

### Questionário 3 - Tipos de índices para situações distintas

Nome: Gabriel Stankevix Soares

1. Criar duas tabelas, sendo que em qualquer uma delas possua: um campo texto, um campo discreto (categórico) e uma chave estrangeira em relação à outra tabela.

```
CREATE TABLE produto (  
    cod_produto serial primary key,  
    produto varchar,  
    origem varchar(1) check (origem in ('I','N'))  
);  
  
CREATE TABLE venda (  
    cod_venda serial primary key,  
    quantidade integer,  
    preço_unitario float,  
    cod_produto serial references produto(cod_produto)  
);
```

2. Inserir dados nas duas tabelas de maneira que possamos testar os índices. Faça um analyze em cada tabela.

```
CREATE OR REPLACE FUNCTION get_random_number(INTEGER, INTEGER) RETURNS  
INTEGER AS $$  
DECLARE  
    start_int ALIAS FOR $1;  
    end_int ALIAS FOR $2;  
BEGIN  
    RETURN trunc(random() * (end_int-start_int) + start_int);  
END;  
$$ LANGUAGE 'plpgsql' STRICT;  
  
-- IMPORTADO  
do $$  
begin  
    for i IN 1..3000 LOOP  
        insert into produto values (i,  
                                     (CASE get_random_number(1,5)  
                                      WHEN 1 THEN 'Cerveja'  
                                      WHEN 2 THEN 'Vinho'  
                                      WHEN 3 THEN 'Agua'  
                                      WHEN 4 THEN 'Pinga'  
                                      WHEN 5 THEN 'Vodka' END),  
                                     'I');  
    end loop;  
end $$;  
  
-- NACIONAL  
do $$  
begin
```

```

        for i IN 3001..6000 LOOP
            insert into produto values (i,
                                      (CASE get_random_number(1,5)
                                       WHEN 1 THEN 'Cerveja'
                                       WHEN 2 THEN 'Vinho'
                                       WHEN 3 THEN 'Agua'
                                       WHEN 4 THEN 'Pinga'
                                       WHEN 5 THEN 'Vodka' END),
                                      'N');
        end loop;
end $$;

do $$
begin
    for i IN 1..600000 LOOP
        insert into venda values (i,
                                cast(random()*10 + 1 as int),
                                random() * 10 + 1,

                                get_random_number(1,6000));
    end loop;
end $$;

analyze public.venda; -- Query returned successfully in 238 msec.
analyze public.produto; -- Query returned successfully in 186 msec.

```

**explain analyze select \* from public.venda;**

Data Output	Explain	Messages	Notifications
<div> <div>▲</div> <div> <b>QUERY PLAN</b>  text </div> <div>🔒</div> </div>			
1	Seq Scan on venda (cost=0.00..9822.00 rows=600000 width=20) (actual time=0.017..71.157 rows=600000 loops=1)		
2	Planning Time: 0.062 ms		
3	Execution Time: 94.158 ms		

**explain analyze select \* from public.produto;**

Data Output	Explain	Messages	Notifications
<div> <div>▲</div> <div> <b>QUERY PLAN</b>  text </div> <div>🔒</div> </div>			
1	Seq Scan on produto (cost=0.00..93.00 rows=6000 width=12) (actual time=0.020..0.457 rows=6000 loops=1)		
2	Planning Time: 0.076 ms		
3	Execution Time: 0.623 ms		

✓

Success

3. Criar três índices, um para o campo textual, um para o campo discreto (bitmap), e para a chave estrangeira.

```
create index idxproduto on produto(produto);

--create extension btree_gin;
create index idxOrigemBitmap on produto using gin (origem);

create index idxcod_produto on venda(cod_produto);
```

4. Formule e mostre o plano das consultas SQL que utilizem cada os índices criados no item anterior.

```
explain analyze
  select v.cod_produto,p.produto,sum((v.quantidade * v.preço_unitario)) as total
  from venda v
 inner join produto p
      on v.cod_produto = p.cod_produto
 group by v.cod_produto,p.produto;
```

Finalize HashAggregate (cost=16678.02..16918.30 rows=24028 width=18) (actual time=746.004..747.166 rows=5999 loops=1)
Group Key: v.cod_produto, p.produto
-> Gather (cost=11271.72..16317.60 rows=48056 width=18) (actual time=701.003..791.913 rows=17994 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial HashAggregate (cost=10271.72..10512.00 rows=24028 width=18) (actual time=389.247..391.405 rows=5998 loops=3)
Group Key: v.cod_produto, p.produto
-> Hash Join (cost=168.00..7146.72 rows=250000 width=22) (actual time=3.990..189.754 rows=200000 loops=3)
Hash Cond: (v.cod_produto = p.cod_produto)
-> Parallel Seq Scan on venda v (cost=0.00..6322.00 rows=250000 width=16) (actual time=0.015..81.293 rows=200000 loops=3)
-> Hash (cost=93.00..93.00 rows=6000 width=10) (actual time=3.911..3.911 rows=6000 loops=3)
Buckets: 8192 Batches: 1 Memory Usage: 322kB
-> Seq Scan on produto p (cost=0.00..93.00 rows=6000 width=10) (actual time=0.666..1.661 rows=6000 loops=3)
Planning Time: 1.708 ms
Execution Time: 807.795 ms

```
explain analyze
  select v.cod_produto,p.produto,sum((v.quantidade * v.preço_unitario)) as total
  from venda v
 inner join produto p
      on v.cod_produto = p.cod_produto
 where p.origem = 'N'
 group by v.cod_produto,p.produto;
```

Finalize HashAggregate (cost=15090.77..15331.05 rows=24028 width=18) (actual time=610.292..610.844 rows=2999 loops=1)
Group Key: v.cod_produto, p.produto
-> Gather (cost=9684.47..14730.35 rows=48056 width=18) (actual time=412.824..619.318 rows=5998 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial HashAggregate (cost=8684.47..8924.75 rows=24028 width=18) (actual time=148.451..149.220 rows=1999 loops=3)
Group Key: v.cod_produto, p.produto
-> Hash Join (cost=143.25..7121.97 rows=125000 width=22) (actual time=1.412..97.534 rows=99889 loops=3)
Hash Cond: (v.cod_produto = p.cod_produto)
-> Parallel Seq Scan on venda v (cost=0.00..6322.00 rows=250000 width=16) (actual time=0.541..23.851 rows=200000 loops=3)
-> Hash (cost=105.75..105.75 rows=3000 width=10) (actual time=1.227..1.227 rows=3000 loops=2)
Buckets: 4096 Batches: 1 Memory Usage: 161kB
-> Bitmap Heap Scan on produto p (cost=35.25..105.75 rows=3000 width=10) (actual time=0.328..0.736 rows=3000 loops=2)
Recheck Cond: ((origem)::text = 'N'::text)
Heap Blocks: exact=17
-> Bitmap Index Scan on idxorigembitmap (cost=0.00..34.50 rows=3000 width=0) (actual time=0.309..0.309 rows=3000 loops=2)
Index Cond: ((origem)::text = 'N'::text)
Planning Time: 0.363 ms
Execution Time: 624.701 ms

explain analyze

```

select v.cod_produto,p.produto,sum((v.quantidade * v.preço_unitario)) as total
from venda v
inner join produto p
    on v.cod_produto = p.cod_produto
where p.origem = 'N' and p.produto LIKE '%n%'
group by v.cod_produto,p.produto;

```

HashAggregate (cost=13384.53..13624.81 rows=24028 width=18) (actual time=402.879..422.533 rows=1483 loops=1)
Group Key: v.cod_produto, p.produto
-> Hash Join (cost=131.42..11529.53 rows=148400 width=22) (actual time=1.405..299.839 rows=148048 loops=1)
Hash Cond: (v.cod_produto = p.cod_produto)
-> Seq Scan on venda v (cost=0.00..9822.00 rows=600000 width=16) (actual time=0.020..93.050 rows=600000 loops=1)
-> Hash (cost=112.87..112.87 rows=1484 width=10) (actual time=1.368..1.368 rows=1484 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 80kB
-> Bitmap Heap Scan on produto p (cost=34.87..112.87 rows=1484 width=10) (actual time=0.434..1.063 rows=1484 loops=1)
Recheck Cond: ((origem)::text = 'N'::text)
Filter: ((produto)::text ~ '%n% '::text)
Rows Removed by Filter: 1516
Heap Blocks: exact=17
-> Bitmap Index Scan on idxorigembitmap (cost=0.00..34.50 rows=3000 width=0) (actual time=0.419..0.419 rows=3000 loops=1)
Index Cond: ((origem)::text = 'N'::text)
Planning Time: 0.307 ms
Execution Time: 423.522 ms