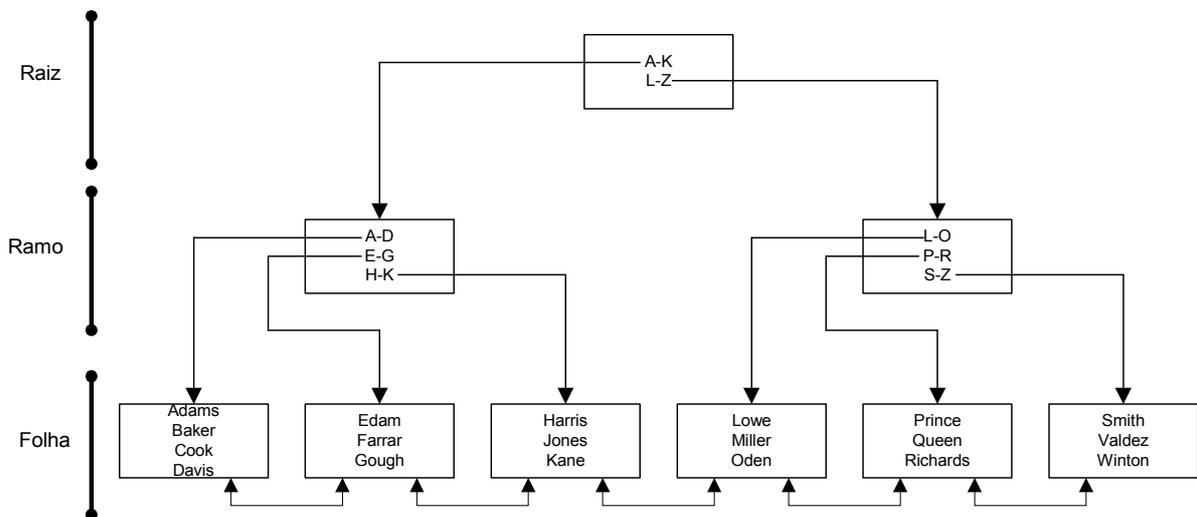


## 1.1 Como funcionam os índices B\*tree?



Os índices B\*tree do Oracle têm uma estrutura em árvore. No topo está o bloco Raiz que contém apontadores para vários blocos Ramo. Estes apontam para blocos Folha ou para outro bloco Ramo, no caso do índice ser muito grande. Os blocos Folha contêm o valor usado no índice, por exemplo uma chave primária, e o respectivo rowid. Este permite a localização directa da linha da tabela que está associada ao valor do índice. Um bloco Folha tem ligações para o próximo bloco Folha e para o anterior. Isto permite percorrer a árvore, por ordem crescente e decrescente, sem grande esforço, o que dá suporte aos queries com resultado ordenado pela coluna que se usou no índice, pesquisas de valores “maior que” ou “menor que” ou pesquisas “entre dois valores”.

Uma pesquisa de um valor através de um índice B\*Tree necessita de 3, no máximo 4, leituras de blocos antes de encontrar a entrada desejada. Depois requer mais uma leitura da linha da tabela via rowid. Se o índice é muito utilizado é provável que o bloco Raiz e os blocos Folha estejam na [SGA](#), o que ainda reduz mais o I/O.

A manutenção do índice quando os dados da tabela são alterados é uma operação com algum peso. Suponha que pretende inserir uma nova entrada num bloco folha e não tem espaço livre. O sistema terá que criar um novo bloco e distribuir as entradas do anterior pelo novo. Depois terá que ser adicionada uma entrada no bloco ramo. Se este também estiver cheio, então teremos que criar um bloco Ramo novo, dividir as entradas pelos dois e actualizar os apontadores no bloco Raiz ou no bloco Ramo de nível acima.



**A utilização de uma sequência de números como chave primária reduz a manutenção do respectivo índice. Isto porque anula a necessidade de alterar o valor da chave e os novos valores são sempre números superiores aos existentes, o que significa que a árvore cresce só para a direita.**

## 1.2 Quando é que um índice é muito selectivo?

A selectividade de um índice é uma medida da sua utilidade. Um índice é tanto mais selectivo quantos menos valores repetidos tiver. Um índice criado sobre a coluna Data de Aniversário é mais selectivo que outro criado sobre a coluna Estado Civil. Quando o Optimizer tem à sua disposição vários índices, começa por avaliar a selectividade de cada um, decidindo-se a usar o que tiver maior selectividade.

## 1.3 O que são índices implícitos?

São índices criados pelo Oracle para implementar algumas restrições (“constraints”). Quando se define a restrição “Unique” o Oracle cria um índice que ajuda a garantir que não haverá valores repetidos na coluna ou conjunto de colunas. Quando se define uma “Primary key” o Oracle cria um índice que além de “Unique”

garante “Not Null” sobre a coluna ou colunas seleccionadas. Estes índices são criados automaticamente e são usados para acelerar queries sempre que necessário.

## 1.4 Quais as vantagens e inconvenientes dos índices concatenados?

Para executar os exemplos a seguir indicados vamos usar um hint que força o Optimizer a trabalhar em modo Rule. Isto porque o volume de dados envolvido é pequeno e o Optimizer em modo Cost (Choose com estatísticas) prefere fazer full table scan a usar um índice que o obrigará a mais operações de I/O. Com um volume de dados de produção este problema já não se colocaria.

Um índice concatenado é composto por mais de uma coluna da tabela. Na nossa tabela EMPLOYEE vamos criar um índice sobre as colunas FIRST\_NAME, LAST\_NAME e HIRE\_DATE, nesta ordem:

```
CREATE INDEX I_EMPLOYEE$FIRST_LAST_HIRE ON EMPLOYEE(FIRST_NAME, LAST_NAME, HIRE_DATE);
```

O query abaixo sobre as colunas FIRST\_NAME, LAST\_NAME e HIRE\_DATE vai usar o índice concatenado:

```
select /*+ rule */ e.first_name,e.last_name,e.hire_date,e.salary
from employee e
where e.first_name like 'D%'
and e.last_name like 'S%'
and e.hire_date = to_date('1986-12-09','yyyy-mm-dd');

SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=22)
TABLE ACCESS (BY INDEX ROWID) OF EMPLOYEE (Cost=2 Card=1 Bytes=22)
INDEX (RANGE SCAN) OF I_EMPLOYEE$FIRST_LAST_HIRE (NON-UNIQUE) (Cost=1 Card=1)
```

O query abaixo sobre as colunas FIRST\_NAME e LAST\_NAME também vai usar o índice concatenado:

```
select /*+ rule */ e.first_name,e.last_name,e.hire_date,e.salary
from employee e
where e.first_name like 'D%'
and e.last_name like 'S%';

SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=22)
TABLE ACCESS (BY INDEX ROWID) OF EMPLOYEE (Cost=2 Card=1 Bytes=22)
INDEX (RANGE SCAN) OF I_EMPLOYEE$FIRST_LAST_HIRE (NON-UNIQUE) (Cost=1 Card=1)
```

O query sobre a coluna FIRST\_NAME também usa o índice concatenado:

```
select /*+ rule */ e.first_name,e.last_name,e.hire_date,e.salary
from employee e
where e.first_name like 'D%';

SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=3 Bytes=66)
TABLE ACCESS (BY INDEX ROWID) OF EMPLOYEE (Cost=2 Card=3 Bytes=66)
INDEX (RANGE SCAN) OF I_EMPLOYEE$FIRST_LAST_HIRE (NON-UNIQUE) (Cost=1 Card=3)
```

No entanto os queries a seguir apresentados já não usam o índice concatenado:

```
select /*+ rule */ first_name,last_name,hire_date,salary
from employee e
where e.last_name like 'S%';

select /*+ rule */ first_name,last_name,hire_date,salary
from employee e
where e.hire_date = to_date('1986-12-09','yyyy-mm-dd');

select /*+ rule */ first_name,last_name,hire_date,salary
from employee e
where e.last_name like 'S%'
and e.hire_date = to_date('1986-12-09','yyyy-mm-dd')

SELECT STATEMENT Optimizer=HINT: RULE
TABLE ACCESS (FULL) OF EMPLOYEE
```



**Um índice concatenado é mais selectivo e mais rápido na pesquisa que a intercalação dos vários índices criados sobre cada coluna individualmente. As colunas mais selectivas devem aparecer primeiro.**



**Um índice concatenado sobre as colunas {A,B,C} pode ser usado para pesquisas sobre {A}, {A,B} e {A,B,C}. Não pode ser usado para pesquisas sobre {B} e {B,C}. A ordem das colunas na cláusula WHERE não é relevante.**



**Se o índice contiver todas as colunas que são pedidas no SELECT, então o pedido é resolvido apenas com consulta ao índice, poupando-se o acesso à tabela de dados.**

## 1.5 Quais as vantagens e inconvenientes da intercalação de índices?

Para executar os exemplos a seguir indicados vamos usar um hint que força o Optimizer a trabalhar em modo Rule. Isto porque o volume de dados envolvido é pequeno e o Optimizer em modo Cost (Choose com estatísticas) prefere fazer full table scan a usar um índice que o obrigará a mais operações de I/O. Com um volume de dados de produção este problema já não se colocaria.

Na tabela SALES\_ORDER vamos criar um índice sobre a coluna ORDER\_DATE e outro sobre SHIP\_DATE.

```
CREATE INDEX I_sales_order$order_date ON SALES_ORDER(ORDER_DATE)
CREATE INDEX I_sales_order$ship_date ON SALES_ORDER(SHIP_DATE)
```

Quando fazemos um query sobre ambas as colunas o Optimizer verifica que não tem um índice concatenado, mas tem índices individuais. Decide-se então por uma operação AND-EQUAL. Esta consiste em pesquisar um índice para obter um conjunto com as linhas que satisfazem a primeira condição. Em seguida cria um segundo conjunto com o segundo índice e a segunda condição. Finalmente faz a intercalação dos dois conjuntos, devolvendo as linhas que se encontram em ambos.

```
select /*+ rule */ *
from   sales_order
where  order_date = to_date('1990-06-05', 'yyyy-mm-dd')
and    ship_date = to_date('1990-06-05', 'yyyy-mm-dd');

SELECT STATEMENT Optimizer=HINT: RULE
TABLE ACCESS (BY INDEX ROWID) OF SALES_ORDER
AND-EQUAL
INDEX (RANGE SCAN) OF I_SALES_ORDER$ORDER_DATE (NON-UNIQUE)
INDEX (RANGE SCAN) OF I_SALES_ORDER$SHIP_DATE (NON-UNIQUE)
```



**A intercalação de índices é mais lenta que a pesquisa por um índice concatenado, mas aumenta a flexibilidade do uso dos índices em pesquisas.**

## 1.6 Os índices guardam valores NULL?

Os índices em Oracle não guardam os valores null. Isto significa que as linhas da tabela que têm null não terão apontador no índice.



**Uma pesquisa por "IS NULL" força um "FULL TABLE SCAN".**