如何解读执行计划

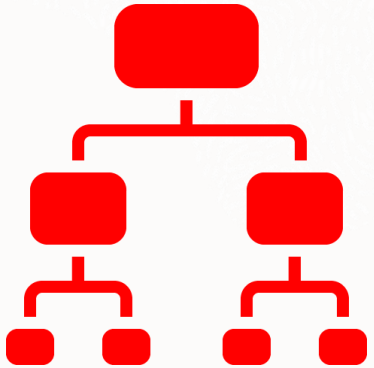
公益讲座11：00分准时开始，请大家先浏览云技术微信公众号技术文章资料会在各群同步发布，已入群客户请勿重复入群！

20-18

数据库和云讲座群

甲骨文云技术公众号

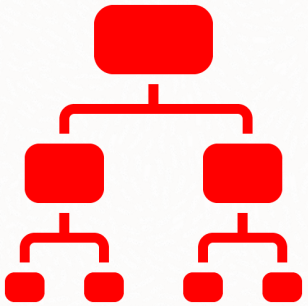
如何解读执行计划



周宇文
Oracle RWP 资深解决方案工程师

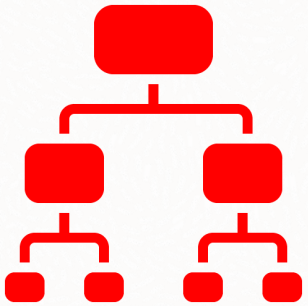
2022年9月2日

议程



1. 执行计划是什么
2. 如何产生执行计划
3. 解读执行计划
4. 案例分析

议程



1. 执行计划是什么
2. 如何产生执行计划
3. 解读执行计划
4. 案例分析

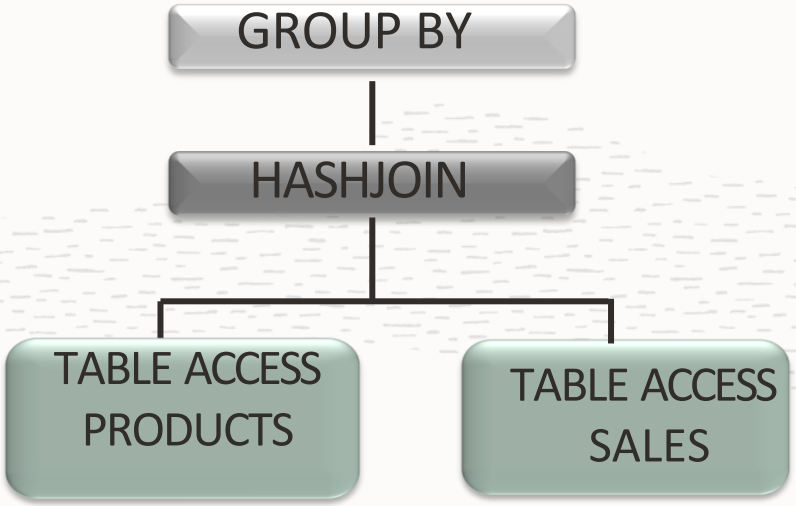
执行计划是什么？

```
SELECT    prod_category,   avg(amount_sold)
FROM      sales s, products p
WHERE      p.prod_id = s.prod_id
GROUP BY prod_category;
```

表格展示执行计划

```
-------------------------------------------
| Id  | Operation            | Name       |
-------------------------------------------
|   0 | SELECT STATEMENT     |            |
|   1 |  HASH GROUP BY       |            |
|*  2 |   HASH JOIN          |            |
|   3 |    TABLE ACCESS FULL| PRODUCTS   |
|   4 |    TABLE ACCESS FULL| SALES      |
-------------------------------------------
```

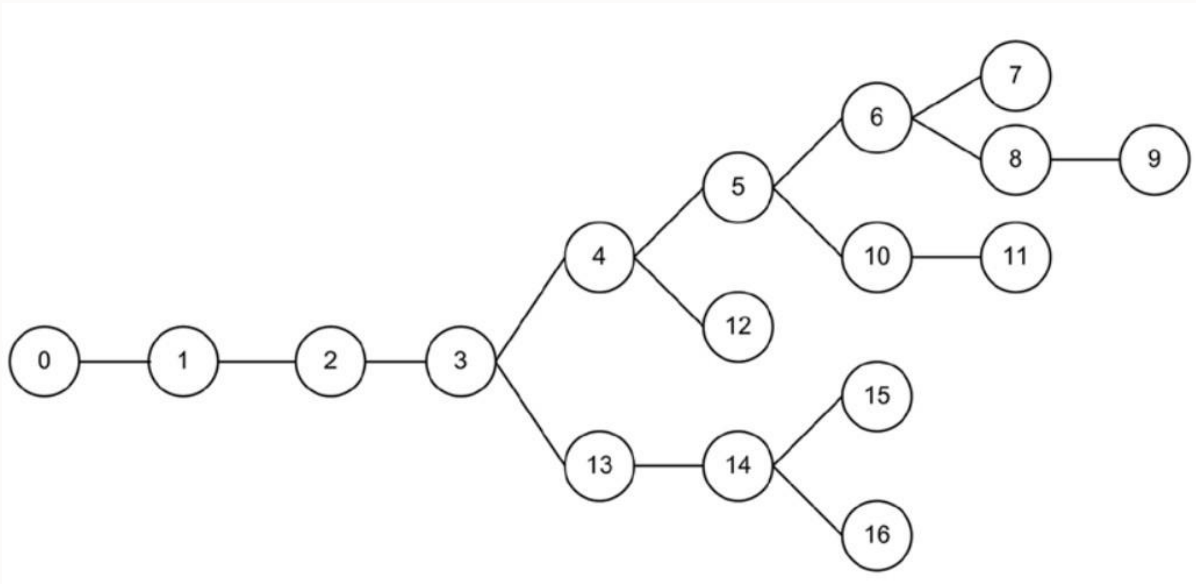
树状展示执行计划



执行计划的执行顺序

```
0  SELECT STATEMENT
1   FILTER
2    SORT GROUP BY
3     FILTER
4       HASH JOIN OUTER                                  G
5         NESTED LOOPS OUTER                             E
6           NESTED LOOPS                                 C
7             TABLE ACCESS FULL                          A
8             TABLE ACCESS BY INDEX ROWID                B
9              INDEX UNIQUE SCAN
10          TABLE ACCESS BY INDEX ROWID                  D
11           INDEX UNIQUE SCAN
12        TABLE ACCESS FULL                              F
13      SORT UNIQUE                                      J
14       UNION-ALL
15        TABLE ACCESS FULL                             H
16        TABLE ACCESS FULL                             I
```

以块结构组成的执行计划，左边标识操作ID，右边的字母表示执行块



按照缩进的规则，步骤7最先执行，然后是9，8，6，11，10，5，12，4，15，16，14，13，3，2，1，0

关注执行计划下面的信息

```
SELECT /*+ gather_plan_statistics */ count(*)  FROM    sales2 WHERE
prod_id=to_number('139')

Plan hash value: 1631620387

-------------------------------------------------------------------------------------
| Id  | Operation         | Name        | Starts | E-Rows | Cost (%CPU)| A-Rows |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |      1 |        |    35 (100)|      1 |
|   1 |  SORT AGGREGATE   |             |      1 |      1 |            |      1 |
|*  2 |   INDEX RANGE SCAN| MY_PROD_IND |      1 |  12762 |    35   (0)|  11574 |
-------------------------------------------------------------------------------------

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

   2 - access("PROD_ID"=139)
```

Access predicate

- 可以使用高效的访问结构，比如内存中的hash表、索引迅速定位记录。

关注执行计划下面的信息

```
SQL> SELECT username
  2  FROM   my_users
  3  WHERE  username LIKE 'MAR%';

USERNAME
------------------------------------------------------------------------------
MARIA

Plan hash value: 2982854235

-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |       |       |     2 (100)|          |
|*  1 |  TABLE ACCESS FULL| MY_USERS |     1 |    66 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("USERNAME" LIKE 'MAR%')
```

Filter predicate

- 用于在获取数据后，通过谓词条件进行数据过滤

关注执行计划下面的信息

```
SELECT      p.prod_name, sum(s.amount_sold) amt FROM        Sales s,
Products p  WHERE       s.prod_id=p.prod_id AND         p.supplier_id =
:sup_id  group by p.prod_name

Plan hash value: 187119048

-------------------------------------------------------------------------------------
| Id  | Operation               | Name               | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                    |       |       |   573 (100)|
|   1 |  HASH GROUP BY          |                    |    71 |  3550 |   573  (10)|
|*  2 |   HASH JOIN             |                    |    72 |  3600 |   572  (10)|
|   3 |    VIEW                 | VW_GBC_5           |    72 |  1224 |   570  (10)|
|   4 |     HASH GROUP BY       |                    |    72 |   648 |   570  (10)|
|   5 |      PARTITION RANGE ALL|                    |   918K|  8075K|   530   (3)|
|   6 |       TABLE ACCESS FULL | SALES              |   918K|  8075K|   530   (3)|
|*  7 |    INDEX RANGE SCAN     | PROD_SUPP_ID_INDX  |    72 |  2376 |     1   (0)|
-------------------------------------------------------------------------------------

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

   2 - access("ITEM_1"="P"."PROD_ID")
   7 - access("P"."SUPPLIER_ID"=:SUP_ID)

Note
-----
   - SQL plan baseline SQL_PLAN_11v9s0fh9t3z1aa1ba510 used for this statement
```

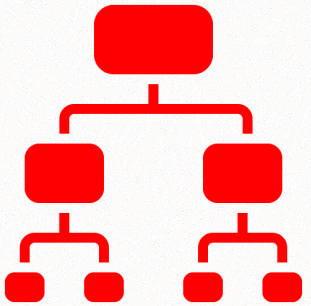
Note部分

关于当前计划使用优化器特性的详细信息，比如:

- RuleBasedOptimizer(RBO)
- DynamicSampling
- Outlines
- SQL Profiles or Plan baselines
- AdaptivePlans
- Hint Report (19c开始支持)

```
Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 2  U - Unused (1))
-------------------------------------------------------------------------------

   0 -  STATEMENT
           -  rule

   2 -  SEL$1 / T@SEL$1
         U -  pq_distribute(t hash hash)
```

议程



1. 执行计划是什么
2. 如何产生执行计划
3. 解析执行计划
4. 执行计划用例

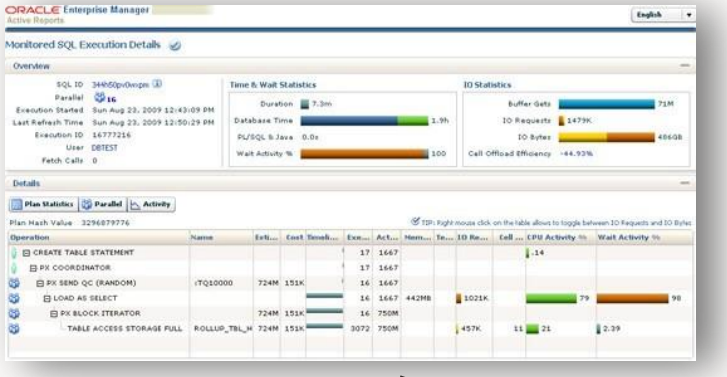
多种方法查看执行计划

Autotrace

```
SQL> set autotrace on
SQL> select * from dual;

D
-
X

Elapsed: 00:00:00.74

Execution Plan
----------------------------------------------------------
Plan hash value: 272002086

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


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          6  consistent gets
          0  physical reads
          0  redo size
        342  bytes sent via SQL*Net to client
        471  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> _
```

SQLDeveloper

SQLMonitor



TKPROF

```
SELECT job_id,SUM(salary),COUNT(*) FROM employees GROUP BY job_id
HAVING SUM(salary)=(SELECT MAX(SUM(salary)) FROM EMPLOYEES GROUP BY
job_id)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.04          0         14          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.04          0         14          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 85

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  FILTER  (cr=14 pr=0 pw=0 time=0 us)
     19   HASH GROUP BY (cr=7 pr=0 pw=0 time=90 us cost=4 size=13 card=1)
    107    TABLE ACCESS FULL EMPLOYEES (cr=7 pr=0 pw=0 time=318 us cost=3 size=1391 card=107)
      1   SORT AGGREGATE (cr=7 pr=0 pw=0 time=0 us cost=4 size=13 card=1)
     19    SORT GROUP BY (cr=7 pr=0 pw=0 time=72 us cost=4 size=13 card=1)
    107     TABLE ACCESS FULL EMPLOYEES (cr=7 pr=0 pw=0 time=318 us cost=3 size=1391 card=107)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------    Waited  ----------  ------------
  SQL*Net message to client                        2        0.00          0.00
  Disk file operations I/O                         1        0.03          0.03
  SQL*Net message from client                      2        0.01          0.01
  asynch descriptor resize                         1        0.00          0.00
********************************************************************************
```

.....但是实际上只有两种方法显示执行计划

如何显示执行计划

1. EXPLAIN PLANcommand
   - 显示一个SQL的执行计划，但是实际上不执行这个语句
2. V$SQL_PLAN
   - 从Oracle9i开始使用的数据字典视图，在游标缓存里面包含了SQL实际的执行计划

某些情况下使用EXPLAIN 看到的计划和V$SQL_PLAN会不同

如何显示执行计划

EXPLAINPLAN 命令&dbms_xplan.display函数

```
SQL>EXPLAIN PLAN FOR
      SELECT    prod_category,   avg(amount_sold)
      FROM      sales s, products p
      WHERE       p.prod_id = s.prod_id
      GROUP BY prod_category;
```

```
SQL>  SELECT   *   FROM
          table( dbms_xplan.display ( 'plan_table',     null,      'basic'));
```

PLAN TABLE NAME　　STATEMENT ID　　FORMAT

如何显示执行计划

产生并显示当前会话最后一个SQL的执行计划

```
SQL> SELECT   prod_category,  avg(amount_sold)
     FROM     sales s, products p
     WHERE    p.prod_id = s.prod_id
     GROUP BY prod_category;
```

```
SQL> SELECT  *  FROM
        table( dbms_xplan.display_cursor ( null,     null,     'basic'));
```

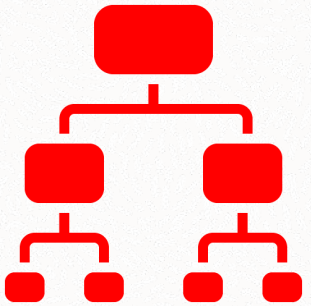
SQL_ID　　CHILD NUMBR　　FORMAT

Format有如下几个可选参数:

Basic:仅显示计划包含的操作和对象名称 Typical：显示大部分信息，但是outline、字段投影不显示

All：显示除了outline以外所有信息　Advanced:显示所有信息

议程



1 什么是执行计划

2 产生执行计划

3 解析执行计划

- Cardinality
- Accesspaths
- Joinmethods
- Join order

4 案例分析

Cardinality（基数）

预估每个操作返回的行数

优化如何计算基数的?

Cardinality单列等值谓词条件=**行总数/唯一值个数**

比如:表有100行数据,列a有5个不同的唯一值

=> cardinality=**20**

复杂的谓词条件需要更高级的cardinality估算

为什么要关注Cardinality?

它会影响执行计划的每个步骤! 访问方法, Join类型, Join 顺序 等.

确定执行计划中的基数

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				12 (100)	
1	NESTED LOOPS					
2	NESTED LOOPS		1	211	12 (9)	00:00:01
3	NESTED LOOPS		1	185	11 (10)	00:00:01
* 4	HASH JOIN		1	155	10 (10)	00:00:01
5	MERGE JOIN CARTESIAN		107	8774	6 (0)	00:00:01
* 6	TABLE ACCESS FULL	DEPARTMENTS	1	30	3 (0)	00:00:01
7	BUFFER SORT		107	5564	3 (0)	00:00:01
8	TABLE ACCESS FULL	EMPLOYEES	107	5564	3 (0)	00:00:01
9	TABLE ACCESS FULL	EMPLOYEES	107	7811	3 (0)	00:00:01
* 10	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	30	1 (0)	00:00:01
* 11	INDEX UNIQUE SCAN	DEPT_ID_PK	1		0 (0)	
* 12	INDEX UNIQUE SCAN	JOB_ID_PK	1		0 (0)	
13	TABLE ACCESS BY INDEX ROWID	JOBS	1	26	1 (0)	00:00:01

Predicate Information (identified by operation id):

```
   4 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID" AND
              "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
       filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISS
              ARY"+"E"."COMMISSION_PCT"))
   6 - filter("D"."DEPARTMENT_NAME"='Sales')
  10 - filter("D"."DEPARTMENT_NAME"='Sales')
  11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
  12 - access("E"."JOB_ID"="J"."JOB_ID")
```

Cardinality – 预估返回的行数

可以通过对每个表使用SELECT COUNT(*)from表where条件加上相关的谓词条件来计算正确的cardinality

如何校验预估基数的准确性?

```
SELECT /*+ gather_plan_statistics */  p.prod_name,
           SUM(s.quantity_sold)
FROM       sales s, products p
WHERE      s.prod_id =p.prod_id
GROUP  BY p.prod_name ;

SELECT * FROM table (
                DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALLSTATS LAST'));
```

alter session set statistics_level=all也可以使后面的计划显示A-rows

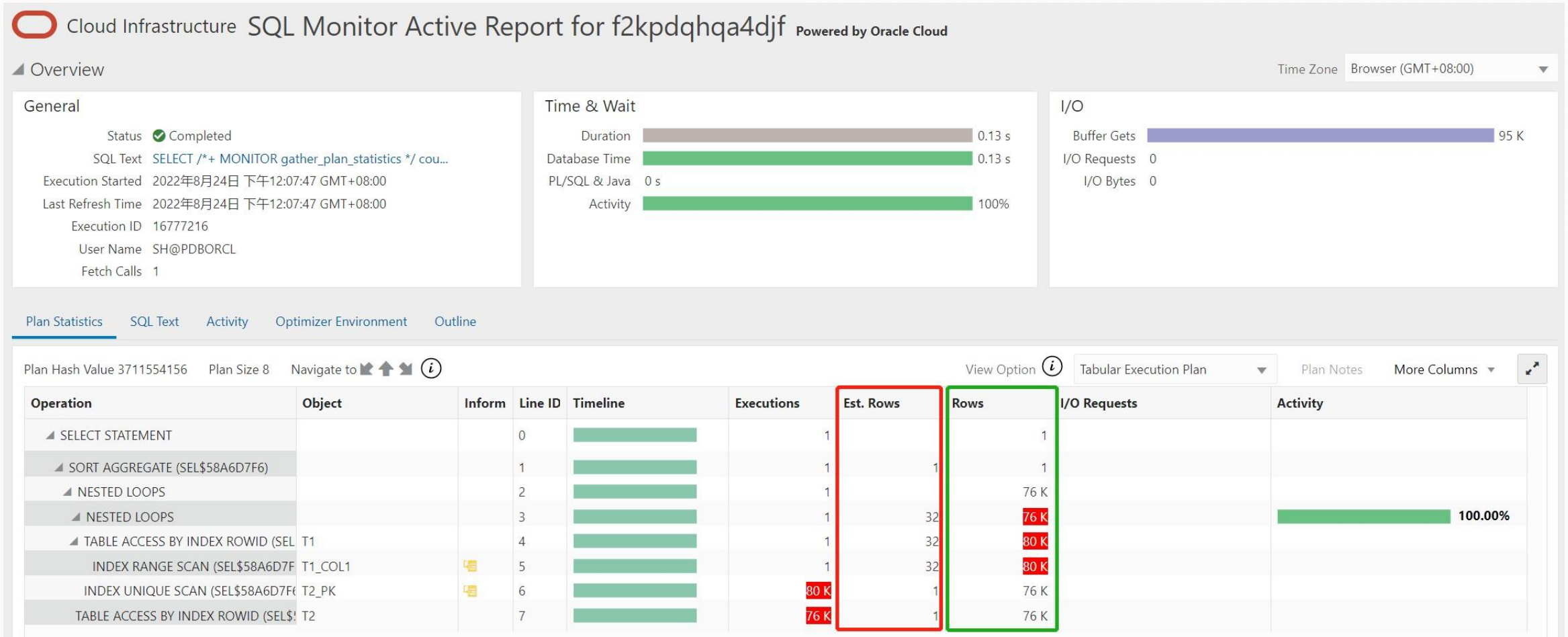
校验预估基数

```
SELECT * FROM table (
        DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALLSTATS LAST'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:00.57	1638			
1	HASH GROUP BY		1	71	71	00:00:00.57	1638	799K	799K	3079K (0)
* 2	HASH JOIN		1	918K	918K	00:00:00.85	1638	933K	933K	1279K (0)
3	TABLE ACCESS STORAGE FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	PARTITION RANGE ALL		1	918K	918K	00:00:00.37	1635			
5	TABLE ACCESS STORAGE FULL	SALES	28	918K	918K	00:00:00.20	1635			

比较预估行数(E-Rows)和实际返回行数(A-Rows)

使用SQL Monitor查看基数

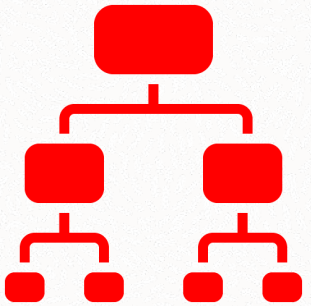


这是最简单的比较预估基数和实际返回行数的方法

错误预估基数应对策略

原因	解决方法
无统计信息或统计信息过时	DBMS_STATS收集统计信息
DataSkew数据倾斜	创建直方图
单个表上面多个单列的谓词	使用DBMS_STATS.CREATE_EXTENDED_STATS创建column group
列上面使用函数	使用DBMS_STATS.CREATE_EXTENDED_STATS创建函数列的统计信息
Join条件中使用多个列	使用DBMS_STATS.CREATE_EXTENDED_STATS创建column group
包含多个表的多个列的表达式	使用动态采样级别4或以上

议程



1 什么是执行计划

2 产生执行计划

3 解析执行计划

- Cardinality
- Accesspaths
- Joinmethods
- Join order

4 案例分析

Access Paths —— 获取数据

访问路径	说明
Full table scan	读取所有的数据并且过滤掉不符合谓词条件的数据。常见于当表没有index或使用并行的情况下。
Table access by Rowid	通过提供的rowid查询表。rowid一般通过index或者在where条件中提供。
Index unique scan	仅返回一行。比如：唯一约束或主键约束上面的等值查询操作
Index range scan	返回多个rowid。常见于非唯一索引的等值查询或唯一索引的范围查询
Index skip scan	查询条件没有包含索引的前导列，而索引前导列唯一值比较少，索引第二列唯一值比较多，这种情况下获益较多。
Index full scan	索引全扫描读取索引的所有叶块。当所有必要的列都在索引中并且返回非空记录适用，返回的记录是有序的。
Index fast full scan	扫描所有index块来代替全表扫描，可以利用多块读能力并且可以并行，但是结果集是无序的
Index joins	查询的所有列都在相关的index中，通过hash join把多个index连接起来，可以避免回表，返回的数据是无序的
Bitmap indexes	对键值使用位图和将每个位的位置转换为rowid 的映射函数。WHERE子句中的多个条件相对应的索引可以高效地合并从而获得交集

确定执行计划中的访问路径

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                 |       |       |    12 (100)|          |
|   1 |  NESTED LOOPS                 |                 |       |       |            |          |
|   2 |   NESTED LOOPS                |                 |     1 |   211 |    12   (9)| 00:00:01 |
|   3 |    NESTED LOOPS               |                 |     1 |   185 |    11  (10)| 00:00:01 |
|*  4 |     HASH JOIN                 |                 |     1 |   155 |    10  (10)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN     |                 |   107 |  8774 |     6   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL       | DEPARTMENTS     |     1 |    30 |     3   (0)| 00:00:01 |
|   7 |       BUFFER SORT             |                 |   107 |  5564 |     3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL      | EMPLOYEES       |   107 |  5564 |     3   (0)| 00:00:01 |
|   9 |      TABLE ACCESS FULL        | EMPLOYEES       |   107 |  7811 |     3   (0)| 00:00:01 |
|* 10 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS    |     1 |    30 |     1   (0)| 00:00:01 |
|* 11 |      INDEX UNIQUE SCAN        | DEPT_ID_PK      |     1 |       |     0   (0)|          |
|* 12 |    INDEX UNIQUE SCAN          | JOB_ID_PK       |     1 |       |     0   (0)|          |
|  13 |   TABLE ACCESS BY INDEX ROWID | JOBS            |     1 |    26 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   4 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID" AND
              "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISSION_PCT")>="E"."SALARY"+("E"."SAL
              ARY"+"E"."COMMISSION_PCT"))
   6 - filter("D"."DEPARTMENT_NAME"='Sales')
  10 - filter("D"."DEPARTMENT_NAME"='Sales')
  11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  12 - access("E"."JOB_ID"="J"."JOB_ID")
```

在 Operation 部分可以查看每个对象的访问方法

假如使用了错误的访问方法，先检查cardinality, join顺序…

访问路径-例1

下面查询使用什么计划?

表customers有10000条记录并且在列cust_id上面有主键

```
SELECT    country_id, name
FROM      customers
WHERE     cust_id IN (100,200,100000);
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |     3 |    39 |     3   (0)| 00:00:01 |
|   1 |  INLIST ITERATOR             |            |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS  |     3 |    39 |     3   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | C_ID_IDX   |     3 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

Predicate Information (identified by operation id):

---

3 - access("CUST_ID"=100 OR "CUST_ID"=200 OR "CUST_ID"=100000)

访问路径-例2

下面查询使用什么计划?

表customers有10000条记录并且在列cust_id上面有主键

```
SELECT   country_id, name
FROM     customers
WHERE    cust_id BETWEEN 100  AND 150;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                                | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |            |     1 |    13 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMERS  |     1 |    13 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                       | C_ID_IDX   |     1 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

访问路径-例3

下面查询使用什么计划?

表customers有10000条记录并且在列cust_id上面有主键

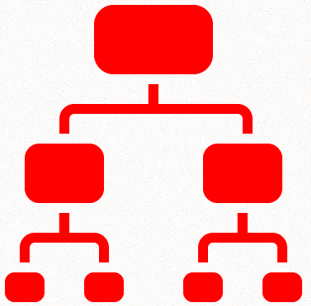
```
SELECT    country_id, name
FROM      customers
WHERE     country_name = 'USA';
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |    30 |   480 |     5   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL | CUSTOMERS |    30 |   480 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

常见的访问路径的问题

问题	可能原因
使用FULL TABLE SCAN而非index scan	表设置了并行、谓词无索引、初始参数MBRC比较大、Index的Clustering_factor非常大
选择了错误的索引	统计信息过时或缺失 全索引访问的成本比索引查找后再回表成本更低 选择与大多数列匹配的索引

议程



1 什么是执行计划

2 产生执行计划

3 解析执行计划

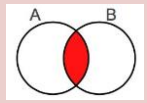
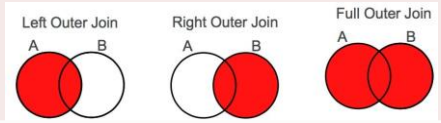
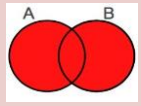
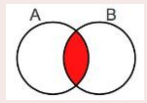
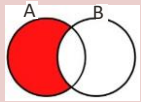
- Cardinality
- Accesspaths
- Joinmethods
- Join order

4 案例分析

Join methods连接方法

连接方法	说明
Nested Loops joins	对于驱动表的每一行需要访问被驱动表一次，适合于当驱动表结果集很小而且被驱动表有高效的访问方法（索引）的情况下
Hash Joins	参与连接的两个表在应用了过滤条件后，得到结果集小的表通过关联键在内存中创建哈希表。然后扫描大表，对每一行都做同样的hash算法并在哈希表中查找是否匹配。适用于大表之间的等值查询。
Sort Merge joins	包含两个步骤: 1. 排序操作: 两个输入的结果都通过连接条件来排序. 2. 合并操作: 合并前面排好序的结果集. 适用于两个表之间的连接条件是非等值条件，或其中一个表已经排序（比如索引访问）时很有用

Join types连接类型

连接类型	说明
Inner Joins 	返回所有满足连接条件的行
Outer Joins 	结果集除了包含完全满足连接条件的记录之外还会包含特定表中不满足该连接条件的记录
Cartesian Joins 	笛卡尔连接，两个数据源的每一行都要关联一次, 适用于表很小的情况，在没有join条件下唯一的方法
Semi-Join 	常见于表join使用exists或in子句，返回满足子查询条件的行
Anti-Join 	常见于表join使用not exists或not in子句，返回不满足子查询条件的行

确定执行计划中的连接方法

```
-----------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |       |       |    12 (100)|          |
|   1 |  NESTED LOOPS                 |             |       |       |            |          |
|   2 |   NESTED LOOPS                |             |     1 |   211 |    12   (9)| 00:00:01 |
|   3 |    NESTED LOOPS               |             |     1 |   185 |    11  (10)| 00:00:01 |
|*  4 |     HASH JOIN                 |             |     1 |   155 |    10  (10)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN     |             |   107 |  8774 |     6   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL       | DEPARTMENTS |     1 |    30 |     3   (0)| 00:00:01 |
|   7 |       BUFFER SORT             |             |   107 |  5564 |     3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL      | EMPLOYEES   |   107 |  5564 |     3   (0)| 00:00:01 |
|   9 |      TABLE ACCESS FULL        | EMPLOYEES   |   107 |  7811 |     3   (0)| 00:00:01 |
|* 10 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |     1 |    30 |     1   (0)| 00:00:01 |
|* 11 |      INDEX UNIQUE SCAN        | DEPT_ID_PK  |     1 |       |     0   (0)|          |
|* 12 |    INDEX UNIQUE SCAN          | JOB_ID_PK   |     1 |       |     0   (0)|          |
|  13 |   TABLE ACCESS BY INDEX ROWID | JOBS        |     1 |       |            |          |
-----------------------------------------------------------------------------------------
```

查看计划中Operation部分确定表使用的连接方法，如果连接方法不对，首先检查Cardinality是否准确

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

   4 - access("E","MANAGER_ID"="E","EMPLOYEE_ID" AND
              "E","DEPARTMENT_ID"="D","DEPARTMENT_ID")
       filter("E","SALARY"+("E","SALARY"+"E","COMMISSION_PCT")>="E","SALARY"+("E","SAL
              ARY"+"E","COMMISSION_PCT"))
   6 - filter("D","DEPARTMENT_NAME"='Sales')
  10 - filter("D","DEPARTMENT_NAME"='Sales')
  11 - access("E","DEPARTMENT_ID"="D","DEPARTMENT_ID")
  12 - access("E","JOB_ID"="J","JOB_ID")
```

表连接方法 - 例1

下面查询应该使用什么连接方法?

```
SELECT    e.last_name, e.salary, d.department_name
FROM      hr.employees e, hr.departments d
WHERE     d.departments_name IN ('Marketing','Sales')
AND       e.department_id = d.department_id;
```

Employees： 107行

Departments： 27行

主外键约束：Employees和Departments的dept_id列

表连接方法 - 例1

下面查询应该使用什么连接方法?

```
SELECT    e.last_name, e.salary, d.department_name
FROM      hr.employees e, hr.departments d
WHERE     d.departments_name IN ('Marketing','Sales')
AND       e.department_id = d.department_id;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     |    19 |   722 |     3   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                     |       |       |            |          |
|   2 |   NESTED LOOPS               |                     |    19 |   722 |     3   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL         | DEPARTMENTS         |     2 |    32 |     2   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN          | EMP_DEPARTMENT_IX   |    10 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES           |    10 |   220 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("D"."DEPARTMENT_NAME"='Marketing' OR "D"."DEPARTMENT_NAME"='Sales')
   4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

表连接方法 - 例2

下面查询应该使用什么连接方法?

```
SELECT    o.customer_id, l.unit_price * l.quantity
FROM      oe.orders o, oe.order_items l
WHERE     l.order_id = o.order_id;
```

Orders： 105行

Order Items： 665行

表连接方法 - 例2

下面查询应该使用什么连接方法?

```
SELECT   o.customer_id, l.unit_price * l.quantity
FROM     oe.orders o, oe.order_items l
WHERE    l.order_id = o.order_id;
```

```
-----------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |   665 | 13300 |     8  (25)|
|*  1 |  HASH JOIN          |              |   665 | 13300 |     8  (25)|
|   2 |   TABLE ACCESS FULL | ORDERS       |   105 |   840 |     4  (25)|
|   3 |   TABLE ACCESS FULL | ORDER_ITEMS  |   665 |  7980 |     4  (25)|
-----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("L"."ORDER_ID"="O"."ORDER_ID")
```

表连接方法 - 例3

下面查询应该使用什么连接方法?

```
SELECT o.order_id, o.order_date ,e.name
FROM   oe.orders o , hr.employees e;
```

Orders： 105行

Employees： 107行

表连接方法 - 例3

下面查询应该使用什么连接方法?

```
SELECT o.order_id, o.order_date ,e.name

FROM   oe.orders o , hr.employees e;
```

```
Plan hash value: 3229651169

-------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             | 11235 |   120K|     33   (7)| 00:00:01 |
|   1 |  MERGE JOIN CARTESIAN    |             | 11235 |   120K|     33   (7)| 00:00:01 |
|   2 |   INDEX FULL SCAN        | ORDER_PK    |   105 |   420 |      1   (0)| 00:00:01 |
|   3 |   BUFFER SORT            |             |   107 |   749 |     32   (7)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| EMP_NAME_IX |   107 |   749 |      0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

表连接方法 - 例4

下面查询应该使用什么连接方法?

```
SELECT    s.quantity_sold
FROM      sales s, customers c
WHERE     s.cust_id =c.cust_id;
```

Sales： 960行

Customer： 55,500行

Customer在列cust_id有主键， Sales列cust_id存在外键参照表Customer的cust_id列

表连接方法 - 例4

下面查询应该使用什么连接方法?

```
SELECT    s.quantity_sold
FROM      sales s, customers c
WHERE     s.cust_id =c.cust_id;
```

不需要join条件

表消除查询转换优化：因为有主外键关系的存在，表连接是冗余的。

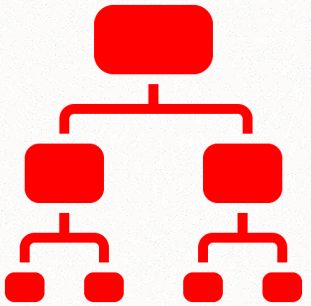
```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
Plan hash value: 2489314924

-------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |   960 |  2880 |     5   (0)| 00:00:01 |       |       |
|   1 |  PARTITION RANGE ALL|       |   960 |  2880 |     5   (0)| 00:00:01 |     1 |    16 |
|   2 |   TABLE ACCESS FULL | SALES |   960 |  2880 |     5   (0)| 00:00:01 |     1 |    16 |
-------------------------------------------------------------------------------------------------------
```

造成选择错误连接方法的原因

问题	原因
选择nested loop而不是hash join	驱动表的Cardinality比实际低估了
Cartesian Joins	Cardinality被低估了或者无关联条件

12c的自适应执行计划可以根据实际情况决定哪种连接方法

议程



1. 什么是执行计划
2. 产生执行计划
3. 解析执行计划
   - Cardinality
   - Accesspaths
   - Joinmethods
   - Join order
4. 案例分析

表连接顺序

- 多个表之间join，需要关注表join的顺序
- 过滤掉行最多的表适合做为连接的驱动表
- 受访问路径的影响比较大
- 有外连接时候，外连接操作需要在其它表谓词操作完成后进行
- 如果无法视图合并，那么视图内引用的表先执行join，然后再和视图外面的表join

在执行计划中确定连接方法

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				12 (100)	
1	NESTED LOOPS					
2	NESTED LOOPS		1	211	12 (9)	00:00:01
3	NESTED LOOPS		1	185	11 (10)	00:00:01
* 4	HASH JOIN		1	155	10 (10)	00:00:01
5	MERGE JOIN CARTESIAN		107	8774	6 (0)	00:00:01
* 6	TABLE ACCESS FULL	DEPARTMENTS	1	30	3 (0)	00:00:01
7	BUFFER SORT		107	5564	3 (0)	00:00:01
8	TABLE ACCESS FULL	EMPLOYEES	107	5564	3 (0)	00:00:01
9	TABLE ACCESS FULL	EMPLOYEES	107	7811	3 (0)	00:00:01
* 10	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	30	1 (0)	00:00:01
* 11	INDEX UNIQUE SCAN	DEPT_ID_PK	1		0 (0)	
* 12	INDEX UNIQUE SCAN	JOB_ID_PK	1		0 (0)	
13	TABLE ACCESS BY INDEX ROWID	JOBS	1	26	1 (0)	00:00:01

1 2 3 4 5

Predicate Information (identified by operation id):

```
   4 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID" AND
              "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISSION_PCT")>="E"."SALARY"+("E"."SAL
              ARY"+"E"."COMMISSION_PCT"))
   6 - filter("D"."DEPARTMENT_NAME"='Sales')
  10 - filter("D"."DEPARTMENT_NAME"='Sales')
  11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  12 - access("E"."JOB_ID"="J"."JOB_ID")
```

返回行数最少的表优先执行

如果表连接顺序不对，检查统计信息, cardinality和访问方法

找到复杂SQL表连接顺序

通过查询执行计划的outline部分，可以找到复杂SQL语句表连接顺序

```
SELECT * FROM table(dbms_xplan.display_cursor(format=>'TYPICAL +OUTLINE'));
```

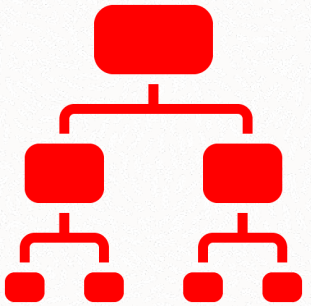
```
Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
      DB_VERSION('11.2.0.2')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$5428C7F1")
      MERGE(@"SEL$2")
      MERGE(@"SEL$3")
      OUTLINE(@"SEL$1")
      OUTLINE(@"SEL$2")
      OUTLINE(@"SEL$3")
      FULL(@"SEL$5428C7F1" "D"@"SEL$3")
      INDEX_RS_ASC(@"SEL$5428C7F1" "E"@"SEL$3" ("EMPLOYEES"."DEPARTMENT_ID"))
      INDEX_RS_ASC(@"SEL$5428C7F1" "E"@"SEL$2" ("EMPLOYEES"."MANAGER_ID"))
      INDEX_RS_ASC(@"SEL$5428C7F1" "J"@"SEL$2" ("JOBS"."JOB_ID"))
      INDEX(@"SEL$5428C7F1" "D"@"SEL$2" ("DEPARTMENTS"."DEPARTMENT_ID"))
      LEADING(@"SEL$5428C7F1" "D"@"SEL$3" "E"@"SEL$3" "E"@"SEL$2" "J"@"SEL$2" "D"@"SEL$2")
      USE_NL(@"SEL$5428C7F1" "E"@"SEL$3")
```

Leading这个hint可以看到join的顺序

导致错误连接顺序的原因

原因
单个表的cardinality估算不正确
表连接的cardinality估算不正确

议程

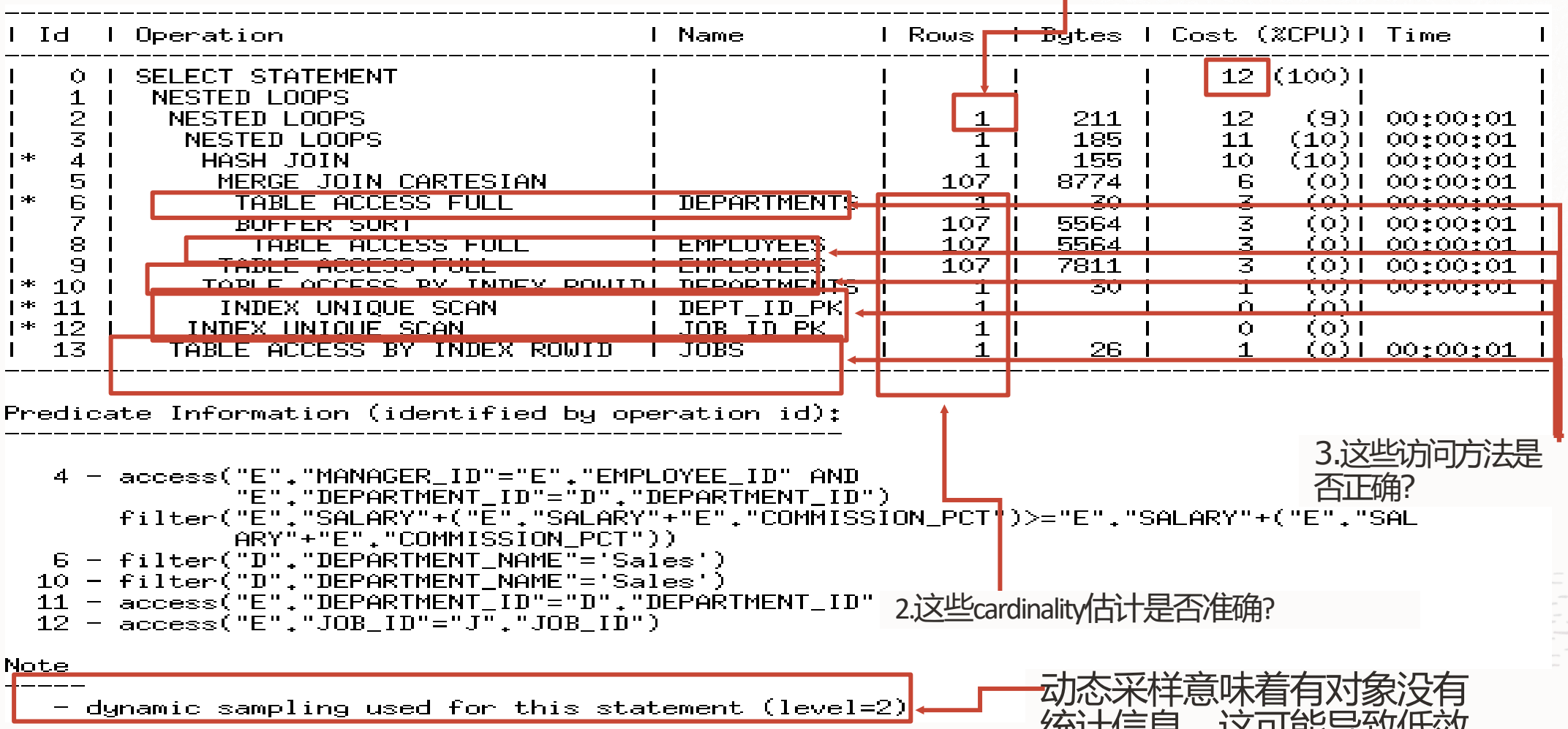


1. 什么是执行计划
2. 产生执行计划
3. 解析执行计划
   - Cardinality
   - Accesspaths
   - Joinmethods
   - Join order
4. 案例分析

案例1

```
SELECT   e1.last_name, e1.job_title, e1.total_comp
FROM     (SELECT   e.manager_id, e.last_name, j.job_title,
                   e.salary+(e.salary+e.commission_pct) total_comp
          FROM     employees e, jobs          j, departments d
          WHERE    d.department_name          = 'Sales'
          AND      e.department_id            = d.department_id
          AND      e.job_id                   = j.job_id ) e1,
         (SELECT   e.employee_id, e.salary+(e.salary+e.commission_pct) tc  employees e,
          FROM     departments d
          WHERE    d.department_name = 'Sales'
          AND      e.department_id            = d.department_id ) e2
WHERE    e1.manager_id = e2.employee_id
AND      e1.total_comp >= e2.tc;
```

这是一个好的计划吗？



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				12 (100)	
1	NESTED LOOPS					
2	NESTED LOOPS		1	211	12 (9)	00:00:01
3	NESTED LOOPS		1	185	11 (10)	00:00:01
* 4	HASH JOIN		1	155	10 (10)	00:00:01
5	MERGE JOIN CARTESIAN		107	8774	6 (0)	00:00:01
* 6	TABLE ACCESS FULL	DEPARTMENTS	1	30	3 (0)	00:00:01
7	BUFFER SORT		107	5564	3 (0)	00:00:01
8	TABLE ACCESS FULL	EMPLOYEES	107	5564	3 (0)	00:00:01
9	TABLE ACCESS FULL	EMPLOYEES	107	7811	3 (0)	00:00:01
* 10	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	30	1 (0)	00:00:01
* 11	INDEX UNIQUE SCAN	DEPT_ID_PK	1		0 (0)	
* 12	INDEX UNIQUE SCAN	JOB_ID_PK	1		0 (0)	
13	TABLE ACCESS BY INDEX ROWID	JOBS	1	26	1 (0)	00:00:01

Predicate Information (identified by operation id):

```
   4 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID" AND
              "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISSION_PCT")>="E"."SALARY"+("E"."SAL
              ARY"+"E"."COMMISSION_PCT"))
   6 - filter("D"."DEPARTMENT_NAME"='Sales')
  10 - filter("D"."DEPARTMENT_NAME"='Sales')
  11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
  12 - access("E"."JOB_ID"="J"."JOB_ID")

Note
-----
   - dynamic sampling used for this statement (level=2)
```

1.预估返回的行数是否准确?

2.这些cardinality估计是否准确?

3.这些访问方法是否正确?

动态采样意味着有对象没有统计信息，这可能导致低效的执行计划

继续分析这个计划

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                 |       |       |    12 (100)|          |
|   1 |  NESTED LOOPS                  |                 |       |       |            |          |
|   2 |   NESTED LOOPS                 |                 |     1 |   211 |    12   (9)| 00:00:01 |
|   3 |    NESTED LOOPS                |                 |     1 |   185 |    11  (10)| 00:00:01 |
|*  4 |     HASH JOIN                  |                 |     1 |   155 |    10  (10)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN      |                 |   107 |  8774 |     6   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL        | DEPARTMENTS     |     1 |    30 |     3   (0)| 00:00:01 |
|   7 |       BUFFER SORT              |                 |   107 |  5564 |     3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL       | EMPLOYEES       |   107 |  5564 |     3   (0)| 00:00:01 |
|   9 |      TABLE ACCESS FULL         | EMPLOYEES       |   107 |  7811 |     3   (0)| 00:00:01 |
|* 10 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS     |     1 |    30 |     1   (0)| 00:00:01 |
|* 11 |      INDEX UNIQUE SCAN         | DEPT_ID_PK      |     1 |       |     0   (0)|          |
|* 12 |    INDEX UNIQUE SCAN           | JOB_ID_PK       |     1 |       |     0   (0)|          |
|  13 |   TABLE ACCESS BY INDEX ROWID  | JOBS            |     1 |    26 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   4 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID" AND
              "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISSION_
              ARY"+"E"."COMMISSION_PCT"))
   6 - filter("D"."DEPARTMENT_NAME"='Sales')
  10 - filter("D"."DEPARTMENT_NAME"='Sales')
  11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  12 - access("E"."JOB_ID"="J"."JOB_ID")

Note
-----
   - dynamic sampling used for this statement (level=2)
```

1 2 3 4 5

4. 使用了正确的join方法吗?

5. Join顺序是否正确? 消除行最多的表最先被使用?

正确的计划

1.实际上仅仅返回1行，计划的成本降低了4

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |       |       |     8  (100)|          |
|   1 |  NESTED LOOPS                 |                     |       |       |             |          |
|   2 |   NESTED LOOPS                |                     |     1 |   102 |     8   (0)| 00:00:01 |
|   3 |    NESTED LOOPS               |                     |     1 |    86 |     7   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |                     |     1 |    59 |     6   (0)| 00:00:01 |
|   5 |      NESTED LOOPS             |                     |    10 |   290 |     4   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL       | DEPARTMENTS         |     1 |    16 |     3   (0)| 00:00:01 |
|   7 |       TABLE ACCESS BY INDEX ROWID| EMPLOYEES        |    10 |   130 |     1   (0)| 00:00:01 |
|   8 |        INDEX RANGE SCAN       | EMP_DEPARTMENT_IX   |    10 |       |     0   (0)|          |
|   9 |      TABLE ACCESS BY INDEX ROWID | EMPLOYEES        |     1 |    30 |     1   (0)| 00:00:01 |
|  10 |       INDEX RANGE SCAN        | EMP_MANAGER_IX      |     6 |       |     0   (0)|          |
|  11 |     TABLE ACCESS BY INDEX ROWID | JOBS              |     1 |    27 |     1   (0)| 00:00:01 |
|* 12 |      INDEX UNIQUE SCAN        | JOB_ID_PK           |     1 |       |     0   (0)|          |
|* 13 |     INDEX UNIQUE SCAN         | DEPT_ID_PK          |     1 |       |     0   (0)|          |
|* 14 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS         |     1 |    16 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operati
-----------------------------------------------------------------------------------------------

   6 - filter("D"."DEPARTMENT_NAME"='Sales')
   8 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   9 - filter("E"."SALARY"+("E"."SALARY"+"E"."COMMISSION_PCT")>="E"."SA
              "."COMMISSION_PCT"))
  10 - access("E"."MANAGER_ID"="E"."EMPLOYEE_ID")
  12 - access("E"."JOB_ID"="J"."JOB_ID")
  13 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  14 - filter("D"."DEPARTMENT_NAME"='Sales')
```

4. 所有的join方法变成NL了

5. Join顺序发生了变化

1 2 3 4 5

3. 部分的访问方法发生了变化

2. Cardinality正确且每个连接的行数减少

案例2:这是一个好的计划吗?

```
SQL> SELECT * FROM table(dbms_xplan.display_cursor('f2kpdqhqa4djf',null,'allstats last'));
SQL_ID  f2kpdqhqa4djf, child number 0
-------------------------------------
SELECT /*+ MONITOR gather_plan_statistics */ count(t2.col2) FROM t1
JOIN t2 USING (id) WHERE t1.col1 = 123

Plan hash value: 3711554156

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name     | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |      1 |        |      1 |00:00:00.13 |   94709 |
|   1 |  SORT AGGREGATE                     |          |      1 |      1 |      1 |00:00:00.13 |   94709 |
|   2 |   NESTED LOOPS                      |          |      1 |        |  75808 |00:00:00.14 |   94709 |
|   3 |    NESTED LOOPS                     |          |      1 |     32 |  75808 |00:00:00.08 |   18901 |
|   4 |     TABLE ACCESS BY INDEX ROWID     | T1       |      1 |     32 |  80016 |00:00:00.04 |    1749 |
|*  5 |      INDEX RANGE SCAN               | T1_COL1  |      1 |     32 |  80016 |00:00:00.01 |     169 |
|*  6 |     INDEX UNIQUE SCAN               | T2_PK    |  80016 |      1 |  75808 |00:00:00.03 |   17152 |
|   7 |    TABLE ACCESS BY INDEX ROWID      | T2       |  75808 |      1 |  75808 |00:00:00.04 |   75808 |
-----------------------------------------------------------------------------------------------------------------

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

   5 - access("T1"."COL1"=123)
   6 - access("T1"."ID"="T2"."ID")
```

E-Rows和A-Rows有数量级的差异，导致了错误的计划!

解决方法：收集直方图信息

```
SQL> SELECT histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'T1' AND column_name = 'COL1';

HISTOGRAM       NUM_BUCKETS
--------------- -----------
NONE                      1

SQL> BEGIN
 dbms_stats.gather_table_stats(
   ownname=>user,
   tabname=>'T1',
   cascade=>TRUE,
   estimate_percent=>100,
   method_opt=>'for all columns size 254',
   no_invalidate=>FALSE);
END;
/
SQL> SELECT histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'T1' AND column_name = 'COL1';

HISTOGRAM       NUM_BUCKETS
--------------- -----------
HEIGHT BALANCED         254
```

```
SELECT /*+ monitor gather_plan_statistics */ count(t2.col2) FROM t1
JOIN t2 USING (id) WHERE t1.col1 = 123

Plan hash value: 906334482

-------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      1 |00:00:00.03 |
|   1 |  SORT AGGREGATE     |      |      1 |      1 |      1 |00:00:00.03 |
|*  2 |   HASH JOIN         |      |      1 |  80000 |  75808 |00:00:00.06 |
|*  3 |    TABLE ACCESS FULL| T1   |      1 |  80000 |  80016 |00:00:00.02 |
|   4 |    TABLE ACCESS FULL| T2   |      1 |   151K |   151K |00:00:00.01 |
-------------------------------------------------------------------------------------

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

   2 - access("T1"."ID"="T2"."ID")
   3 - filter("T1"."COL1"=123)
```

数据字典提供了真实的数据分布信息后，计划从NL变成和Hash，E-Rows和A-Rows完全匹配

案例3，长时间运行的INSERT

```
INSERT INTO TMP_017 (BRCD, BLN_BRANCH_BRCODE, A1, A3, A4, DLEVEL, DSTEP) SELECT /*+index (T
IDX_CLOCK_TLA_LNCINO_BASE)*/ B.BRCODE, B.BLN_BRANCH_BRCODE, '34170', SUM(T.LNBAL), SUM(CASE WHEN
T.SIDT BETWEEN :B2 AND :B1 THEN T.LNAMT ELSE 0 END), '1', '0' FROM                _TLA_LNCINO_BASE T,
BCTL B WHERE T.CUSTNO IN (SELECT C.CUSTCD FROM        BASIC_INFO C WHERE C.POVERTY_STATUS = '1') AND
T.BRCD = B.BRCODE AND B.BRNO <> '01946' AND B.BLN_UP_BRCODE <> '01946' AND T.BHDATE = :B1 AND T.LNCO
IN (SELECT L.CONTRACTNO FROM      INFO L WHERE SUBSTR(L.GUATTYPE, 2, 1) = '1' AND
INSTR(SUBSTR(L.GUATTYPE, 0, 1), '1') = 0 AND INSTR(SUBSTR(L.GUATTYPE, 3, 3), '1') = 0) GROUP BY
B.BLN_BRANCH_BRCODE, B.BRCODE ORDER BY B.BLN_BRANCH_BRCODE, B.BRCODE

Plan hash value: 1686022727

-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT              |                     |       |       | 25771 (100)|          |
|   1 |  SORT GROUP BY                |                     |     1 |   118 | 25771   (1)| 00:05:10 |
|   2 |   NESTED LOOPS                |                     |     1 |   118 | 25770   (1)| 00:05:10 |
|   3 |    NESTED LOOPS               |                     |     1 |    94 | 25769   (1)| 00:05:10 |
|   4 |     MERGE JOIN CARTESIAN      |                     |    15 |   540 | 11414   (1)| 00:02:17 |
|*  5 |      TABLE ACCESS FULL        |      INFO           |     1 |    24 |  5853   (1)| 00:01:11 |
|   6 |      BUFFER SORT              |                     |    53 |   636 |  5561   (2)| 00:01:07 |
|*  7 |       TABLE ACCESS FULL       |      BASIC_INFO     |    53 |   636 |  5561   (2)| 00:01:07 |
|*  8 |     TABLE ACCESS BY INDEX ROWID|     _TLA_LNCINO_BASE|     1 |    58 |   958   (1)| 00:00:12 |
|*  9 |      INDEX RANGE SCAN         |     CK_TLA_LNCINO_BASE|   1 |       |   957   (1)| 00:00:12 |
|* 10 |    TABLE ACCESS BY INDEX ROWID| BCTL                |     1 |    24 |     1   (0)| 00:00:01 |
|* 11 |     INDEX UNIQUE SCAN         | PK_BCTL             |     1 |       |     0   (0)|          |
-----------------------------------------------------------------------------------------------------
```

出现笛卡尔连接，是否合理?

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

   5 - filter((SUBSTR("L"."GUATTYPE",2,1)='1' AND INSTR(SUBSTR("L"."GUATTYPE",0,1),'1')=0 AND
              INSTR(SUBSTR("L"."GUATTYPE",3,3),'1')=0))
```

谓词过滤后返回1行数据?

分析问题

```
SQL_ID  dgbtmsn7x604u, child number 0
-------------------------------------
SELECT  count(*) from t0902 where substr(gttype,2,1)='1' and
instr(substr(gttype,0,1),'1')=0 and instr(substr(gttype,3,3),'1') = 0

Plan hash value: 3447774045

-----------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      1 |00:00:00.06 |     821 |
|   1 |  SORT AGGREGATE    |       |      1 |      1 |      1 |00:00:00.06 |     821 |
|*  2 |   TABLE ACCESS FULL| T0902 |      1 |      1 |  42181 |00:00:00.06 |     821 |
-----------------------------------------------------------------------------------------

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

   2 - filter((SUBSTR("GTTYPE",2,1)='1' AND INSTR(SUBSTR("GTTYPE",0,1),'1')=0
              AND INSTR(SUBSTR("GTTYPE",3,3),'1')=0))
```

优化器估算返回1行，实际上返回42181行

```
**************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: T0902  Alias: T0902
    #Rows: 291426  #Blks:  874  AvgRowLen:  11.00  ChainCnt:  0.00
Access path analysis for T0902
**************************************
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T0902[T0902]
  Table: T0902  Alias: T0902
    Card: Original: 291426.000000  Rounded: 1  Computed: 0.29  Non Adjusted: 0.29
  Access Path: TableScan
    Cost:  241.14  Resp: 241.14  Degree: 0
      Cost_io: 238.00  Cost_cpu: 100216309
      Resp_io: 238.00  Resp_cpu: 100216309
  Best:: AccessPath: TableScan
         Cost: 241.14  Degree: 1  Resp: 241.14  Card: 0.29  Bytes: 0
```

通过10053 event trace看到：估计查询返回0.29行，取整为1行

计算过程如下，每个条件都包含了函数，没有附加统计信息情况下，优化器认为返回1%的数据，三个and条件选择率就是1%*1%*1%

最终谓词过滤返回行数

=291426*1%*1%*1%

=0.29

可能的解决问题方法

```
SQL_ID  53mg52p7uc8y7, child number 0
-------------------------------------
 select /*+ cardinality(t 42181) */ count(*) from t0902 t  where
substr(gttype,2,1)='1' and instr(substr(gttype,0,1),'1')=0 and
instr(substr(gttype,3,3),'1')=0

Plan hash value: 3447774045

-------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      1 |00:00:00.05 |     821 |
|   1 |  SORT AGGREGATE    |       |      1 |      1 |      1 |00:00:00.05 |     821 |
|*  2 |   TABLE ACCESS FULL| T0902 |      1 |  42181 |  42181 |00:00:00.05 |     821 |
-------------------------------------------------------------------------------------------
```

方法1、使用Cardinality 这个hint，提示返回的行数

```
SQL_ID  9506mxp4xms32, child number 0
-------------------------------------
select /*+ OPT_PARAM('optimizer_dynamic_sampling' 4) */ count(*) from
t0902  where substr(gttype,2,1)='1' and instr(substr(gttype,0,1),'1')=0
and instr(substr(gttype,3,3),'1') = 0

Plan hash value: 3447774045

-------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      1 |00:00:00.06 |     821 |
|   1 |  SORT AGGREGATE    |       |      1 |      1 |      1 |00:00:00.06 |     821 |
|*  2 |   TABLE ACCESS FULL| T0902 |      1 |  52365 |  42181 |00:00:00.06 |     821 |
-------------------------------------------------------------------------------------------
```

方法2、级别4的动态采样，预估的数量级是准确的，保证后续的join方法不会错。

可能的解决方法

```
SQL> select gttype,count(*) from t0902
  2  group by gttype;

GTTYPE       COUNT(*)
---------- ----------
00110           35820
01000           42181
00010           36417
01100           37112
01010           34462
00100           34457
10000           36053
01110           34924

8 rows selected.
```

```
SQL_ID  8g380zyvyzfd3, child number 0
-------------------------------------
select count(*) from t0902 where gttype='01000'

Plan hash value: 3447774045

-----------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      1 |
|   1 |  SORT AGGREGATE    |       |      1 |      1 |      1 |
|*  2 |   TABLE ACCESS FULL| T0902 |      1 |  40681 |  42181 |
-----------------------------------------------------------------------
```

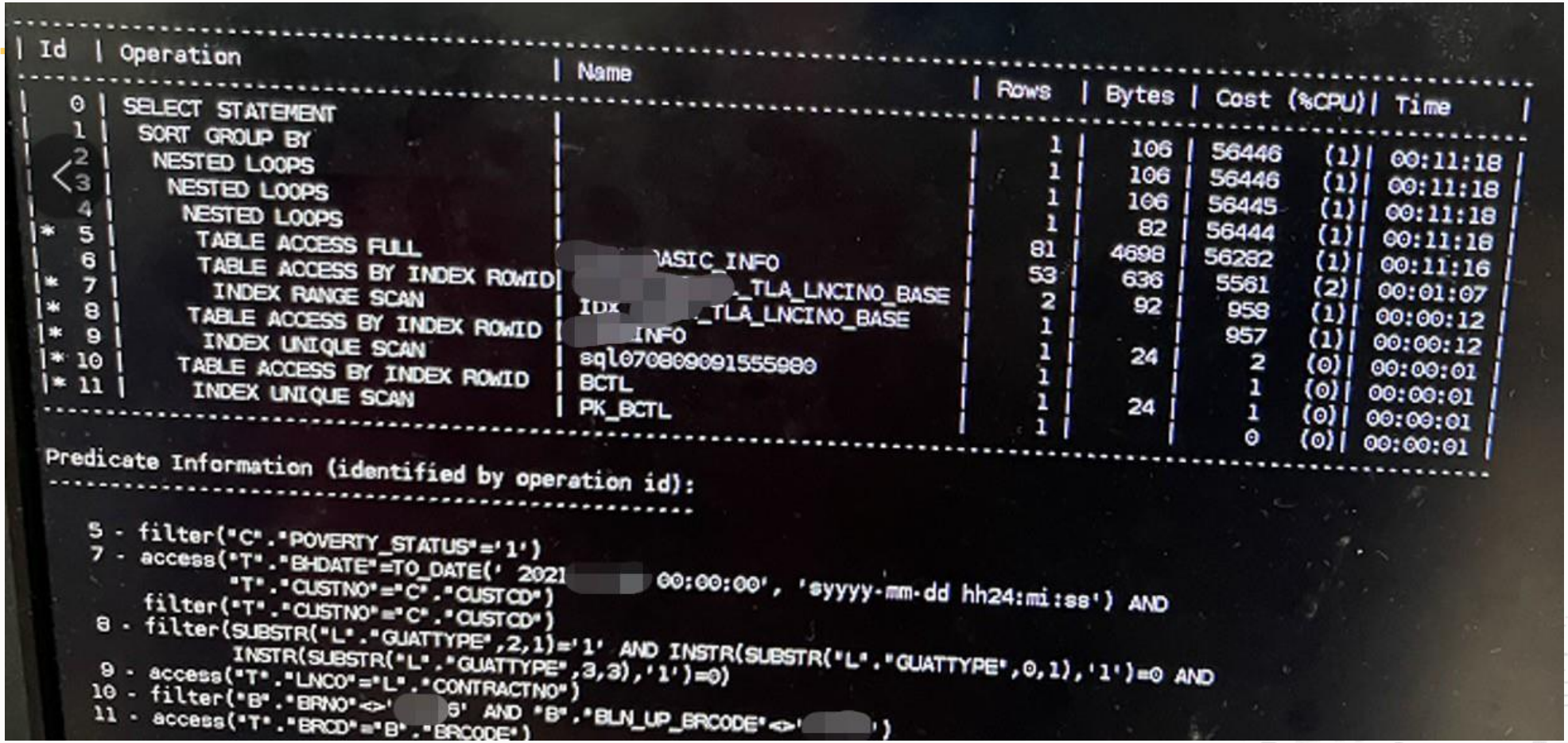
```
select dbms_stats.create_extended_stats(null,'T0902',q'[(instr(substr(gttype,3,3),'1'))]') from dual;
select dbms_stats.create_extended_stats(null,'T0902',q'[(instr(substr(gttype,0,1),'1'))]') from dual;
select dbms_stats.create_extended_stats(null,'T0902','(substr(gttype,2,1))') from dual;
exec dbms_stats.gather_table_stats(user,'t0902',method_opt=>'for all columns size skewonly',no_invalidate=>false);

SQL_ID  bh8prtdym5g4y, child number 0
-------------------------------------
select count(*) from t0902 t where substr(gttype,2,1)='1' and
instr(substr(gttype,0,1),'1')=0 and instr(substr(gttype,3,3),'1') = 0

Plan hash value: 3447774045

-----------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      1 |00:00:00.05 |     821 |
|   1 |  SORT AGGREGATE    |       |      1 |      1 |      1 |00:00:00.05 |     821 |
|*  2 |   TABLE ACCESS FULL| T0902 |      1 |  35764 |  42181 |00:00:00.05 |     821 |
-----------------------------------------------------------------------------------------

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

   2 - filter((SUBSTR("GTTYPE",2,1)='1' AND INSTR(SUBSTR("GTTYPE",3,3),'1')=0
              AND INSTR(SUBSTR("GTTYPE",0,1),'1')=0))
```

方法3、情况根据实际情况，改写查询，数据估算也是准确的。

方法4、创建扩展统计信息，E-rows的估算基本准确。

方法5、可以运行sql tuning advisor，然后根据建议，创建sql profile绑定执行计划。

最终的执行计划



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	106	56446 (1)	00:11:18
1	SORT GROUP BY		1	106	56446 (1)	00:11:18
2	NESTED LOOPS		1	106	56445 (1)	00:11:18
3	NESTED LOOPS		1	82	56444 (1)	00:11:18
4	NESTED LOOPS		81	4698	56282 (1)	00:11:16
* 5	TABLE ACCESS FULL	[redacted]ASIC_INFO	53	636	5561 (2)	00:01:07
6	TABLE ACCESS BY INDEX ROWID	[redacted]_TLA_LNCINO_BASE	2	92	958 (1)	00:00:12
* 7	INDEX RANGE SCAN	IDX[redacted]_TLA_LNCINO_BASE	1		957 (1)	00:00:12
* 8	TABLE ACCESS BY INDEX ROWID	[redacted]INFO	1	24	2 (0)	00:00:01
* 9	INDEX UNIQUE SCAN	sql070809091555980	1		1 (0)	00:00:01
* 10	TABLE ACCESS BY INDEX ROWID	BCTL	1	24	1 (0)	00:00:01
* 11	INDEX UNIQUE SCAN	PK_BCTL	1		0 (0)	00:00:01

Predicate Information (identified by operation id):

```
   5 - filter("C"."POVERTY_STATUS"='1')
   7 - access("T"."BHDATE"=TO_DATE(' 2021[redacted]00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "T"."CUSTNO"="C"."CUSTCD")
       filter("T"."CUSTNO"="C"."CUSTCD")
   8 - filter(SUBSTR("L"."GUATTYPE",2,1)='1' AND INSTR(SUBSTR('L'."GUATTYPE",0,1),'1')=0 AND
              INSTR(SUBSTR("L"."GUATTYPE",3,3),'1')=0)
   9 - access("T"."LNCO"="L"."CONTRACTNO")
  10 - filter("B"."BRNO"<>'[redacted]5' AND "B"."BLN_UP_BRCODE"<>'[redacted]')
  11 - access("T"."BRCD"="B"."BRCODE")
```

最终执行计划可以看到，表join 顺序和join 方法都发生了改变

数据库和云讲座群

20-18

甲骨文云技术公众号

技术专家1V1深入交流