ORACLE

Maclean讲SQL调优精要

SH'OUG
SHANGHAI ORACLE USERS GROUP
上海ORACLE用户组

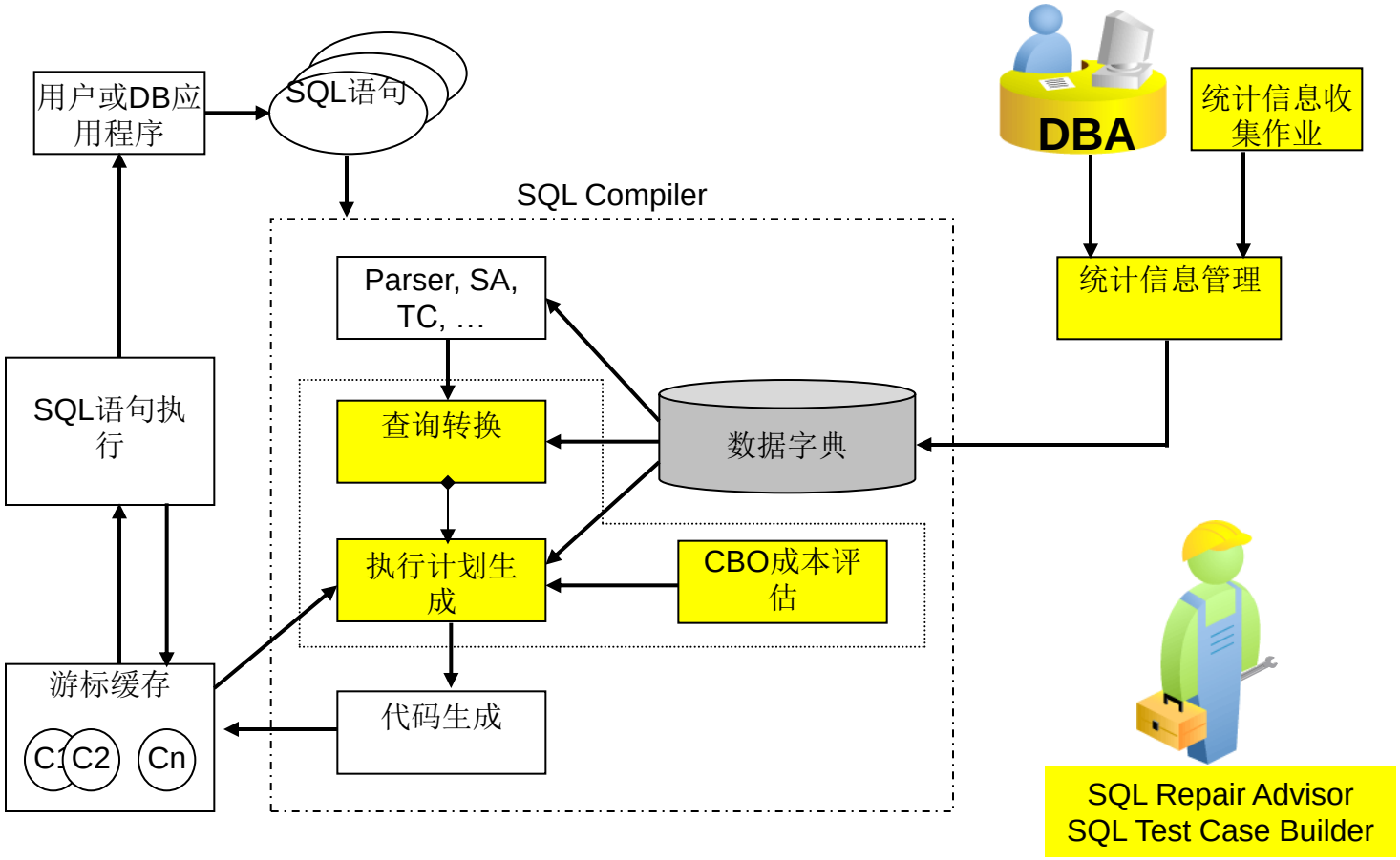
刘相兵(Maclean Liu)
liu.maclean@gmail.com

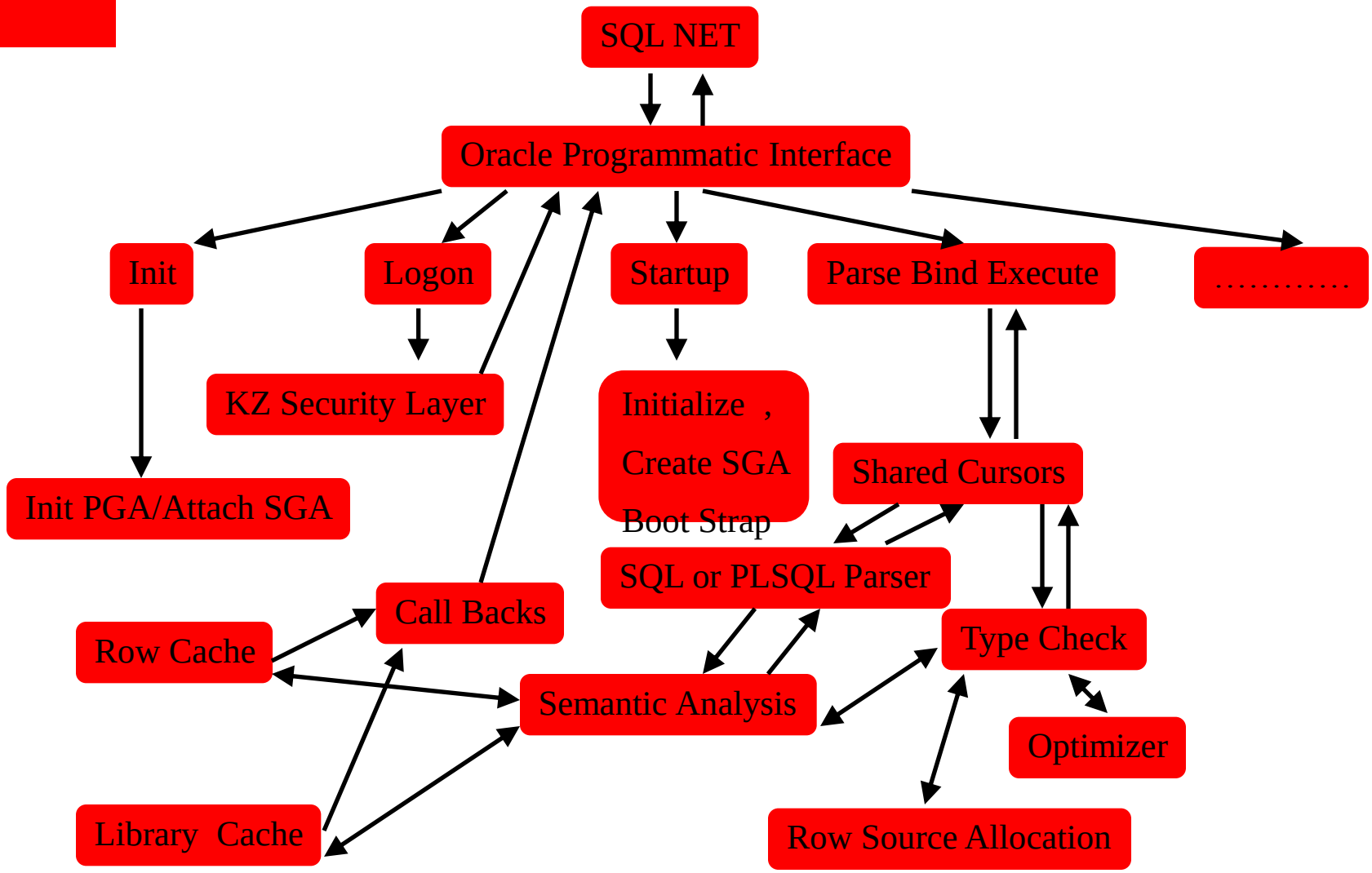
SHOUG- 上海Oracle用户组线上活动

优化的思想

- 80%的性能问题由10%的Top SQL引起
- 优化就是通过各种手段降低SQL语句所需要消耗的逻辑读、物理读、CPU时间、热点争用、Latch/Mutex。最有效的调整手段是调整执行计划
- 兴一利远远不如除一害，通过修改几个参数把Top SQL造成的问题彻底解决是不可能的，除非这些参数影响了SQL的执行计划。
- 不需要搞懂所有的执行计划和优化器原理，只要对准SQL的症结下手即可。

SQL语句处理过程

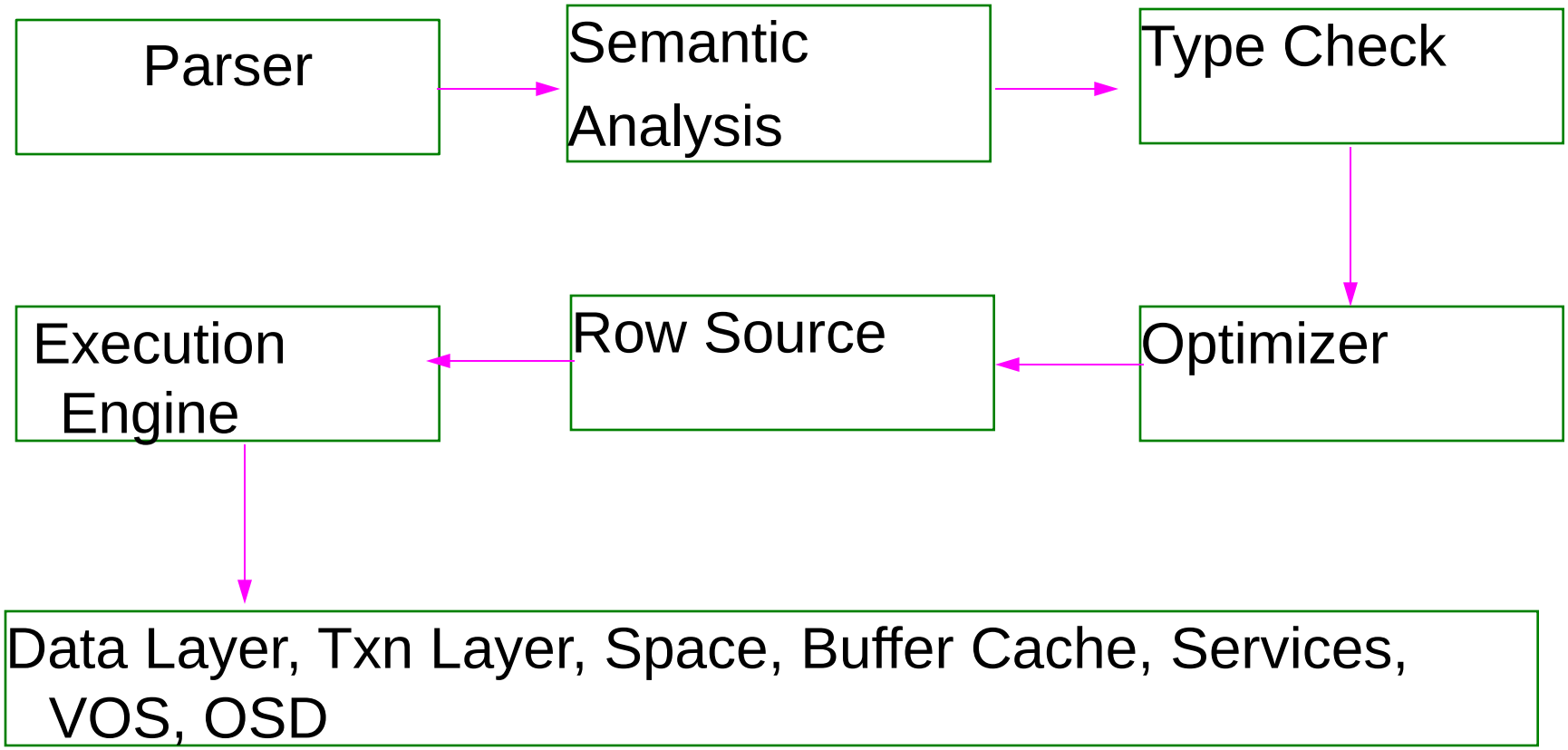

用户或DB应用程序
SQL语句
SQL Compiler
DBA
统计信息收集作业
统计信息管理
Parser, SA, TC, …
查询转换
数据字典
执行计划生成
CBO成本评估
SQL语句执行
游标缓存
C1
C2
Cn
代码生成
SQL Repair Advisor
SQL Test Case Builder


SQL NET
Oracle Programmatic Interface
Init
Logon
Startup
Parse Bind Execute
……
KZ Security Layer
Initialize ,
Create SGA
Boot Strap
Shared Cursors
Init PGA/Attach SGA
SQL or PLSQL Parser
Call Backs
Row Cache
Type Check
Semantic Analysis
Optimizer
Library Cache
Row Source Allocation

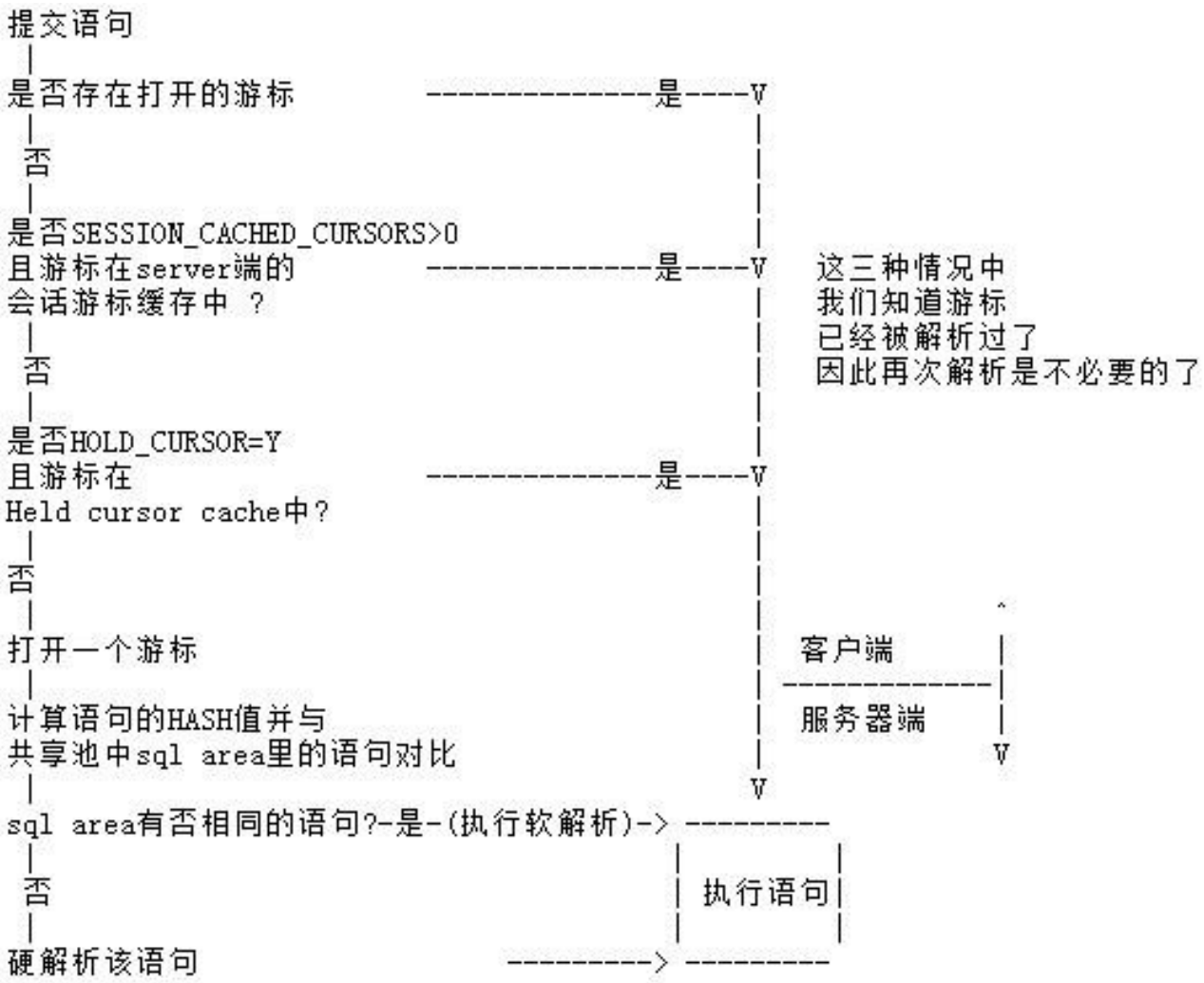
SQL语句的一生

- Oracle Call Interface/JDBC提供API以便与DB交流
- Oracle Programmatic Interface OPI提供了与SQL*NET交流数据或命令的代码层
- SQL Parser语法解析SQL语句以便构造基本解析树
- SQL Semantic语义解析使用library Cache和Row Cache，在基本解析树的基础上检验和生成结果
- Optimizer优化器层面负责优化该解析树，以便获得成本最低的执行计划
- SQL执行层面最后生成一个执行树，以便执行并获得最终结果
- Cursor游标层面在library cache中提供缓存执行计划的框架，以便给大量session反复执行

SQL语句的一生


Parser
Semantic Analysis
Type Check
Optimizer
Row Source
Execution Engine
Data Layer, Txn Layer, Space, Buffer Cache, Services, VOS, OSD

SQL解析的流程


提交语句
是否存在打开的游标
是
否
是否SESSION_CACHED_CURSORS>0
且游标在server端的
会话游标缓存中 ?
是
否
是否HOLD_CURSOR=Y
且游标在
Held cursor cache中?
是
否
打开一个游标
计算语句的HASH值并与
共享池中sql area里的语句对比
sql area有否相同的语句?-是-(执行软解析)->
否
硬解析该语句
执行语句
这三种情况中
我们知道游标
已经被解析过了
因此再次解析是不必要的了
客户端
服务器端

SQL语句运行可以分作三个运行阶段

1. **解析Parse：**
   - 硬解析 – 包括语法解析、语义解析、查询转换、生成执行计划等等
   - 软解析 – 包括语法解析、语义解析和定位游标缓存
   - 软软解析
   - 直接复用打开的游标

2. **执行Execute，具体取决于执行计划：**
   - Row access
   - 排序
   - Join

3. **返回结果集Fetch，Client-Server间的网络交互ResultSet**

session在执行SQL时的状态

进程在执行SQL语句时，其Status总是为ACTIVE

对于上述三个运行阶段，要么跑在CPU上，要么在等待事件上

- Parse阶段常见的等待事件： row cache lock、cursor: pin S wait on X、library cache pin/lock 等等

- Execute阶段常见的等待事件：db file sequential read、db file scattered read、buffer busy waits等

- Fetch阶段常见的等待事件 ：SQL*Net more data to client

如何了解一条SQL的运行状况

使用10046+tkprof 了解Parse、Execute、Fetch的状况，以及CPU时间和等待事件

Alter session set events ‘10046 trace name context forever,level 12’;

select * from maclean;

tkprof g10r25_ora_8249.trc g10r25_ora_8249.tkf

```
select *
from
 maclean


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    20002      0.05       0.61        829      20780          0      300001
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    20004      0.05       0.61        829      20780          0      300001

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: SYS

Rows     Row Source Operation
-------  ---------------------------------------------------
 300001  TABLE ACCESS FULL MACLEAN (cr=20780 pr=829 pw=0 time=1501420 us)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                   20002        0.00          0.06
  db file sequential read                         1        0.00          0.00
  db file scattered read                         60        0.00          0.00
  SQL*Net message from client                 20002        0.00          2.31
```

如何快速了解一条SQL的历史运行情况

sql execution history.sql

SnapId	PLAN_HASH_VALUE	Date time	No. of exec	LIO/exec	CPUTIM/exec	ETIME/exec	PIO/exec	ROWs/exec
2338	3020000309	05/30/13_1500_1600	3	4364191.00	61.05	185.69	16591.00	4.00
2339	3020000309	05/30/13_1600_1700	6	2136884.00	32.02	62.47	33511.00	1.33
2340	3020000309	05/30/13_1700_1800	1	12048990.00	148.24	351.38	150510.00	9.00
2370	326151572	05/31/13_2300_0000	6	2928786.17	6.81	6.86	.00	2.33
2391	3020000309	06/01/13_1509_1600	779	81951.16	.24	.26	.93	5.37
2392	3020000309	06/01/13_1600_1700	1132	136760.77	.39	.43	.00	5.11
2393	3020000309	06/01/13_1700_1800	217	309310.45	1.04	1.19	.00	4.59
2394	3020000309	06/01/13_1800_1810	315	171784.19	.58	.72	.00	8.24
2395	3020000309	06/01/13_1810_1813	65	467600.37	1.75	2.15	.02	44.14
2396	3020000309	06/01/13_1813_1900	1761	242481.31	.69	.75	.00	7.84
2397	3020000309	06/01/13_1900_2000	2090	333452.12	.85	.90	.00	13.15
2398	3020000309	06/01/13_2000_2100	2130	340486.04	.81	.86	.00	19.55
2399	3020000309	06/01/13_2100_2200	1954	424351.28	.99	1.02	.00	27.80
2400	3020000309	06/01/13_2200_2300	1980	411204.19	.95	1.00	.00	35.54
2401	3020000309	06/01/13_2300_2346	679	402201.79	.94	.97	.00	112.16
2402	3020000309	06/01/13_2346_2346	8	390077.88	.94	.95	.00	9529.63
2403	3020000309	06/01/13_2346_0100	474	332207.91	.78	.80	.00	169.66
2411	3020000309	06/02/13_0800_0900	126	458337.02	1.18	1.19	.14	643.48
2412	3020000309	06/02/13_0900_1000	326	280842.13	.72	.74	.03	252.76
2413	3020000309	06/02/13_1000_1100	453	134573.87	.34	.35	3.26	185.91
2414	3020000309	06/02/13_1100_1200	536	.00	.52	.58	69.05	161.68
2415	3020000309	06/02/13_1200_1300	547	82730.33	.19	.20	.00	163.25
2416	3020000309	06/02/13_1300_1400	472	101107.11	.24	.25	2.75	192.81

如何快速了解一条SQL的历史执行计划

- **From AWR:**

```
select plan_table_output from table
(dbms_xplan.display_awr('&sql_id',null,null,'ADVANCED +PEEKED_BINDS'));
```

- **From Cursor Cache:**

```
SELECT plan_table_output FROM
TABLE(DBMS_XPLAN.DISPLAY_CURSOR('&SQL_ID',0,'ALLSTATS'));
```

Parse与Execute的因果关系

Parse的最终产物是 SQL PLAN执行计划

CBO Optimizer 根据 优化器算法、统计信息、优化器参数 选择成本最低的执行计划

对于Optimizer而言，他不可能真的去执行这个语句，所以最后获得的执行计划是Optimizer 筹划、猜想出来的最佳执行计划

Real World总是和理论有很大的差距，所以CBO Optimizer 认为最好的执行计划未必是好的。 Cost成本最低的执行计划也未必是最好的执行计划

DBA的优化SQL主要工作是 优化Parse 减少不必要的软硬解析，和优化执行计划。

总而言之，Optimizer是脑袋负责思考，Execute Layer只负责执行脑袋给出的命令。

所以Parse/Optimizer阶段至关重要！

SQL – Hard Parse硬解析

- **SQL说到底是一种编程语言**
  - 要运行一种计算机语言，就需要compiler编译器，optimizer优化器和code generator 代码生成器
- **SQL是一种只说明需求的语言**
  - 所以optimizer优化器显得更重要
- **在Oracle中SQL编译的成本很昂贵**
  - Oracle的理念是 编译一次 处处运行
  - 第一次Parse解析称之为Hard Parse
  - 今后的执行最好能重用第一次编译后获得的”程序”，这个编译好的程序称之为”shared Cursor”共享游标
  - 是否能重用一个游标取决于优化和运行环境optimizer env和execution env v$sql_shared_cursor
- **愿景**
  - 优化的目标是 减少 编译和 执行时间
  - 对于DW和OLAP，SQL特点复杂而不频繁，为了更好的执行计划，可以增加编译时间
  - 对于OLTP，SQL特点简单而频繁，较短的编译时间也可以得到较佳执行计划

SQL – Hard Parse硬解析

一个完整的Hard Parse由以下部分组成：

- Syntax Parse 语法解析， 首先确认语句语法是否有问题；从SQL文本转换为解析树结构
- Semantic Parse 语义解析， 分析这个SQL到底什么意思
- Type Check 类型检查， 例如 给’1’=1 做转换 ‘1-JAN-1998’转换为日期 1+2 转换为3
- Optimizer - Transformations优化器转换 ，等价重写该SQL语句
- Optimizer – Physical Optimizer：
  - Access path analysis 访问路径分析
  - Join order and Join method 连接方式和连接顺序
  - Partition pruning 分区裁剪
  - 星型转换或和OR-expansion
- Code Generator 代码生成， 创建最后可执行的数据结构，并做一些例如Parallel、Partition Push up之类的优化

游标共享(复用游标) 开发人员视角

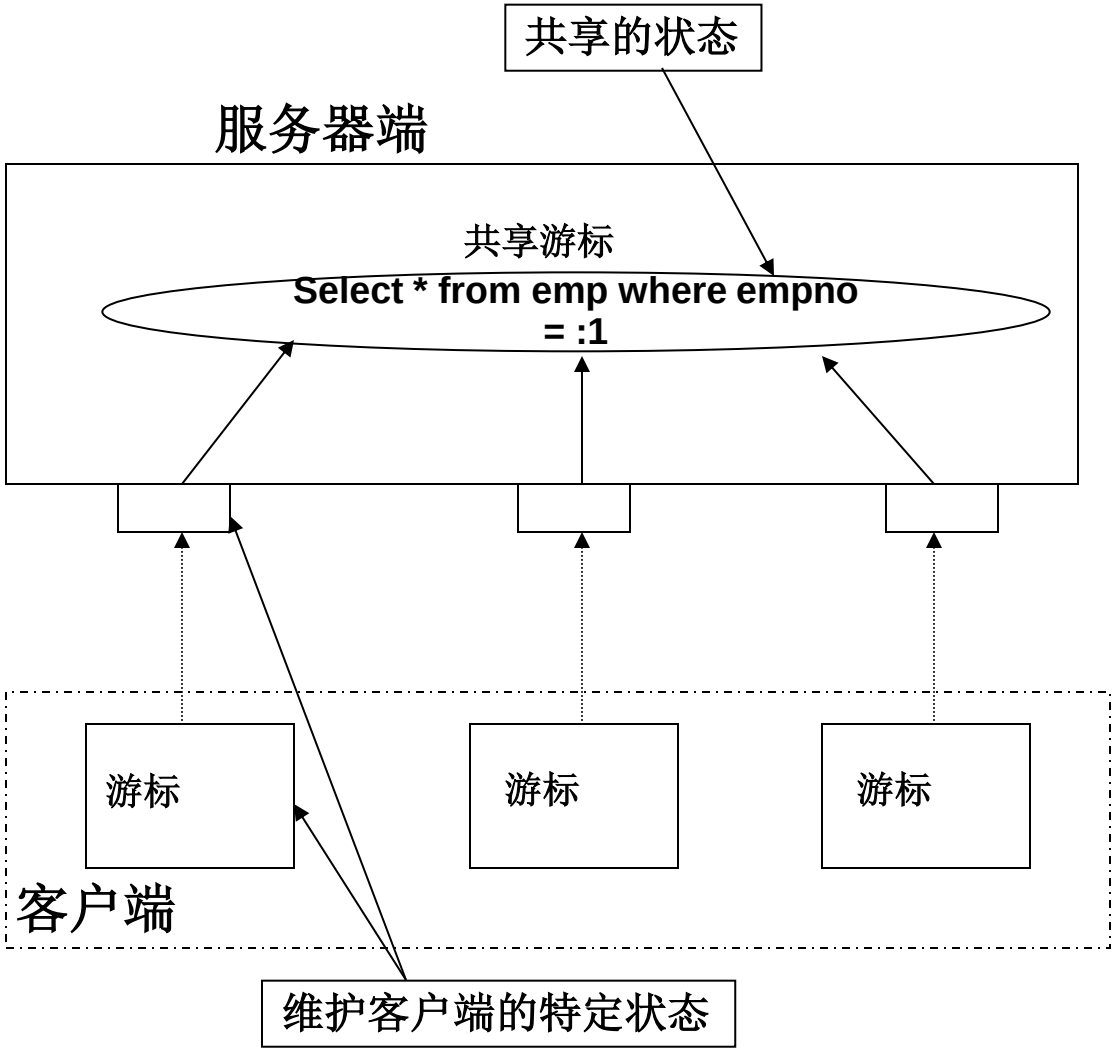
- 游标可以被共享使用的前提是，SQL文本需要一致
  - 当使用常量硬编码时游标默认无法共享
    - Select * from emp where empno='10' 不匹配于
    - Select * from emp where empno='20'
- 因此需要绑定变量(有点像函数的变量)
  - Select * from emp where empno=:1;
  - 动态SQL中绑定：
    - Execute immediate 'select * from emp where empno=:1' using var_empno;
  - Sqlplus中绑定：
    - Variable var_empno;
    - Select * from emp where empno=:var_empno;
  - Java中PreparedStatement绑定：
    - String insert = "insert into insertit values (?,?,?,?)";
    - PreparedStatement pstmt4 = cnn1.prepareStatement(insert);
    - pstmt4.setInt(1, u);

游标共享(复用游标) 开发人员视角

为了实现游标共享，其他一些因素也需要匹配，任何因素均可能影响计划

- 编译模式 **Parsing Schema**
- **NLS** 设置
- 优化环境：
  - 统计信息
  - 优化器参数: optimizer_features_enable、optimizer_index_cost_adj etc
  - 11g Adaptive Cursor Sharing, 考虑关掉ACS
  - 11g Sql Plan Management
  - 11g Cardinality Feedback ， 考虑关掉CF

搞清楚Server和Client端不同的Cursor游标



在客户端cursor代表行的迭代

在服务器端游标是一种数据结构

不绑定变量导致游标无法共享，DBA视角

在OLTP环境中，游标无法共享：

- 引起大量的硬解析 – 万恶之源，具体体现：
  - 硬解析消耗大量的CPU
  - 硬解析引起大量的Concurrency类等待事件：cursor: pin S wait on X、latch: shared pool、row cache lock、library cache lock/pin等
  - 产生过多不重用的游标，消耗Shared Pool的Free Chunk，最后无chunk可用
  - 高CPU，高负载情况下系统容易不稳定

某网上商城，优化前用户根本打不开商品页面：

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	21899	31-May-12 13:00:36	283	5.4
End Snap:	21901	31-May-12 15:01:01	418	4.3
Elapsed:		120.42 (mins)		
DB Time:		28,865.15 (mins)		

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	1,728,400.65	99.80
parse time elapsed	729,751.51	42.14
hard parse elapsed time	594,512.01	34.33
DB CPU	28,477.33	1.64
failed parse elapsed time	521.94	0.03
PL/SQL execution elapsed time	350.13	0.02
connection management call elapsed time	276.99	0.02
PL/SQL compilation elapsed time	2.38	0.00
hard parse (bind mismatch) elapsed time	0.96	0.00
hard parse (sharing criteria) elapsed time	0.96	0.00
repeated bind elapsed time	0.15	0.00
sequence load elapsed time	0.03	0.00
DB time	1,731,909.23	
background elapsed time	1,922.23	
background cpu time	52.53	

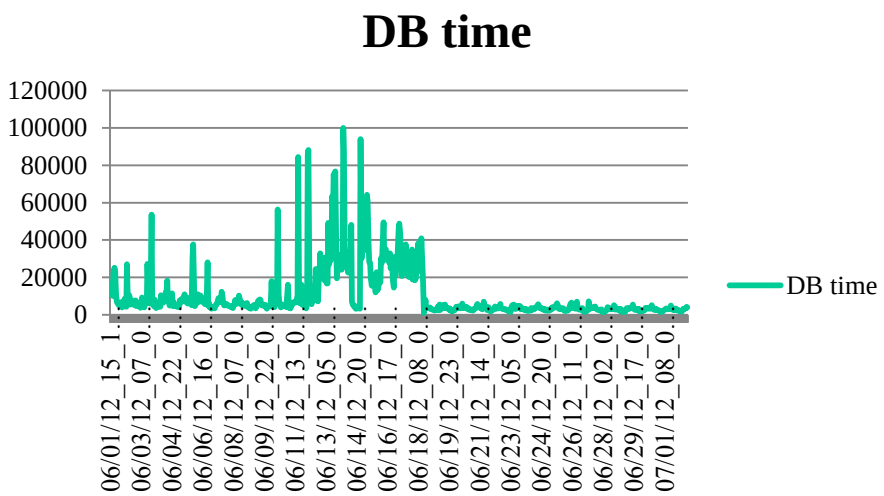
不绑定变量导致游标无法共享，DBA能做的

1. 要求开发修改程序使用绑定变量，使用绑定变量的写法并不比硬编码耗时，为了程序通用之类的说法，都是借口。
2. 短期内确实无法修改代码的话，可以用**Cursor_Sharing=FORCE**

Cursor_Sharing的优点：

1. 确实可以让大多数语句做到游标共享
2. 减少硬解析后，CPU使用率大幅下降
3. 减少解析类并发等待事件
4. 减少对Shared_Pool的无谓浪费

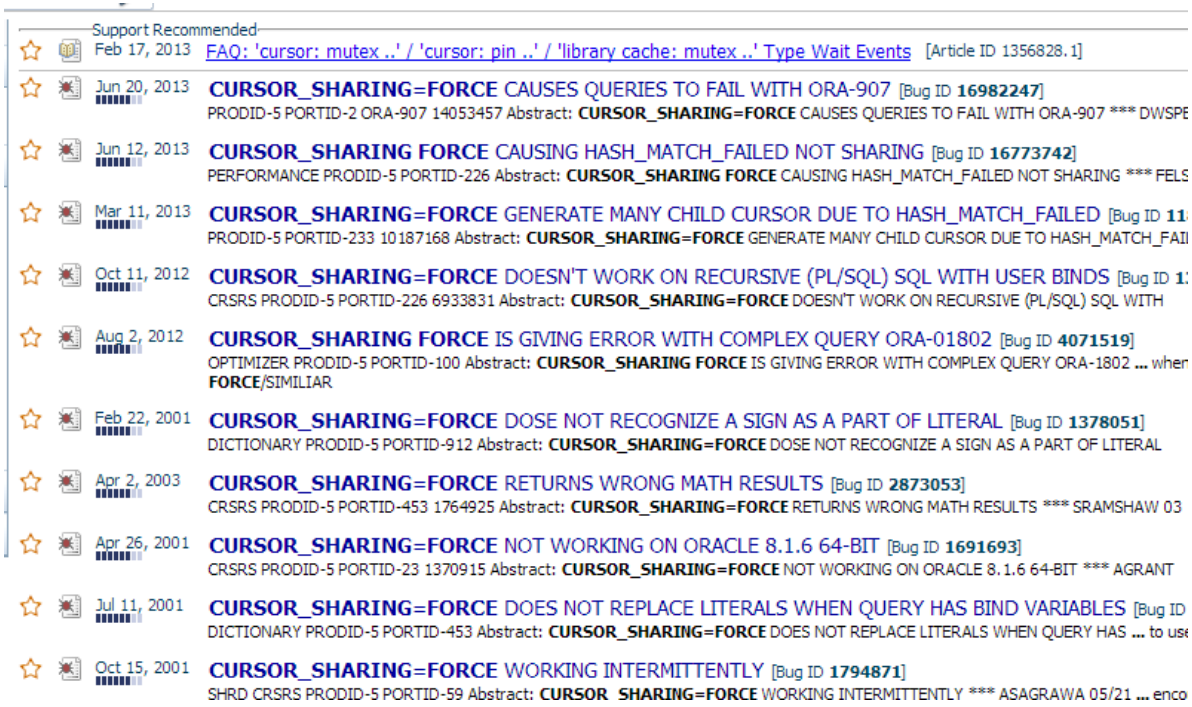
右图为一个优化实例效果



Cursor_sharing=Force的缺点

可能触发一些Bug，从而：

- 导致Version Count过多
- 引起ORA-600/7445等错误， 早期版本
- 导致SQL查询结果不准确 ， 早期版本
- 其性能总是不如原生态绑定变量的



Optimizer优化器- 物理优化

- 2种经典优化器：
  - Rule Based Optimizer RBO(10g以后废弃)
  - Cost Based Optimizer CBO基于成本的优化器

- 优化器的主要任务：
  - Access path analysis 访问路径分析 (评估使用 索引 还是全表扫描)
  - Join order and Join method 连接方式和连接顺序
  - Partition pruning 分区裁剪
  - 星型转换或和OR-expansion

- CBO优化器依赖于统计信息和约束
  - 列的Distinct 值， 行数，块数，直方图Histogram等等

RBO rule Based Optimizer

- RBO基于规则的优化器access paths优先级:
- RBO Path 1: Single Row by Rowid
- RBO Path 2: Single Row by Cluster Join
- RBO Path 3: Single Row by Hash Cluster Key with Unique or Primary Key
- RBO Path 4: Single Row by Unique or Primary Key
- RBO Path 5: Clustered Join
- RBO Path 6: Hash Cluster Key
- RBO Path 7: Indexed Cluster Key
- RBO Path 8: Composite Index
- RBO Path 9: Single-Column Indexes
- RBO Path 10: Bounded Range Search on Indexed Columns
- RBO Path 11: Unbounded Range Search on Indexed Columns
- RBO Path 12: Sort Merge Join
- RBO Path 13: MAX or MIN of Indexed Column
- RBO Path 14: ORDER BY on Indexed Column
- RBO Path 15: Full Table Scan

注意在不违反如上优先级的前提下，若有2个优化级一样的索引可用，则RBO会选择晚建的那个索引， 解决方法是重建你想要让RBO使用的那个索引，或者使用CBO…….. :lol:

在Oracle 10g以后虽然RBO (optimizer_mode=RULE)仍可用，但是不受官方的支持认可。

糟糕的RBO

select t3 from maclean where t1 = 9999 and t2 = 1;

```
Execution Plan
----------------------------------------------------------
Plan hash value: 2857504654

-----------------------------------------------
| Id  | Operation                    | Name    |
-----------------------------------------------
|   0 | SELECT STATEMENT             |         |
|*  1 |  TABLE ACCESS BY INDEX ROWID| MACLEAN |
|*  2 |   INDEX RANGE SCAN           | IND_T23 |
-----------------------------------------------

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

   1 - filter("T1"=9999)
   2 - access("T2"=1)

Note
-----
   - rule based optimizer used (consider using cbo)


Statistics
----------------------------------------------------------
        144  recursive calls
          0  db block gets
       1740  consistent gets
          0  physical reads
          0  redo size
        513  bytes sent via SQL*Net to client
        492  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          4  sorts (memory)
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 2854081138

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    14 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| MACLEAN |     1 |    14 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IND_T13 |     1 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - filter("T2"=1)
   2 - access("T1"=9999)


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          5  consistent gets
          0  physical reads
          0  redo size
        513  bytes sent via SQL*Net to client
        492  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
```

影响CBO的因素

- CBO的输出产物是 可执行的SQL PLAN

- CBO的输入物是,注意GIGO Garbage In Garbage Out：
  - 统计信息， 包括表的block数，行数，列的Distinct，索引的Cluster Factor等等
  - 优化器参数
  - HINT
  - OUTLINE 、SQL PROFILE、SPM、Cardinality Feedback等等

可能造成CBO给出不恰当执行计划的原因

- 没有统计信息，采用动态采样
- 没有统计信息，也没有动态采样
- 不准确的统计信息，包括不准确的数据分布
- 不合理的优化器参数
- 不合理的Hint、SQL PROFILE、SPM、ACS等等
- 建立成本模型所做的某些假设不合适
- 优化器算法本身有BUG

CBO的单位，有点诡异

- Cost成本的单位 为 single-block read time=sreadtim
- sreadtime = ioseektim + db_block_size/iotfrspeed
- mreadtim = ioseektim + db_file_multiblock_read_count * db_block_size / iotrfspeed
- #MRds = #Blks/MBRC

- Cost = (#SRds * sreadtim +
- #MRds * mreadtim +
- #CPUCycles/cpuspeed)/sreadtim

对于不恰当的CBO执行计划，DBA能做的(全能版)

按照推荐程度递减排序：

- 收集合理的统计信息
- 使用SQL PROFILE(10g新特性)
- 使用SPM 稳定执行计划(11g新特性)
- 给SQL加提示 HINT ： First_Rows、USE_NL、USE_HASH
- 使用 OUTLINE存储大纲(10g以后不推荐)
- 增加更合适的索引
- 使用并行技术
- 使用分区 、物化视图、ADG等技术
- 要求开发重新编写SQL
- 调整优化器参数： optimizer_index_cost_adj、 optimizer_secure_view_merging

对于不恰当的CBO执行计划，DBA能做的(不能改SQL版)

按照推荐程度递减排序：

- 收集合理的统计信息
- 使用SQL PROFILE(10g新特性)
- 使用SPM 稳定执行计划(11g新特性)
- 使用 OUTLINE存储大纲(10g以后不推荐)
- 增加更合适的索引
- 使用并行技术
- 使用分区 、物化视图、ADG等技术
- 调整优化器参数： optimizer_index_cost_adj、 optimizer_secure_view_merging

预备知识:CBO术语

NDV – number of distinct values

Card – cardinality

NULLS: Number of Nulls in Column

TYPE: Histogram Type

#BKTS: Histogram Buckets

UNCOMPBKTS: Histogram Uncompressed Buckets

ENDPTVALS: Histogram End Point Values

Freq 频率直方图

HtBal 高度平衡直方图

DEN: Column Density 密度

sel – selectivity 选择性

详见 **http://www.askmaclean.com/archives/cbo-terms.html**

统计信息对CBO的影响

```
SQL>  select t3
  2    from maclean
  3   where t1 = 9999
  4     and t2 = 1;

T3
------
sample


Execution Plan
----------------------------------------------------------
Plan hash value: 2854081138

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    12 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| MACLEAN |     1 |    12 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IND_T13 |     1 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - filter("T2"=1)
   2 - access("T1"=9999)


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          5  consistent gets
          0  physical reads
```

```
SQL> exec dbms_stats.set_table_stats(user,'MACLEAN', numblks=>0,numrows=>0);

PL/SQL procedure successfully completed.

SQL>  select t3
  2    from maclean
  3   where t1 = 9999
  4     and t2 = 1;

T3
------
sample


Execution Plan
----------------------------------------------------------
Plan hash value: 2568761675

-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |    12 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| MACLEAN |     1 |    12 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("T1"=9999 AND "T2"=1)


Statistics
----------------------------------------------------------
         42  recursive calls
          0  db block gets
         10  consistent gets
          0  physical reads
```

手动修改表的统计信息

CBO 如何计算全表扫描的成本

```
Using NOWORKLOAD Stats
  CPUSPEED: 714 millions instruction/sec
  IOTFRSPEED: 4096 bytes per millisecond (default is 4096)
  IOSEEKTIM: 10 milliseconds (default is 10)


SQL> show parameter db_file_multiblock_read_count

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_multiblock_read_count        integer     16
~
```

```
SQL> exec dbms_stats.set_table_stats(user,'MACLEAN', numblks=>10000);

SQL> select * from maclean;

-------------------------------------+-------------------------------------+
| Id  | Operation          | Name    | Rows  | Bytes | Cost  | Time       |
-------------------------------------+-------------------------------------+
| 0   | SELECT STATEMENT   |         |       |       |  2197 |            |
| 1   |  TABLE ACCESS FULL | MACLEAN |     1 |    12 |  2197 |  00:00:27  |
-------------------------------------+-------------------------------------+

#Mrds= 10000/16

mreadtim/sreadtim=

sreadtime = ioseektim + db_block_size/iotfrspeed=10+ 8192/4096=12
mreadtim =  ioseektim + db_file_multiblock_read_count * db_block_size / iotrfspeed= 10+ 16*8192/4096= 42

#MRds = #Blks/MBRC= 10000/16
I/O Cost = 1 + CEIL(#MRds * (mreadtim/sreadtim)) = 1+ CEIL ( 10000/16*42/12)=  2189

CPU Cost = ROUND(#CPUCycles/cpuspeed/1000/sreadtim)=ROUND(71214400/714/1000/12)= 8

FTS Cost = I/O Cost + CPU Cost= 2189+8=2197
~
```

CBO 如何计算Index Range Scan的成本

```
  Index Stats::
  Index: IND_T13  Col#: 1 3
    LVLS: 2  #LB: 961  #DK: 300001  LB/K: 1.00  DB/K: 1.00  CLUF: 828.00
  Access Path: index (RangeScan)
    Index: IND_T13
    resc_io: 15.00  resc_cpu: 988092
    ix_sel: 0.0066733  ix_sel_with_filters: 0.0066733
    Cost: 15.06  Resp: 15.06  Degree: 1



 select t3
  from maclean
 where t1 between 9999 and 11999
   and t2 = 1;

 ---------------------------------------------+-----------------------------------+
| Id  | Operation                    | Name    | Rows  | Bytes | Cost  | Time      |
---------------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT             |         |       |       |    15 |           |
| 1   |  TABLE ACCESS BY INDEX ROWID | MACLEAN |     1 |    12 |    15 |  00:00:01 |
| 2   |   INDEX RANGE SCAN           | IND_T13 |  2002 |       |     9 |  00:00:01 |
---------------------------------------------+-----------------------------------+

I/O Cost = Index Access I/O Cost + Table Access I/O Cost
Index Access I/O Cost = LVLS + CEIL(#LB * ix_sel)     =  2 + CEIL(961*0.0066733)=9
Table Access I/O Cost = CEIL(CLUF * ix_sel_with_filters)= CEIL( 828*0.0066733)= 6
CPU Cost = ROUND(#CPUCycles/cpuspeed/1000/sreadtim)= ROUND(988092/714/10000/12)=0

CBO Cost=I/O Cost+ CPU COST= 9+6+0= 15
```

System Statistics

- 2种system statistics方式，区别在于收集system statistics的时段：
  - Workload Statistics - 有工作负载，包括IO delay、Latch和Task Switching因素
    - 收集方式： dbms_stats.gather_system_stats('start') =》 dbms_stats.gather_system_stats('stop')
    - dbms_stats.gather_system_stats('interval', interval=>N)
  - Noworkload Statistics - 无工作负载，仅包括IO Delay
    - 收集方式：dbms_stats.gather_system_stats()

默认采用Noworkload Statistics，且会在实例启动时自动初始化为默认值：

ioseektim = 10ms

iotrfspeed = 4096 bytes/ms

cpuspeednw 由smon每10分钟收集一次

当有Workload Statistics的情况下，默认采用Workload Statistics

System Statistics存放在aux_stats$基表上

```
SQL> exec dbms_stats.gather_system_stats();

PL/SQL procedure successfully completed.

SQL> select * from aux_stats$;

SNAME                PNAME                PVAL1 PVAL2
-------------------- -------------------- ---------- --------------------------------------------------
SYSSTATS_INFO        STATUS                          COMPLETED
SYSSTATS_INFO        DSTART                          06-26-2013 14:55
SYSSTATS_INFO        DSTOP                           06-26-2013 14:55
SYSSTATS_INFO        FLAGS                        1
SYSSTATS_MAIN        CPUSPEEDNW                1612
SYSSTATS_MAIN        IOSEEKTIM                   14
SYSSTATS_MAIN        IOTFRSPEED                4096
```

Exadata的system statistics

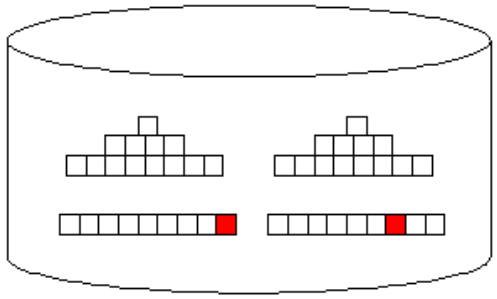
```
SQL> select * from aux_stats$;

SNAME                PNAME                PVAL1 PVAL2
-------------------- -------------------- ---------- ----------------------------------------
SYSSTATS_INFO        STATUS                          COMPLETED
SYSSTATS_INFO        DSTART                          06-26-2013 11:00
SYSSTATS_INFO        DSTOP                           06-26-2013 11:00
SYSSTATS_INFO        FLAGS                        1
SYSSTATS_MAIN        CPUSPEEDNW                2843
SYSSTATS_MAIN        IOSEEKTIM                    3
SYSSTATS_MAIN        IOTFRSPEED                144319
```

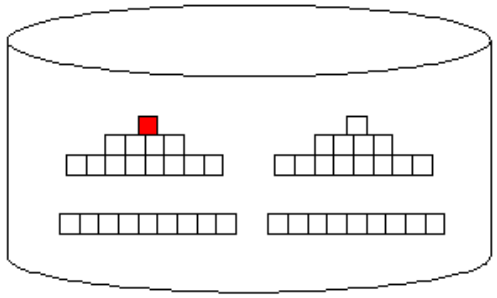
IO变强的结果是IO Cost下降！！

优化器模式

- **Optimizer_mode= ALL_ROWS**
  - 降低返回全部结果集所消耗的资源
  - 更适合DSS和DW应用程序
  - 更青睐于于全表扫描



- **Optimizer_mode= FIRST_ROWS**
  - 降低返回前N行的响应时间
  - 更青睐于索引

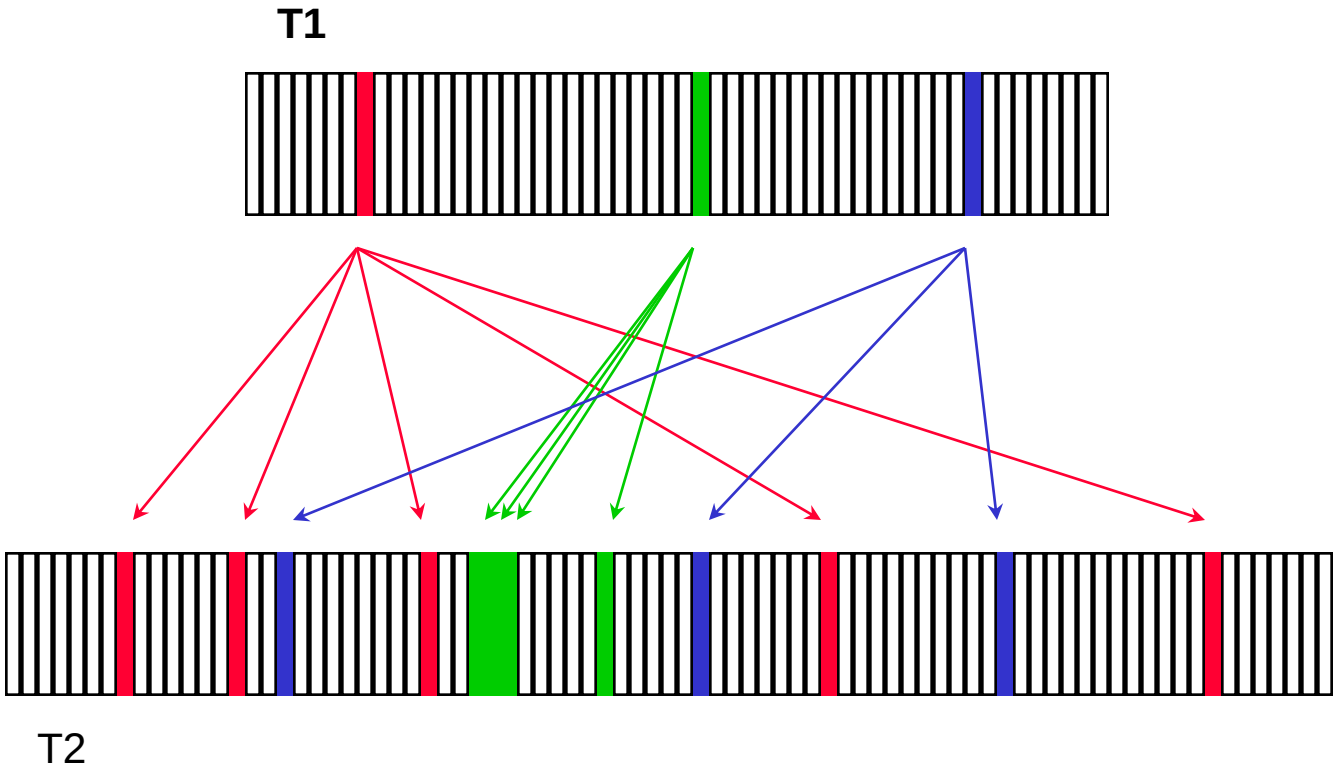


几种**Join Method**

- Nested loop join (use_nl hint)
- Hash join (use_hash hint)
- Sort-merge join (use_merge hint)
- STAR join (star hint)

Nested Loop Join

- 适合小的结果集和有较好的索引的大数据集Join,并返回较小结果集的场景



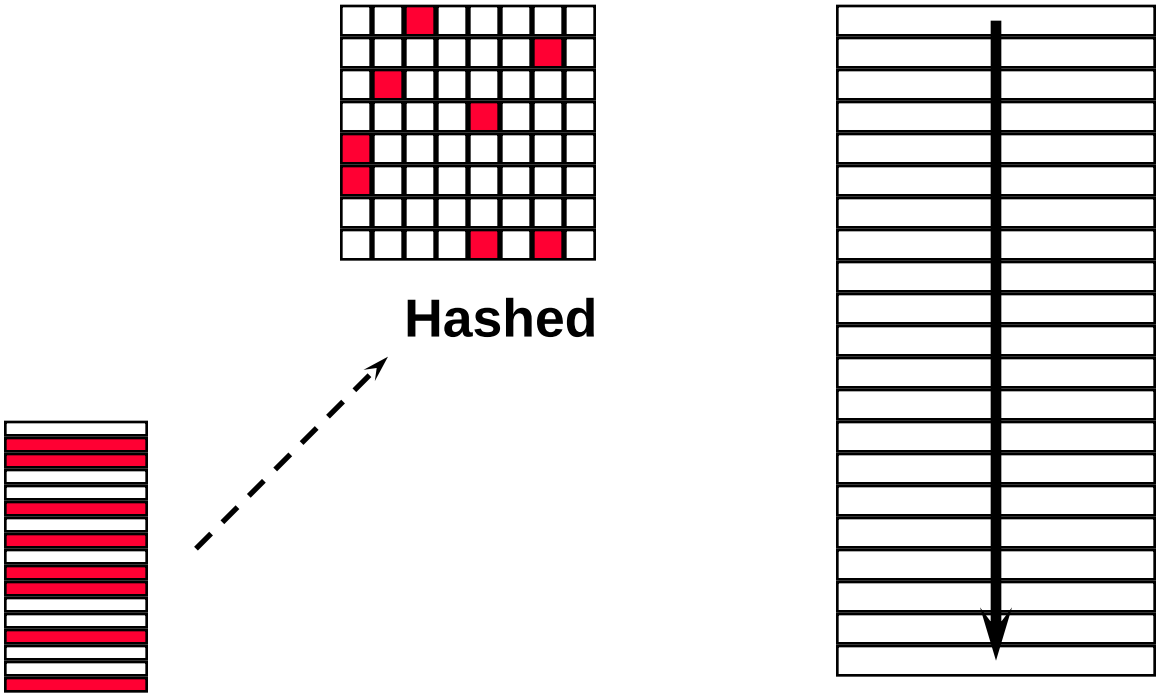
强制使用nested loop

```
select
      /*+ ordered use_nl(t1) index(t1) */
      t2.n1, t1.n2
from        t2,t1
where t1.n2 = 45
and         t2.n1 = t1.n1;
```

```
NESTED LOOPS (Cost=45 Card=225)
  TABLE ACCESS (FULL) OF T2 (Cost=15, Card=15)
  TABLE ACCESS (BY ROWID) OF T1(Cost=2,Card=3000)
    INDEX(RANGE SCAN) OF T_I1(NON-UNIQUE)(Cost=1)
```

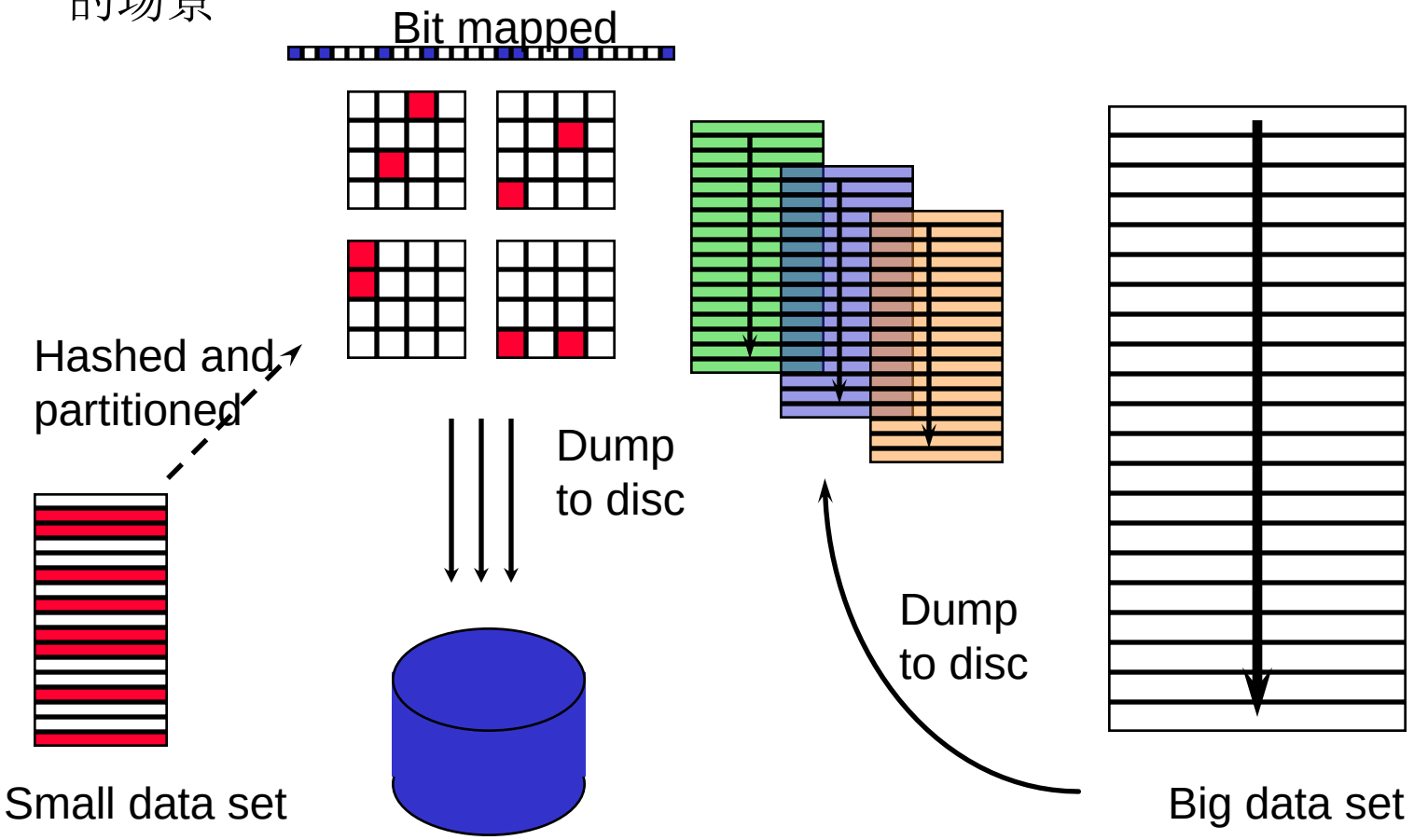
Hash Join

适合小的结果集和大的结果集Join，并返回大结果集的场景 equijoin 等式链接



Hash Join

适合小的结果集和大的结果集Join，并返回大结果集的场景



Sort-Merge Join

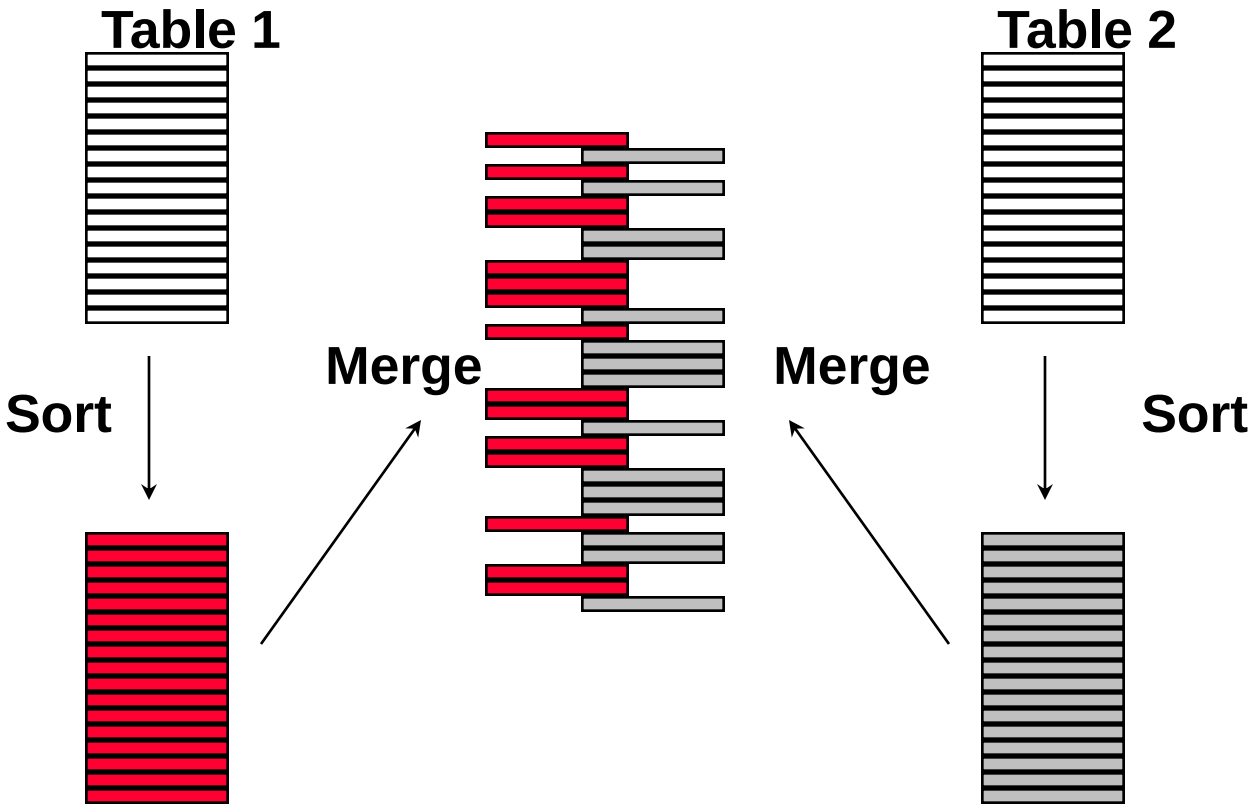
- 适合没有合适连接谓词或者没有合适的索引的情况下，当数据量很大时要比Nested Loop好 equijoin 等式链接

```
select
       count(t1.v1)       ct_v1,
       count(t2.v2)       ct_v2
from   big1   t1,    big2   t2
where t2.n2 = t1.n1;
```

```
SELECT STATEMENT (choose) Cost (963)
SORT (aggregate)
    MERGE JOIN      Cost (963 = 174 + 789)
      SORT (join)   Cost (174)
        TABLE ACCESS (analyzed) T1 (full)   Cost (23)
      SORT (join)   Cost (789)
        TABLE ACCESS (analyzed) T2 (full)   Cost (115)
```

Sort-Merge Join

- 适合没有合适连接谓词或者没有合适的索引的情况下，当数据量很大时要比Nested Loop好



三种Join 的成本模型

- Nested loops:
- Cost(outer) + Cost(inner)*cardinality(outer)
- Sort merge:
- Cost(outer) + Cost(inner) + Sort(outer) + Sort(inner)
- Hash:
- Cost(outer) + Cost(inner) + Build(outer) + Probe(inner)

统计信息管理 - 基础

- 优化器相关的有哪些统计信息？
  - 表 - 行数， 块数， 平均行长
  - 列 - Distinct 值， 最小，最大值， 直方图Hisotgram
  - 索引 - Level， Leaf block叶子块数， #distinct key
- 均存放在数据字典中；通过library/row cache去访问
- CBO依赖于统计信息
  - 使用统计信息来评估执行计划的成本
  - 选择最低成本的执行计划
- 理论上来说 统计信息总是最新的好
- 10g以后推荐使用dbms_stats管理统计信息

DBMS_STATS.AUTO_SAMPLE_SIZE

默认的自动采样大小参数
DBMS_STATS.AUTO_SAMPLE_SIZE

AUTO_SAMPLE_SIZE时会优先采样5500行,dbms_stats包内部算法会评估采样的5500行数据是否有效，若无效则会再次采样55000行数据，依此类推。

为什么是5500这个数字？

5500这个数字是经过数学论证的。 可以90%以上保证采样获得的直方图Histogram的桶buckets中数据分布式均匀。

Number of distinct values (NDV)

在没有Histogram或非BIND PEEK的情况下NDV的作用很大：

A4Nulls = (Orig_Card - NNulls)/Orig_Card

单表查询 Sel = （1/NDV) * A4Nulls

Comp_Card = Orig_Card * Sel

演示NDV对基数计算的作用

Histogram

参见《**拨开Oracle CBO优化器迷雾,探究Histogram直方图之秘**》

案例：为什么CBO不使用索引？

create table maclean1 as select * from dba_objects;

update maclean1 set status='INVALID' where owner='MACLEAN';

commit;

create index ind_maclean1 on maclean1(status);

exec dbms_stats.gather_table_stats('SYS','MACLEAN1',cascade=>true);

explain plan for select * from maclean1 where status='INVALID';

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 11320 |  1028K|    85   (0)| 00:00:02 |
|*  1 |  TABLE ACCESS FULL| MACLEAN1 | 11320 |  1028K|    85   (0)| 00:00:02 |
-------------------------------------------------------------------------------

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

   1 - filter("STATUS"='INVALID')
```

案例：为什么CBO不使用索引？

Access path analysis for MACLEAN1

****

SINGLE TABLE ACCESS PATH

Single Table Cardinality Estimation for MACLEAN1[MACLEAN1]

Column (#10): STATUS(

AvgLen: 7 NDV: 2 Nulls: 0 Density: 0.500000

Table: MACLEAN1 Alias: MACLEAN1

可以从以上**10053**中看到因为没有直方图存在，所以这里的**Density = 0.5** 是从 **1/ NDV** 算得的

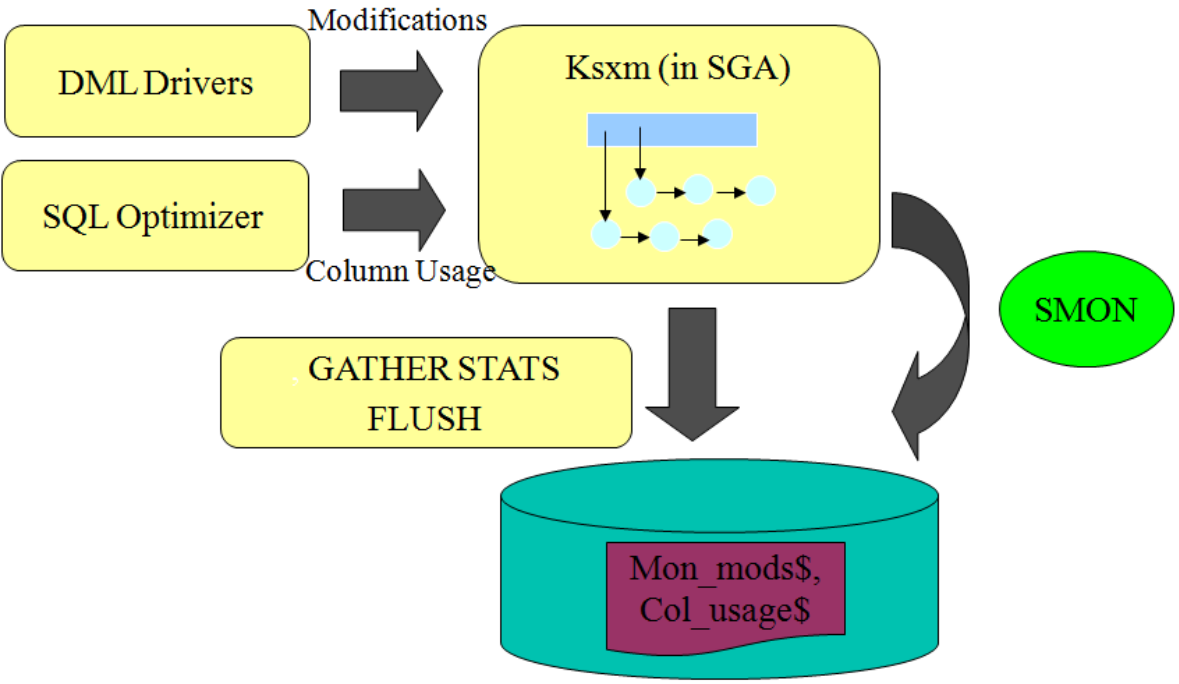
也就意味着粗糙的统计信息显示**STATUS='INVALID"**的数据行占总行数的一半，

所以优化器选择做全表扫描是有道理的

Histogram直方图的历史

版本8~9i默认不收集直方图Histogram，需要手动指定method_opt size (默认size=1 仅收集最小/大值，distinct值等信息)

从版本10g开始Oracle会自动收集Histogram，Histogram是否收集取决于col_usage$中记录的关于该列用作SQL中谓词条件的信息和数据分布情况



SMON定期将shared pool中的谓词使用状况刷新到col_usage$表中

例如:

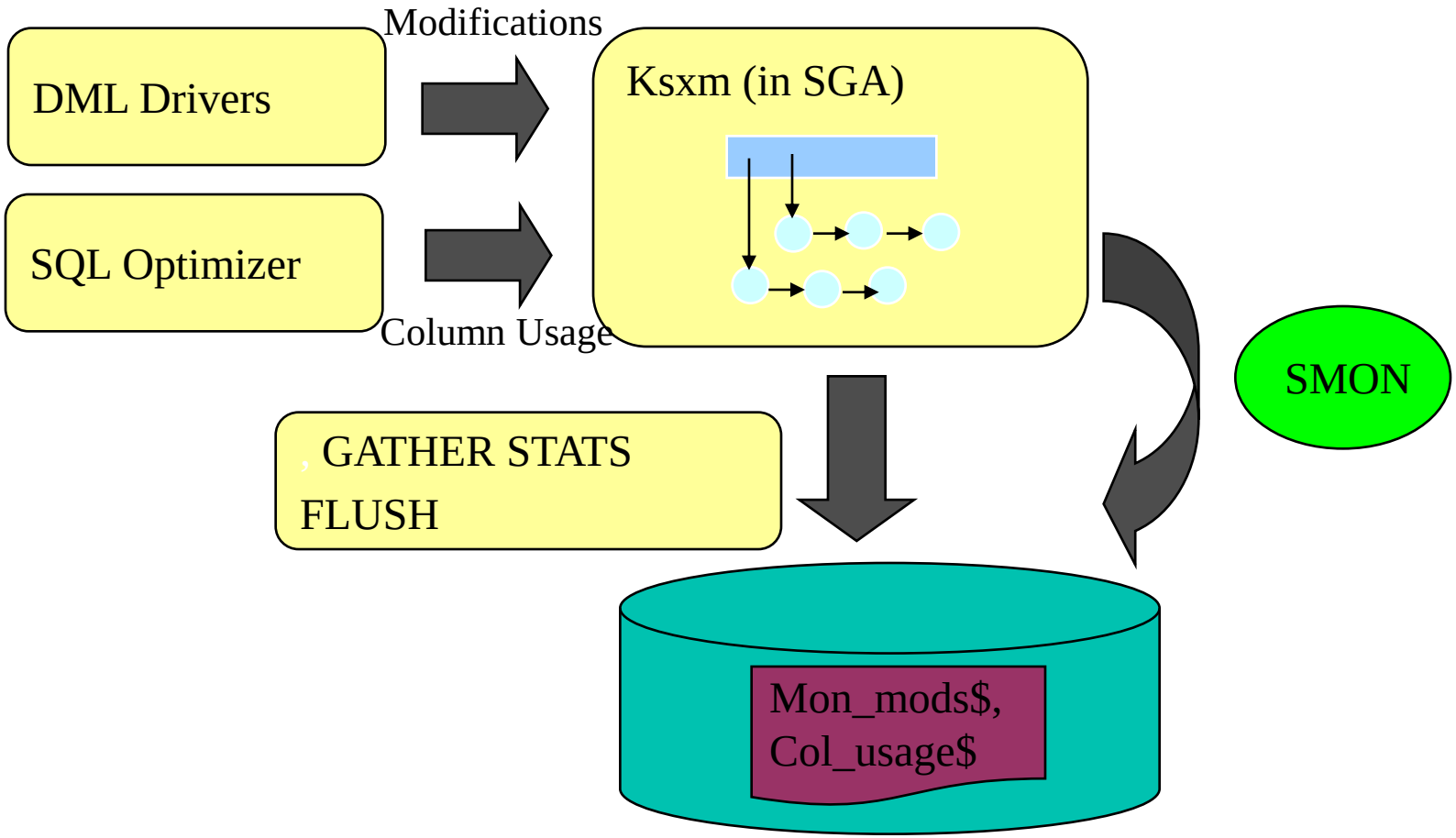
Select * from tab where colA=1;

则记录为对colA充当EQUALITY_PREDS ➔ equality predicates等式谓词

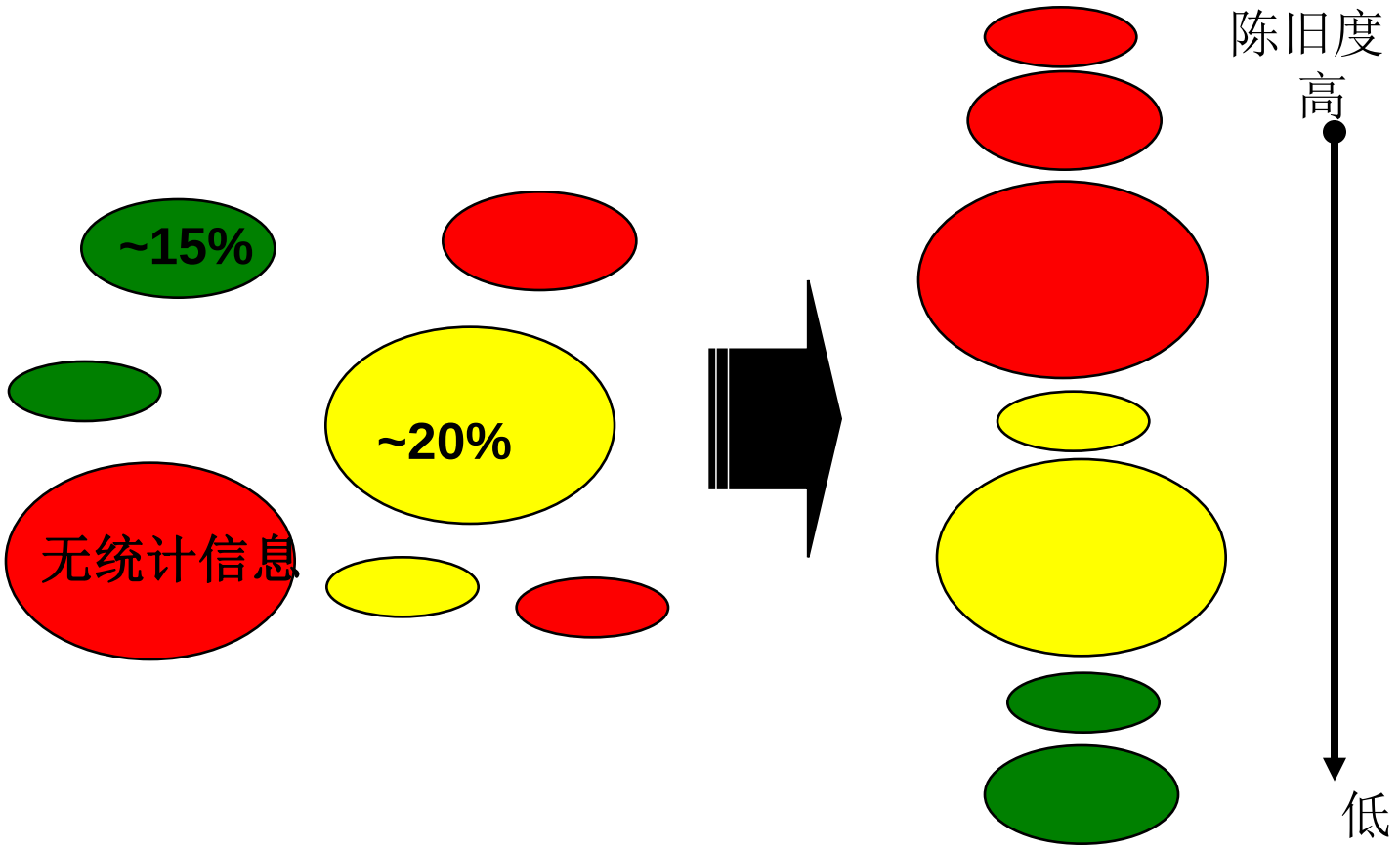
自动统计信息收集作业

- 预配置的自动任务，无需额外配置自动运行
- 启动后仅为那些没有或陈旧统计信息的对象收集信息
  - 基于DML 修改监控
    - 跟踪行的插入，删除，更新和Truncate
  - 为数据字典和用户对象收集统计信息
- 仅在以下情况下收集直方图
  - 基于列使用监控column usage monitoring
    - 跟踪列及其谓词类型(=,>,<)
  - 倾斜的数据分布
- 自动滚动游标失效
  - 不在第一时间使依赖的cursor游标失效
  - 避免大量解析毛刺
- 10g中使用Scheduler框架，11g使用Autotask框架
- 在维护窗口打开

监控机制


Modifications
DML Drivers
SQL Optimizer
Column Usage
Ksxm (in SGA)
SMON
GATHER STATS
FLUSH
Mon_mods$,
Col_usage$

统计信息收集作业 - 收集优先度



锁定统计信息

- 避免统计信息以下原因而改变：
  - 自动统计信息收集作业
  - 手动统计信息管理操作

- 对易变的表使用：
  - 删除并锁住其统计信息，以便使用动态采样
- 用以冻结执行计划变化

- 接口：
  - lock_*_stats, unlock_*_stats
  - 可以指定FORCE=TRUE来强制更新统计信息

Restore Statistics

- 老版本的统计信息自动保存在数据字典中
  - 这些数据字典表在SYSAUX表空间上
- 允许用户将统计信息还原到过去的某一个时间点
- 用以处理由于收集统计信息所造成的不当执行计划
- 接口：
  - restore_table_stats, restore_schema_stats
- 历史清理
  - 基于保留期限的自动清理 默认保留31天
    - 使用AWR清理框架
  - 可以手动使用purge_stats存储过程

Pending statistics

- 默认情况下 统计信息是一经收集立即发布的，这可能造成执行计划不可预见的变化

- 在11g中提供了新功能， 可以将统计信息存为pending
  - set_global_prefs('PUBLISH', 'FALSE')
  - Gather_*_stats

- 使用以下参数来使用pending statistics
- optimizer_use_pending_statistics => true (default false)

- 一旦测试完整再发布：
  - publish_pending_stats

准确的统计信息，准确的基数评估

```
exec dbms_stats.gather_table_stats('','MACLEAN_ROW1', estimate_percent=>100);

exec dbms_stats.gather_table_stats('','MACLEAN_ROW2', estimate_percent=>100);


select /*+ gather_plan_statistics */ avg(a.blocks), a.owner from maclean_row1 a, maclean_row2 b
where a.table_name=b.table_name and a.AVG_ROW_LEN>5
group by a.owner;

select * from TABLE(dbms_xplan.display_cursor(NULL,NULL,'ALLSTATS LAST'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		18	00:00:00.01	260			
1	HASH GROUP BY		1	22	18	00:00:00.01	260	1004K	1004K	1217K (0)
* 2	HASH JOIN		1	1777	1490	00:00:00.01	260	1199K	1199K	1312K (0)
* 3	TABLE ACCESS FULL	MACLEAN_ROW1	1	799	799	00:00:00.01	100			
4	TABLE ACCESS FULL	MACLEAN_ROW2	1	4515	4515	00:00:00.01	160			

如何确定基数评估不准确？

使用gather_plan_statistics获取实际返回行数

```
create table maclean_row1 as select * from dba_tables;


create table maclean_row2 as select * from dba_indexes;


exec dbms_stats.gather_table_stats('','MACLEAN_ROW1', estimate_percent=>0.1);

exec dbms_stats.gather_table_stats('','MACLEAN_ROW2', estimate_percent=>0.1);



select /*+ gather_plan_statistics */ avg(a.blocks), a.owner from maclean_row1 a, maclean_row2 b
where a.table_name=b.table_name and a.AVG_ROW_LEN>5
group by a.owner;

select * from TABLE(dbms_xplan.display_cursor(NULL,NULL,'ALLSTATS LAST'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		18	00:00:00.01	260			
1	HASH GROUP BY		1	22	18	00:00:00.01	260	1004K	1004K	1232K (0)
* 2	HASH JOIN		1	4735	1490	00:00:00.01	260	1199K	1199K	1309K (0)
* 3	TABLE ACCESS FULL	MACLEAN_ROW1	1	2482	799	00:00:00.01	100			
4	TABLE ACCESS FULL	MACLEAN_ROW2	1	4515	4515	00:00:00.01	160			

使用动态采样如何？

```
select /*+ gather_plan_statistics dynamic_sampling(b 10) dynamic_sampling_est_cdn(b) dynamic_sampling(a 10) dynamic_sampling_est_cdn(a)*/
 avg(a.blocks), a.owner
  from maclean_row1 a, maclean_row2 b
 where a.table_name = b.table_name
   and a.AVG_ROW_LEN > 5
 group by a.owner;

select * from TABLE(dbms_xplan.display_cursor(NULL,NULL,'ALLSTATS LAST'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		18	00:00:00.01	260			
1	HASH GROUP BY		1	22	18	00:00:00.01	260	1004K	1004K	1217K (0)
* 2	HASH JOIN		1	1777	1490	00:00:00.01	260	1199K	1199K	1312K (0)
* 3	TABLE ACCESS FULL	MACLEAN_ROW1	1	799	799	00:00:00.01	100			
4	TABLE ACCESS FULL	MACLEAN_ROW2	1	4515	4515	00:00:00.01	160			

动态采样可以作为验证更精确统计信息是否能够有效改善执行计划的试金石！

自动SQL调优 Automatic SQL Tuning

- 自动SQL调优 **Automatic SQL Tuning将整个SQL调优的过程自动化了**
- 优化模式：
  - 普通模式
  - 优化模式或Automatic Tuning Optimizer (ATO)
- 这对高负载SQL使用sql tuning mode

ATO 生成SQL Profile

- 语句运行的统计信息是对优化器重要的启发
- ATO验证如下语句的统计信息：
  - 谓词选择性
  - 优化器设置(FIRST_ROWS和ALL_ROWS比较)
- Automatic Tuning Optimizer (ATO)使用：
  - 动态采样
  - 部分执行SQL语句
  - 该语句之前的历史执行信息
- ATO将可能生成一个profile，例如：
  - execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_12236',task_owner => 'SYS', replace => TRUE);

SQL Profile

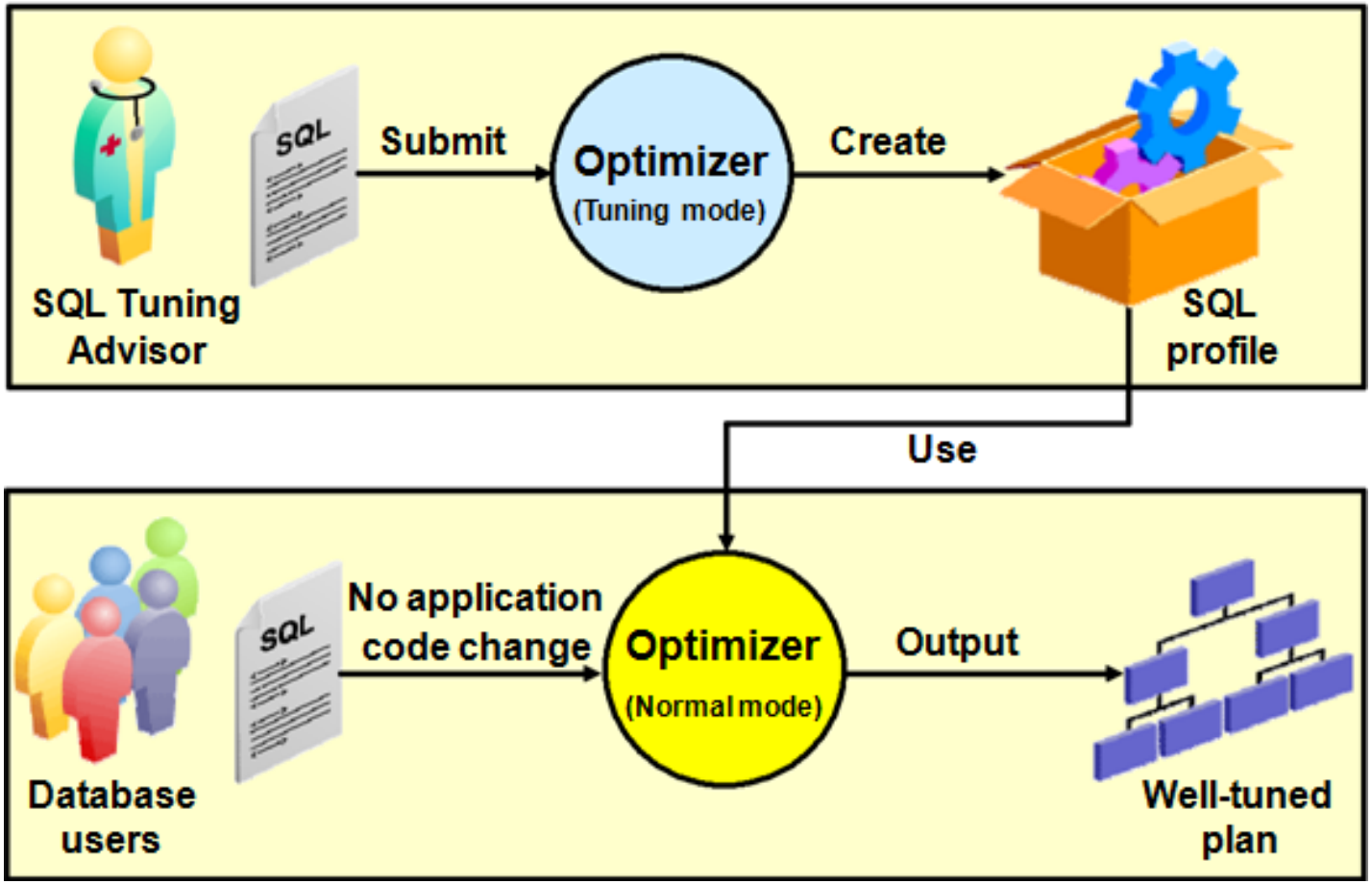
- 版本10g中引入，今后会持续发展并彻底替代outline的框架

- SQL Profile是数据字典中信息集合，用来让查询优化器创建最佳的执行计划

- SQL Profile中包含了在自动SQL调优过程中获得的对糟糕的优化器评估的纠正，这些信息有助于改善优化器的基数和选择性评估，以便让优化器获得更佳的执行计划。

- SQL Profile不会冻结执行计划，这一点不像outline,当表或索引增长，执行计划仍可能变化

SQL Profile生成流程



SQL Profile一个使用实例

```
create table profile_test tablespace users as select * from dba_objects;
create index ix_objd on profile_test(object_id);
exec dbms_stats.gather_table_stats('','PROFILE_TEST');
set autotrace traceonly;
select /*+ FULL( profile_test) */ * from profile_test where object_id=5060;

@?/rdbms/admin/sqltrpt
```

SQL Profile一个使用实例

```
Schema Name   : SYS
Container Name: CDB$ROOT
SQL ID        : f3v7dxj4bggvq
SQL Text      :  select /*+ FULL( profile_test) */ * from profile_test where
                 object_id=5060

-------------------------------------------------------------------------------
FINDINGS SECTION (1 finding)
-------------------------------------------------------------------------------

1- SQL Profile Finding (see explain plans section below)
--------------------------------------------------------
  A potentially better execution plan was found for this statement.

  Recommendation (estimated benefit: 99.79%)
  ------------------------------------------
  - Consider accepting the recommended SQL profile.
    execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_226',
            task_owner => 'SYS', replace => TRUE);

                                  Original Plan  With SQL Profile  % Improved
                                  -------------  ----------------  ----------
  Completion Status:                   COMPLETE          COMPLETE
  Elapsed Time (s):                    .005407          .000034      99.37 %
  CPU Time (s):                        .004599                0        100 %
  User I/O Time (s):                         0                0
  Buffer Gets:                            1470                3      99.79 %
  Physical Read Requests:                    0                0
  Physical Write Requests:                   0                0
  Physical Read Bytes:                       0                0
  Physical Write Bytes:                      0                0
  Rows Processed:                            1                1
  Fetches:                                   1                1
  Executions:                                1                1
```

SQL Profile一个使用实例

```
execute dbms_sqltune.accept_sql_profile(task_name =>
'TASK_226',task_owner => 'SYS', replace =>
TRUE,category=>'MACLEAN_TEST');
```

可以使用category测试SQL PROFILE的性能

```
alter session set sqltune_category= 'MACLEAN_TEST;
set autotrace on;
select /*+ FULL( profile_test) */ * from profile_test where object_id=5060;
```

```
alter session set sqltune_category=DEFAULT;
```

再次与运行，当sqltune_category=DEFAULT时Profile不生效

讨论帖地址

http://t.askmaclean.com/thread-2702-1-1.html