

alpha   
<ed/tech>

BANCOS DE  
DADOS



# Banco de dados Relacional

## Análise e Performance



Olá pessoal! Hoje vamos explorar uma forma para que possamos entender, analisar e debugar nossas consultas e operações no Postgres, e o primeiro tópico que lhes apresento é o: **EXPLAIN**.

### EXPLAIN

Se você pensou em algo como “me explique”, não está muito longe do conceito... não seria bom se pudéssemos entender o que está acontecendo por trás das operações que fazemos?

Então pedimos para o Postgres explicar o que está acontecendo:

```
EXPLAIN ANALYZE
SELECT name
FROM employees
WHERE department = 'IT' AND salary >
50000;
```

```
QUERY PLAN
-----
Seq Scan on employees  (cost=0.00..20.50 rows=1 width=64)
(actual time=0.026..0.037 rows=3 loops=1)
  Filter: ((department = 'IT'::text) AND (salary > 50000))
  Rows Removed by Filter: 1
Planning Time: 0.073 ms
Execution Time: 0.065 ms
```

O postgres precisa sempre de um plano de ação, que irá determinar de que forma ele vai encontrar a informação que foi pedida.

Podemos ver “**Seq Scan**” como a primeira informação, referenciando a tabela “employees”, que nos mostra que a pesquisa nesta tabela está sendo feita de forma sequencial, item a item.

O que pode nos dar alguns insights, por exemplo: “Este banco está indexado? Se está, o que levou o banco a escolher este caminho?”.

Vamos indexar a coluna de departamento e tentar novamente? Qual será o resultado?

```
CREATE INDEX idx_department ON employees
(department);

EXPLAIN ANALYZE
SELECT name
FROM employees
WHERE department = 'IT' AND salary > 50000;
```

```
QUERY PLAN
-----
Index Scan using idx_department on employees
(cost=0.14..8.16 rows=2 width=64) (actual
time=0.033..0.036 rows=3 loops=1)
  Index Cond: (department = 'IT'::text)
  Filter: (salary > 50000)
  Rows Removed by Filter: 1
Planning Time: 0.134 ms
Execution Time: 0.075 ms
```

Agora podemos ver o Index Scan trabalhando, agora temos uma ferramenta para compreender o funcionamento interno e testar nossas operações e consultas!

# Banco de dados Relacional

## Análise e Performance



O explain sem dúvidas é uma ferramenta muito valiosa quando a questão é analisar nossas operações, porém ele não resolve o problema por si só.

É necessário que ativamente façamos uma interpretação e busquemos possíveis soluções para o performance das nossas tabelas.

## Normalização

A normalização em banco de dados é um processo fundamental no design de banco de dados relacionais, cujo objetivo é organizar os dados de forma eficiente para minimizar a redundância e evitar anomalias de inserção, atualização e exclusão.

Isso é feito dividindo as tabelas em estruturas menores e mais coesas, reduzindo a duplicação de dados e garantindo a integridade referencial.

Existem várias formas normais (1NF, 2NF, 3NF, BCNF, 4NF, 5NF, etc.), cada uma delas com regras específicas que uma tabela deve atender para ser considerada normalizada. Como por exemplo:

- Primeira Forma Normal (1NF): Uma tabela está na 1NF se ela não tiver grupos repetidos de colunas e cada coluna conter apenas valores atômicos.
- Segunda Forma Normal (2NF): Uma tabela está na 2NF se estiver na 1NF e se todos os seus atributos não-chave forem completamente dependentes da chave primária.
- Terceira Forma Normal (3NF): Uma tabela está na 3NF se estiver na 2NF e se todos os seus atributos não-chave forem transitivamente dependentes da chave primária.
- Forma Normal de Boyce-Codd (BCNF): Uma tabela está na BCNF se estiver na 3NF e se cada dependência funcional não trivial for uma dependência funcional de superchave para chave candidata.
- Quarta Forma Normal (4NF) e Quinta Forma Normal (5NF): Essas formas normais tratam de dependências multivaloradas e dependências de junção, respectivamente.

Normalizar um banco de dados geralmente envolve identificar as dependências funcionais entre os atributos, dividindo tabelas grandes em tabelas menores e mais coesas, e, em alguns casos, introduzindo tabelas de junção para eliminar dependências transitivas.

A normalização é essencial para garantir a integridade dos dados, facilitar a manutenção do banco de dados e otimizar o desempenho das consultas. No entanto, também é importante encontrar um equilíbrio entre a normalização e a desnormalização para atender aos requisitos específicos de desempenho e funcionalidade do sistema.



# Banco de dados Relacional

## Análise e Performance



Mas para conseguir encontrar o equilíbrio precisamos antes falar do que é o processo de desnormalização.

## Desnormalização

A desnormalização é o processo oposto à normalização em banco de dados. Enquanto a normalização busca dividir os dados em tabelas menores e mais coesas para minimizar a redundância e evitar anomalias, a desnormalização envolve a reintrodução de redundância controlada para melhorar o desempenho de consultas e operações de escrita em um banco de dados.

Existem várias razões pelas quais alguém pode optar por desnormalizar um banco de dados:

- Melhorar o desempenho das consultas: Ao armazenar dados redundantes em tabelas desnormalizadas, é possível evitar joins complexos e reduzir o tempo de resposta das consultas.
- Reduzir a complexidade das consultas: Em alguns casos, desnormalizar o banco de dados pode simplificar consultas complexas, tornando o código mais legível e mais fácil de manter.
- Reduzir o número de operações de leitura/gravação: Em sistemas onde há mais operações de leitura do que de gravação, a desnormalização pode ser benéfica, pois reduz a necessidade de joins e agiliza as consultas.
- Suportar relatórios ou análises específicas: Em certas situações, é mais eficiente ter dados pré-agregados ou transformados em uma estrutura desnormalizada para suportar consultas analíticas ou relatórios complexos.
- Reduzir a sobrecarga do servidor: Em sistemas com alto volume de transações ou consultas, a desnormalização pode reduzir a carga sobre o servidor de banco de dados, resultando em melhor desempenho geral do sistema.

É importante notar que a desnormalização também tem suas desvantagens, como aumento do armazenamento necessário, aumento da complexidade da manutenção dos dados devido à redundância e o risco de inconsistência de dados se as atualizações não forem gerenciadas adequadamente. Portanto, a desnormalização deve ser cuidadosamente considerada e implementada apenas quando realmente necessário, equilibrando os benefícios de desempenho com as possíveis consequências negativas.

# Banco de dados Relacional

## Exemplo Prático



Agora podemos voltar ao nosso problema anterior... Como vamos melhorar nossas consultas através da desnormalização? Vejamos um exemplo a seguir:

### Tabelas Normalizadas

Vamos pressupor que temos a mesma biblioteca da aula passada.

```
CREATE EXTENSION IF NOT EXISTS "uuid-oss";

CREATE TABLE IF NOT EXISTS alunos(
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  nome VARCHAR(50),
  cpf VARCHAR(11),
  ra VARCHAR(20)
);

CREATE INDEX IF NOT EXISTS idx_nome_alunos ON alunos(nome);
CREATE INDEX IF NOT EXISTS idx_ra_alunos ON alunos(ra);

CREATE TABLE IF NOT EXISTS livros(
  id SERIAL PRIMARY KEY,
  nome VARCHAR(50),
  categoria VARCHAR(100),
  resumo TEXT,
);

CREATE INDEX IF NOT EXISTS idx_nome_livros ON livros(nome);

CREATE TABLE IF NOT EXISTS emprestimos(
  id SERIAL PRIMARY KEY,
  aluno_id UUID FOREIGN KEY REFERENCES alunos(id),
  livro_id INTEGER FOREIGN KEY REFERENCES livros(id),
  data_retirada DATE,
  data_prazo DATE,
  data_entrega DATE
);

CREATE INDEX IF NOT EXISTS idx_aluno_emprestimos ON emprestimos(aluno_id);
CREATE INDEX IF NOT EXISTS idx_livro_emprestimos ON emprestimos(livro_id);
CREATE INDEX IF NOT EXISTS idx_atraso_emprestimos
ON livro_aluno(data_entrega) WHERE data_entrega IS NULL;
CREATE INDEX IF NOT EXISTS idx_entrega_emprestimos
ON livro_aluno(data_prazo) WHERE data_entrega IS NOT NULL;
```

# Banco de dados Relacional

## Exemplo Prático



Vamos Popular nosso banco?

### Inserindo Livros

Vamos fazer um multi-insert, inserindo várias linhas na nossa tabela de uma única vez.

```
INSERT INTO livros ("nome", "resumo", "categoria")
VALUES
('Dom Quixote', 'Este clássico da literatura espanhola segue as aventuras de um nobre sonhador
que se torna cavaleiro errante, buscando justiça e idealismo em um mundo que muitas vezes não
compartilha suas visões.', 'Romance de Aventura'),
('1984', 'George Orwell descreve um mundo distópico governado por um regime totalitário, onde
a manipulação da verdade e a vigilância constante são ferramentas de controle sobre a
população.', 'Ficção Distópica'),
('Cem Anos de Solidão', 'Gabriel García Márquez tece uma saga multigeracional que acompanha a
família Buendía e a cidade fictícia de Macondo, explorando temas de amor, solidão, magia e
decadência.', 'Realismo Mágico'),
('O Senhor dos Anéis', 'J.R.R. Tolkien apresenta um mundo de fantasia épica, onde elfos,
anões, hobbits e humanos unem forças em uma jornada para destruir um anel maligno e derrotar o
Senhor das Trevas, Sauron.', 'Fantasia Épica'),
('A Origem das Espécies', 'Charles Darwin propõe sua teoria revolucionária da evolução por
meio da seleção natural, lançando as bases para a compreensão moderna da diversidade e da
origem da vida na Terra.', 'Ciências Naturais'),
('O Pequeno Príncipe', 'Antoine de Saint-Exupéry conta a história de um jovem príncipe que
viaja por diferentes planetas, aprendendo lições sobre amor, amizade e humanidade.',
'Literatura Infantil e Juvenil'),
('Crime e Castigo', 'Neste romance psicológico, Fiódor Dostoiévski mergulha na mente
atormetada de Raskólnikov, um ex-estudante que comete um assassinato e enfrenta as
consequências morais e emocionais de seus atos.', 'Ficção Psicológica'),
('A Revolução dos Bichos', 'George Orwell utiliza alegoria para satirizar regimes
totalitários, retratando uma fazenda onde os animais se rebelam contra seus donos humanos,
apenas para descobrir que a tirania pode surgir mesmo entre os supostos libertadores.',
'Fábula Política'),
('O Nome do Vento', 'Patrick Rothfuss narra a história de Kvothe, um prodígio mágico e músico,
enquanto ele relata sua vida em uma pousada remota, revelando seus feitos lendários e os
segredos que o assombram.', 'Fantasia'),
('A Arte da Guerra', 'Sun Tzu oferece conselhos intemporais sobre estratégia militar,
explorando a arte de vencer batalhas e conquistar objetivos com sabedoria e tática.',
'Estratégia e Liderança');
```

### Inserindo Usuários

E vamos fazer o mesmo em usuários, inserindo várias linhas na nossa tabela de uma única vez.

```
INSERT INTO alunos("id", "nome", "cpf", "ra")
VALUES
('7e08f23a-ab5f-4a4e-aa49-0e27f331dba4', 'João Silva', '123.456.789-00', '2021001'),
('98f6354f-30bb-4b7a-87df-856002c5c498', 'Maria Oliveira', '987.654.321-00', '2021002'),
('d5de5e50-b84e-4ff0-aa5a-6e15d0a754dc', 'Pedro Santos', '456.789.123-00', '2021003'),
('da887c26-fedf-4c04-af73-58dd9d595632', 'Ana Souza', '789.123.456-00', '2021004'),
('eeaa1fc5-1563-4525-bbdd-3d5d3280a3f0', 'Lucas Pereira', '321.654.987-00', '2021005');
```

# Banco de dados Relacional

## Exemplo Prático



### Inserindo Empréstimos

Por fim, vamos criar os empréstimos fictícios de livros reais pelos nossos usuários fictícios.

```
INSERT INTO empréstimos (aluno id, livro id, data retirada, data prazo, data entrega) VALUES
('7e08f23a-ab5f-4a4e-aa49-0e27f331dba4', 4, '2024-02-15', '2024-03-15', NULL),
('98f6354f-30bb-4b7a-87df-856002c5c498', 7, '2024-02-17', '2024-03-17', '2024-03-15'),
('d5de5e50-b84e-4ff0-aa5a-6e15d0a754dc', 9, '2024-02-20', '2024-03-20', NULL),
('da887c26-fedf-4c04-af73-58dd9d595632', 2, '2024-02-22', '2024-03-22', NULL),
('eeaa1fc5-1563-4525-bbdd-3d5d3280a3f0', 6, '2024-02-25', '2024-03-25', '2024-03-23'),
('7e08f23a-ab5f-4a4e-aa49-0e27f331dba4', 8, '2024-02-26', '2024-03-26', NULL),
('98f6354f-30bb-4b7a-87df-856002c5c498', 1, '2024-02-28', '2024-03-28', '2024-03-27'),
('d5de5e50-b84e-4ff0-aa5a-6e15d0a754dc', 5, '2024-03-01', '2024-04-01', NULL),
('da887c26-fedf-4c04-af73-58dd9d595632', 10, '2024-03-03', '2024-04-03', NULL),
('eeaa1fc5-1563-4525-bbdd-3d5d3280a3f0', 3, '2024-03-05', '2024-04-05', '2024-04-04'),
('7e08f23a-ab5f-4a4e-aa49-0e27f331dba4', 4, '2024-03-07', '2024-04-07', NULL),
('98f6354f-30bb-4b7a-87df-856002c5c498', 8, '2024-03-09', '2024-04-09', NULL),
('d5de5e50-b84e-4ff0-aa5a-6e15d0a754dc', 9, '2024-03-11', '2024-04-11', '2024-04-10'),
('da887c26-fedf-4c04-af73-58dd9d595632', 3, '2024-03-13', '2024-04-13', NULL),
('eeaa1fc5-1563-4525-bbdd-3d5d3280a3f0', 1, '2024-03-15', '2024-04-15', '2024-04-14'),
('7e08f23a-ab5f-4a4e-aa49-0e27f331dba4', 5, '2024-03-17', '2024-04-17', NULL),
('98f6354f-30bb-4b7a-87df-856002c5c498', 10, '2024-03-19', '2024-04-19', '2024-04-18'),
('d5de5e50-b84e-4ff0-aa5a-6e15d0a754dc', 2, '2024-03-21', '2024-04-21', NULL),
('da887c26-fedf-4c04-af73-58dd9d595632', 6, '2024-03-23', '2024-04-23', '2024-04-22'),
('eeaa1fc5-1563-4525-bbdd-3d5d3280a3f0', 7, '2024-03-25', '2024-04-25', NULL);
```


Ok, configuração pronta... qual nosso próximo passo? Vamos testar uma consulta usando o comando **EXPLAIN ANALYZE**!

```
EXPLAIN ANALYZE
SELECT * FROM
alunos
WHERE nome =
'João';
```

```
EXPLAIN ANALYZE
SELECT * FROM livros
WHERE nome = 'O Senhor dos
Anéis';
```

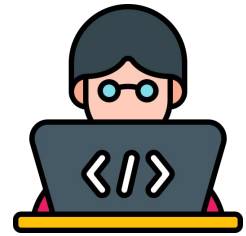
```
EXPLAIN ANALYZE
SELECT * FROM
empréstimos
WHERE livro_id =
1;
```

Vamos executar o primeiro, fique a vontade para também testar os outros:

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on alunos (cost=4.16..9.50 rows=2 width=232) (actual time=0.015..0.015 rows=0 loops=1)	
2	Recheck Cond: ((nome)::text = 'João':text)	
3	-> Bitmap Index Scan on idx_nome_alunos (cost=0.00..4.16 rows=2 width=0) (actual time=0.013..0.014 rows=0 loop...	
4	Index Cond: ((nome)::text = 'João':text)	
5	Planning Time: 0.053 ms	
6	Execution Time: 0.031 ms	

# Banco de dados Relacional

## Exemplo Prático



Mas agora vamos pensar num cenário mais realista, onde queremos receber a junção entre as tabelas:

```
SELECT
    empréstimos.id AS emprestimo_id,
    alunos.nome AS aluno_nome,
    alunos.ra AS aluno_ra,
    livros.nome AS livro_nome,
    empréstimos.data_retirada,
    empréstimos.data_prazo,
    empréstimos.data_entrega
FROM
    empréstimos
INNER JOIN
    alunos ON empréstimos.aluno_id = alunos.id
INNER JOIN
    livros ON empréstimos.livro_id = livros.id;
```

Aqui devemos receber uma lista de empréstimos, e para cada elemento queremos do:

- aluno: o nome e o RA;
- livro: o nome;

Como estamos usando as colunas de chaves primárias como chaves estrangeiras da nossa tabela de empréstimos, praticamente só temos consultas indexadas e com isso ganhamos bastante performance nas operações!

Mas, caso quiséssemos uma busca ainda mais rápida, poderíamos pular a pesquisa do index?

Sim! Através da desnormalização podemos adicionar as informações que queremos coletar "aluno\_nome", "aluno\_ra" e "livro\_nome" diretamente na nossa tabela de empréstimos, evitando assim um JOIN na nossa consulta.

A busca vai ser mais rápida, porém com o custo de repetição e integridade dos dados... alterar a coluna nome para o aluno pode acarretar em divergência das informações entre as tabelas alunos e empréstimos.

Mas será que podemos ter o melhor dos dois mundos?

Em alguns casos sim, vejamos as: **VIEWS** e **MATERIALIZED VIEWS**



# Banco de dados Relacional

## VIEWS e MATERIALIZED VIEWS



Uma VIEW em um banco de dados é uma consulta armazenada que é acessível como uma tabela virtual. Ela é útil quando você tem consultas complexas ou frequentes que deseja reutilizar sem ter que escrever a consulta inteira toda vez. As VIEWS são geralmente usadas para simplificar consultas complexas, garantir consistência nos resultados e ocultar detalhes de implementação subjacentes.

Por outro lado, uma MATERIALIZED VIEW é semelhante a uma VIEW normal, mas os resultados da consulta são armazenados fisicamente em disco como uma tabela real, em vez de serem recalculados toda vez que a MATERIALIZED VIEW é consultada. Isso pode ser útil em situações onde os dados subjacentes mudam com menos frequência do que as consultas são executadas, ou quando o desempenho é uma preocupação e é aceitável ter dados ligeiramente desatualizados.

Vamos ver exemplos de ambas em um cenário hipotético onde temos uma tabela de vendas e uma tabela de produtos:

```
-- Exemplo de VIEW
CREATE VIEW sales_view AS
SELECT sales.*, products.name AS product_name
FROM sales
JOIN products ON sales.product_id = products.id;

-- Exemplo de MATERIALIZED VIEW
CREATE MATERIALIZED VIEW materialized sales view AS
SELECT sales.*, products.name AS product_name
FROM sales
JOIN products ON sales.product_id = products.id;
```

Ambos os comandos acima criam uma visão que mostra as vendas junto com os nomes dos produtos vendidos. No entanto, a VIEW é uma consulta que é executada toda vez que é acessada, enquanto a MATERIALIZED VIEW é uma cópia física dos dados que é atualizada apenas quando especificamente solicitada ou programada para atualização automática.

Todas as ferramentas vistas hoje, em conjunto com o que já vimos, são ferramentas diferentes para melhorar a performance da nossa aplicação.

Porém, é importante estar atento porque estas ferramentas devem ser usadas com sabedoria para extrair a melhor performance das suas consultas!