Banco de Dados – IMD0401

Aula 23 – Indexação em Banco de Dados

João Carlos Xavier Júnior

jcxavier@imd.ufrn.br

INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Índices em Banco de Dados

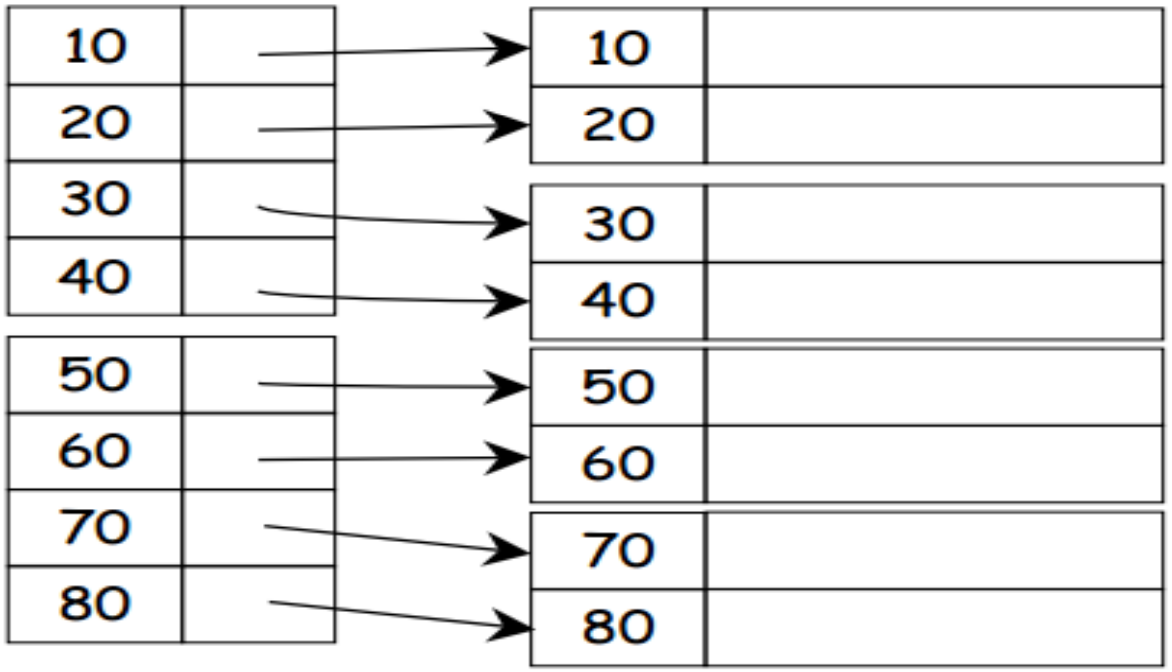
- **Índices**:
  - Índices são estruturas associadas aos bancos de dados com a capacidade de localizar diretamente um ou mais registros através de uma "**chave**" de procura.
  - Existem dois tipos de índices:
    - **Ordenados**;
    - **Hash**.
  - A escolha de uma técnica de indexação avalia os seguintes 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**.

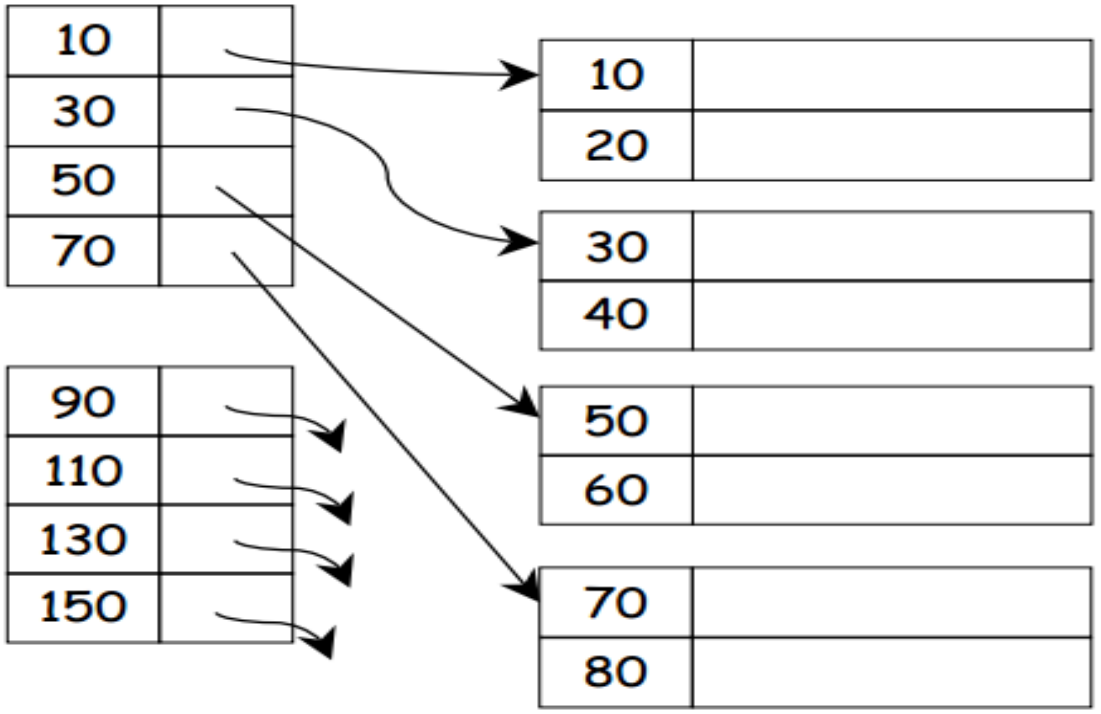
Índices Ordenados

- **Índices Densos:**
  - Uma entrada no **arquivo de índices** é criada para cada registro no **arquivo de dados**.



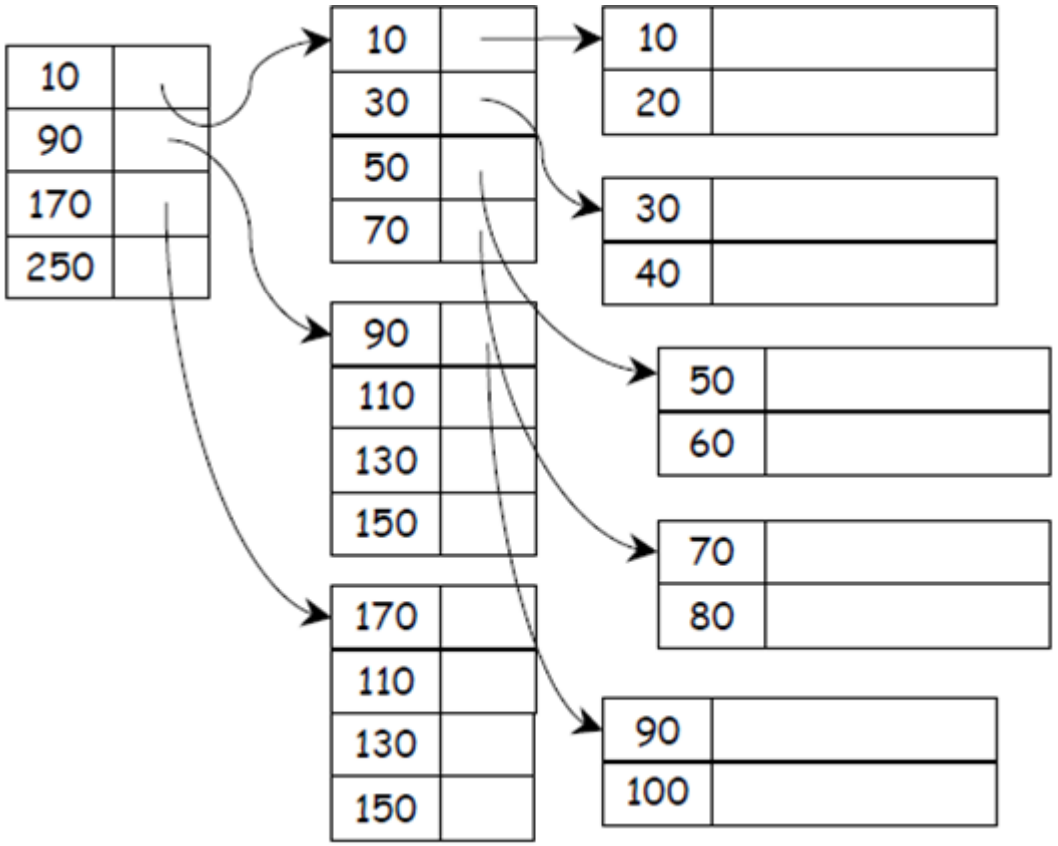
Índices Ordenados

- **Índices Esparsos:**
  - Apenas alguns registros de dados são representados no arquivo de índices.



Índices Ordenados

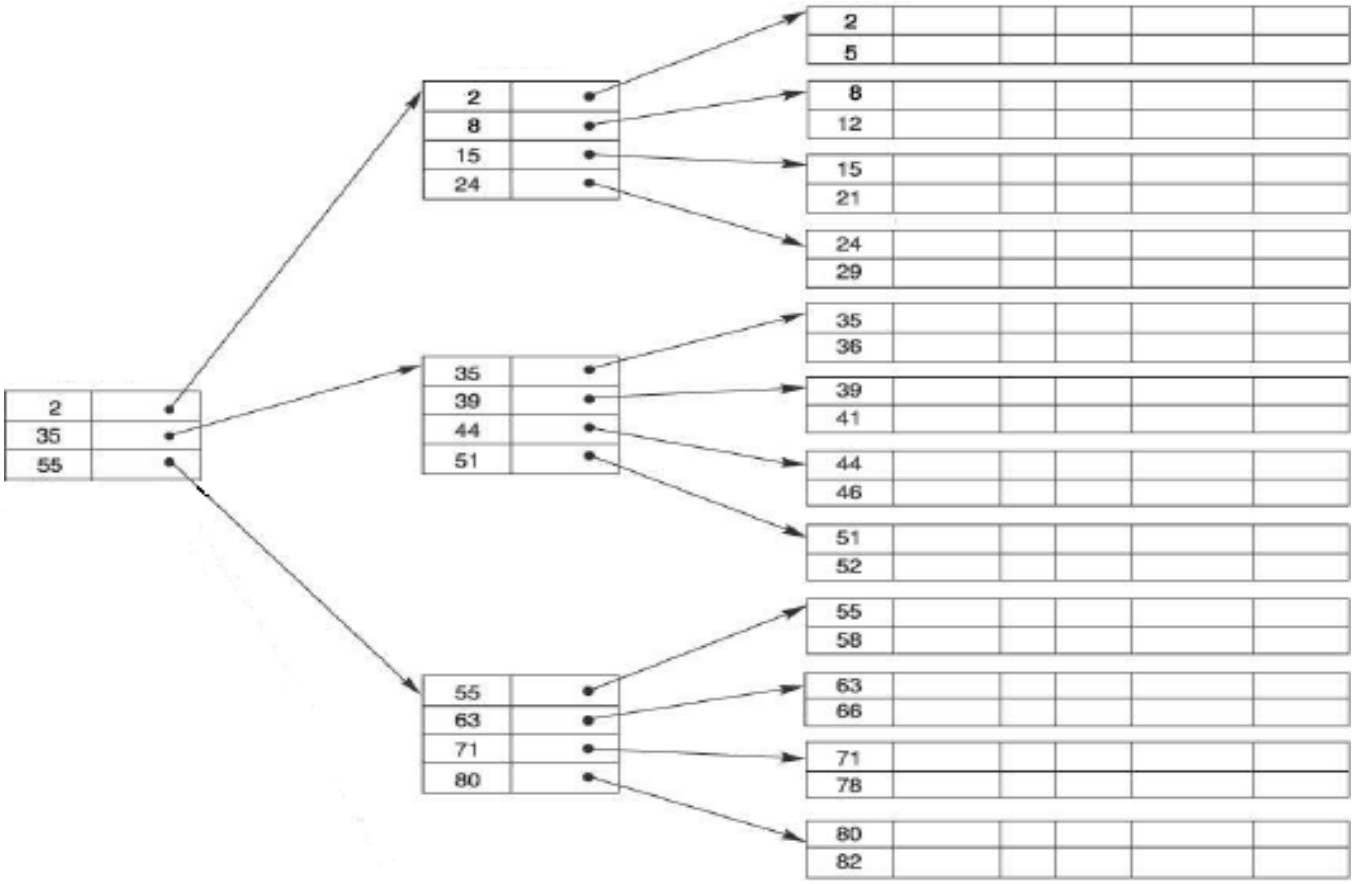
- **Índices de Níveis Múltiplos:**



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

Índices Ordenados

- Índices de Níveis Múltiplos:

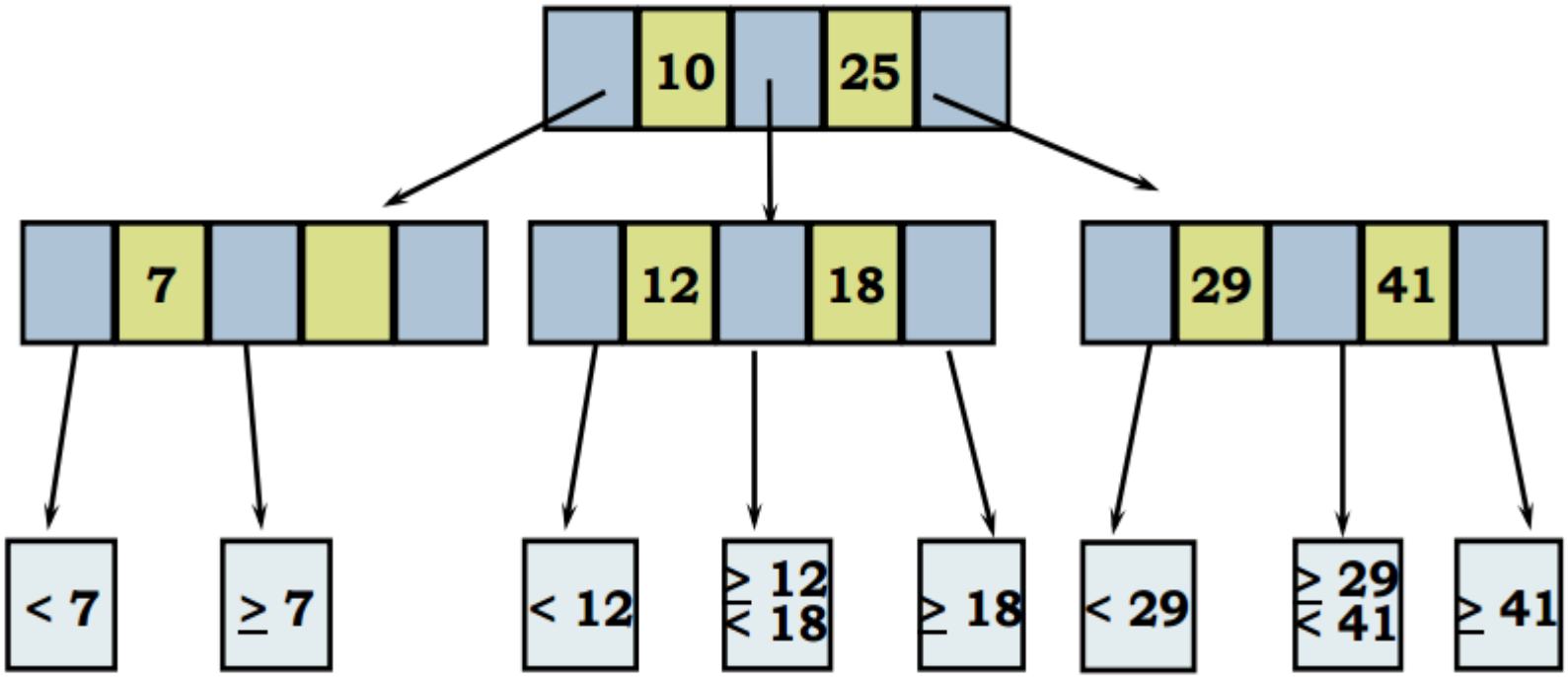


Índices Ordenados

- **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:

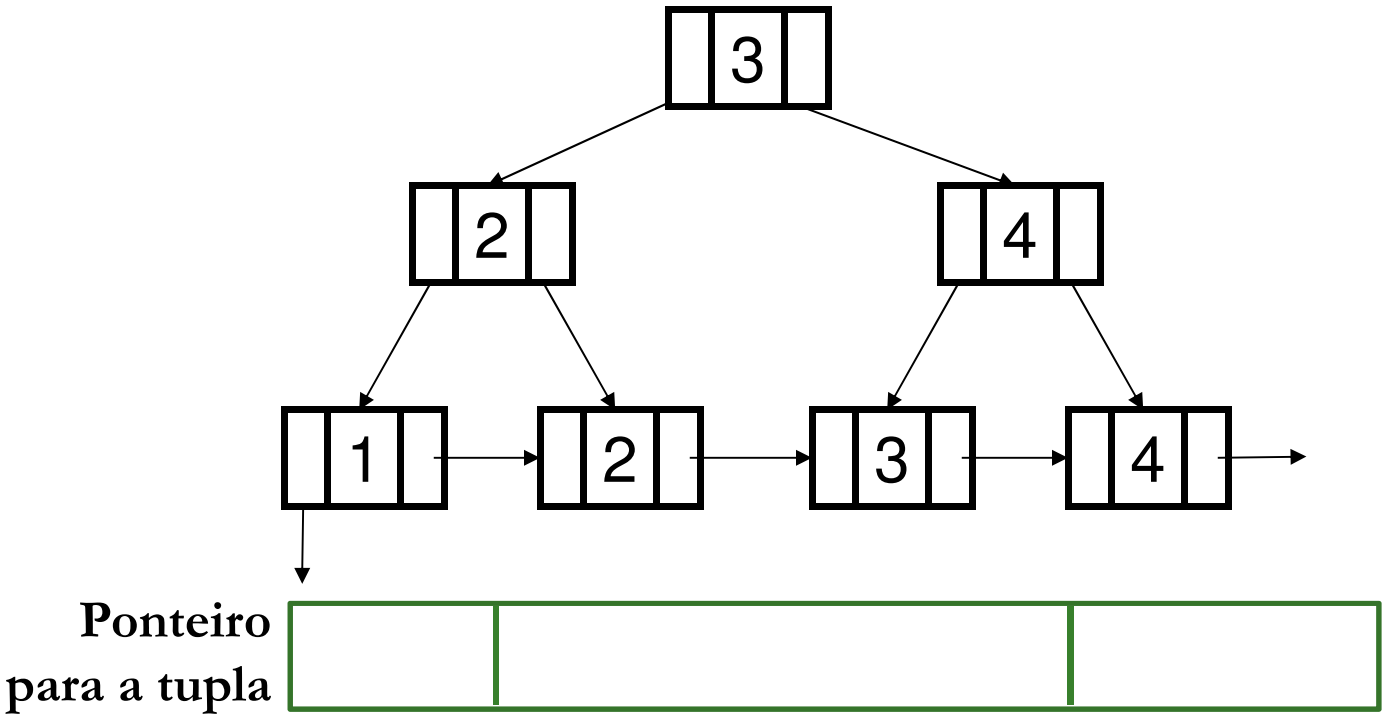


Índices Ordenados

- **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**.

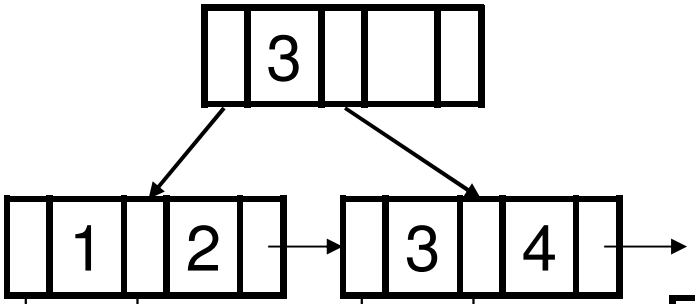
Índices Ordenados

- **Árvore B+:**
  - Implemetação com 2 ponteiros e 1 valor ($n = 2$)



Índices Ordenados

- **Árvore B+:**
  - Implemetação com 3 ponteiros e 2 valores ($n = 3$)



Código	Nome	Estado
1	João	RN
2	José	RN
3	Maria	PB
4	Ana	PB

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.

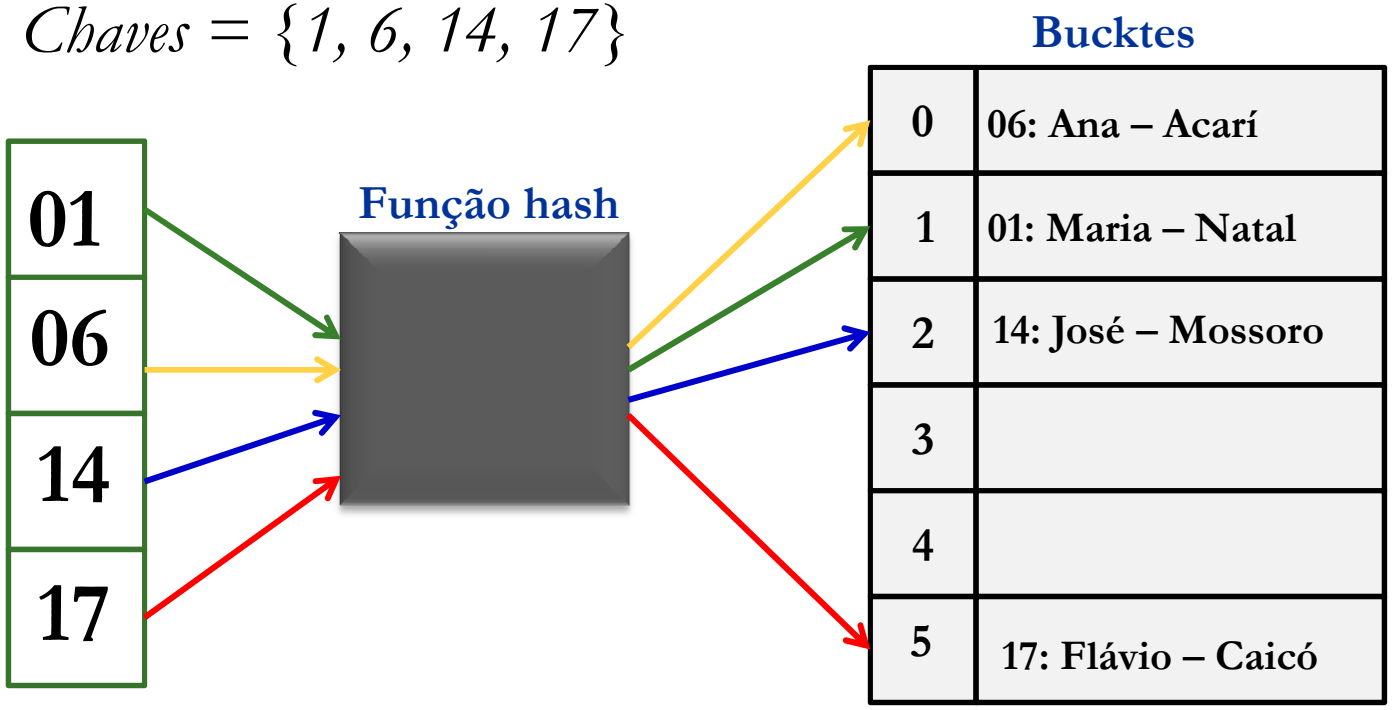
Índices Ordenados

- **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 B-tree**:
  - Vários Sistemas de Gerência de Banco de Dados como IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASE, PostgreSQL, Firebird, MySQL e SQLite suportam os tipos **B-Tree** para indexar tabelas.

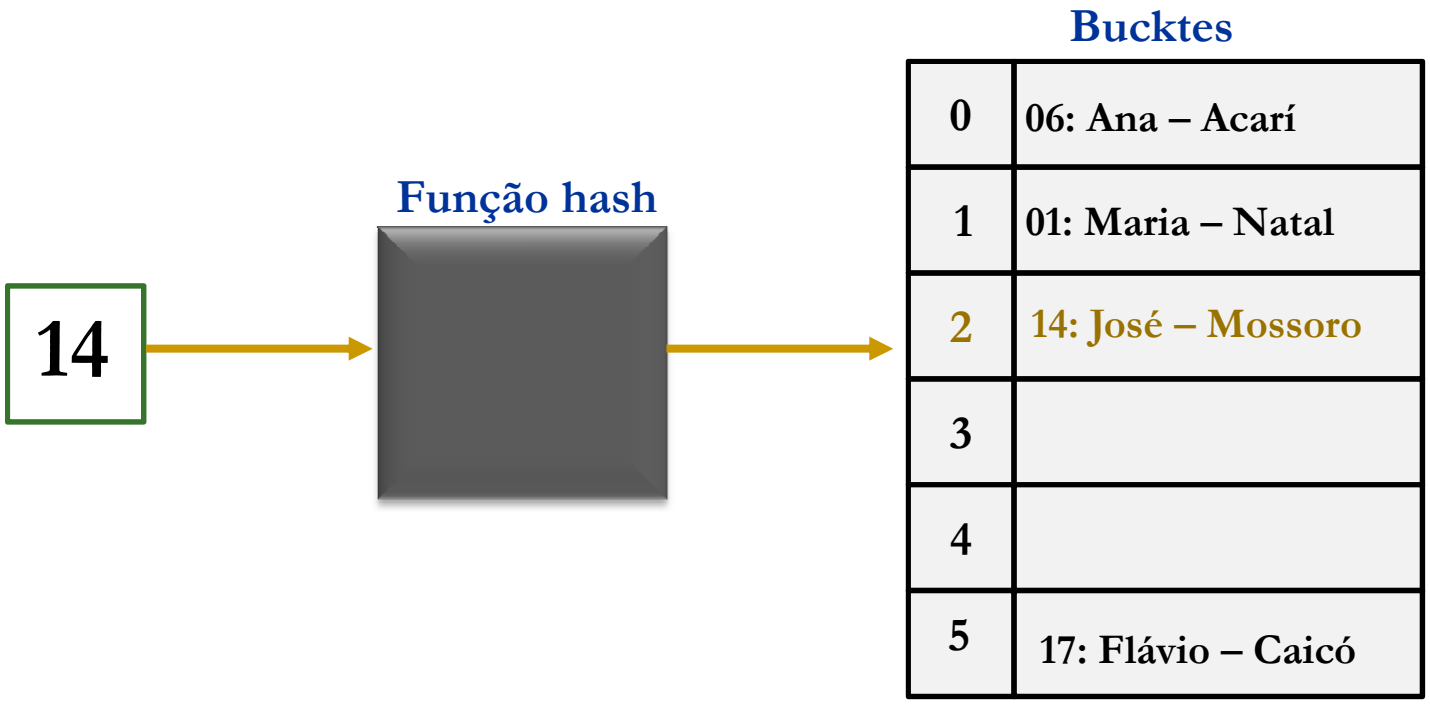
Índices Hash

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



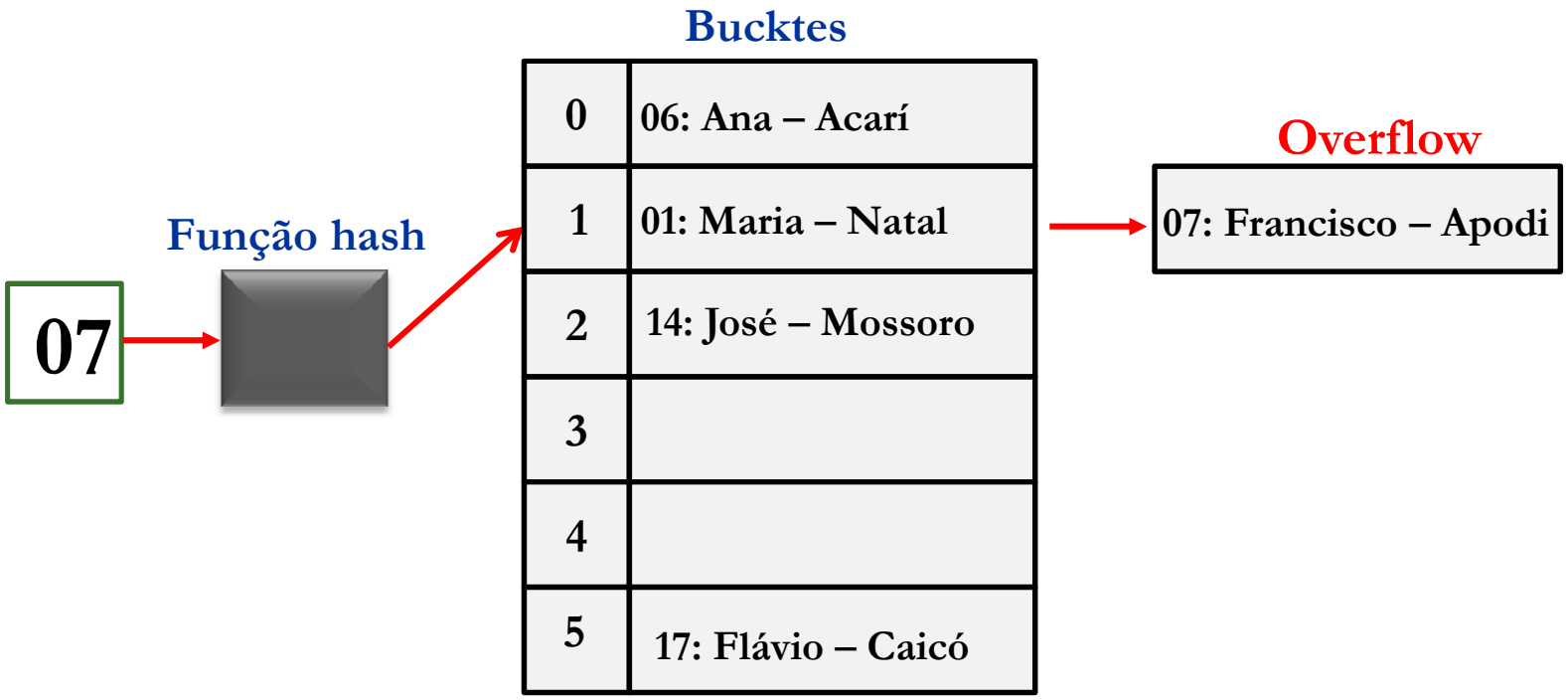
Índices Hash

- **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 estrutra 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_indice>
     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_indice>
    ```

Criação de Índices no PostgreSQL

- **Usando índices em SQL**:
  - Para forçar uma consulta SQL 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

- Agora insira registros (30) 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

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

```
EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_literal = 'IMD000500';

EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_numerico BETWEEN 1 AND 1000;
```

Trabalho Prático

- Agora crie dois índices na tabela:

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

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

Trabalho Prático

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

```
EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_literal = 'IMD000500';

EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_numerico BETWEEN 1 AND 1000;
```

Trabalho Prático

- O comando SET enable_seqscan To OFF força o PostgreSQL a utilizar os índices.

```
SET enable_seqscan To OFF

EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_literal = 'IMD000500';

EXPLAIN ANALYSE SELECT * FROM tabela1
WHERE id_numerico BETWEEN 1 AND 1000;
```