DTCC2013

All China Oracle User Group
中国 Oracle 用户组

YUNHE

YUNHE ENMO (BEIJING) TECHNOLOGY CO.,LTD

运用之妙 存乎一心
- Oracle优化器案例与原理分析

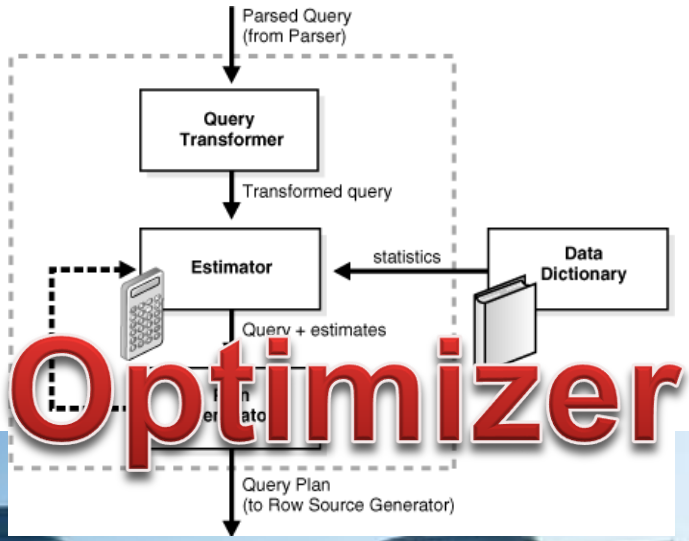
盖国强（Eygle）
电话:13911812803
邮件:eygle@enmotech.com
微博:weibo.com/eygle

Who am I

- 盖国强　云和恩墨（北京）信息技术有限公司　创始人
  - 盖国强是国内第一个Oracle ACE及ACE总监
  - 有超过10年的Oracle从业经验

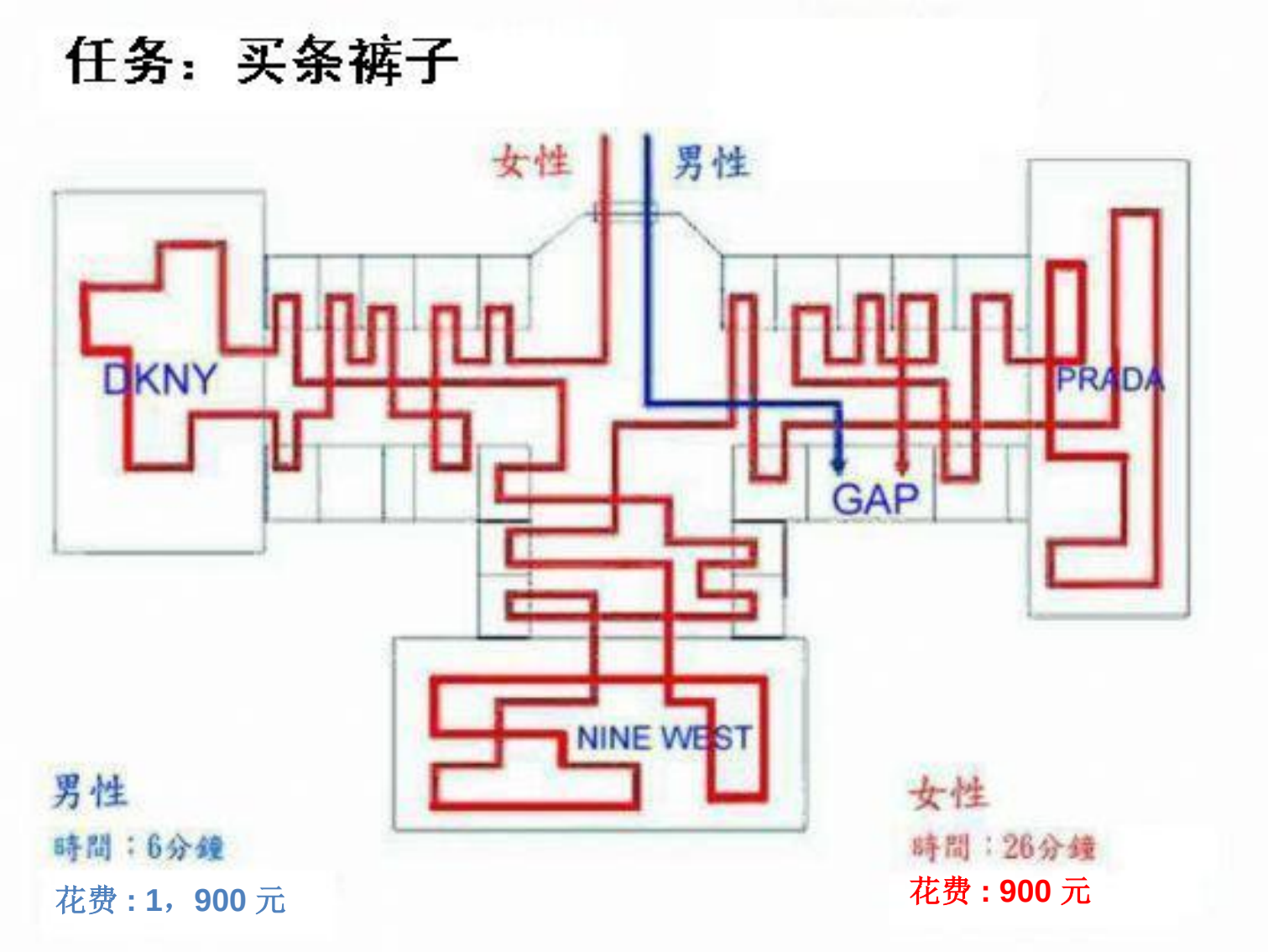
    至今仍然奋战在技术前线
  - 国内最大数据库技术论坛ITPUB的主要发起人之一，致力于技术分享与传播，截至2011年已经出版了10本技术书籍　自2010年开始，主编出版《Oracle DBA手记》系列书
  - 2010年，他和张乐奕共同创建了旨在开展技术交流的中国Oracle用户组（ACOUG - All China Oracle User Group）并开展了持续的公益活动

Oracle 数据库的发动机

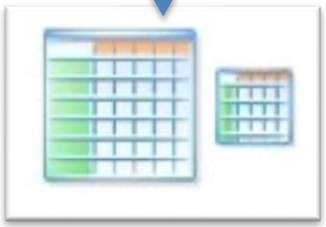


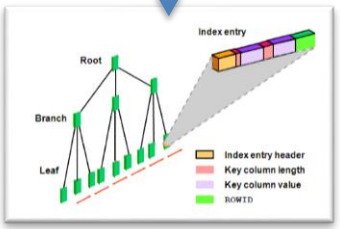
Optimizer

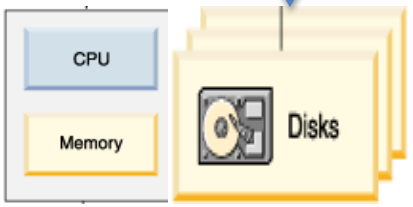
基于成本的优化器 - CBO



Oracle 的统计信息







- 块数
- 行数
- 平均行长
- 叶块数
- 层级
- Clustering factor
- NDV
- NNV
- Histogram
- I/O性能和效率
- CPU性能和效率

优化器算法 – 成败之所系


```
Elapsed:  00: 00: 12.17

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=HINT: RULE
   1    0   NESTED LOOPS
   2    1     NESTED LOOPS
   3    2       NESTED LOOPS
   4    3         NESTED LOOPS
   5    4           NESTED LOOPS
   6    5             NESTED LOOPS
   7  Elapsed:  00: 29: 1785.47
   8
   9  Execution Plan
  10  ----------------------------------------------------------
  11     0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2057 Card=1 Bytes=288)
  12     1    0   NESTED LOOPS (Cost=2057 Card=1 Bytes=288)
  13     2    1     NESTED LOOPS (Cost=2056 Card=1 Bytes=256)
  14     3    2       NESTED LOOPS (Cost=2054 Card=1 Bytes=219)
  15     4    3         NESTED LOOPS (Cost=2053 Card=1 Bytes=178)
  16     5    4           NESTED LOOPS (Cost=2009 Card=1 Bytes=131)
  17     6    5             MERGE JOIN (Cost=2008 Card=1 Bytes=100)
  18     7    6               SORT (JOIN) (Cost=950 Card=36412 Bytes=1747776)
  19     8    7                 TABLE ACCESS (FULL) OF 'SP_CHK_SUB' (Cost=59 Card=36412 Bytes=1747776)
         9    6               SORT (JOIN) (Cost=1058 Card=36730 Bytes=1909960)
        10    9                 TABLE ACCESS (FULL) OF 'SP_RECEIVE_SUB' (Cost=89 Card=36730 Bytes=1909960)
        11    5             TABLE ACCESS (BY INDEX ROWID) OF 'SP_CHK' (Cost=1 Card=3870 Bytes=119970)
        12   11               INDEX (UNIQUE SCAN) OF 'PK_SP_CHK' (UNIQUE)
        13    4           TABLE ACCESS (FULL) OF 'SP_TRANS' (Cost=44 Card=1717 Bytes=80699)
        14    3         TABLE ACCESS (BY INDEX ROWID) OF 'SP_RECEIVE' (Cost=1 Card=7816 Bytes=320456)
        15   14           INDEX (UNIQUE SCAN) OF 'PK_SP_RECEIVE' (UNIQUE)
        16    2       TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB' (Cost=2 Card=136371 Bytes=5045727)
        17   16         INDEX (UNIQUE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE) (Cost=1 Card=136371)
        18    1     TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM' (Cost=1 Card=29763 Bytes=952416)
        19   18       INDEX (UNIQUE SCAN) OF 'SYS_C0012193' (UNIQUE)
```

优化器算法 – 润物细无声

- 最重要的变化

```
_optimizer_adaptive_cursor_sharing  = true
_replace_virtual_columns            = true
parallel_min_time_threshold         = 10
iot_internal_cursor                 = 0
_pivot_implementation_method        = choose
_optimizer_reuse_cost_annotations   = true
_px_partition_scan_threshold        = 64
_optimizer_unnest_all_subqueries    = true
_suppress_scn_chk_for_cqn           = nosuppress_1466
dst_upgrade_insert_conv             = true
_optimizer_join_factorization       = true
_direct_path_insert_features        = 0
_parallel_scalability               = 50
cell_offload_plan_display           = AUTO
_optimizer_use_feedback             = true
_optimizer_distinct_placement       = true
_query_mmvrewrite_maxdmaps          = 10
_optimizer_multi_level_push_pred    = true
_query_mmvrewrite_maxcmaps          = 20
_bloom_predicate_enabled            = true
_force_tmp_segment_loads            = false
```

优化器算法 – 成本计算

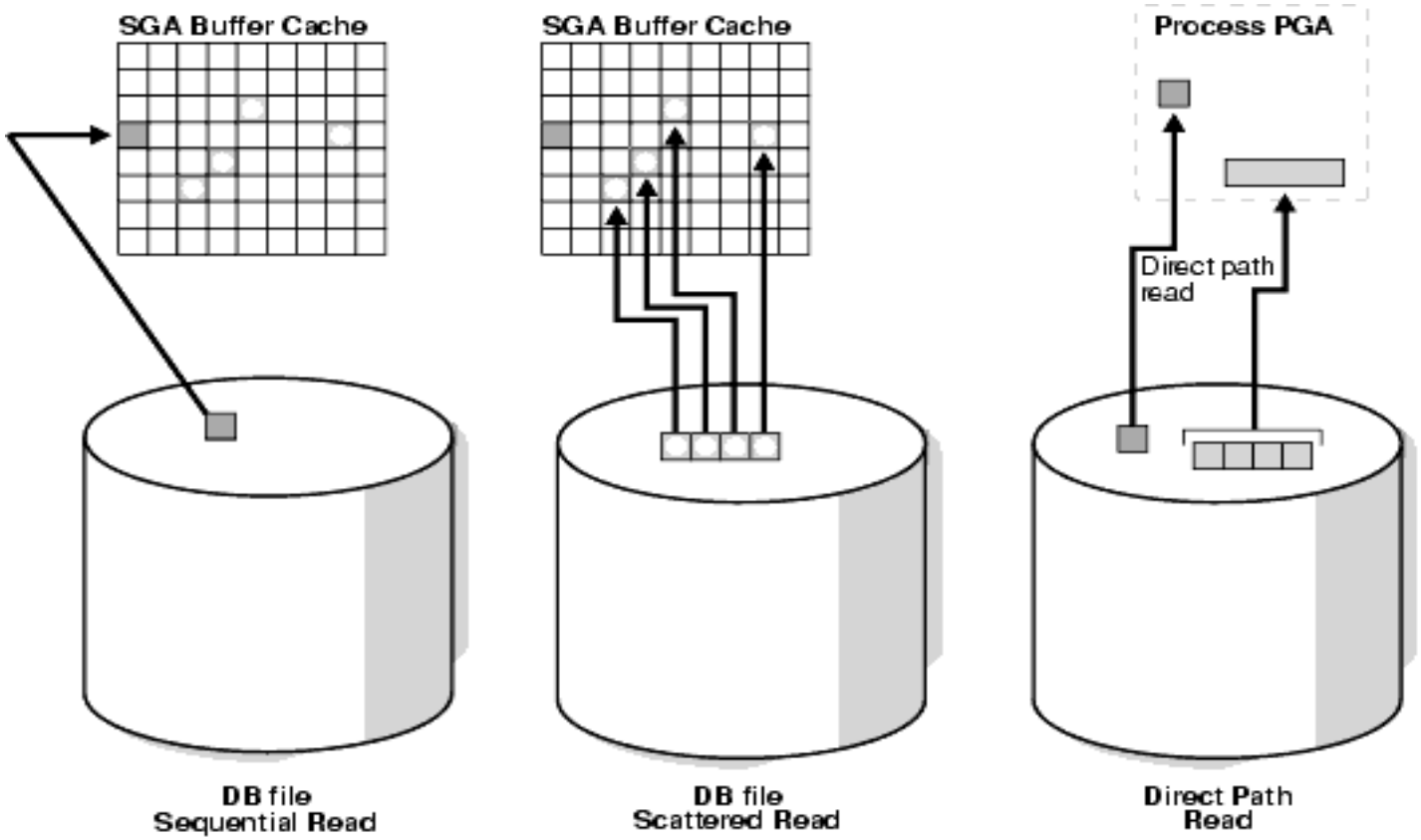
- 成本计算
  - 以单块读成本为单位进行转换
  - 量化所有操作的代价

Cost（成本） = IOCOST + CPUCOST = # Srds

多次读的量度：读次数 * 单次读时间

优化器算法 – 成本计算公式

- 成本计算公式的细化
  - 单块读 与 多块读



最基本的计算 – 索引 vs 全表

- 索引优化
  - 一直是数据库优化的核心手段

```
*******************************
SYSTEM STATISTICS INFORMATION
*******************************
  Using NOWORKLOAD Stats
  CPUSPEEDNW: 704 millions instructions/sec (default is 100)
  IOTFRSPEED: 4096 bytes per millisecond (default is 4096)
  IOSEEKTIM:  10 mil
```

```
Access path analysis for
**************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinalit
  Table: YUNHE  Alias: YU
    Card: Original: 15094
  Access Path: TableScan
    Cost:  506.61  Resp:
      Cost_io: 502.00  Co
      Resp_io: 502.00  Re
```

```
select ceil(ceil(2286/16))*(10+16*8192/4096)
         /
         (10+8192/4096) +1 "IOCOST"
  from dual;

    IOCOST
----------
     501.5
```

```
NAME                           VALUE DESCRIB
------------------------------ ----- --------------------------------
_optimizer_ceil_cost           TRUE  CEIL cost in CBO
_table_scan_cost_plus_one      TRUE  bump estimated full table scan and index ffs cost by one
```

算法决定成败 – 分区变化与计划选择

- SQL弃索引而选择全表
  - 数据库从10g升级到11g
  - 分区表新增了分区

```
Select *
  From payment
 Where ID = '11478584201212190020696328382O'

============
Plan Table
============
| Id  | Operation                | Name    | Rows  | Bytes | Cost  | Time      | Pstart| Pstop |
| 0   | SELECT STATEMENT         |         |       |       |   162 |           |       |       |
| 1   |  PARTITION RANGE ALL     |         |    10 |  1470 |   162 |  00:00:02 | 1     | 355   |
| 2   |   PARTITION HASH SINGLE  |         |    10 |  1470 |   162 |  00:00:02 | 1     | 1     |
| 3   |    TABLE ACCESS FULL     | PAYMENT |    10 |  1470 |   162 |  00:00:02 | 1     | 355   |
Predicate Information:
----------------------
3 - filter("ID"='11478584201212190020696328382O')
```

算法决定成败 – 分区变化与计划选择

- SQL弃索引而选择全表
  - 选择源于算法计算
  - 全表优于索引 - 然而与真实情况不符

```
Access path analysis for PAYMENT
****************************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for PAYMENT[PAYMENT]
  Column (#2):
    NewDensity:0.000001, OldDensity:0.000447 BktCnt:75, PopBktCnt:15, PopValCnt:1, NDV:706153
  Column (#2): ID(
    AvgLen: 16 NDV: 706153 Nulls: 3678540 Density: 0.000001
    Histogram: HtBal  #Bkts: 75  UncompBkts: 75  EndPtVals: 61
  Table: PAYMENT  Alias: PAYMENT
    Card: Original: 12177100.000000  Rounded: 10  Computed: 9.63  Non Adjusted: 9.63
  Access Path: TableScan
    Cost:  161.67  Resp: 161.67  Degree: 0
      Cost_io: 161.00  Cost_cpu: 12743804
      Resp_io: 161.00  Resp_cpu: 12743804
  Access Path: index (AllEqRange)
    Index: PAYMENT_IDX_ID
    resc_io: 717.00  resc_cpu: 5117772
    ix_sel: 0.000001  ix_sel_with_filters: 0.000001
    Cost: 717.27  Resp: 717.27  Degree: 1
  Best:: AccessPath: TableScan
         Cost: 161.67  Degree: 1  Resp: 161.67  Card: 9.63  Bytes: 0
****************************************
```

算法决定成败 – 分区变化与计划选择

- SQL弃索引而选择全表
  - 原始信息无误
  - 经过调整之后端倪初显

```
***************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: PAYMENT  Alias: PAYMENT  (Using composite stats)
  (making adjustments for partition skews)
  ORIGINAL VALUES::     #Rows: 12177100  #Blks:  257673  AvgRowLen:  147.00  ChainCnt:  0.00
  PARTITIONS::
  PRUNED: 355
  ANALYZED: 355  UNANALYZED: 0
    #Rows: 12177100  #Blks:  727  AvgRowLen:  147.00  ChainCnt:  0.00
Index Stats::
```

```
select ceil(ceil(727/16))*(10+16*8192/4096)
         /
         (10+8192/4096) +1 "IOCOST"
  from dual;

    IOCOST
----------
       162
```

算法决定成败 – 分区变化与计划选择

- SQL弃索引而选择全表

收集统计信息 – 你关注背后了么？

- 在新增分区之后收集统计信息
  - exec dbms_stats.gather_table_stats('EYGLE','T_PART_HASH');

```
PROCEDURE GATHER_TABLE_STATS
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 OWNNAME                        VARCHAR2                IN
 TABNAME                        VARCHAR2                IN
 PARTNAME                       VARCHAR2                IN     DEFAULT
 ESTIMATE_PERCENT               NUMBER                  IN     DEFAULT
 BLOCK_SAMPLE                   BOOLEAN                 IN     DEFAULT
 METHOD_OPT                     VARCHAR2                IN     DEFAULT
 DEGREE                         NUMBER                  IN     DEFAULT
 GRANULARITY                    VARCHAR2                IN     DEFAULT
 CASCADE                        BOOLEAN                 IN     DEFAULT
 STATTAB                        VARCHAR2                IN     DEFAULT
 STATID                         VARCHAR2                IN     DEFAULT
 STATOWN                        VARCHAR2                IN     DEFAULT
 NO_INVALIDATE                  BOOLEAN                 IN     DEFAULT
 STATTYPE                       VARCHAR2                IN     DEFAULT
 FORCE                          BOOLEAN                 IN     DEFAULT
 CONTEXT                        CCONTEXT                IN     DEFAULT
```

统计信息收集 – 粒度和级别

- 不同粒度收集的统计信息级别不同
  - 这些差异对于分区表尤为重要

Granularity	Global Statistics Level
Global	Table
Partition	Partition
SubPartition	SubPartition
Default	Table + Partition
All	Table + Partition + Subpartition

```
SQL> select dbms_stats.get_param('GRANULARITY') from dual;

DBMS_STATS.GET_PARAM('GRANULARITY')
-----------------------------------
AUTO
```

微观Oracle 10g/11g

- 通过实例再现问题

```
create table t_part_hash (id number, name varchar2(30), type varchar2(30), created date)
partition by range (created) subpartition by hash (id) subpartitions 1 (
                  partition p1
              values less than (to_date('201201', 'yyyymm'))
                   , partition p2
              values less than (to_date('201202', 'yyyymm'))
                   , partition p3
              values less than (to_date('201203', 'yyyymm'))
                   , partition p4
              values less than (to_date('201204', 'yyyymm'))
                   , partition p5
              values less than (to_date('201205', 'yyyymm'))
                   , partition p6
              values less than (to_date('201206', 'yyyymm'))
                   , partition p7
              values less than (to_date('201207', 'yyyymm'))
                   , partition p8
              values less than (to_date('201208', 'yyyymm'))
                   , partition p9
              values less than (to_date('201209', 'yyyymm'))
                   , partition p10
              values less than (to_date('201210','yyyymm'))
                   , partition p11
              values less than (to_date('201211', 'yyyymm'))
                   , partition p12
              values less than (to_date('201212', 'yyyymm'))
                   , partition p13
              values less than (to_date('201301', 'yyyymm'))
                   , partition p14
              values less than (to_date('201302', 'yyyymm'))
                   , partition p15
              values less than (to_date('201303', 'yyyymm'))
                   , partition p16
              values less than (to_date('201304', 'yyyymm'))
             );

create index ind_part_hash_name on t_part_hash(name) local;
```

```
insert into t_part_hash
select object_id
     , object_name
     , object_type
     , to_date('20110101','yyyymmdd')
  from dba_objects
 where rownum < 10001;
insert into t_part_hash
select object_id
     , object_name
     , object_type
     , to_date('20120101','yyyymmdd')
  from dba_objects
 where rownum < 10001;
insert into t_part_hash
select object_id
     , object_name
     , object_type
     , to_date('20120201','yyyymmdd')
  from dba_objects
 where rownum < 10001;
insert into t_part_hash
select object_id
     , object_name
     , object_type
     , to_date('20120301','yyyymmdd')
  from dba_objects
 where rownum < 10001;
insert into t_part_hash
select object_id
     , object_name
     , object_type
     , to_date('20120401','yyyymmdd')
  from dba_objects
 where rownum < 10001;
```

微观Oracle 10g/11g

- 执行计划的评估

```
*******************************
SYSTEM STATISTICS INFORMATION
*******************************
  Using NOWORKLOAD Stats
  CPUSPEED: 2336 millions instruction/sec
  IOTFRSPEED: 4096 bytes per millisecond (default is
  IOSEEKTIM: 10 milliseconds (default is 10)
***************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: T_PART_HASH  Alias: T_PART_HASH  (Using com
    #Rows: 160264  #Blks:  928  AvgRowLen:  36.00
Index Stats::
  Index: IND_PART_HASH_NAME  Col#: 2
    USING COMPOSITE STATS
    LVLS: 1  #LB: 880  #DK: 7705  LB/K: 1.00  DB/K:
***************************************
SINGLE TABLE ACCESS PATH
  ----------------------------------------
  BEGIN Single Table Cardinality Estimation
  ----------------------------------------
  Column (#2): NAME(VARCHAR2)
    AvgLen: 19.00 NDV: 6685 Nulls: 0 Density: 1.4959
  Table: T_PART_HASH  Alias: T_PART_HASH
    Card: Original: 160264  Rounded: 24  Computed: 2
  ----------------------------------------
  END   Single Table Cardinality Estimation
  ----------------------------------------
  Access Path: TableScan
    Cost:  205.49  Resp: 205.49  Degree: 0
      Cost_io: 204.00  Cost_cpu: 41867256
      Resp_io: 204.00  Resp_cpu: 41867256
  Access Path: index (AllEqRange)
    Index: IND_PART_HASH_NAME
    resc_io: 25.00  resc_cpu: 187396
    ix_sel: 1.4984e-004  ix_sel_with_filters: 1.4984
    Cost: 25.01  Resp: 25.01  Degree: 1
  Best:: AccessPath: IndexRange  Index: IND_PART_HAS
         Cost: 25.01  Degree: 1  Resp: 25.01  Card:
***************************************
```

```
*******************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: T_PART_HASH  Alias: T_PART_HASH  (Using composite stats)
  (making adjustments for partition skews)
  ORIGINAL VALUES::    #Rows: 160000  #Blks:  1760  AvgRowLen:  36.00  C
  PARTITIONS::
  PRUNED: 33
  ANALYZED: 33  UNANALYZED: 0
    #Rows: 160000  #Blks:  54  AvgRowLen:  36.00  ChainCnt:  0.00
Index Stats::
  Index: IND_PART_HASH_NAME  Col#: 2
    USING COMPOSITE STATS
    LVLS: 1  #LB: 864  #DK: 8119  LB/K: 1.00  DB/K: 6.00  CLUF: 50544.00
Access path analysis for T_PART_HASH
***************************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T_PART_HASH[T_PART_HASH]
  Column (#2): NAME(
    AvgLen: 19 NDV: 8119 Nulls: 0 Density: 0.000123
  Table: T_PART_HASH  Alias: T_PART_HASH
    Card: Original: 160000.000000  Rounded: 20  Computed: 19.71  Non Adj
  Access Path: TableScan
    Cost:  14.05  Resp: 14.05  Degree: 0
      Cost_io: 14.00  Cost_cpu: 1465112
      Resp_io: 14.00  Resp_cpu: 1465112
  Access Path: index (AllEqRange)
    Index: IND_PART_HASH_NAME
    resc_io: 41.00  resc_cpu: 299779
    ix_sel: 0.000123  ix_sel_with_filters: 0.000123
    Cost: 41.01  Resp: 41.01  Degree: 1
  Best:: AccessPath: TableScan
         Cost: 14.05  Degree: 1  Resp: 14.05  Card: 19.71  Bytes: 0

***************************************
```

Oracle的灵活之处 – 开关

- _optim_adjust_for_part_skews

```
SQL> alter session set "_optim_adjust_for_part_skews"=true;

Session altered.

SQL> select id,name,type from t_part_hash where name = 'ALL_TAB_STATS_HISTORY';

32 rows selected.


Execution Plan
----------------------------------------------------------
Plan hash value: 3712007960

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pst
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             |    20 |   600 |    16   (0)| 00:00:01 |       |
|   1 |  PARTITION RANGE ALL     |             |    20 |   600 |    16   (0)| 00:00:01 |     1 |
|   2 |   PARTITION HASH SINGLE  |             |    20 |   600 |    16   (0)| 00:00:01 |     1 |
|*  3 |    TABLE ACCESS FULL     | T_PART_HASH |    20 |   600 |    16   (0)| 00:00:01 |     1 |
-------------------------------------------------------------------------------------------------------------

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

   3 - filter("NAME"='ALL_TAB_STATS_HISTORY')
```

微观Oracle 11g – 正确的计算

- 正确的统计信息

Oracle Database Version	分区特点	复合分区
8.0.5	Range	
8i	Range、Hash	Range-Hash
9i	Range、Hash、List	Range-Hash、Range-List
10G	Range、Hash、List, Range、List 、Hash for IOT.	Range-Hash、Range-List
11G	Interval、REF、Virtual Column-based Partition	Range-Range/Hash、List-Range/Hash/List、Interval-Range/Hash/List

处于变更边界的诡异案例

- 每月出现一次，每次一天

Plan Statistics

- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	11,760,005	11,760,005.04	12.58
CPU Time (ms)	11,755,542	11,755,541.57	19.94
Executions	1		
Buffer Gets	450,262,425	450,262,425.00	24.85
Disk Reads	9,237	9,237.00	0.03
Parse Calls	1	1.00	0.00
Rows	32	32.00	
User I/O Wait Time (ms)	14,112		
Cluster Wait Time (ms)	0		
Application Wait Time (ms)	0		
Concurrency Wait Time (ms)	6		
Invalidations	0		
Version Count	4		
Sharable Mem(KB)	95		

Plan Statistics

- % Total DB Time is the Elapsed Time of the SQL statement divided into the To

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	17,510	17,510.06	0.02
CPU Time (ms)	1,026	1,026.29	0.00
Executions	1		
Buffer Gets	99,242	99,242.00	0.01
Disk Reads	1,740	1,740.00	0.01
Parse Calls	1	1.00	0.00
Rows	32	32.00	
User I/O Wait Time (ms)	16,543		
Cluster Wait Time (ms)	0		
Application Wait Time (ms)	0		
Concurrency Wait Time (ms)	0		
Invalidations	0		
Version Count	1		
Sharable Mem(KB)	6		

解决之道

- 知己知彼 百战不殆
  - 优化器依赖统计信息，在变更和维护的过程中，充分考虑统计信息的变化；
  - 给DBA时间充分研究那些重要的技术；
  - 扬长避短，手工设置统计信息（SET_TABLE_STATS ）维持计划的稳定；
  - 在变化和边界中识别风险：月底月初、年底年初、分区增减等都可能带来问题；
  - 了解和研究那些被隐藏的秘密；
- DBA 成功之道
  - 不断积累，不断成长，举一反三
  - 绕过问题是一种能力

Q&A