Arquitetura e Administração de Bases de Dados

Parâmetros Físicos

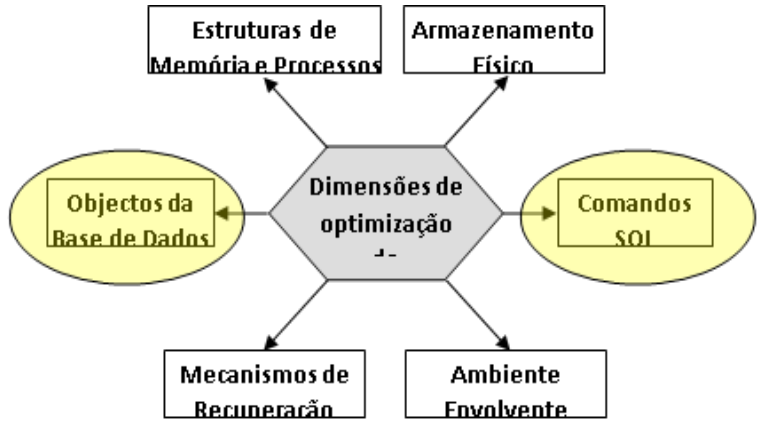
João Costa

jcosta@isec.pt

Agenda

- Otimização de pesquisas

Otimização



Otimização

- **Estruturas de memória e processos**:
  - otimizar as operações de entrada/saída em disco através da correta definição dos tamanhos das diferentes áreas de memória alocadas
  - otimizar o funcionamento de processos e a quantidade de processos
- **Mecanismos de recuperação**:
  - otimização balanceada em termos de desempenho e de recuperação
  - conciliar o tempo de recuperação e o número de transações perdidas em situações de erro, com o desempenho do sistema
  - numa otimização apenas para desempenho estes mecanismos devem ser configurados de modo a causarem o menor impacto possível
- **Ambiente envolvente**:
  - componentes que podem afetar o desempenho do sistema
    - exemplo o hardware, o sistema operativo e a infraestrutura de rede.

Otimização

- **Armazenamento físico**:
  - minimizar a contenção no acesso aos dados e a fragmentação no armazenamento dos objetos da base de dados
  - distribuição dos ficheiros de dados por vários discos
  - outros parâmetros usados na alocação de espaço
- **Acesso aos objetos da base de dados**:
  - normalização das tabelas,
  - a criação dos índices adequados,
  - *clustering* e o particionamento de alguns objetos,
- **Comandos SQL**:
  - o perfil das transações afeta também o desempenho
  - otimização da execução dos comandos SQL
  - o objetivo passa por evitar a contenção em transações devido ao bloqueio de objetos, de modo a minimizar o impacto dos mecanismos de controlo de concorrência no desempenho do sistema

Otimização

Através da Criação de Índices

- estruturas facultativas associadas a colunas de tabelas ou agrupamento de tabelas destinadas a acelerar o acesso aos dados
- o equivalente a um índice remissivo de um livro pois permitem localizar rapidamente a informação pretendida.
- A criação ou remoção de índices não interfere com os dados da tabela nem com a estrutura da base de dados.

Os índices podem ser de dois tipos:

- **B*Tree :**

```
CREATE INDEX nome
     ON tabela(coluna1 [,coluna2 [,...]])
```

- **BitMap :**

```
CREATE BITMAP INDEX nome
     ON tabela(coluna1 [,coluna2 [,...]])
```

Otimização

***Cluster* Indexado:**

1) Cria-se o Cluster:

```
CREATE CLUSTER nome_do_cluster (cod_socio number(3))
       PCTFREE 15
       PCTUSED 70
       SIZE 1000
       TABLESPACE users;
```

2) Cria-se o índice da *cluster key*:

```
CREATE INDEX nome ON CLUSTER nome_do_cluster
       TABLESPACE index;
```

3) Criam-se as várias tabelas que devem ser contidas no *cluster*:

```
CREATE TABLE nome_da_tabela
      ( cod_socio number(3),
        nome varchar(30),
        idade number (4) )
CLUSTER nome_do_cluster
        (atributos_na_tabela que definem o cluster key);
```


Otimização

Hash Cluster

1) Cria-se o Cluster:

```
CREATE CLUSTER nome_do_cluster (cod_socio number(3))
      HASH IS funçao_de_hashing
              HASHKEYS 501
              PCTFREE 15
              PCTUSED 70
              SIZE 1000
              TABLESPACE users;
```

2) Criam-se as várias tabelas que devem ser contidas no *cluster*:

```
CREATE TABLE nome_da_tabela
      ( cod_socio number(3),
        nome varchar(30),
        idade number (4) )
CLUSTER nome_do_cluster
        (atributos_na_tabela que definem o cluster key);
```

Otimização

Otimização de Comandos SQL

- O optimizador de comandos é o componente responsável pela escolha da melhor forma de execução para cada comando SQL.
- Essa escolha, realizada através da avaliação das várias hipóteses possíveis, depende do modo de otimização: baseado em regras ou em custos.

Otimização

Otimização baseado em regras

- o melhor plano de execução é escolhido segundo um conjunto de regras que definem como os comandos SQL devem ser processados
- independentemente dos dados a serem acedidos.

```
1  - Single row by rowid
2  - Single row by cluster join
3  - Single row by hash cluster key with unique or primary key
4  - Single row by unique or primary key
5  - Cluster join
6  - Hash cluster key
7  - Indexed cluster key
8  - Composite key
9  - Single-column indexes
10 - Bounded range search on indexed columns
11 - Unbounded range search on indexed columns
12 - Sort-merge join
13 - MAX or MIN of indexed column
14 - ORDER BY on indexed columns
15 - Full table scan
```

Otimização

Otimização baseado em custos

- são elaborados um conjunto de planos de execução alternativos
- a cada plano é associado um custo, tomando em consideração várias variáveis tais como
  - a seletividade,
  - o número de registos do resultado,
  - número de blocos lidos de disco,
  - taxa de transferência de disco,
  - memória disponível para dados e para ordenações, etc ....
- Escolhe o plano de execução mais eficiente tendo também em consideração informação estatística sobre os dados existentes nas tabelas, *clusters* e índices
- Por usar estatísticas sobre os dados, a otimização baseada em custos é normalmente mais eficiente do que a otimização baseada em regras.

Otimização

Reescrita de pesquisas

```
select  count(*)
from    SUPPLIER
```

```
select  count(S_SUPPKEY)
from    SUPPLIER
```

```
select *
from    PART
where   UPPER(P_TYPE) = UPPER('PLATED STEEL')
```

```
select *
from    PART
where   P_TYPE = UPPER('PLATED STEEL')
```

Otimização

Reescrita de pesquisas

```
Select  count(o_custkey)
from    orders, customer
where   o_custkey = c_custkey
and     substr(c_mktsegment,1,1) ='A'
```

```
Select  count(o_custkey)
from    orders, customer
where   o_custkey = c_custkey
and     c_mktsegment like 'A%'
```


Tuning

```
SET TIMING ON;
-- para obter os tempos de execução

SET AUTOTRACE ON EXPLAIN;
-- para obter o plano de execução
```

Tuning

```
select nome, count(*)
from    livros l , autores a
where  l.codigo_autor = a.codigo_autor
group by nome;
```

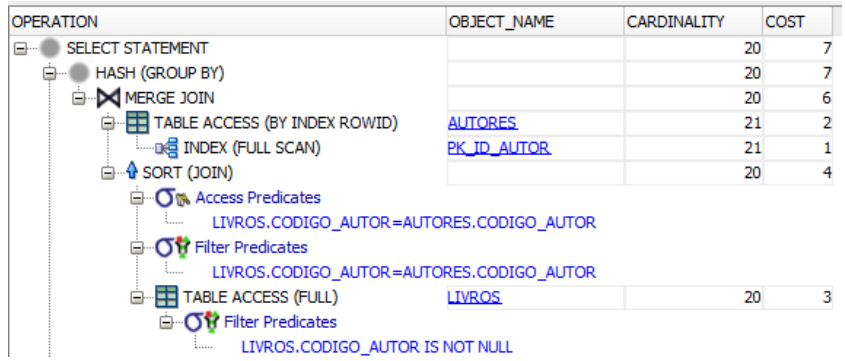

Tuning – Plano de execução

```
-----------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |    20 |   420 |     7  (29)| 00:00:01 |
|   1 |  HASH GROUP BY                |             |    20 |   420 |     7  (29)| 00:00:01 |
|   2 |   MERGE JOIN                  |             |    20 |   420 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| AUTORES     |    21 |   378 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | PK_ID_AUTOR |    21 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |             |    20 |    60 |     4  (25)| 00:00:01 |
|*  6 |     TABLE ACCESS FULL         | LIVROS      |    20 |    60 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("LIVROS"."CODIGO_AUTOR"="AUTORES"."CODIGO_AUTOR")
       filter("LIVROS"."CODIGO_AUTOR"="AUTORES"."CODIGO_AUTOR")
   6 - filter("LIVROS"."CODIGO_AUTOR" IS NOT NULL)

Elapsed: 00:00:01.187
```

Tuning – Plano de execução



OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		20	7
HASH (GROUP BY)		20	7
MERGE JOIN		20	6
TABLE ACCESS (BY INDEX ROWID)	AUTORES	21	2
INDEX (FULL SCAN)	PK_ID_AUTOR	21	1
SORT (JOIN)		20	4
Access Predicates			
LIVROS.CODIGO_AUTOR=AUTORES.CODIGO_AUTOR			
Filter Predicates			
LIVROS.CODIGO_AUTOR=AUTORES.CODIGO_AUTOR			
TABLE ACCESS (FULL)	LIVROS	20	3
Filter Predicates			
LIVROS.CODIGO_AUTOR IS NOT NULL			

Tuning

```
CREATE INDEX XX ON LIVROS(CODIGO_AUTOR);


-- EXECUTAR NOVAMENTE A PESQUISA;
```

Tuning – Plano de execução

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |    20 |   420 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY      |         |    20 |   420 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS      |         |    20 |   420 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| AUTORES |    21 |   378 |     3   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN | XX      |     1 |     3 |     0   (0)| 00:00:01 |

Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("LIVROS"."CODIGO_AUTOR"="AUTORES"."CODIGO_AUTOR")
       filter("LIVROS"."CODIGO_AUTOR" IS NOT NULL)

Elapsed: 00:00:00.088
```


Tuning – Plano de execução

SQL | 1,578 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		20	4
HASH (GROUP BY)		20	4
NESTED LOOPS		20	3
TABLE ACCESS (FULL)	AUTORES	21	3
INDEX (RANGE SCAN)	XX	1	0
Access Predicates			
LIVROS.CODIGO_AUTOR=AUTORES.CODIGO_AUTOR			
Filter Predicates			
LIVROS.CODIGO_AUTOR IS NOT NULL			

Join Operation

- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join
  - Hash-join
- Choice based on cost estimate

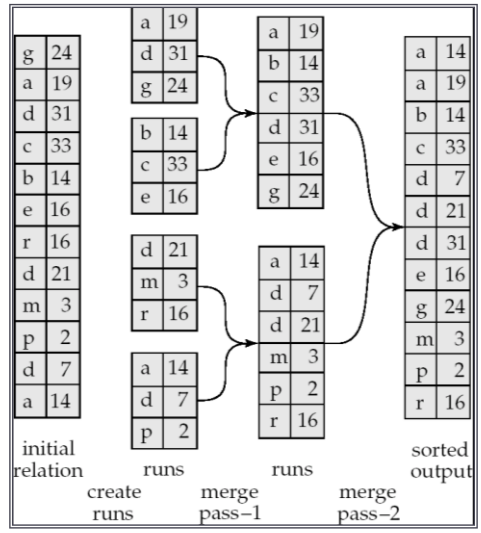
Nested-Loop (NL) Join

- To compute the theta join $r \underset{\theta}{\bowtie} s$

```
for each tuple t_r in r do begin
   for each tuple t_s  in s do begin
      test pair (t_r,t_s) to see if they satisfy the join condition θ
      if they do, add t_r.t_s to the result.
   end
end
```

- ***r*** is called the outer relation and ***s*** the inner relation of the join
- Requires no indices and can be used with any kind of join condition
- Expensive since it examines every pair of tuples in the two relations

External Sorting Using Sort-Merge



Hash-Join (Cont.)