**统计信息**

云和恩墨(北京)信息技术有限公司

技术顾问 燕鑫

http://www.enmotech.com

**文档控制：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序** | **版本号** | **更改人** | **日期** | **备注** |
| 1 | 1.0版 | 燕鑫 | 2018-06-25 | 初始版本 |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 统计信息的概述 - 4 -](#_Toc517872104)

[2. 统计信息的收集 - 4 -](#_Toc517872105)

[2.1 ANALYZE - 4 -](#_Toc517872106)

[2.1.1 各类指令演示 - 4 -](#_Toc517872107)

[2.1.2 ANALYZE指令小结 - 10 -](#_Toc517872108)

[3. 统计信息的分类 - 11 -](#_Toc517872109)

[3.1 表的统计信息 - 11 -](#_Toc517872110)

# 统计信息的概述

Oracle数据库里的统计信息：

1. 存储方式：存储在数据字典中；
2. 存储内容：多维度描述数据库里对象的详细信息
3. 目的：让CBO自己生成正确的执行计划

ORACLE数据库里的统计信息可分为如下6类：

1. 表的统计信息
2. 索引的统计信息
3. 列的统计信息
4. 系统统计信息（CPU cost）
5. 数据字典统计信息（基表：TAB$等）
6. 内部对象统计信息（X$）

# 统计信息的收集

统计信息的收集方式有俩种，一种是ANALYZE，一种是DBMS\_STATS包。这里我们主要介绍ANALYZE的一些用法，因为这个不常用，后面除了一些特殊情况就不多提了。

在后面的章节中我们主要是使用DBMS\_STATS包进行实验和介绍。

## ANALYZE

### 常用指令演示

首先我们先建立实验表T22:

SQL> create table t22 as select \* from dba\_objects;

Table created.

SQL> create index idx\_t22 on t22(object\_id);

Index created.

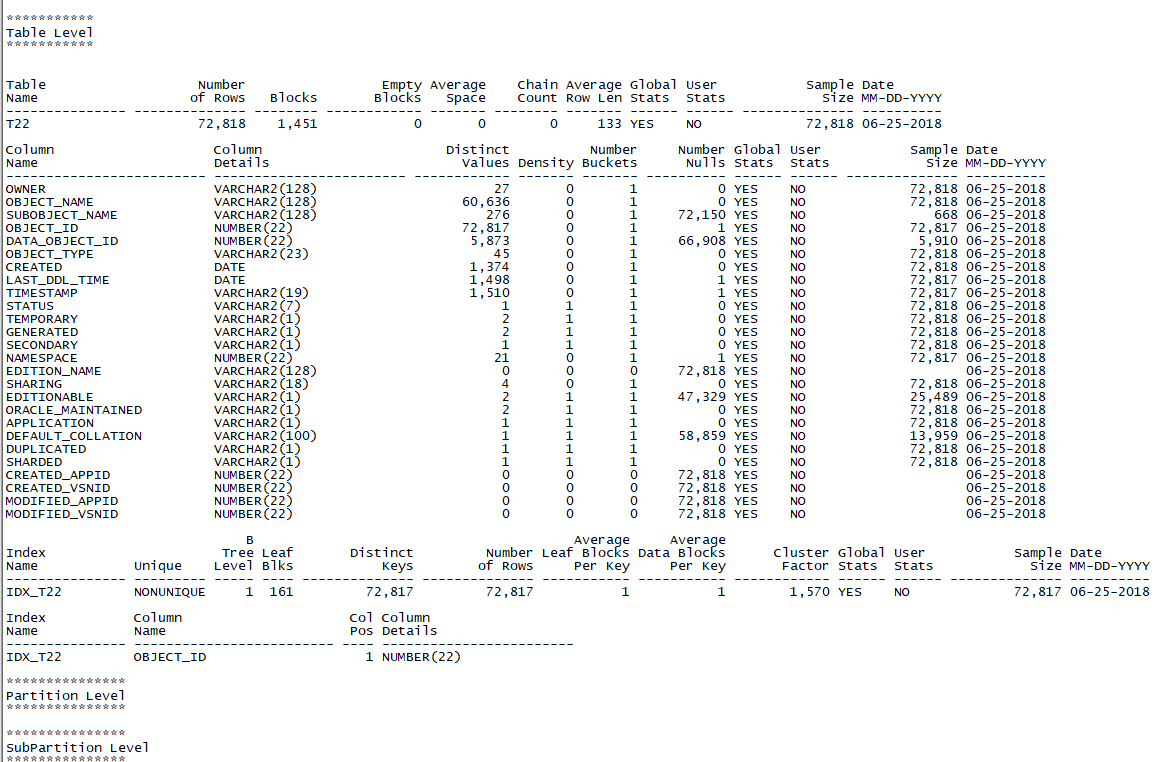
SQL> select count(1) from t22;

COUNT(1)

----------

72818

然后我们用崔老师的脚本跑一下看看：



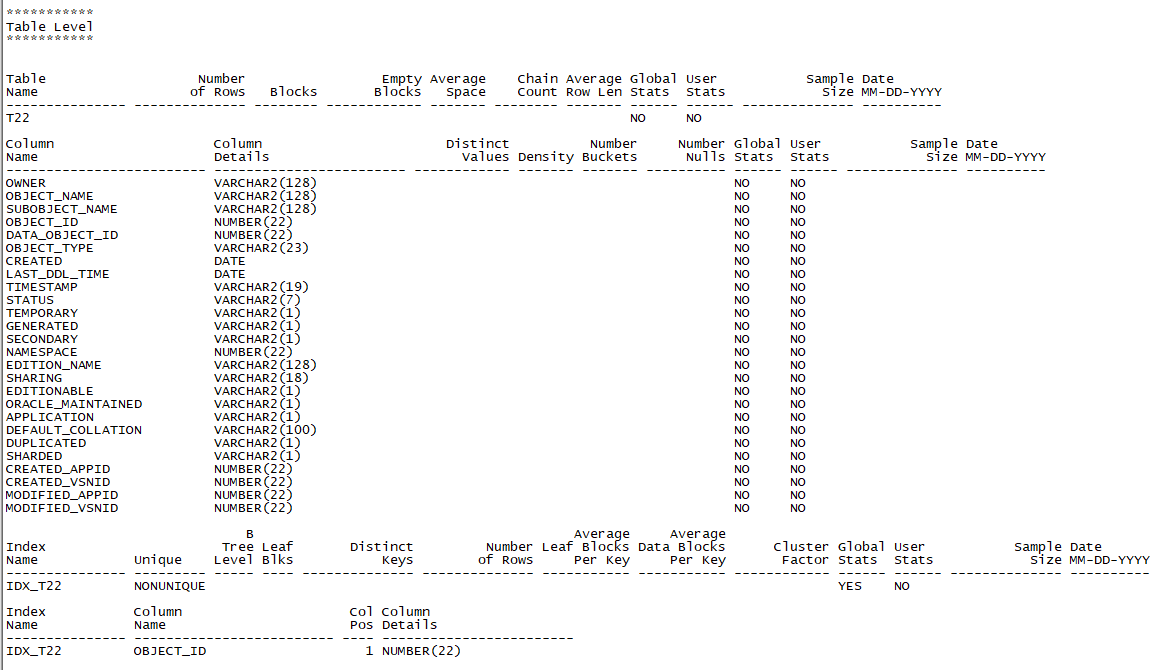
发现统计信息已经有了，11g我不确定，12.2是用create table … as select… 建的表ORACLE就会自动收集统计信息（就算源表并没有统计信息），当然建索引ORACLE会自动收集索引的统计信息，这之前就有。

那我们把统计信息先删掉：

SQL> analyze table t22 delete statistics;

Table analyzed.

再看一下统计信息：



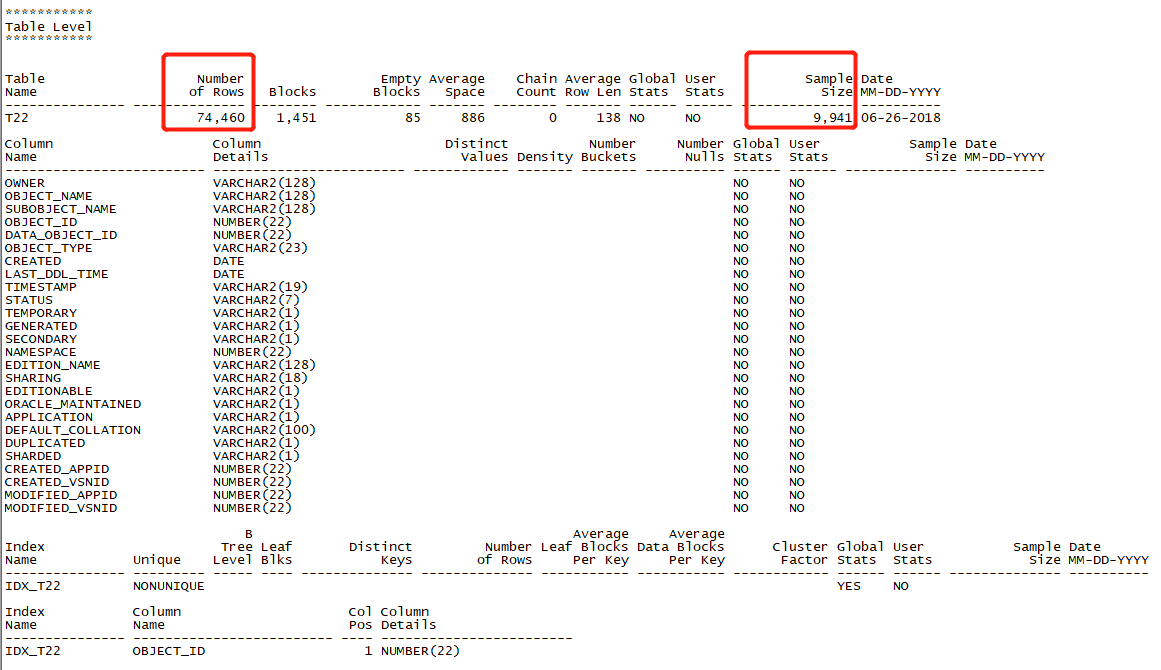
现在看到统计信息被清空了。

然后我们来收集表的统计信息，收集分俩种模式：计算模式和估算模式。

估算模式：analyze table t22 estimate statistics sample 15 percent for table;

计算模式：analyze table t22 compute statistics for table;

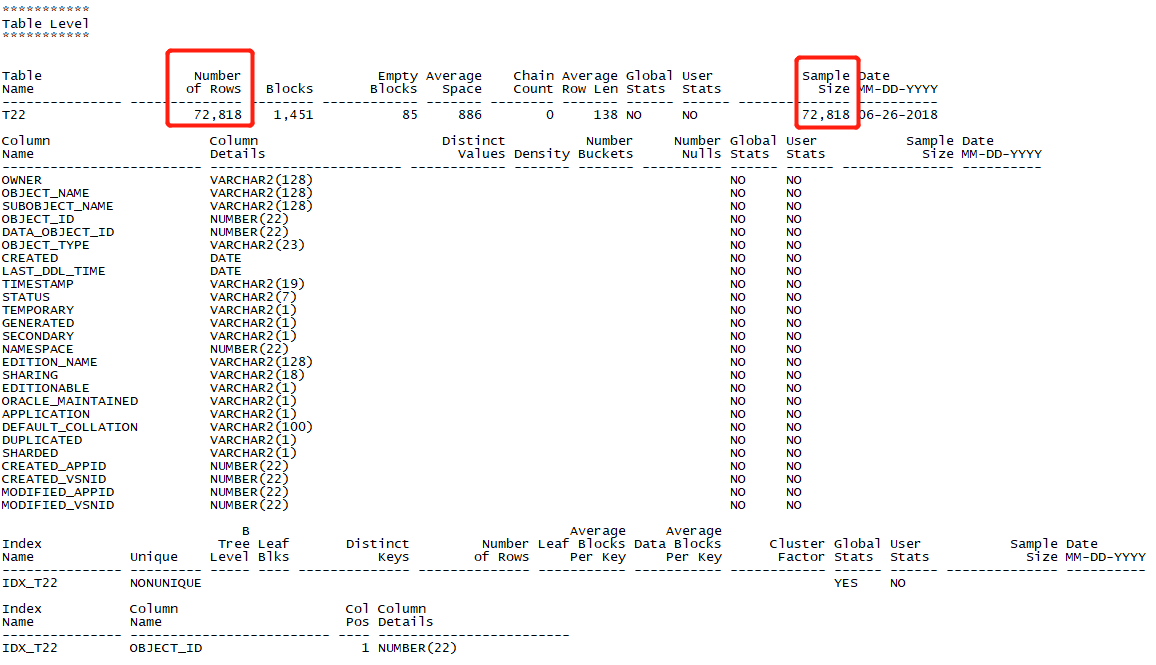
我们分别来执行看下效果，估算模式下：



这里我们可以看到，首先行数并不准确，表的真实行数是72818。Sample size的值接近72818\*0.15。

同时，可以看到这里只收集了表的统计信息，没有列和索引的统计信息。这主要取决于命令最后的for后的内容。

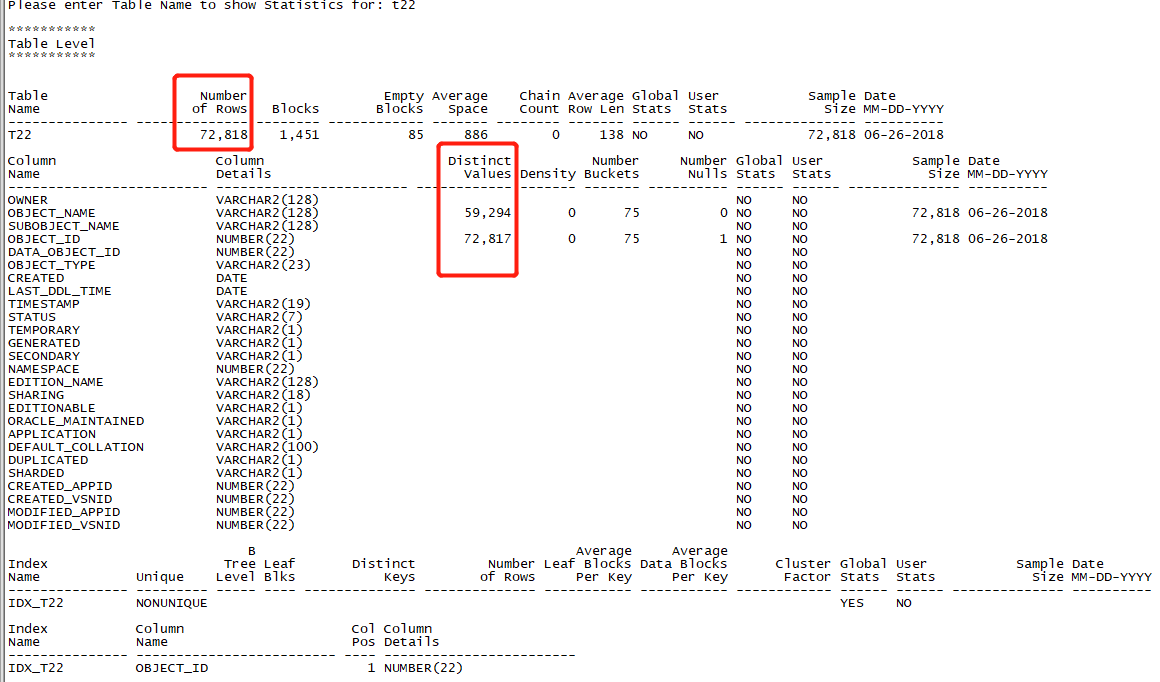
再来看下计算模式：



这次我们看到，行数是准群的，sample size也是72818。说明计算模式其实就是百分百采样。

我们再来用计算模式，只对t22表的列object\_id,object\_name收集下统计信息：

analyze table t22 compute statistics for columns object\_name,object\_id;



我们这里需要注意俩个事情：

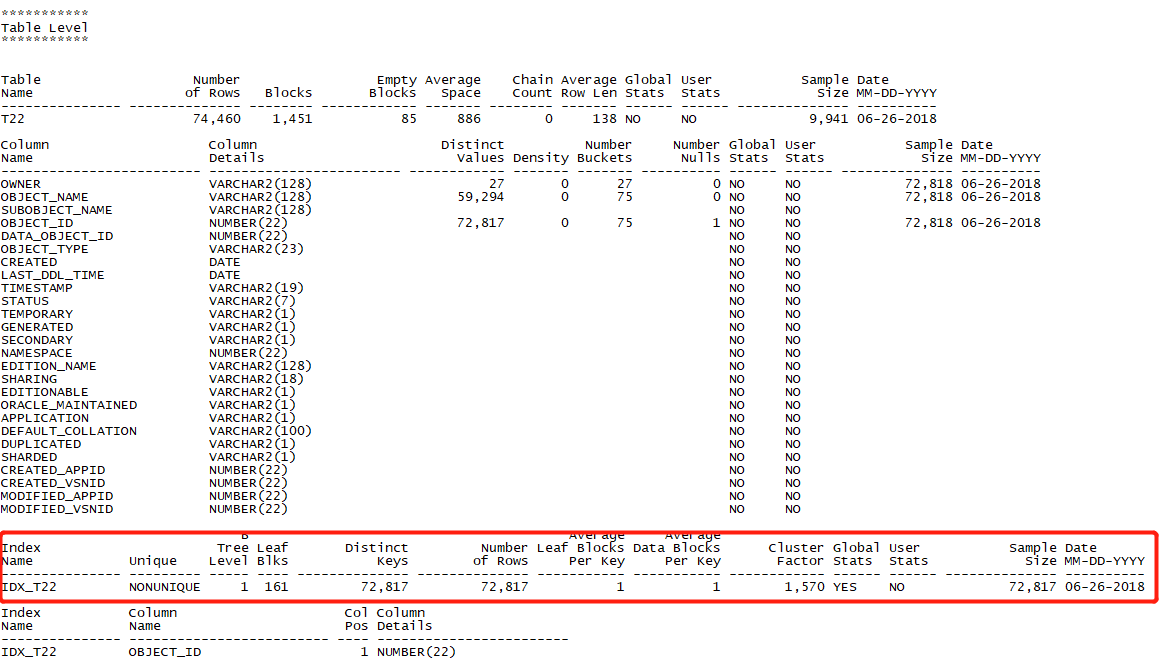
1. 依然是因为for columns后面只有这俩列，所以只收集了这俩列的统计信息；
2. 而同时，崔老师的实验表的统计信息会被清除，因为这里我们只要求收集列的统计信息。这是analyze之前的一个特性，但是显然在12.2中，已经没有这种情况了，即对同一个对象重复analyze，只会覆盖之前相应的记录，而不会清除其他已存在的记录。（这一点我已反复做过实验，这里不深究了）

然后我们又要对表同时也要对列进行统计信息收集：

analyze table t22 compute statistics for table for columns object\_name,object\_id;

上面说的都是analyze table，现在我们说收集索引：

analyze index idx\_t22 compute statistics;

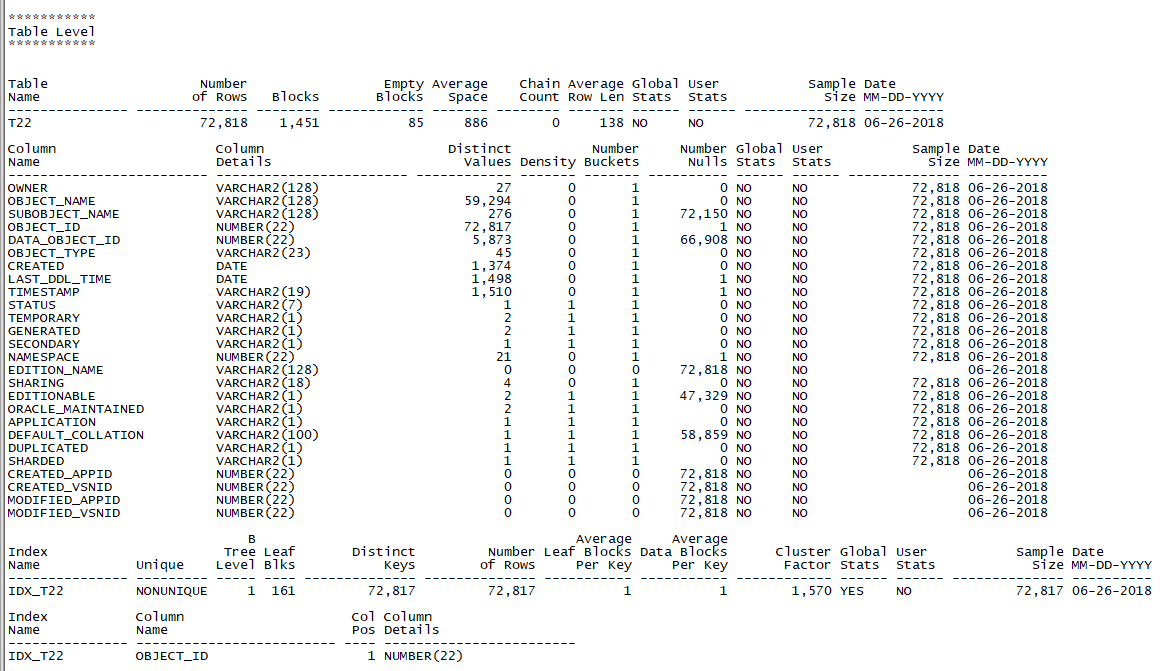


可以看到索引的统计信息已经有了。

那如果我们想一次性把表的表、列、索引的统计信息收集，则执行如下指令：

analyze table t22 compute statistics;

说白了，就是后面不跟for子句。



现在可以看到，啥的统计信息都有了。

### ANALYZE指令小结

删除统计信息：

analyze table t22 delete statistics;

估算模式：

analyze table t22 estimate statistics sample 15 percent for table;

计算模式对表对列收集：

analyze table t22 compute statistics for table;

analyze table t22 compute statistics for columns object\_name,object\_id;

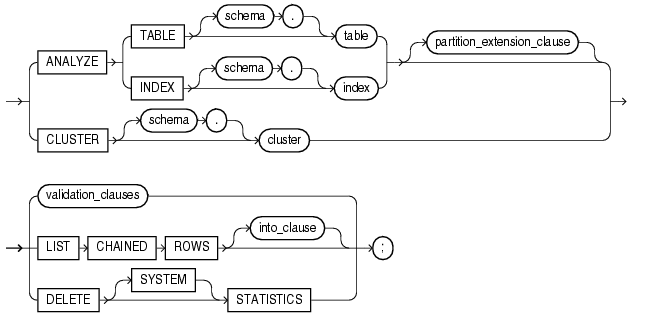
analyze table t22 compute statistics for table for columns object\_name,object\_id;

计算模式对索引收集：

analyze index idx\_t22 compute statistics;

计算模式全收集：

analyze table t22 compute statistics;



## DBMS\_STATS

### 常用指令列举

这里只列举dbms\_stats包里最常用的7个存储过程：

1. GATHER\_TABLE\_STATS
2. GATHER\_INDEX\_STATS
3. GATHER\_SCHEMA\_STATS
4. GATHER\_DATABASE\_STATS
5. GATHER\_SYSTEM\_STATS：系统统计信息
6. GATHER\_DICTIONARY\_STATS：数据字典的统计信息
7. GATHER\_FIXED\_OBJECTS\_STATS：内部对象X$表的统计信息
8. DELETE\_\*\_STATS：删除各种统计信息

### DBMS\_STATS指令小结

Dbms\_stats包的常量（可修改，下面列出的是oracle自己的默认值）：

1. AUTO\_CASCADE：自动级联收集索引统计信息，默认值为null
2. AUTO\_DEGREE：收集统计信息的并行数，默认值为32768
3. AUTO\_INVALIDATE：收集统计信息后是否让相关游标失效，默认值为null
4. AUTO\_SAMPLE\_SIZE：采样比，默认值为0

常用的一些属性，以GATHER\_TABLE\_STATS为例：

DBMS\_STATS.GATHER\_TABLE\_STATS (

ownname VARCHAR2,

tabname VARCHAR2,

partname VARCHAR2 DEFAULT NULL,

estimate\_percent NUMBER DEFAULT to\_estimate\_percent\_type

(get\_param('ESTIMATE\_PERCENT')),

block\_sample BOOLEAN DEFAULT FALSE,

method\_opt VARCHAR2 DEFAULT get\_param('METHOD\_OPT'),

degree NUMBER DEFAULT to\_degree\_type(get\_param('DEGREE')),

granularity VARCHAR2 DEFAULT GET\_PARAM('GRANULARITY'),

cascade BOOLEAN DEFAULT to\_cascade\_type(get\_param('CASCADE')),

stattab VARCHAR2 DEFAULT NULL,

statid VARCHAR2 DEFAULT NULL,

statown VARCHAR2 DEFAULT NULL,

no\_invalidate BOOLEAN DEFAULT to\_no\_invalidate\_type (

get\_param('NO\_INVALIDATE')),

stattype VARCHAR2 DEFAULT 'DATA',

force BOOLEAN DEFAULT FALSE,

context DBMS\_STATS.CCONTEXT DEFAULT NULL, -- non operative

options VARCHAR2 DEFAULT 'GATHER');

这里介绍一些常用的属性：

1. owname
2. tabname
3. partname
4. estimate\_percent：采样比，该值为null就是计算模式，默认值是auto\_sample\_size
5. method\_opt：默认值是FOR ALL COLUMNS SIZE AUTO  
   （1）FOR ALL [INDEXED | HIDDEN] COLUMNS [size\_clause]  
   （2）FOR COLUMNS [column\_clause] [size\_clause]  
   size\_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}  
   column\_clause := column\_name | extension name | extension  
   然后来说明一下这个重要的属性:  
   第一，对于database,shema,dictionary只能使用for all。  
   第二，size子句是用来收集直方图用的，integer是1就不收集直方图，大于1则是收集直方图；repeat是只对以有直方图的列收集直方图；auto是oracle自己决定是否收集直方图；skewonly是列倾斜的才收集直方图。  
   第三：extension就是可以是多列(column\_name, Colume\_name [, ...])，也可以是个表达式。
6. degree：收集统计信息时使用的并行度，默认值null，即不开并行
7. granularity：粒度，这是收集全局统计信息时用到的，后面详述，有如下几个值：  
   'ALL '，'APPROX\_GLOBAL AND PARTITION '，'AUTO '，  
   'GLOBAL '，'GLOBAL AND PARTITION '，'PARTITION '，'SUBPARTITION'
8. cascade：是否级联收集索引的统计信息，true就是级联收集
9. no\_invalidate：是否立即将相关游标失效，ture是不失效，false是立马失效。

下面我们查看一下上述的一些默认值：

SQL> select dbms\_stats.get\_param('CASCADE') from dual;

DBMS\_STATS.GET\_PARAM('CASCADE')

--------------------------------------------------------------------------------

DBMS\_STATS.AUTO\_CASCADE

SQL> select dbms\_stats.get\_param('DEGREE') from dual;

DBMS\_STATS.GET\_PARAM('DEGREE')

--------------------------------------------------------------------------------

NULL

SQL> select dbms\_stats.get\_param('NO\_INVALIDATE') from dual;

DBMS\_STATS.GET\_PARAM('NO\_INVALIDATE')

--------------------------------------------------------------------------------

DBMS\_STATS.AUTO\_INVALIDATE

SQL> select dbms\_stats.get\_param('ESTIMATE\_PERCENT') from dual;

DBMS\_STATS.GET\_PARAM('ESTIMATE\_PERCENT')

--------------------------------------------------------------------------------

DBMS\_STATS.AUTO\_SAMPLE\_SIZE

SQL> select dbms\_stats.get\_param('METHOD\_OPT') from dual;

DBMS\_STATS.GET\_PARAM('METHOD\_OPT')

--------------------------------------------------------------------------------

FOR ALL COLUMNS SIZE AUTO

SQL> select dbms\_stats.GET\_PARAM('GRANULARITY') from dual;

DBMS\_STATS.GET\_PARAM('GRANULARITY')

--------------------------------------------------------------------------------

AUTO

## ANALYZE和DBMS\_STATS的区别

ORACLE推荐是用dbms\_stats，是因为analyze主要有下面俩点缺陷：

1. 收集分区表的统计信息是只收集分区的（如果有子分区就只收集子分区的）统计信息，然后推导出其余的统计信息，那比如正张表的唯一值啊这些就肯定不准了，dbms\_stats就不会这样。
2. Analyze不能并行收集统计信息，而dbms\_stats可以（degree属性）。

但是与cbo无关的统计信息，dbms\_stats就收集不了了，analyze能收集行迁移的数量、分析索引结构等。

# 统计信息的分类

## 表的统计信息

### 概要

表的统计信息存储在数据字典基表：TAB$、TABPART$、TABSUBPART$。我们可以通过视图DBA\_TABLES、DBA\_TAB\_PARTITIONS、DBA\_TAB\_SUBPARTITIONS查看到。

下面我们把sosi脚本里的这部分代码贴出来看下：

select

TABLE\_NAME,

NUM\_ROWS,

BLOCKS,

EMPTY\_BLOCKS,

AVG\_SPACE,

CHAIN\_CNT,

AVG\_ROW\_LEN,

GLOBAL\_STATS,

USER\_STATS,

SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from dba\_tables t

where owner = upper(nvl('&&Owner',user))

and table\_name = upper('&&Table\_name')

然后我们说下这些列的含义：

TABLE\_NAME：表名

NUM\_ROWS：行数

BLOCKS：已使用过的数据块个数

EMPTY\_BLOCKS：未使用过的数据块个数

AVG\_ROW\_LEN：平均行长度

GLOBAL\_STATS：

1. 这个值是yes，表示收集的是全局的统计信息；
2. 是no，表示表的统计信息是从分区子分区推算出来的。

USER\_STATS：是否未用户自己设置的统计信息（dbms\_stats.set\_table\_stats）

SAMPLE\_SIZE：采样集合大小

LAST\_ANALYZED：统计信息收集的时间

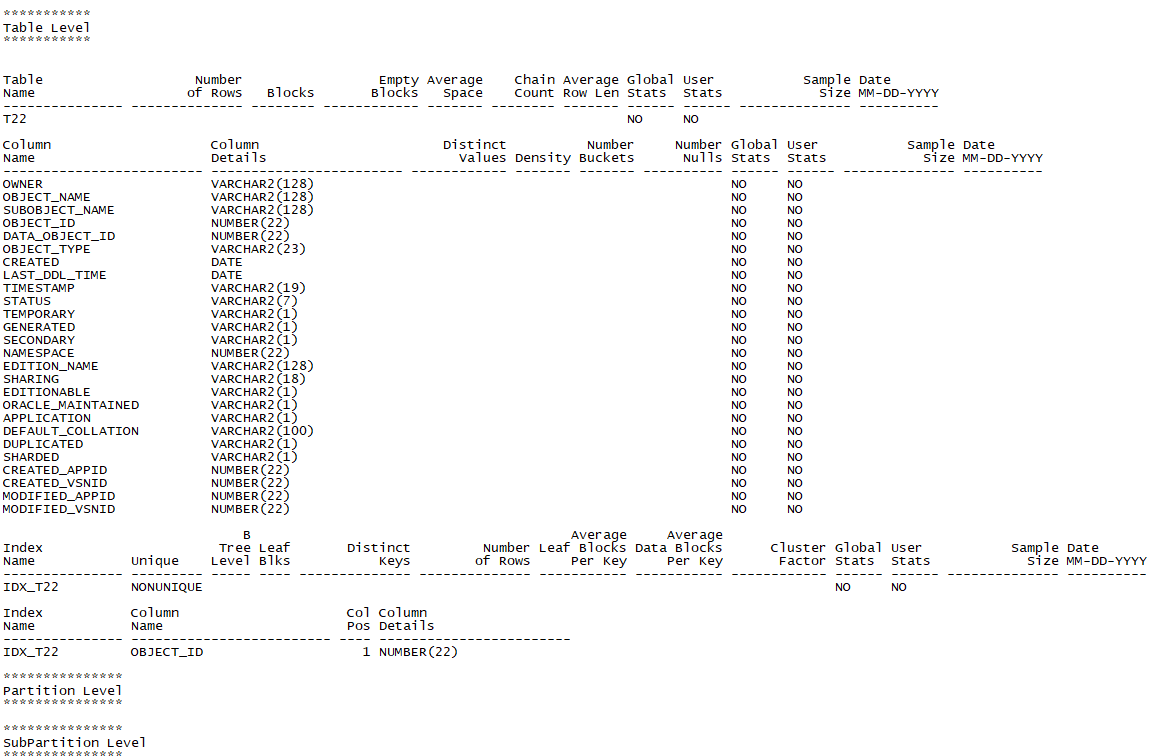
对于分区表的统计信息会在接下来的全局统计信息中总结。

### 收集方法

那么收集表的统计信息，前面介绍的是analyze的方法，这里举例使用dbms\_stats的方法。我们还是用之前的实验表：

先删除之前的统计信息：

SQL> exec dbms\_stats.delete\_table\_stats(OWNNAME=>'SCOTT',TABNAME=>'T22');



可以看到，这样的删除方式默认是级联的，这里还是贴一下dbms\_stats.delete\_table\_stats：

DBMS\_STATS.DELETE\_TABLE\_STATS (

ownname VARCHAR2,

tabname VARCHAR2,

partname VARCHAR2 DEFAULT NULL,

stattab VARCHAR2 DEFAULT NULL,

statid VARCHAR2 DEFAULT NULL,

cascade\_parts BOOLEAN DEFAULT TRUE,

cascade\_columns BOOLEAN DEFAULT TRUE,

cascade\_indexes BOOLEAN DEFAULT TRUE,

statown VARCHAR2 DEFAULT NULL,

no\_invalidate BOOLEAN DEFAULT to\_no\_invalidate\_type (

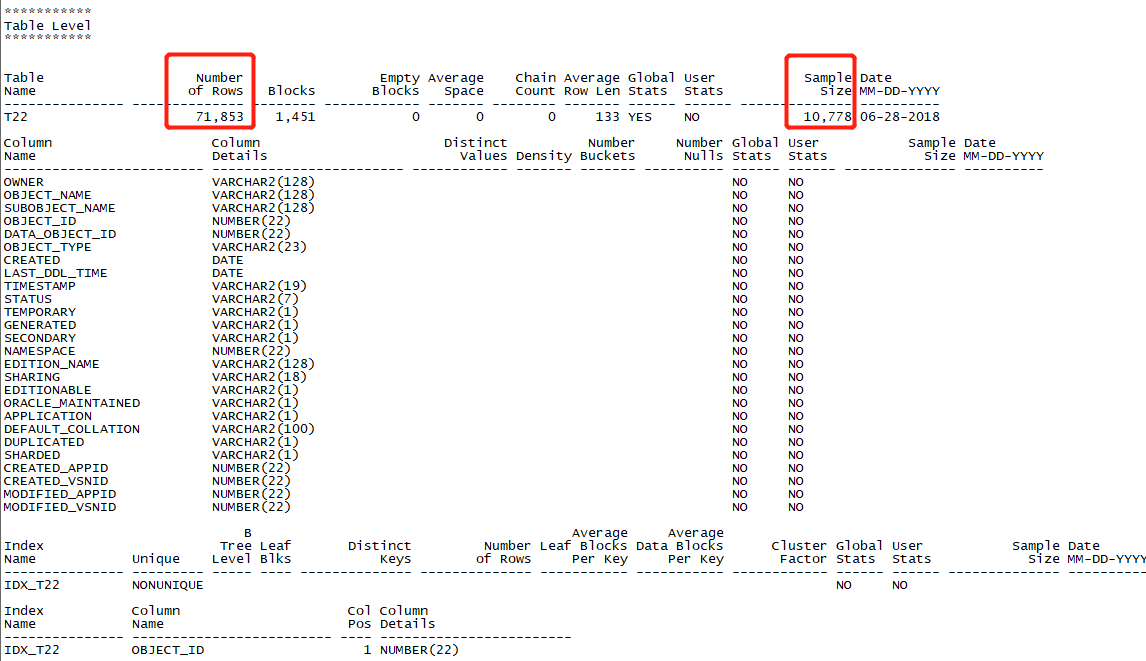
get\_param('NO\_INVALIDATE')),

force BOOLEAN DEFAULT FALSE);

这样就一目了然，为什么会连同列和索引的统计信息都删除了。

接着我们收集统计信息：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22',estimate\_percent => 15,cascade => false,method\_opt => 'FOR TABLE');



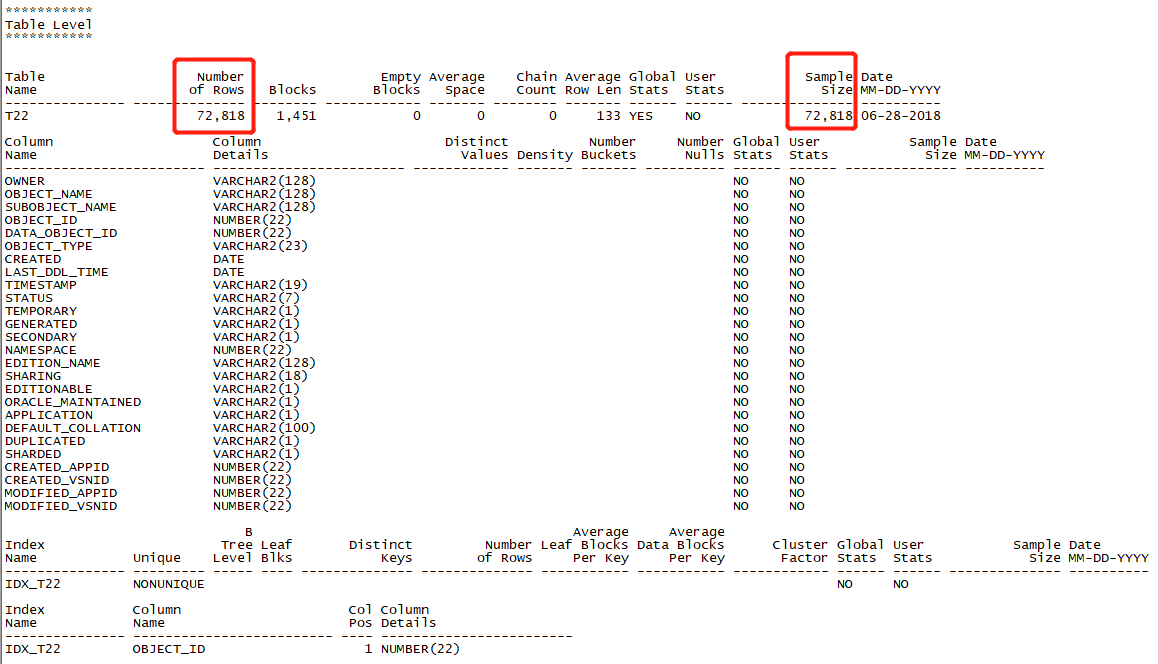
这里我们注意，sample\_size/0.15正是NUM\_ROWS，所以dbms\_stats的估算模式跟analyze还是有区别的。

我们看到我们的method\_opt是FOR TABLE，就是只收集表的统计信息。

当estimate\_percent的值为null或100时，就是计算模式，即百分之百采样：

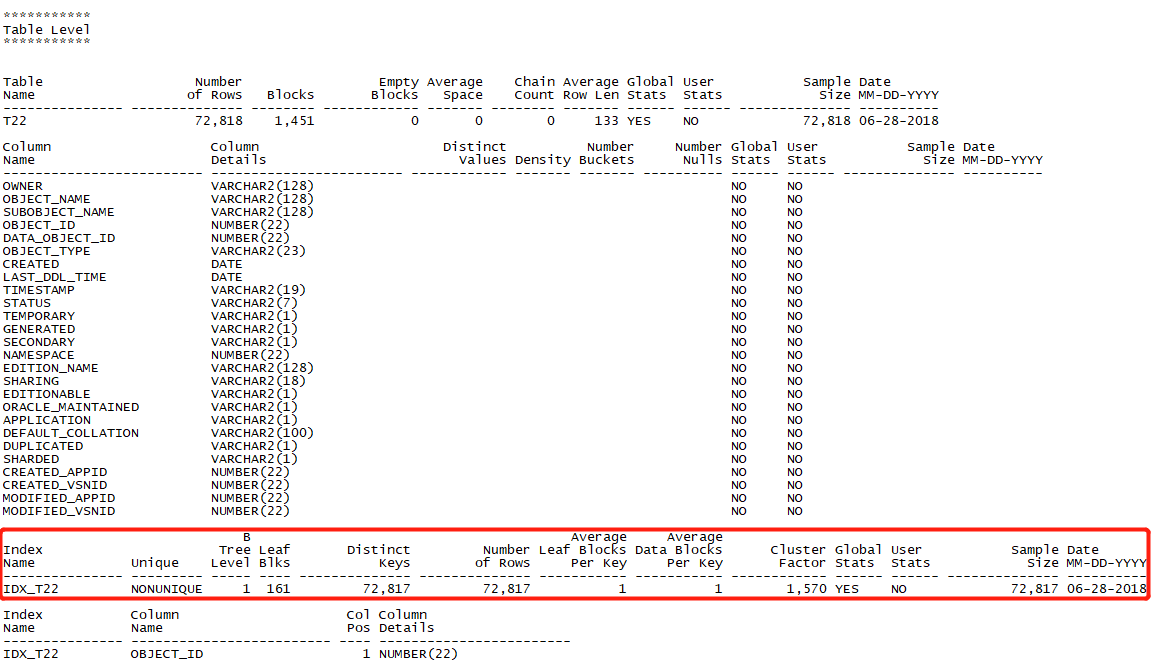
SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22',estimate\_percent => null,cascade => false,method\_opt => 'FOR TABLE');

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22',estimate\_percent => 100,cascade => false,method\_opt => 'FOR TABLE');



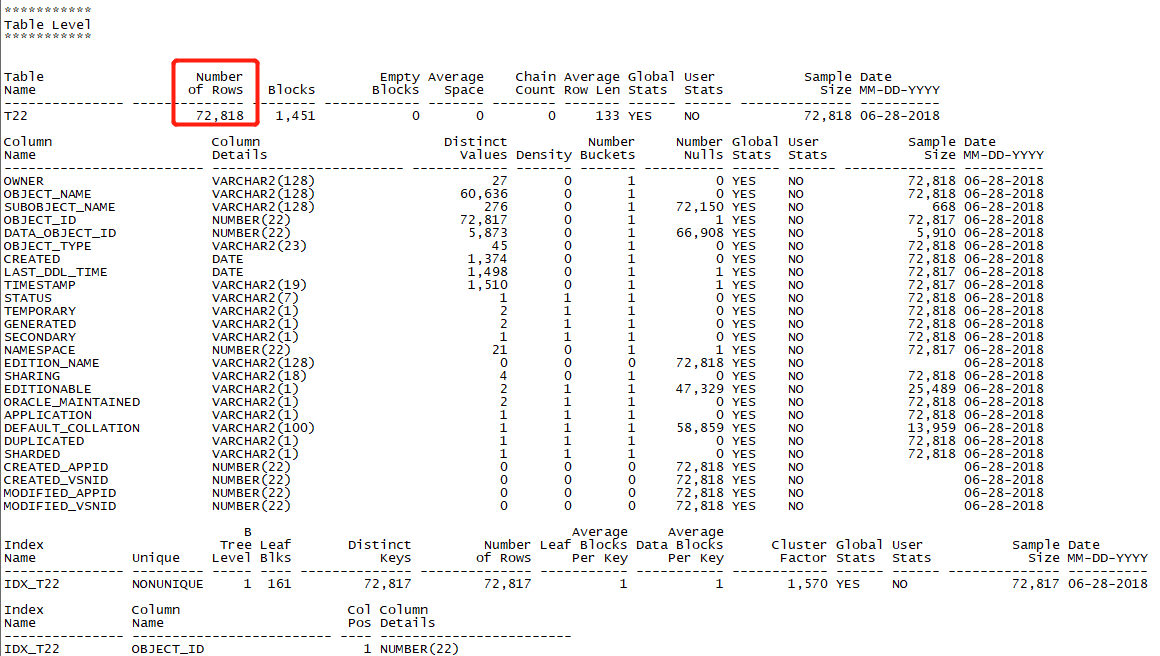
现在我们把cascade置为true，即级联收集索引的统计信息：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22',estimate\_percent => 100,cascade => true,method\_opt => 'FOR TABLE');



现在我们只指定用户名和表名：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22');



这里我们看到：

1. ORACLE自动收集列及索引的统计信息
2. Oracle对于T22表默认选择了百分之百的采样率

ORACLE默认选择的采样率，即DBMS\_STATS.AUTO\_SAMPLE\_SIZE会自行判断合适的采样比，而且即使使用百分之百的采样率，由于11g以上的新版本对于采样做了进一步优化，也不会像之前那么慢（这块知识的具体内容我是真的不会）。总之，在11g以后进行表的统计信息的收集时，完全可以直接使用estimate\_percent的默认值，即不用特别指定。

## 全局统计信息

### 概要

ORACLE的全局统计信息（global statistics）就是指直接从对象本身这一级收集到的统计信息，而不是从对象下一级（underlying)推导或汇总出来的统计信息。这一点只有dbms\_stats能做到，analyze做不到，前面也说了的。

那也就是说，全局统计信息这一说法只对分区表才有意义。

脚本部分的分区及子分区部分如下：

分区级：

select

PARTITION\_NAME,

NUM\_ROWS,

BLOCKS,

EMPTY\_BLOCKS,

AVG\_SPACE,

CHAIN\_CNT,

AVG\_ROW\_LEN,

GLOBAL\_STATS,

USER\_STATS,

SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from

dba\_tab\_partitions t

where

table\_owner = upper(nvl('&&Owner',user))

and table\_name = upper('&&Table\_name')

order by partition\_position

/

子分区级：

select

PARTITION\_NAME,

SUBPARTITION\_NAME,

NUM\_ROWS,

BLOCKS,

EMPTY\_BLOCKS,

AVG\_SPACE,

CHAIN\_CNT,

AVG\_ROW\_LEN,

GLOBAL\_STATS,

USER\_STATS,

SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from

dba\_tab\_subpartitions t

where

table\_owner = upper(nvl('&&Owner',user))

and table\_name = upper('&&Table\_name')

order by SUBPARTITION\_POSITION

/

这些列在表的统计信息介绍中说明过了，这里就不赘述了。

### 收集方法

全局统计信息只能用dbms\_stats来收集，只需要对一个参数granularity指定值即可。

Granularity有如下几个常用值：

1. GLOBAL（表）
2. PARTITION（分区）
3. SUBPARTITION（子分区）
4. GLOBAL AND PARTITION（表+分区）
5. ALL（表+分区+子分区）
6. AUTO（至少为表+分区，子分区为LIST才会+子分区）

是否是全局统计信息，前面有备注过，就是看GLOBAL\_STATS列，yes就是是全局统计信息。那这一列在如下表中有：

DBA\_TABLES、DBA\_TAB\_PARTITIONS、DBA\_TAB\_SUBPARTITIONS、  
DBA\_INDEXES、DBA\_IND\_PARTITIONS、DBA\_IND\_SUBPARTITIONS、DBA\_(TAB/PART/SUBPART)\_COL\_STATISTICS

接着我们来创建实验表：

create table t23(i number,p number,sp number)

partition by range(p)

subpartition by hash(sp) subpartitions 2

(

partition q1 values less than(3) tablespace users,

partition q2 values less than(maxvalue) tablespace users

);

declare

i number;

begin

for i in 1..100000 loop

insert into t23 values(i,mod(i,7),mod(i,8));

if( mod(i, 1000) = 0)

then commit;

end if;

end loop;

for i in 1..50000 loop

insert into t23 values(i,mod(i,7),mod(i,8)+5);

if( mod(i, 1000) = 0)

then commit;

end if;

end loop;

end;

/

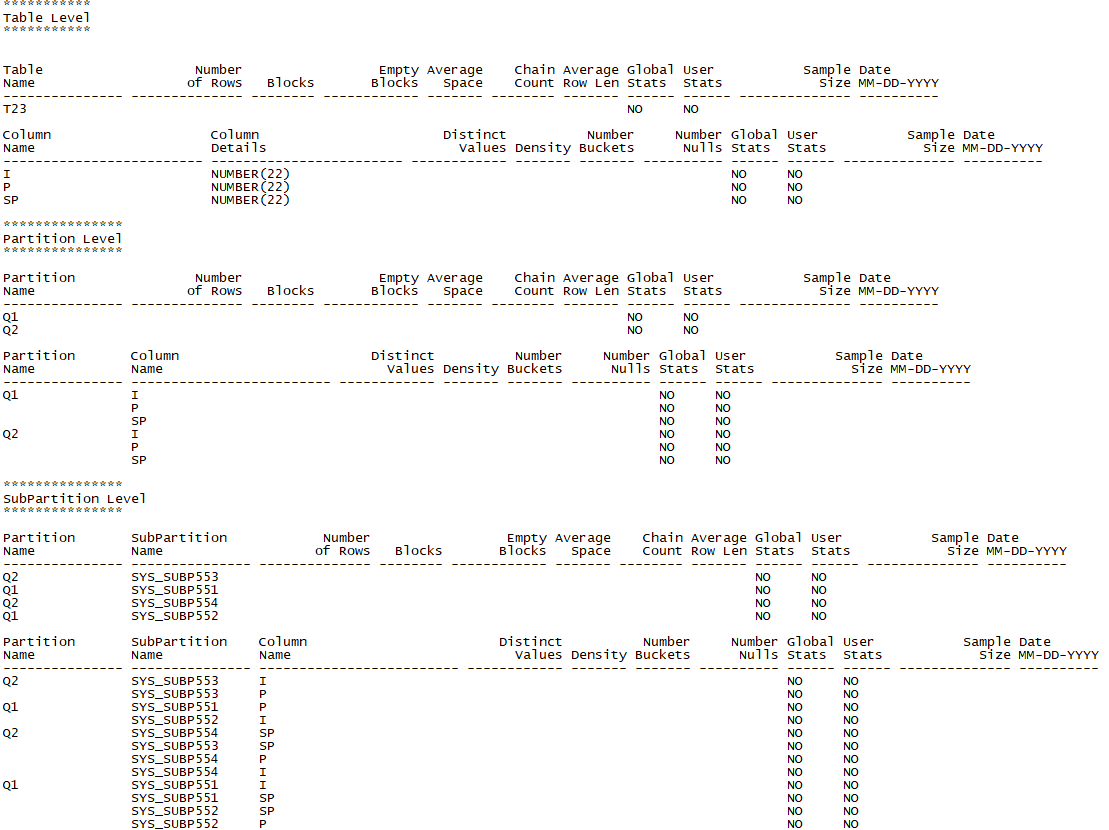
SQL> select count(\*) from t23;

COUNT(\*)

----------

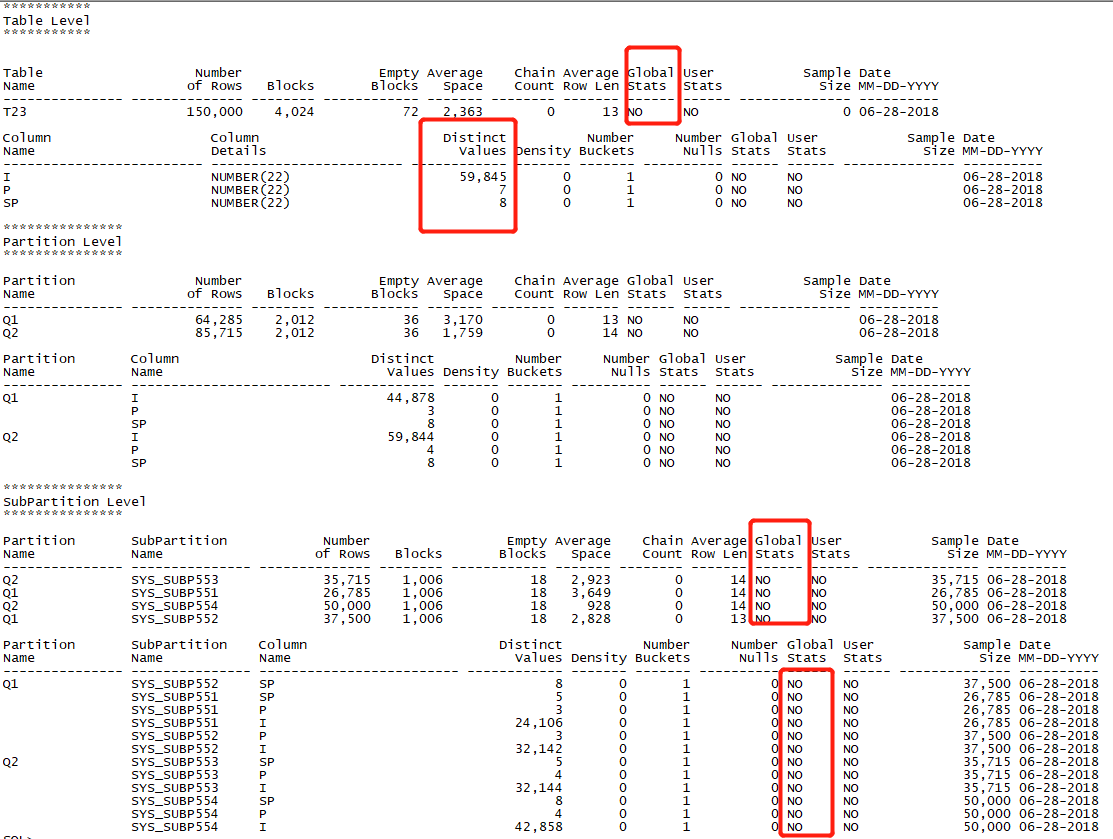
150000

现在我们来看下统计信息：



我们看到目前没有统计信息，现在我们先用analyze来收集：

SQL> analyze table t23 compute statistics;



我们看到，因为列distinct值是由子分区统计信息推导出来的，这个显然非常不准。

SQL> select count(distinct i) from t23 subpartition(SYS\_SUBP551);

COUNT(DISTINCTI)

----------------

24106

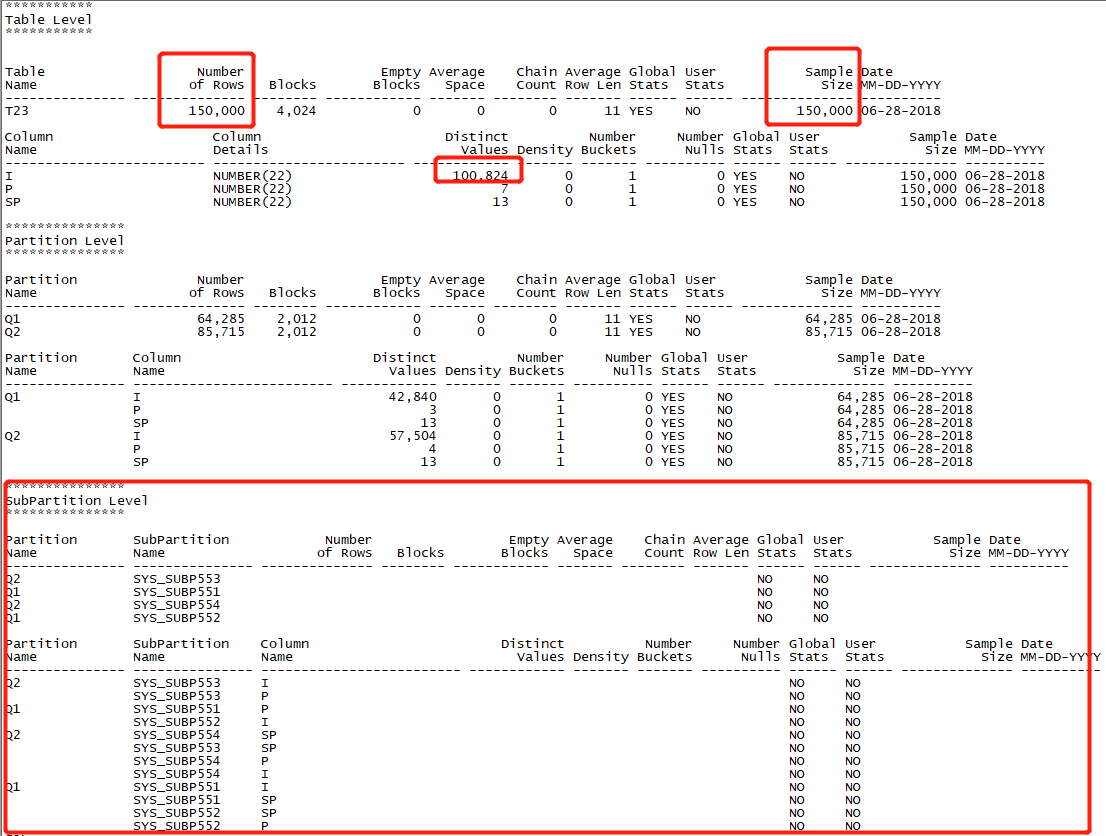
但是我们可以看到子分区的统计信息是准确的。

我们现在把统计信息删了重新来收集：

SQL> exec dbms\_stats.delete\_table\_stats('SCOTT','T23');

我们这里顺便可以看一下指定estimate\_percent=>100和让其按默认auto\_sample\_size的区别：

SQL> exec dbms\_stats.gather\_table\_stats(ownname=>'SCOTT',tabname=>'T23');



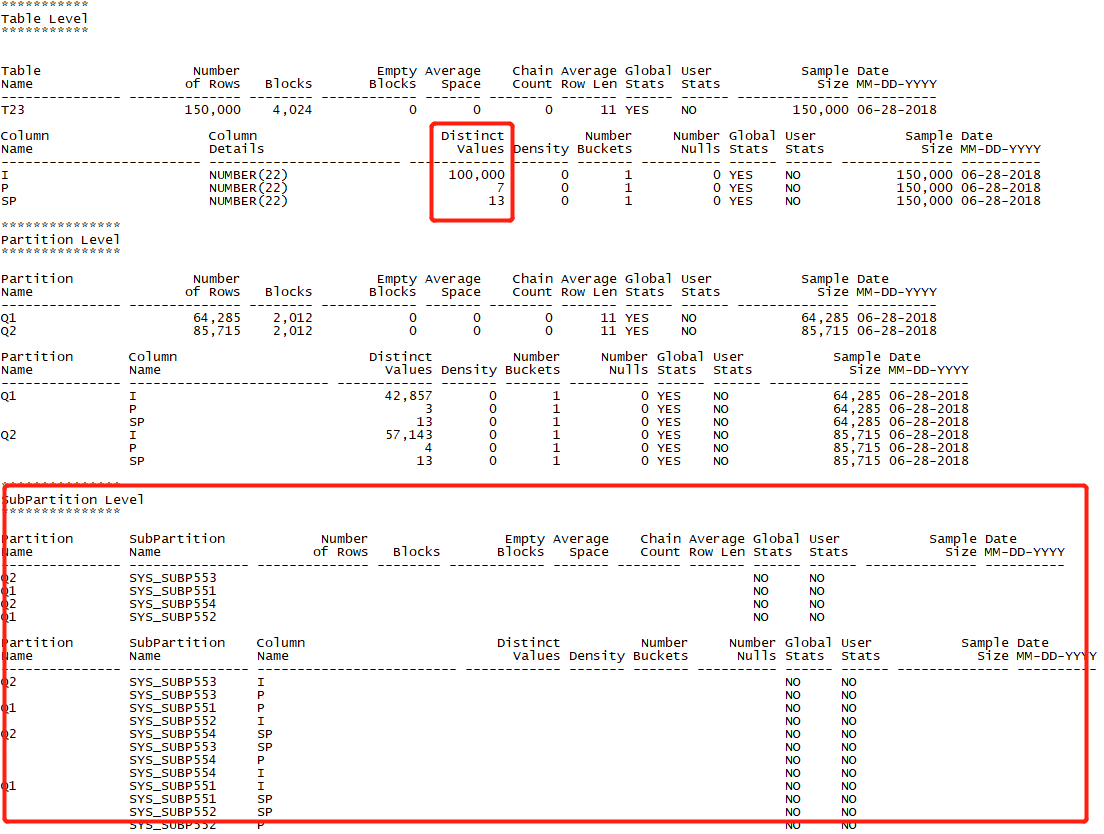
首先我们看到，因为granularity默认是auto，且子分区不是list分区，所以这里没有子分区的统计信息。

其次，可以看到global\_stats现在是yes了。

最后，我们发现看似百分之百的采样，但是列I的distinct值是不准的，很明显应该是10000才对。

下面我们指定estimate\_percent=>100：

SQL> exec dbms\_stats.gather\_table\_stats(ownname=>'SCOTT',tabname=>'T23',estimate\_percent=>100);

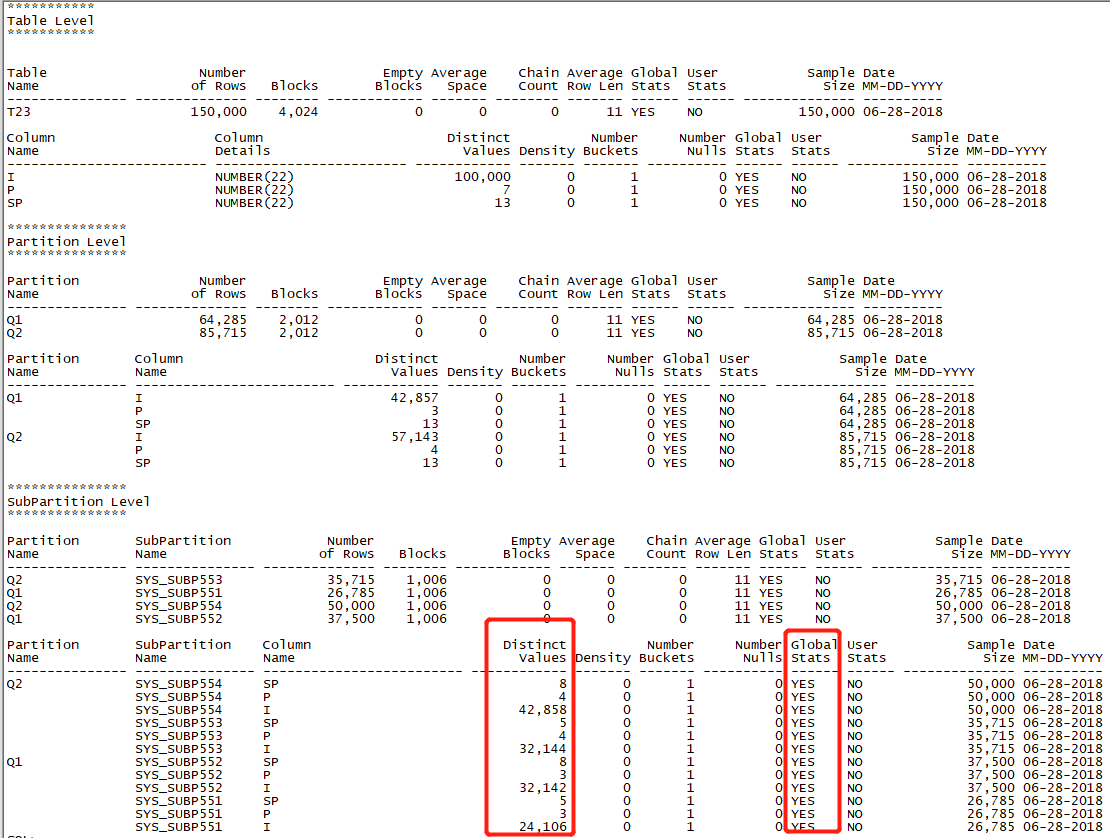


现在我们看到，这次统计信息才是绝对准确的。所以说，确实estimate\_percent为auto\_sample\_size跟100是有区别的。

这次我们指定granularity为ALL，再看下效果：

SQL> exec dbms\_stats.delete\_table\_stats('SCOTT','T23');

SQL> exec dbms\_stats.gather\_table\_stats(ownname=>'SCOTT',tabname=>'T23',estimate\_percent=>100, granularity=>'ALL');



现在我们确实看到，子分区的统计信息也被收集了，而且global\_stats也变为了yes。

### 关于global\_stats的俩个实验

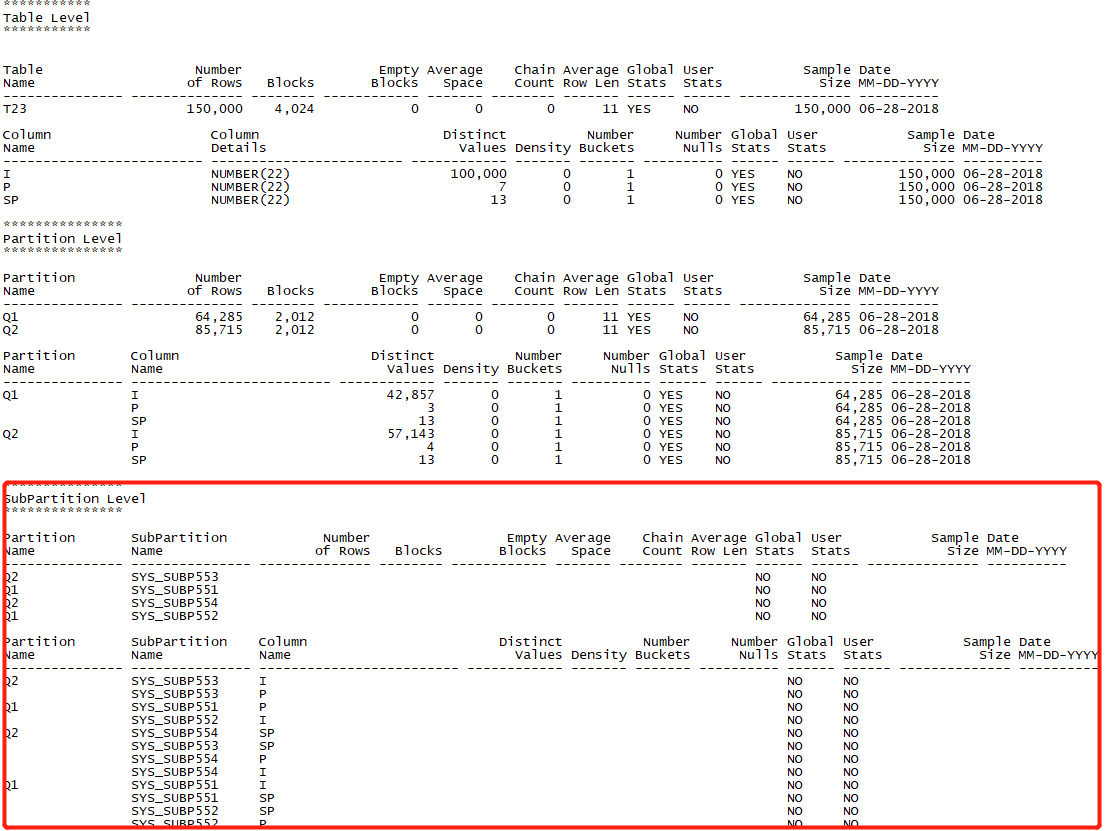
#### 只收集分区统计信息

我们看到，每次我们收集全局统计信息之前都要先进行统计信息的删除。为什么要这样做呢？我们做2个实验：

第一个，我们把统计信息先删除，然后指定granularity为partition。

这里我们先使用analyze删除：

SQL> analyze table t23 delete statistics;

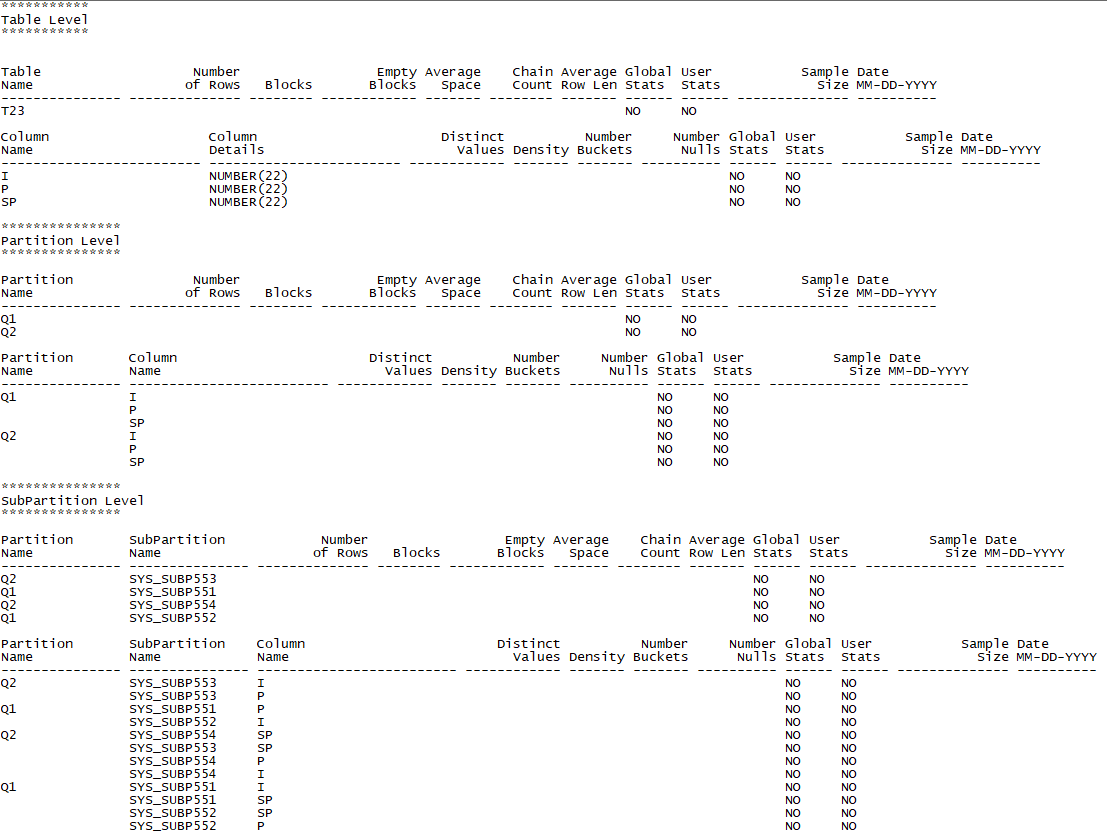


可以看到，只有子分区的统计信息被删除了。这里先说下原因，也不再做过多验证：原因就是子分区的上一级的统计信息，即分区的统计信息global\_stats是yes，就不能级联删除了。

所以对于分区表的统计信息的收集或是删除都不要用ANALYZE。

接着我们用dbms\_stats来删：

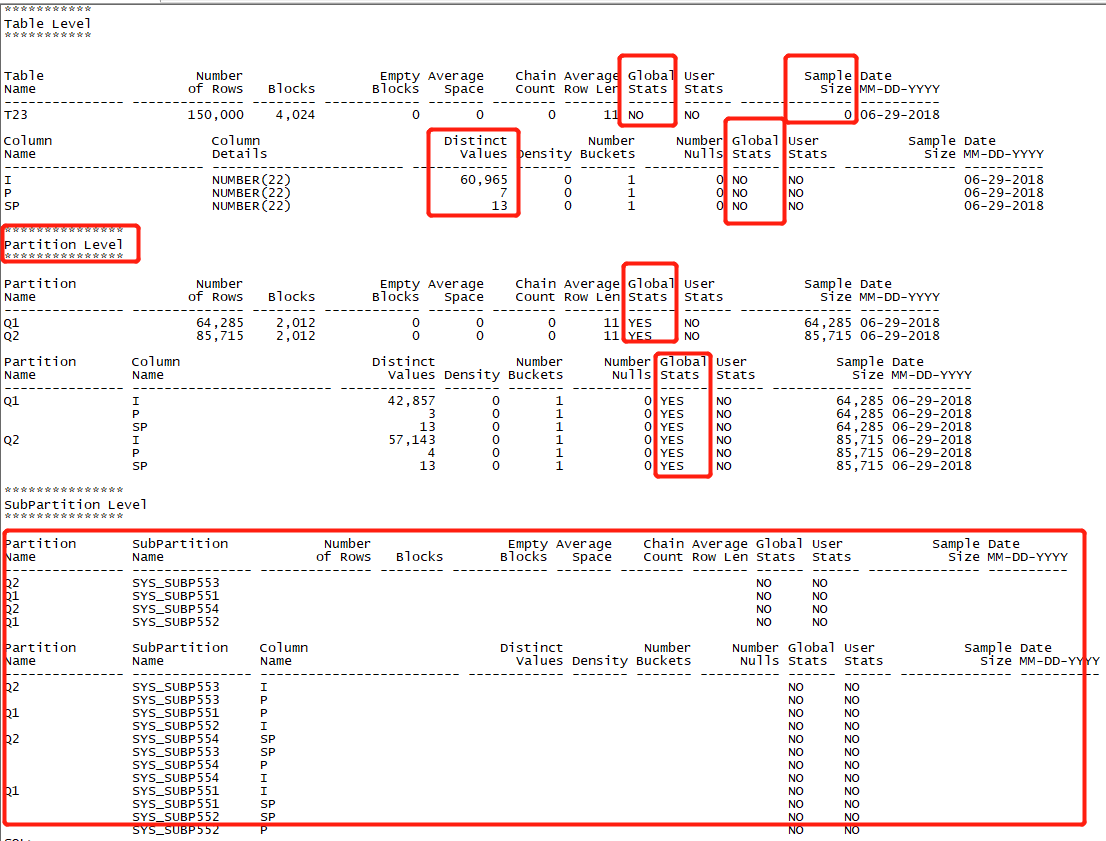
SQL> exec dbms\_stats.delete\_table\_stats('SCOTT','T23');



可以看到这才能删干净。

现在我们按计划，指定granularity为PARTITION：

SQL> exec dbms\_stats.gather\_table\_stats(ownname=>'SCOTT',tabname=>'T23',estimate\_percent=>100, granularity=>'PARTITION');



我们可以看到有如下俩点：

1. 分区的上一级表级的统计信息被收集了，但是表级的global\_stats是NO，也就是说表级的统计信息是由分区的统计信息推导出来的，且我们注意到sample\_size是0。
2. 子分区的统计信息并没有被收集。

这里第一个实验就做完了。

第一个实验说明dbms\_stats只收集分区统计信息的时候：

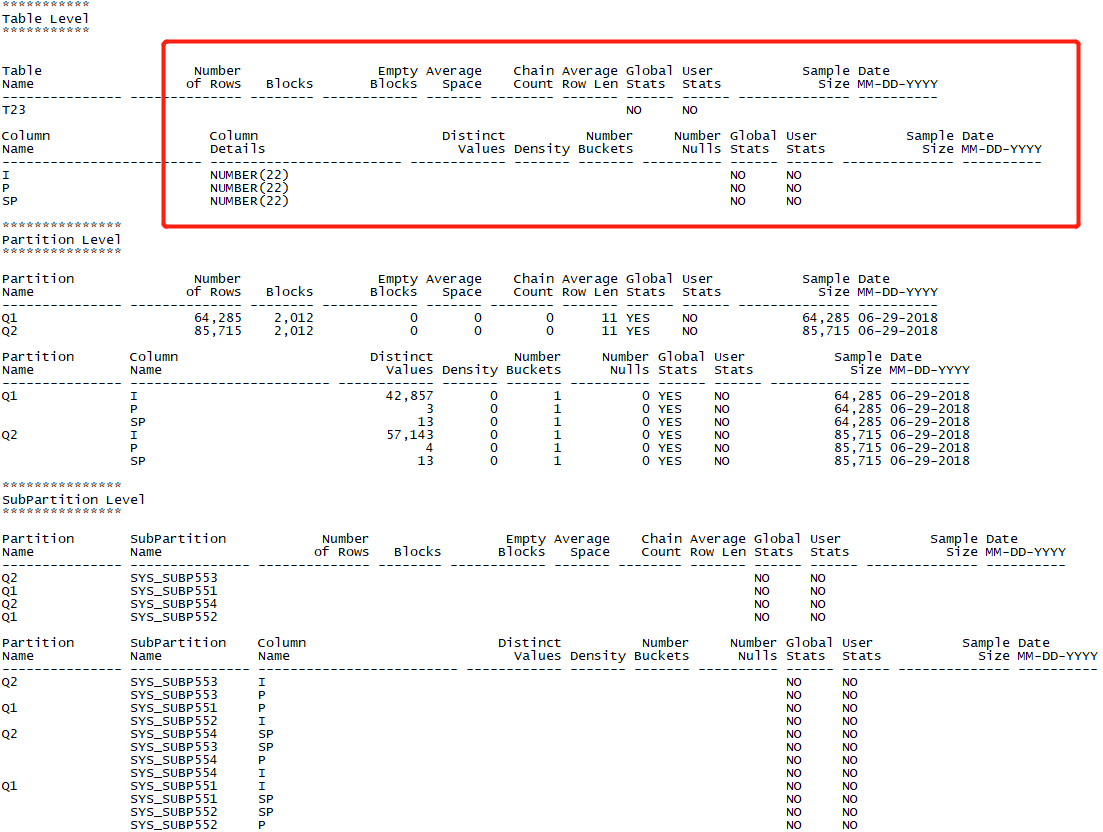
1. 本级的统计信息会准确收集，且global\_stats为yes；
2. 上一级的统计信息如果没有会被级联收集，且global\_stats为no；
3. 下一级的统计信息并不会被收集。

#### 继续只收集子分区的统计信息

我们现在接着上个实验，首先只删除表级的统计信息，然后删部分数据，最后指定granularity为SUBPARTITION再次收集统计信息，我们来看看，分区的统计信息会不会重新收集，表级的统计信息又会不会被收集到。

只删除表级的统计信息：

SQL> exec dbms\_stats.delete\_table\_stats('SCOTT','T23',cascade\_parts=>FALSE);



我们看到，只有表级的统计信息被删掉了。

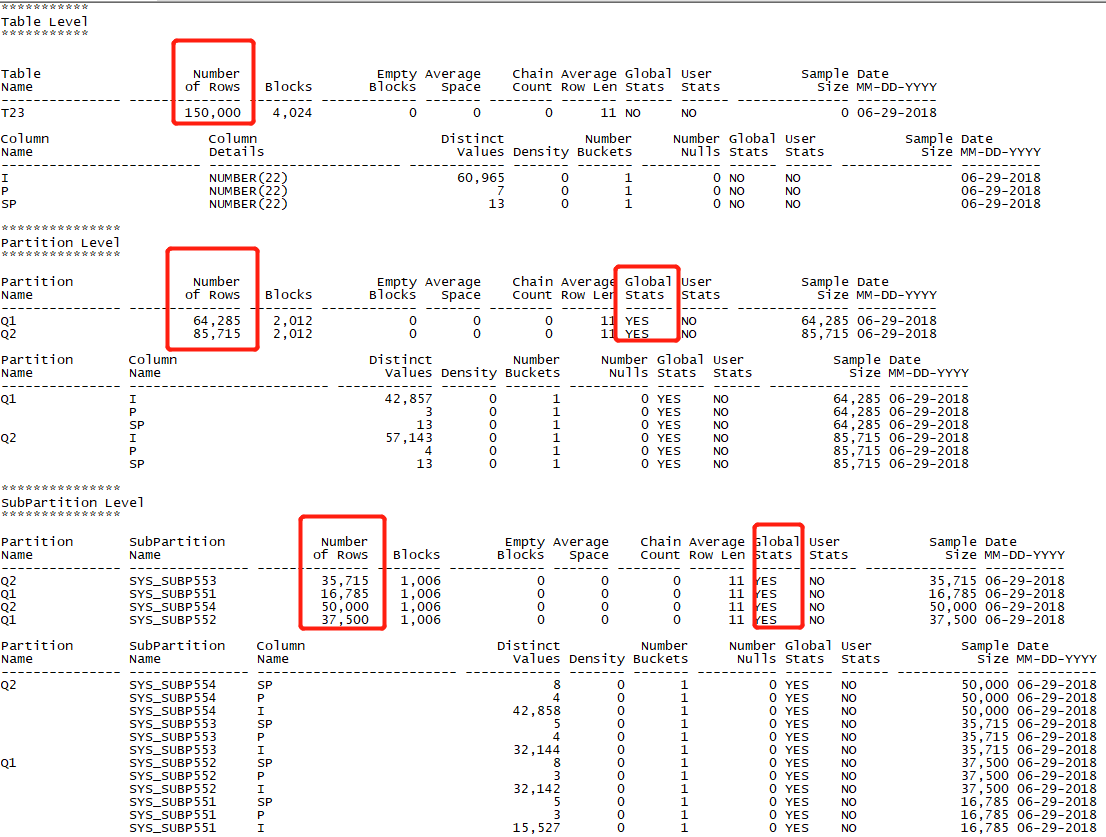
然后我们删一部分数据：

SQL> delete from t23 where rownum<10001;

SQL> commit;

接着我们重新收集统计信息，指定granularity为SUBPARTITION：

SQL> exec dbms\_stats.gather\_table\_stats(ownname=>'SCOTT',tabname=>'T23',estimate\_percent=>100, granularity=>'SUBPARTITION');



从上图，我们可以看到如下俩点：

1. 表级的统计信息依然被收集了，且global\_stats为no，即统计信息是由分区统计信息推导出来的。因为我们同时也看到表级统计出来的行数仍然是150000。
2. 子分区的统计信息global\_stats变为yes，这很正常，但是我们发现分区的统计信息的global\_stats仍为yes，也就是说分区的统计信息并没有被重新收集。因为我们看到子分区的行数统计是准确的，但是分区统计的行数仍然是旧的统计信息。

由第二个实验我们可以知道：

1. 只收集子分区统计信息，会级联收集上级或上上级的global\_stats不为yes的统计信息
2. 可以跳级收集，虽然并没有重新收集分区的统计信息，但是仍会从分区推导出表级的统计信息
3. 如果非本级已有的统计信息global\_stats为yes，则不会被重新收集。

#### 实验总结

由上述实验，我们做如下总结：

1. 千万别用ANALYZE对分区表进行统计信息的收集和删除，更别说与dbms\_stats进行混用；
2. 对分区表进行再次收集的时候，建议使用ALL；
3. 如果不使用ALL，那一定要先对之前的统计信息进行删除，否则可能会导致各级统计信息不一致。

## 列的统计信息

### 概要

列的统计信息可通过DBA\_(TAB/PART/SUBPART)\_COL\_STATISTICS来查看。

下面贴出脚本关于查询列的统计信息部分的代码：

表级的：

select

COLUMN\_NAME,

decode(t.DATA\_TYPE,

'NUMBER',t.DATA\_TYPE||'('||

decode(t.DATA\_PRECISION,

null,t.DATA\_LENGTH||')',

t.DATA\_PRECISION||','||t.DATA\_SCALE||')'),

'DATE',t.DATA\_TYPE,

'LONG',t.DATA\_TYPE,

'LONG RAW',t.DATA\_TYPE,

'ROWID',t.DATA\_TYPE,

'MLSLABEL',t.DATA\_TYPE,

t.DATA\_TYPE||'('||t.DATA\_LENGTH||')') ||' '||

decode(t.nullable,

'N','NOT NULL',

'n','NOT NULL',

NULL) col,

NUM\_DISTINCT,

DENSITY,

NUM\_BUCKETS,

NUM\_NULLS,

GLOBAL\_STATS,

USER\_STATS,

SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from dba\_tab\_columns t

where

table\_name = upper('&Table\_name')

and owner = upper(nvl('&Owner',user))

/

分区级的：

select

PARTITION\_NAME,

COLUMN\_NAME,

NUM\_DISTINCT,

DENSITY,

NUM\_BUCKETS,

NUM\_NULLS,

GLOBAL\_STATS,

USER\_STATS,

SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from

dba\_PART\_COL\_STATISTICS t

where

table\_name = upper('&Table\_name')

and owner = upper(nvl('&Owner',user))

/

子分区级的：

select

p.PARTITION\_NAME,

t.SUBPARTITION\_NAME,

t.COLUMN\_NAME,

t.NUM\_DISTINCT,

t.DENSITY,

t.NUM\_BUCKETS,

t.NUM\_NULLS,

t.GLOBAL\_STATS,

t.USER\_STATS,

t.SAMPLE\_SIZE,

to\_char(t.last\_analyzed,'MM-DD-YYYY')

from

dba\_SUBPART\_COL\_STATISTICS t,

dba\_tab\_subpartitions p

where

t.table\_name = upper('&Table\_name')

and t.owner = upper(nvl('&Owner',user))

and t.subpartition\_name = p.subpartition\_name

and t.owner = p.table\_owner

and t.table\_name=p.table\_name

/

然后介绍一下这些列的含义：

COLUMN\_NAME：列名

DATA\_TYPE：数据类型

DATA\_LENGTH：数据长度

NUM\_DISTINCT：列的唯一值个数

DENSITY：密度

NUM\_BUCKETS：直方图桶数

NUM\_NULLS：列的空值个数

我认为，列的统计信息是最关键的。因为就我个人而言，对于我判断哪列该加索引，判断执行计划该走哪一列上的索引至关重要。虽然说这些信息自己distinct也能做到，统计信息更多的是帮助cbo进行正确判断的，但是有了列的统计信息，人为判断起来就更快更方便。

言归正传，我们下面来看一下列的统计信息对于cbo的影响。

### 收集方法

收集方法还是使用dbms\_stats.gather\_(table|database|schema|dictionary)\_stats，主要由参数METHOD\_OPT来控制，下面来翻译一下官文：

METHOD\_OPT ：

除了table级别能指定只收集具体的列，database,schema,dictionary这些都只能是FOR ALL。

FOR ALL [INDEXED | HIDDEN] COLUMNS [size\_clause]

FOR COLUMNS [column\_clause] [size\_clause]

size\_clause为SIZE {integer | REPEAT | AUTO | SKEWONLY}

column\_clause为column\_name | extension name | extension

size\_clause是决定直方图收集方式的，它可以是下面4种设置：

1. integer ：直方图桶数，12c中该值为1到2048。如果是1，则表示不收集直方图。
2. REPEAT ：只在已经存在了直方图的列上重新收集直方图
3. AUTO（默认值） ：ORACLE根据列的数据分布和列的负载自动判断是否收集列的直方图
4. SKEWONLY ：ORACLE根据列的倾斜程度来判断收集哪些列的直方图。

column\_clause是定义具体收集哪一列或哪些列，再次提醒这个只能在gather\_table\_stats中使用：

1. column\_name ：某一列的列名
2. extension ：收集多列直方图时，（列名，列名，列名…）

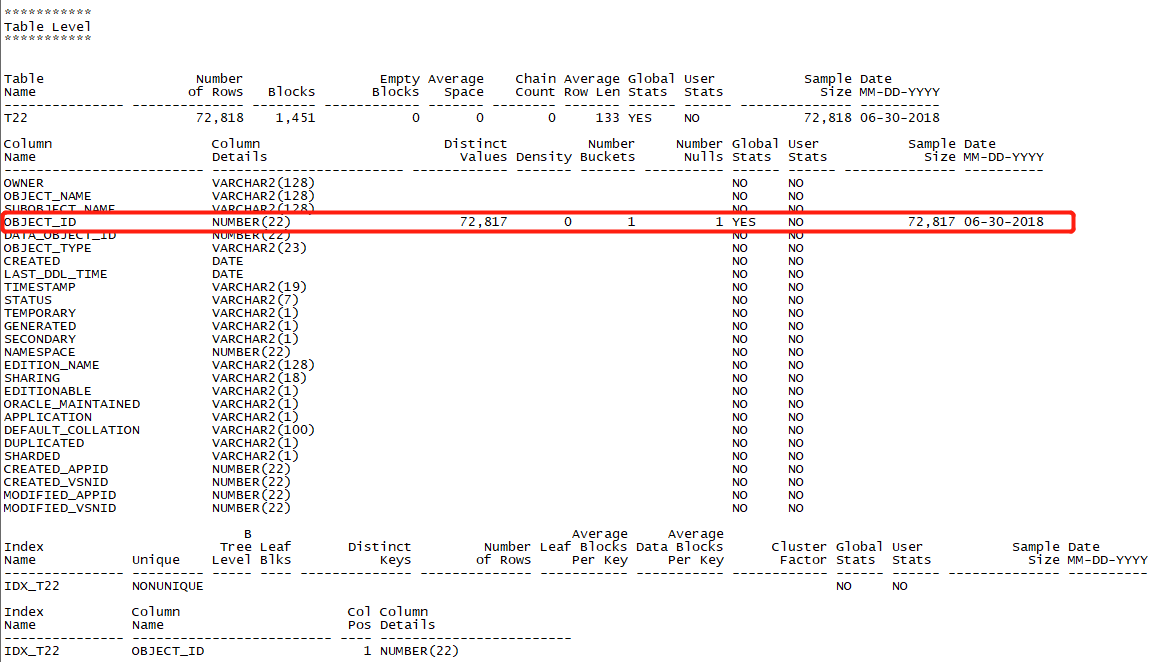
method\_opt的默认值是FOR ALL COLUMNS SIZE AUTO。也就是收集全部列，自动判断是否收集直方图。

接下来我们看具体的一些收集实验。

#### 收集单列统计信息

执行如下指令，只收集T22表的列object\_id的统计信息，不收集直方图（统计信息已提前清空）：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T22',estimate\_percent => 100,method\_opt => 'for columns object\_id size 1',cascade=>false);



我们看到，事实上并不是只收集了列的统计信息，同时也收集了表级的统计信息。

#### 收集直方图

收集直方图的方式就是指定size的值，如果显式指定了size xxx，就是“定制”直方图，否则就是默认的size auto。SIZE后面可跟的选项在上面已经列举，这里不再赘述。

直方图的信息可以在DBA\_(TAB/PART/SUBPART)\_HISTOGRAMS中查到。

##### 收集直方图的意义

如果列上没有直方图，那ORACLE计算选择率的时候，是按照数据是均匀分布的来计算的。那如果数据并不是均匀分布，比如如下我们做的这个实验：

先布置实验数据：

create table t24(a number(5));

begin

for i in 1..5000 loop

insert into t24 values(1);

end loop;

insert into t24 values(2);

commit;

end;

/

SQL> create index idx\_a on t24(a);

SQL> select a,count(\*) from t24 group by a;

A COUNT(\*)

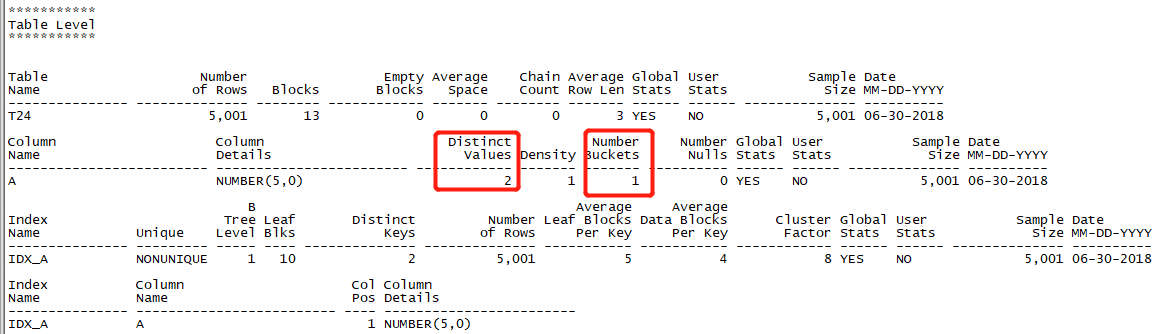
---------- ----------

1 5000

2 1

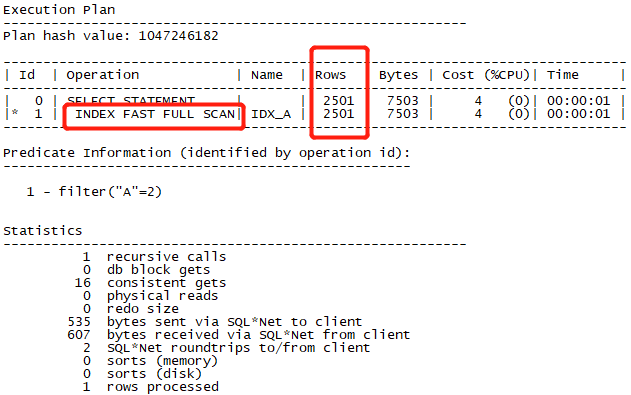
现在我们可以看到表t24的a列就俩个值共5001行，a=2就只有1行，a列上有索引，现在我们收集统计信息，指定size为1：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T24',estimate\_percent => 100,method\_opt => 'for columns a size 1');



现在我们来做如下查询并查看执行计划：

SQL> select \* from t24 where a=2;

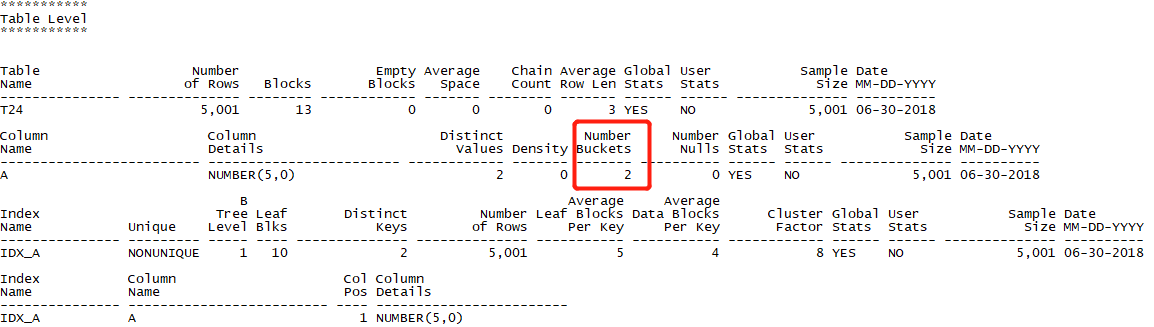


我们看到，由于CBO认为数据是均匀分布的，所以这里估算出的e-rows为2501行，即认为有2501行a=2，而事实上a=2只有1行。

显然错误的估算导致CBO选择了错误的执行计划，这里使用了索引快速全扫描。

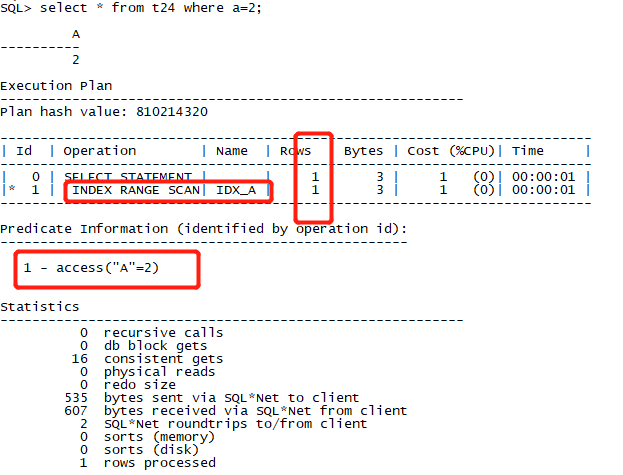
现在我们重新收集a列的统计信息，使用size 2，意味着这次要收集直方图，使用俩个桶：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T24',estimate\_percent => 100,method\_opt => 'for columns a size 2');



再次执行上面的查询：

SQL> select \* from t24 where a=2;



这次我们看到，CBO由于正确的估算出了e-rows为1,索引使用了索引范围扫描，走了正确的执行计划。

由上述的实验，我们至少可以得到俩个结论：

1. 对于数据分布严重不均匀的列，收集直方图可以让cbo算出正确的选择率，从而选择正确的执行路径；
2. 如果该列上没有索引，收集直方图也是没有意义的。因为并没有多余的执行计划可供cbo进行选择。

##### 直方图的分类

###### Frequency

如果我们一个distinct值放入一个bucket，也就是说count(distinct xxx)是多少，就指定size是多少，那么这样收集出来的就是Frequency类直方图。12c之前size最大值为254。我们本次实验环境就是12c，所以根据官文，size可以指定到2048。

我再来准备一张实验表：

create table t25(x number(5));

begin

for i in 1..3000 loop

insert into t25 values(1);

end loop;

commit;

for i in 1..100 loop

insert into t25 values(2);

end loop;

commit;

for i in 1..800 loop

insert into t25 values(3);

end loop;

commit;

for i in 1..4000 loop

insert into t25 values(4);

end loop;

commit;

for i in 1..16000 loop

insert into t25 values(5);

end loop;

commit;

for i in 1..3400 loop

insert into t25 values(6);

end loop;

commit;

for i in 1..3600 loop

insert into t25 values(7);

end loop;

commit;

for i in 1..3800 loop

insert into t25 values(8);

end loop;

commit;

for i in 1..3500 loop

insert into t25 values(9);

end loop;

commit;

for i in 1..1000 loop

insert into t25 values(10);

end loop;

commit;

end;

/

SQL> select x,count(\*) from t25 group by x order by x;

X COUNT(\*)

---------- ----------

1 3000

2 100

3 800

4 4000

5 16000

6 3400

7 3600

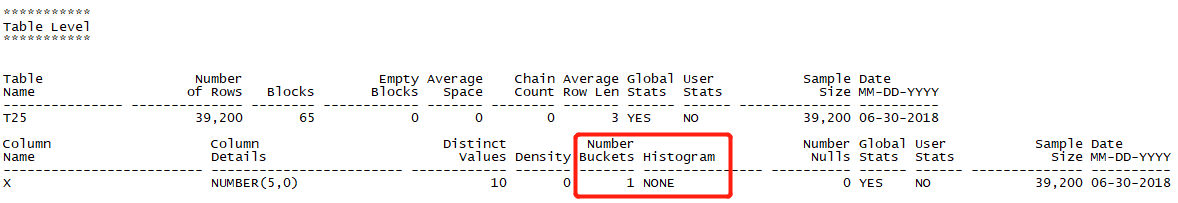
8 3800

9 3500

10 1000

我们先用auto来收集x列的frequency类型的直方图：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T25',estimate\_percent => 100,method\_opt => 'for columns x size auto');



我们看到并没有收集到直方图，这是因为ORACLE只会对那些使用过的列自动收集直方图统计信息。ORACLE自动收集统计信息时会到SYS.COL\_USAGE$中查一下，如果有该列的记录，才会自动收集该列的直方图。关于SYS.COL\_USAGE$可以参考该博客（<http://www.askmaclean.com/archives/smon-maintain-col-usage.html>）

我们现在来查一下：

SQL> select object\_id from dba\_objects where object\_name='T25' and owner='SCOTT';

OBJECT\_ID

----------

73653

SQL> select name,intcol# from sys.col$ where obj#=73653;

NAME INTCOL#

---------- ----------

X 1

SQL> select obj#,intcol#,equality\_preds from sys.col\_usage$ where obj#=73653 and intcol#=1;

no rows selected

确实没有记录，我们使用一下这个列：

SQL> select count(1) from t25 where x=1;

COUNT(1)

----------

3000

SQL> select obj#,intcol#,equality\_preds from sys.col\_usage$ where obj#=73653 and intcol#=1;

no rows selected

我们发现还是木有记录，这个从查询到数据刷新到该视图有延迟，我们可以使用：

SQL> exec DBMS\_STATS.FLUSH\_DATABASE\_MONITORING\_INFO;

也可以直接进行直方图的收集，我们直接进行直方图的收集：

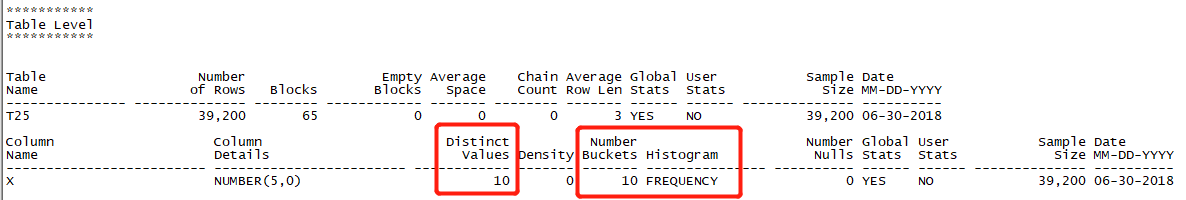
SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T25',estimate\_percent => 100,method\_opt => 'for columns x size auto');

SQL> select obj#,intcol#,equality\_preds from sys.col\_usage$ where obj#=73653 and intcol#=1;

OBJ# INTCOL# EQUALITY\_PREDS

---------- ---------- --------------

73653 1 1



现在我们看到x列上有了直方图，且是frequency的直方图。

我们再来查dba\_tab\_histograms视图：

SQL> select column\_name,table\_name,endpoint\_number,endpoint\_value from dba\_tab\_histograms where table\_name='T25' and column\_name='X' ;

COLUMN\_NAME TABLE\_NAME ENDPOINT\_NUMBER ENDPOINT\_VALUE

-------------------- --------------- --------------- --------------

X T25 3000 1

X T25 3100 2

X T25 3900 3

X T25 7900 4

X T25 23900 5

X T25 27300 6

X T25 30900 7

X T25 34700 8

X T25 38200 9

X T25 39200 10

可以看到，对于Frequency直方图的endpoint\_value就是每一个distinct值。而endpoint\_number是每一个distinct值得行数得累加值，具体来说，比如x=6得行数就是endpoint\_value为6的endpoint\_number减去endpoint\_value为5的endpoint\_number。

这里还是展示崔老师的脚本：

select x endpoint\_value,

count(\*) row\_numbers,

sum(count(\*)) over(order by x range unbounded preceding) endpoint\_number

from t25

group by x;

ENDPOINT\_VALUE ROW\_NUMBERS ENDPOINT\_NUMBER

-------------- ----------- ---------------

1 3000 3000

2 100 3100

3 800 3900

4 4000 7900

5 16000 23900

6 3400 27300

7 3600 30900

8 3800 34700

9 3500 38200

10 1000 39200

这次看的很清晰明了了。

上面是列是数字类型的，那么我们看看字符型的是怎么存的：

SQL> create table t26 (y varchar2(33));

SQL> begin

2 for i in 1..10001 loop

3 insert into t26 values('12');

4 end loop;

5 commit;

6 end;

7 /

SQL> insert into t26 values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1');（有32个a）

SQL> insert into t26 values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1');

SQL> insert into t26 values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa2');

SQL> commit;

SQL> select y,count(\*) from t26 group by y;

Y COUNT(\*)

--------------------------------- --------------------

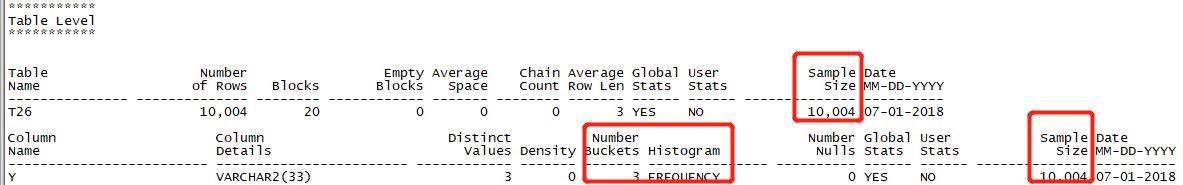
12 10001

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1 2

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa2 1

然后收集y的直方图：

exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T26',estimate\_percent => 100, method\_opt => 'for columns y size auto');



我们现在来查一下endpoint\_numer,endpoint\_value：

SQL> select column\_name,table\_name,endpoint\_number,endpoint\_value from dba\_tab\_histograms where table\_name='T26' and column\_name='Y' ;

COLUMN\_NAME TABLE\_NAME ENDPOINT\_NUMBER ENDPOINT\_VALUE

--------------- ---------- -------------------- --------------------

Y T26 10001 2.55436666548389E+35

Y T26 10003 5.05627904278968E+35

Y T26 10004 5.05627904278968E+35

我们发现有俩个一样的endpoint\_value，但是为了进一步确定这俩个值到底相不相同，我们做如下查询：

select t1.column\_name, t1.table\_name, t1.endpoint\_number, t1.endpoint\_value

from dba\_tab\_histograms t1, dba\_tab\_histograms t2

where t1.table\_name = t2.table\_name

and t1.endpoint\_value = t2.endpoint\_value

and t1.table\_name = 'T26'

and t1.column\_name = 'Y';

COLUMN\_NAME TABLE\_NAME ENDPOINT\_NUMBER ENDPOINT\_VALUE

--------------- ---------- -------------------- --------------------

Y T26 10001 2.55436666548389E+35

Y T26 10003 5.05627904278968E+35

Y T26 10003 5.05627904278968E+35

Y T26 10004 5.05627904278968E+35

Y T26 10004 5.05627904278968E+35

这次我们可以很确定，这俩个endpoint\_value确实是相同的。

我们先按照崔老师教的办法一个值一个值的算。

SQL> select dump('12',16) from dual;

DUMP('12',16)

-------------------

Typ=96 Len=2: 31,32

然后将这个3132用0补齐到30位，即15个字节，再转成10进制：

SQL> select to\_number('313200000000000000000000000000','xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx') endpoint\_value from dual;

ENDPOINT\_VALUE

--------------------

2.55436666548389E+35

可以看到，与上面的第一个endpoint\_value一字不差。

现在我们来看“32个a+1”怎么处理。

DUMP('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1',16)

-----------------------------------------------------------------------------------------------------------------

Typ=96 Len=33: 61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,61,31

按照前面的算法，这里超过了15个字节，那就只取前15个字节：

SQL> select to\_number('616161616161616161616161616161','xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx') endpoint\_value from dual;

ENDPOINT\_VALUE

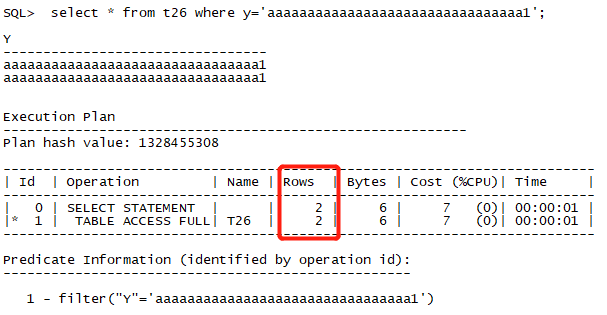
--------------------

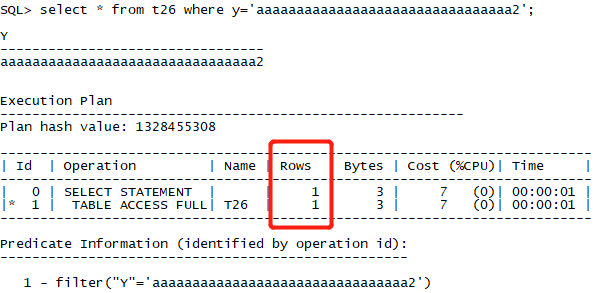
5.05627904278968E+35

我们看到，跟上面的endpoint\_value一模一样。

但是！在12c可以明显看到，虽然算出了同样的endpoint\_value，但是ORACLE依然使用了3个桶去装，这与崔老师所实验的数据库版本情况是不同的。

那我们看一下，oracle是否能真的区分出来这俩个值得不同。其实我们从上面的endpoint\_number也能看出端倪来，肯定是oracle能区分出这俩个值的不同，但我们还是进行查询看下执行计划：





没问题，ORACLE确实能明确的分辨出这俩个值的不同。

崔老师写这块主要是为了证明收集frequency直方图在判断超过32位文本记录的distinct值时会存在误判。但是明显这一点在12c得到了完善。

我们再建个表t27：

SQL> create table t27(id number);

begin

for i in 1..2048 loop

insert into t27 values(i);

insert into t27 values(1);

commit;

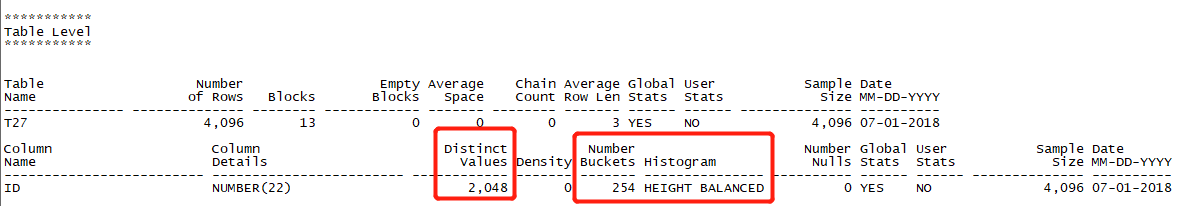
end loop;

end;

/

我们来收集统计信息id的直方图：

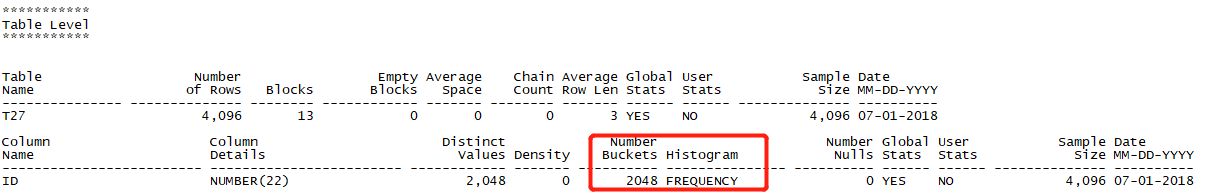
SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T27',estimate\_percent => 100,method\_opt => 'for columns id size auto');



我们看到，自动收集的话最大只给254个桶，所以收集出来的就是height balanced直方图。

我们这次直接指定桶数2048：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T27',estimate\_percent => 100,method\_opt => 'for columns id size 2048');



可以看到，12c确实能分配2048个桶，而不是之前版本的254，但默认还是只分配254个桶。

我们再多插一个distinct值进去，看下能分配2049个桶不。

SQL> insert into t27 values(2049);

SQL> commit;

SQL> select count(distinct id) from t27;

COUNT(DISTINCTID)

-----------------

2049

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T27',estimate\_percent => 100,method\_opt => 'for columns id size 2049');

BEGIN dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T27',estimate\_percent => 100,method\_opt => 'for columns id size 2049'); END;

\*

ERROR at line 1:

ORA-20000: Cannot parse for clause: for columns id size 2049

ORA-06512: at "SYS.DBMS\_STATS", line 36873

ORA-06512: at "SYS.DBMS\_STATS", line 36507

ORA-06512: at "SYS.DBMS\_STATS", line 35428

ORA-06512: at "SYS.DBMS\_STATS", line 21023

ORA-06512: at "SYS.DBMS\_STATS", line 35009

ORA-06512: at "SYS.DBMS\_STATS", line 36230

ORA-06512: at "SYS.DBMS\_STATS", line 36716

ORA-06512: at line 1

我们看要报错。这也是很明显的，官文说size最大就能指定2048的嘛。

###### Height Balanced

当我们分配的桶数比唯一值的总数小或者唯一值个数超过254且自动收集直方图的时候，收集到的直方图就是height balanced直方图。

这种直方图的收集原理是：先把该列的值从小到大排序，然后假设该列的非空值的行数为M，给该直方图分配的桶数为N，那每个桶中的记录数O=M/N。

那么这个桶的编号，要从1开始编，记录在endpoint\_number中，而endpoint\_value中记录该桶中的最大值。但是如果相邻俩个桶的endpoint\_value一样，那就把这俩个桶在数据字典的记录合并，取较大的endpoint\_number作为新的endpoint\_number。这种发生了合并记录的endpoint\_value就叫做popular value，也就说endpoint\_number与上一条记录差值越大，那说明这个endpoint\_value占比越大，所谓的越受“欢迎”。

注意，这里还需要有一个最小值的记录，因为不能光记录这一列的上限，还要记录这一列的下限。于是ORACLE就用一个endpoint\_number=0的记录，把该列的最小值记录在对应的endpoint\_value中。

我们用前面的表T25解释该直方图：

SQL> select x,count(1) from t25 group by x order by x;

X COUNT(1)

---------- ----------

1 3000

2 100

3 800

4 4000

5 16000

6 3400

7 3600

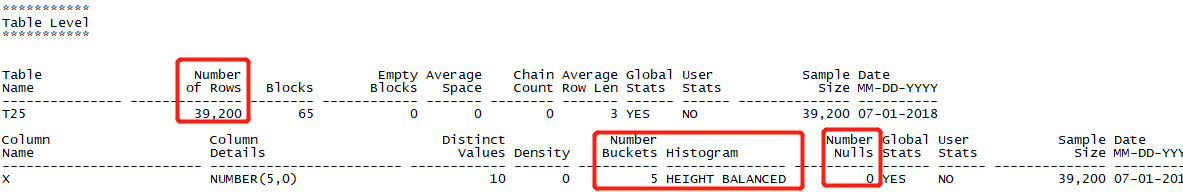
8 3800

9 3500

10 1000

我们用5个桶来收集x列的直方图：

SQL> exec dbms\_stats.gather\_table\_stats(ownname => 'SCOTT',tabname => 'T25',estimate\_percent => 100,method\_opt => 'for columns x size 5');



我先算每个桶的记录数，因为没有null值，所以每个桶的记录数就是39200/5=7840。然后我们算endpoint\_number和endpoint\_value：

select 0 endpoint\_number,min(x) endpoint\_value from t25

union all

select 1 endpoint\_number,max(x) endpoint\_value from (select x from t25 order by x) where rownum<=7840

union all

select 2 endpoint\_number,max(x) endpoint\_value from (select x from t25 order by x) where rownum<=7840\*2

union all

select 3 endpoint\_number,max(x) endpoint\_value from (select x from t25 order by x) where rownum<=7840\*3

union all

select 4 endpoint\_number,max(x) endpoint\_value from (select x from t25 order by x) where rownum<=7840\*4

union all

select 5 endpoint\_number,max(x) endpoint\_value from (select x from t25 order by x) where rownum<=7840\*5;

ENDPOINT\_NUMBER ENDPOINT\_VALUE

--------------- --------------

0 1

1 4

2 5

3 5

4 8

5 10

那么这时候我们可以看到endpoint\_number为2和3的endpoint\_value都是5，所以5是个popular value（有16000行，确实是这张表里最popular的），同时，我们要把这俩行记录合并，endpoint\_number取3。

最终得到的结果就是下面的查询结果：

SQL> select column\_name,table\_name,endpoint\_number,endpoint\_value from dba\_tab\_histograms where table\_name='T25' and column\_name='X';

COLUMN\_NAME TABLE\_NAME ENDPOINT\_NUMBER ENDPOINT\_VALUE

------------ ---------- --------------- --------------

X T25 0 1

X T25 1 4

X T25 3 5

X T25 4 8

X T25 5 10

确实与我们的计算结果是吻合的。

### 选择率的计算

在没有直方图也没有谓词越界的情况下：

等值查询的选择率：

Selectivity = (1/NUM\_DISTINCT)\*((NUM\_ROWS-NUM\_NULLS)/NUM\_ROWS)

范围查询的选择率：

NULL\_Adjust = (NUM\_ROWS-NUM\_NULLS)/NUM\_ROWS

< VAL:

Selectivity = ((VAL-LOW\_VALUE)/(HIGH\_VALUE-LOW\_VALUE)) \* NULL\_Adjust

<= VAL:

Selectivity = ((VAL-LOW\_VALUE)/(HIGH\_VALUE-LOW\_VALUE) + 1/NUM\_DISTINCT) \* NULL\_Adjust

>VAL:

Selectivity = ((HIGH\_VALUE-VAL)/(HIGH\_VALUE-LOW\_VALUE)) \* NULL\_Adjust

>= VAL:

Selectivity = ((HIGH\_VALUE-VAL)/(HIGH\_VALUE-LOW\_VALUE) + 1/NUM\_DISTINCT) \* NULL\_Adjust

Between VAL1 and VAL2

Selectivity = ((VAL2-VAL1)/(HIGH\_VALUE-LOW\_VALUE) + 2/NUM\_DISTINCT) \* NULL\_Adjust

既然是学习笔记，这里我还是把崔老师总结的这部分完全的记录了下来，因为这些就是cbo计算选择率的公式，对于深入了解cbo的机制有很有帮助的。

其实对于人为判断而言，看的主要还是NUM\_DISTINCT/(NUM\_ROWS-NUM\_NULLS)。

那么cbo用这个选择率算什么呢？就是算E-rows：

E-rows = round(NUM\_ROWS \* Selectivity)

这里我们演示一个between and的例子。

首先我们查看一下之前收集过统计信息的T22表data\_object\_id列的统计信息：

SQL> select num\_rows from dba\_tables where table\_name='T22';

NUM\_ROWS

--------------

72,818

SQL> select low\_value,high\_value,num\_distinct,num\_nulls from dba\_tab\_col\_statistics where table\_name='T22' and column\_name='DATA\_OBJECT\_ID';

LOW\_VALUE HIGH\_VALUE NUM\_DISTINCT NUM\_NULLS

-------------------- -------------------- ------------ ----------

C103 C308251F 5,873 66,908

我们算一下这个最小值和最大值：

SQL> select min(data\_object\_id) min,dump(min(data\_object\_id),16) low\_value from t22;

MIN LOW\_VALUE

------- --------------------

2 Typ=2 Len=2: c1,3

SQL> select max(data\_object\_id) max,dump(max(data\_object\_id),16) high\_value from t22;

MAX HIGH\_VALUE

---------- -------------------------

73630 Typ=2 Len=4: c3,8,25,1f

我们看到，由于该实验表数据是不动的且统计信息是百分百采样获得的，所以所得最大值最小值与统计信息中记录的完全一致。

不过我们这里正确的转换low\_value，high\_value的方法如下：

首先要确认这一列的datatype是什么，是number就要先申明一个number的变量：

SQL> var temp number

SQL> exec dbms\_stats.convert\_raw\_value('C308251F',:temp);

PL/SQL procedure successfully completed.

SQL> select :temp from dual;

:TEMP

----------

73630

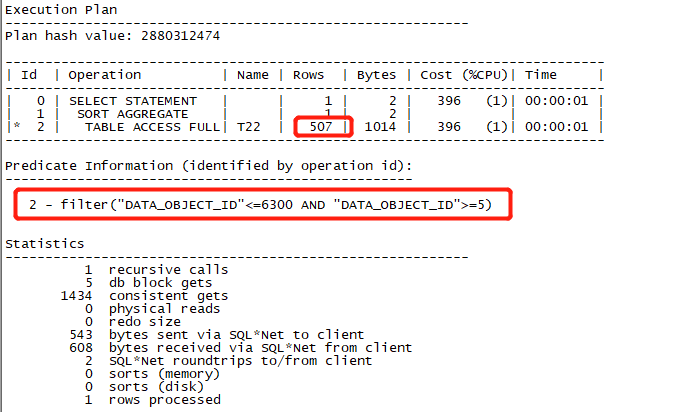
现在我们做如下查询：

SQL> select count(1) from t22 where data\_object\_id between 5 and 6300;

COUNT(1)

----------

1574



我们来用公式算下E-rows：

SQL> select round(72818\*(((6300-5)/(73630-2)+2/5873)\*((72818-66908)/72818))) "E-rows" from dual;

E-rows

----------

507

确实一样，这个公式基本上是准的，可能有偏差，但是不大。