

# Banco de Dados – IMD0401

## Aula 24 – Indexação em Banco de Dados

João Carlos Xavier Júnior

[jcxavier@imd.ufrn.br](mailto:jcxavier@imd.ufrn.br)

# Índices em Banco de Dados

## □ Índices:

- ❖ Índices são estruturas associadas aos bancos de dados com a capacidade de **localizar diretamente** um ou mais registros através de uma "**chave**" de procura.
- ❖ Existem dois tipos de índices:
  - **Ordenados;**
  - **Hash.**
- ❖ A escolha de uma técnica de indexação avalia os seguintes fatores:
  - Tipos de acesso (sequencial, aleatório); e
  - Tempo de resposta (consulta, atualização).

# Índices em Banco de Dados

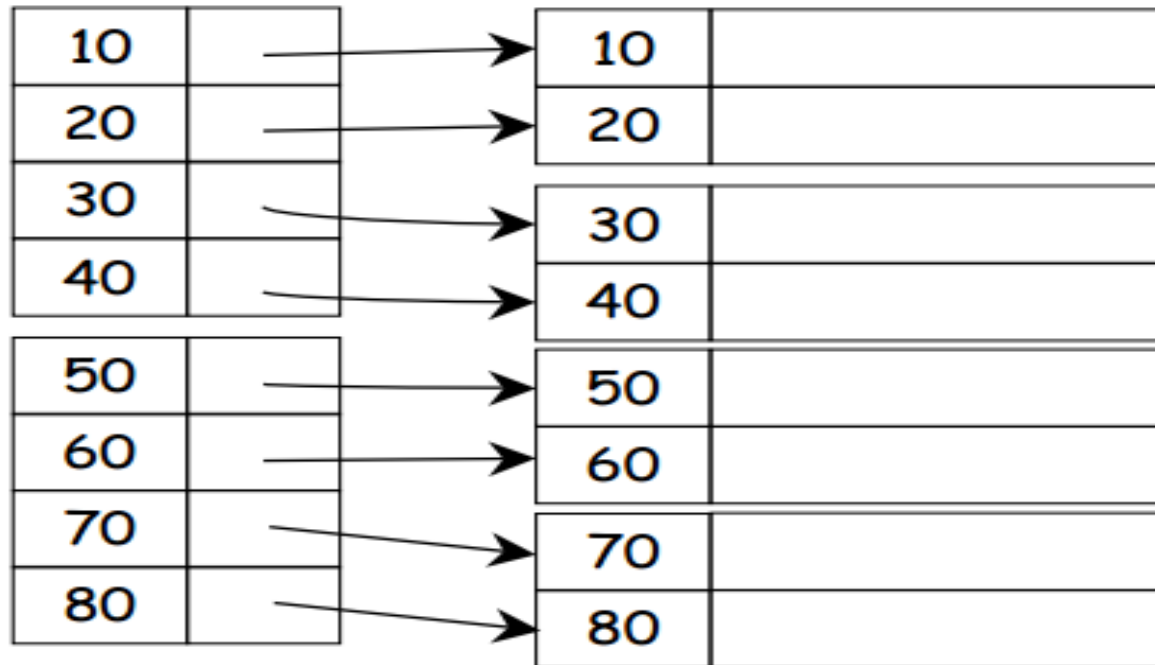
## ❑ Índices Ordenados:

- ❖ Baseiam-se na ordenação de valores.
- ❖ São chamados de **primários** ou **secundários**.
- ❖ Os índices primários utilizam geralmente a **chave primária** como critério de ordenação, desde que ela represente a ordem seqüencial.
- ❖ Os índices secundários utilizam a **chave candidata**.
- ❖ Um índice é **composto pelo valor da chave** e um **ponteiro para a tupla**.

# Índices Ordenados

## ❑ Índices Densos:

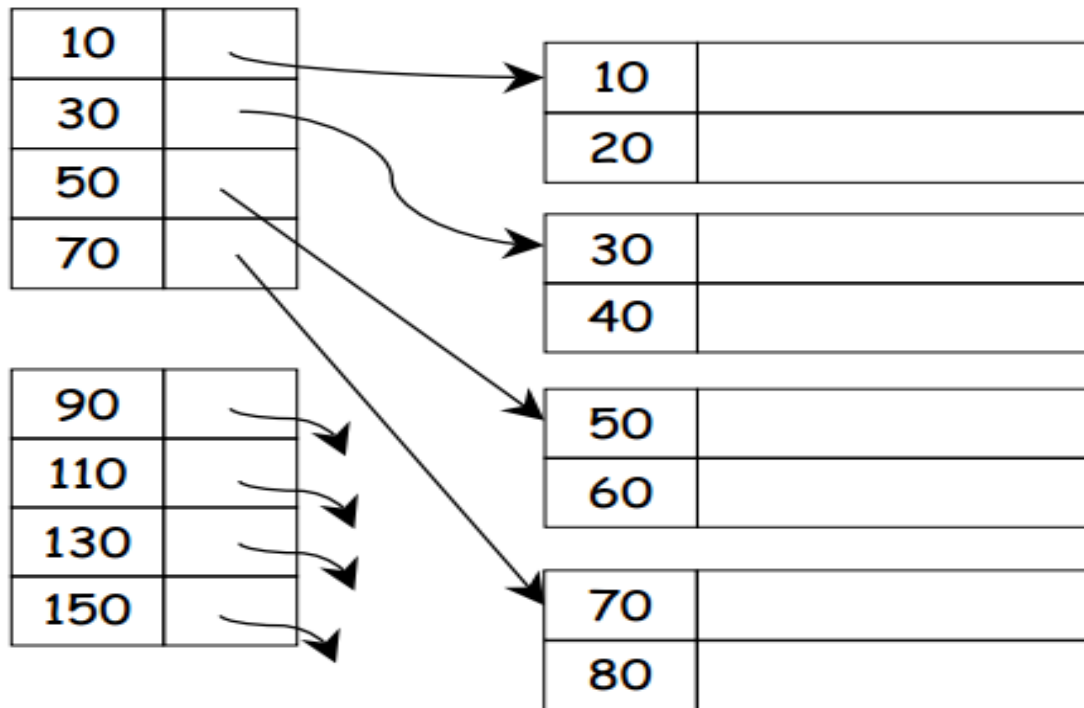
- ❖ Uma entrada no **arquivo de índices** é criada para cada registro no **arquivo de dados**.



# Índices Ordenados

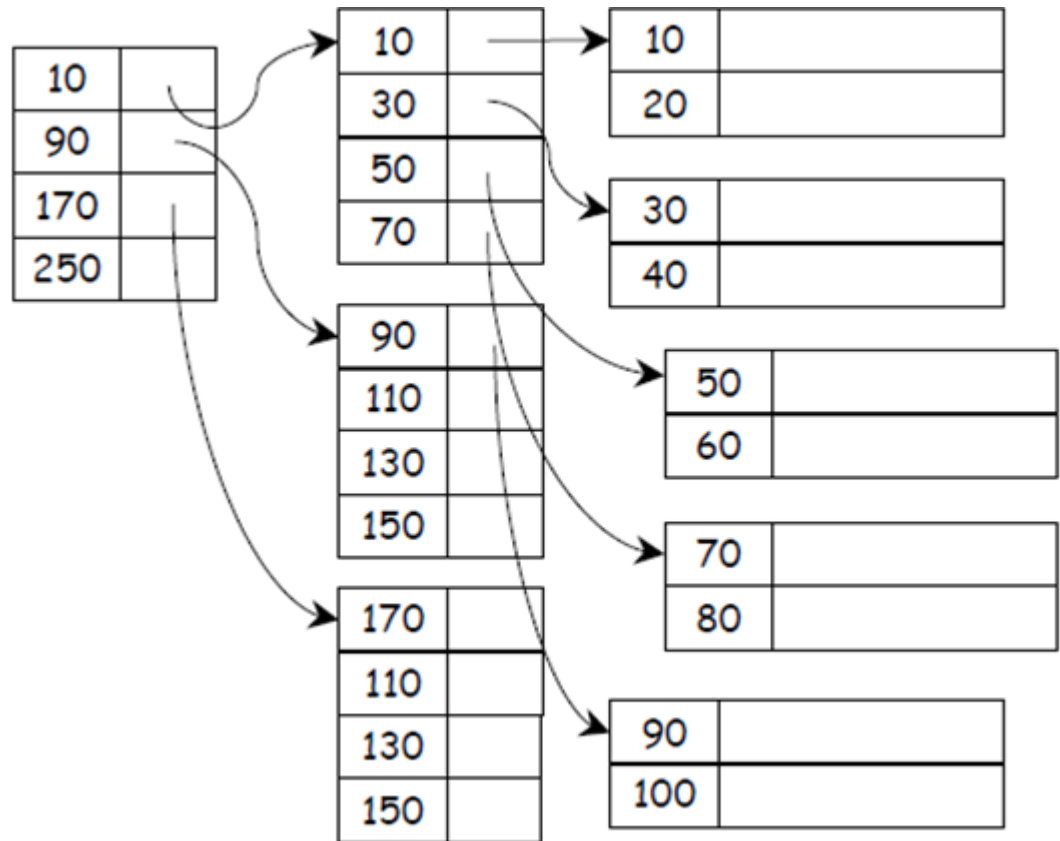
## ❑ Índices Esparsos:

- ❖ Apenas **alguns registros** de dados **são representados** no arquivo de índices.



# Índices Ordenados

## ❑ Índices de Níveis Múltiplos:

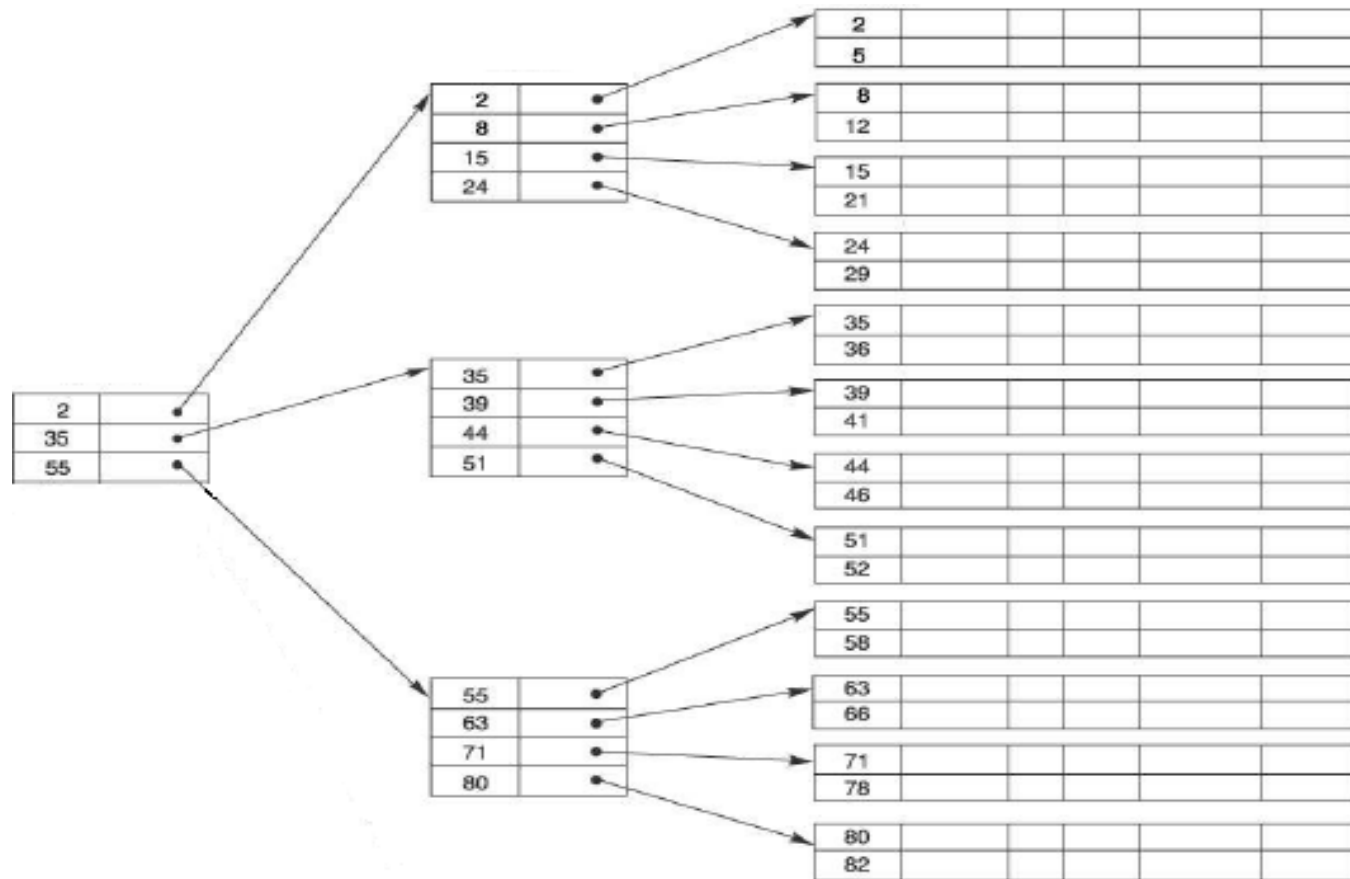


## ❑ Desvantagem:

- ❖ Muitos níveis de índices podem **aumentar a complexidade** do sistema.

# Índices Ordenados

## □ Índices de Níveis Múltiplos:



# Índices Ordenados

## □ Árvore B:

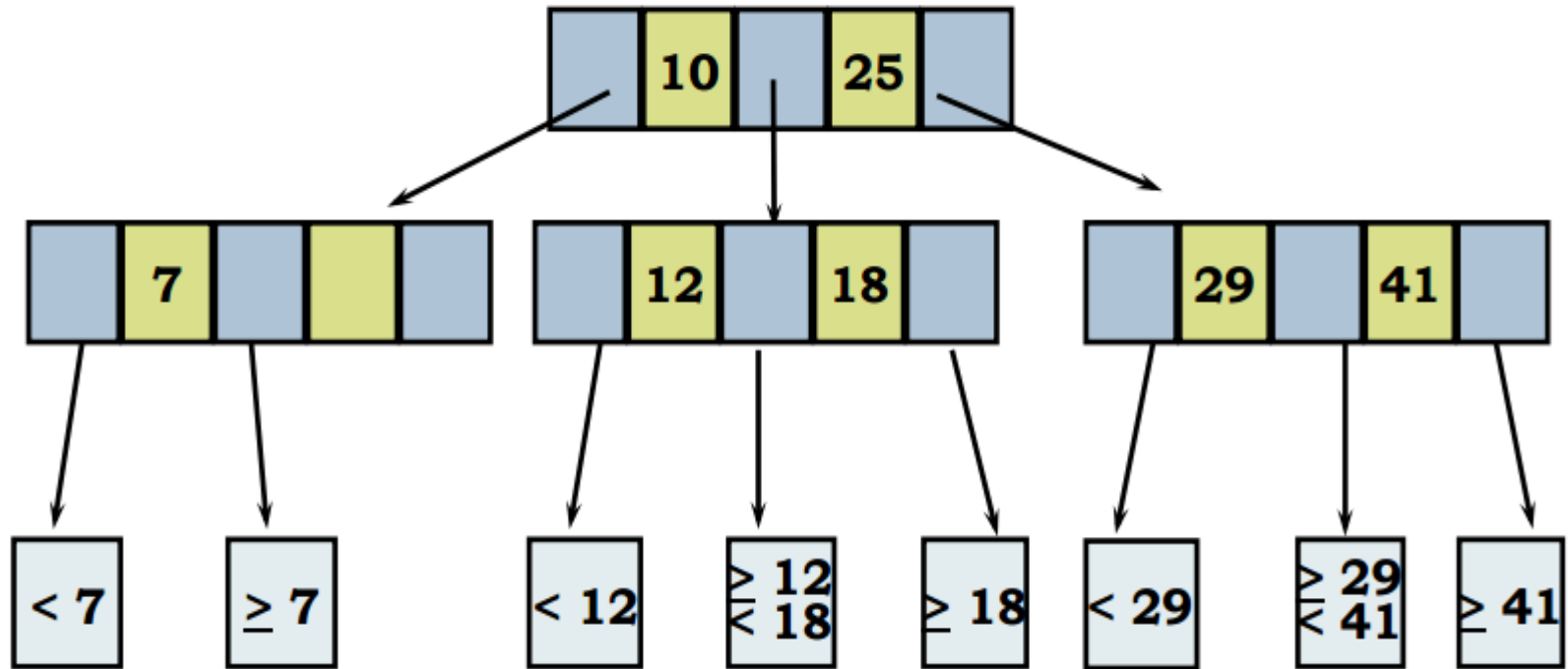
- ❖ É uma generalização da **árvore binária de busca**.
  - Cada nó de uma ABB armazena uma **única chave** de busca.
  - Já a árvore B armazena um número **maior** ou igual a **1** de **chaves de busca** em cada nó.
- ❖ **Árvores B** são projetadas com dois objetivos:
  - Manter a árvore balanceada; e
  - **Evitar o desperdício de espaço dentro de um nó.**



# Índices Ordenados

## □ Árvore B:

❖ Exemplo:



# Índices Ordenados

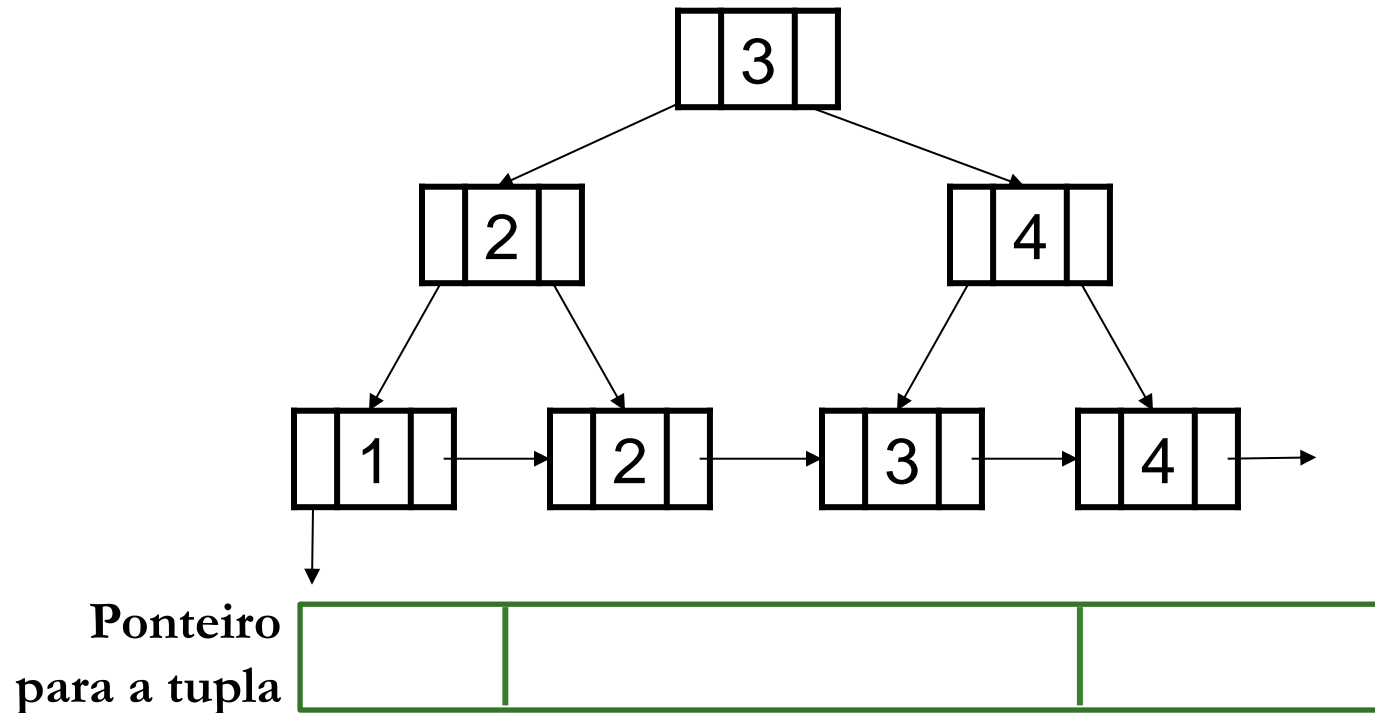
## □ Árvore B+:

- ❖ É semelhante à árvore B, exceto por duas características muito importantes:
  - Armazena dados somente nas folhas. Os nós internos servem apenas de ponteiros.
  - As folhas são encadeadas.
- ❖ Isso permite o armazenamento dos dados em um arquivo, e do índice em outro arquivo separado.

# Índices Ordenados

## □ Árvore B+:

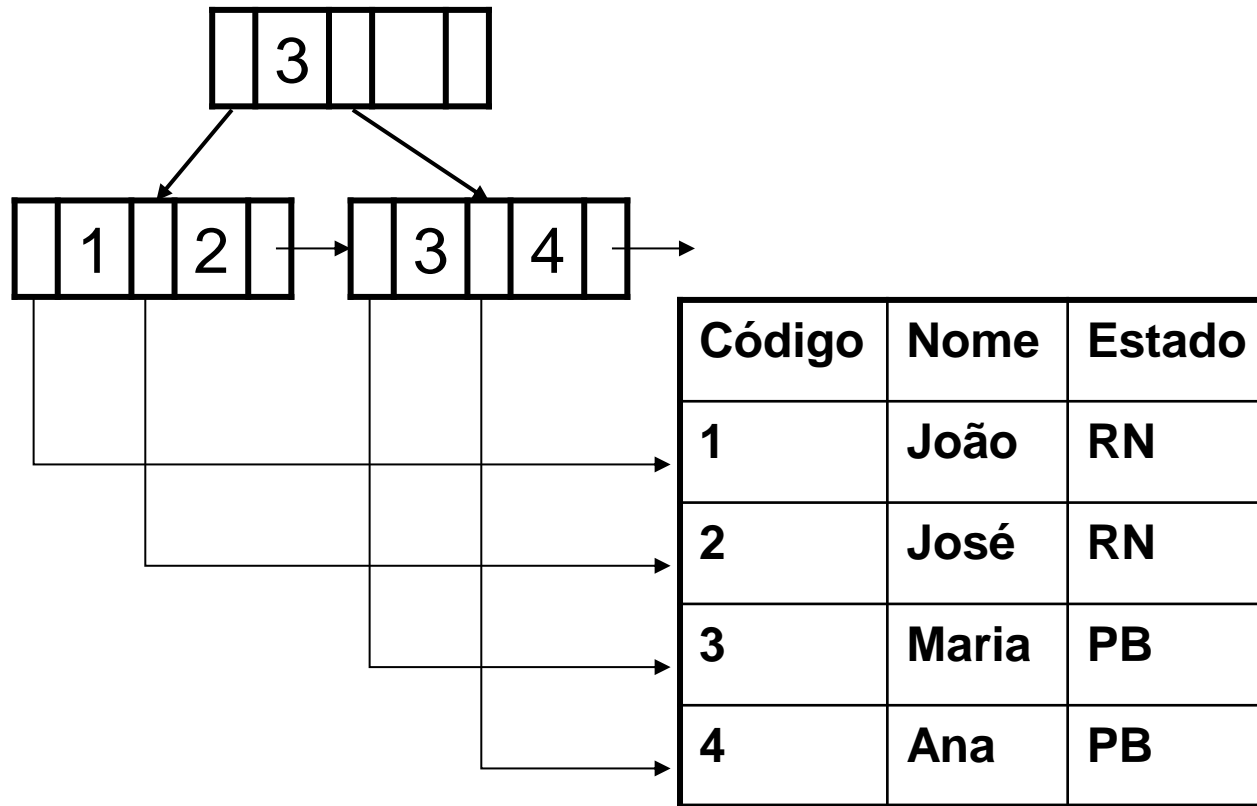
❖ Implementação com 2 ponteiros e 1 valor ( $n = 2$ )



# Índices Ordenados

## □ Árvore B+:

❖ Implementação com 3 ponteiros e 2 valores ( $n = 3$ )



# Índices Ordenados

## □ Árvore-B x Árvore-B+:

- ❖ As árvores B não apresentam redundância de valores;
- ❖ Possuem um ponteiro adicional para cada nó não folha.
- ❖ Ocupam menos espaço, porém as operações de atualização são mais complexas.

# Índices Ordenados

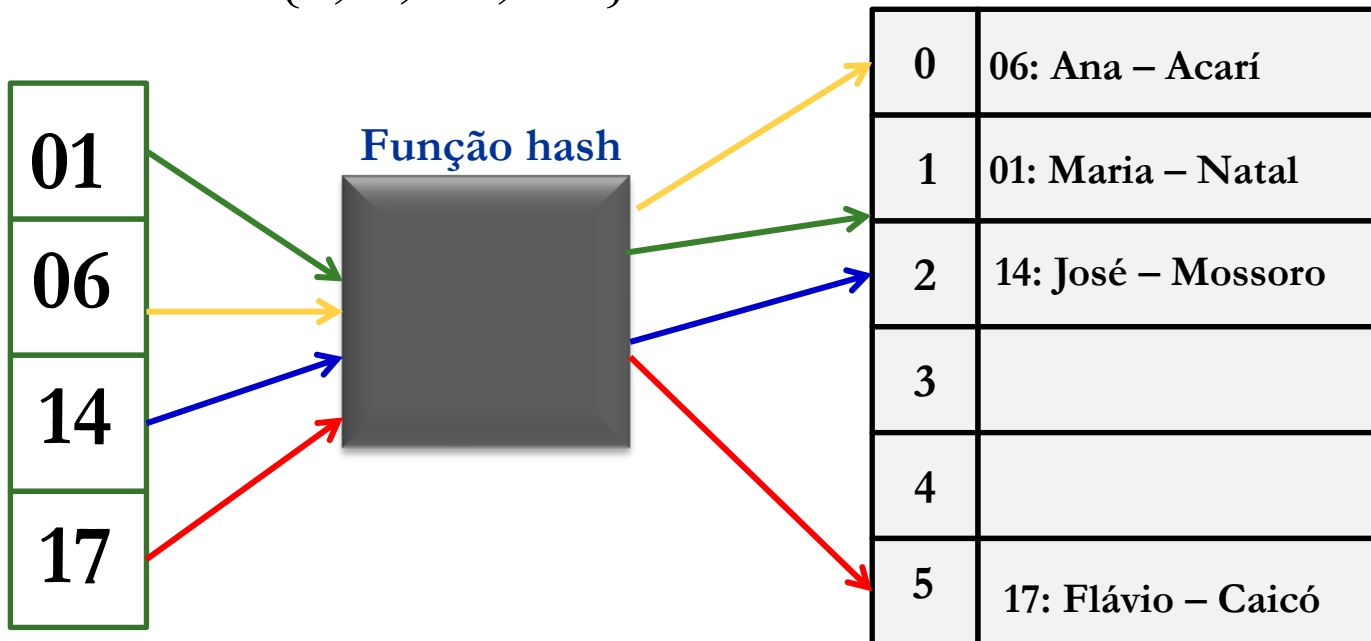
## □ Índices B-tree:

- ❖ Vários Sistemas de Gerência de Banco de Dados como:
  - IBM DB2;
  - Informix;
  - Microsoft SQL Server;
  - Oracle;
  - Sybase ASE;
  - PostgreSQL;
  - Firebird;
  - MySQL e SQLite.
- ❖ Suportam os tipos **B-Tree** para indexar tabelas.

# Índices Hash

## ❑ Hash estático:

- ❖ Para um número de 6 buckets ( $n = 6$ ).
- ❖  $h(x) = x \bmod n$
- ❖  $Chaves = \{1, 6, 14, 17\}$

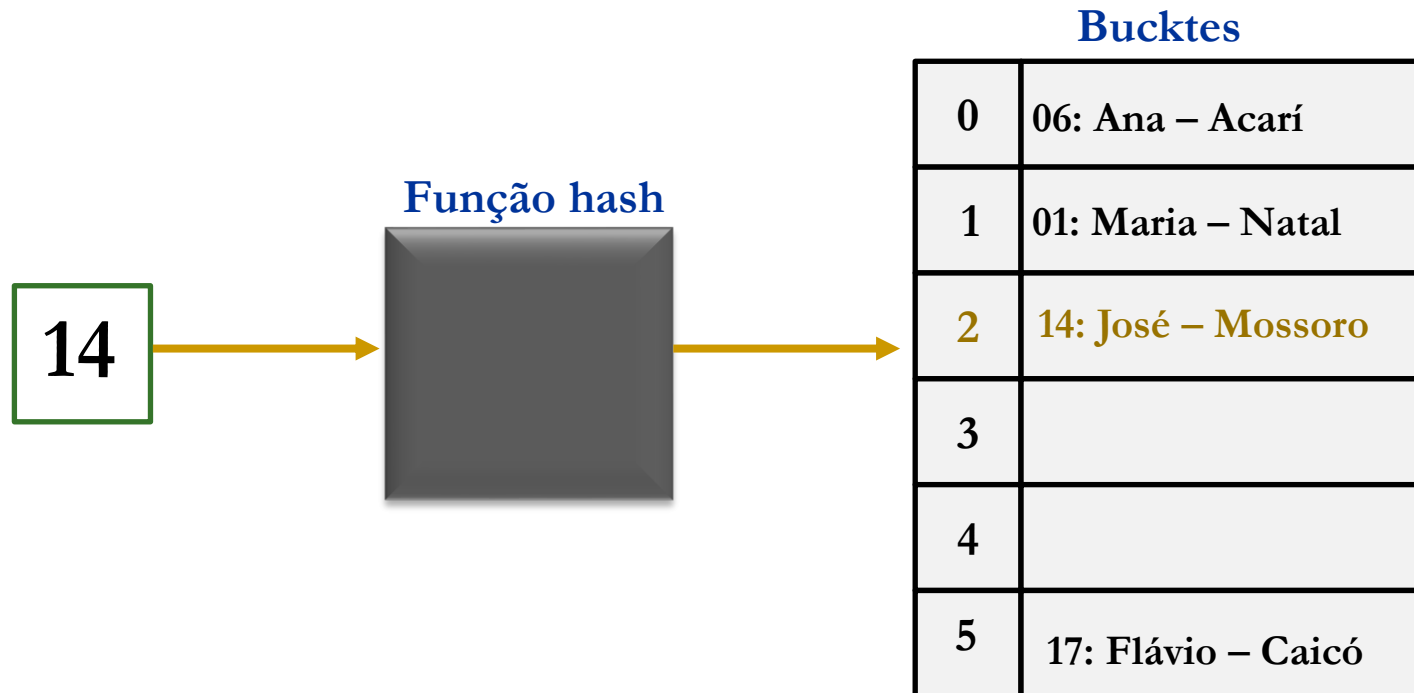


# Índices Hash

## ❑ Hash estático:

❖ Busca/consulta:

❖  $h(14) = 14 \bmod 6 = 2$



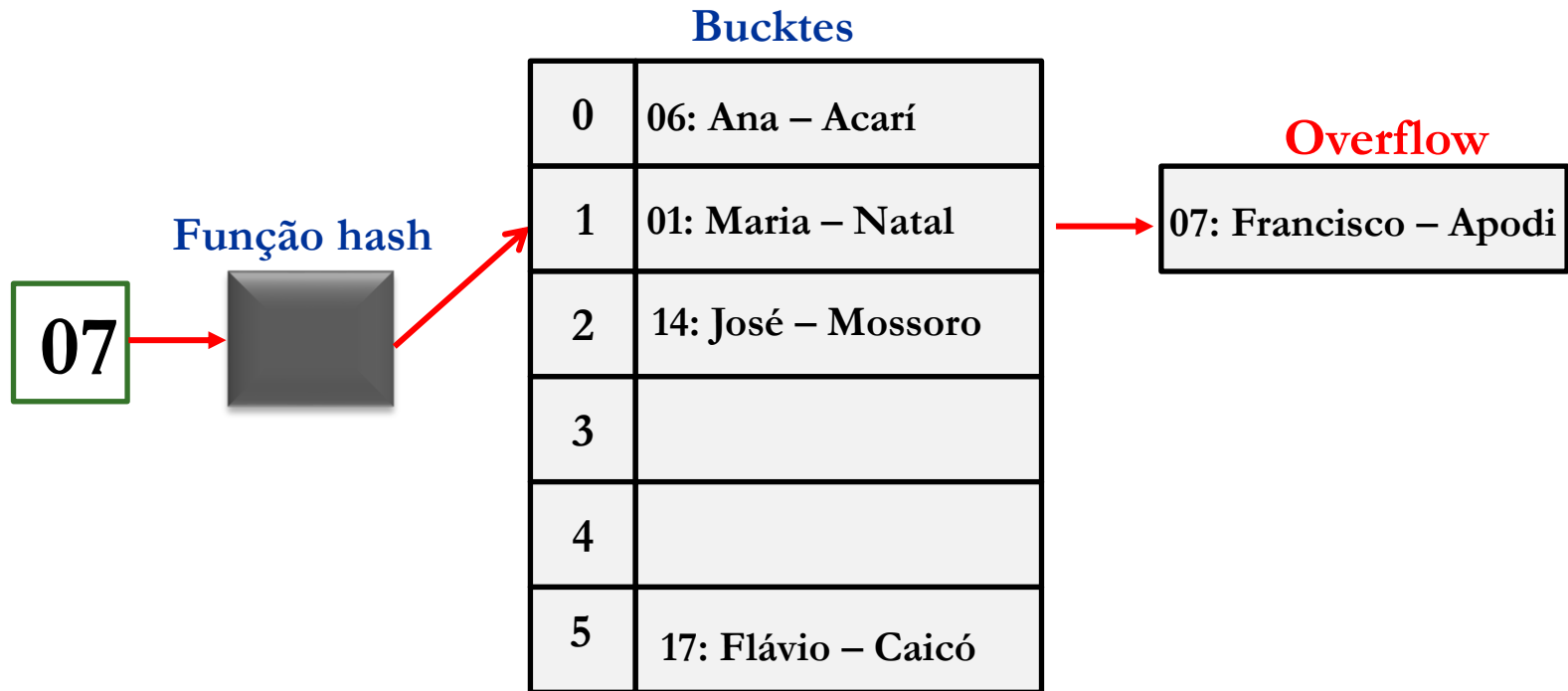


# Índices Hash

## ❑ Hash estático:

❖ Inserindo 7:

❖  $h(7) = 7 \bmod 6 = 1$



# Criação de Índices no PostgreSQL

□ Há quatro tipos de índice suportados pelo PostgreSQL. Eles são:

- ❖ **B-Tree**: utilizado para indexar colunas que geralmente serão consultadas por intervalo. Por exemplo o campo *salário*. **B-Tree é tipo padrão.**
- ❖ **Hash**: utilizado para indexar colunas que serão consultadas por um valor exato. Por exemplo o campo *CPF*.
- ❖ **GiST**: não são índices básicos, mas sim uma estrutura onde se podem ter várias estratégias diferentes de Indexação. Usados para **dados geográficos**.
- ❖ **GIN**: são índices invertidos que lidam (ou podem lidar) com valores com mais do que uma chave. Usados para lidar com **arrays unidimensionais**.

# Criação de Índices no PostgreSQL

## ❑ Criando índices em SQL:

- ❖ Podemos criar um índice através do comando:

```
CREATE    [UNIQUE]    INDEX    <nome_index>  
ON <nome_tabela> (<atributo>)
```

- ❖ Exemplos:

```
CREATE INDEX codturma_idx    ON          aluno  
(codturma);
```

```
CREATE INDEX dept_salario_idx ON empregado  
(dept ASC, salario DESC);
```

- ❖ Para apagar um índice usamos o comando:

```
DROP INDEX <nome_index>
```

# Criação de Índices no PostgreSQL

## ❑ Usando índices em SQL:

❖ Para forçar uma consulta SQL a usar um índice, recomendamos na cláusula WHERE colocar os atributos na mesma ordem do índice.

❖ Exemplo:

```
SELECT * FROM CLIENTE WHERE codigo = 100;
```

❖ O uso do operador **LIKE** não permite a utilização de índice.

# Dúvidas...



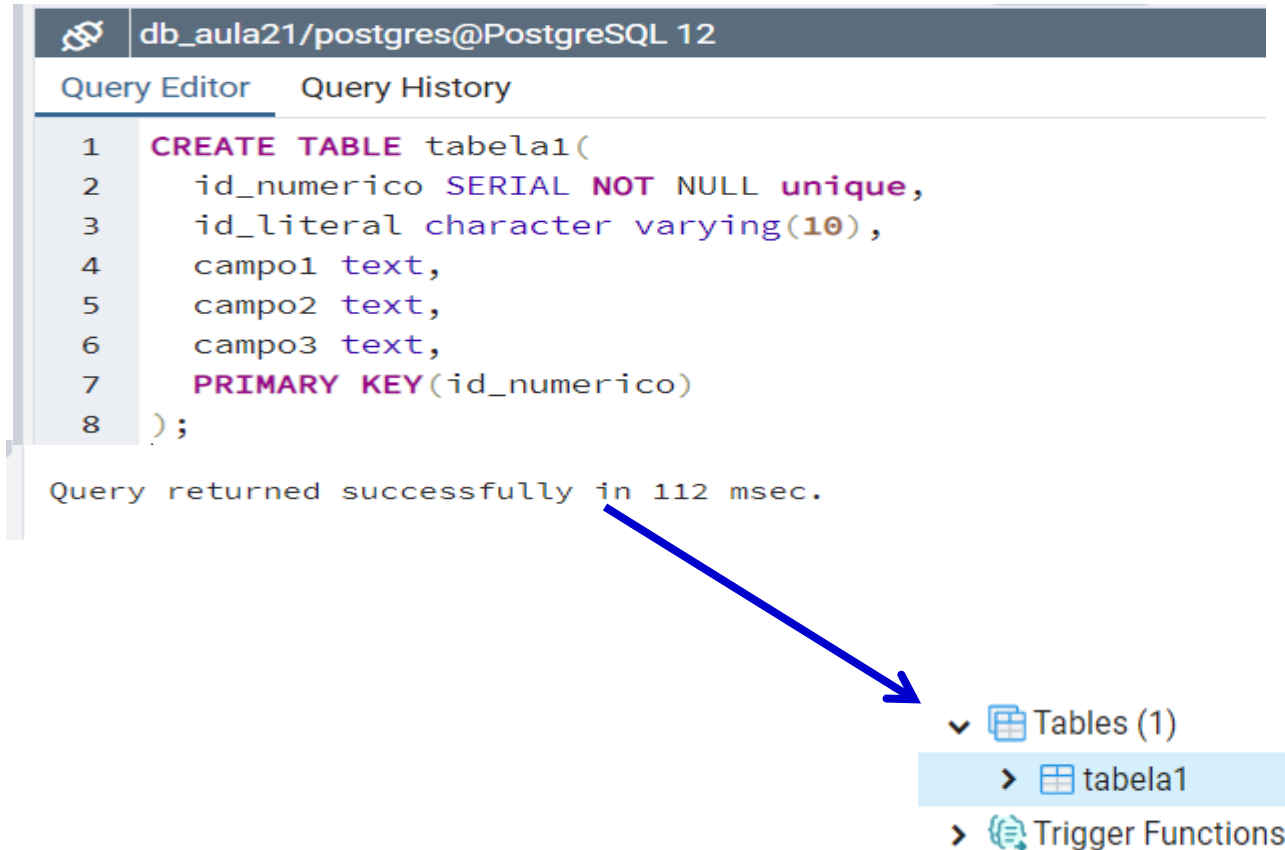
# Trabalho Prático

- ❑ Crie a tabela abaixo, segundo o script:

```
CREATE TABLE tabela1(  
    id_numerico SERIAL NOT NULL unique,  
    id_literal character varying(10),  
    campo1 text,  
    campo2 text,  
    campo3 text,  
    PRIMARY KEY(id_numerico)  
);
```

# Trabalho Prático

## ❑ Criando tabela:



The screenshot displays a PostgreSQL Query Editor window for the database 'db\_aula21/postgres@PostgreSQL 12'. The 'Query Editor' tab is active, showing a SQL statement to create a table named 'tabela1'. The statement is as follows:

```
1 CREATE TABLE tabela1(  
2     id_numerico SERIAL NOT NULL unique,  
3     id_literal character varying(10),  
4     campo1 text,  
5     campo2 text,  
6     campo3 text,  
7     PRIMARY KEY(id_numerico)  
8 );
```

Below the query, a status message indicates: 'Query returned successfully in 112 msec.' A blue arrow points from this message to the 'Tables (1)' section of the database schema browser on the right. In this section, the table 'tabela1' is listed under the 'Tables (1)' category, which is expanded to show its details.

# Trabalho Prático

- Agora insira registros (20) na tabela criada. Use para tal o script abaixo:

```
insert into tabela1 (id_literal, campo1, campo2,
campo3) values ('IMD0000001', 'VAMOS POPULAR ESSA
TABELA COM MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM
MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM MUITO
TEXTO');
```

```
insert into tabela1 (id_literal, campo1, campo2,
campo3) values ('IMD0000002', 'VAMOS POPULAR ESSA
TABELA COM MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM
MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM MUITO
TEXTO');
```



# Trabalho Prático

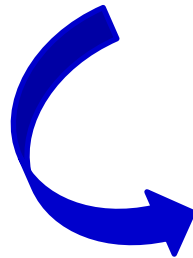
## □ Registros inseridos:

```
56 values ('IMD0000019', 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO',  
57 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO');  
58 insert into tabela1 (id_literal, campo1, campo2, campo3)  
59 values ('IMD0000020', 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO',  
60 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO', 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO');
```

Data Output Explain Messages Notifications

INSERT 0 1

Query returned successfully in 108 msec.



Data Output		Explain	Messages	Notifications	
	<div><div>id_numerico</div><div>[PK] integer</div></div>	<div><div>id_literal</div><div>character varying (10)</div></div>	<div><div>campo1</div><div>text</div></div>	<div><div>campo2</div><div>text</div></div>	<div><div>campo3</div><div>text</div></div>
1	1	IMD0000001	VAMOS POP...	VAMOS POP...	VAMOS POP...
2	2	IMD0000002	VAMOS POP...	VAMOS POP...	VAMOS POP...
3	3	IMD0000003	VAMOS POP...	VAMOS POP...	VAMOS POP...
4	4	IMD0000004	VAMOS POP...	VAMOS POP...	VAMOS POP...
5	5	IMD0000005	VAMOS POP...	VAMOS POP...	VAMOS POP...
6	6	IMD0000006	VAMOS POP...	VAMOS POP...	VAMOS POP...
7	7	IMD0000007	VAMOS POP...	VAMOS POP...	VAMOS POP...
8	8	IMD0000008	VAMOS POP...	VAMOS POP...	VAMOS POP...

# Trabalho Prático

- ❑ Agora, depois de inseridos os registros, analise a tabela de duas formas diferentes:

```
EXPLAIN ANALYSE SELECT * FROM tabela1  
WHERE id_literal = 'IMD000020';
```

	Data Output	Explain	Messages	Notifications
	<b>QUERY PLAN</b>			
	text			
1	Seq Scan on tabela1 (cost=0.00..16.13 rows=2 width=138) (actual time=0.021..0.021 rows=0 loops...			
2	Filter: ((id_literal)::text = 'IMD000020'::text)			
3	Rows Removed by Filter: 20			
4	Planning Time: 0.776 ms			
5	Execution Time: 0.035 ms			

# Trabalho Prático

- Agora, depois de inseridos os registros, analise a tabela de duas formas diferentes:

```
EXPLAIN ANALYSE SELECT * FROM tabela1  
WHERE id_numerico BETWEEN 1 AND 20;
```

	Data Output	Explain	Messages	Notifications
	<b>QUERY PLAN</b> text			
1	Bitmap Heap Scan on tabela1 (cost=4.17..9.51 rows=2 width=138) (actual time=0.023..0.024 rows=20 loops=1)			
2	Recheck Cond: ((id_numerico >= 1) AND (id_numerico <= 20))			
3	Heap Blocks: exact=1			
4	-> Bitmap Index Scan on tabela1_pkey (cost=0.00..4.17 rows=2 width=0) (actual time=0.013..0.013 rows=20 loops=1)			
5	Index Cond: ((id_numerico >= 1) AND (id_numerico <= 20))			
6	Planning Time: 0.271 ms			
7	Execution Time: 0.092 ms			

# Trabalho Prático

❑ Agora crie dois índices na tabela:

```
CREATE INDEX index_hash ON tabela1 USING  
HASH (id_literal);
```

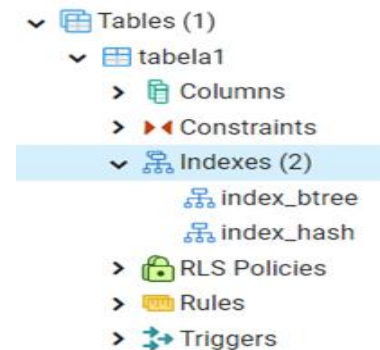
```
CREATE INDEX
```

```
Query returned successfully in 126 msec.
```

```
CREATE INDEX index_btree ON tabela1 USING  
btree (id_numerico);
```

```
CREATE INDEX
```

```
Query returned successfully in 67 msec.
```



# Trabalho Prático

- ❑ Agora analise novamente a tabela com seus registros de duas formas:

```
SET enable_seqscan To OFF;
```

```
EXPLAIN ANALYSE SELECT * FROM tabela1  
WHERE id_literal = 'IMD000020';
```

Data Output	Explain	Messages	Notifications
	<b>QUERY PLAN</b> text		
1	Index Scan using index_hash on tabela1 (cost=0.00..8.02 rows=1 width=138) (actual time=0.036..0.0...		
2	Index Cond: ((id_literal)::text = 'IMD000020'::text)		
3	Planning Time: 0.144 ms		
4	Execution Time: 0.078 ms		

# Trabalho Prático

- Agora analise novamente a tabela com seus registros de duas formas:

```
SET enable_seqscan To OFF;
```

```
EXPLAIN ANALYSE SELECT * FROM tabela1  
WHERE id_numerico BETWEEN 1 AND 20;
```

Data Output	Explain	Messages	Notifications
<div>QUERY PLAN</div> <div>text</div> <div>1 Index Scan using index_btree on tabela1 (cost=0.14..8.16 rows=1 width=138) (actual time=0.069..0.073 rows...</div> <div>2 Index Cond: ((id_numerico &gt;= 1) AND (id_numerico &lt;= 20))</div> <div>3 Planning Time: 1.586 ms</div> <div>4 Execution Time: 0.097 ms</div>			

# Trabalho Prático

❑ Utilize o script abaixo:

```
1  /* Função alterada */
2  CREATE OR REPLACE FUNCTION criarTabela() RETURNS VOID AS
3  $BODY$
4      DECLARE
5          informacao TEXT;
6          i INTEGER;
7      BEGIN
8          i := 1;
9          informacao := 'VAMOS POPULAR ESSA TABELA COM MUITO TEXTO';
10         DROP TABLE IF EXISTS tabela2;
11         CREATE TABLE tabela2 (
12             id_numerico INTEGER,
13             id_literal VARCHAR(15),
14             campo1 TEXT,
15             campo2 TEXT,
16             campo3 TEXT
17         );
18         LOOP
19             INSERT INTO tabela2 (id_numerico, id_literal, campo1, campo2, campo3)
20                 values (i, ('IMD000' || i), informacao, informacao, informacao);
21             EXIT WHEN i > 1000;
22             i := i + 1;
23         END LOOP;
24     END;
25 $BODY$
26 LANGUAGE plpgsql VOLATILE;
27
28 /* Para executar a função */
29 Select criartabela();
```

# Trabalho Prático

❑ Refaça os experimentos:

```
EXPLAIN ANALYSE SELECT * FROM tabela2  
WHERE id_literal = 'IMD000500';
```

```
EXPLAIN ANALYSE SELECT * FROM tabela2  
WHERE id_numerico BETWEEN 1 AND 1000;
```

```
CREATE INDEX index_hash ON tabela2 USING  
HASH (id_literal);
```

```
CREATE INDEX index_btree ON tabela2 USING  
btree (id_numerico);
```



# Questões...

