

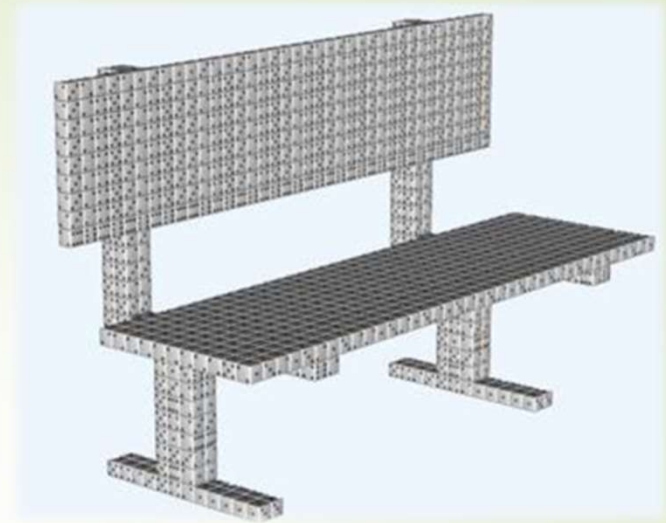
Banco de Dados II

- Introdução à Otimização e Indexação de BD

Prof. Angelo Augusto Frozza, Dr.
<http://about.me/TilFrozza>



INSTITUTO FEDERAL
Catarinense
Campus Camboriú



Roteiro



- Otimização
- Indexação

Roteiro



- Otimização
- Indexação

Otimização

➤ Visa aumentar o desempenho do BD.

➤ **Técnicas:**

➤ *Tuning*

➤ Envolve configuração do SO, SGBD ou otimização de consultas.

➤ Otimização de consultas

➤ Envolve analisar e reescrever as consultas que demandam maior desempenho.

Otimização

➤ Otimização de consultas

- Usa **álgebra relacional** para analisar as consultas.
- Dica prática:
 - Reescrever as *queries*, colocando as partes mais restritivas da consulta para que executem antes das demais.
 - Usa ‘(...)’ para destacar precedência.

Otimização



➤ Otimização de consultas

- Tens uma tabela de *Pessoas*, na qual tem uma chave estrangeira *Sexo*, que aponta para outra tabela com apenas 3 registros (*Feminino*, *Masculino*, *Outros*).

➤ Consulta:

- Contar quantas pessoas de cada sexo tem?

```
SELECT sexo.descricao, COUNT(*)  
FROM pessoas  
      JOIN sexo ON pessoas.fk_sexo = sexo.pk  
GROUP BY sexo.descricao;
```

Otimização



► Otimização de consultas

► Consulta clássica:

- Contar quantas pessoas de cada sexo tem?

```
SELECT sexo.descricao, COUNT (*)  
FROM pessoas  
      JOIN sexo ON pessoas.fk_sexo = sexo.pk  
GROUP BY sexo.descricao;
```

- Faz o JOIN de tudo, depois conta e faz o GROUP BY.
 - Se Pessoas = 1 milhão de tuplas, então faz o JOIN de 1 milhão de registros com 3 registros (de sexo).
 - Tempo de execução estimado: 3-4 segundos.

Otimização

➤ Otimização de consultas

➤ Consulta otimizada:

- Contar quantas pessoas de cada sexo tem?

```
SELECT sexo.descricao, p.valor
FROM sexo
      JOIN (SELECT fk_sexo, COUNT(id_pessoa) as valor
            FROM pessoa
            GROUP BY fk_sexo) p
ON sexo.pk = p.fk_sexo
```

- Agora agrupa e conta os `fk_sexo` de Pessoa (vai retornar 3 registros em vez de 1 milhão).
- Depois faz o JOIN com Sexo (JOIN de 3 com 3).
- Tempo de execução estimado: 200 ms.

Roteiro



- Otimização
- Indexação

Indexação

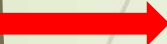
➤ Índices

- São estruturas de dados otimizadas que permitem a localização de um determinado dado/tupla mais rapidamente.
- A busca em uma tabela se dá de maneira linear $O(n)$.
 - Índices implementam algoritmos que tornam essa busca mais rápida ($< O(n)$).

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:




codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:



codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➡ Índices

➡ Busque as informações do médico com código = 5:

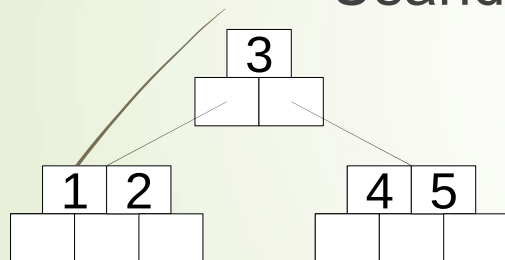
codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

➤ Usando um **índice binário**



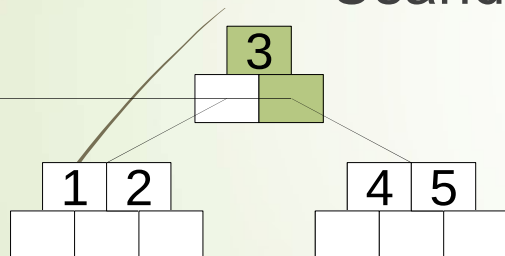
codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatologia	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinville
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

➤ Usando um **índice binário**



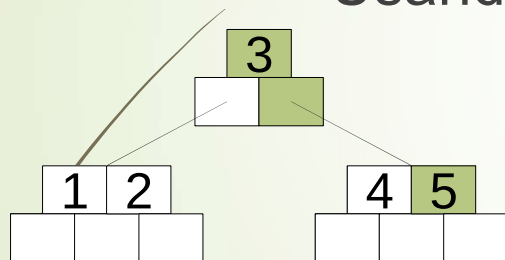
codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatolog	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinvile
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

➤ Usando um **índice binário**



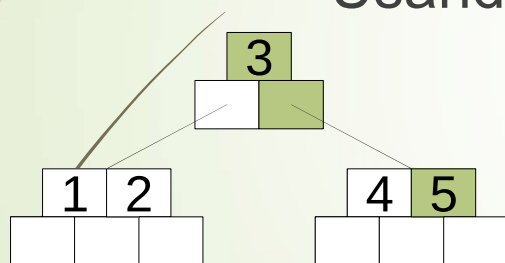
codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatologia	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinville
5	3	Marcia	33	neurologia	11000111000	Biguacu

Indexação

➤ Índices

➤ Busque as informações do médico com código = 5:

➤ Usando um **índice binário**



codm integer	nroa integer	nome character varying(40)	idade smallint	especialidade character(20)	cpf numeric(11,0)	cidade character varying(30)
1	1	Joao	40	ortopedia	10000100000	Florianopolis
2	2	Maria	42	traumatologia	10000110000	Blumenau
3	2	Pedro	51	pediatria	11000100000	Sao Jose
4		Carlos	28	ortopedia	11000110000	Joinville
5	3	Marcia	33	neurologia	11000111000	Biguacu

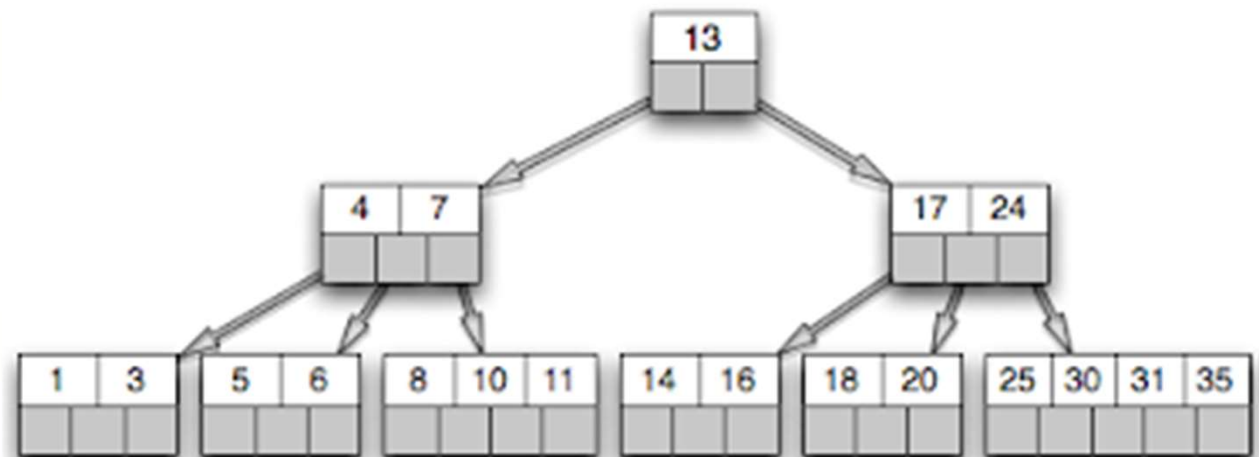
Indexação

➤ Índices

➤ Em um cenário maior...

➤ Busque as informações do médico com código = 31:

➤ Usando um **índice**



Indexação

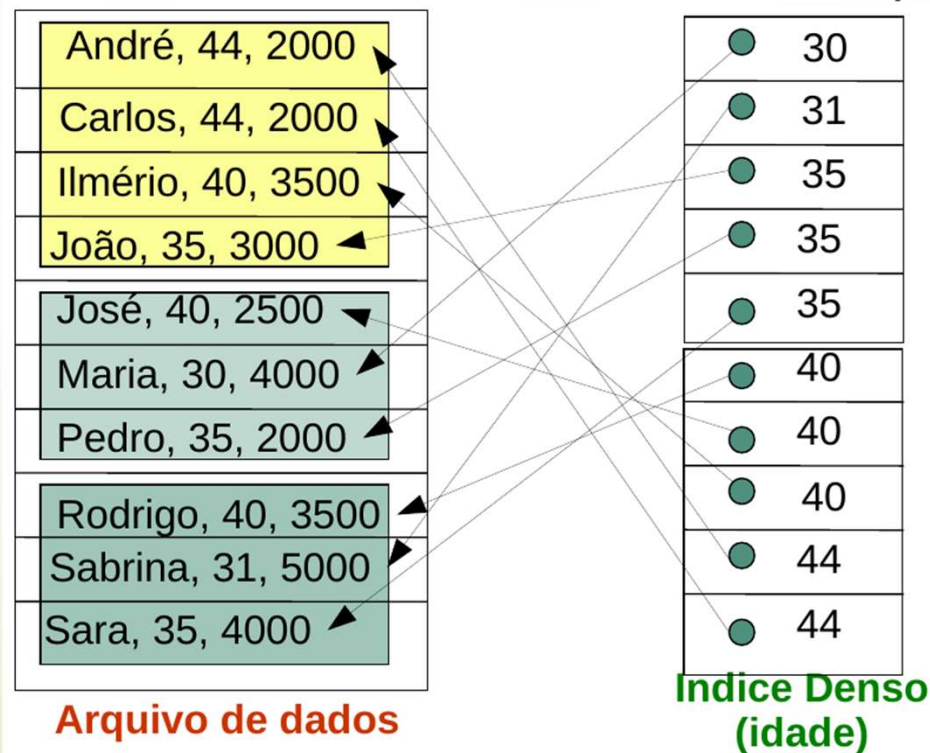
➤ Índices

- Para criar um índice devem ser considerados alguns aspectos:
 - **Chave que irá compor:**
 - Chave primária, de ordenação ou mesmo chave de pesquisa;
 - É criado um novo documento apenas contendo os dados da chave e o ponteiro do dado
 - local pode ser a localização exata da *tupla* ou pode ser o *cluster* ou o bloco
 - Os índices podem ser **densos** ou **esparsos**

Indexação

Índices

- *Índices densos* possuem entrada para todas as chaves de dados:



Indexação

➤ Índices

- *Índices densos* possuem entrada para todas as chaves de dados:
 - **Vantagens** (além de otimizar a consulta):
 - O número de blocos para armazenar o índice é, geralmente, menor do que para armazenar os dados;
 - Pode-se utilizar a busca binária para buscar um registro;
 - O índice pode caber na memória principal (*buffer*), diminuindo o número de I/Os em uma busca.

Índices

- *Índices esparsos* possuem chaves que apontam para blocos de dados:



Indexação

➤ Índices

- *Índices esparsos* possuem chaves que apontam para blocos de dados:
 - **Características:**
 - São menores que os índices densos;
 - Faz um *scan* no bloco de dados;
 - A busca é feita no índice esparsos *valor* \geq *key*
 - O arquivo de dados deve estar ordenado pela chave de busca.

Indexação

Tipos de índices

– Primário

- A chave do índice é composta pela *chave primária* da tabela.
 - A maioria dos SGBD cria índices primários automaticamente.
- Não permitem duplicatas.
- Podem ser agrupados.

– Secundário

- Outras colunas da tabela participam.
- Permitem duplicatas.

Indexação

Tipos de índices

- **Simples**
 - Envolve apenas uma coluna.
- **Composto**
 - Envolve várias colunas.
- **Algoritmos** (depende de cada SGBD)
 - *B-Tree*
 - *R-Tree*
 - *Gist*
 - *Hash*

Indexação

Tipos de índices

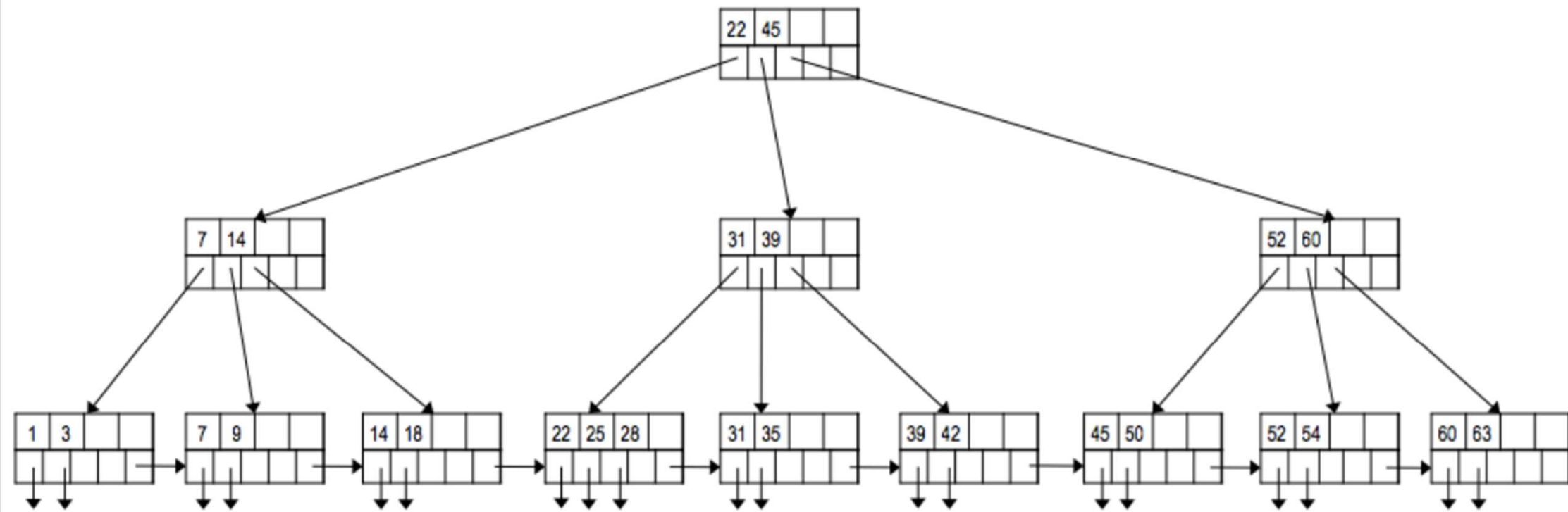
- **Simples**
 - Envolve apenas uma coluna.
- **Composto**
 - Envolve várias colunas.
- **Algoritmos** (depende de cada SGBD)
 - *B-Tree*
 - *R-Tree*
 - *Gist*
 - *Hash*

Indexação

➡ *B-Tree*

- É o índice padrão para a maioria dos BDs
- Utilizado para operações com qualquer operador matemático: =, <>, >, <, >=, <=
- A *B-Tree* implementada pelo *PostgreSQL* apenas armazena os ponteiros nos nodos folha.
- As folhas armazenam os dados e são mantidas como uma lista encadeada.

➔ B-Tree



➡ *B-Tree*

- O tamanho da árvore dita o número mínimo de operações.
- Qual a altura máxima de uma árvore *B* com *m* chaves?
 - Sendo *N* o maior número de chaves na árvore:

$$h = \log_{\frac{m}{2}} \left(\frac{N+1}{2} \right)$$

Indexação

➡ *B-Tree*

- Mas como funciona isso?

- *Let's play*

<https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

Indexação

Tipos de índices

- **Simples**
 - Envolve apenas uma coluna.
- **Composto**
 - Envolve várias colunas.
- **Algoritmos** (depende de cada SGBD)
 - *B-Tree*
 - *R-Tree*
 - *Gist*
 - *Hash*

Indexação

➡ Hash

- Possui um desempenho elevado.
 - A busca usualmente é $O(1)$.
- Apenas pode ser utilizado em buscas por igualdade (=)
- Já utilizaram algum tipo de *hash*?

Indexação

➡ Hash

- Possui um desempenho elevado.
 - A busca usualmente é $O(1)$.
- Apenas pode ser utilizado em buscas por igualdade (=)
- Já utilizaram algum tipo de *hash*?
 - Uma tabela de dispersão ou tabela de *hash* (***hash table***) é um **vetor** em que cada uma de suas posições armazena zero, uma ou mais chaves (e valores associados).

➡ Hash

– Tabela de *hash* ou *bucket*

- Uma tabela de *hash* armazena os dados em *buckets*.
- *Bucket* é considerado uma unidade de armazenamento.
- Normalmente armazena um bloco de disco completo, que por sua vez pode armazenar um ou mais registros.



– Função de *hash* ou função de dispersão

- A função *hash* mapeia o conjunto de chaves de pesquisa para o endereço no qual os registros reais são armazenados.
- De maneira geral é uma chave de acesso para cada *bucket*.

Índices no *PostgreSQL*

- Como criar um índice

```
CREATE [UNIQUE] INDEX nome ON tabela (atr1, atrn)
```

- Exemplo:

```
CREATE INDEX esp_med ON medicos (especialidade)
```

- Excluir índice

```
DROP INDEX nome;
```

Índices no PostgreSQL

- Como criar um índice

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ]  
nome_indice  
  
ON tabela [ USING method ] ( {column|(  
expression ) }  
  
[ ASC | DESC ] [ NULLS { FIRST | LAST } ] [,  
... ] )
```

- Exemplo:

```
CREATE INDEX esp_med ON medicos USING hash  
(especialidade)
```

```
CREATE INDEX nro_med ON medicos USING btree (nroa  
ASC NULLS FIRST)
```

Contato



➡ Prof. Angelo Augusto Frozza, Dr.



angelo.frozza@ifc.edu.br

<http://www.ifc-camboriu.edu.br/~frozza>



@TilFrozza

<http://www.twitter.com/TilFrozza>

<http://about.me/TilFrozza>