

# Utilização de Índices

Gabriela Alcaide - 14746492

Gustavo Pompermayer Fulanetti Silva - 14760280

Kauê Patrick de Oliveira - 14586261

Pedro Henrique Resnitzky Barbedo - 14657691

Rodrigo Gonçalves Cardoso - 14658330

## CRIAÇÃO E TESTE DE ÍNDICES

O PostgreSQL, SGBD escolhido pelo grupo, cria índices automaticamente para os atributos que estão definidos como chave primária. No entanto, com objetivo de otimizar o banco, foram idealizados e criados, pelo grupo, outros índices em tabelas com um número de dados elevado ou tabelas que, em uma situação real, seriam bastante utilizadas no banco de dados utilizado.

Nesta seção, serão abordados os ganhos fornecidos pelos índices criados automaticamente e criados pelo grupo. Para isso, algumas consultas foram realizadas (consultas que estressam o sistema - devido a sua complexidade-, consultas que são favorecidas pela criação dos índices e consultas que provavelmente seriam realizadas em uma situação real para esse banco de dados).

### Tabela Limpeza

#### Índice padrão:

O índice padrão, criado ao adicionar a restrição de chave primária na tabela Limpeza, é o seguinte:

	indexname name	indexdef text
1	id_limpeza_id	CREATE UNIQUE INDEX id_limpeza_id ON public.limpeza USING btree (id_ambiente, data)

#### Alterações e testes:

Para testar o quanto o índice padrão pode ajudar ou não em consultas, ele foi retirado para testes:

```
ALTER TABLE limpeza DROP CONSTRAINT ID_LIMPEZA_ID;  
DROP INDEX id_limpeza_id;
```

Consultas realizadas para análise de ganho com o índice padrão da tabela Limpeza:

```
SELECT data  
FROM limpeza  
JOIN ambiente  
ON ambiente.id_ambiente = limpeza.id_ambiente
```

Sem a utilização desse índice a consulta levou 293 msec.

Ao reinserir o índice, a consulta passou a levar 238 msec, ou seja, houve uma redução de 55 msec (19%).

```
SELECT data
FROM limpeza
```

Sem a utilização de índice, a consulta levou 176 msec.

Ao reinserir o índice, a consulta passou a levar 156 msec, ou seja, houve uma redução de 20 msec (11%).

```
-- Saber quais funcionários limpam os ambientes 1000, 1001, 1002 ou 1003
SELECT l.id_ambiente, l.data, f.nome, f.sobrenome
FROM limpeza l
JOIN funcionario f ON l.cpf = f.cpf
WHERE l.id_ambiente in ('1000', '1001', '1002', '1003')
AND l.data > '2020-01-01'
ORDER BY l.id_ambiente, l.data;
```

Sem a utilização do índice padrão, essa consulta demorou 120 msec para ser concluída.

Com o índice, a consulta passou a levar 87 msec, ou seja, houve uma redução de 27,5%.

#### Idealização e teste de novo índice:

É comum querer levantar quais limpezas foram realizadas por um determinado funcionário. Por isso, na tabela limpeza, foi interessante colocar um índice no atributo “cpf”:

```
CREATE INDEX idx_cpf_limpeza ON limpeza (cpf)
```

O índice criado é do tipo B-Tree, uma vez que a consulta mais realizada para esse tipo de atributo é uma consulta de igualdade. Esse tipo de consulta é muito bem atendido por índices desse tipo (melhor índice para esse caso).

As seguintes consultas foram realizadas para testar o novo índice inserido no banco de dados:

```
SELECT *
FROM limpeza
WHERE cpf = '424.964.401-23'
```

Sem o índice idealizado pelo grupo: 130 milissegundos

Com o índice idealizado pelo grupo: 90 milissegundos (redução de 30%)

#### **Tabela oferece**

\* é a tabela com mais registros no nosso banco de dados.

#### Índice Padrão

É o índice criado pela restrição de chave primária:

	indexname name	indexdef text
1	id_oferece_id	CREATE UNIQUE INDEX id_oferece_id ON public.oferece USING btree (id_restaurante, tipo, data)

### Alterações e testes:

A primeira consulta testada foi uma consulta que seria favorecida pelo índice criado por padrão:

```
SELECT *  
FROM public.oferece  
ORDER BY tipo ASC, id_restaurante ASC
```

Sem o índice, a consulta levou 2 segundos e 150 milissegundos para ser concluída.

Com o índice, a consulta levou apenas 365 milissegundos para se concluir, ou seja, houve uma redução de 82%.

A seguinte consulta também foi testada:

```
-- Quantidade de cafés da manhã oferecidas por cada restaurante  
SELECT o.id_restaurante, nome, COUNT(*)  
FROM oferece o JOIN restaurante_universitario ru ON o.id_restaurante =  
ru.id_restaurante  
WHERE tipo = 'cafe_manha'  
GROUP BY o.id_restaurante, nome  
ORDER BY id_restaurante ASC;
```

Sem o índice padrão, o tempo exigido para a conclusão da consulta foi de 248 milissegundos.

Já com o índice, a consulta levou apenas 76 milissegundos, havendo, assim, uma redução de 69%.

A última consulta testada para verificar a contribuição do índice padrão foi:

```
SELECT data  
FROM oferece
```

Sem o índice a consulta demorou 300 milissegundos. Porém, ao realizar a inserção do índice, o tempo caiu para 158 milissegundos, ou seja, houve uma redução de 47%

### Idealização e teste de novo índice:

Novo índice: pensando em análises de refeições oferecidas, o grupo percebeu que seria interessante ter um índice no tempo (coluna “data”), uma vez que esse tipo de análise geralmente está atrelado a separar por ano, por mês e assim em diante.

Como “data” é um tipo ordenável e comumente utilizado em comparações de igualdade ou de intervalos, o índice do tipo B-tree se torna a opção mais eficiente para uso. Devido também ao tamanho do banco de dados, não há necessidade da utilização de BRIN, um índice mais leve e pouco menos eficiente. O BRIN só valeria a pena se os dados estivessem fisicamente armazenados em ordem pela data.

Criação do índice para data:

```
CREATE INDEX idx_data_oferece ON oferece (data)
```

Consulta utilizada para análise: para analisar a eficiência e o ganho gerado pelo novo índice, a seguinte consulta foi realizada:

```
SELECT *  
FROM oferece  
WHERE data BETWEEN '2020-01-01' AND '2021-01-01';
```

Tempo sem o novo índice criado pelo grupo: 190 milissegundos

Tempo com o novo índice criado pelo grupo: 130 milissegundos (redução de 31%)

Houveram tentativas de criar esse mesmo índice com outros tipos de índice, porém os resultados foram piores, podem até mesmo se comparar a não criação de índices em algumas consultas.

### Tabela inclui

\* terceira tabela com mais linhas do banco de dados (31307 linhas)

Visualizando o índice padrão que veio na tabela (devido à chave primária):

	indexname name	indexdef text
1	id_inclui_id	CREATE UNIQUE INDEX id_inclui_id ON public.inclui USING btree (nome, tipo, data)

### Alterações e testes:

Para saber o ganho gerado pela utilização do índice criado por padrão pelo SGBD, foram realizados os seguintes testes:

```
SELECT * FROM public.inclui  
ORDER BY nome ASC, tipo ASC, data ASC
```

Sem o índice, a consulta demorou 200 ms para ser concluída. E com o índice, a consulta demorou 130 ms, ou seja, houve uma redução de 35%

```
-- Entender as vezes que um prato foi oferecido em uma refeicao  
SELECT i.Nome as prato, i.tipo as tipo_refeicao, r.valor  
FROM inclui i  
JOIN refeicao r ON i.tipo = r.tipo AND i.data = r.data  
WHERE i.Nome = 'bife_a_parmegiana' AND i.tipo = 'almoco'
```

Sem o índice, essa consulta demorou 100 ms. Com o índice, ela passou a demorar 75 ms, representando uma queda de 25% no tempo gasto.

### Idealização e teste de um novo índice

O grupo percebeu a importância de ter um índice no tipo de comida para análises futuras sobre estatísticas por tipo de refeição.

Criação de índice:

```
CREATE INDEX idx_tipo_inclui ON inclui (tipo)
```

Consulta utilizada para teste:

```
SELECT *  
FROM inclui  
WHERE tipo = 'almoco';
```

Consulta sem o índice criado pelo grupo: 170 ms

Consulta com o índice criado pelo grupo: 130 ms (redução de 23%)

Esse índice é do tipo B-Tree, uma vez que a operação que será realizada é a de igualdade. O Hash suporta operações de igualdade, mas para esse caso, não apresentou melhoras em relação ao índice do tipo B-Tree.

### **Otimizando consultas específicas que possivelmente serão utilizadas várias vezes no banco de dados**

Foi elaborada, com ajuda do Chatgpt, uma consulta que é realmente pesada, mas com consultas comuns entre as tabelas (onde os índices atuam) para verificar a eficiência e o ganho com novos índices.

```
WITH pratos_por_mes AS (  
  SELECT  
    i.nome AS prato,  
    TO_CHAR(i.data, 'YYYY-MM') AS mes,  
    COUNT(*) AS vezes_oferecido  
  FROM inclui i  
  GROUP BY i.nome, mes  
) ,  
  
rank_pratos AS (  
  SELECT  
    prato,  
    tipo,  
    RANK() OVER (PARTITION BY tipo ORDER BY total DESC) AS rank_popularidade  
  FROM (  
    SELECT  
      i.nome AS prato,  
      i.tipo,  
      COUNT(*) AS total  
    FROM inclui i  
    WHERE i.data BETWEEN '2021-01-01' AND '2021-02-01'  
    GROUP BY i.nome, i.tipo  
  ) AS contagem  
)  
  
SELECT  
  c.nome AS campus,  
  i.nome AS prato_oferecido,  
  ru.id_restaurante AS ru_responsavel,  
  r.tipo AS refeicao,  
  ru.capacidade,  
  pratos_por_mes.vezes_oferecido,  
  i.data,  
  rank_pratos.rank_popularidade,
```

```

-- total global do prato por nome e tipo
(
  SELECT COUNT(*)
  FROM inclui i2
  WHERE i2.nome = i.nome AND i2.tipo = i.tipo
) AS total_global_do_prato,

-- média da capacidade por campus
  AVG(ru.capacidade) OVER (PARTITION BY c.nome) AS
media_capacidade_por_campus,

-- campo adicional da tabela prato
p.Cal_100g,

-- extras
CASE
  WHEN r.tipo LIKE '%jantar%' THEN 'Noturno'
  WHEN r.tipo LIKE '%almoco%' OR r.tipo LIKE '%manha%' THEN 'Diurno'
  ELSE 'Outro'
END AS periodo_refeicao,
EXTRACT(DOW FROM i.data) AS dia_da_semana,
LENGTH(i.nome) AS tamanho_nome_prato

FROM campus c
JOIN restaurante_universitario ru ON c.nome = ru.nome
JOIN oferece o ON ru.id_restaurante = o.id_restaurante
JOIN refeicao r ON o.tipo = r.tipo AND o.data = r.data
JOIN inclui i ON r.tipo = i.tipo AND r.data = i.data
LEFT JOIN pratos_por_mes ON pratos_por_mes.prato = i.nome AND
pratos_por_mes.mes = TO_CHAR(i.data, 'YYYY-MM')
LEFT JOIN rank_pratos ON rank_pratos.prato = i.nome AND rank_pratos.tipo = r.tipo
LEFT JOIN prato p ON p.nome = i.nome

WHERE i.data BETWEEN '2021-01-01' AND '2021-02-01'
ORDER BY
  c.nome,
  rank_pratos.rank_popularidade NULLS LAST,
  EXTRACT(DOW FROM i.data),
  LENGTH(i.nome) DESC,
  ru.capacidade DESC;

```

Com os índices padrão + os inseridos até agora, ela demora 1.16 segundos.

Percebe-se que é normal utilizar a data de “inclui” nos joins e nas condições. Por isso, o grupo criou um índice para esse caso.

```
CREATE INDEX idx_inclui_data ON inclui(data);
```

Com o novo índice, a consulta leva 0.97 segundos (uma melhora de 16%). Anteriormente, levava 1,15s.

O índice criado é do tipo B-Tree, uma vez que a operação realizada é de igualdade ou então do tipo BETWEEN. Além de funcionar bem em consultas que utilizam o order by.

### Novo índice na tabela funcionário

É muito comum procurarmos pelo nome e sobrenome do funcionário nas consultas, por isso, é conveniente criar um índice para essa situação.

```
CREATE INDEX idx_funcionario_nome_sobrenome on funcionario (nome, sobrenome)
```

Foi criado um índice do tipo B-Tree, pois ele lida bem com situações em que é necessário utilizar comparações de igualdade e também ordenações.

Foi realizado um teste com a operação ORDER BY:

```
SELECT nome, sobrenome  
FROM funcionario  
ORDER BY nome asc, sobrenome asc
```

- 180 milissegundos sem índice
- 130 milissegundos com índice (redução de 27%)

### Teste para usabilidade das tabelas relacionadas à manutenção

O grupo pediu para o ChatGpt gerar uma consulta complexa que envolvesse as tabelas campus, restaurante universitário, manutenção, funcionário e equipamento:

```
WITH manutencoes_funcionario AS (  
  SELECT  
    f.cpf,  
    COUNT(m.id_equipamento) AS total_manutencoes,  
    AVG(EXTRACT(YEAR FROM AGE(CURRENT_DATE, f.data_contratacao))) AS  
    media_anos_servico  
  FROM funcionario f  
  LEFT JOIN manutencao m ON f.cpf = m.cpf  
  GROUP BY f.cpf  
,  
manutencoes_equipamento AS (  
  SELECT  
    e.id_equipamento,  
    COUNT(m.data) AS total_manutencoes_equipamento  
  FROM equipamento e  
  LEFT JOIN manutencao m ON e.id_equipamento = m.id_equipamento  
  GROUP BY e.id_equipamento  
,  
salario_por_restaurante AS (  
  SELECT  
    id_restaurante,  
    AVG(salario) AS media_salarial  
  FROM funcionario  
  GROUP BY id_restaurante  
,  
gasto_total_por_campus AS (  
  SELECT  
    ru.nome AS nome_campus,
```

```

SUM(e.valor_compra) AS gasto_total_equipamentos
FROM equipamento e
JOIN restaurante_universitario ru ON e.id_restaurante = ru.id_restaurante
GROUP BY ru.nome
)

SELECT
f.nome || ' ' || f.sobrenome AS funcionario,
f.setor,
f.turno,
f.salario,
f.nivel_estudo,
f.data_contratacao,
f.telefone,
mf.total_manutencoes,
mf.media_anos_servico,

e.tipo AS tipo_equipamento,
e.marca,
e.estado AS estado_equipamento,
e.valor_compra,
me.total_manutencoes_equipamento,

ru.nome AS restaurante,
ru.tipo_gestao,
sr.media_salarial,

c.nome AS campus,
c.cidade,
c.estado,
gc.gasto_total_equipamentos,

m.data AS data_manutencao

FROM funcionario f
LEFT JOIN manutencao m ON f.cpf = m.cpf
LEFT JOIN equipamento e ON m.id_equipamento = e.id_equipamento
JOIN restaurante_universitario ru ON f.id_restaurante = ru.id_restaurante
JOIN campus c ON ru.nome = c.nome
LEFT JOIN manutencoes_funcionario mf ON mf.cpf = f.cpf
LEFT JOIN manutencoes_equipamento me ON me.id_equipamento = e.id_equipamento
LEFT JOIN salario_por_restaurante sr ON sr.id_restaurante = ru.id_restaurante
LEFT JOIN gasto_total_por_campus gc ON gc.nome_campus = c.nome

ORDER BY
mf.total_manutencoes DESC,
me.total_manutencoes_equipamento DESC,
f.salario DESC;

```

Essa consulta demorou 370 milissegundos. Para saber onde possíveis índices poderiam ajudar, o grupo solicitou dicas ao ChatGpt, que recomendou o comando EXPLAIN ANALYZE. Através dele, notou-se locais onde a busca por valores estava sendo sequencial, ou seja, onde havia o registro de seq scan na explicação de como o SGBD realizou a



consulta. Porém, mesmo inserindo novos índices nos locais corretos, não foi possível otimizar essa consulta.

Em muitos casos é desejado saber quem foi o funcionário responsável por uma manutenção, e para isso, pode ser desejado adicionar um índice à coluna CPF:

```
CREATE INDEX idx_manutencao_cpf ON manutencao(cpf);
```

A escolha do índice por um índice B-Tree se deve ao fato das operações geralmente realizadas. Geralmente se busca por um funcionário específico ou então, se deseja ordenar por CPF, e nesses casos, esse tipo de índice consegue desempenhar muito bem. Para essa consulta complexa não houveram melhorias, mas para outras consultas que também poderiam ser utilizadas houve:

- Para uma consulta específica, que depende de saber qual é o CPF que realiza cada manutenção:

```
- SELECT cpf, COUNT(*) AS total_manutencoes  
- FROM manutencao  
- GROUP BY cpf  
- ORDER BY total_manutencoes DESC  
- LIMIT 10;
```

- Demora 140 milissegundos sem o índice, 90 milissegundos com o índice (35% de melhoria)

Em muitos casos estamos interessados em saber quais são os equipamentos de um determinado restaurante para levantar uma análise, por isso, pode ser interessante criar um índice para isso:

```
CREATE INDEX idx_equipamento_id_restaurante ON equipamento(id_restaurante);
```

Como uma comparação do tipo igualdade geralmente é utilizada nesses casos, o índice B-Tree é o mais indicado para a situação.

A seguinte consulta demorou 115 milissegundos com o índice, e sem o índice 150 milissegundos (redução de 23%):

Para realizar o teste do novo índice, a seguinte consulta foi realizada:

```
SELECT e.*  
FROM equipamento e  
JOIN restaurante_universitario ru ON e.id_restaurante = ru.id_restaurante  
WHERE ru.nome = 'de_Engenharia';
```

Sem o índice, a consulta levou um tempo de 150 milissegundos. Porém, quando há a utilização do novo índice, a consulta passa a levar 115 milissegundos (ou seja, há uma redução de 23%).

E, por fim, mas não menos importante, pode ser útil saber o restaurante que um funcionário trabalha em várias consultas, por isso, torna-se interessante ter um índice na tabela de funcionário na coluna de id\_restaurante:

```
CREATE INDEX idx_funcionario_id_restaurante ON funcionario (id_restaurante)
```

O índice foi criado como B-Tree devido a natureza da utilização da coluna, geralmente essa coluna é utilizada para operações de igualdade ou de agrupamento, e para esses casos o índice escolhido atende perfeitamente.

A consulta utilizada para teste do novo índice foi a seguinte:

```
SELECT id_restaurante, COUNT(*) AS total_funcionarios  
FROM funcionario  
GROUP BY id_restaurante;
```

Sem a utilização do índice, essa consulta levava 140 milissegundos. Agora, devido à existência do índice, ela leva 120 milissegundos (ou seja, ocorreu uma melhoria de 14%).