

Bases de Dados

T20 - Índices Parte I

Prof. Daniel Faria

Prof. Flávio Martins

Sumário

- Recapitulação Breve
- Índices: Introdução
- Classificação de Índices
- Custo de Índices
- Métodos/Tipos de Índice

Recapitulação Breve

Desenvolvimento de Aplicações Web

```
...  
$sql = "INSERT INTO account VALUES  
( '$account_number', '$branch_name',  
$balance)";  
...
```



Insert a new account

Account no.:

Branch:

Balance:

Usar sempre prepared statements para evitar
injeção de SQL:

```
$stmt = $connection->prepare(  
    "INSERT INTO account VALUES  
(:account_number, :branch_name, :balance)");
```



Retorcedendo: Interrogação de BD

```
[WITH with_query [, ...]]  
SELECT [ALL | DISTINCT [ON (expression [, ...])]]  
      [* | expression [[AS] output_name] [, ...]]  
      [FROM from_item [, ...]]  
      [WHERE condition]  
      [GROUP BY [ALL | DISTINCT] grouping_element [, ...]]  
      [HAVING condition]  
      [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]  
      [ORDER BY expression [ASC | DESC | USING operator]  
            [NULLS { FIRST | LAST}] [, ...]]  
      [LIMIT {count | ALL}]
```

Determinação do Elemento Distintivo

- Qual o porco que realizou menos vendas (de entre os que realizaram alguma)?

GROUP BY seller

COUNT (*)

```
SELECT seller
```

```
FROM buys
```

```
GROUP BY seller HAVING COUNT(*) <= ALL(
```

```
    SELECT COUNT(*)
```

```
    FROM seller
```

```
    GROUP BY seller
```

```
);
```

Retornar agrupador

Ligar agrupador ao agregado


Agrupar, agregar e comparar


Equivalente à query externa mas retornando agregado


Divisão

- Que **porcos** realizaram vendas de produtos de **todas as espécies** de animais?

```
SELECT id
FROM pig p
WHERE NOT EXISTS (
  SELECT species
    FROM nonpig
  EXCEPT
  SELECT species
    FROM nonpig JOIN produce ON (id=producer) JOIN buys USING (code)
    WHERE seller=p.id
);
```

 **Quociente** (porcos a retornar)

 **Divisor** (espécies)

 **Dividendo**
(espécies das vendas dos porcos)

Índices: Introdução

Execução de Interrogações

- Como resolver a seguinte *query*?

```
SELECT DISTINCT customer_name FROM customer c WHERE NOT EXISTS (  
    SELECT branch_name FROM branch WHERE branch_city = c.customer_city  
EXCEPT  
    SELECT branch_name FROM depositor d JOIN account USING (account_number)  
    WHERE d.customer_name = c.customer_name);
```

Execução de Interrogações

- Algoritmo “naive”

- For n in customer:
 - For m in branch:
 - If $\text{branch_city} = \text{customer_city}$:
 - Add $m.\text{branch_name}$ to $\text{selected_branches}[]$
 - For o in depositor:
 - If $o.\text{customer_name} = n.\text{customer_name}$:
 - For p in account:
 - If $o.\text{account_number} = p.\text{account_number}$:
 - Remove $p.\text{branch_name}$ from $\text{selected_branches}[]$
 - If $\text{select_branches}[]$ is empty:
 - Add $n.\text{customer_name}$ to $\text{names}[]$

Na Realidade:

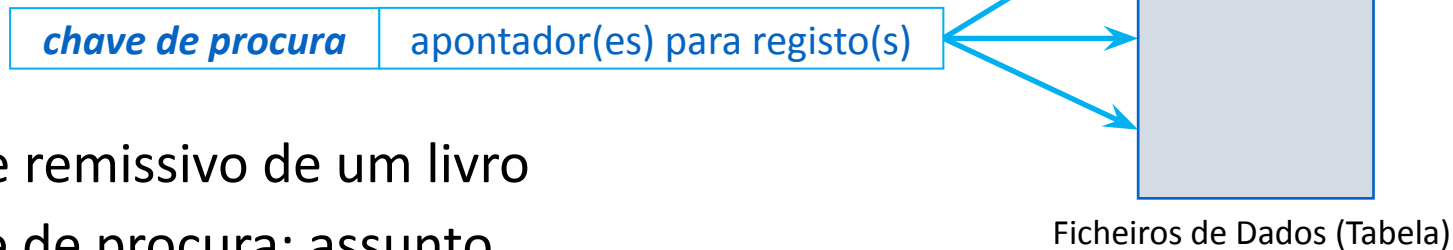
QUERY PLAN

```
-----  
Unique  (cost=26.97..27.01 rows=8 width=6)  
  -> Sort  (cost=26.97..26.99 rows=8 width=6)  
      Sort Key: c.customer_name  
      -> Seq Scan on customer c  (cost=0.00..26.85 rows=8 width=6)  
          Filter: (NOT (SubPlan 1))  
          SubPlan 1  
            -> HashSetOp Except  (cost=0.00..3.43 rows=2 width=182)  
                -> Append  (cost=0.00..3.42 rows=3 width=182)  
                    -> Subquery Scan on "*SELECT* 1"  (cost=0.00..1.13 rows=2 width=13)  
                        -> Seq Scan on branch  (cost=0.00..1.11 rows=2 width=9)  
                            Filter: ((branch_city)::text = (c.customer_city)::text)  
                        -> Subquery Scan on "*SELECT* 2"  (cost=1.14..2.27 rows=1 width=12)  
                            -> Hash Join  (cost=1.14..2.26 rows=1 width=8)  
                                Hash Cond: (account.account_number = d.account_number)  
                                -> Seq Scan on account  (cost=0.00..1.09 rows=9 width=14)  
                                -> Hash  (cost=1.12..1.12 rows=1 width=6)  
                                    -> Seq Scan on depositor d  (cost=0.00..1.12 rows=1 width=6)  
                                        Filter: ((customer_name)::text = (c.customer_name)::text)
```

↓ ↓
→ (18 rows) ←

Índice

- Estrutura de dados auxiliar que melhora a eficiência de alguns tipos de pesquisa sobre os dados (de uma tabela)
- Contém registos na forma:



- E.g. índice remissivo de um livro
 - Chave de procura: assunto
 - Apontador: nº da página

Consulta sem Índice

- Implica pesquisa ou varrimento sequencial do ficheiro onde está armazenada a tabela
 - $\mathcal{O}(n)$ para consultas de uma tabela
 - $\mathcal{O}(n^p)$ para combinação de p tabelas
 - Complexidade escala com selects encadeados

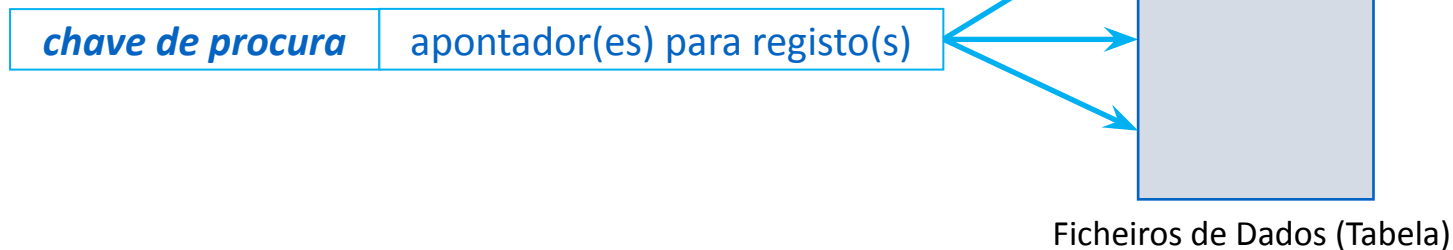
= k ?



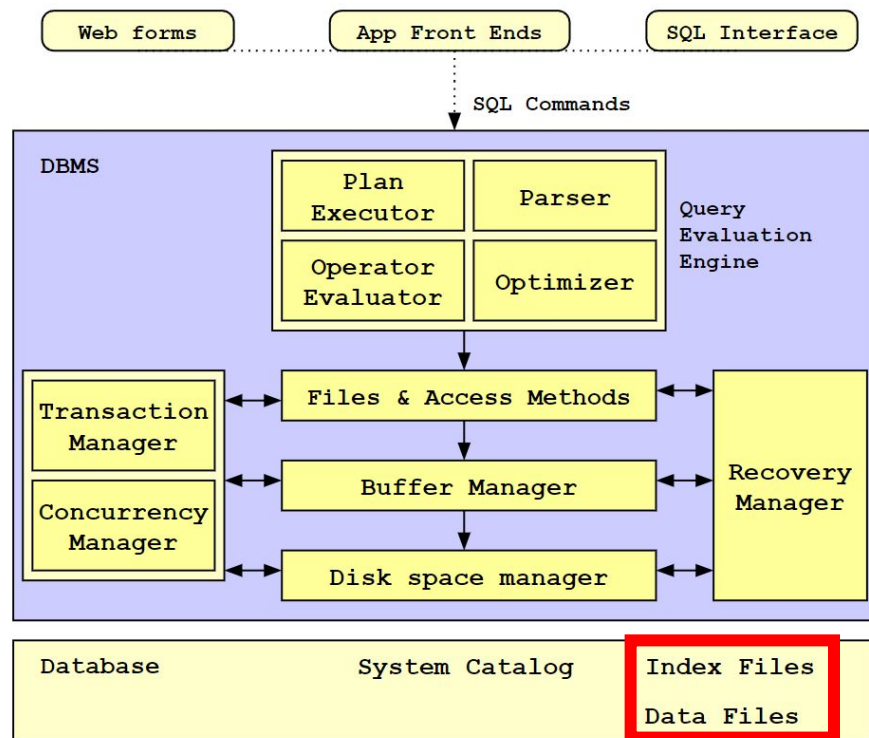
A-217	Brighton	750
A-101	Downtown	500
A-1 10	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Consulta com Índice

- Mais eficiente porque Índice:
 - Reduz o espaço de procura
 - Em geral é muito mais pequeno que a tabela indexada, podendo estar *cached* em memória



Componentes de um SGBD



Criar Índice em SQL

Create Index (versão abreviada)

```
CREATE [UNIQUE] INDEX name ON table_name [USING method]
    ({column_name | (expression)} [ASC|DESC] [NULLS {FIRST|LAST}] [, ...] )
    [INCLUDE (column_name[, ...])]
    [NULLS [NOT] DISTINCT]
    [WHERE predicate]
```

Exemplo:

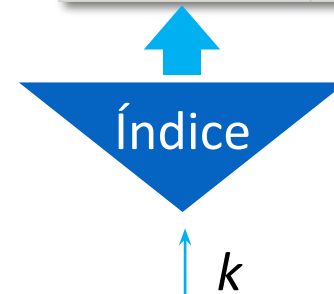
```
CREATE INDEX bal_acc ON account (balance);
```


Classificação de Índices

Índice Ordenado / *Clustering* / Primário

- Indexa a coluna pela qual os dados estão fisicamente ordenados no ficheiro
- Normalmente pretende-se que corresponda à chave primária
- Por defeito, no PostgreSQL os dados estão fisicamente ordenados pela ordem com que são inseridos
 - É preciso reordenar (*cluster*) a tabela pela chave primária para obter um índice primário

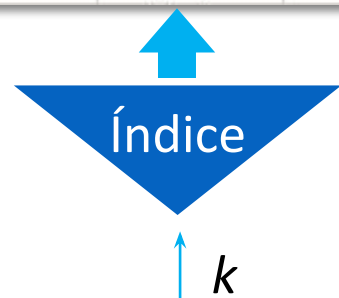
🔑 account_number	branch_name	balance
A-101	Downtown	500.00
A-102	Perryridge	400.00
A-201	Perryridge	900.00
A-215	Mianus	700.00
A-217	Brighton	750.00
A-222	Redwood	700.00
A-240	Downtown	600.00
A-305	Round Hill	350.00
A-333	Central	850.00
A-444	North Town	625.00
A-500	Perryridge	700.00



Índice Não-Ordenado / Secundário

- Indexa uma coluna que não aquela pela qual os dados estão ordenados
- Especifica uma ordem diferente da da tabela
- Se os dados não estiverem ordenados pela chave primária, então o índice para a chave primária é secundário (é o caso defeito no PostgreSQL)

🔑 account_number	branch_name	balance
A-101	Downtown	500.00
A-102	Perryridge	400.00
A-201	Perryridge	900.00
A-215	Mianus	700.00
A-217	Brighton	750.00
A-222	Redwood	700.00
A-240	Downtown	600.00
A-305	Round Hill	350.00
A-333	Central	850.00
A-444	North Town	625.00
A-500	Perryridge	700.00



Ordenar Tabela por Índice em SQL

Cluster Table

```
CLUSTER table_name [USING index_name]
```

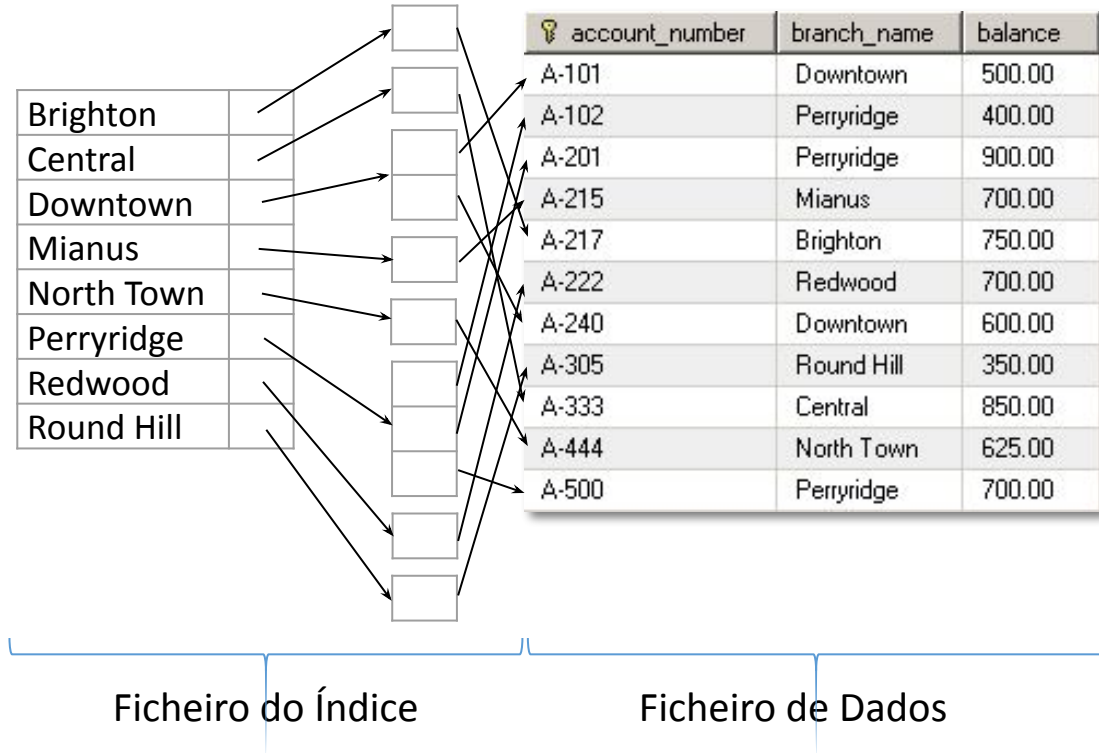
Exemplo:

```
SELECT * FROM account;
account_number | branch_name | balance
-----+-----+-----
A-201          | Uptown      | 900.0000
A-215          | Metro       | 600.0000
A-217          | University  | 650.0000
A-222          | Central     | 550.0000
A-305          | Round Hill  | 800.0000
A-333          | Central     | 750.0000
A-444          | Downtown   | 850.0000
A-101          | Downtown   | 400.0000
A-102          | Uptown      | 800.0000
```

CLUSTER
account
USING
pk_account;

```
SELECT * FROM account;
account_number | branch_name | balance
-----+-----+-----
A-101          | Downtown   | 400.0000
A-102          | Uptown     | 800.0000
A-201          | Uptown     | 900.0000
A-215          | Metro      | 600.0000
A-217          | University | 650.0000
A-222          | Central    | 550.0000
A-305          | Round Hill | 800.0000
A-333          | Central    | 750.0000
A-444          | Downtown   | 850.0000
```

Índice Não-Ordenado / Secundário



Ficheiros: Armazenamento Físico

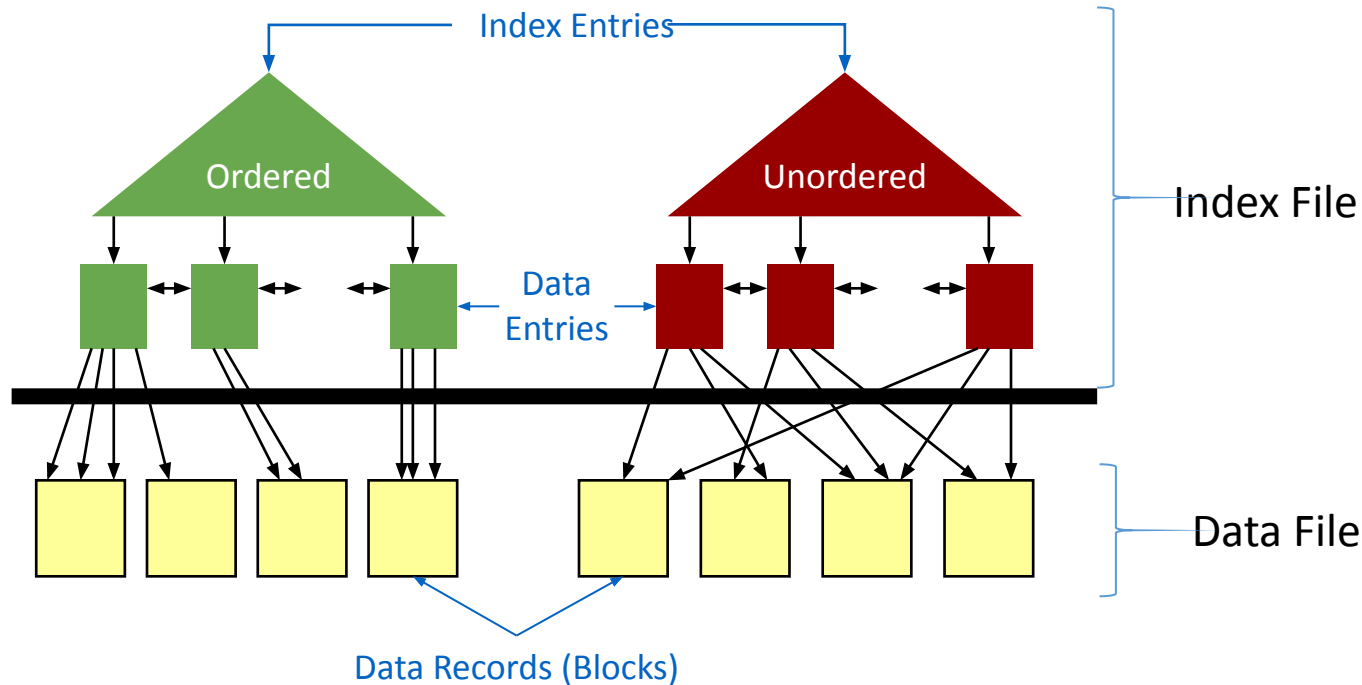
- Acesso ao dispositivo físico de armazenamento de dados faz-se não endereço a endereço, mas página a página (bloco a bloco)
 - Valor típico: 4 kbytes
- Válido tanto para HDD como para SSD



Ficheiros Usados Pelos SGBD

Tipo	Organização dos Registos
Heaps	Registos depositados no 1º espaço livre
Sorted Files	Registos ordenados por valor de k
Hashed Files	Ficheiro organizado em “buckets”; registos dispersos pelos buckets bucket $b = \text{hash}(k)$

Índice Ordenado vs. Desordenado



Índice Denso vs. Esparso

- Os índices **ordenados** podem ser:

Densos

10101	→	10101	Srinivasan	Comp. Sci.	65000	
12121	→	12121	Wu	Finance	90000	
15151	→	15151	Mozart	Music	40000	
22222	→	22222	Einstein	Physics	95000	
32343	→	32343	El Said	History	60000	
33456	→	33456	Gold	Physics	87000	
45565	→	45565	Katz	Comp. Sci.	75000	
58583	→	58583	Califieri	History	62000	
76543	→	76543	Singh	Finance	80000	
76766	→	76766	Crick	Biology	72000	
83821	→	83821	Brandt	Comp. Sci.	92000	
98345	→	98345	Kim	Elec. Eng.	80000	

Há pelo menos uma entrada de dados por cada valor da chave de pesquisa

Esparsos

10101	→	10101	Srinivasan	Comp. Sci.	65000	
32343	→	12121	Wu	Finance	90000	
76766	→	15151	Mozart	Music	40000	
		22222	Einstein	Physics	95000	
		32343	El Said	History	60000	
		33456	Gold	Physics	87000	
		45565	Katz	Comp. Sci.	75000	
		58583	Califieri	History	62000	
		76543	Singh	Finance	80000	
		76766	Crick	Biology	72000	
		83821	Brandt	Comp. Sci.	92000	
		98345	Kim	Elec. Eng.	80000	

Há várias entradas de dados entre valores da chave de pesquisa (tipicamente cada chave aponta para um bloco)
A pesquisa é sequencial entre chaves

Índice Denso vs. Esparso

- Apenas índices **ordenados** podem ser **esparsos**
- Índices não-ordenados têm sempre que ser densos, uma vez que não é possível pesquisa sequencial entre índices

Índice denso em
account_number
(não ordenado)

A-101	
A-102	
A-201	
A-215	
A-217	
A-222	
A-240	
A-305	
A-333	
A-444	
A-500	

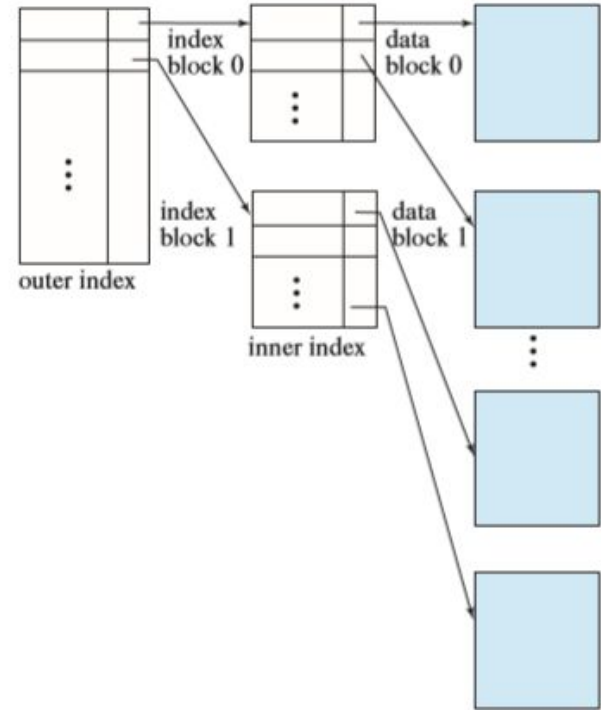
🔑 account_number	branch_name	balance
A-217	Brighton	750.00
A-333	Central	850.00
A-101	Downtown	500.00
A-240	Downtown	600.00
A-215	Mianus	700.00
A-444	North Town	625.00
A-500	Perryridge	700.00
A-201	Perryridge	900.00
A-102	Perryridge	400.00
A-222	Redwood	700.00
A-305	Round Hill	350.00

Brighton	
Mianus	
Perryridge	

Índice esparso
em branch_name
(ordenado)

Índice Multinível

- Num índice multinível há pelo menos duas camadas de indexação
 - Os apontadores da camada externa em vez de apontarem para (blocos de registos de) dados, apontam para (blocos contendo) chaves do índice da camada seguinte



Índices de Chave Composta

- A chave de um índice pode ser composta, i.e., indexar não um atributo único mas um conjunto de dois ou mais atributos
- Índices compostos podem ser vantajosos para consultas que requerem várias comparações, e.g.:

```
SELECT account_number  
FROM account  
WHERE branch_name = "Perryridge" AND balance = 1000;
```

Índices de Chave Composta

```
SELECT account_number  
FROM account  
WHERE branch_name = "Perryridge" AND balance = 1000;
```

- Estratégias possíveis de indexação:
 - Criar índice para *branch_name* e testar valor de *balance*
 - Criar índice para *balance* e testar valor de *branch_name*
 - Criar índices para *branch_name* e *balance* e intersectar os resultados
 - Criar índice composto para *branch_name* e *balance*

Índices de Chave Composta

```
CREATE INDEX bran_bal_acc ON account (branch_name,balance);
```

- São pesquisados por ordem lexicográfica na concatenação dos valores dos atributos:
 - $(branch_1, balance_1) < (branch_2, balance_2)$ se:
 - $(branch_1 < branch_2)$
ou
 - $(branch_1 = branch_2)$ e $(balance_1 < balance_2)$

Índices de Chave Composta

Índice:

branch_name, balance

Central	550.0000	
Central	750.0000	
Downtown	400.0000	
Downtown	850.0000	
Metro	600.0000	
Round Hill	800.0000	
University	650.0000	
Uptown	800.0000	
Uptown	900.0000	

account_number	branch_name	balance
-----+-----+-----		
A-101	Downtown	400.0000
A-102	Uptown	800.0000
A-201	Uptown	900.0000
A-215	Metro	600.0000
A-217	University	650.0000
A-222	Central	550.0000
A-305	Round Hill	800.0000
A-333	Central	750.0000
A-444	Downtown	850.0000

Índice:

balance, branch_name

400.0000	Downtown	
550.0000	Central	
600.0000	Metro	
650.0000	University	
750.0000	Central	
800.0000	Round Hill	
800.0000	Uptown	
850.0000	Downtown	
900.0000	Uptown	

Custo de Índices

Custos de Índices

- Índices aceleram consultas, mas têm custos:
 - De armazenamento (geralmente pequenos por comparação com os ficheiros de dados, mas nem sempre negligíveis)
 - De desempenho em operações de escrita (delete, insert, update)
 - É preciso atualizar não só a(s) tabela(s) das operações mas também todos os índices sobre elas

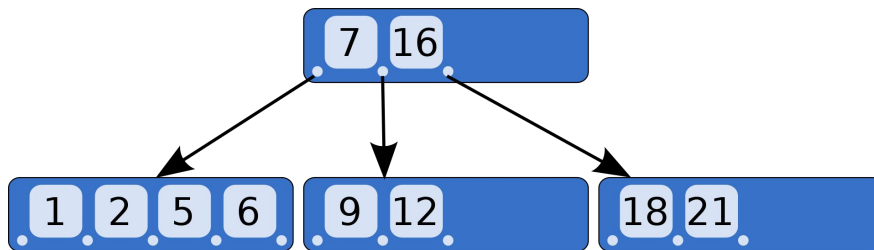
Atualização de Índices

- Remoção:
 - Se o registo a remover era o único com o valor indexado, remove-se a respectiva entrada do índice; remove-se sempre o apontador para o registo
- Inserção:
 - Se o novo valor não estiver indexado, inserir nova entrada no índice; se estiver indexado, atualizar apontadores do índice
- Atualização:
 - Combinação de remoção e inserção

Métodos/Tipos de Índice

Índices B-tree

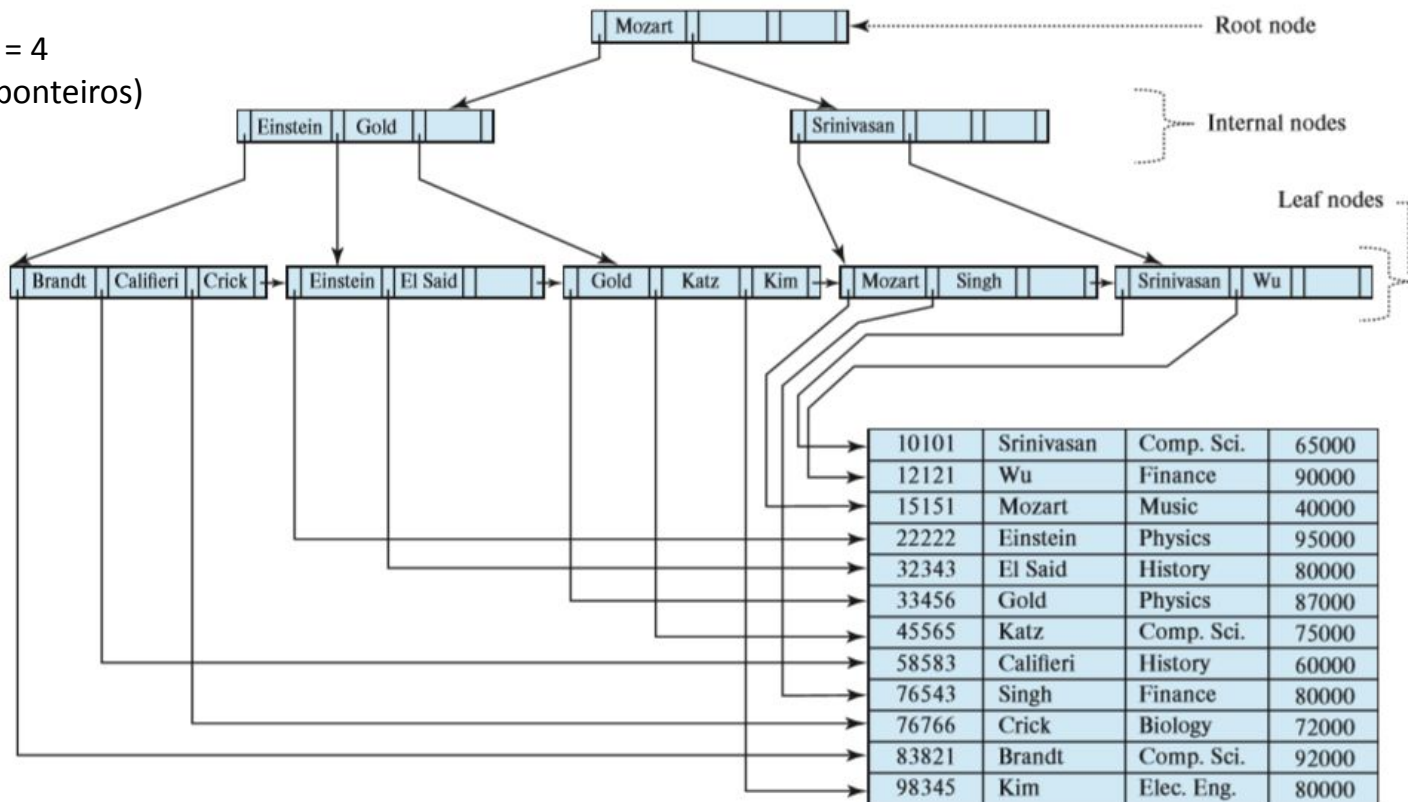
- Método de índice por defeito no PostgreSQL, por ser o mais frequente
- B-tree: uma estrutura de dados em árvore auto-balanceada que mantém dados ordenados
 - Generalização da árvore de procura binária, em que cada nó pode ter n filhos
- Ideal para consultas de *range* ($>$, $<$ ou intervalo) mas também adequada para igualdade



Índices B-tree

B-tree com $n = 4$
(cada nó tem até 4 ponteiros)

As folhas do índice
estão sempre
ordenadas



Índices B-tree

Configuração:

- Geralmente um nó é do tamanho de um bloco (tipicamente 4KB)
- Numa tabela com N registos, a procura na Btree obriga à leitura de no máximo de $\log_{[n/2]}(N)$ nós do índice.
- Se cada entrada no índice (valor, apontador) ocupa 40 bytes, escolhe-se $n \approx 100$
- Com 1 milhão de valores de chave e $n=100$ acede-se a apenas $\log_{50}(1,000,000) = 4$ nós para encontrar qualquer valor
- Se a árvore fosse binária seriam precisos 20 nós

Índices B-tree

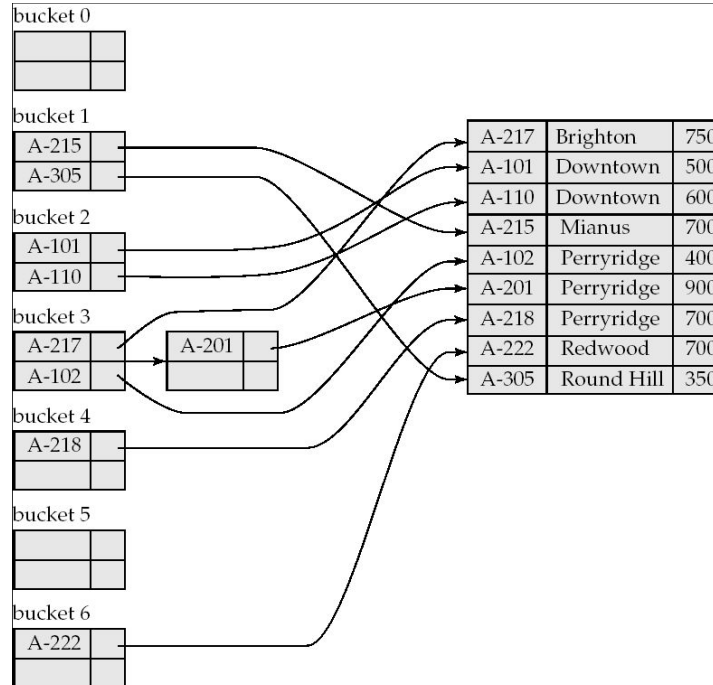
Vantagens:

- Com ligações feitas por apontadores, não é necessário ter os blocos fisicamente juntos no disco
- Partindo da raiz, é possível chegar rapidamente a qualquer folha, mesmo que o número de folhas seja muito grande
- Se houver inserções ou remoções de registos na tabela, o índice pode ser actualizado apenas com pequenas alterações locais

Índices Hash (Dispersão)

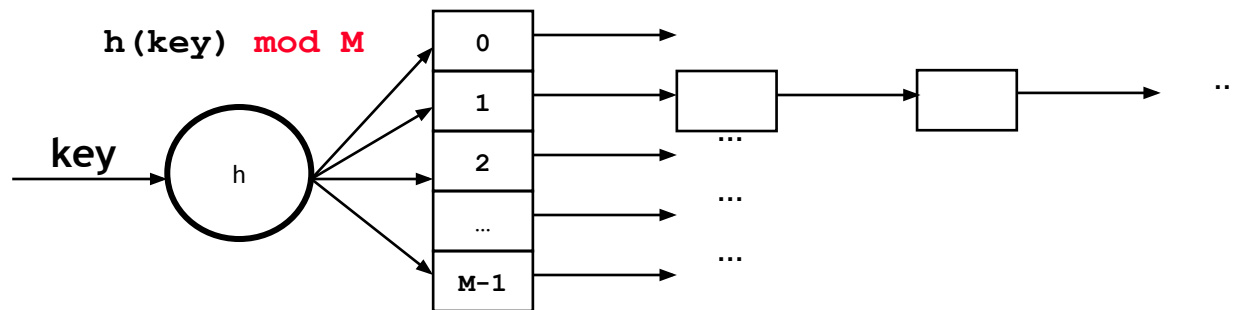
- Um contentor armazena um conjunto de entradas (valor, apontador(es))
 - Tipicamente um contentor ocupa um bloco em disco
- Função de dispersão (hash) recebe como parâmetro um valor e determina o contentor, mas no mesmo contentor podem haver diferentes valores
- Dentro dos contentores, as entradas são pesquisadas sequencialmente
- São os melhores para seleção por igualdade (e.g. joins)
- Não suportam consultas de *range* ($>$, $<$ ou intervalo)

Índices Hash (Dispersão)



Índices Hash (Dispersão)

- Dispersão pode ser:
 - Estática: nº de contentores inalterado uma vez criado o índice
 - Dinâmica: nº de contentores varia ao longo do ciclo de vida do índice



Primary bucket pages Overflow pages

Índices Hash (Dispersão)

Desafios:

- Se o número de contentores for pequeno, poderá ser necessário contentores extra (overflow)
- Se o número de contentores for demasiado grande, haverá desperdício de espaço
- Os sistemas actuais usam hashing dinâmico, em que o número de contentores varia dinamicamente
 - Exige mudar a função de hash e reorganizar o índice quando o número de contentores se altera

Índices Bitmap

- Um índice bitmap sobre um atributo tem um bitmap sobre cada valor do atributo
- Cada bitmap tem tantos bits como registos
- Respostas às consultas são obtidas com operações lógicas sobre bitmaps (AND, OR, NOT)
- São especialmente vantajosos em colunas de cardinalidade baixa (i.e., número de valores proporcionalmente baixo) para suportar interrogações sobre vários atributos

Índices Bitmap

- Cada operação usa 2 bitmaps do mesmo tamanho e obtém resultado num outro bitmap

- $100110 \text{ AND } 110011 = 100010$
- $100110 \text{ OR } 110011 = 110111$
- $\text{NOT } 100110 = 011001$

- E.g. homens com rendimento L1:

$10010 \text{ AND } 10100 = 10000$

- Ficam identificados os tuplos que satisfazem a condição
- Contagem do nº de tuplos no resultado ainda mais imediata

record number	ID	gender	income_level
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m

10010

f

01101

Bitmaps for *income_level*

L1

10100

L2

01000

L3

00001

L4

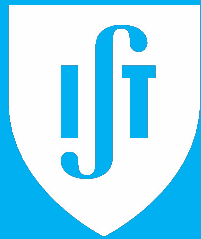
00010

L5

00000

Índices Suportados pelo PostgreSQL

- **B-tree:** método defeito; único que suporta UNIQUE indexes
 - Criados automaticamente para PRIMARY KEY ou UNIQUE constraints
- **Hash:** não suportam índices de chave composta
- **BRIN** = Block Range Index: desenhados para tabelas grandes onde uma coluna está correlacionada com a localização física dos dados (e.g. date)
- **GIN** = Generalized Inverted Index: semelhantes a hash mas suportando chaves composta
- **GiST** = Generalized Search Tree: para desenvolvimento
- **SP-GiST** = space-partitioned GiST: para desenvolvimento



TÉCNICO LISBOA