

A grayscale photograph of a hand holding a small elephant. The background is a world map composed of dots. The title 'Testes de Performance e Índices' is overlaid on the map. A small orange and teal bar is at the top left.

Testes de Performance e Índices

Aplicação de índices e monitoramento de desempenho.

- Antônio Carlos D.
- Elimar Macena

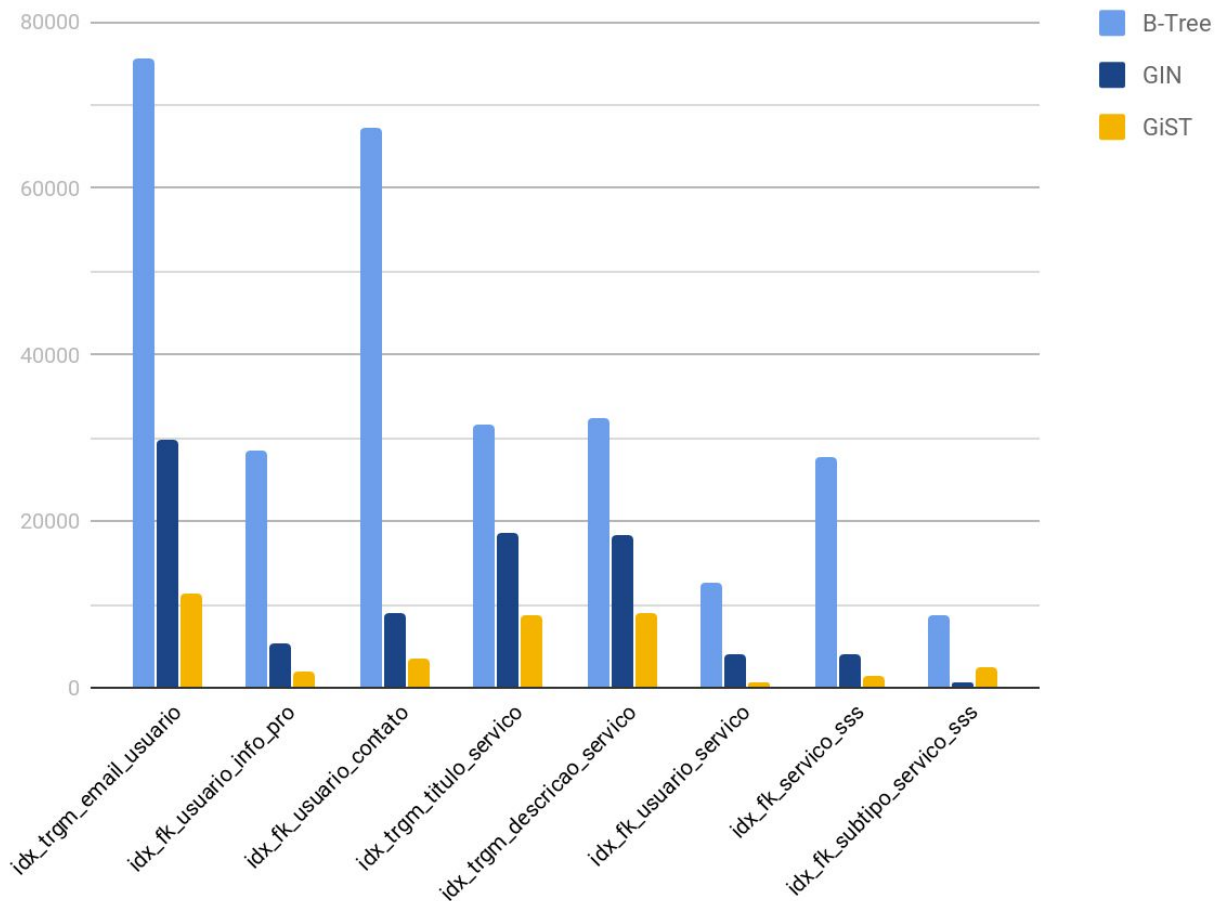


Tempo para criação dos índices

- Índices do tipo GiST e GIN são criados cerca de 1.69 a 19.76 vezes mais rápido do que índices B-Tree.
- Índices GiST foram criados em menor tempo do que os do tipo GIN (Com exceção do último caso).

Nome do índice	B-Tree	GIN	GiST
idx_trgm_email_usuario	75478,977	29838,993	11291,060
idx_fk_usuario_info_pro	28515,702	5412,203	1964,828
idx_fk_usuario_contato	67331,685	8880,372	3436,841
idx_trgm_titulo_servico	31579,654	18686,769	8822,955
idx_trgm_descricao_servico	32404,043	18423,241	8877,492
idx_fk_usuario_servico	12622,656	3968,364	759,514
idx_fk_servico_sss	27595,427	4062,878	1396,636
idx_fk_subtipo_servico_sss	8839,872	585,818	2556,507

Tempo de criação dos índices (ms)



Query #1 - S/ Índice

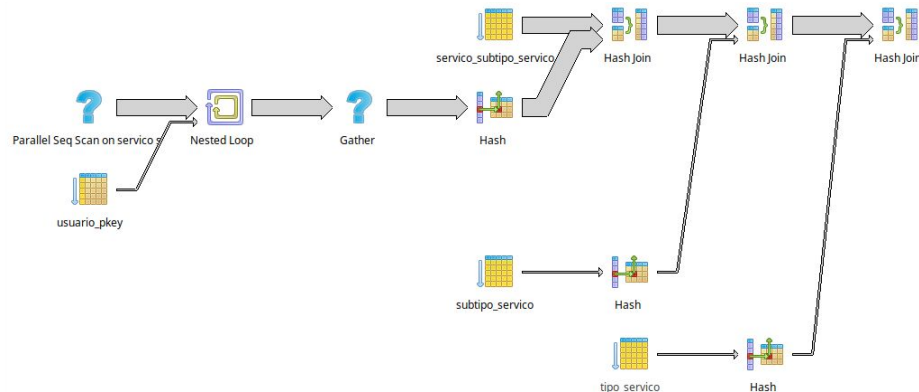
1: `servico_subtipo_servico` [Linha 7]: 1333624 linhas candidatas e custo de 20545.24, devido a busca ser realizada sequencialmente.

2: Loop aninhado [Linha 13]: 2080 linhas candidatas e custo de 31490.62, devido a busca sequencial realizada na tabela “servico” (Linha 14, custo 17919.22) para buscar linhas contendo a palavra buscada.

3: A quantidade de linhas candidatas é bastante alta, por mais que não serão todas executadas necessariamente, isso indica pouca eficiência para restringir os registros.

EXPLAIN ANALYZE

```
SELECT u.*, s., ts.name, ss.name FROM usuario AS u
INNER JOIN servico AS s ON u.id = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE s.descricao ILIKE '%python%';
```



QUERY PLAN	
1	Hash Join (cost=33055.37..58759.87 rows=9988 width=421) (actual time=920.168..4536.562 rows=9455 loops=1)
2	Hash Cond: (ss.fk_tipo_servico = ts.id)
3	-> Hash Join (cost=33054.19..58729.55 rows=9988 width=347) (actual time=920.016..4508.841 rows=9455 loops=1)
4	Hash Cond: (sbs.fk_subtipo_servico = ss.id)
5	-> Hash Join (cost=33052.34..58698.55 rows=9988 width=229) (actual time=919.835..4479.927 rows=9455 loops=1)
6	Hash Cond: (sbs.fk_servico = s.id)
7	-> Seq Scan on servico_subtipo_servico sbs (cost=0.00..20545.24 rows=1333624 width=8) (actual time=0.029..1731.941 rows=1333624 loops=1)
8	-> Hash (cost=32989.92..32989.92 rows=4993 width=225) (actual time=918.169..918.170 rows=4707 loops=1)
9	Buckets: 8192 Batches: 1 Memory Usage: 1393kB
10	-> Gather (cost=1000.43..32989.92 rows=4993 width=225) (actual time=0.740..901.367 rows=4707 loops=1)
11	Workers Planned: 2
12	Workers Launched: 2
13	-> Nested Loop (cost=0.43..31490.62 rows=2080 width=225) (actual time=0.543..897.734 rows=1569 loops=3)
14	-> Parallel Seq Scan on servico s (cost=0.00..17919.22 rows=2080 width=140) (actual time=0.450..756.531 rows=1569 loops=3)
15	Filter: ((descricao)::text ~* 'python')::text)
16	Rows Removed by Filter: 220653
17	-> Index Scan using usuario pkey on usuario u (cost=0.43..6.52 rows=1 width=85) (actual time=0.079..0.079 rows=1 loops=4707)
18	Index Cond: (id = s.fk_usuario)
19	-> Hash (cost=1.38..1.38 rows=38 width=126) (actual time=0.158..0.159 rows=38 loops=1)
20	Buckets: 1024 Batches: 1 Memory Usage: 11kB
21	-> Seq Scan on subtipo_servico ss (cost=0.00..1.38 rows=38 width=126) (actual time=0.026..0.095 rows=38 loops=1)
22	-> Hash (cost=1.08..1.08 rows=8 width=82) (actual time=0.099..0.100 rows=8 loops=1)
23	Buckets: 1024 Batches: 1 Memory Usage: 9kB
24	-> Seq Scan on tipo_servico ts (cost=0.00..1.08 rows=8 width=82) (actual time=0.065..0.078 rows=8 loops=1)
25	Planning time: 1.096 ms
26	Execution time: 4547.825 ms

Query #1 - C/ GIN

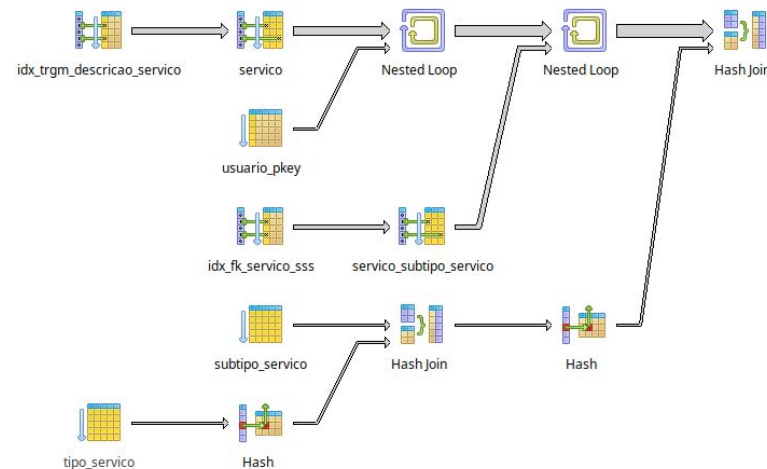
1: Ao realizar a consulta com o GIN, o planejador é bastante astuto, em vez de realizar uma busca sequencial na tabela “servico”, ele usa o índice “descricao_servico” [Linha 8] para buscar os que possuem o termo em sua descrição.

2: O uso de índice também é aplicado à tabela “servico_subtipo_servico” (tabela que interliga as tabelas “servico” e “subtipo_servico”) [Linha 15].


3: Ao realizar o filtro nas duas tabelas acima(algo que não ocorreu na consulta sem índice), a quantidade de linhas a serem trabalhadas cai consideravelmente , o que diminui o tempo e outros recursos gastos com joins e loops.

EXPLAIN ANALYZE

```
SELECT u.*, s.*, ts.nome, ss.nome FROM usuario AS u
INNER JOIN servico AS s ON u.id = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE s.descricao ILIKE '%python%';
```



QUERY PLAN	
1	Hash Join (cost=63.81..1816.86 rows=98 width=421) (actual time=3.753..380.892 rows=9455 loops=1)
2	Hash Cond: (sbs.fk_subtipo_servico = ss.id)
3	-> Nested Loop (cost=60.66..1812.37 rows=98 width=229) (actual time=3.429..353.867 rows=9455 loops=1)
4	-> Nested Loop (cost=44.81..646.24 rows=49 width=225) (actual time=3.368..192.381 rows=4707 loops=1)
5	-> Bitmap Heap Scan on servico s (cost=44.38..232.43 rows=49 width=140) (actual time=3.311..77.243 rows=4707 loops=1)
6	Recheck Cond: ((descricao)::text ~* '%python%')::text)
7	Heap Blocks: exact=4051
8	-> Bitmap Index Scan on idx_trgm_descricao_servico (cost=0.00..44.37 rows=49 width=0) (actual time=2.655..2.656 rows=4707 loops=1)
9	Index Cond: ((descricao)::text ~* '%python%')::text)
10	-> Index Scan using usuario_pkey on usuario u (cost=0.43..0.45 rows=1 width=85) (actual time=0.020..0.020 rows=1 loops=4707)
11	Index Cond: (id = s.fk_usuario)
12	-> Bitmap Heap Scan on servico_subtipo_servico sbs (cost=15.85..23.78 rows=2 width=0) (actual time=0.023..0.025 rows=2 loops=4707)
13	Recheck Cond: (fk_servico = s.id)
14	Heap Blocks: exact=4730
15	-> Bitmap Index Scan on idx_fk_servico_sss (cost=0.00..15.85 rows=2 width=0) (actual time=0.012..0.012 rows=2 loops=4707)
16	Index Cond: (fk_servico = s.id)
17	-> Hash (cost=2.67..2.67 rows=38 width=200) (actual time=0.280..0.281 rows=38 loops=1)
18	Buckets: 1024 Batches: 1 Memory Usage: 12kB
19	-> Hash Join (cost=1.18..2.67 rows=38 width=200) (actual time=0.065..0.219 rows=38 loops=1)
20	Hash Cond: (ss.fk_tipo_servico = ts.id)
21	-> Seq Scan on subtipo_servico ss (cost=0.00..1.38 rows=38 width=126) (actual time=0.009..0.061 rows=38 loops=1)
22	-> Hash (cost=1.08..1.08 rows=8 width=82) (actual time=0.037..0.038 rows=8 loops=1)
23	Buckets: 1024 Batches: 1 Memory Usage: 9kB
24	-> Seq Scan on tipo_servico ts (cost=0.00..1.08 rows=8 width=82) (actual time=0.007..0.019 rows=8 loops=1)
25	Planning time: 1.362 ms
26	Execution time: 391.591 ms

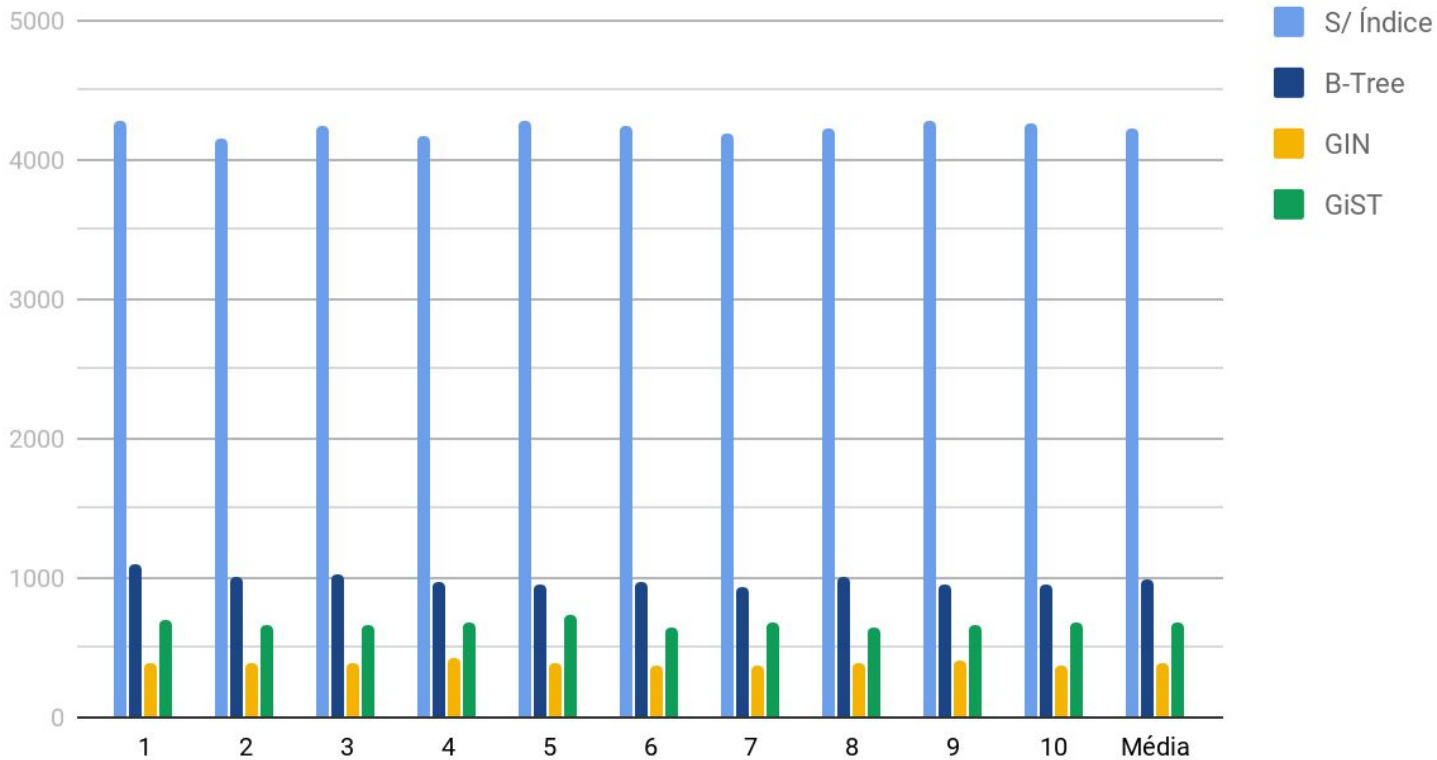


Tempo para execução da query #1

- Nesta query o speedup foi pequeno, entre 3.87 e 11.4 vezes, devido o uso de índices.

Núm. Teste	S/ Índice	B-Tree	GIN	GiST
1	4284,868	1107,781	398,768	694,101
2	4152,059	1016,05	389,521	658,281
3	4239,17	1021,336	381,307	657,276
4	4174,291	973,195	418,207	679,957
5	4276,289	952,328	398,668	736,23
6	4250,92	970,29	372,769	646,744
7	4182,755	938,355	378,098	676,183
8	4226,226	1012,982	390,085	652,099
9	4284,036	956,974	401,202	670,077
10	4258,737	958,401	372,791	676,616
Média	4232,935	990,769	390,142	674,756

Tempo de execução da query #1 (ms)



Query #2 - S/ Índice

1: `servico_subtipo_servico` [Linha 9]: 555677 linhas candidatas e custo de 12765.77, devido a busca ser realizada sequencialmente.

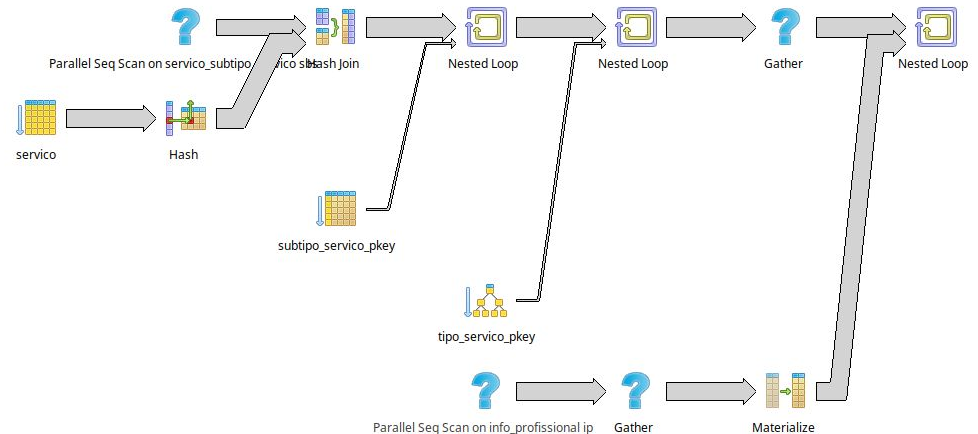
2: `servico` [Linha 12]: Custo de 22780.34, devido busca sequencial pela chave estrangeira igual à solicitada.

3: `info_profissional` [Linha 24]: Busca sequencial com custo 25591.83, em busca da chave estrangeira solicitada.

4: pouca eficiência em filtrar os registros novamente aumentou o custo por laço.

EXPLAIN ANALYZE

```
SELECT ip.* FROM info_profissional AS ip
INNER JOIN servico AS s ON ip.fk_usuario = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE ip.fk_usuario = 1042666;
```



QUERY PLAN	TEXT
1	Nested Loop (cost=24780.62..64597.77 rows=4 width=78) (actual time=1988.312..2034.093 rows=2 loops=1)
2	-> Gather (cost=23780.62..38005.68 rows=2 width=4) (actual time=1885.399..1885.510 rows=2 loops=1)
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Nested Loop (cost=22780.62..37005.48 rows=1 width=4) (actual time=1678.689..1881.550 rows=1 loops=3)
6	-> Hash Join (cost=22780.49..37004.94 rows=1 width=8) (actual time=1678.664..1881.522 rows=1 loops=3)
7	-> Hash Join (cost=22780.35..37004.78 rows=1 width=8) (actual time=1678.645..1881.500 rows=1 loops=3)
8	Hash Cond: (sbs.fk_servico = s.id)
9	-> Parallel Seq Scan on servico_subtipo_servico sbs (cost=0.00..12765.77 rows=555677 width=8) (actual time=0.021..783.585 rows=44454)
10	-> Hash (cost=22780.34..22780.34 rows=1 width=8) (actual time=295.674..295.676 rows=1 loops=3)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on servico s (cost=0.00..22780.34 rows=1 width=8) (actual time=272.455..295.648 rows=1 loops=3)
13	Filter: (fk_usuario = 1042666)
14	Rows Removed by Filter: 666666
15	-> Index Scan using subtipo_servico_pkey on subtipo_servico ss (cost=0.14..0.16 rows=1 width=8) (actual time=0.022..0.023 rows=1 loops=2)
16	Index Cond: (id = sbs.fk_subtipo_servico)
17	-> Index Only Scan using tipo_servico_pkey on tipo_servico ts (cost=0.13..0.47 rows=1 width=4) (actual time=0.032..0.032 rows=1 loops=2)
18	Index Cond: (id = ss.fk_tipo_servico)
19	Heap Fetches: 0
20	-> Materialize (cost=1000.00..26592.04 rows=2 width=78) (actual time=51.454..74.328 rows=1 loops=2)
21	-> Gather (cost=1000.00..26592.03 rows=2 width=78) (actual time=102.897..148.680 rows=1 loops=1)
22	Workers Planned: 2
23	Workers Launched: 2
24	-> Parallel Seq Scan on info_profissional ip (cost=0.00..25591.83 rows=1 width=78) (actual time=121.370..135.669 rows=0 loops=3)
25	Filter: (fk_usuario = 1042666)
26	Rows Removed by Filter: 454453
27	Planning time: 0.891 ms
28	Execution time: 2034.389 ms

Query #2 - C/ B-Tree

Neste caso, como todos dados envolvidos na consulta são do tipo chave estrangeira e primárias, portanto, inteiros, optou-se por usar o tipo B-Tree.

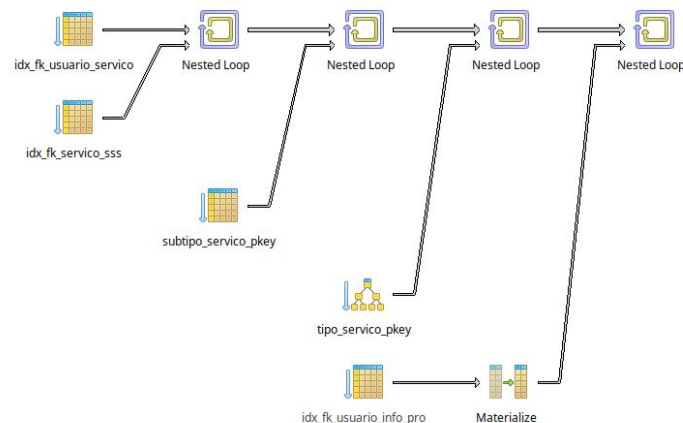
1: `servico_subtipo_servico` [Linha 7]: Aplicação do índice “`idx_fk_servico_sss`”, custo de 8.46.

2: `servico` [Linha 5]: Aplicação do índice “`idx_fk_usuario_servico`”, custo de 8.44.

3: `info_profissional` [Linha 15]: Aplicação do índice “`idx_fk_usuario_info_pro`”, custo de 8.46.

EXPLAIN ANALYZE

```
SELECT ip.* FROM info_profissional AS ip
INNER JOIN servico AS s ON ip.fk_usuario = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE ip.fk_usuario = 1042666;
```



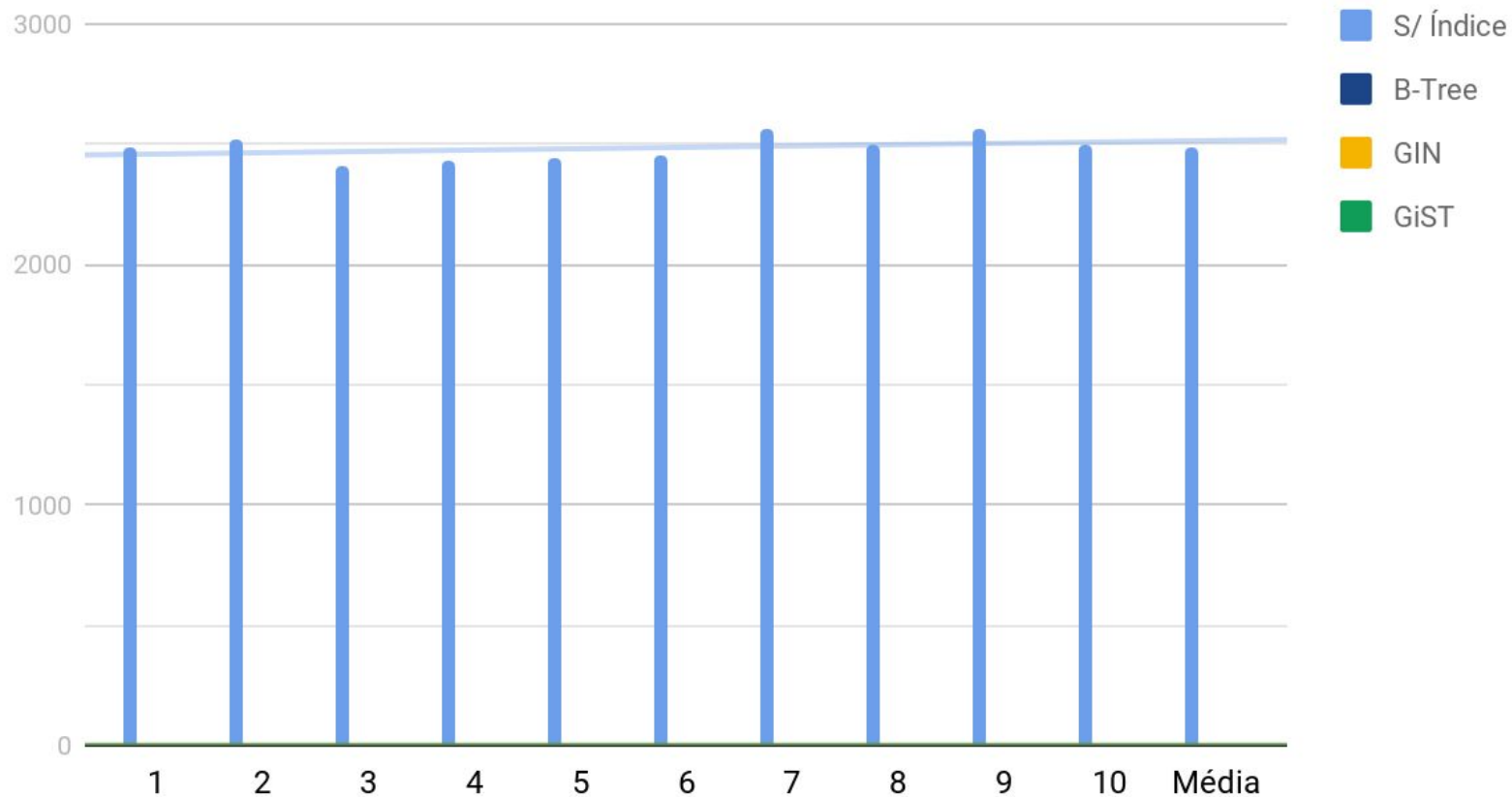
	QUERY PLAN text
1	Nested Loop (cost=1.55..26.76 rows=4 width=78) (actual time=0.092..0.135 rows=2 loops=1)
2	-> Nested Loop (cost=1.12..18.25 rows=2 width=4) (actual time=0.069..0.098 rows=2 loops=1)
3	-> Nested Loop (cost=0.99..17.24 rows=2 width=8) (actual time=0.047..0.067 rows=2 loops=1)
4	-> Nested Loop (cost=0.85..16.93 rows=2 width=8) (actual time=0.034..0.044 rows=2 loops=1)
5	-> Index Scan using idx fk usuario servico on servico s (cost=0.42..8.44 rows=2) (actual time=0.034..0.044 rows=2 loops=1)
6	Index Cond: (fk usuario = 1042666)
7	-> Index Scan using idx fk servico sss on servico_subtipo_servico sbs (cost=0.42..8.44 rows=2) (actual time=0.034..0.044 rows=2 loops=1)
8	Index Cond: (fk servico = s.id)
9	-> Index Scan using subtipo_servico_pkey on subtipo_servico ss (cost=0.14..0.16 rows=2) (actual time=0.034..0.044 rows=2 loops=1)
10	Index Cond: (id = sbs.fk_subtipo_servico)
11	-> Index Only Scan using tipo_servico_pkey on tipo_servico ts (cost=0.13..0.47 rows=1 width=4) (actual time=0.034..0.044 rows=1 loops=1)
12	Index Cond: (id = ss.fk_tipo_servico)
13	Heap Fetches: 2
14	-> Materialize (cost=0.43..8.47 rows=2 width=78) (actual time=0.010..0.013 rows=1 loops=2)
15	-> Index Scan using idx fk usuario info pro on info_profissional ip (cost=0.43..8.46 rows=2) (actual time=0.010..0.013 rows=1 loops=2)
16	Index Cond: (fk usuario = 1042666)
17	Planning time: 0.605 ms
18	Execution time: 0.245 ms

Tempo para execução da query #2

- O Speedup foi tão alto que (chegando a 14 mil) que não é possível comparar o tempo de execução visualmente.
- Em TODAS consultas com B-Tree, GIN, GiST, o tempo de planejamento foi superior ao tempo de execução.

#	TEMPO DE EXECUÇÃO (ms)				SpeedUp de execução (Comparado a execução sem índice)		
	S/ Índice	B-Tree	GIN	GiST	B-Tree	GIN	GiST
1	2487,198	0,198	0,296	0,358	12561,606	8402,696	6947,48
2	2521,016	0,182	0,231	0,265	13851,736	10913,489	9513,268
3	2413,946	0,176	0,228	0,258	13715,602	10587,482	9356,38
4	2430,192	0,176	0,227	0,292	13807,909	10705,692	8322,575
5	2436,683	0,211	0,268	0,304	11548,261	9092,101	8015,405
6	2455,441	0,175	0,219	0,309	14031,091	11212,059	7946,411
7	2568,601	0,208	0,235	0,309	12349,043	10930,217	8312,625
8	2498,995	0,203	0,25	0,294	12310,32	9995,98	8499,983
9	2566,559	0,207	0,237	0,25	12398,836	10829,363	10266,236
10	2492,673	0,223	0,262	0,262	11177,906	9514,019	9514,019
Média	2487,13	0,196	0,245	0,29	12775,231	10218,31	8669,438

Tempo de execução da query #2



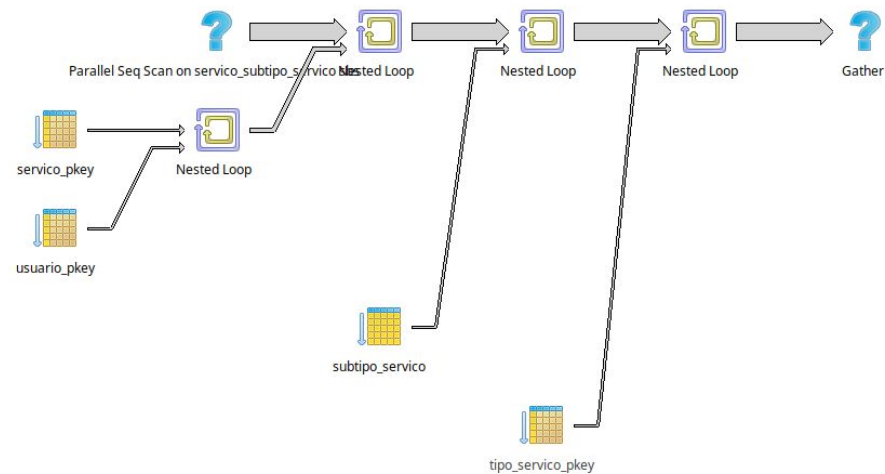
Query #3 - S/ Índice

1: `servico_subtipo_servico` [Linha 9]: Dos 15174.45 de custo total, 14171.86 (93.39%) é originário da busca sequencial por uma chave estrangeira específica.

2: A consulta só não foi mais custosa pelo fato do planejador utilizar os índices nativos nas cláusulas "ON" dos join's [Linhas 13, 15, 18].

EXPLAIN ANALYZE

```
SELECT u.*, s.*, ts.nome, ss.nome FROM usuario AS u
INNER JOIN servico AS s ON u.id = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE sbs.fk_servico = 666566;
```



QUERY PLAN	text
1	Gather (cost=1000.99..15174.45 rows=2 width=421) (actual time=106.587..109.225 rows=2 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Nested Loop (cost=0.99..14174.25 rows=1 width=421) (actual time=102.216..102.238 rows=1 loops=3)
5	-> Nested Loop (cost=0.85..14173.71 rows=1 width=347) (actual time=102.203..102.221 rows=1 loops=3)
6	Join Filter: (sbs.fk_subtipo_servico = ss.id)
7	Rows Removed by Join Filter: 0
8	-> Nested Loop (cost=0.85..14171.86 rows=1 width=229) (actual time=102.189..102.203 rows=1 loops=3)
9	-> Parallel Seq Scan on servico_subtipo_servico sbs (cost=0.00..14154.96 rows=1 width=8) (actual time=
10	Filter: (fk_servico = 666566)
11	Rows Removed by Filter: 444541
12	-> Nested Loop (cost=0.85..16.89 rows=1 width=225) (actual time=0.054..0.059 rows=1 loops=2)
13	-> Index Scan using servico_pkey on servico s (cost=0.42..8.44 rows=1 width=140) (actual time=0.
14	Index Cond: (id = 666566)
15	-> Index Scan using usuario_pkey on usuario u (cost=0.43..8.45 rows=1 width=85) (actual time=0.
16	Index Cond: (id = s.fk_usuario)
17	-> Seq Scan on subtipo_servico ss (cost=0.00..1.38 rows=38 width=126) (actual time=0.016..0.017 rows=2 loop
18	-> Index Scan using tipo_servico_pkey on tipo_servico ts (cost=0.13..0.47 rows=1 width=82) (actual time=0.015..0.
19	Index Cond: (id = ss.fk_tipo_servico)
20	Planning time: 0.544 ms
21	Execution time: 109.310 ms

Query #3 - C/ B-Tree

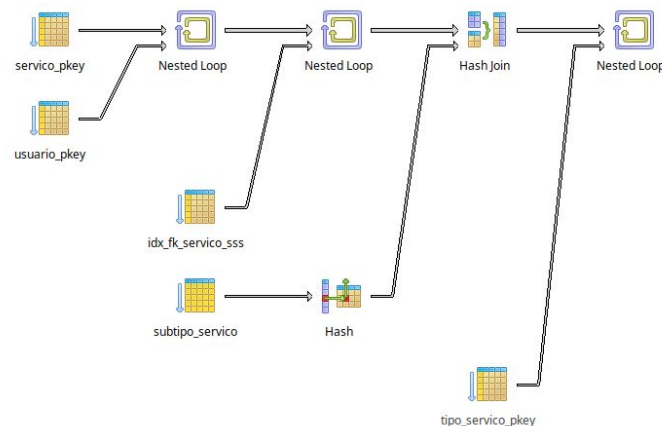
1: O planejador decidiu utilizar os índices nativos nas tabelas “servico”, “usuario”, “tipo_servico” [Linhas 6, 8, 15].

2: Somente no caso da tabela “servico_subtipo_servico”, responsável pelo gargalo na consulta sem índice, que o planejador decidiu aplicar o índice “personalizado”.

3: Seq Scan [Linha 14]: Como a tabela “subtipo_servico” têm apenas 38 linhas candidatas, o planejador pode ter inferido que a busca sequencial seria menos custosa que uma ordenação e busca binária, operações necessárias para indexação.

EXPLAIN ANALYZE

```
SELECT u.*, s.*, ts.nome, ss.nome FROM usuario AS u
INNER JOIN servico AS s ON u.id = s.fk_usuario
INNER JOIN servico_subtipo_servico AS sbs ON s.id = sbs.fk_servico
INNER JOIN subtipo_servico AS ss ON ss.id = sbs.fk_subtipo_servico
INNER JOIN tipo_servico AS ts ON ts.id = ss.fk_tipo_servico
WHERE sbs.fk_servico = 666566;
```



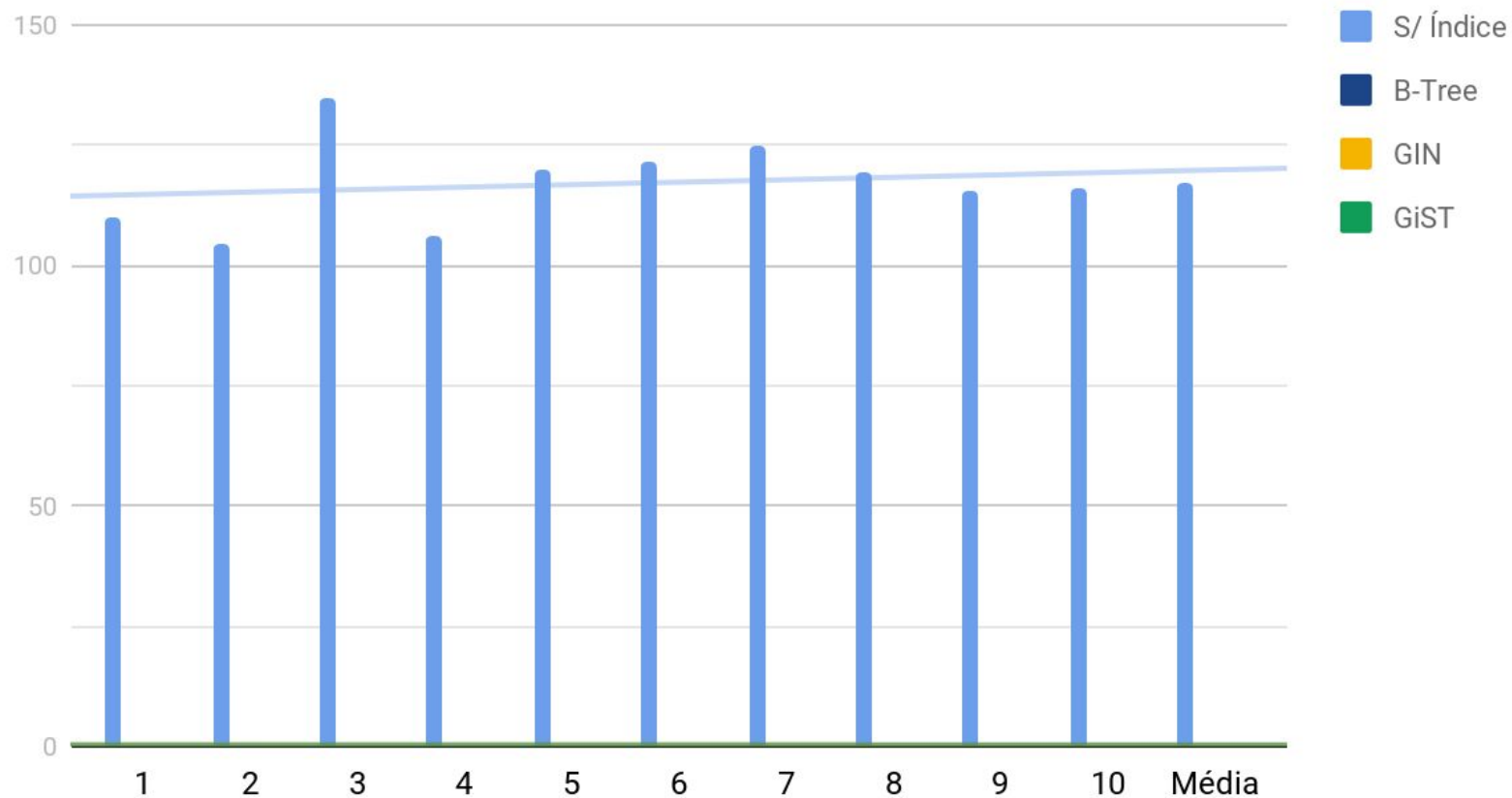
QUERY PLAN	text
1	Nested Loop (cost=3.27..28.24 rows=2 width=421) (actual time=0.182..0.211 rows=2 loops=1)
2	-> Hash Join (cost=3.14..27.23 rows=2 width=347) (actual time=0.172..0.191 rows=2 loops=1)
3	Hash Cond: (sbs.fk_subtipo_servico = ss.id)
4	-> Nested Loop (cost=1.28..25.37 rows=2 width=229) (actual time=0.040..0.054 rows=2 loops=1)
5	-> Nested Loop (cost=0.85..16.89 rows=1 width=225) (actual time=0.027..0.031 rows=1 loops=1)
6	-> Index Scan using servico_pkey on servico s (cost=0.42..8.44 rows=1 width=140) (act
7	Index Cond: (id = 666566)
8	-> Index Scan using usuario_pkey on usuario u (cost=0.43..8.45 rows=1 width=85) (actu
9	Index Cond: (id = s.fk_usuario)
10	-> Index Scan using idx_fk_servico_sss on servico_subtipo_servico sbs (cost=0.43..8.46 rows
11	Index Cond: (fk_servico = 666566)
12	-> Hash (cost=1.38..1.38 rows=38 width=126) (actual time=0.119..0.121 rows=38 loops=1)
13	Buckets: 1024 Batches: 1 Memory Usage: 11kB
14	-> Seq Scan on subtipo_servico ss (cost=0.00..1.38 rows=38 width=126) (actual time=0.009..0.010
15	-> Index Scan using tipo_servico_pkey on tipo_servico ts (cost=0.13..0.47 rows=1 width=82) (actual time
16	Index Cond: (id = ss.fk_tipo_servico)
17	Planning time: 0.709 ms
18	Execution time: 0.293 ms

Tempo para execução da query #3

- O Speedup foi tão alto que (chegando a 14 mil) que não é possível comparar o tempo de execução visualmente.
- Em TODAS consultas com B-Tree, GIN, GiST, o tempo de planejamento foi superior ao tempo de execução.

#	TEMPO DE EXECUÇÃO (ms)				SpeedUp de execução (Comparado a execução sem índice)		
	S/ Índice	B-Tree	GIN	GiST	B-Tree	GIN	GiST
1	110,028	0,361	0,419	0,367	304,787	262,597	299,804
2	104,738	0,27	0,33	0,343	387,919	317,388	305,359
3	134,541	0,316	0,359	0,315	425,763	374,766	427,114
4	106,147	0,305	0,29	0,35	348,023	366,024	303,277
5	119,873	0,328	0,299	0,324	365,466	400,913	369,978
6	121,75	0,276	0,357	0,3	441,123	341,036	405,833
7	124,989	0,273	0,301	0,333	457,835	415,246	375,342
8	119,614	0,297	0,289	0,303	402,741	413,889	394,766
9	115,633	0,27	0,336	0,301	428,27	344,146	384,163
10	115,814	0,324	0,295	0,307	357,451	392,59	377,244
Média	117,313	0,302	0,328	0,324	391,938	362,859	364,288

Tempo de execução da query #3



**Não vai ter slide de
“OBRIGADO”. FLW, VLW.**



PostgreSQL