

# BASE DE DADOS



**Índices**

Teóricas  
Ano Lectivo 2018/2019  
Rosa Reis

## Índices

Search Key	Apontador
index table	

- **Índices** ou **estruturas de acesso secundarias** são estruturas de dados auxiliares que visam minimizar o tempo de acesso a registos em resposta a operações de procura sobre determinados atributos.
- **Índices em bases de dados são listas de valores (index key) e de apontadores guardados numa estrutura de memória (index table), apontando para a localização física da informação, nos ficheiros de dados associados às tabelas**
- Os índices são construídos tendo por base os valores de um dos atributos (**atributo de indexação**).
  - Qualquer atributo pode ser utilizado para construir um índice.
  - Diferentes atributos podem ser utilizados para construir múltiplos índices.
- Os índices não alteram a disposição física dos registos nos blocos em disco.

## Índices

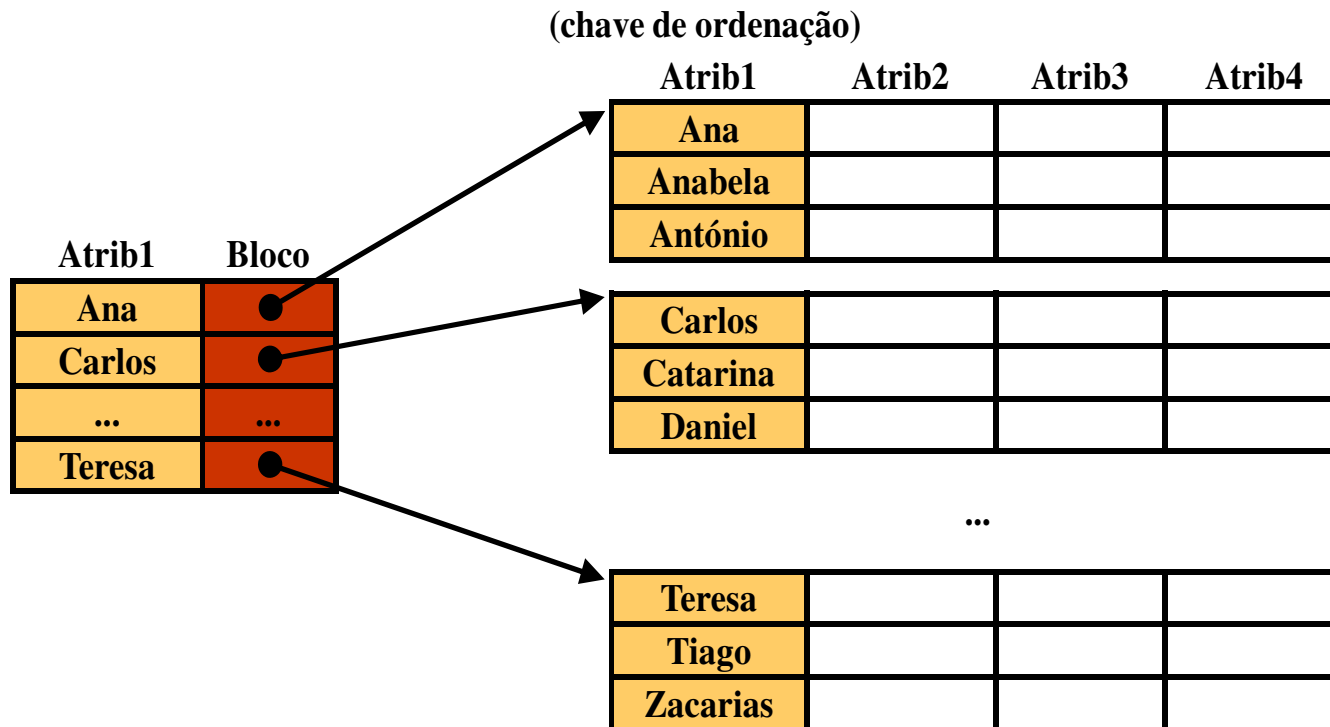
- Os SGBD criam automaticamente índices para certos tipos de constraints (PRIMARY KEY e UNIQUE)
- **Os benefícios dos índices têm custos**
  - **É maior o espaço alocado** na base de dados
  - **Instruções de inserção (insert) e eliminação (delete) têm maior tempo de processamento** devido às **respectivas operações de manutenção do índice**
  - **Instruções de atualização (update) podem ter maior tempo de processamento** devido às **respectivas operações de manutenção do índice**, se a atualização **incidir sobre pelo menos uma das colunas do índice (index key)**

## Índice Primário e Secundário

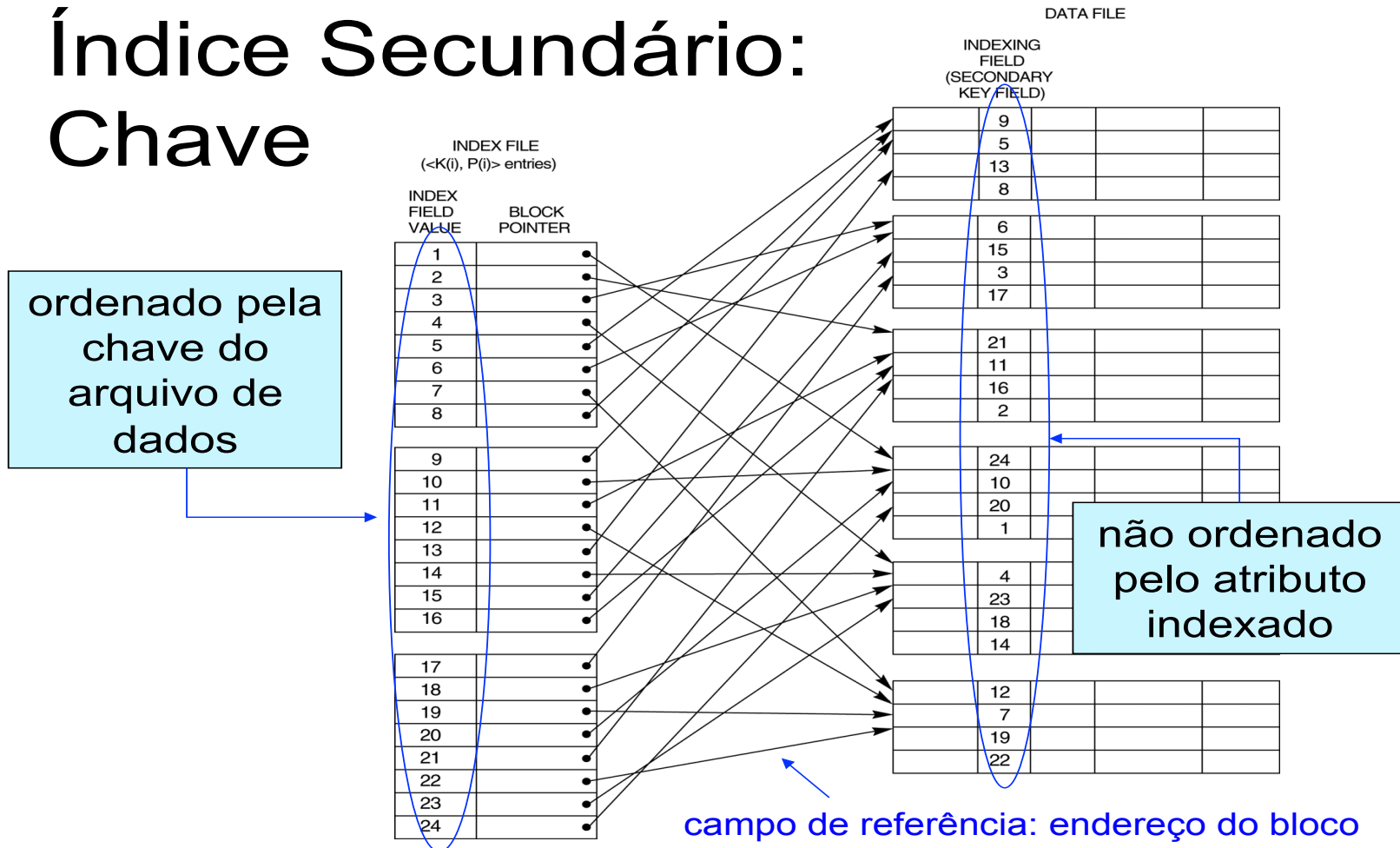
- Se a chave de pesquisa contém a chave primária diz-se que o **índice é primário**
- Os outros índices dizem-se **secundários**
- Duas entradas de dados no índice são duplicadas se tiverem o mesmo valor para a chave de pesquisa associada com o índice
  - Índice primário é garantido não conter duplicados
  - Em geral, índice secundário contém duplicados
- **Índice Único** (Unique): Chave de pesquisa contém chave candidata

## Índice Primário

Baseado na chave de ordenação



## Índice Secundário: Chave

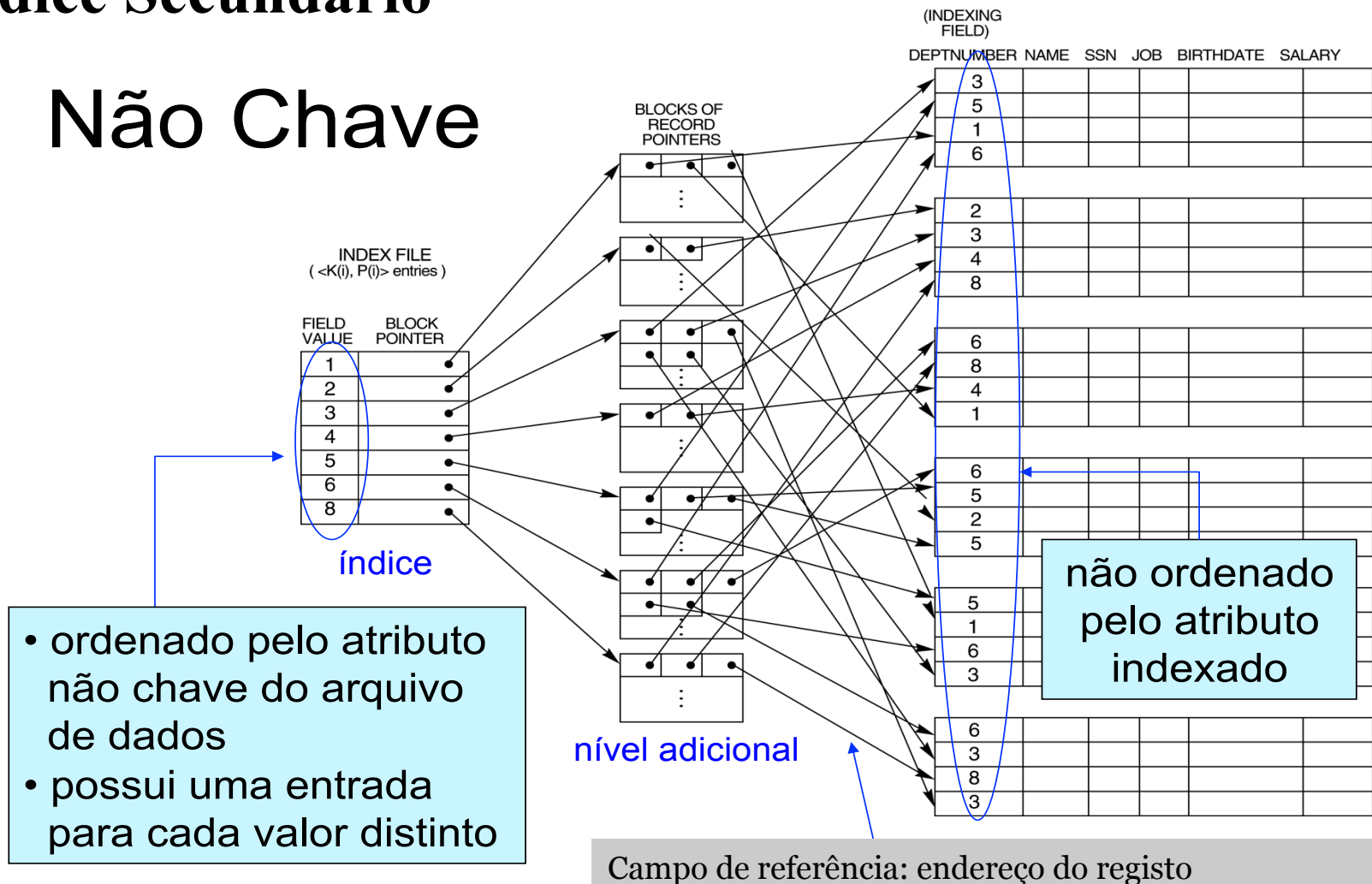


baseado em qualquer atributo não ordenado de um ficheiro

## Índice Secundário

### Não Chave

DATA FILE



- ordenado pelo atributo não chave do arquivo de dados
- possui uma entrada para cada valor distinto

## Índice Denso e Esparso

### Densos:

se há pelo menos uma entrada de dados por cada valor da chave de pesquisa  
(em algum registo de dados)

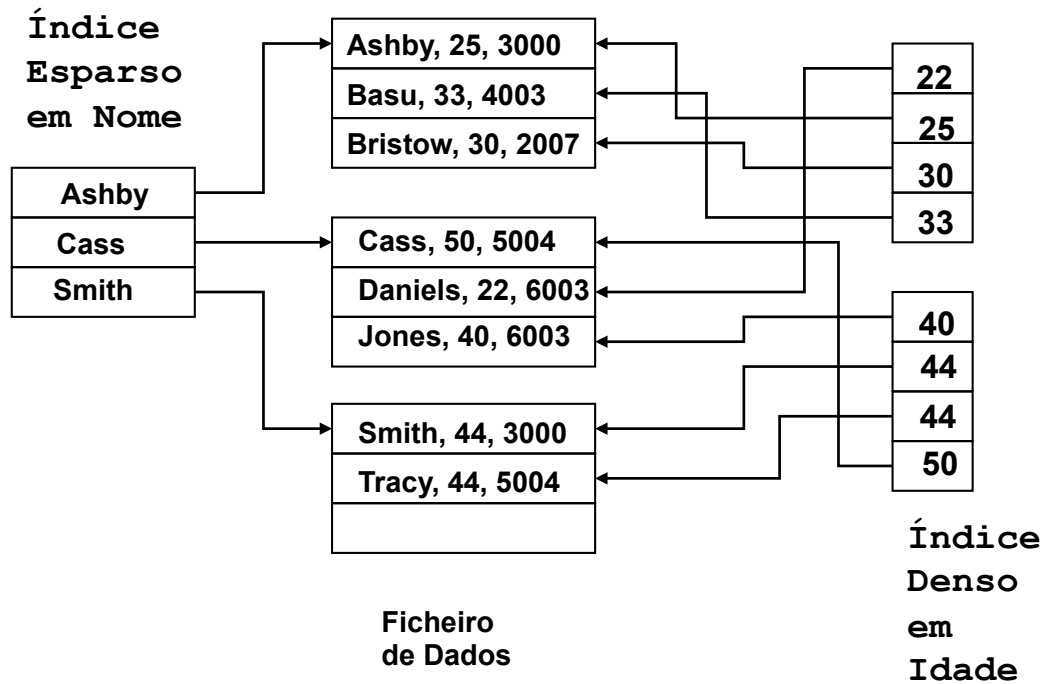
### Esparsos:

se existe uma entrada de dados no índice apenas para alguns valores que a  
chave de procura pode tomar no ficheiro de dados

- Um índice esperso é sempre agrupado
- Índices esparsos são menores; contudo algumas optimizações úteis são baseadas em índices densos

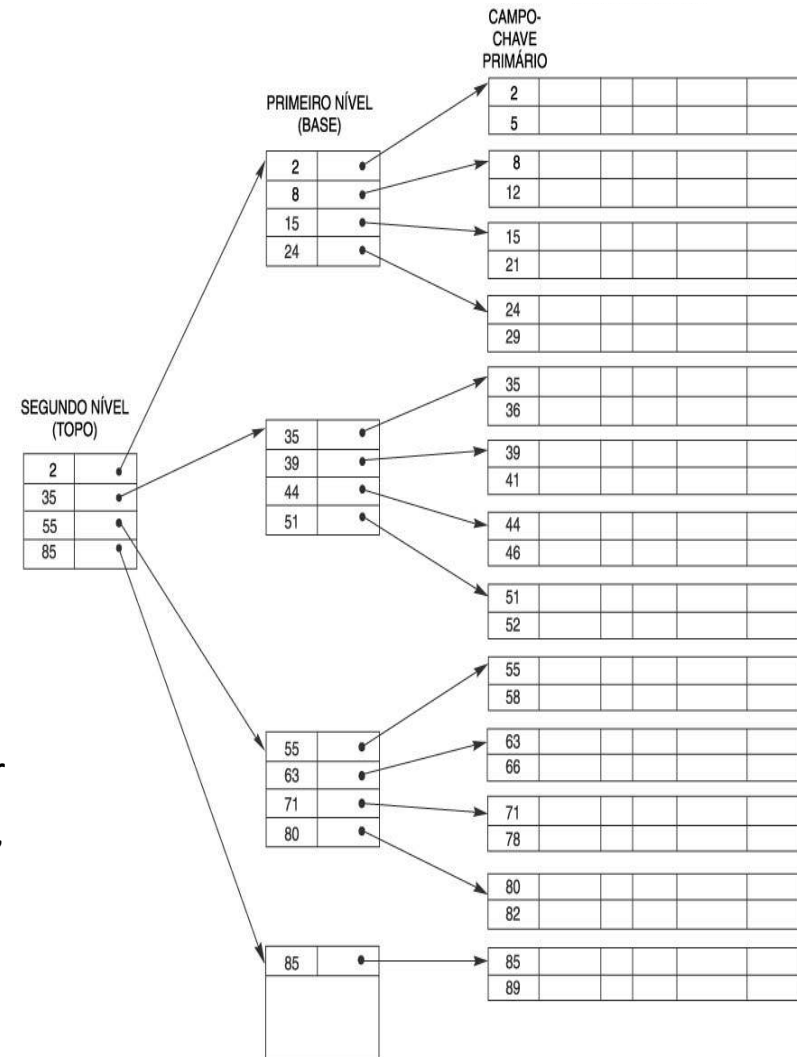


## Índice Denso e Esparso



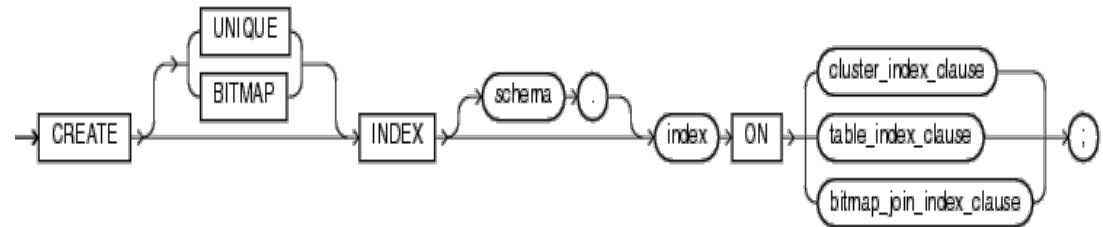
## Índice Multinível

- ❑ Se um índice primário não couber todo na memória, o acesso complica-se
  - para reduzir o número de acessos ao disco, **trata-se o índice como um ficheiro sequencial e**
  - **cria-se um índice esperso para o índice primário**
  - se este for demasiado grande para caber em memória, cria-se **ainda outro nível**
- ❑ Um índice de níveis múltiplos pode ser criado a partir de qualquer tipo de índice de primeiro nível (primário, secundário).
- ❑ Os índices têm que ser mantidos atualizados em todos os níveis



## Indices Oracle

### ➤ Criar Índices



```
CREATE [ UNIQUE | BITMAP ] INDEX [ schema. ]index  
ON { cluster_index_clause  
    | table_index_clause  
    | bitmap_join_index_clause  
};
```

- Os índices estão subdivididos em 4 grupos:
  - B<sup>+</sup> Tree (Reverse Key)
  - Function Based
  - Bitmap
  - Application domain

## B TREE

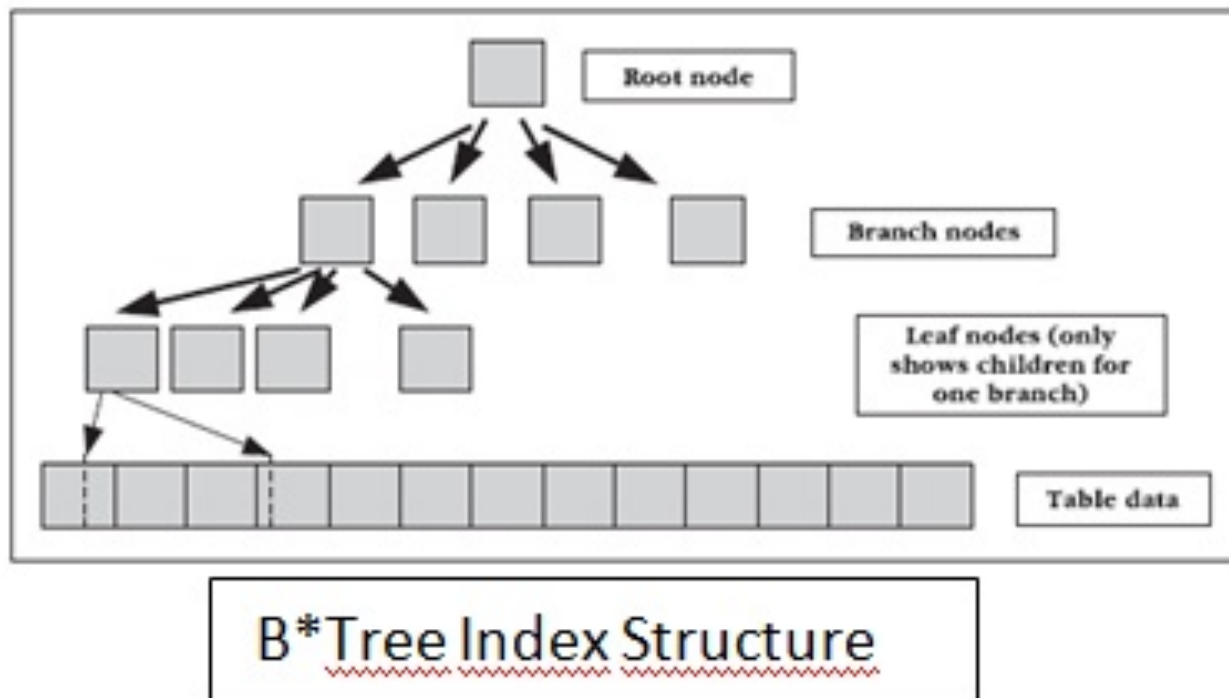
- É considerado o índice padrão- através dele, o Oracle gere corretamente os bloco de dados, controlando a alocação dos ponteiros dentro de cada bloco.
- A criação deste índice é **recomendada** quando se torna necessária a execução de **consultas de colunas com alta cardinalidade** (alto grau de distintos valores) .
  - Como por exemplo, colunas em que há unicidade de valores, como as chaves primárias de uma tabela.

```
-- criando um indice btree na coluna NM_EMAIL da tabela ECOMMERCE.CLIENTE
CREATE INDEX ECOMMERCE.IX_CLIENTE_NMEMAIL ON ECOMMERCE.CLIENTE(NM_EMAIL);

EXPLAIN PLAN FOR
  SELECT  CD_CLIENTE, NM_CLIENTE, DT_NASCIMENTO
  FROM    ECOMMERCE.CLIENTE
  WHERE   NM_EMAIL = 'sagko@com.br';
SELECT * FROM TABLE(dbms_xplan.DISPLAY);
```

## B<sup>+</sup> TREE

- Tem uma arquitetura de árvore para encadeamento das informações.



**Leaf Nodes:** contêm entradas que se referem diretamente as linhas das tabelas;

**Branch Nodes:** contêm entradas que se referem a “Leaf Nodes”; e

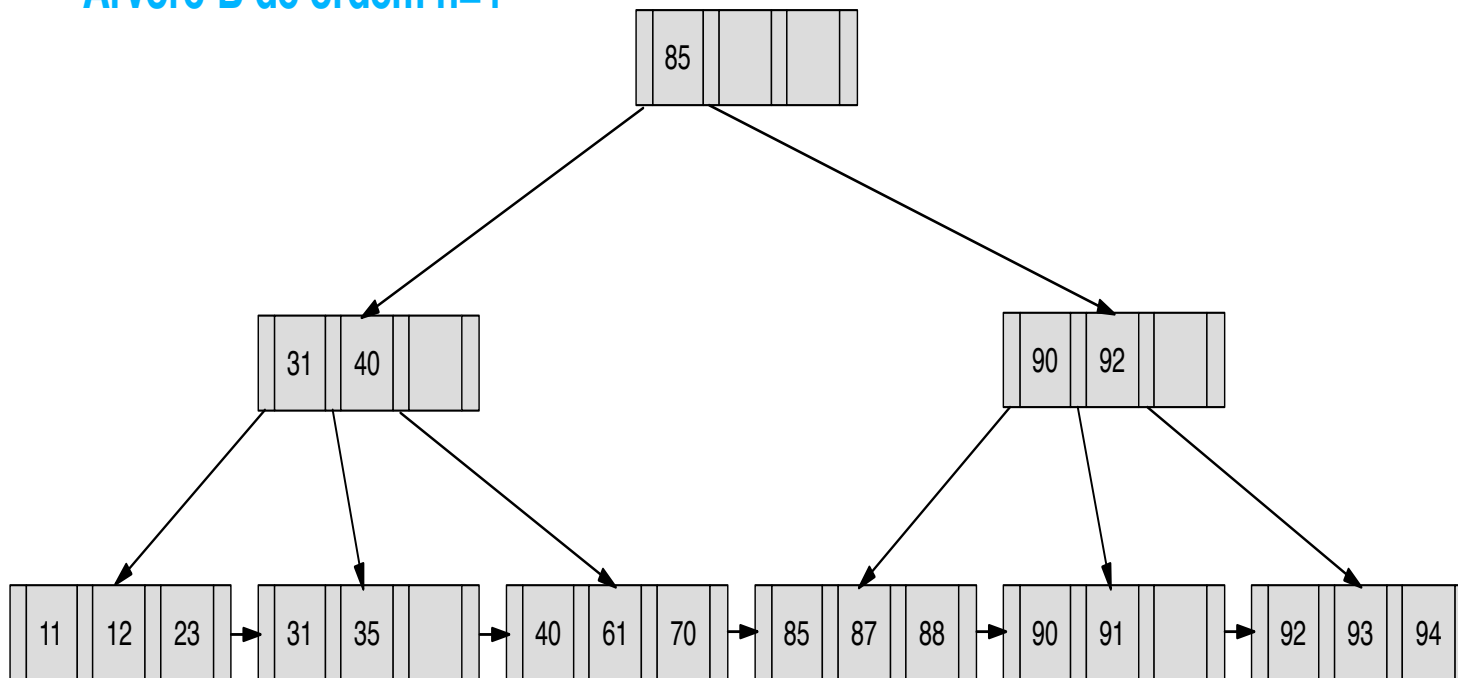
**Um único Root Node:** que é um “Ramo”, o topo da estrutura da árvore

## B<sup>+</sup> TREE

- **Árvores B** armazenam as chaves de pesquisa sem redundância e uma em cada nó, sendo possível encontrar a informação durante o caminho, sem a necessidade de chegar até o nó folha.
- **Árvore B<sup>+</sup>** tem a forma balanceada da raiz até as folhas
  - todo caminho da raiz até a folha é igual e previsível ( $n/2$ ) onde  $n$  é o número de registos da tabela.
  - endereço do registo está sempre em um nó folha, ou seja é necessário percorrer toda a árvore (a raiz até a folha ) para identificar um registo.
  - O B<sup>+</sup> indica um “balanceamento” de carga, o que permite a igualdade de tempo para cada registo solicitado.
  - Inserção e remoção são difíceis devido às constantes necessidades de “splits” no nó folha afim de manter a árvore balanceada.

## B<sup>+</sup> TREE

Árvore-B de ordem n=4

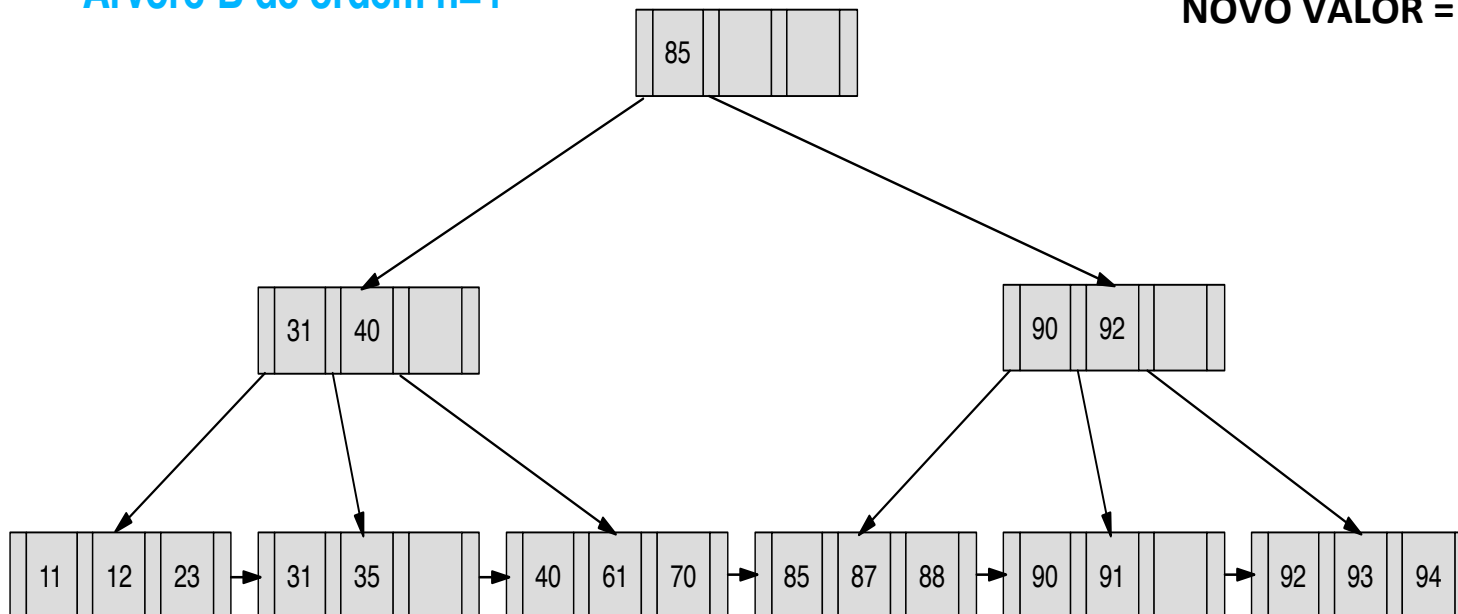


ROWID = '00000DC5.0000.0001

## Índices em Arvore B<sup>+</sup> - Inserção

Árvore-B de ordem  $n=4$

NOVO VALOR = 15

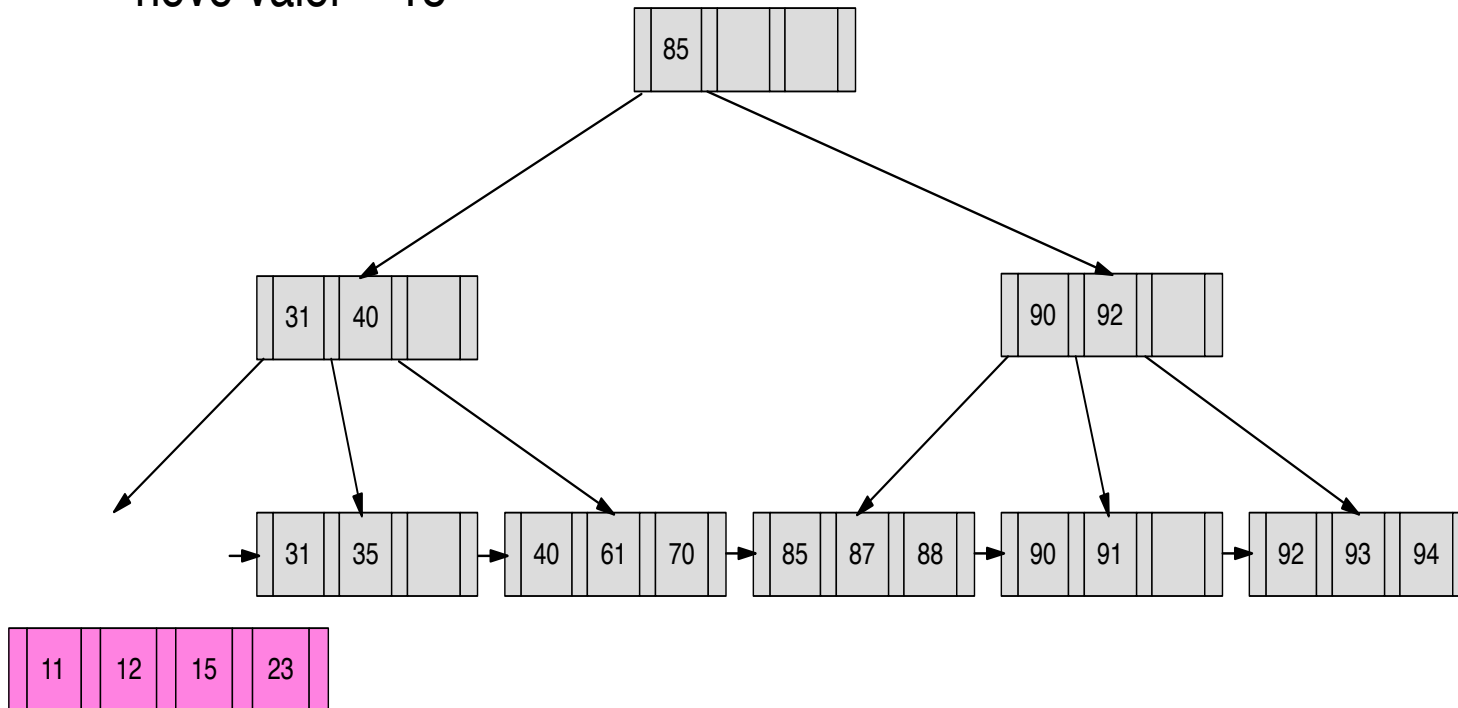




## Índices em Arvore B<sup>+</sup> - Inserção

### ■ Inserções

► novo valor = 15

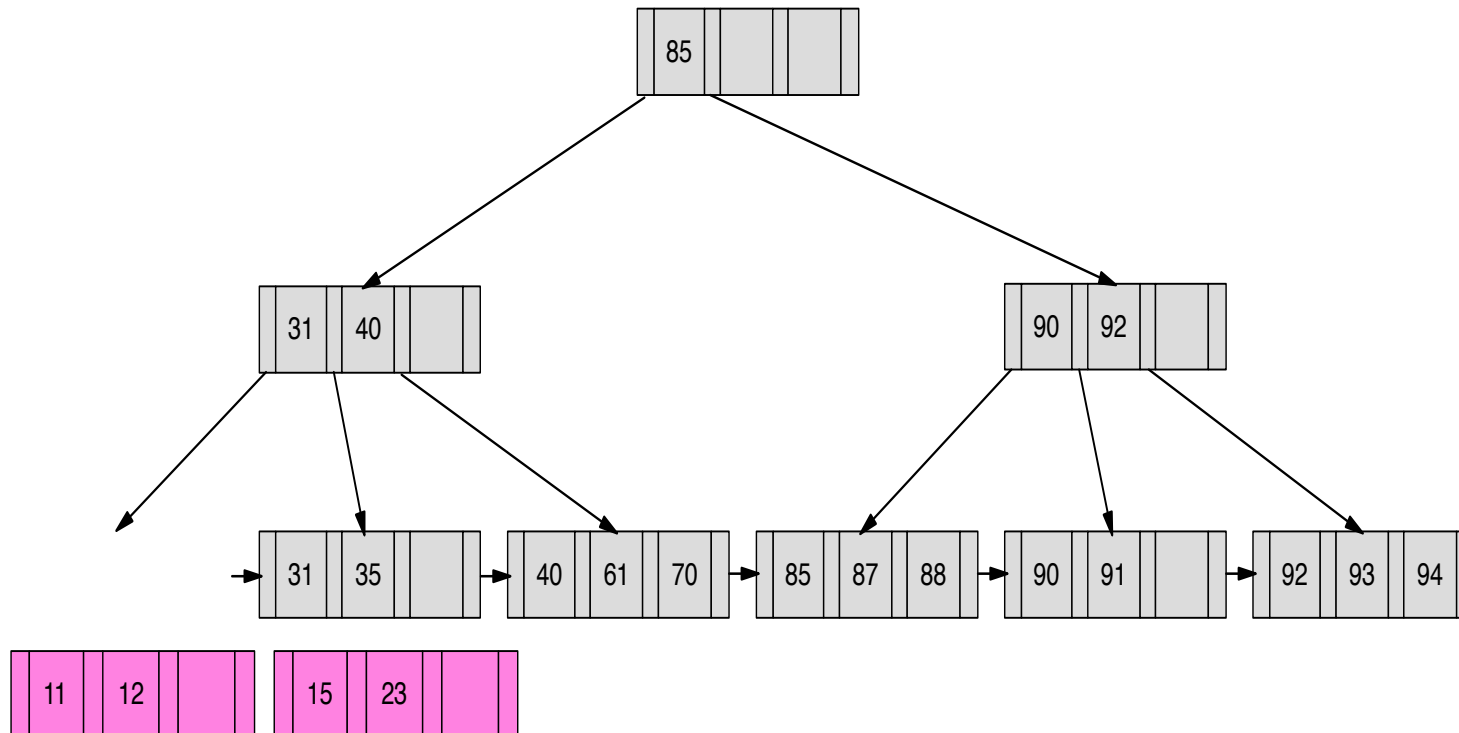


insere-se o novo valor 15 na folha correta, que neste caso ficará grande demais

## Índices em Arvore B<sup>+</sup> - Inserção

### ■ Inserções

► novo valor = 15

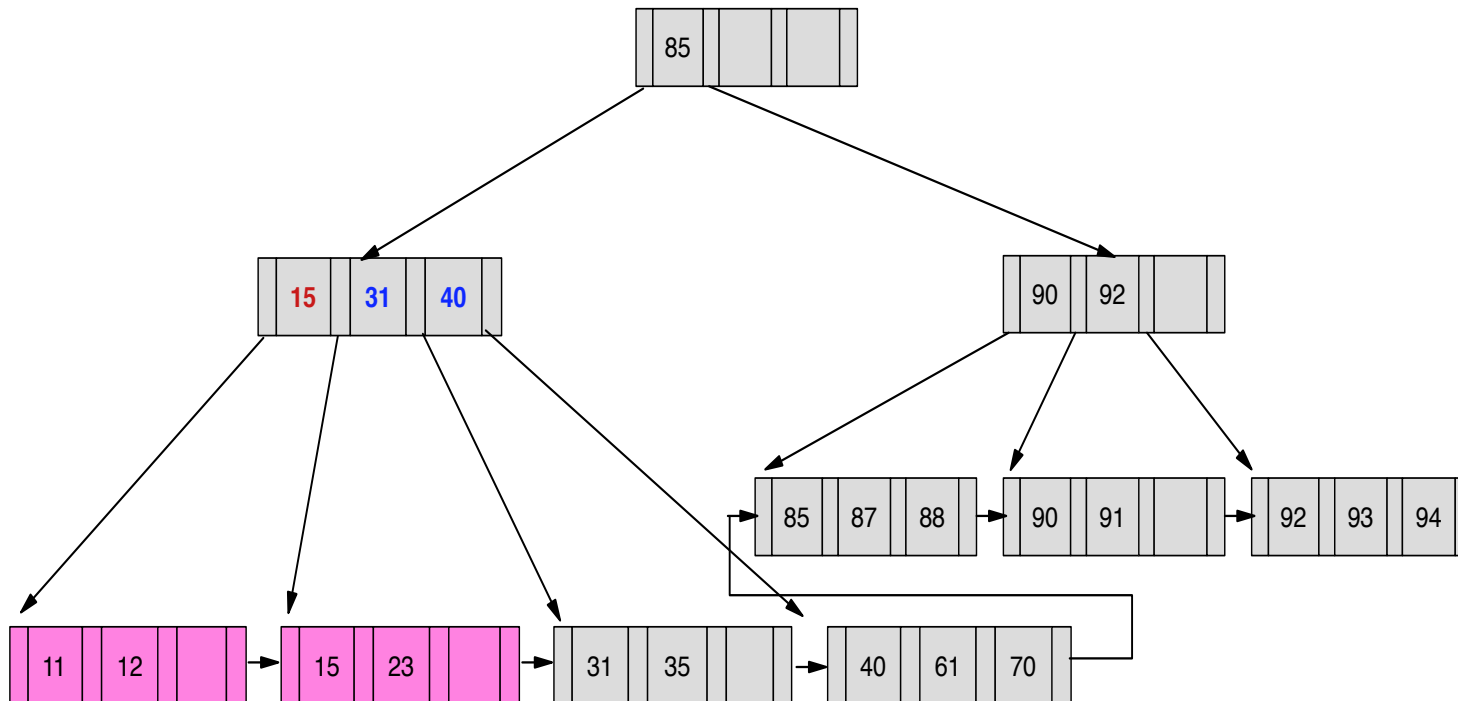


a folha é então quebrada em duas

## Índices em Arvore B<sup>+</sup> - Inserção

### ■ Inserção

► novo valor = 15



primeiro valor da folha da direita, que por acaso é 15, é inserido no pai, que passa a ter 1 novo filho

## Function Based:

```
create index emp_upper_idx on emp(upper(ename));
```

- Criado quando é necessário usar função(ões) na(s) coluna(s) do(s) filtro(s).
- O Oracle analisa sempre a cláusula where da instrução SQL para ver se existe um índice correspondente.
- Usando índices baseados em função, o designer do Oracle pode criar um índice que corresponda exatamente aos predicados dentro da cláusula where do SQL. Isso garante que a consulta seja recuperada com uma quantidade mínima de E/S de disco e a velocidade mais rápida possível.

```
select ename, empno, sal from emp where upper(ename) = 'KING';
```

## Bitmap e Bitmap Join

- Usados em Data Warehouse;
- pode ser usado fora do data warehouse,
  - o índice Bitmap Join foi projetado especificamente para ser usado dentro do esquema em estrela, que é um modelo de dados criado para o ambiente do data warehouse.

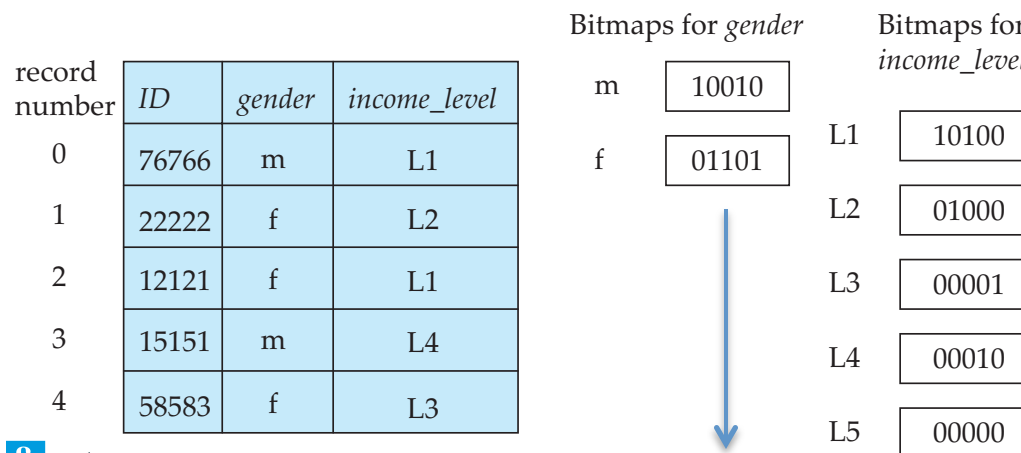
### Vantagens:

- podem ser criados muito rapidamente
- ocupam muito menos espaço do que um índice B-Tree.
- O desempenho da consulta é auxiliado por índices de bitmap, pois eles podem ser verificados rapidamente, porque são menores.

```
create bitmap index normal_empno_bmx on test_normal(empno);
```

## Bitmap

- é recomendado em colunas com baixa cardinalidade. Não sendo recomendada sua utilização em colunas com alterações de dados frequentes;
  - o Oracle não permite a sua criação em chaves primárias
- podem ser utilizados para comparações com null;
- **usa um conjunto de bits para cada valor de chave e uma função de mapeamento que converte cada posição de bit em um rowid**



Tem tantos bits como registros

## Application Domain

- usado para indexar dados não tradicionais, como dados LOB(Large Object) , dados de vídeo e outras colunas não-textuais.
  - em pesquisas de texto ou processamento de imagens.
  - dois índices, o Oracle Text e o Oracle Spatial.
    - O primeiro é muito recomentado na otimização de consultas de texto, onde as colunas possuem o tipo VARCHAR2 ou CLOB.

```
1  -- Execute o comando abaixo para criar o indice IX_NO_CLIENTE_OT:
2  create index ECOMMERCE.IX_NO_CLIENTE_OT on ECOMMERCE.CLIENTE(NM_CLIENTE) indextype is ctxsys.context;
3
4  -- Para testar o acesso ao indice, execute:
5  EXPLAIN PLAN FOR
6  SELECT *
7  FROM   ECOMMERCE.CLIENTE
8  WHERE  CONTAINS(NM_CLIENTE, '%ul%') > 0;
9  SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

## Reverse Key

- Objetivo é reduzir a contenção de blocos (esperas de buffer ocupado) no segmento de índice.
  - o índice normal armazena o valor da coluna do índice e o rowid no segmento do índice. Mas,
  - o índice reverse key armazena o valor da coluna invertida e o rowid.
    - depois será distribuído em vários blocos de índice. Isso evita a contenção do bloco de índice.
- **Usar num sistema de instância única**, onde várias sessões estão a tentar inserir/atualizar a coluna do índice ao mesmo tempo, o valor da coluna do índice está a ser extraída de uma sequência.

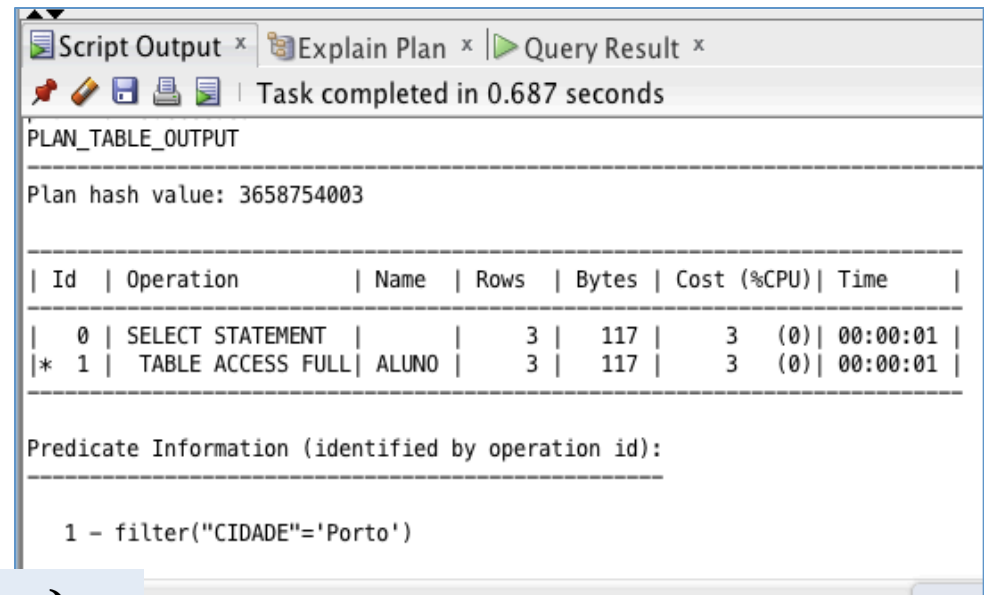
```
CREATE INDEX test_indexi ON test_table (a,b,c) REVERSE;
```



## Explain Plan

- Otimizador do Oracle para determinar o plano de execução mais eficiente das queries;
- Um plano de execução mostra as etapas detalhadas necessárias para executar uma instrução SQL. Essas etapas são expressas como um conjunto de operadores da base de dados que consomem e produzem linhas.

**Explain Plan for**  
Select \* from aluno  
where cidade='Porto';



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	117	3 (0)	00:00:01
* 1	TABLE ACCESS FULL	ALUNO	3	117	3 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("CIDADE"='Porto')

**Select \* from table (dbms\_xplan.display)**

# Índices

Worksheet | Query Builder

```
1  
2 select nome from marinheiros where upper(nome) = 'HORACIO';  
3  
4
```

Script Output x | Query Result x | Explain Plan x

SQL | 0.385 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	3
TABLE ACCESS (FULL)	MARINHEIROS	1	3
Filter Predicates			
UPPER(NOME)='HORACIO'			

```
1  
2 select nome from marinheiros where upper(nome) = 'HORACIO';  
3  
4 create index nomeinx on marinheiros(upper(nome));  
5  
6
```

Script Output x | Query Result x | Explain Plan x

SQL | 0.313 seconds

PERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	2
TABLE ACCESS (BY INDEX ROWID BATCHE	MARINHEIROS	1	2
INDEX (RANGE SCAN)	NOMEINX	1	1
Access Predicates			
UPPER(NOME)='HORACIO'			

## Quando devemos criar Índices

Índices otimizam consultas, mas degradam a performance de instruções DML (insert, update, delete e join), portanto, criar criteriosamente, seguindo algumas das dicas abaixo:

1. Para otimizar consultas frequentes;
2. Nas colunas da cláusula WHERE, ordenação, agrupamentos e ligações;

Colunas que são frequentemente usadas nestas clausulas.

3. Em geral, quando a consulta retorna poucas linhas da(s) tabela(s);
4. Em FKs para otimizar joins e DMLs na tabela pai;

em casos em que um grande número de INSERT, UPDATE e DELETE concorrentes acedem as tabelas pai e filho.

Estes índices permitem que o Oracle modifique os dados na tabela filho sem bloquear a tabela pai.

## Quando devemos criar Índices

6. Quando o filtro não compara valores nulos (NULL), pois eles não estão indexados nos índices mais comuns (B-tree).
7. Colunas pelas quais junções de tabelas são feitas com frequência.
8. Somente colunas de índice com boa seletividade.

A seletividade de um índice é a percentagem de linhas numa tabela que possui o mesmo valor para a coluna indexada.

A seletividade de um índice é boa se algumas linhas tiverem o mesmo valor.

**Nota:** O Oracle cria automaticamente índices para as colunas que fazem parte das chaves primárias e exclusivas que são definidas como restrições de integridade. Esses índices são os mais eficazes para otimizar o desempenho.

## Não usar Índices

- Não indexar colunas com poucos valores diferentes.
  - Geralmente, estas consultas têm pouca seletividade e, portanto, não otimizam o desempenho, a menos que os valores usados com mais frequência sejam aqueles que aparecem com menos frequência na coluna.
- Não indexar colunas que são modificadas com frequência.
  - As instruções UPDATE, INSERT e DELETE que modificam índices demoram mais tempo do que se não o fizessem, uma vez que devem modificar os dados na tabela e no índice.
- Não indexar colunas que só aparecem em instruções WHERE com operadores ou funções (exceto MIN e MAX).

## ORACLE

Access Path	Explanation
Full table scan	Reads all rows from table & filters out those that do not meet the where clause predicates. Used when no index, DOP set etc
Table access by Rowid	Rowid specifies the datafile & data block containing the row and the location of the row in that block. Used if rowid supplied by index or in where clause
Index unique scan	Only one row will be returned. Used when stmt contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed
Index range scan	Accesses adjacent index entries returns ROWID values Used with equality on non-unique indexes or range predicate on unique index (<.>, between etc)
Index skip scan	Skips the leading edge of the index & uses the rest Advantageous if there are few distinct values in the leading column and many distinct values in the non-leading column
Full index scan	Processes all leaf blocks of an index, but only enough branch blocks to find 1 <sup>st</sup> leaf block. Used when all necessary columns are in index & order by clause matches index struct or if sort merge join is done
Fast full index scan	Scans all blocks in index used to replace a FTS when all necessary columns are in the index. Using multi-block IO & can go parallel
Index joins	Hash join of several indexes that together contain all the table columns that are referenced in the query. Won't eliminate a sort operation
Bitmap indexes	uses a bitmap for key values and a mapping function that converts each bit position to a rowid. Can efficiently merge indexes that correspond to several conditions in a WHERE clause