bem como a pesquisa de texto completo

Mas faz diferença usar indexes?

Gráfico com B-tree



Como saber se estou utilizando um index de forma correta?

Explain

- **Fornece informações detalhadas sobre como o PostgreSQL executará a consulta e acessará os dados;**
  - **Tabelas;**
  - **Índices;**
  - **Junções e algoritmos.**

Explain

```
create table teste(nome varchar);

do $$
begin
for i in 1..1000000 loop
insert into teste values (texto(10));
end loop;
end; $$
language plpgsql;

analyze teste;

explain analyze
select * from teste where nome LIKE 'eQi%';

-- Gather  (cost=1000.00..11624.33 rows=100 width=
--   Workers Planned: 2
--   Workers Launched: 2
--   ->  Parallel Seq Scan on teste  (cost=0.00..
--         Filter: ((nome)::text ~~ 'eQi%'::text)
--         Rows Removed by Filter: 333333
-- Planning Time: 6.960 ms
-- Execution Time: 70.286 ms
```

```
create index idxtext on teste(nome);

analyze teste;

-- index não acessado pois usa % pesquisa
-- e btree faz busca de texto completa
explain analyze
select * from teste where nome LIKE 'eQi%';

-- Gather  (cost=1000.00..11624.33 rows=100 wi
--   Workers Planned: 2
--   Workers Launched: 2
--   ->  Parallel Seq Scan on teste  (cost=0.0
--         Filter: ((nome)::text ~~ 'eQi%'::te
--         Rows Removed by Filter: 333333
-- Planning Time: 3.241 ms
-- Execution Time: 43.890 ms
```

Vamos melhorar

```
CREATE EXTENSION pg_trgm;

create index idxtextTrgm on teste using GIN(nome gin_trgm_ops);

analyze teste;

explain analyze
select * from teste where nome LIKE 'eQi%';

-- Bitmap Heap Scan on teste  (cost=40.77..401.22 rows=100 width
--   Recheck Cond: ((nome)::text ~~ 'eQi%'::text)
--   Rows Removed by Index Recheck: 27
--   Heap Blocks: exact=29
--   ->  Bitmap Index Scan on idxtexttrgm  (cost=0.00..40.75 row
--         Index Cond: ((nome)::text ~~ 'eQi%'::text)
-- Planning Time: 2.704 ms
-- Execution Time: 2.277 ms

-- Conclusão index acessado
```

Bitmap

```
do $$
begin
for i in 1..1000000 loop
insert into pessoa(nome, genero) values (texto(10),'M');
end loop;
end; $$
language plpgsql;

do $$
begin
for i in 1..1000000 loop
insert into pessoa(nome, genero) values (texto(10),'F');
end loop;
end; $$
language plpgsql;

analyze pessoa;
```

```
explain analyze
select * from pessoa where genero = 'M';

-- Seq Scan on pessoa  (cost=0.00..37739.00 rows=989800
--   Filter: ((genero)::text = 'M'::text)
--   Rows Removed by Filter: 1000000
-- Planning Time: 1.150 ms
-- Execution Time: 334.616 ms

create extension btree_gin;
```

```
create index idxgeneroBitmap on pessoa using gin (genero);
analyze pessoa;

explain analyze select * from pessoa
where genero = 'M';

-- Bitmap Heap Scan on pessoa  (cost=9268.23..34568.90 rows=1004933
--   Recheck Cond: ((genero)::text = 'M'::text)
--   Heap Blocks: exact=6370
--   ->  Bitmap Index Scan on idxgenerobitmap  (cost=0.00..9017.00 r
--         Index Cond: ((genero)::text = 'M'::text)
-- Planning Time: 0.161 ms
-- Execution Time: 139.615 ms
```

Explain mais a fundo

```
explain analyze Select info->>'student'
FROM students
where info->>'score' = '10'
group by info->>'student';


-- Group  (cost=57197.59..58756.02 rows=15000 width=32) (actual time=562.209..577.642 rows=29594 loops=1)
--   Group Key: ((info ->> 'student'::text))
--   ->  Gather Merge  (cost=57197.59..58687.27 rows=12500 width=32) (actual time=562.208..574.561 rows=29594 loops=1)
--         Workers Planned: 2
--         Workers Launched: 2
--         ->  Group  (cost=56197.56..56244.44 rows=6250 width=32) (actual time=505.352..507.072 rows=9865 loops=3)
--               Group Key: ((info ->> 'student'::text))
--               ->  Sort  (cost=56197.56..56213.19 rows=6250 width=32) (actual time=505.349..505.759 rows=9956 loops=3)
--                     Sort Key: ((info ->> 'student'::text))
--                     Sort Method: quicksort  Memory: 891kB
--                     Worker 0:  Sort Method: quicksort  Memory: 829kB
--                     Worker 1:  Sort Method: quicksort  Memory: 834kB
--                     ->  Parallel Seq Scan on students  (cost=0.00..55803.51 rows=6250 width=32) (actual time=0.599..486.594 rows=9956 loops=3)
--                           Filter: ((info ->> 'score'::text) = '10'::text)
--                           Rows Removed by Filter: 990044
-- Planning Time: 7.870 ms
-- Execution Time: 579.353 ms
```

Dúvidas?

Obrigado

Utilize o QR Code e me add no LinkedIn!