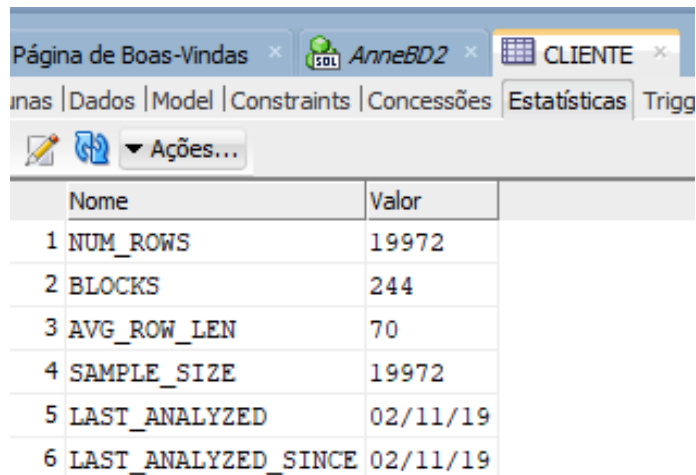
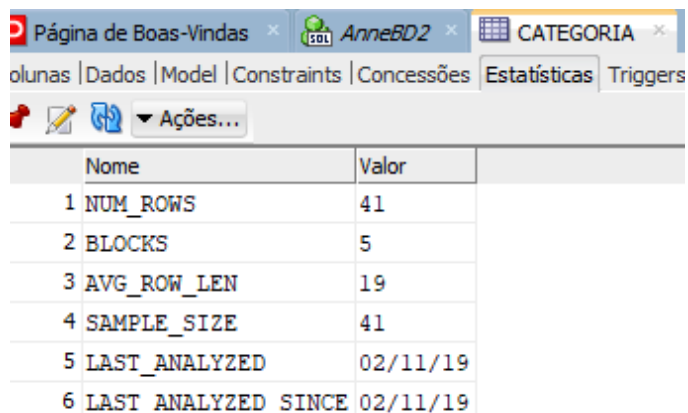


Questão 5) (1 ponto) Solicite ao SGBD que atualize as estatísticas de índices e tabelas. Apresente o resultado das estatísticas para cada tabela.

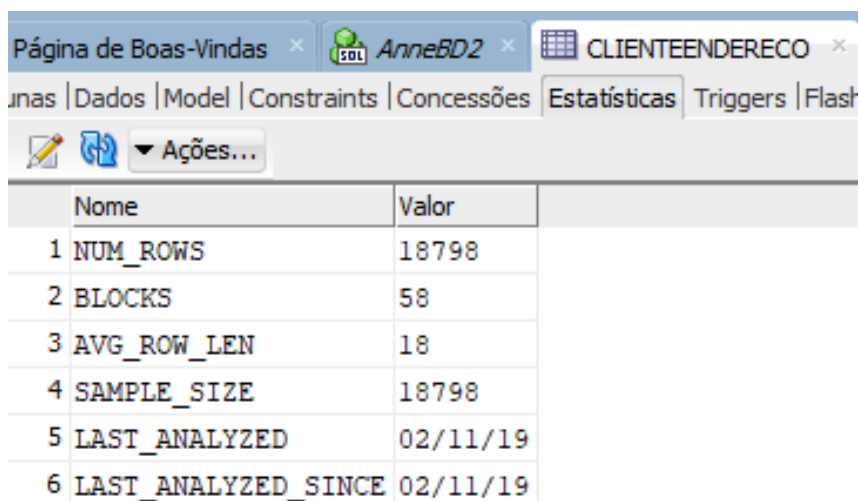
`ANALYZE TABLE categoria compute statistics;`



	Nome	Valor
1	NUM_ROWS	19972
2	BLOCKS	244
3	AVG_ROW_LEN	70
4	SAMPLE_SIZE	19972
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19



	Nome	Valor
1	NUM_ROWS	41
2	BLOCKS	5
3	AVG_ROW_LEN	19
4	SAMPLE_SIZE	41
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19



	Nome	Valor
1	NUM_ROWS	18798
2	BLOCKS	58
3	AVG_ROW_LEN	18
4	SAMPLE_SIZE	18798
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x DESCRICAO x

unais | Dados | Model | Constraints | Concessões | Estatísticas | Triggers

✎ 🔄 ⚙️ Ações...

	Nome	Valor
1	NUM_ROWS	762
2	BLOCKS	28
3	AVG_ROW_LEN	136
4	SAMPLE_SIZE	762
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x DETALHESPEDIDO x

unais | Dados | Model | Constraints | Concessões | Estatísticas | Triggers | Fl

✎ 🔄 ⚙️ Ações...

	Nome	Valor
1	NUM_ROWS	113480
2	BLOCKS	622
3	AVG_ROW_LEN	31
4	SAMPLE_SIZE	113480
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x ENDERECO x

Colunas | Dados | Model | Constraints | Concessões | Estatísticas | Trig

📌 ✎ 🔄 ⚙️ Ações...

	Nome	Valor
1	NUM_ROWS	19614
2	BLOCKS	244
3	AVG_ROW_LEN	66
4	SAMPLE_SIZE	19614
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x IDIOMA x

unais | Dados | Model | Constraints | Concessões | Estatísticas | T

✎ 🔄 ⚙️ Ações...

	Nome	Valor
1	NUM_ROWS	7
2	BLOCKS	5
3	AVG_ROW_LEN	14
4	SAMPLE_SIZE	7
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x MODELO x		
Colunas Dados Model Constraints Concessões Estatísticas Triggers		
Ações...		
	Nome	Valor
1	NUM_ROWS	128
2	BLOCKS	5
3	AVG_ROW_LEN	23
4	SAMPLE_SIZE	128
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x PEDIDO x		
Colunas Dados Model Constraints Concessões Estatísticas Triggers		
Ações...		
	Nome	Valor
1	NUM_ROWS	30935
2	BLOCKS	496
3	AVG_ROW_LEN	89
4	SAMPLE_SIZE	30935
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x PRODUTO x		
Colunas Dados Model Constraints Concessões Estatísticas Triggers		
Ações...		
	Nome	Valor
1	NUM_ROWS	504
2	BLOCKS	5
3	AVG_ROW_LEN	64
4	SAMPLE_SIZE	504
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x TRANSPORTADORA x		
Colunas Dados Model Constraints Concessões Estatísticas Triggers Flags		
Ações...		
	Nome	Valor
1	NUM_ROWS	5
2	BLOCKS	5
3	AVG_ROW_LEN	34
4	SAMPLE_SIZE	5
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Página de Boas-Vindas x AnneBD2 x VENDEDOR x		
Colunas Dados Model Constraints Concessões Estatísticas Triggers		
Ações...		
	Nome	Valor
1	NUM_ROWS	17
2	BLOCKS	5
3	AVG_ROW_LEN	102
4	SAMPLE_SIZE	17
5	LAST_ANALYZED	02/11/19
6	LAST_ANALYZED_SINCE	02/11/19

Questão 6) (5 pontos) Para cada consulta da Questão 4, tente realizar a otimização de cada consulta, para tanto utilize o comando explain plan ou a ferramenta visual disponível no SQLDeveloper.

Contudo, em todos os casos, deve-se apresentar as respostas do custo anterior à otimização e o do novo custo após a otimização para cada consulta, discutindo qual foi a hipótese seguida para a otimização.

Apresente “printscreens” ou apresente o resultado do comando explain plan. Nos casos em que foi necessário recorrer ao fornecimento de hints para o gerenciador, também os apresente.

Explain plan for SELECT count(cliente.primeironome)

FROM CLIENTE, PEDIDO, VENDEDOR

WHERE cliente.codigo = pedido.codigocliente and vendedor.codigo=pedido.codigovendedor and vendedor.primeironome= 'Michael'

and vendedor.sobrenome='Blythe';

SELECT plan_table_output FROM TABLE(dbms_xplan.display());

1. Query

Sem indice.

```
PLAN_TABLE_OUTPUT
-----
Plan hash value: 3910424444

   Id | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---- |-----
0 | SELECT STATEMENT    |      |    1 |    23 |    140 (1)| 00:00:02 |
1 |   SORT AGGREGATE    |      |    1 |    23 |          |          |
2 |    HASH JOIN        |      |    13 |    299 |    140 (1)| 00:00:02 |
3 |     TABLE ACCESS FULL| VENDEDOR |    1 |    17 |     3 (0)| 00:00:01 |
4 |     TABLE ACCESS FULL| PEDIDO  |  3297 | 19782 |    137 (1)| 00:00:02 |

PLAN_TABLE_OUTPUT
-----
Predicate Information (identified by operation id):

   2 - access("VENDEDOR"."CODIGO"="PEDIDO"."CODIGOVENDEDOR")
   3 - filter("VENDEDOR"."PRIMEIRONOME"='Michael' AND
             "VENDEDOR"."SOBRENOME"='Blythe')
   4 - filter("PEDIDO"."CODIGOVENDEDOR" IS NOT NULL)

19 linhas selecionadas.
```

Faz um full access (4) em PEDIDO com custo de 137 e retornando 3297 tuplas

Full access em VENDEDOR (3) com custo de 3 e retornando apenas 1 tupla.

Hash join (2) ao custo de 140 retornado 13 tuplas.

Com Indice.

```
Plan hash value: 4284770801

   Id | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---- |-----
0 | SELECT STATEMENT    |      |    1 |    23 |     4 (0)| 00:00:01 |
1 |   SORT AGGREGATE    |      |    1 |    23 |          |          |
2 |    NESTED LOOPS     |      |    13 |    299 |     4 (0)| 00:00:01 |
3 |     TABLE ACCESS FULL| VENDEDOR |    1 |    17 |     3 (0)| 00:00:01 |
4 |     BITMAP CONVERSION COUNT |      |  220 | 1320 |     4 (0)| 00:00:01 |
5 |      BITMAP INDEX SINGLE VALUE| IDXPEDCODIGOVENDEDOR |      |      |          |          |

PLAN_TABLE_OUTPUT
-----
Predicate Information (identified by operation id):

   3 - filter("VENDEDOR"."PRIMEIRONOME"='Michael' AND "VENDEDOR"."SOBRENOME"='Blythe')
   5 - access("VENDEDOR"."CODIGO"="PEDIDO"."CODIGOVENDEDOR")
      filter("PEDIDO"."CODIGOVENDEDOR" IS NOT NULL)

19 linhas selecionadas.

SELECT count(cliente.primeironome)
FROM CLIENTE, PEDIDO, VENDEDOR
WHERE cliente.codigo = pedido.codigocliente and vendedor.codigo=pedido.codigovendedor and vendedor.primeironome= 'Michael'
and vendedor.sobrenome='Blythe';
```

Full access APENAS em VENDEDOR (3) com custo de 3 e retornando apenas 1 tupla.

As outras operações que tinham custo 137 e 140 foram reduzidas para 4.

O índice evitou o full access a table PEDIDO que era uma operação muito cara e impactava nas outras de custo parecido. Com o uso do index, o acesso a PEDIDO retornou apenas 220 tuplas.

Tendo no select um count e um predicado indexável, possibilitou a criação de um índice do tipo bitmap sobre codigovendedor.

2. Query

Sem índice/ Com indice

```
PLAN_TABLE_OUTPUT
-----
Plan hash value: 2564383598

-----
| Id | Operation          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----
| 0 | SELECT STATEMENT   |                     |      |      |             |          |
| 1 |   SORT ORDER BY    |                     |      |      |             |          |
| 2 |     HASH GROUP BY   |                     |      |      |             |          |
|* 3 |       HASH JOIN     |                     | 30935 | 845K |    141 (2)  | 00:00:02 |
| 4 |         TABLE ACCESS FULL | TRANSPORTADORA |      |      |             |          |
|* 5 |           TABLE ACCESS FULL | PEDIDO          | 30935 | 271K |     137 (1) | 00:00:02 |
-----

PLAN_TABLE_OUTPUT
-----

Predicate Information (identified by operation id):

   3 - access("PEDIDO"."CODIGOTRANSPORTADORA"="TRANSPORTADORA"."CODIGO")
   5 - filter("PEDIDO"."DTENVIO" IS NOT NULL)
```

Índices bitmap são adequados apenas para otimizar consultas em colunas com baixa cardinalidade. Usar o Bitmap não seria adequada para essa consulta. Não foi possível otimizar a consulta.

3. Query

Sem indice

```
PLAN_TABLE_OUTPUT
-----
Plan hash value: 4235772510

-----
| Id | Operation          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----
| 0 | SELECT STATEMENT   |                     |      |      |             |          |
|* 1 |   TABLE ACCESS FULL | PRODUTO             |      |      |             |          |
| 2 |     SORT AGGREGATE  |                     |      |      |             |          |
| 3 |       TABLE ACCESS FULL | PRODUTO             | 504   | 2016 |      3 (0)  | 00:00:01 |
-----

PLAN_TABLE_OUTPUT
-----

Predicate Information (identified by operation id):

   1 - filter("PRODUTO"."PRECO"= (SELECT MAX("PRODUTO"."PRECO") FROM
        "PRODUTO" "PRODUTO"))

16 linhas selecionadas.
```

Faz um full scan em PRODUTO duas vezes retornando no primeiro 504 tuplas ao custo de 3.

Aplica o sort aggregate no primeiro full scan e usa o valor como filtro para o segundo que retorna apenas 5 tuplas.

Com indice

```
Explicado.

PLAN_TABLE_OUTPUT
-----
Plan hash value: 4160308425

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|
| 0 | SELECT STATEMENT | | 5 | 160 | 4 (0) | 00:00:01 |
|* 1 | INDEX RANGE SCAN | IDXPRODPRECOCODNOME | 5 | 160 | 2 (0) | 00:00:01 |
| 2 | SORT AGGREGATE | | 1 | 4 | | |
| 3 | INDEX FULL SCAN (MIN/MAX) | IDXPRODPRECOCODNOME | 1 | 4 | 2 (0) | 00:00:01 |

PLAN_TABLE_OUTPUT
-----
Predicate Information (identified by operation id):

1 - access("PRODUTO"."PRECO"= (SELECT MAX("PRODUTO"."PRECO") FROM "PRODUTO" "PRODUTO"))

15 linhas selecionadas.
```

O uso do índice não alterou muito o uso de CPU, porém diminuiu o uso da memória retornando o valor do filtro já na leitura. IDXPRODPRECOCODNOME ON PRODUTO (PRECO, CODIGO, NOME) o preço como o primeiro atributo do index foi uma fator bastante relevante para a otimização de espaço.

4. Query

Sem índice

```
PLAN_TABLE_OUTPUT
-----
Plan hash value: 1934750648

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|
| 0 | SELECT STATEMENT | | 1547 | 34034 | 211 (4) | 00:00:03 |
| 1 | SORT ORDER BY | | 1547 | 34034 | 211 (4) | 00:00:03 |
|* 2 | FILTER | | | | | |
| 3 | HASH GROUP BY | | 1547 | 34034 | 211 (4) | 00:00:03 |
|* 4 | HASH JOIN | | 30935 | 664K | 206 (1) | 00:00:03 |
| 5 | TABLE ACCESS FULL | PEDIDO | 30935 | 120K | 137 (1) | 00:00:02 |

PLAN_TABLE_OUTPUT
-----
| 6 | TABLE ACCESS FULL | CLIENTE | 19972 | 351K | 68 (0) | 00:00:01 |

Predicate Information (identified by operation id):

2 - filter(COUNT(*)>=15)
4 - access("CLIENTE"."CODIGO"="PEDIDO"."CODIGOCLIENTE")

19 linhas selecionadas.
```

O impacto dessa leitura são os dois acessos FULL a CLIENTE e PEDIDO.

Com indice

```
PLAN_TABLE_OUTPUT
-----
Plan hash value: 516446904

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-----|-----|-----|-----|-----|-----|
| 0 | SELECT STATEMENT | | 999 | 21978 | 51 (12) | 00:00:01 |
| 1 | SORT ORDER BY | | 999 | 21978 | 51 (12) | 00:00:01 |
|* 2 | FILTER | | | | | |
| 3 | HASH GROUP BY | | 999 | 21978 | 51 (12) | 00:00:01 |
|* 4 | HASH JOIN | | 30935 | 664K | 47 (5) | 00:00:01 |
| 5 | INDEX FAST FULL SCAN | IDXPEDCODIGOCLIENTE | 30935 | 120K | 20 (0) | 00:00:01 |

PLAN_TABLE_OUTPUT
-----
| 6 | INDEX FAST FULL SCAN | IDXCLICODIGONOME | 19972 | 351K | 25 (0) | 00:00:01 |

Predicate Information (identified by operation id):

2 - filter(COUNT(*)>=15)
4 - access("CLIENTE"."CODIGO"="PEDIDO"."CODIGOCLIENTE")

19 linhas selecionadas.
```

O uso dos dois índices tem um impacto significativo no custo de CPU e no número de tuplas retornadas no Group by, utilização de menos espaço para armazenamento na memória - uso do operador index fast scan, que é mais rapido que o table access full, consequentemente diminuído o tempo da consulta.

5. Query

Sem indice

PLAN_TABLE_OUTPUT

Plan hash value: 1617003003

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	69	210 (2)	00:00:03
* 1	FILTER		1			
2	HASH GROUP BY		1	69	210 (2)	00:00:03
3	NESTED LOOPS		1			
4	NESTED LOOPS		1	69	209 (1)	00:00:03
5	NESTED LOOPS		1	37	208 (1)	00:00:03

PLAN_TABLE_OUTPUT

* 6	HASH JOIN		1	24	206 (1)	00:00:03
* 7	TABLE ACCESS FULL	CLIENTE	1	16	69 (2)	00:00:01
8	TABLE ACCESS FULL	PEDIDO	30935	241K	137 (1)	00:00:02
* 9	INDEX RANGE SCAN	SYS_C001150084	4	52	2 (0)	00:00:01
* 10	INDEX UNIQUE SCAN	SYS_C001150067	1		0 (0)	00:00:01
11	TABLE ACCESS BY INDEX ROWID	PRODUTO	1	32	1 (0)	00:00:01
12	SORT AGGREGATE		1	13		
13	VIEW		1	13	210 (2)	00:00:03
14	SORT GROUP BY		1	69	210 (2)	00:00:03
15	NESTED LOOPS		1			
16	NESTED LOOPS		1	69	209 (1)	00:00:03

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
14	SORT GROUP BY		1	69	210 (2)	00:00:03
15	NESTED LOOPS		1			
16	NESTED LOOPS		1	69	209 (1)	00:00:03

PLAN_TABLE_OUTPUT

17	NESTED LOOPS		1	37	208 (1)	00:00:03
* 18	HASH JOIN		1	24	206 (1)	00:00:03
* 19	TABLE ACCESS FULL	CLIENTE	1	16	69 (2)	00:00:01
20	TABLE ACCESS FULL	PEDIDO	30935	241K	137 (1)	00:00:02
* 21	INDEX RANGE SCAN	SYS_C001150084	4	52	2 (0)	00:00:01
* 22	INDEX UNIQUE SCAN	SYS_C001150067	1		0 (0)	00:00:01
23	TABLE ACCESS BY INDEX ROWID	PRODUTO	1	32	1 (0)	00:00:01

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

```
1 - filter(COUNT(*)= (SELECT MAX("COUNTING") FROM (SELECT "P"."NOME"
"NOME","P"."PRECO" "PRECO",COUNT(*) "COUNTING" FROM "CLIENTE" "C","PEDIDO"
"PD","DETALHESPEDIDO" "D","PRODUTO" "P" WHERE "P"."CODIGO"="D"."CODIGOPRODUTO" AND
"PD"."CODIGO"="D"."CODIGOPEDIDO" AND "C"."CODIGO"="PD"."CODIGOCLIENTE" AND
"C"."SOBRENOME"='Taylor' AND "C"."PRIMEIRONOME"='Jennifer' GROUP BY
"P"."NOME","P"."PRECO") "froms_subquery5_005"))
6 - access("CLIENTE"."CODIGO"="PEDIDO"."CODIGOCLIENTE")
7 - filter(("CLIENTE"."SOBRENOME"='Taylor' AND "CLIENTE"."PRIMEIRONOME"='Jennifer')
9 - access("PEDIDO"."CODIGO"="DETALHESPEDIDO"."CODIGOPEDIDO")
10 - access("PRODUTO"."CODIGO"="DETALHESPEDIDO"."CODIGOPRODUTO")
```

PLAN_TABLE_OUTPUT

```
18 - access("C"."CODIGO"="PD"."CODIGOCLIENTE")
19 - filter("C"."SOBRENOME"='Taylor' AND "C"."PRIMEIRONOME"='Jennifer')
21 - access("PD"."CODIGO"="D"."CODIGOPEDIDO")
22 - access("P"."CODIGO"="D"."CODIGOPRODUTO")
```

48 linhas selecionadas.

Nessa primeira etapa, sem índices, o access full de Pedido retorna 30935 tuplas, o que tem um impacto grande no restante da consulta fazendo com que o custo de 12 operações fique acima de 200.

Com indice

Plan hash value: 1505496820						
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	69	37 (6)	00:00:01
* 1	FILTER					
2	HASH GROUP BY		1	69	37 (6)	00:00:01
* 3	HASH JOIN		7	483	36 (3)	00:00:01
4	NESTED LOOPS		7	259	32 (0)	00:00:01
5	NESTED LOOPS		2	48	28 (0)	00:00:01
PLAN_TABLE_OUTPUT						
* 6	INDEX FAST FULL SCAN	IDXCLICODIGONOME	1	16	25 (0)	00:00:01
7	TABLE ACCESS BY INDEX ROWID	PEDIDO	2	16	3 (0)	00:00:01
* 8	INDEX RANGE SCAN	IDXPEDCODIGOCLIENTE	2		1 (0)	00:00:01
* 9	INDEX RANGE SCAN	SYS_C001150084	4	52	2 (0)	00:00:01
10	INDEX FAST FULL SCAN	IDXPRODPRECOCODNOME	504	16128	3 (0)	00:00:01
11	SORT AGGREGATE		1	13		
12	VIEW		7	91	37 (6)	00:00:01
13	SORT GROUP BY		7	483	37 (6)	00:00:01
* 14	HASH JOIN		7	483	36 (3)	00:00:01
15	NESTED LOOPS		7	259	32 (0)	00:00:01
16	NESTED LOOPS		2	48	28 (0)	00:00:01
PLAN_TABLE_OUTPUT						

nlha	Query Builder
* 17	INDEX FAST FULL SCAN
18	TABLE ACCESS BY INDEX ROWID
* 19	INDEX RANGE SCAN
* 20	INDEX RANGE SCAN
21	INDEX FAST FULL SCAN

Predicate Information (identified by operation id):

1 - filter(COUNT(*)= (SELECT MAX("COUNTING") FROM (SELECT "P"."NOME" "NOME","P"."PRECO"

PLAN_TABLE_OUTPUT

```
"PRECO",COUNT(*) "COUNTING" FROM "CLIENTE" "C","PEDIDO" "PD","DETALHESPEDIDO" "D","PRODUTO" "P"
WHERE "P"."CODIGO"="D"."CODIGOPRODUTO" AND "PD"."CODIGO"="D"."CODIGOPEDIDO" AND
"C"."CODIGO"="PD"."CODIGOCLIENTE" AND "C"."SOBRENOME"='Taylor' AND "C"."PRIMEIRONOME"='Jennifer'
GROUP BY "P"."NOME","P"."PRECO") "from$_subquery$005"))
3 - access("PRODUTO"."CODIGO"="DETALHESPEDIDO"."CODIGOPRODUTO")
6 - filter(("CLIENTE"."SOBRENOME"='Taylor' AND "CLIENTE"."PRIMEIRONOME"='Jennifer')
8 - access("CLIENTE"."CODIGO"="PEDIDO"."CODIGOCLIENTE")
9 - access("PEDIDO"."CODIGO"="DETALHESPEDIDO"."CODIGOPEDIDO")
14 - access("P"."CODIGO"="D"."CODIGOPRODUTO")
17 - filter("C"."SOBRENOME"='Taylor' AND "C"."PRIMEIRONOME"='Jennifer')
19 - access("C"."CODIGO"="PD"."CODIGOCLIENTE")
```

PLAN_TABLE_OUTPUT

20 - access("PD"."CODIGO"="D"."CODIGOPEDIDO")

Essa consulta foi bastante otimizada: Com a utilização de índices, PEDIDO retorna apenas 2 tuplas fazendo com que o custo de CPU fique abaixo de 40 em todas as operações.

Indexes Criados

1. CREATE BITMAP INDEX IDXPEDCODIGOVENDEDOR ON PEDIDO (CODIGOVENDEDOR);
2. CREATE INDEX IDXPEDCODTRANSPORTADORA ON PEDIDO (CODIGOTRANSPORTADORA);
3. CREATE UNIQUE INDEX IDXPRODPRECOCODNOME ON PRODUTO (PRECO, CODIGO, NOME);
4. CREATE INDEX IDXCLICODIGONOME ON CLIENTE (CODIGO, PRIMEIRONOME, NOME, SOBRENOME);
5. CREATE INDEX IDXPEDCODIGOCLIENTE ON PEDIDO (CODIGOCLIENTE);
6. CREATE INDEX IDXPRODCODIGONOME ON PRODUTO (CODIGO, NOME);