

Extend Statistics - Multicolumn Statistics

Extendend Statistics é uma feature do 11g que ajuda o otimizador baseado em custo (CBO) tomar melhores decisões utilizando estatísticas em grupo de colunas, quando várias colunas de um tabela estão na clausula where da consulta.

Além das estatísticas padrão, histogramas também podem ser criados nestes grupos de colunas melhorando a estimativa de custos quando existe um desvio na distribuição dos dados do grupo de colunas.

Para o CBO decidir como executar um consulta, ele utiliza as estatísticas disponíveis para calcular o custo de possíveis metodos de acesso. Seletividade por exemplo, é um dos fatores utilizados neste calculo e consequentemente na escolha do melhor metodo de acesso. Antes do 11g, as estatísticas so podiam ser criadas em colunas separadamente. A inabilidade do CBO de perceber o relacionamento entre colunas de uma tabela limitava significativamente a exatidão na estimativa do custo.

Para demonstrar como funciona a feature primeiro vamos criar uma tabela de pedidos

```
ALESSANDRO@orcl> create table pedidos
2 as
select level nr
3   4           , 'Cliente ' || to_char(level) Nome_Cliente
5           , case
6           when level <= 500 then 'Ipanema'
when level <= 550 then 'Lagoa'
7   8           when level <= 600 then 'Olinda'
9           when level <= 650 then 'Centro'
10          when level <= 700 then 'Bexiga'
11          when level <= 750 then 'Areinha'
when level <= 800 then 'Aldeota'
12  13         when level <= 850 then 'Itapoa'
14          when level <= 900 then 'Pajucara'
15          else 'Bessa'
16          end Bairro
, case
17  18         when level < 500 then 'RJ'
when level <= 550 then 'MG'
when level <= 600 then 'PE'
19  20  21         when level <= 650 then 'TE'
22          when level <= 700 then 'SP'
when level <= 750 then 'MA'
23  24         when level <= 800 then 'CE'
25          when level <= 850 then 'BA'
26          when level <= 900 then 'AL'
else 'PB'
27  28         end UF
from dual
29  30 connect by level <= 1000
/
31
Table created.
```

Vamos dar uma olhada no conteúdo da tabela

```
ALESSANDRO@orcl> select bairro, count(*)
2  from pedidos
3  group by bairro
4  order by bairro desc;
```

BAIRRO	COUNT(*)
Pajucara	50
Olinda	50
Lagoa	50
Itapoa	50
Ipanema	500
Centro	50
Bexiga	50
Bessa	100
Areinha	50
Aldeota	50

10 rows selected.

```
ALESSANDRO@orcl> select uf, count(*)
2  from pedidos
3  group by uf
4  order by count(*) desc;
```

UF	COUNT(*)
RJ	499
PB	100
MG	51
TE	50
MA	50
CE	50
BA	50
AL	50
SP	50
PE	50

10 rows selected.

Temos então 10 bairros e 10 UF, com uma associação clara entre bairro e UF.

Vamos então gerar as estatísticas

```
ALESSANDRO@orcl> exec dbms_stats.gather_table_stats(user,'pedidos');
```

PL/SQL procedure successfully completed.

Ok, Agora vamos dar uma olhada na seguinte query.

```
ALESSANDRO@orcl> explain plan
2  for
3  select * from pedidos
4  where bairro='Ipanema'
```

```
5 and UF = 'RJ'
6 /
```

```
1* select * from table(dbms_xplan.display)
ALESSANDRO@orcl> /
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 3882451908
```

```
-----
| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)|
Time |                    |           |      |      |             |
-----|-----|-----|-----|-----|-----|
|  0 | SELECT STATEMENT   |           |    10 |    260 |    4 (0)|
00:00:01 |
|*  1 | TABLE ACCESS FULL| PEDIDOS   |    10 |    260 |    4 (0)|
00:00:01 |
-----
```

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

```
PLAN_TABLE_OUTPUT
```

```
-----
1 - filter("BAIRRO"='Ipanema' AND "UF"='RJ')
```

```
13 rows selected.
```

```
ALESSANDRO@orcl>
```

Como não geramos histogramas o CBO previu 10 linhas. Isto aconteceu por que o predicado bairro='Ipanema' tem 10 valores distintos, o mesmo acontece com UF='RJ' . O calculo que o CBO faz : 1000 linhas *(1/10)*(1/10) = 10 linhas.

Veja o que acontece quando faço esta pesquisa agora

```
ALESSANDRO@orcl> explain plan
2 for
3 select * from pedidos
4 where bairro='Ipanema' and UF='PE';
```

```
Explained.
```

```
ALESSANDRO@orcl> select * from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 3882451908
```

```

-----
-----
| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)|
Time |                    |           |      |      |             |
-----
|  0 | SELECT STATEMENT   |           |    10 |   260 |      4  (0)|
00:00:01 |                    |           |      |      |             |
|*  1 | TABLE ACCESS FULL| PEDIDOS   |    10 |   260 |      4  (0)|
00:00:01 |                    |           |      |      |             |
-----
-----

```

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

```
1 - filter("BAIRRO"='Ipanema' AND "UF"='PE')
```

13 rows selected.

Esta consulta não retorna linhas e ainda assim o resultado é o mesmo.

Vamos dar uma ajudinha para o otimizador gerando histogramas

```
ALESSANDRO@orcl> exec dbms_stats.gather_table_stats(user, 'pedidos',
method_opt=>'FOR ALL COLUMNS');
```

PL/SQL procedure successfully completed.

```
ALESSANDRO@orcl> explain plan
2 for
3 select * from pedidos
4 where bairro='Ipanema'
5 and uf='RJ';
```

Explained.

```
ALESSANDRO@orcl> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 3882451908

```

-----
-----
| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)|
Time |                    |           |      |      |             |
-----
|  0 | SELECT STATEMENT   |           |    250 |  6500 |      4  (0)|
00:00:01 |                    |           |      |      |             |
-----
-----

```

```
|* 1 | TABLE ACCESS FULL| PEDIDOS | 250 | 6500 | 4 (0) |
00:00:01 |
```


Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("BAIRRO"='Ipanema' AND "UF"='RJ')

13 rows selected.

Agora a previsão foi de 250 linhas. Isto foi calculado através do histogramas que dizem ao CBO que ipanema ocorre 500 vezes em 1000 e rj ocorre 499 vezes em 100

E para Ipanema em PE temos :

```
ALESSANDRO@orcl> explain plan
 2 for
 3 select * from pedidos
 4 where bairro='Ipanema'
 5 and uf='PE';
```

Explained.

```
ALESSANDRO@orcl> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 3882451908

```
-----  
-----  
| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)|  
Time |  
-----  
-----  
| 0 | SELECT STATEMENT  |           | 25   | 650   | 4 (0) |  
00:00:01 |  
|* 1 | TABLE ACCESS FULL| PEDIDOS  | 25   | 650   | 4 (0) |  
00:00:01 |
```


Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("BAIRRO"='Ipanema' AND "UF"='PE')

13 rows selected.

Ai está 25 linhas.

Antes do 11g isso é o melhor que podíamos conseguir. Com multicolumns statistics o calculo fica bem mais exato

```
ALESSANDRO@orcl> exec
dbms_stats.gather_table_stats(user,'pedidos',method_opt=>'FOR COLUMNS
(bairro,uf)');
```

```
ALESSANDRO@orcl> explain plan
2 for
3 select * from pedidos
4 where bairro='Ipanema'
5 and uf='PE';
```

Explained.

```
ALESSANDRO@orcl> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 3882451908

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	27	4 (0)
* 1	TABLE ACCESS FULL	PEDIDOS	1	27	4 (0)

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("BAIRRO"='Ipanema' AND "UF"='PE')

13 rows selected.

```
ALESSANDRO@orcl>
```

Agora sim. Como o otimizador não vai calcular 0 (zero) ele usa 1. Quanto mais preciso for o calculo, melhor vai ser o metodo de acesso que o otimizador vai escolher para uma determinada consulta, se vai utilizar indexes ou não por exemplo

Isto pode fazer uma enorme diferença em um ambiente de produção.

Com esta pequena demonstração vimos como influenciar o otimizador sem alterar uma linha de código.

