Otimizações no BD

Gabriela Alcaide - 14746492
Gustavo Pompermayer Fulanetti Silva - 14760280
Kauê Patrick de Oliveira - 14586261
Pedro Henrique Resnitzky Barbedo - 14657691
Rodrigo Gonçalves Cardoso - 14658330

ÍNDICES

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 limparam 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%).

VIEWS MATERIALIZADAS

Views materializadas geram ganho de desempenho por evitar que o sistema tenha que fazer joins todas as vezes, por exemplo.

O join é feito uma única vez, e seu resultado é saldo e acessado na view materializada.

Caso alguma das tabelas envolvidas seja constantemente atualizada, perde-se desempenho, ao invés de otimizar, dado que a própria view teria que ser recriada várias vezes.

Nesse sentido, notamos que em nosso trabalho haveria ganho apenas quando aplicado às tabelas Campus e Restaurante Universitários.

Pode ser uma consulta frequente no sistema checar quais são os restaurantes de cada campus e, ao mesmo tempo, estas informações não são atualizadas sempre.

A criação foi feita da seguinte forma:

```
CREATE MATERIALIZED VIEW Campus_Restaurantes AS SELECT
campus.nome as Campus,
restaurante_universitario.id_restaurante as Restaurante
from campus
left join restaurante_universitario
on restaurante_universitario.nome = campus.nome
WITH DATA;
```

Foram, de fato, observados ganhos.

Fazendo a consulta nas tabelas em si, o tempo médio gasto foi de 120ms. Fazendo um SELECT na View Materializada, foi de 90ms. Portanto, houve um ganho de 25% em tempo médio.

PARTICIONAMENTOS

Notamos que o particionamento de tabelas gera ganho expressivo de desempenho, porém também crescimento do tamanho em bytes da tabela. Optamos, portanto, em focar nos casos que trariam maiores ganhos -> as maiores tabelas.

TABELA “LIMPEZA”

Pode ser uma consulta frequente em nosso sistema a busca por quais limpezas foram realizadas em dado período. Portanto, optamos pelo particionamento considerando as colunas de mês e ano.

Caso o usuário do sistema queira observar quais limpezas foram feitas em março de 2025, por exemplo, o mecanismo olhará diretamente para esta partição, e a retornará.

```
-- Criando tabela com partição.
create table LIMPEZA_2 (
 Id_ambiente varchar(5) not null,
 Data date not null,
 CPF varchar(14) not null,
 constraint ID_LIMPEZA primary key (Id_ambiente, Data)
) PARTITION BY RANGE (data);

-- Criando as partições especificamente.
DO $$
DECLARE
 start_date DATE := DATE '2018-01-01';
 end_date DATE := DATE '2025-02-28'; -- até o último mês desejado (maio/2025)
 current_start DATE;
 current_end DATE;
 partition_name TEXT;
BEGIN
 current_start := start_date;
 WHILE current_start < end_date LOOP
 current_end := (current_start + INTERVAL '1 month')::DATE;
 partition_name := format(
 'limpeza_%s_%s',
 EXTRACT(YEAR FROM current_start),
 LPAD(EXTRACT(MONTH FROM current_start)::TEXT, 2, '0')
 );
 EXECUTE format(
 'CREATE TABLE %I PARTITION OF limpeza_2 FOR VALUES FROM (%L)
TO (%L);',
 partition_name, current_start, current_end
 );
 current_start := current_end;
 END LOOP;
END $$;

-- Transferindo os dados para a nova tabela.
insert into limpeza_2
select * from limpeza;

-- Criando índice na nova tabela (igual ao que havia na anterior).
CREATE INDEX idx_cpf_limpeza_2 ON limpeza_2 (cpf);
```

```
-- Recriando chaves estrangeiras com a nova tabela (FKs que a tabela a ser apagada possui e FKs que apontam para a tabela a ser apagada).
alter table LIMPEZA_2 add constraint REF_LIMPE_FUNCI_FK
 foreign key (CPF)
 references FUNCIONARIO;

alter table LIMPEZA_2 add constraint REF_LIMPE_AMBIE
 foreign key (Id_ambiente)
 references AMBIENTE;

-- Excluir a tabela anterior, que foi substituída por sua versão particionada.
drop table limpeza;

-- Renomear a nova tabela.
ALTER TABLE limpeza_2 RENAME TO limpeza;
```

Observação dos ganhos:

Melhoria de 40,3% no desempenho para a seguinte consulta:

Com a tabela anterior	Com a tabela nova
236ms	141ms
select * from limpeza where data between '2024-01-01' and '2024-01-31'	select * from limpeza_2 where data between '2024-01-01' and '2024-01-31'

Melhoria de 46,5% no desempenho para a seguinte consulta:

Com a tabela anterior	Com a tabela atual
258ms	138ms
select * from limpeza where data = '2024-01-18'	select * from limpeza_2 where data = '2024-01-18'

TABELA “OFERECE”

Pode ser uma consulta frequente em nosso sistema a busca por quais refeições foram oferecidas por um restaurante em dado período. Portanto, optamos pelo particionamento considerando as colunas de mês e ano.

Caso o usuário do sistema queira observar quais refeições foram oferecidas em março de 2025, por exemplo, o mecanismo olhará diretamente para esta partição, e a retornará.

```
-- Primeiro passo é renomear a tabela oferece (necessário para que a nova tabela possa ter o nome correto quando criada)
ALTER TABLE oferece RENAME TO oferece_antiga;
```

```
ALTER INDEX idx_data_oferece RENAME TO idx_data_oferece_antigo;

-- criação da nova base de dados com partições por data

CREATE TABLE oferece (
  tipo VARCHAR(20),
  data DATE,
  id_restaurante VARCHAR(4),
  PRIMARY KEY (tipo, data, id_restaurante)
) PARTITION BY RANGE (data);

-- criação do índice que já existia antes
CREATE INDEX idx_data_oferece ON oferece (data);

-- Não é necessário tirar nenhuma chave extraingeira de outra tabela que aponta para ela, uma vez que não existe essa situação

-- criação das chaves estrangeiras da própria tabela
alter table OFERECE add constraint EQU_OFERE_RESTA
foreign key (Id_restaurante)
references RESTAURANTE_UNIVERSITARIO;

alter table OFERECE add constraint EQU_OFERE_REFEI_FK
foreign key (Tipo, Data)
references REFEICAO;

-- criando as partições da base:
CREATE TABLE oferece_2018_01 PARTITION OF oferece FOR VALUES FROM ('2018-01-01') TO ('2018-02-01');
CREATE TABLE oferece_2018_02 PARTITION OF oferece FOR VALUES FROM ('2018-02-01') TO ('2018-03-01');
CREATE TABLE oferece_2018_03 PARTITION OF oferece FOR VALUES FROM ('2018-03-01') TO ('2018-04-01');
CREATE TABLE oferece_2018_04 PARTITION OF oferece FOR VALUES FROM ('2018-04-01') TO ('2018-05-01');
CREATE TABLE oferece_2018_05 PARTITION OF oferece FOR VALUES FROM ('2018-05-01') TO ('2018-06-01');
CREATE TABLE oferece_2018_06 PARTITION OF oferece FOR VALUES FROM ('2018-06-01') TO ('2018-07-01');
CREATE TABLE oferece_2018_07 PARTITION OF oferece FOR VALUES FROM ('2018-07-01') TO ('2018-08-01');
CREATE TABLE oferece_2018_08 PARTITION OF oferece FOR VALUES FROM ('2018-08-01') TO ('2018-09-01');
CREATE TABLE oferece_2018_09 PARTITION OF oferece FOR VALUES FROM ('2018-09-01') TO ('2018-10-01');
CREATE TABLE oferece_2018_10 PARTITION OF oferece FOR VALUES FROM ('2018-10-01') TO ('2018-11-01');
```

```
CREATE TABLE oferece_2018_11 PARTITION OF oferece FOR VALUES FROM ('2018-11-01') TO ('2018-12-01');
CREATE TABLE oferece_2018_12 PARTITION OF oferece FOR VALUES FROM ('2018-12-01') TO ('2019-01-01');

CREATE TABLE oferece_2019_01 PARTITION OF oferece FOR VALUES FROM ('2019-01-01') TO ('2019-02-01');
CREATE TABLE oferece_2019_02 PARTITION OF oferece FOR VALUES FROM ('2019-02-01') TO ('2019-03-01');
CREATE TABLE oferece_2019_03 PARTITION OF oferece FOR VALUES FROM ('2019-03-01') TO ('2019-04-01');
CREATE TABLE oferece_2019_04 PARTITION OF oferece FOR VALUES FROM ('2019-04-01') TO ('2019-05-01');
CREATE TABLE oferece_2019_05 PARTITION OF oferece FOR VALUES FROM ('2019-05-01') TO ('2019-06-01');
CREATE TABLE oferece_2019_06 PARTITION OF oferece FOR VALUES FROM ('2019-06-01') TO ('2019-07-01');
CREATE TABLE oferece_2019_07 PARTITION OF oferece FOR VALUES FROM ('2019-07-01') TO ('2019-08-01');
CREATE TABLE oferece_2019_08 PARTITION OF oferece FOR VALUES FROM ('2019-08-01') TO ('2019-09-01');
CREATE TABLE oferece_2019_09 PARTITION OF oferece FOR VALUES FROM ('2019-09-01') TO ('2019-10-01');
CREATE TABLE oferece_2019_10 PARTITION OF oferece FOR VALUES FROM ('2019-10-01') TO ('2019-11-01');
CREATE TABLE oferece_2019_11 PARTITION OF oferece FOR VALUES FROM ('2019-11-01') TO ('2019-12-01');
CREATE TABLE oferece_2019_12 PARTITION OF oferece FOR VALUES FROM ('2019-12-01') TO ('2020-01-01');

CREATE TABLE oferece_2020_01 PARTITION OF oferece FOR VALUES FROM ('2020-01-01') TO ('2020-02-01');
CREATE TABLE oferece_2020_02 PARTITION OF oferece FOR VALUES FROM ('2020-02-01') TO ('2020-03-01');
CREATE TABLE oferece_2020_03 PARTITION OF oferece FOR VALUES FROM ('2020-03-01') TO ('2020-04-01');
CREATE TABLE oferece_2020_04 PARTITION OF oferece FOR VALUES FROM ('2020-04-01') TO ('2020-05-01');
CREATE TABLE oferece_2020_05 PARTITION OF oferece FOR VALUES FROM ('2020-05-01') TO ('2020-06-01');
CREATE TABLE oferece_2020_06 PARTITION OF oferece FOR VALUES FROM ('2020-06-01') TO ('2020-07-01');
CREATE TABLE oferece_2020_07 PARTITION OF oferece FOR VALUES FROM ('2020-07-01') TO ('2020-08-01');
CREATE TABLE oferece_2020_08 PARTITION OF oferece FOR VALUES FROM ('2020-08-01') TO ('2020-09-01');
CREATE TABLE oferece_2020_09 PARTITION OF oferece FOR VALUES FROM ('2020-09-01') TO ('2020-10-01');
```

```
CREATE TABLE oferece_2020_10 PARTITION OF oferece FOR VALUES FROM ('2020-10-01') TO ('2020-11-01');
CREATE TABLE oferece_2020_11 PARTITION OF oferece FOR VALUES FROM ('2020-11-01') TO ('2020-12-01');
CREATE TABLE oferece_2020_12 PARTITION OF oferece FOR VALUES FROM ('2020-12-01') TO ('2021-01-01');

CREATE TABLE oferece_2021_01 PARTITION OF oferece FOR VALUES FROM ('2021-01-01') TO ('2021-02-01');
CREATE TABLE oferece_2021_02 PARTITION OF oferece FOR VALUES FROM ('2021-02-01') TO ('2021-03-01');
CREATE TABLE oferece_2021_03 PARTITION OF oferece FOR VALUES FROM ('2021-03-01') TO ('2021-04-01');
CREATE TABLE oferece_2021_04 PARTITION OF oferece FOR VALUES FROM ('2021-04-01') TO ('2021-05-01');
CREATE TABLE oferece_2021_05 PARTITION OF oferece FOR VALUES FROM ('2021-05-01') TO ('2021-06-01');
CREATE TABLE oferece_2021_06 PARTITION OF oferece FOR VALUES FROM ('2021-06-01') TO ('2021-07-01');
CREATE TABLE oferece_2021_07 PARTITION OF oferece FOR VALUES FROM ('2021-07-01') TO ('2021-08-01');
CREATE TABLE oferece_2021_08 PARTITION OF oferece FOR VALUES FROM ('2021-08-01') TO ('2021-09-01');
CREATE TABLE oferece_2021_09 PARTITION OF oferece FOR VALUES FROM ('2021-09-01') TO ('2021-10-01');
CREATE TABLE oferece_2021_10 PARTITION OF oferece FOR VALUES FROM ('2021-10-01') TO ('2021-11-01');
CREATE TABLE oferece_2021_11 PARTITION OF oferece FOR VALUES FROM ('2021-11-01') TO ('2021-12-01');
CREATE TABLE oferece_2021_12 PARTITION OF oferece FOR VALUES FROM ('2021-12-01') TO ('2022-01-01');

CREATE TABLE oferece_2022_01 PARTITION OF oferece FOR VALUES FROM ('2022-01-01') TO ('2022-02-01');
CREATE TABLE oferece_2022_02 PARTITION OF oferece FOR VALUES FROM ('2022-02-01') TO ('2022-03-01');
CREATE TABLE oferece_2022_03 PARTITION OF oferece FOR VALUES FROM ('2022-03-01') TO ('2022-04-01');
CREATE TABLE oferece_2022_04 PARTITION OF oferece FOR VALUES FROM ('2022-04-01') TO ('2022-05-01');
CREATE TABLE oferece_2022_05 PARTITION OF oferece FOR VALUES FROM ('2022-05-01') TO ('2022-06-01');
CREATE TABLE oferece_2022_06 PARTITION OF oferece FOR VALUES FROM ('2022-06-01') TO ('2022-07-01');
CREATE TABLE oferece_2022_07 PARTITION OF oferece FOR VALUES FROM ('2022-07-01') TO ('2022-08-01');
CREATE TABLE oferece_2022_08 PARTITION OF oferece FOR VALUES FROM ('2022-08-01') TO ('2022-09-01');
```

```
CREATE TABLE oferece_2022_09 PARTITION OF oferece FOR VALUES FROM ('2022-09-01') TO ('2022-10-01');
CREATE TABLE oferece_2022_10 PARTITION OF oferece FOR VALUES FROM ('2022-10-01') TO ('2022-11-01');
CREATE TABLE oferece_2022_11 PARTITION OF oferece FOR VALUES FROM ('2022-11-01') TO ('2022-12-01');
CREATE TABLE oferece_2022_12 PARTITION OF oferece FOR VALUES FROM ('2022-12-01') TO ('2023-01-01');

CREATE TABLE oferece_2023_01 PARTITION OF oferece FOR VALUES FROM ('2023-01-01') TO ('2023-02-01');
CREATE TABLE oferece_2023_02 PARTITION OF oferece FOR VALUES FROM ('2023-02-01') TO ('2023-03-01');
CREATE TABLE oferece_2023_03 PARTITION OF oferece FOR VALUES FROM ('2023-03-01') TO ('2023-04-01');
CREATE TABLE oferece_2023_04 PARTITION OF oferece FOR VALUES FROM ('2023-04-01') TO ('2023-05-01');
CREATE TABLE oferece_2023_05 PARTITION OF oferece FOR VALUES FROM ('2023-05-01') TO ('2023-06-01');
CREATE TABLE oferece_2023_06 PARTITION OF oferece FOR VALUES FROM ('2023-06-01') TO ('2023-07-01');
CREATE TABLE oferece_2023_07 PARTITION OF oferece FOR VALUES FROM ('2023-07-01') TO ('2023-08-01');
CREATE TABLE oferece_2023_08 PARTITION OF oferece FOR VALUES FROM ('2023-08-01') TO ('2023-09-01');
CREATE TABLE oferece_2023_09 PARTITION OF oferece FOR VALUES FROM ('2023-09-01') TO ('2023-10-01');
CREATE TABLE oferece_2023_10 PARTITION OF oferece FOR VALUES FROM ('2023-10-01') TO ('2023-11-01');
CREATE TABLE oferece_2023_11 PARTITION OF oferece FOR VALUES FROM ('2023-11-01') TO ('2023-12-01');
CREATE TABLE oferece_2023_12 PARTITION OF oferece FOR VALUES FROM ('2023-12-01') TO ('2024-01-01');

CREATE TABLE oferece_2024_01 PARTITION OF oferece FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
CREATE TABLE oferece_2024_02 PARTITION OF oferece FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');
CREATE TABLE oferece_2024_03 PARTITION OF oferece FOR VALUES FROM ('2024-03-01') TO ('2024-04-01');
CREATE TABLE oferece_2024_04 PARTITION OF oferece FOR VALUES FROM ('2024-04-01') TO ('2024-05-01');
CREATE TABLE oferece_2024_05 PARTITION OF oferece FOR VALUES FROM ('2024-05-01') TO ('2024-06-01');
CREATE TABLE oferece_2024_06 PARTITION OF oferece FOR VALUES FROM ('2024-06-01') TO ('2024-07-01');
CREATE TABLE oferece_2024_07 PARTITION OF oferece FOR VALUES FROM ('2024-07-01') TO ('2024-08-01');
```

```
CREATE TABLE oferece_2024_08 PARTITION OF oferece FOR VALUES FROM ('2024-08-01') TO ('2024-09-01');
CREATE TABLE oferece_2024_09 PARTITION OF oferece FOR VALUES FROM ('2024-09-01') TO ('2024-10-01');
CREATE TABLE oferece_2024_10 PARTITION OF oferece FOR VALUES FROM ('2024-10-01') TO ('2024-11-01');
CREATE TABLE oferece_2024_11 PARTITION OF oferece FOR VALUES FROM ('2024-11-01') TO ('2024-12-01');
CREATE TABLE oferece_2024_12 PARTITION OF oferece FOR VALUES FROM ('2024-12-01') TO ('2025-01-01');

CREATE TABLE oferece_2025_01 PARTITION OF oferece FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');
CREATE TABLE oferece_2025_02 PARTITION OF oferece FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');

-- inserindo os dados agora na tabela já particionada
INSERT INTO oferece
SELECT * FROM oferece_antiga;

-- dropando a tabela antiga que não será mais utilizada
DROP TABLE oferece_antiga;
```

Observação dos ganhos:

Ganho de 32% em desempenho:

Com a tabela anterior	Com a tabela atual
165ms	112ms
SELECT * FROM oferece WHERE data BETWEEN '2023-06-01' AND '2023-07-01'	SELECT * FROM oferece <nova> WHERE data BETWEEN '2023-06-01' AND '2023-07-01'

Com a tabela anterior	Com a tabela atual
180ms	111ms
SELECT id_restaurante, DATE_TRUNC('month', data) AS mes, tipo, COUNT(*) AS total_refeicoes FROM oferece WHERE data >= DATE '2023-01-01' AND data < DATE '2024-01-01'	SELECT id_restaurante, DATE_TRUNC('month', data) AS mes, tipo, COUNT(*) AS total_refeicoes FROM oferece <nova> WHERE data >= DATE '2023-01-01' AND data < DATE '2024-01-01'

GROUP BY id_restaurante, mes, tipo ORDER BY id_restaurante, mes, tipo;	GROUP BY id_restaurante, mes, tipo ORDER BY id_restaurante, mes, tipo;