

Processamento e Otimização de Consultas

Prof. Humberto Razente

humberto.razente@ufu.br

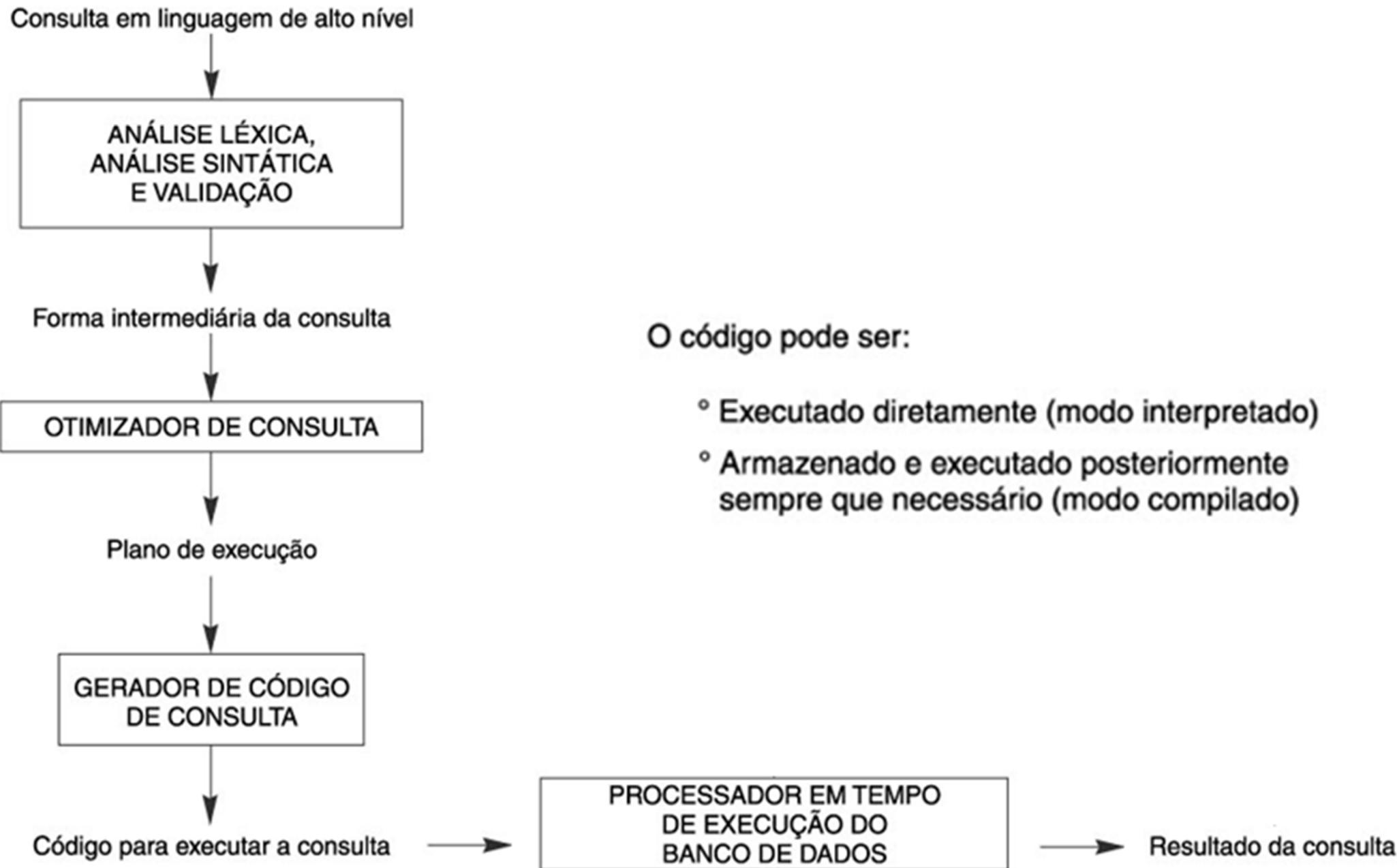
Bloco B – sala 1B144

Introdução ao Processamento de Consultas

● Otimização de consulta

- processo de escolha de uma estratégia adequada de execução para processamento de uma consulta

Introdução ao Processamento de Consultas

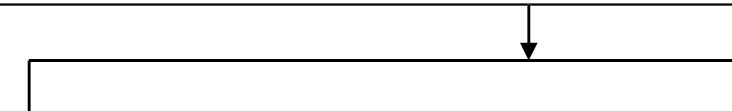


1. Tradução de consultas SQL em álgebra relacional

- **Bloco de consulta:** unidade básica a ser traduzida em operadores algébricos a ser otimizada
- Um bloco contém uma única expressão SELECT-FROM-WHERE, bem como uma cláusula GROUP BY e HAVING se forem parte do bloco
- **Consultas aninhadas** em uma consulta são identificadas como blocos separados

1. Tradução de consultas SQL em álgebra relacional

```
SELECT          LNAME, FNAME  
FROM           EMPLOYEE  
WHERE          SALARY > (SELECT      MAX (SALARY)  
                      FROM        EMPLOYEE  
                      WHERE       DNO = 5);
```



```
SELECT          MAX (SALARY)  
FROM           EMPLOYEE  
WHERE          DNO = 5
```

```
 $\pi_{\text{LNAME, FNAME}} (\sigma_{\text{SALARY} > \text{C}}(\text{EMPLOYEE}))$ 
```

```
 $\mathcal{F}_{\text{MAX SALARY}} (\sigma_{\text{DNO} = 5} (\text{EMPLOYEE}))$ 
```

1. Tradução de consultas SQL em álgebra relacional

SELECT	LNAME, FNAME
FROM	EMPLOYEE
WHERE	SALARY > (
	SELECT MAX (SALARY)
	FROM EMPLOYEE
	WHERE DNO = 5);

- Nesse exemplo, a consulta aninhada não é correlacionada com a consulta externa
 - então, será executada apenas 1 vez e seu resultado será empregado pelo WHERE
- Em uma consulta correlacionada, a otimização é mais complexa

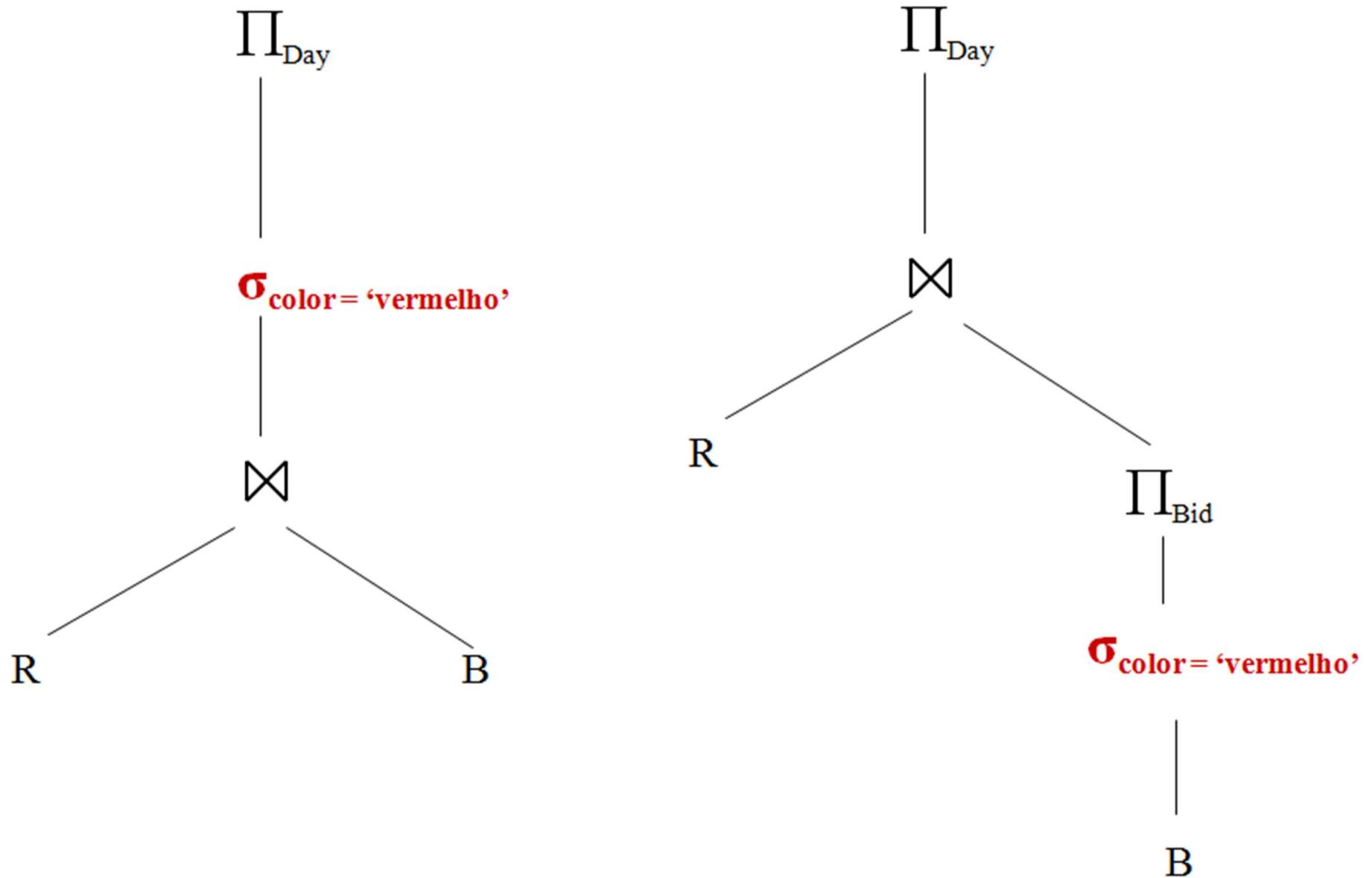
Exemplo

- R(sid,bid,day,rname): RESERVAS
- S(sid,sname,rating,age): MARINHEIROS
- B(bid,bname, color): BARCOS
- Quais os dias em que foram reservados barcos vermelhos ?

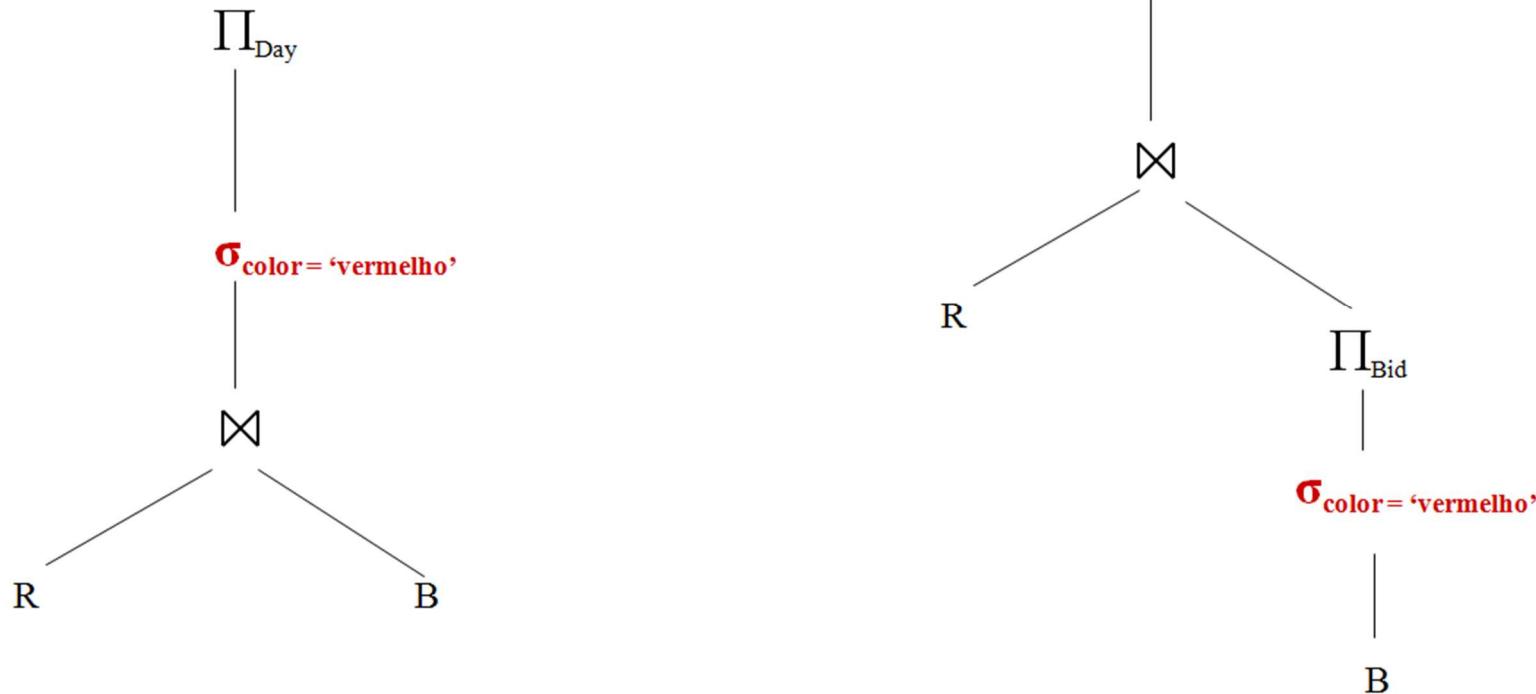
```
SELECT R.Day  
FROM R JOIN B ON (R.Bid = B.Bid)  
WHERE B.Color = 'Vermelho'
```

Exemplo

```
SELECT R.Day  
      FROM R JOIN B ON (R.Bid = B.Bid)  
 WHERE B.Color = 'Vermelho'
```



Exemplo



Há propriedades que permitem determinar e gerar expressões algébricas equivalentes

- além disso, a cada expressão pode ser atribuído um custo (baseado em metadados das tabelas e índices) e o SGBD pode escolher uma estratégia com custo baixo → o número de combinações pode ser exponencial, logo, em geral, usam-se heurísticas para calcular apenas algumas combinações

2. Algoritmos para Ordenação Externa

- Intercalação
 - um dos principais algoritmos utilizados no processamento de consulta
- Exemplos de uso:
 - cláusula ORDER BY: o resultado precisa ser ordenado
 - componente chave nos algoritmos ordenação-intercalação (sort-merge) usados na JUNÇÃO, UNIÃO e INTERSECÇÃO, além de serem usados para eliminação de duplicatas na PROJEÇÃO (quando se especifica no SELECT a opção DISTINCT)

2. Algoritmos para Ordenação Externa

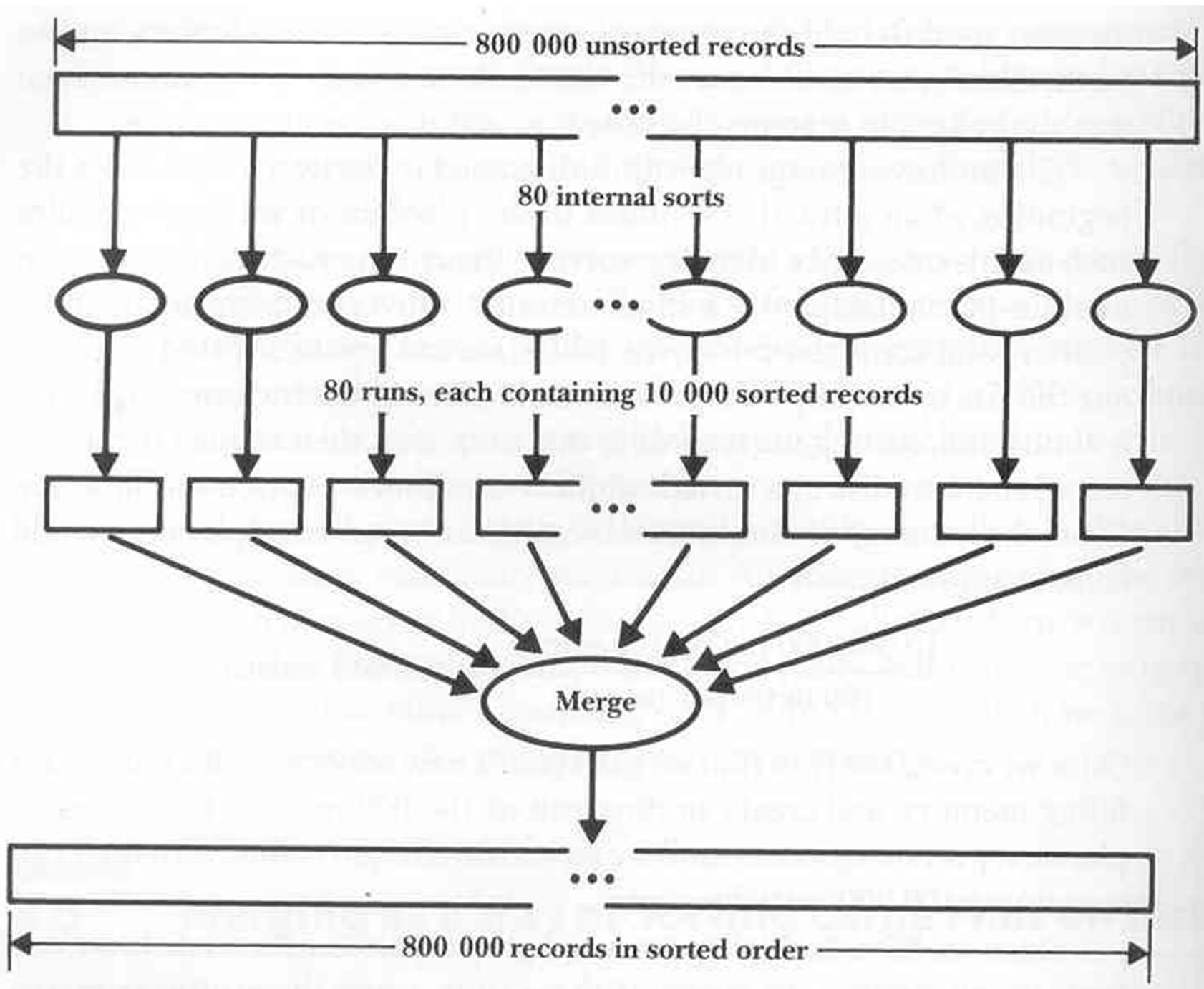
- Intercalação

- observe que o processamento da intercalação pode ser evitado se um índice apropriado existir para o atributo que se precisa ter acesso ordenado
 - índice primário ou de agrupamento

2. Algoritmos para Ordenação Externa

- **Ordenação externa:** refere-se aos algoritmos de ordenação que são adequados para grandes conjuntos de registros armazenados em disco rígido e que não cabem inteiramente em memória principal (RAM)
- **Estratégia ordenação-intercalação:** inicia pela ordenação de pequenas partes do arquivo e então intercala as ordenações
 - ordenação de cada parte em memória principal
 - intercalação das partes ordenadas (temporárias) salvas em disco

2. Algoritmos para Ordenação Externa



3. Algoritmos para operações SELECT

Implementação da operação SELECT:

- Exemplos:

- (1): $\sigma_{NUMDEPTO=5}(\text{FUNCIONARIO})$

- **Busca linear (força bruta)**: recupera cada registro e testa se os valores dos atributos satisfazem o critério de seleção

3. Algoritmos para operações SELECT

Implementação da operação SELECT:

- Exemplos:

(2): $\sigma_{CPF='123456789'}(\text{FUNCIONARIO})$

- **Usar a chave primária ou chave hash** para recuperar um único registro: se a condição de seleção envolver uma comparação de igualdade, usar o índice para recuperar o registro.

(3): $\sigma_{NUMDEPTO=5 \text{ AND } CPF > '123456789' \text{ AND } SEX=M}(\text{FUNCIONARIO})$

- **Usar um índice primário** para recuperar múltiplos registros: se a condição de comparação for $>$, \geq , $<$, ou \leq sobre uma chave com um índice primário, usar o índice para encontrar o registro que satisfaz a igualdade e então percorrer o índice em ordem para recuperar os registros seguintes que também satisfazem a condição

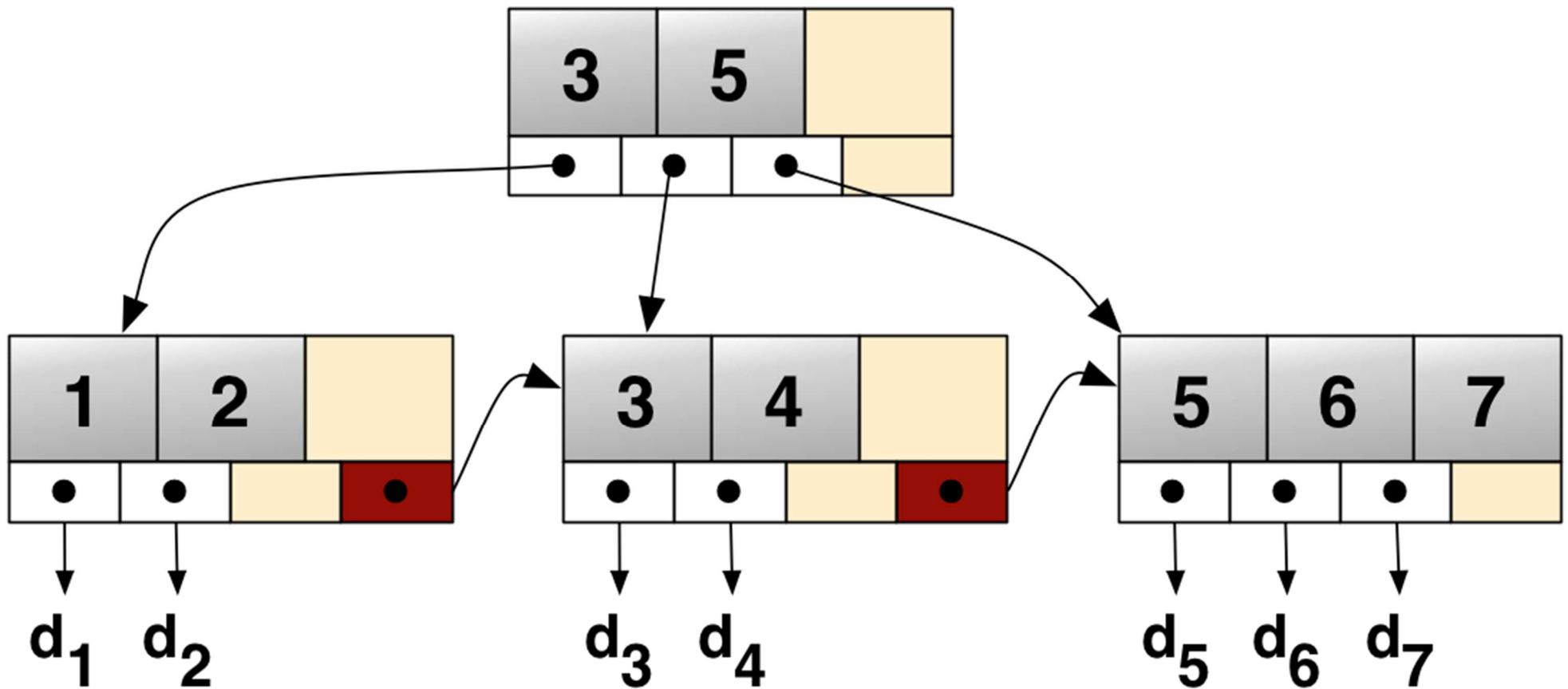
3. Algoritmos para operações SELECT

Implementação da operação SELECT:

Método de busca para seleção simples:

- (4) **Uso de um índice secundário (B+-tree)**: em uma comparação por igualdade, a B+tree pode ser usada para recuperar um único registro (se o campo for chave) ou recuperar múltiplos registros (se o campo não for chave). Adicionalmente, pode ser usado para recuperar registros em condições envolvendo $>$, $>=$, $<$, ou $<=$ (abrangência)

● Exemplo B+-tree



● Características da B+tree

- construção bottom-up, sempre balanceada
- utiliza páginas de disco de tamanho fixo
- com uma página típica de 4 KB, chave de 8 bytes e offset de 8 bytes, temos 256 pares por página
 - com altura 3, ordenam-se 16 milhões de chaves
 - com altura 4, ordenam-se 4 bilhões de chaves
- com uma página de 8 KB, temos 512 pares por página
 - com altura 3, ordenam-se 134 milhões de chaves
 - se for mantida no **cache** apenas o nó raiz, é possível encontrar um chave entre 134 milhões com 2 acessos a disco

3. Algoritmos para operações SELECT

Implementação da operação SELECT:

Método de busca para seleção simples:

- (5) **Uso de um índice de bitmap:** atributos com baixa cardinalidade de valores distintos podem se beneficiar de um índice de bitmap
 - exemplos atributos como: estado civil, região, estado
 - desde que apresentem baixa cardinalidade
 - tabelas majoritariamente estáticas: tabelas que não sofram atualizações constantes, pois atualizações nesses índices são caras!

- Exemplo índice bitmap

	Rows ----->																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
North			●		●				●		●				●		●	
South	●			●		●			●		●		●		●		●	
East																		
West	●				●		●				●		●				●	

● Consultas espaciais: árvores R

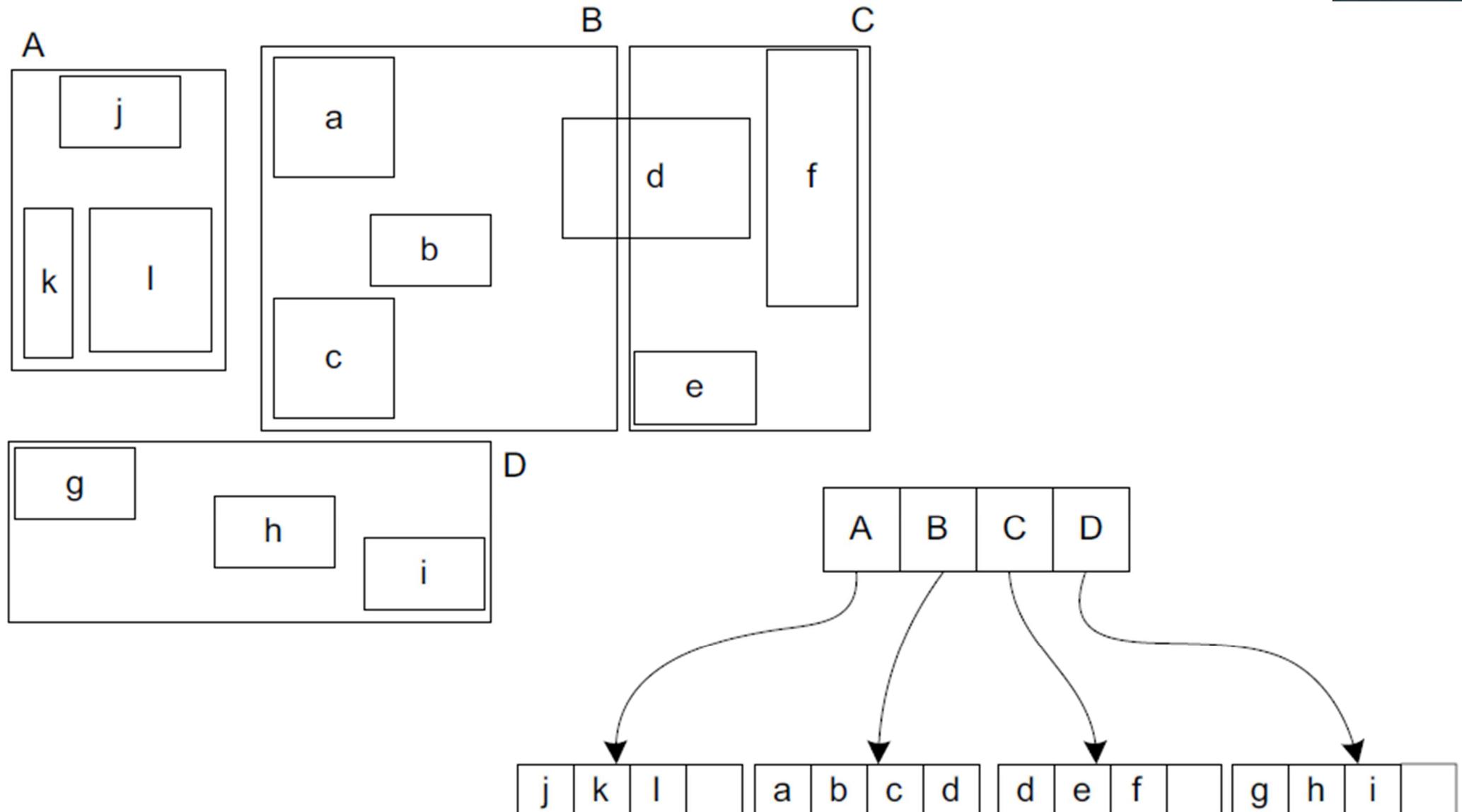


Fig. 2.1. An R⁺-tree example.

3. Algoritmos para operações JOIN

EQUIJUNÇÃO

- Supondo que as listas estejam ordenadas

- lê-se um elemento de cada lista
 - ADAMS é inserido na resposta
- lê-se o próximo elemento de cada lista
 - CARTER (lista 1) e lê-se sequencialmente a lista 2 até encontrar o elemento ou outro elemento "maior" que CARTER

List 1	List 2
ADAMS	ADAMS
CARTER	ANDERSON
CHIN	ANDREWS
DAVIS	BECH
FOSTER	BURNS
GARWICK	CARTER
JAMES	DAVIS
JOHNSON	DEMPSEY
KARNS	GRAY
LAMBERT	JAMES
MILLER	JOHNSON
PETERS	KATZ
RESTON	PETERS
ROSEWALD	ROSEWALD
TURNER	SCHMIDT
	THAYER
	WALKER
	WILLIS

3. Algoritmos para operações JOIN

Implementação da operação JOIN:

Métodos para implementação de junções:

- (J1) **Nested-loop join** (força bruta): para cada registro t em R (loop externo), recuperar cada registro s (loop interno) e testar se os dois registros satisfazem a condição de junção $t[A] = s[B]$

(a)

```
For each tuple r in R do
    For each tuple s in S do
        If r and s satisfy the join condition
            Then output the tuple <r,s>
```

(b)

```
For each block block_r in R do
    For each tuple s in S do
        For each tuple r in block_r do
            If r and s satisfy the join condition
                Then output the tuple <r,s>
```

- (a) para cada **tupla** de R, percorre todas as **tuplas** de S (memória 1^a)
- (b) para cada **bloco** de R, percorre todos os **blocos** de S (memória 2^a)

3. Algoritmos para operações JOIN

Implementação da operação JOIN:

Métodos para implementação de junções:

- (J2) **Single-loop join** (usando um índice para recuperar os registros):
 - se um índice B+ (ou hash em disco) existir para um dos dois atributos de junção (B or S)
 - recuperar cada registro t em R, um de cada vez, e então usar o índice para recuperar diretamente todos os registros s de S que satisfazem $s[B] = t[A]$

3. Algoritmos para operações SELECT e JOIN

Implementação da operação JOIN:

Métodos para implementação de junções:

- (J3) **Sort-merge join:** se os registros de R e S estão fisicamente ordenados pelo valor dos atributos de junção A e B, é possível implementar a junção no modo mais eficiente possível
 - ambos arquivos são lidos na ordem dos atributos de junção
 - os registros de cada arquivo são lidos apenas uma vez para cada par que satisfaça a condição da junção

3. Algoritmos para operações SELECT e JOIN

Implementação da operação JOIN:

Métodos para implementação de junções:

- (J4) **Hash-join:** se para os registros de R e S há uma estrutura hash, uma passagem pelo arquivo com menos registros (R) encontra registros que podem ser consultados pelo hash do outro arquivo (S)

3. Algoritmos para operações SELECT e JOIN

Implementação da operação JOIN:

- Fatores que afetam o desempenho da junção
 - Espaço disponível de buffers
 - Fator de seleção da junção
 - Escolha da relação interna *vs* externa

5. Implementação de Operações de Agregação

Implementação de operações de agregação:

- Operadores de agregação: MIN, MAX, SUM, COUNT e AVG
- Opções de implementação dos operadores de agregação:
 - **Table Scan**
 - **Index**
- **Example**

```
SELECT MAX (SALARY) FROM EMPLOYEE;
```

Se existir um índice ascendente em SALARY, então o otimizador poderia decidir em fazer a travessia no índice para recuperar o maior valor → buscar em profundidade o ponteiro mais à direita da última folha

6. Combinação de Operações com Pipelines

● Motivação

- uma consulta é mapeada em uma sequência de operações
- Cada execução produz um resultado temporário
- Salvar arquivos temporários em disco consome tempo

● Alternativa:

- Evitar ao máximo construir resultados temporários
- Criar pipeline de dados através de múltiplas operações: passando os resultados de um operador prévio ao próximo sem esperar completar a operação prévia

Planos de Consulta

- O melhor plano pode não ser óbvio
 - CREATE TABLE foo (a integer, txt varchar);
 - CREATE INDEX foo_a ON foo (a);
 - ...inserts...
 - SELECT * FROM foo WHERE a = 1;
 - O que o gerador de planos deve fazer?

Planos de Consulta: distribuição dos dados

- A distribuição dos dados afeta a escolha do plano:

```
SELECT * FROM foo WHERE a = 1
```

- Plan #1 (10000 linhas, a = 1 .. 10000):
Index Scan using foo_a on foo
Index Cond: (a = 1)
- Plan #2 (10000 linhas, 90% tem a = 1):
Seq Scan on foo
Filter: (a = 1)

Planos de Consulta: distribuição dos dados

- A distribuição dos dados afeta a escolha do plano:

```
SELECT * FROM foo WHERE a = 1
```

- Plan #3 (10000 linhas, a = 1 .. 10, 1000 vezes cada):

```
Bitmap Heap Scan on foo
```

```
    Recheck Cond: (a = 1)
```

```
    -> Bitmap Index Scan on foo_a
```

```
        Index Cond: (a = 1)
```

Planos de Consulta: junções

- CREATE TABLE foo (a integer, txt varchar);
- CREATE TABLE bar (a integer, txt varchar);
- CREATE INDEX foo_a ON foo (a);
- CREATE INDEX bar_a ON bar (a);
- ...inserts...
- SELECT * FROM foo, bar WHERE foo.a = bar.a
- O que o gerador de planos deve fazer?

Objetivos do planejamento de consultas

- Tornar consultas mais rápidas
- Minimizar E/S de disco
- Preferir E/S sequencial a E/S aleatória
- Minimizar processamento CPU
- Economizar memória RAM no processo
- Encontrar resultados corretos

Analizando um plano de consulta

● EXPLAIN

- <http://www.postgresql.org/docs/9.4/static/sql-explain.html>

```
EXPLAIN SELECT * FROM foo;

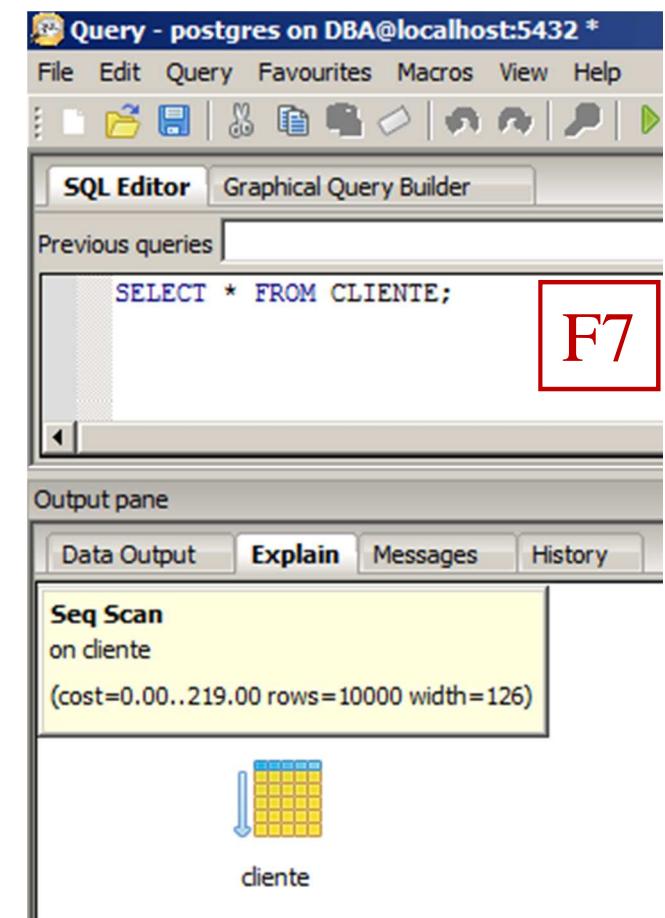
QUERY PLAN
-----
Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```

custo inicial estimado

custo total estimado

número estimado de linhas

número estimado de bytes por linha



Analisando um plano de consulta

```
explain
```

```
select nome  
  from cliente  
 where codcli >= 50  
   and codcli <= 100;
```

```
Index Scan using cliente_pkey on cliente  
(cost=0.00..9.27 rows=51 width=21)  
  Index Cond: ((codcli >= 50) AND (codcli <= 100))
```

custo total e número de linhas estimados leva em consideração condições

apesar do atributo nome ser um varchar(100), o comprimento médio das strings é 21

Analisando um plano de consulta

```
explain analyse
```

```
select nome  
      from cliente  
     where codcli >= 50  
       and codcli <= 100
```

```
Index Scan using cliente_pkey on cliente  
(cost=0.00..9.27 rows=51 width=21)  
(actual time=0.010..0.024 rows=51 loops=1)  
    Index Cond: ((codcli >= 50) AND (codcli <= 100))  
Total runtime: 0.044 ms
```

Opção **analyse** executa efetivamente a consulta e apresenta tempo de execução e outras estatísticas!

Analisando um plano de consulta

```
explain  
select nome  
  from cliente  
 where nome = 'Carlos Vitor Sanches';
```

```
Seq Scan on cliente  
(cost=0.00..244.00 rows=1 width=21)  
  Filter: ((nome)::text = 'Carlos Vitor Sanches'::text)
```

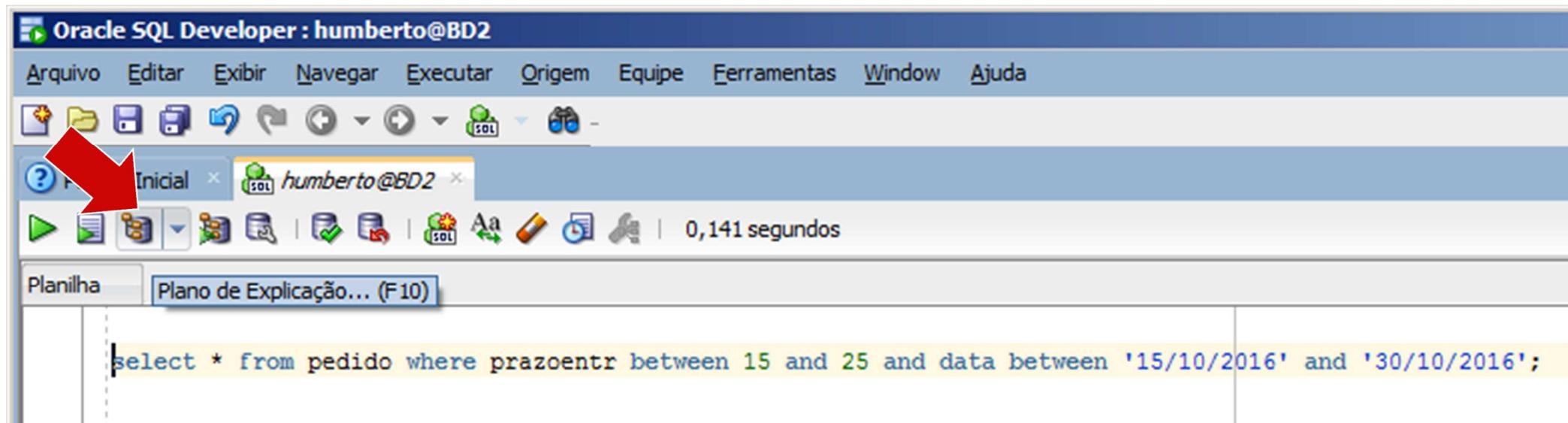
```
create index cliente_nome_idx on cliente(nome);
```

```
explain  
select nome  
  from cliente  
 where nome = 'Carlos Vitor Sanches';
```

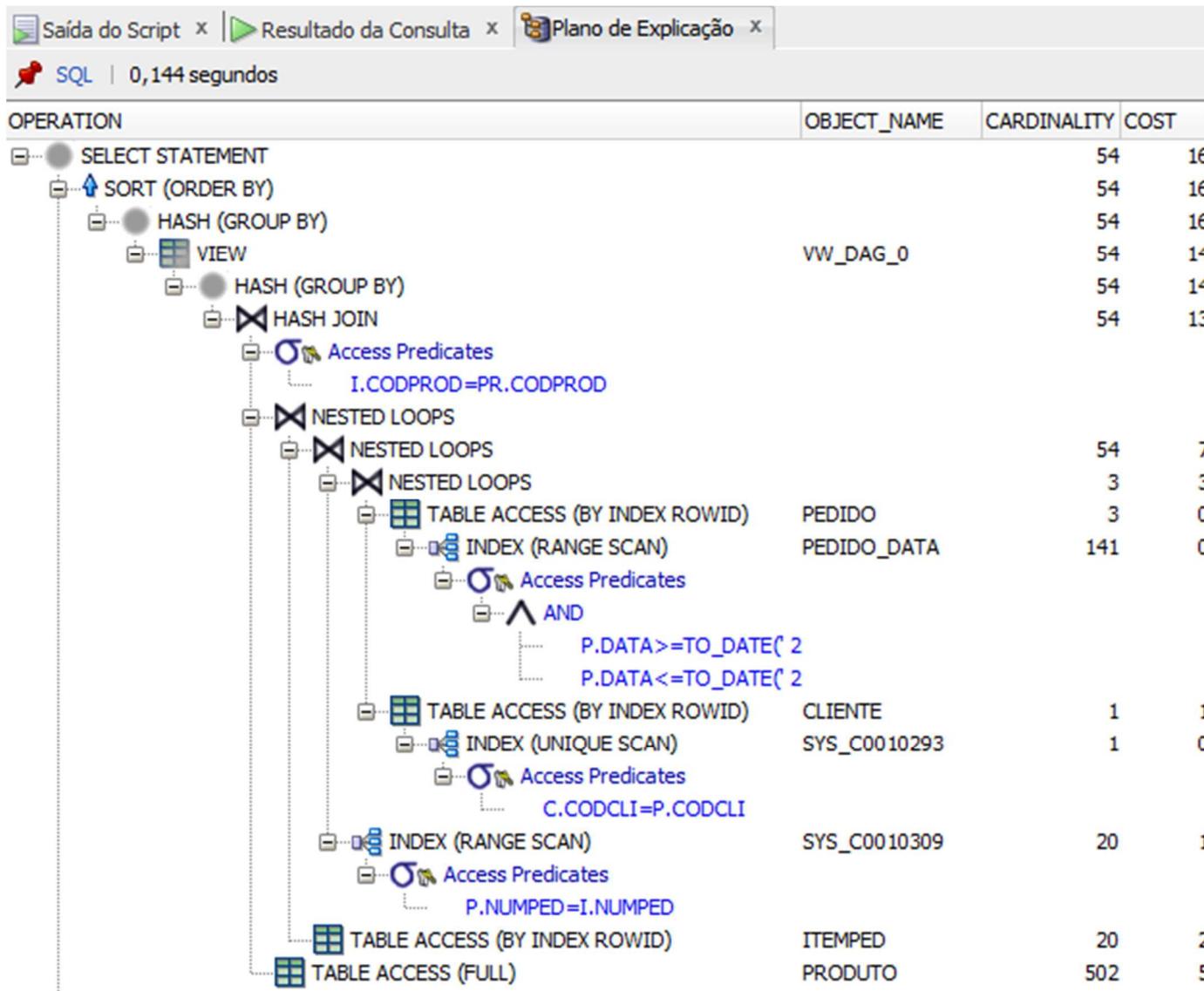
```
Index Only Scan using cliente_nome_idx on cliente  
(cost=0.00..8.27 rows=1 width=21)  
  Index Cond: (nome = 'Carlos Vitor Sanches'::text)
```

custo total antes e depois da criação do índice no atributo cliente.nome

Oracle SQL Developer



Oracle SQL Developer

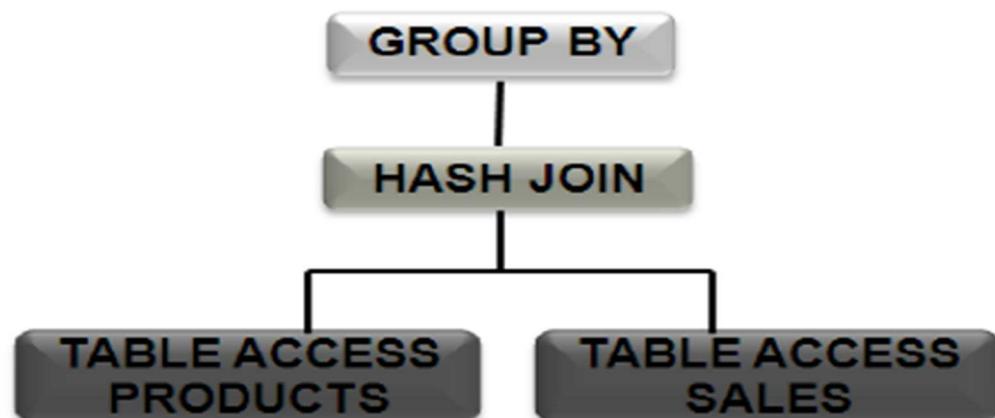


Oracle

```
SELECT      prod_category, AVG(amount_sold)
FROM        sales s, products p
WHERE       p.prod_id = s.prod_id
GROUP BY    prod_category;
```

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart Pstop	
0 SELECT STATEMENT				1140 (100)			
1 HASH GROUP BY		4	80	1140 (45)	00:00:14		
* 2 HASH JOIN		489KI	9555KI	792 (21)	00:00:10		
3 TABLE ACCESS FULL	PRODUCTS	767	8437	10 (0)	00:00:01		
4 PARTITION RANGE ALL		489KI	4300KI	741 (17)	00:00:09	1	16
5 TABLE ACCESS FULL	SALES	489KI	4300KI	741 (17)	00:00:09	1	16

A representação tabular é resultado da travessia de cima para baixo, esquerda para direita na árvore de execução



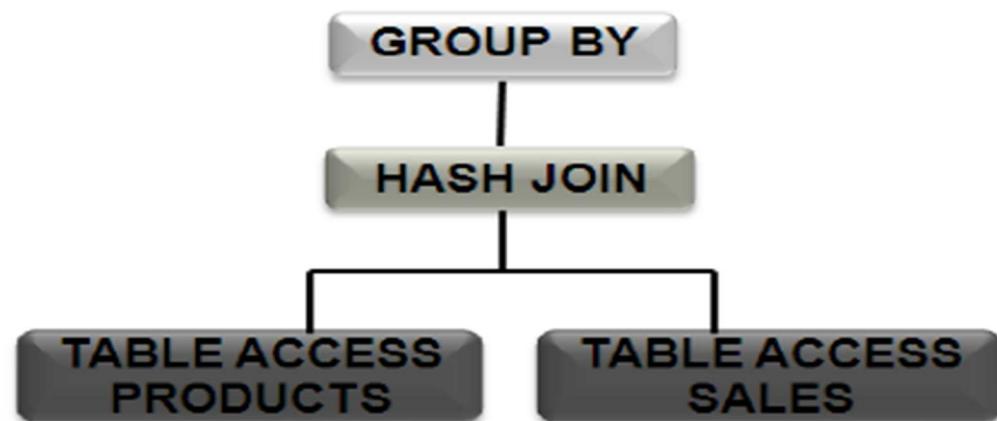
Oracle

```
SELECT      prod_category, AVG(amount_sold)
FROM        sales s, products p
WHERE       p.prod_id = s.prod_id
GROUP BY    prod_category;
```

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart Pstop	
0 SELECT STATEMENT				1140 (100)			
1 HASH GROUP BY		4	80	1140 (45)	00:00:14		
* 2 HASH JOIN		489K	9555K	792 (21)	00:00:10		
3 TABLE ACCESS FULL	PRODUCTS	767	8437	10 (0)	00:00:01		
4 PARTITION RANGE ALL		489K	4300K	741 (17)	00:00:09	1	16
5 TABLE ACCESS FULL	SALES	489K	4300K	741 (17)	00:00:09	1	16

O custo total do plano (linha 0) e de cada operação individual é apresentada.

O custo é resultado de uma representação interna (representa uma unidade de processamento próprio) e é apresentado para permitir comparações entre planos.



Oracle

```
explain plan for
update pedido P1
set VALORTOTAL = (
    select sum(quant*valunit)
        from cliente c, pedido p, itemped i, produto pr
       where c.codcli = p.codcli
         and p.numped = i.numped
         and i.codprod = pr.codprod
         and p.numped = P1.numped);

select * from table(dbms_xplan.display);
```

Editor do Script x Resultado da Consulta x

SQL | Todas as Linhas Extraídas: 24 em 0,031 segundos

PLAN_TABLE_OUTPUT

Plan hash value: 367323205

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time	
0	UPDATE STATEMENT		45217	1148K	542K	(9)	01:48:33	
1	UPDATE	PEDIDO						
2	TABLE ACCESS FULL	PEDIDO	45217	1148K	69	(2)	00:00:01	
3	SORT AGGREGATE		1	65				
* 4	HASH JOIN		1929	122K	12	(9)	00:00:01	
5	TABLE ACCESS FULL	PRODUTO	502	13052	5	(0)	00:00:01	
6	TABLE ACCESS BY INDEX ROWID	ITEMPED	1929	75231	6	(0)	00:00:01	
* 7	INDEX RANGE SCAN	SYS_C0024594	771		2	(0)	00:00:01	

Oracle

```
SQL> EXPLAIN PLAN FOR
  2 Select prod_category, avg(amount_sold)
  3 From   sales s, products p
  4 Where  p.prod_id = s.prod_id
  5 Group by prod_category;
```

Explained.

```
SQL>
SQL> Select plan_table_output
  2 From table(dbms_xplan.display('plan_table',null,'basic'));
```

PLAN_TABLE_OUTPUT

Plan hash value: 504757596

Id Operation	Name
0 SELECT STATEMENT	
1 HASH GROUP BY	
2 HASH JOIN	
3 VIEW	vw_gbc_5
4 HASH GROUP BY	
5 PARTITION RANGE ALL	
6 TABLE ACCESS STORAGE FULL	SALES
7 TABLE ACCESS STORAGE FULL	PRODUCTS

formatação

null: última
sentença

Oracle

```
SQL> Select prod_category, avg(amount_sold)
  2 From sales s, products p
  3 Where p.prod_id = s.prod_id
  4 Group by prod_category;
```

PROD_CATEGORY	AVG(AMOUNT SOLD)
Software/Other	34.1313997
Hardware	1344.50776
Electronics	125.551667
Photo	188.064642
Peripherals and Accessories	108.824588

```
SQL>
```

```
SQL> Select plan_table_output
  2 From table(dbms_xplan.display_cursor(NULL,NULL,'basic'));
```

PLAN_TABLE_OUTPUT

EXPLAINED SQL STATEMENT:

Select prod_category, avg(amount_sold) From sales s, products p Where p.prod_id = s.prod_id Group by prod_category
--

Plan hash value: 504757596

Id Operation	Name
0 SELECT STATEMENT	
1 HASH GROUP BY	
2 HASH JOIN	
3 VIEW	VW_GBC_5
4 HASH GROUP BY	
5 PARTITION RANGE ALL	
6 TABLE ACCESS STORAGE FULL	SALES
7 TABLE ACCESS STORAGE FULL	PRODUCTS

Oracle

```
SELECT plan_table_output
FROM    TABLE(DBMS_XPLAN.DISPLAY('plan_table',null,'basic +predicate +cost'));
```

Id	Operation	Name	Cost (%CPU)
0	SELECT STATEMENT		1101 (44)
1	HASH GROUP BY		1101 (44)
* 2	HASH JOIN		1100 (43)
3	TABLE ACCESS FULL	PRODUCTS	10 (0)
4	VIEW	VW_GBC_5	1089 (44)
5	HASH GROUP BY		1089 (44)
6	PARTITION RANGE ALL		741 (17)
7	TABLE ACCESS FULL	SALES	741 (17)

Predicate Information (identified by operation id):

```
2 - access("P"."PROD_ID"="ITEM_1")
```

Oracle

- Custo: representa uma estimativa de uso de recursos → quanto menor, melhor
 - modelo de custo: I/O, CPU, rede

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0 SELECT STATEMENT				1140 (100)			
1 HASH GROUP BY		4	80	1140 (45)	00:00:14		
* 2 HASH JOIN		489K	9555K	792 (21)	00:00:10		
3 TABLE ACCESS FULL	PRODUCTS	767	8437	10 (0)	00:00:01		
4 PARTITION RANGE ALL		489K	4300K	741 (17)	00:00:09	1	16
5 TABLE ACCESS FULL	SALES	489K	4300K	741 (17)	00:00:09	1	16

Oracle – entendendo o plano de execução

Understanding the execution plan

In order to determine if you are looking at a good execution plan or not, you need to understand how the Optimizer determined the plan in the first place. You should also be able to look at the execution plan and assess if the Optimizer has made any mistake in its estimations or calculations, leading to a suboptimal plan. The components to assess are:

- **Cardinality** – Estimate of the number of rows coming out of each of the operations.
- **Access method** – The way in which the data is being accessed, via either a table scan or index access.
- **Join method** – The method (e.g., hash, sort-merge, etc.) used to join tables with each other.
- **Join type** – The type of join (e.g., outer, anti, semi, etc.).
- **Join order** – The order in which the tables are joined to each other.
- **Partition pruning** – Are only the necessary partitions being accessed to answer the query?
- **Parallel Execution** – In case of parallel execution, is each operation in the plan being conducted in parallel? Is the right data redistribution method being used?

Oracle – entendendo o plano de execução

- Cardinalidade: número estimado de linhas de cada operação

```
SQL> SELECT employee_id, last_name, job_id  
  2  FROM hr.employees  
  3  WHERE job_id = 'AD_VP';  
  
SQL> Select plan_table_output  
  2  From table(dbms_xplan.display_cursor(null,null,'TYPICAL'));
```

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT				2 (100)	
1 TABLE ACCESS BY INDEX ROWID EMPLOYEES	6	126	2 (0) 00:00:01		
* 2 INDEX RANGE SCAN	EMP_JOB_IX	6	1 (0) 00:00:01		

Predicate Information (identified by operation id):

```
2 - access("JOB_ID">'AD_VP')
```

Figure 9: The CARDINALITY estimate is found in the Rows column of the execution plan

Oracle – entendendo o plano de execução

- Método de acesso

- **Full table scan** – lê todas as linhas de uma tabela e filtra as linhas que não atendem aos predicados da cláusula WHERE
- Table access by ROWID – especifica a localização das linhas que serão recuperadas
- Index unique scan – apenas uma linha é retornada após busca em árvore B+
- Index range scan – acesso ao índice para recuperar ROWID das linhas que serão recuperadas

	Id	Operation		Name		Rows		Bytes		Cost (%CPU)		Time		Pstart		Pstop	
	0	SELECT STATEMENT								1140 (100)							
	1	HASH GROUP BY				4		80		1140 (45)		00:00:14					
*	2	HASH JOIN				489K		9555K		792 (21)		00:00:10					
	3	TABLE ACCESS FULL		PRODUCTS		767		8437		10 (0)		00:00:01					
	4	PARTITION RANGE ALL				489K		4300K		741 (17)		00:00:09		1		16	
	5	TABLE ACCESS FULL		SALES		489K		4300K		741 (17)		00:00:09		1		16	

Oracle – entendendo o plano de execução

● Método de acesso

- Full table scan – lê todas as linhas de uma tabela e filtra as linhas que não atendem aos predicados da cláusula WHERE
- Table access by ROWID – especifica a localização das linhas que serão recuperadas
- **Index unique scan** – apenas uma linha é retornada após busca em árvore B+
- Index range scan – acesso ao índice para recuperar ROWID das linhas que serão recuperadas

	Id	Operation		Name		Rows		Bytes		Cost (%CPU)		Time	
	0	SELECT STATEMENT								1 (100)			
	1	TABLE ACCESS BY INDEX ROWID		PROMOTIONS		1		40		1 (0)		00:00:01	
*	2	INDEX UNIQUE SCAN		PROMO_PK		1				0 (0)			

Predicate Information (identified by operation id):

2 - access("PROMO_ID"=9999)

Equality predicate on
primary key index

Oracle – entendendo o plano de execução

- Método de acesso

- Full table scan – lê todas as linhas de uma tabela e filtra as linhas que não atendem aos predicados da cláusula WHERE
- Table access by ROWID – especifica a localização das linhas que serão recuperadas
- Index unique scan – apenas uma linha é retornada após busca em árvore B+
- **Index range scan** – acesso ao índice para recuperar ROWID das linhas que serão recuperadas

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT				2 (100)	
1 TABLE ACCESS BY INDEX ROWID	PROMOTIONS	1	40	2 (0)	00:00:01
* 2 INDEX RANGE SCAN	PROMO_PK	1	1	1 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("PROMO_ID">>9998)

Non equality predicate
on unique index

Oracle – entendendo o plano de execução

- Método de junção

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				431 (100)			
1	HASH GROUP BY		71	2769	431 (11)	00:00:01		
* 2	HASH JOIN		918KI	34MI	399 (4)	00:00:01		
3	TABLE ACCESS FULL	PRODUCTS	72	2160	3 (0)	00:00:01		
4	PARTITION RANGE ALL		918KI	8075KI	392 (3)	00:00:01	1	28
5	TABLE ACCESS FULL	SALES	918KI	8075KI	392 (3)	00:00:01	1	28

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				16625 (100)			
1	HASH GROUP BY		71	2769	16625 (1)	00:00:01		
2	NESTED LOOPS							
3	NESTED LOOPS							
4	TABLE ACCESS FULL	PRODUCTS	918KI	34MI	16593 (1)	00:00:01		
5	PARTITION RANGE ALL		72	2160	3 (0)	00:00:01		
6	BITMAP CONVERSION TO ROWIDS						1	28
* 7	BITMAP INDEX SINGLE VALUE	SALES_PROD_BIX					1	28
8	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	12762	112KI	16593 (1)	00:00:01	1	1

Oracle – entendendo o plano de execução

- Método de junção

Id Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Pstart Pstop
0 SELECT STATEMENT					2628 (100)		
1 HASH GROUP BY		71	2769		2628 (3)	00:00:01	
2 MERGE JOIN		918KI	34MI		2596 (2)	00:00:01	
3 TABLE ACCESS BY INDEX ROWID	PRODUCTS	72	2160		2 (0)	00:00:01	
4 INDEX FULL SCAN	PRODUCTS_PK	72			1 (0)	00:00:01	
* 5 SORT JOIN		918KI	8075KI	35MI	2594 (2)	00:00:01	
6 PARTITION RANGE ALL		918KI	8075KI		392 (3)	00:00:01	1 28
7 TABLE ACCESS FULL	SALES	918KI	8075KI		392 (3)	00:00:01	1 28

Oracle – entendendo o plano de execução

- Ordem de junção

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop	
0 SELECT STATEMENT				713 (100)				
1 HASH GROUP BY		2193	107Ki	713 (8)	00:00:01			
* 2 HASH JOIN		918Ki	43Mi	681 (3)	00:00:01			
3 3 TABLE ACCESS FULL	CUSTOMERS	55500	812Ki	278 (1)	00:00:01			
* 4 HASH JOIN		918Ki	30Mi	399 (4)	00:00:01			
5 1 TABLE ACCESS FULL	PRODUCTS	72	1512	3 (0)	00:00:01			
6 PARTITION RANGE ALL		918Ki	12Mi	392 (3)	00:00:01	1	28	
7 2 TABLE ACCESS FULL	SALES	918Ki	12Mi	392 (3)	00:00:01	1	28	

Bibliografia

- Elmasri, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados.** 6^a edição, Editora Pearson, 2011.

Agradecimentos

- Contém slides traduzidos dos originais disponibilizados por Elmasri & Navathe
- Contém slides disponibilizados pela Prof^a Sandra de Amo (FACOM/UFU)
- Contém slides disponibilizados por Robert Haas, EnterpriseDB