从一条SQL看基于ORACLE的SQL优化

——丁俊

个人简介

- 姓名：丁俊　网名:dingjun123
- DBA+社群联合发起人、ITPUB开发版版主、ITPUB社区专家、ChinaUnix BLOG专家，ITPUB 2010-2013连续4届最佳精华获得者、2011-2014连续4届最佳版主
- 电子工业出版社终身荣誉作者，《剑破冰山-Oracle开发艺术》副主编
- ORACLE ACE Associate　DBAplus　ACOUG All China Oracle User Group 中国Oracle用户组　ZJOUG MEMBER 中国浙江应用中间件与数据库用户组
- E-mail:dingjunlove@163.com
- Blog:http://blog.chinaunix.net/uid/7655508.html

内容简介

- 本次分享内容简介
  - 从一条巨慢SQL开始
  - 快速解读执行计划
  - 从执行计划看SQL低效根源
  - 统计信息与cardinality
  - 探索性能杀手FILTER
  - 逻辑重写让SQL起飞
  - 两个工具提升疑难SQL优化效率
  - SQL Tuning思考之RoadMap
  - SQL Tuning最佳实践

从一条巨慢的SQL的开始

预计执行12小时以上

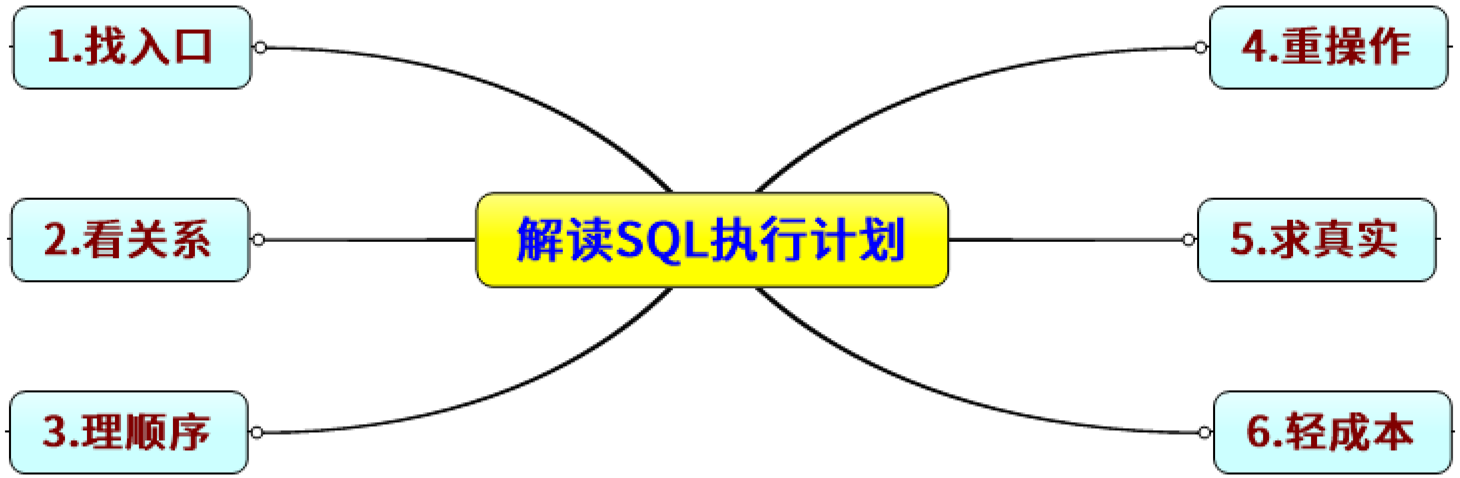
```sql
SELECT   TMISID,
         OTHER_CLASS,
         START_DATETIME,
         DEAL_DURATION,
         SUBSTR(TMISID, 2, 3),
         '201608'
   FROM DEALREC_ERR_201608 A
  WHERE (SUBSTR(TMISID, 1, 8) IN (SELECT I.BILID_HEAD
            FROM TMI_NO_INFOS I
           WHERE LENGTH(I.BILID_HEAD) = 8) OR
         SUBSTR(TMISID, 1, 9) IN (SELECT I.BILID_HEAD
            FROM TMI_NO_INFOS I
           WHERE LENGTH(I.BILID_HEAD) = 9) OR
         SUBSTR(TMISID, 1, 10) IN (SELECT  I.BILID_HEAD
            FROM TMI_NO_INFOS I
           WHERE LENGTH(I.BILID_HEAD) = 10) OR
         SUBSTR(TMISID, 1, 11) IN (SELECT I.BILID_HEAD
            FROM TMI_NO_INFOS I
           WHERE LENGTH(I.BILID_HEAD) = 11))
    AND SUBSTR(OTHER_CLASS, 1, 6) NOT IN
        (SELECT C.ORDINAR_CODE
           FROM B_DEALING_DONE_TYPE C
          WHERE C.DEALRECD_CODE IN ('1002', '1004')
            AND LENGTH(C.ORDINAR_CODE) = '6')
    AND SUBSTR(OTHER_CLASS, 1, 5) NOT IN
        (SELECT C.ORDINAR_CODE
           FROM B_DEALING_DONE_TYPE C
          WHERE C.DEALRECD_CODE IN ('1002', '1004')
            AND LENGTH(C.ORDINAR_CODE) = '5')
    AND SUBSTR(OTHER_CLASS, 1, 4) != '4007'
    AND SUBSTR(OTHER_CLASS, 1, 7) NOT IN
        (SELECT C.ORDINAR_CODE
           FROM B_DEALING_DONE_TYPE C
          WHERE C.DEALRECD_CODE IN ('1002', '1004')
            AND LENGTH(C.ORDINAR_CODE) = '7')
    AND SUBSTR(OTHER_CLASS, 1, 8) NOT IN
        (SELECT C.ORDINAR_CODE
           FROM B_DEALING_DONE_TYPE C
          WHERE C.DEALRECD_CODE IN ('1002', '1004')
            AND LENGTH(C.ORDINAR_CODE) = '8')
    AND SUBSTR(OTHER_CLASS, 1, 3) NOT IN
        ('147','151', ...)
```

性能杀手FILTER

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name                     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT                   |                          |       |       |   129K(100)|          |
| * 1 |  FILTER                            |                          |       |       |            |          |
|   2 |   NESTED LOOPS ANTI                |                          |     1 |    93 |   129K  (1)| 00:25:54 |
|   3 |    NESTED LOOPS ANTI               |                          |     1 |    80 |   129K  (1)| 00:25:54 |
|   4 |     NESTED LOOPS ANTI              |                          |     1 |    67 |   129K  (1)| 00:25:54 |
|   5 |      NESTED LOOPS ANTI             |                          |     1 |    54 |   129K  (1)| 00:25:54 |
| * 6 |       TABLE ACCESS FULL            | DEALREC_ERR_201608       |     1 |    41 |   129K  (1)| 00:25:54 |
| * 7 |       TABLE ACCESS BY INDEX ROWID  | B_DEALING_DONE_TYPE      |     1 |    13 |     2  (0)| 00:00:01 |
| * 8 |        INDEX RANGE SCAN            | IND_B_DEALING_DONE_TYPE  |     1 |       |     1  (0)| 00:00:01 |
| * 9 |      TABLE ACCESS BY INDEX ROWID   | B_DEALING_DONE_TYPE      |     1 |    13 |     2  (0)| 00:00:01 |
| * 10|       INDEX RANGE SCAN             | IND_B_DEALING_DONE_TYPE  |     1 |       |     1  (0)| 00:00:01 |
| * 11|     TABLE ACCESS BY INDEX ROWID    | B_DEALING_DONE_TYPE      |     1 |    13 |     2  (0)| 00:00:01 |
| * 12|      INDEX RANGE SCAN              | IND_B_DEALING_DONE_TYPE  |     1 |       |     1  (0)| 00:00:01 |
| * 13|    TABLE ACCESS BY INDEX ROWID     | B_DEALING_DONE_TYPE      |     1 |    13 |     2  (0)| 00:00:01 |
| * 14|     INDEX RANGE SCAN               | IND_B_DEALING_DONE_TYPE  |     1 |       |     1  (0)| 00:00:01 |
| * 15|   TABLE ACCESS FULL                | TMI_NO_INFOS             |     8 |    64 |     5  (0)| 00:00:01 |
| * 16|    TABLE ACCESS FULL               | TMI_NO_INFOS             |     8 |    64 |     5  (0)| 00:00:01 |
| * 17|     TABLE ACCESS FULL              | TMI_NO_INFOS             |     1 |     8 |     5  (0)| 00:00:01 |
| * 18|      TABLE ACCESS FULL             | TMI_NO_INFOS             |     1 |     8 |     5  (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   6 - filter((SUBSTR("OTHER_CLASS",1,4)<>'4007'  AND SUBSTR("OTHER_CLASS",1,3)<>'147'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'151'  AND SUBSTR("OTHER_CLASS",1,3)<>'187'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'157'  AND SUBSTR("OTHER_CLASS",1,3)<>'139'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'197'  AND SUBSTR("OTHER_CLASS",1,3)<>'068'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'134'  AND SUBSTR("OTHER_CLASS",1,3)<>'135'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'136'  AND SUBSTR("OTHER_CLASS",1,3)<>'137'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'138'  AND SUBSTR("OTHER_CLASS",1,3)<>'150'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'158'  AND SUBSTR("OTHER_CLASS",1,3)<>'159'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'188'  AND SUBSTR("OTHER_CLASS",1,3)<>'152'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'183'  AND SUBSTR("OTHER_CLASS",1,3)<>'184'  AND
              SUBSTR("OTHER_CLASS",1,3)<>'182'))
   7 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
   8 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,8))
       filter(LENGTH("C"."ORDINAR_CODE")=8)
   9 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  10 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,7))
       filter(LENGTH("C"."ORDINAR_CODE")=7)
  11 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  12 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,5))
       filter(LENGTH("C"."ORDINAR_CODE")=5)
  13 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  14 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,6))
       filter(LENGTH("C"."ORDINAR_CODE")=6)
  15 - filter((LENGTH("I"."BILID_HEAD")=8 AND "I"."BILID_HEAD"=SUBSTR(:B1,1,8)))
  16 - filter((LENGTH("I"."BILID_HEAD")=9 AND "I"."BILID_HEAD"=SUBSTR(:B1,1,9)))
  17 - filter((LENGTH("I"."BILID_HEAD")=10 AND "I"."BILID_HEAD"=SUBSTR(:B1,1,10)))
  18 - filter((LENGTH("I"."BILID_HEAD")=11 AND "I"."BILID_HEAD"=SUBSTR(:B1,1,11)))
```

关注cardinality准确性

快速解读执行计划要点



- **找入口**：通过最右最上最先执行原则找出执行计划入口操作（对于巨长执行计划COPY到UE里使用光标缩进下探法则可找出入口）。
- **看关系**：各操作之间的关系：NESED LOOPS、HASH JOIN、FILTER等是否准确，直接关系此操作甚至影响整个SQL的执行效率。
- **理顺序**：一步走错，满盘皆输。通过理清执行计划顺序找出key steps。
- **重操作**：执行计划中的Operation和Predicate部分是需要关注的核心内容，从操作中看出不合理部分，以此建立正确索引等优化措施。
- **求真实**：执行计划中指标是估算的，估算的指标和实际情况很可能不匹配。所以优化SQL需要了解每步骤真实的基数、真实执行时间和Buffer gets等，从而准确找出问题Root cause。（可以根据谓词手动计算、建议采用display_cursor方式获取A-ROWS、A-TIME等信息，工具有很多，也可以使用sql monitor等）
- **轻成本**：COST虽然是CBO的核心内容，但是因为执行计划中COST不一定准确反应SQL快慢，因此不要唯COST论，COST只是一个参考指标，当然可以通过执行计划判断一些COST是否明显存在问题，比如COST非常小，但是SQL执行很慢，可能就是统计信息不准确了。

执行计划是通向SQL性能优化的一把钥匙

快速解读执行计划实例

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                         | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT                  |                            |       |       |   129K(100)|          |
|*  1 |  FILTER                           |                            |       |       |            |          |
|   2 |   NESTED LOOPS ANTI               |                            |     1 |    93 |   129K  (1)| 00:25:54 |
|   3 |    NESTED LOOPS ANTI              |                            |     1 |    80 |   129K  (1)| 00:25:54 |
|   4 |     NESTED LOOPS ANTI             |                            |     1 |    67 |   129K  (1)| 00:25:54 |
|   5 |      NESTED LOOPS ANTI            |                            |     1 |    54 |   129K  (1)| 00:25:54 |
|*  6 |       TABLE ACCESS FULL           | DEALREC_ERR_201608         |     1 |    41 |   129K  (1)| 00:25:54 |
|*  7 |       TABLE ACCESS BY INDEX ROWID | B_DEALING_DONE_TYPE        |     1 |    13 |     2   (0)| 00:00:01 |
|*  8 |        INDEX RANGE SCAN           | IND_B_DEALING_DONE_TYPE    |     1 |       |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS BY INDEX ROWID  | B_DEALING_DONE_TYPE        |     1 |    13 |     2   (0)| 00:00:01 |
|* 10 |       INDEX RANGE SCAN            | IND_B_DEALING_DONE_TYPE    |     1 |       |     1   (0)| 00:00:01 |
|* 11 |     TABLE ACCESS BY INDEX ROWID   | B_DEALING_DONE_TYPE        |     1 |    13 |     2   (0)| 00:00:01 |
|* 12 |      INDEX RANGE SCAN             | IND_B_DEALING_DONE_TYPE    |     1 |       |     1   (0)| 00:00:01 |
|* 13 |    TABLE ACCESS BY INDEX ROWID    | B_DEALING_DONE_TYPE        |     1 |    13 |     2   (0)| 00:00:01 |
|* 14 |     INDEX RANGE SCAN              | IND_B_DEALING_DONE_TYPE    |     1 |       |     1   (0)| 00:00:01 |
|* 15 |   TABLE ACCESS FULL               | TMI_NO_INFOS               |     8 |    64 |     5   (0)| 00:00:01 |
|* 16 |    TABLE ACCESS FULL              | TMI_NO_INFOS               |     8 |    64 |     5   (0)| 00:00:01 |
|* 17 |     TABLE ACCESS FULL             | TMI_NO_INFOS               |     1 |     8 |     5   (0)| 00:00:01 |
|* 18 |      TABLE ACCESS FULL            | TMI_NO_INFOS               |     1 |     8 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

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

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   6 - filter((SUBSTR("OTHER_CLASS",1,4)<>'4007' AND SUBSTR("OTHER_CLASS",1,3)<>'147' AND
              SUBSTR("OTHER_CLASS",1,3)<>'151' AND SUBSTR("OTHER_CLASS",1,3)<>'187' AND
              SUBSTR("OTHER_CLASS",1,3)<>'157' AND SUBSTR("OTHER_CLASS",1,3)<>'139' AND
              SUBSTR("OTHER_CLASS",1,3)<>'197' AND SUBSTR("OTHER_CLASS",1,3)<>'068' AND
              SUBSTR("OTHER_CLASS",1,3)<>'134' AND SUBSTR("OTHER_CLASS",1,3)<>'135' AND
              SUBSTR("OTHER_CLASS",1,3)<>'136' AND SUBSTR("OTHER_CLASS",1,3)<>'137' AND
              SUBSTR("OTHER_CLASS",1,3)<>'138' AND SUBSTR("OTHER_CLASS",1,3)<>'150' AND
              SUBSTR("OTHER_CLASS",1,3)<>'158' AND SUBSTR("OTHER_CLASS",1,3)<>'159' AND
              SUBSTR("OTHER_CLASS",1,3)<>'188' AND SUBSTR("OTHER_CLASS",1,3)<>'152' AND
              SUBSTR("OTHER_CLASS",1,3)<>'183' AND SUBSTR("OTHER_CLASS",1,3)<>'184' AND
              SUBSTR("OTHER_CLASS",1,3)<>'182'))
   7 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
```

为什么要分析执行计划各步骤执行顺序和关系？

- 1.光标从Id=0开始，逐行缩进向下查找
- 2.找到Id=6的部分，发现无法缩进查找其子节点，只有同级节点Id=7,而Id=7的子节点是Id=8,找到Id=8的步骤，如果再缩进查找，发现已经没有，也就是说入口在Id=6到Id=8之间。由于6和7是同级的，6先执行，也就是说这条执行计划的入口步骤是Id=6部分，而与Id=6做Nested Loops anti的是Id=7步骤，而I=7步骤的子节点是Id=8步骤。
- 3.其它执行步骤顺序采用反向查找方法，按照同级别方式垂直方式查找。内部步骤的入口按照2的方式查找。比如Id=3步骤与Id=13是同一级别，他们的父操作是Id=2。

快速解读执行计划实例1

- 前面问题：为什么要寻找执行计划入口？为什么要分析执行计划各步骤顺序和关系？
  各种操作之间的关系是由cardinality等各种因素触发的，不正确的cardinality会导致本来应该走Hash Join的走了 Nested loops Join。往往入口处就有问题，导致后续执行计划全部错误，所以明确各种步骤的关系，有助于找出影响问题的根源步骤。
  理清执行计划顺序，有助于理解SQL内部的执行路径，通过执行的实际情况判断出不合理步骤操作。

- 重操作、求真实、轻成本是通过执行计划优化SQL的重要方法。

```
----------------------------------------------------------------------------------------------------------
| Id  | Operation                        | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT                 |                           |       |       |  129K(100)|          |
|*  1 |  FILTER                          |                           |       |       |            |          |
|   2 |   NESTED LOOPS ANTI              |                           |     1 |    93 |  129K   (1)| 00:25:54 |
|   3 |    NESTED LOOPS ANTI             |                           |     1 |    80 |  129K   (1)| 00:25:54 |
|   4 |     NESTED LOOPS ANTI            |                           |     1 |    67 |  129K   (1)| 00:25:54 |
|   5 |      NESTED LOOPS ANTI           |                           |     1 |    54 |  129K   (1)| 00:25:54 |
|*  6 |       TABLE ACCESS FULL          | DEALREC_ERR_201608        |     1 |    41 |  129K   (1)| 00:25:54 |
|*  7 |       TABLE ACCESS BY INDEX ROWID| B_DEALING_DONE_TYPE       |     1 |    13 |     2   (0)| 00:00:01 |
|*  8 |        INDEX RANGE SCAN          | IND_B_DEALING_DONE_TYPE   |     1 |       |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS BY INDEX ROWID | B_DEALING_DONE_TYPE       |     1 |    13 |     2   (0)| 00:00:01 |
|* 10 |       INDEX RANGE SCAN           | IND_B_DEALING_DONE_TYPE   |     1 |       |     1   (0)| 00:00:01 |
|* 11 |     TABLE ACCESS BY INDEX ROWID  | B_DEALING_DONE_TYPE       |     1 |    13 |     2   (0)| 00:00:01 |
|* 12 |      INDEX RANGE SCAN            | IND_B_DEALING_DONE_TYPE   |     1 |       |     1   (0)| 00:00:01 |
|* 13 |    TABLE ACCESS BY INDEX ROWID   | B_DEALING_DONE_TYPE       |     1 |    13 |     2   (0)| 00:00:01 |
|* 14 |     INDEX RANGE SCAN             | IND_B_DEALING_DONE_TYPE   |     1 |       |     1   (0)| 00:00:01 |
|* 15 |   TABLE ACCESS FULL              | TMI_NO_INFOS              |     8 |    64 |     5   (0)| 00:00:01 |
|* 16 |    TABLE ACCESS FULL             | TMI_NO_INFOS              |     8 |    64 |     5   (0)| 00:00:01 |
|* 17 |     TABLE ACCESS FULL            | TMI_NO_INFOS              |     1 |     8 |     5   (0)| 00:00:01 |
|* 18 |      TABLE ACCESS FULL           | TMI_NO_INFOS              |     1 |     8 |     5   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------------

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

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   6 - filter((SUBSTR("OTHER_CLASS",1,4)<>'4007' AND SUBSTR("OTHER_CLASS",1,3)<>'147' AND
              SUBSTR("OTHER_CLASS",1,3)<>'151' AND SUBSTR("OTHER_CLASS",1,3)<>'187' AND
              SUBSTR("OTHER_CLASS",1,3)<>'157' AND SUBSTR("OTHER_CLASS",1,3)<>'139' AND
              SUBSTR("OTHER_CLASS",1,3)<>'197' AND SUBSTR("OTHER_CLASS",1,3)<>'068' AND
              SUBSTR("OTHER_CLASS",1,3)<>'134' AND SUBSTR("OTHER_CLASS",1,3)<>'135' AND
              SUBSTR("OTHER_CLASS",1,3)<>'136' AND SUBSTR("OTHER_CLASS",1,3)<>'137' AND
              SUBSTR("OTHER_CLASS",1,3)<>'138' AND SUBSTR("OTHER_CLASS",1,3)<>'150' AND
              SUBSTR("OTHER_CLASS",1,3)<>'158' AND SUBSTR("OTHER_CLASS",1,3)<>'159' AND
              SUBSTR("OTHER_CLASS",1,3)<>'188' AND SUBSTR("OTHER_CLASS",1,3)<>'152' AND
              SUBSTR("OTHER_CLASS",1,3)<>'183' AND SUBSTR("OTHER_CLASS",1,3)<>'184' AND
```

看完执行计划，解决疑问

- 1.了解执行计划中的性能杀手操作：如NESTED LOOPS、FILTER等
- 2.Rows(cardinality)这么小，一堆NL操作，是不是不合理？
- 3.每步的真实cardinality情况到底是什么样的？真实的cardinality决定了步骤之间执行的操作关系以及执行次数！
- 4.如何调整执行计划，使其最大限度正确？

从执行计划看SQL低效根源

```
-------------------------------------------------------------------------------------------------------------
| Id | Operation                         | Name                     | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT                 |                          |      |       |  129K(100) |          |
| * 1 |  FILTER                          |                          |      |       |            |          |
|   2 |   NESTED LOOPS ANTI              |                          |    1 |    93 |  129K  (1) | 00:25:54 |
|   3 |    NESTED LOOPS ANTI             |                          |    1 |    80 |  129K  (1) | 00:25:54 |
|   4 |     NESTED LOOPS ANTI            |                          |    1 |    67 |  129K  (1) | 00:25:54 |
|   5 |      NESTED LOOPS ANTI           |                          |    1 |    54 |  129K  (1) | 00:25:54 |
| * 6 |       TABLE ACCESS FULL          | DEALREC_ERR_201608       |    1 |    41 |  129K  (1) | 00:25:54 |
| * 7 |       TABLE ACCESS BY INDEX ROWID| B_DEALING_DONE_TYPE      |    1 |    13 |    2   (0) | 00:00:01 |
| * 8 |        INDEX RANGE SCAN          | IND_B_DEALING_DONE_TYPE  |    1 |       |    1   (0) | 00:00:01 |
| * 9 |      TABLE ACCESS BY INDEX ROWID | B_DEALING_DONE_TYPE      |    1 |    13 |    2   (0) | 00:00:01 |
| * 10 |      INDEX RANGE SCAN           | IND_B_DEALING_DONE_TYPE  |    1 |       |    1   (0) | 00:00:01 |
| * 11 |     TABLE ACCESS BY INDEX ROWID | B_DEALING_DONE_TYPE      |    1 |    13 |    2   (0) | 00:00:01 |
| * 12 |      INDEX RANGE SCAN           | IND_B_DEALING_DONE_TYPE  |    1 |       |    1   (0) | 00:00:01 |
| * 13 |    TABLE ACCESS BY INDEX ROWID  | B_DEALING_DONE_TYPE      |    1 |    13 |    2   (0) | 00:00:01 |
| * 14 |     INDEX RANGE SCAN            | IND_B_DEALING_DONE_TYPE  |    1 |       |    1   (0) | 00:00:01 |
| * 15 |   TABLE ACCESS FULL             | TMI_NO_INFOS             |    8 |    64 |    5   (0) | 00:00:01 |
| * 16 |    TABLE ACCESS FULL            | TMI_NO_INFOS             |    8 |    64 |    5   (0) | 00:00:01 |
| * 17 |     TABLE ACCESS FULL           | TMI_NO_INFOS             |    1 |     8 |    5   (0) | 00:00:01 |
| * 18 |      TABLE ACCESS FULL          | TMI_NO_INFOS             |    1 |     8 |    5   (0) | 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   6 - filter((SUBSTR("OTHER_CLASS",1,4)<>'4007' AND SUBSTR("OTHER_CLASS",1,3)<>'147' AND
              SUBSTR("OTHER_CLASS",1,3)<>'151' AND SUBSTR("OTHER_CLASS",1,3)<>'187' AND
              SUBSTR("OTHER_CLASS",1,3)<>'157' AND SUBSTR("OTHER_CLASS",1,3)<>'139' AND
              SUBSTR("OTHER_CLASS",1,3)<>'197' AND SUBSTR("OTHER_CLASS",1,3)<>'068' AND
              SUBSTR("OTHER_CLASS",1,3)<>'134' AND SUBSTR("OTHER_CLASS",1,3)<>'135' AND
              SUBSTR("OTHER_CLASS",1,3)<>'136' AND SUBSTR("OTHER_CLASS",1,3)<>'137' AND
              SUBSTR("OTHER_CLASS",1,3)<>'138' AND SUBSTR("OTHER_CLASS",1,3)<>'150' AND
              SUBSTR("OTHER_CLASS",1,3)<>'158' AND SUBSTR("OTHER_CLASS",1,3)<>'159' AND
              SUBSTR("OTHER_CLASS",1,3)<>'188' AND SUBSTR("OTHER_CLASS",1,3)<>'152' AND
              SUBSTR("OTHER_CLASS",1,3)<>'183' AND SUBSTR("OTHER_CLASS",1,3)<>'184' AND
              SUBSTR("OTHER_CLASS",1,3)<>'182'))
   7 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
   8 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,8))
       filter(LENGTH("C"."ORDINAR_CODE")=8)
   9 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  10 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,7))
       filter(LENGTH("C"."ORDINAR_CODE")=7)
  11 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  12 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,5))
       filter(LENGTH("C"."ORDINAR_CODE")=5)
  13 - filter(("C"."DEALRECD_CODE"='1002' OR "C"."DEALRECD_CODE"='1004'))
  14 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,6))
```

性能杀手FILTER

关注cardinality准确性

```
SEGMENT_NAME                    SEGMENT_TYPE          SIZE_MB
------------------------------- ------------------- ----------
DEALREC_ERR_201608              TABLE                     4608
IDX_DEALREC_ERR_201608          INDEX                1780.1875

SEGMENT_NAME                    SEGMENT_TYPE          SIZE_MB
------------------------------- ------------------- ----------
TMI_NO_INFOS                    TABLE                     .125

SEGMENT_NAME                    SEGMENT_TYPE          SIZE_MB
------------------------------- ------------------- ----------
B_DEALING_DONE_TYPE            TABLE                      10
IDX_B_DEALING_DONE_TYPE        INDEX                 10.1875
```

- 主表DEALREC_ERR_201608在ID=6查询条件中经查要返回2000w行，计划中估算只有1行，因此，会导致NESTED LOOPS次数实际执行千万次，导致效率低下，应该走HASH JOIN，需要更新统计信息。
- 另外ID=1是FILTER,它的子节点是ID=2和ID=15、16、17、18，同样的ID 15-18也被驱动千万次。

找出问题根源后，逐步解决。

第一次分析：解决Id=6步骤估算的cardinality不准确问题

解决cardinality估算不准确问题

- 找出入口操作Id=6,由于Id=6操作的cardinality估算为1导致后续走一系列NESTED LOOPS影响效率。
- cardinality的计算与谓词紧密相关，所以要找出Id=6的谓词，根据谓词手动计算真实card与估算card之间的区别
- 尝试收集统计信息，检验效果

```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT             |                         |       |       |   129K(100)|          |
|*  1 |  FILTER                      |                         |       |       |            |          |
|   2 |   NESTED LOOPS ANTI          |                         |     1 |    93 |   129K  (1)| 00:25:54 |
|   3 |    NESTED LOOPS ANTI         |                         |     1 |    80 |   129K  (1)| 00:25:54 |
|   4 |     NESTED LOOPS ANTI        |                         |     1 |    67 |   129K  (1)| 00:25:54 |
|   5 |      NESTED LOOPS ANTI       |                         |     1 |    54 |   129K  (1)| 00:25:54 |
|*  6 |       TABLE ACCESS FULL      | DEALREC_ERR_201608      |     1 |    41 |   129K  (1)| 00:25:54 |
|*  7 |       TABLE ACCESS BY INDEX ROWID| B_DEALING_DONE_TYPE |     1 |    13 |     2  (0)| 00:00:01 |
|*  8 |        INDEX RANGE SCAN      | IND_B_DEALING_DONE_TYPE |     1 |       |     1  (0)| 00:00:01 |
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   6 - filter((SUBSTR("OTHER_CLASS",1,4)<>'4007' AND SUBSTR("OTHER_CLASS",1,3)<>'147' AND
              SUBSTR("OTHER_CLASS",1,3)<>'151' AND SUBSTR("OTHER_CLASS",1,3)<>'187' AND
              SUBSTR("OTHER_CLASS",1,3)<>'157' AND SUBSTR("OTHER_CLASS",1,3)<>'139' AND
              SUBSTR("OTHER_CLASS",1,3)<>'197' AND SUBSTR("OTHER_CLASS",1,3)<>'068' AND
              SUBSTR("OTHER_CLASS",1,3)<>'134' AND SUBSTR("OTHER_CLASS",1,3)<>'135' AND
              SUBSTR("OTHER_CLASS",1,3)<>'136' AND SUBSTR("OTHER_CLASS",1,3)<>'137' AND
              SUBSTR("OTHER_CLASS",1,3)<>'138' AND SUBSTR("OTHER_CLASS",1,3)<>'150' AND
              SUBSTR("OTHER_CLASS",1,3)<>'158' AND SUBSTR("OTHER_CLASS",1,3)<>'159' AND
              SUBSTR("OTHER_CLASS",1,3)<>'188' AND SUBSTR("OTHER_CLASS",1,3)<>'152' AND
              SUBSTR("OTHER_CLASS",1,3)<>'183' AND SUBSTR("OTHER_CLASS",1,3)<>'184' AND
              SUBSTR("OTHER_CLASS",1,3)<>'182'))
```

解决cardinality估算不准确问题-扩展统计信息收集

- 尝试更新统计信息：
  发现使用size auto,size repeat,对other_class收集直方图均无效果，执行计划中对other_class的查询条件返回行估算还是1（实际返回2000w行）.**如何解决？card的计算和谓词紧密相关，查看谓词：**
  substr(other_class, 1, 3) NOT IN (‘147’ , ‘151’ , …)

怎么办？思绪万千，灵光乍现!
Hints:cardinality(a,20000000)，use_hash等可以
还有更好的办法吗?
突然想起11g有个统计信息收集新特性：扩展统计信息收集

```
exec DBMS_STATS.GATHER_TABLE_STATS(ownname=> ‘xxx',tabname=> ‘DEALREC_ERR_201608',method_opt=>'for columns (substr(other_class, 1, 3)) size skewonly',estimate_percent=>10,no_invalidate=>false,cascade=>true,degree => 10);
```

OK

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	194	3798K (1)	12:39:46
* 1	FILTER					
* 2	HASH JOIN RIGHT ANTI		1226K	113M	129K (1)	00:25:54
* 3	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 4	HASH JOIN RIGHT ANTI		1227K	98M	129K (1)	00:25:54
* 5	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 6	HASH JOIN RIGHT ANTI		1229K	83M	129K (1)	00:25:54
* 7	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 8	HASH JOIN RIGHT ANTI		1231K	68M	129K (1)	00:25:53
* 9	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 10	TABLE ACCESS FULL	DEALREC_ERR_201608	1232K	52M	129K (1)	00:25:53
* 11	TABLE ACCESS FULL	TMI_NO_INFOS	8	64	3 (0)	00:00:01
* 12	TABLE ACCESS FULL	TMI_NO_INFOS	8	64	2 (0)	00:00:01
* 13	TABLE ACCESS FULL	TMI_NO_INFOS	1	8	5 (0)	00:00:01
* 14	TABLE ACCESS FULL	TMI_NO_INFOS	1	8	5 (0)	00:00:01

- DEALREC_ERR_201608与B_DEALING_DONE_TYPE原来走NL的现在正确走HASH JOIN。Build table是小结果集，probe table是ERR表大结果集，正确。
- 但是ID=2与ID=11到14，也就是与TMI_NO_INFOS的OR子查询，还是FILTER,驱动数千万次子节点查询，下一步优化要解决的问题。
- 性能从12小时到2小时。到这里结束了吗？

解决cardinality估算不准确问题-有关统计信息的那些疑问

- **疑问1：100%收集为什么还没有走正确执行计划?**
  统计信息收集比例高不代表就可以翻译对应谓词的特征，而且统计信息内部有很多算法限制以及不完善的情况，比如11g的扩展统计信息来继续完善，12c也有很多统计信息完善的特性，所以并不是比例低就不好，比例高就好！

- **疑问2：统计信息各种维度收集了包括直方图都收集了怎么不起作用?**
  直方图有很多限制，12c之前，只有频度直方图和等高直方图两种，对很多值的分布不能精确表示，所以有很多限制。因此,12c又增加了2种直方图：顶级频度直方图和混合直方图。另外直方图还有只存储前32位字符的限制。

- **疑问3：直方图只对走索引的有作用?**
  很显然不对，直方图只是反应数据的分布，数据的分布正确，对应谓词可以查询出比较准确的cardinality,从而影响执行计划，所以，对全表也是有用的。

- **疑问4：收集或更新了统计信息，执行计划怎么变得更差了?**
  很有可能，比如把原来的直方图给去掉了可能导致执行计划变差，因此，一般更新使用size repeat，除非确认需要修改某些直方图，另外谓词和统计信息紧密相关，某些谓词条件一旦收集统计信息，可能就计算不准确了。

- **疑问5：执行计划中cardinality显示的和已有统计信息计算不一致?**
  oracle cbo内部算法很复杂，而且BUG众多，遇到问题要大胆怀疑。

- **疑问6：统计信息应该按照Oracle建议自动收集?**
  具体问题具体分析，是让Oracle自动还是自己写脚本收集，都需要长期实践总结，对于一个复杂系统来说采样比例和method_opt很多需要定制设置。

- **疑问7：为什么唯一性很好的列，还需要收集直方图?**
  选择性的内部计算是要转成数字的：CBO内部计算选择性会先将字符串转为RAW，然后RAW转为数字，左起ROUND 15位。如果字符串的唯一性好， 但是计算成数字后唯一性不好，则会导致执行计划错误，这时候也需要收集直方图。

- **疑问8：我需要根据统计信息以及CBO公式去计算COST吗?**
  不需要，除非你很喜欢研究，得不偿失，了解各种JOIN算法、查询转换特性、索引等效率和哪些有关即可，COST不是最需要关心的指标，我们应该关心SQL高效运行所需的执行路径和执行方法，是否可以达到及早过滤大量数据，JOIN方法和顺序是否正确，是否可以建立高效访问对象等。

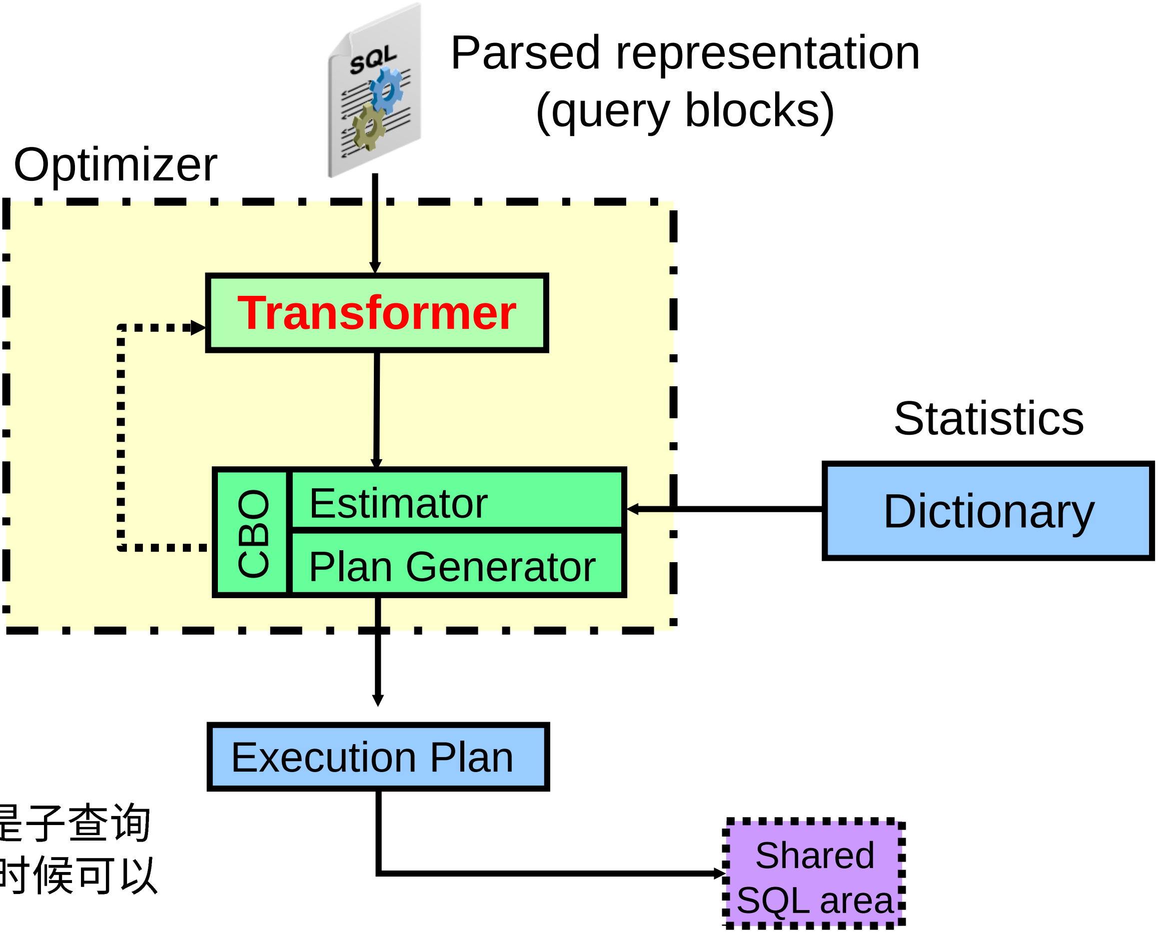
性能杀手FILTER形成机制

- 为什么会形成FILTER操作？（多子节点，单子节点纯粹过滤操作）

FILTER形成于查询转换期间，如果对于子查询无法进行 unnest转换来消除子查询，则会走FILTER，走FILTER说明子查询是受外表结果驱动，类似循环操作！很显然，如果驱动的次数越多，效率越低！

- FILTER什么时候高效？

FILTER本身会构建HASH表来保存输入/输出对，以备后续减少子查询执行次数，这是与纯粹NESED LOOPS操作的典型区别，比如from a where a.status In(select b.staus from b…) 如果status前面已经查过，则后续不需要再次执行子查询，而是直接从保存的HASH表中获取结果，这样减少了子查询执行次数，从而提高效率。也就是说，如果子查询关联条件的重复值很多，FILTER还是有一定的优势，否则就是灾难！



- FILTER与push_subq hints

如果走FILTER则子查询是受制于子查询外结果集驱动，也就是子查询是最后执行，但是实际有时候子查询应该先执行效率更好，这时候可以使用push_subq hints。

熟知各种查询转换规则，让自己成为优化器

性能杀手FILTER形成机制实例

- 简化前面的语句关键部分如下：

```
SELECT phone_no,ext,v1,padding
FROM t1
WHERE SUBSTR(t1.phone_no,1,8) IN
(SELECT t2.phone_no FROM t2 WHERE LENGTH(t2.phone_no)=8)
OR
SUBSTR(t1.phone_no,1,9) IN
(SELECT t2.phone_no FROM t2 WHERE LENGTH(t2.phone_no)=9)
OR
SUBSTR(t1.phone_no,1,10) IN
(SELECT t2.phone_no FROM t2 WHERE LENGTH(t2.phone_no)=10)
OR
SUBSTR(t1.phone_no,1,11) IN
(SELECT t2.phone_no FROM t2 WHERE LENGTH(t2.phone_no)=11);
```

- Oracle内部改写如下，无法unnest,如果unnest：

```
SELECT "T1"."PHONE_NO" "PHONE_NO",
       "T1"."EXT"      "EXT",
       "T1"."V1"       "V1",
       "T1"."PADDING"  "PADDING"
FROM   "DINGJUN123"."T1" "T1"
WHERE  EXISTS (SELECT 0
        FROM   "DINGJUN123"."T2" "T2"
        WHERE  "T2"."PHONE_NO" = SUBSTR("T1"."PHONE_NO", 1, 8)
               AND LENGTH("T2"."PHONE_NO") = 8)
       OR EXISTS
 (SELECT 0
        FROM   "DINGJUN123"."T2" "T2"
        WHERE  "T2"."PHONE_NO" = SUBSTR("T1"."PHONE_NO", 1, 9)
               AND LENGTH("T2"."PHONE_NO") = 9)
       OR EXISTS
 (SELECT 0
        FROM   "DINGJUN123"."T2" "T2"
        WHERE  "T2"."PHONE_NO" = SUBSTR("T1"."PHONE_NO", 1, 10)
               AND LENGTH("T2"."PHONE_NO") = 10)
       OR EXISTS
 (SELECT 0
        FROM   "DINGJUN123"."T2" "T2"
        WHERE  "T2"."PHONE_NO" = SUBSTR("T1"."PHONE_NO", 1, 11)
               AND LENGTH("T2"."PHONE_NO") = 11)
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 2055931425

---------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     1 |   127 |   25M   (1)| 85:10:23 |
|*  1 |  FILTER              |      |       |       |            |          |
|   2 |   TABLE ACCESS FULL  | T1   | 73481 | 9113K |   379   (1)| 00:00:05 |
|*  3 |   TABLE ACCESS FULL  | T2   |     1 |    12 |   373   (1)| 00:00:05 |
|*  4 |   TABLE ACCESS FULL  | T2   |     1 |    12 |   373   (1)| 00:00:05 |
|*  5 |   TABLE ACCESS FULL  | T2   |     1 |    12 |   373   (1)| 00:00:05 |
|*  6 |   TABLE ACCESS FULL  | T2   |     1 |    12 |   373   (1)| 00:00:05 |
---------------------------------------------------------------------------

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

   1 - filter( EXISTS (SELECT 0 FROM "T2" "T2" WHERE
              LENGTH("T2"."PHONE_NO")=8 AND "T2"."PHONE_NO"=SUBSTR(:B1,1,8)) OR
              EXISTS (SELECT 0 FROM "T2" "T2" WHERE LENGTH("T2"."PHONE_NO")=9 AND
              "T2"."PHONE_NO"=SUBSTR(:B2,1,9)) OR  EXISTS (SELECT 0 FROM "T2" "T2"
              WHERE LENGTH("T2"."PHONE_NO")=10 AND "T2"."PHONE_NO"=SUBSTR(:B3,1,10))
              OR  EXISTS (SELECT 0 FROM "T2" "T2" WHERE LENGTH("T2"."PHONE_NO")=11
              AND "T2"."PHONE_NO"=SUBSTR(:B4,1,11)))
   3 - filter(LENGTH("T2"."PHONE_NO")=8 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,8))
   4 - filter(LENGTH("T2"."PHONE_NO")=9 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,9))
   5 - filter(LENGTH("T2"."PHONE_NO")=10 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,10))
   6 - filter(LENGTH("T2"."PHONE_NO")=11 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,11))
```

特征：

1.重写为EXISTS

2.自定义绑定变量

性能杀手FILTER形成机制实例1

- 含有OR的子查询，经常性无法unnest,Oracle大多无法给转换成UNION/UNION ALL形式的查询
- 所以，针对这样的语句优化：
  1.改写为UNION/UNION ALL形式
  2.根据语义、业务含义彻底重写

也就是说，需要重构查询，消除FILTER！慢的根源如下，这里7万多行，只执行了116行打印的执行计划！Id=3~6的执行次数依赖于Id=2的结果行数，id=3~6全表扫描次数太多。

```
-----------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      0 |00:00:00.01  |       0 |
|*  1 |  FILTER             |      |      1 |        |      0 |00:00:00.01  |       0 |
|   2 |   TABLE ACCESS FULL | T1   |      1 |  73481 |    116 |00:00:00.01  |       5 |
|*  3 |   TABLE ACCESS FULL | T2   |    116 |      1 |      0 |00:00:03.75  |    154K |
|*  4 |   TABLE ACCESS FULL | T2   |    115 |      1 |      0 |00:00:03.88  |    154K |
|*  5 |   TABLE ACCESS FULL | T2   |    115 |      1 |      0 |00:00:05.11  |    154K |
|*  6 |   TABLE ACCESS FULL | T2   |    115 |      1 |      0 |00:00:05.28  |    154K |
-----------------------------------------------------------------------------------------

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

   1 - filter(( IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL OR  IS NOT NULL))
   3 - filter((LENGTH("T2"."PHONE_NO")=8 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,8)))
   4 - filter((LENGTH("T2"."PHONE_NO")=9 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,9)))
   5 - filter((LENGTH("T2"."PHONE_NO")=10 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,10)))
   6 - filter((LENGTH("T2"."PHONE_NO")=11 AND
              "T2"."PHONE_NO"=SUBSTR(:B1,1,11)))
```

逻辑改写-构造高效HASH JOIN代替低效FILTER

- 回到原来的SQL中，看如何改写

```
FROM DEALREC_ERR_201608 A
WHERE (SUBSTR(TMISID, 1, 8) IN (SELECT I.BILID_HEAD
         FROM TMI_NO_INFOS I
        WHERE LENGTH(I.BILID_HEAD) = 8) OR
      SUBSTR(TMISID, 1, 9) IN (SELECT I.BILID_HEAD
         FROM TMI_NO_INFOS I
        WHERE LENGTH(I.BILID_HEAD) = 9) OR
      SUBSTR(TMISID, 1, 10) IN (SELECT  I.BILID_HEAD
         FROM TMI_NO_INFOS I
        WHERE LENGTH(I.BILID_HEAD) = 10) OR
      SUBSTR(TMISID, 1, 11) IN (SELECT I.BILID_HEAD
         FROM TMI_NO_INFOS I
        WHERE LENGTH(I.BILID_HEAD) = 11))
```

```
select TMISID,
       OTHER_CLASS,
       START_DATETIME,
       DEAL_DURATION,
       substr(TMISID, 2, 3),
       '201608'
  from DEALREC_ERR_201608 a,(select distinct BILID_HEAD  from TMI_NO_INFOS) i
 where
   ((SUBSTR(a.TMISID, 1, 8) = i.BILID_HEAD AND LENGTH(i.BILID_HEAD)=8)
   OR (SUBSTR(a.TMISID, 1, 9) = i.BILID_HEAD AND LENGTH(i.BILID_HEAD)=9)
   OR (SUBSTR(a.TMISID, 1, 10) = i.BILID_HEAD AND LENGTH(i.BILID_HEAD)=10)
   OR (SUBSTR(a.TMISID, 1, 11) = i.BILID_HEAD AND LENGTH(i.BILID_HEAD)=11)
   )
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		88M	8815M	518K (1)	01:43:45
1	CONCATENATION					
2	MERGE JOIN ANTI		22M	2204M	129K (1)	00:25:57
3	SORT JOIN		22M	1933M	129K (1)	00:25:56
* 4	HASH JOIN		22M	1933M	129K (1)	00:25:56
* 5	VIEW		1795	14360	6 (17)	00:00:01
6	HASH UNIQUE		1795	14360	6 (17)	00:00:01
7	TABLE ACCESS FULL	TMI_NO_INFOS	1795	14360	5 (0)	00:00:01
* 8	HASH JOIN RIGHT ANTI		1227K	98M	129K (1)	00:25:56
* 9	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 10	HASH JOIN RIGHT ANTI		1229K	83M	129K (1)	00:25:55
* 11	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 12	HASH JOIN RIGHT ANTI		1231K	68M	129K (1)	00:25:55
* 13	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 14	TABLE ACCESS FULL	DEALREC_ERR_201608	1232K	52M	129K (1)	00:25:55
* 15	SORT UNIQUE		3	39	16 (7)	00:00:01
* 16	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
17	MERGE JOIN ANTI		22M	2203M	129K (1)	00:25:57
18	SORT JOIN		22M	1933M	129K (1)	00:25:56
* 19	HASH JOIN		22M	1933M	129K (1)	00:25:56
* 20	VIEW		1795	14360	6 (17)	00:00:01
21	HASH UNIQUE		1795	14360	6 (17)	00:00:01
22	TABLE ACCESS FULL	TMI_NO_INFOS	1795	14360	5 (0)	00:00:01
* 23	HASH JOIN RIGHT ANTI		1227K	98M	129K (1)	00:25:56

OR扩展拆分为4个查询最后联合，FILTER消失

- 改写后执行时间从2小时到8分钟，返回360w行+。虽然执行计划更复杂了，但是充分利用了HASH JOIN,MERGE JOIN这种大数据量处理算法代替原来的FILTER，更高效。如果不走OR扩展走什么？（走NESTED LOOPS，对IMS_NUM_INFO扫描从4次到1次，也很慢）
- OR扩展存在缺点，大表还是多次被访问，还能继续优化吗？

彻底重写-消除OR扩展的HASH JOIN重写思路

- 上一次重写，等于使用了第一种方法，用UNION/UNION ALL消除FILTER，那么如何消除UNION/UNION ALL呢，也就是要将OR语句合并为AND!

```
DEALREC_ERR_201608 a
where (substr(TMISID, 1, 8) in
      (select i.BILID_HEAD
          from TMI_NO_INFOS i
         where length(i.BILID_HEAD) = 8) or
      substr(TMISID, 1, 9) in
      (select i.BILID_HEAD
          from TMI_NO_INFOS i
         where length(i.BILID_HEAD) = 9) or
      substr(TMISID, 1, 10) in
      (select i.BILID_HEAD
          from TMI_NO_INFOS i
         where length(i.BILID_HEAD) = 10) or
      substr(TMISID, 1, 11) in
      (select i.BILID_HEAD
          from TMI_NO_INFOS i
         where length(i.BILID_HEAD) = 11))
```

上面含义是ERR表的TMISID截取前8,9,10,11位与TMI_NO_INFOS.BILLID_HEAD匹配，对应匹配BILLID_HEAD长度正好为8,9,10,11。很显然，语义上可以这样改写：
ERR表与TMI_NO_INFOS表关联，ERR.TMISID前8位与ITMI_NO_INFOS.BILLID_HEAD长度在8-11之间的前8位完全匹配，在此前提下，TMISID like BILLID_HEAD || ‘%’ 。
现在就动手彻底改变多个OR子查询，让SQL更加精简，效率更高。

彻底重写-消除OR扩展的HASH JOIN让SQL起飞

```
select TMISID,
       OTHER_CLASS,
       START_DATETIME,
       DEAL_DURATION,
       substr(TMISID, 2, 3),
       '201608'
  from DEALREC_ERR_201608 a,(select distinct BILID_HEAD  from TMI_NO_INFOS) i
 where
   a.TMISID like i.BILID_HEAD||'%'
   and substr(a.TMISID,1,8)=substr(i.BILID_HEAD,1,8)
   and length(i.BILID_HEAD) between 8 and 11
   AND substr(OTHER_CLASS, 1, 6) NOT IN
       (select C.ORDINAR_CODE
          from b_DEALING_DONE_TYPE c
         where c.DEALRECD_CODE in ('1002', '1004')
           and length(c.ORDINAR_CODE) = '6')
   AND substr(OTHER_CLASS, 1, 5) NOT IN
       (select C.ORDINAR_CODE
          from b_DEALING_DONE_TYPE c
         where c.DEALRECD_CODE in ('1002', '1004')
           and length(c.ORDINAR_CODE) = '5')
   AND substr(OTHER_CLASS, 1, 4) != '4007'
   AND substr(OTHER_CLASS, 1, 7) NOT IN
       (select C.ORDINAR_CODE
          from b_DEALING_DONE_TYPE c
         where c.DEALRECD_CODE in ('1002', '1004')
           and length(c.ORDINAR_CODE) = '7')
   AND substr(OTHER_CLASS, 1, 8) NOT IN
       (select C.ORDINAR_CODE
          from b_DEALING_DONE_TYPE c
         where c.DEALRECD_CODE in ('1002', '1004')
           and length(c.ORDINAR_CODE) = '8')
   AND substr(OTHER_CLASS, 1, 3) NOT IN
```

构造HASH JOIN关键

```
3699312 rows selected.
Elapsed: 00:03:09.63

Execution Plan
----------------------------------------------------------
Plan hash value: 809213444
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1042K	104M	129K (1)	00:25:57
* 1	HASH JOIN RIGHT ANTI		1042K	104M	129K (1)	00:25:57
* 2	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 3	HASH JOIN RIGHT ANTI		1043K	91M	129K (1)	00:25:56
* 4	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 5	HASH JOIN RIGHT ANTI		1045K	78M	129K (1)	00:25:56
* 6	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 7	HASH JOIN RIGHT ANTI		1046K	65M	129K (1)	00:25:56
* 8	TABLE ACCESS FULL	B_DEALING_DONE_TYPE	3	39	15 (0)	00:00:01
* 9	HASH JOIN		1047K	52M	129K (1)	00:25:56
10	VIEW		1700	13600	6 (17)	00:00:01
11	HASH UNIQUE		1700	13600	6 (17)	00:00:01
* 12	TABLE ACCESS FULL	TMI_NO_INFOS	1700	13600	5 (0)	00:00:01
* 13	TABLE ACCESS FULL	DEALREC_ERR_201608	1232K	52M	129K (1)	00:25:55

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

   1 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,6))
   2 - filter(LENGTH("C"."ORDINAR_CODE")=6 AND ("C"."DEALRECD_CODE"='1002' OR
              "C"."DEALRECD_CODE"='1004'))
   3 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,5))
   4 - filter(LENGTH("C"."ORDINAR_CODE")=5 AND ("C"."DEALRECD_CODE"='1002' OR
              "C"."DEALRECD_CODE"='1004'))
   5 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,7))
   6 - filter(LENGTH("C"."ORDINAR_CODE")=7 AND ("C"."DEALRECD_CODE"='1002' OR
              "C"."DEALRECD_CODE"='1004'))
   7 - access("C"."ORDINAR_CODE"=SUBSTR("OTHER_CLASS",1,8))
   8 - filter(LENGTH("C"."ORDINAR_CODE")=8 AND ("C"."DEALRECD_CODE"='1002' OR
              "C"."DEALRECD_CODE"='1004'))
   9 - access(SUBSTR("A"."TMISID",1,8)=SUBSTR("I"."BILID_HEAD",1,8))
       filter("A"."TMISID" LIKE "I"."BILID_HEAD"||'%')
  12 - filter(LENGTH("BILID_HEAD")>=8 AND LENGTH("BILID_HEAD")<=11)
```

- 现在的执行计划终于变的更短，更易读，通过逻辑改写走了HASH JOIN，那速度，杠杠的，最终一条返回300多万行数据的SQL原先需要12小时运行的SQL，现在3分钟就执行完了。
- **思考：**结构良好，语义清晰的SQL编写，有助于优化器选择更合理的执行计划，看来编写SQL真的有很多值得注意的地方。

两个工具提升疑难SQL优化效率-10053分析执行计划生成原因

- 一条SQL执行12分钟没有结果：
其中object_id有索引，从查询结构来看，内层查询完全可以独立执行（最多100行），然后与外层的表进行关联，走NL，这样可以利用到object_id索引，然而，事与愿违，Id=4出现FILTER，这样内层查询会驱动N次，问题出在何处?

```
select *
from(
select rowid,t.*
from t where t.object_id in
(
  select object_id
  from(
    select object_id
    from t
    where mod(object_id,10)=0
    and status='VALID'
    and last_ddl_time > trunc(sysdate-200)
    order by timestamp,last_ddl_time
    ) where rownum<=100
)   and t.status='VALID'
    and t.last_ddl_time > trunc(sysdate-200)
    order by last_ddl_time
) where rownum<=100;
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3028954274

-------------------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |   219 |  2100K  (2)| 07:00:08 |
|*  1 |  COUNT STOPKEY              |      |       |       |            |          |
|   2 |   VIEW                      |      |     1 |   219 |  2100K  (2)| 07:00:08 |
|*  3 |    SORT ORDER BY STOPKEY    |      |     1 |   100 |  2100K  (2)| 07:00:08 |
|*  4 |     FILTER                  |      |       |       |            |          |
|*  5 |      TABLE ACCESS FULL      | T    |  4936 |  482K |   855   (3)| 00:00:11 |
|*  6 |      FILTER                 |      |       |       |            |          |
|*  7 |       COUNT STOPKEY         |      |       |       |            |          |
|   8 |        VIEW                 |      |    49 |   637 |   851   (2)| 00:00:11 |
|*  9 |         SORT ORDER BY STOPKEY|     |    49 |  1960 |   851   (2)| 00:00:11 |
|* 10 |          TABLE ACCESS FULL  | T    |    49 |  1960 |   850   (2)| 00:00:11 |
-------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM<=100)
   3 - filter(ROWNUM<=100)
   4 - filter( EXISTS (<not feasible>)
   5 - filter("T"."STATUS"='VALID' AND
              "T"."LAST_DDL_TIME">TRUNC(SYSDATE@!-200))
   6 - filter("OBJECT_ID"=:B1)
   7 - filter(ROWNUM<=100)
   9 - filter(ROWNUM<=100)
  10 - filter("STATUS"='VALID' AND MOD("OBJECT_ID",10)=0 AND
```

使用10053探索优化器行为是研究复杂问题的重要方法

两个工具提升疑难SQL优化效率-10053分析执行计划生成原因1

```
*****************************
Cost-Based Subquery Unnesting
*****************************
SU: Unnesting query blocks in query block SEL$1 (#1) that are valid to unnest.
Subquery removal for query block SEL$3 (#3)
RSW: Not valid for subquery removal SEL$3 (#3)
Subquery unchanged.
Subquery Unnesting on query block SEL$2 (#2)SU: Performing unnesting that does not require costing.
SU: Considering subquery unnest on query block SEL$2 (#2).
SU:   Checking validity of unnesting subquery SEL$3 (#3
SU:     SU bypassed: Subquery in a view with rowid reference.
SU:   Validity checks failed.
```

- 从10053中可以看出，查询转换失败，因为遇到了rowid,当然把rowid改别名是可以，但是此SQL要求必须用rowid名字

- 通过改写消除FILTER运算如下：

```sql
select rd as "ROWID",object_id,object_name,last_ddl_time
from(
select rowid rd,t.*
from t where t.object_id in
(
  select object_id
  from(
    select object_id
    from t
    where mod(object_id,10)=0
    and status='VALID'
    and last_ddl_time > trunc(sysdate-200)
    order by timestamp,last_ddl_time
    ) where rownum<=100
)
    and t.status='VALID'
    and t.last_ddl_time > trunc(sysdate-200)
    order by last_ddl_time
) where rownum<=100;
```

内层查询rowid取别名

外层改回ROWID，注意双引号

```
Elapsed: 00:00:00.26

Execution Plan
----------------------------------------------------------
Plan hash value: 16082276

-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |    49 |  4900 |   931   (2)| 00:00:12 |
|*  1 |  COUNT STOPKEY                     |          |       |       |            |          |
|   2 |   VIEW                             |          |    49 |  4900 |   931   (2)| 00:00:12 |
|*  3 |    SORT ORDER BY STOPKEY           |          |    49 |  5145 |   931   (2)| 00:00:12 |
|*  4 |     TABLE ACCESS BY INDEX ROWID    | T        |     1 |   100 |     2   (0)| 00:00:01 |
|   5 |      NESTED LOOPS                  |          |    49 |  5145 |   930   (2)| 00:00:12 |
|   6 |       VIEW                         | VW_NSO_1 |    49 |   245 |   851   (2)| 00:00:11 |
|   7 |        HASH UNIQUE                 |          |    49 |   245 |            |          |
|*  8 |         COUNT STOPKEY              |          |       |       |            |          |
|   9 |          VIEW                      |          |    49 |   245 |   851   (2)| 00:00:11 |
|* 10 |           SORT ORDER BY STOPKEY    |          |    49 |  1960 |   851   (2)| 00:00:11 |
|* 11 |            TABLE ACCESS FULL       | T        |    49 |  1960 |   850   (2)| 00:00:11 |
|* 12 |       INDEX RANGE SCAN             | IDX_T    |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM<=100)
   3 - filter(ROWNUM<=100)
   4 - filter("T"."STATUS"='VALID' AND "T"."LAST_DDL_TIME">TRUNC(SYSDATE@!-200))
   8 - filter(ROWNUM<=100)
  10 - filter(ROWNUM<=100)
  11 - filter("STATUS"='VALID' AND MOD("OBJECT_ID",10)=0 AND
              "LAST_DDL_TIME">TRUNC(SYSDATE@!-200))
  12 - access("T"."OBJECT_ID"="OBJECT_ID")
```

两个工具提升SQL优化效率-SQLT找出正确执行计划需要设置的参数

- SQL能否生成正确执行计划，不光和统计信息、索引等有关，能否正确执行查询转换是至关重要的，由于各种复杂的查询转换机制导致BUG很多，Oracle对这些已知BUG通过fix control参数管理，有的默认打开，有的默认关闭。所以，如果遇到复杂的SQL，特别包含复杂视图的SQL，比如谓词无法推入这种查询转换，收集统计信息无效，这时候可以考虑是否遇到了BUG。
- BUG那么多，我怎么知道是哪个？，SQLT神器来帮你，使用SQLT里面的XPLORE工具，可以把参数打开关闭一遍，并且生成对应执行计划，这样通过生成的报告，可以一眼定位问题。（当然，是已知BUG，比如前面的rowid问题，也是定位不到的）

•问题背景：11.2.0.2升级到11.2.0.4出现此问题，性能杀手FILTER操作，SQL跑不出来，FILTER产生原因，无法unnest subquery

```
SELECT ID, TBILL_ID, TLE_CATEG_ID, INSERT_TIME, REMARK1
FROM     (SELECT A.ID, A.TBILL_ID, A.TLE_CATEG_ID, A.INSERT_TIME, A.REMARK1
          FROM   DT_MBY_TEST_LOG A,
                 (SELECT TBILL_ID, MIN(INSERT_TIME) AS INSERT_TIME
                  FROM   DT_MBY_TEST_LOG
                  WHERE  INSERT_TIME > '08-APR-15'
                         AND ID NOT IN (SELECT IMEI
                                        FROM   MM_TL_LOG_201607
                                        WHERE  STAND = '5')
                  GROUP  BY TBILL_ID) B
          WHERE  A.TBILL_ID = B.TBILL_ID
                 AND A.INSERT_TIME = B.INSERT_TIME
                 AND A.ID NOT IN (SELECT IMEI
                                  FROM   MM_TL_LOG_201607
                                  WHERE  STAND = '5')
          ORDER  BY INSERT_TIME)
WHERE  ROWNUM < 200
```

```
NAME                                       VALUE
------------------------------------------ ---------
_optimizer_null_aware_antijoin             TRUE
```

```
Execution Plan
----------------------------------------------------------

 Plan hash value: 1234634920

-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name             | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                  |   199 | 20099 |       | 22422   (1)| 00:04:30 |
|*  1 |  COUNT STOPKEY                     |                  |       |       |       |            |          |
|   2 |   VIEW                             |                  |  715K |   68M |       | 22422   (1)| 00:04:30 |
|*  3 |    SORT ORDER BY STOPKEY           |                  |  715K |   36M |   46M | 22422   (1)| 00:04:30 |
|*  4 |     FILTER                         |                  |       |       |       |            |          |
|*  5 |      HASH JOIN                     |                  |  715K |   36M |   22M | 12939   (1)| 00:02:36 |
|   6 |       VIEW                         |                  |  715K |   14M |       |  7907   (1)| 00:01:35 |
|   7 |        HASH GROUP BY               |                  |  715K |   17M |   40M |  7907   (1)| 00:01:35 |
|*  8 |         FILTER                     |                  |       |       |       |            |          |
|*  9 |          TABLE ACCESS FULL         | DT_MBY_TEST_LOG  | 1172K |   29M |       |  1419   (2)| 00:00:18 |
|* 10 |          TABLE ACCESS BY INDEX ROWID| MM_TL_LOG_201607 |     1 |     4 |       |     3   (0)| 00:00:01 |
|* 11 |           INDEX RANGE SCAN         | IDX_STAND        | 1172K |       |       |     2   (0)| 00:00:01 |
|  12 |       TABLE ACCESS FULL            | DT_MBY_TEST_LOG  | 1172K |   36M |       |  1410   (1)| 00:00:17 |
|* 13 |      TABLE ACCESS BY INDEX ROWID   | MM_TL_LOG_201607 |     1 |     4 |       |     3   (0)| 00:00:01 |
|* 14 |       INDEX RANGE SCAN             | IDX_STAND        | 1172K |       |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM<200)
   3 - filter(ROWNUM<200)
   4 - filter( NOT EXISTS (SELECT 0 FROM "ZYSOFT"."MM_TL_LOG_201607" "MM_TL_LOG_201607" WHERE
              "STAND"='5' AND LNNVL(TO_NUMBER("IMEI")<>:B1)))
   5 - access("A"."TBILL_ID"="B"."TBILL_ID" AND "A"."INSERT_TIME"="B"."INSERT_TIME")
   8 - filter( NOT EXISTS (SELECT 0 FROM "ZYSOFT"."MM_TL_LOG_201607" "MM_TL_LOG_201607" WHERE
              "STAND"='5' AND LNNVL(TO_NUMBER("IMEI")<>:B1)))
   9 - filter("INSERT_TIME">'08-APR-15')
  10 - filter(LNNVL(TO_NUMBER("IMEI")<>:B1))
  11 - access("STAND"='5')
  13 - filter(LNNVL(TO_NUMBER("IMEI")<>:B1))
  14 - access("STAND"='5')
```

DBAplus

两个工具提升SQL优化效率-SQLT找出正确执行计划需要设置的参数1

☆ SQLT 使用指南 (文档 ID 1677588.1)

215187.1SQLTXPLAIN (SQLT) 12.1.06 2014年1月30日
帮助诊断性能较差的 SQL 语句的工具

	主要方法	特殊方法	高级方法和模块
• SQLT 概览 • 安全模式 • 安装 SQLT • 卸载 SQLT • 升级 SQLT • 常见问题 • 新增功能！ • 上传SQLT文件给Oracle技术支持	• XTRACT • XECUTE • XTRXEC • XPLAIN • XTRSBY • XPREXT • XPREXC	• COMPARE • TRCANLZR • TRCAXTR • TRCASPLIT • XTRSET	• PROFILE • XGRAM • XPLORE • XHUME

XPLORE 模块

如果在数据库升级后 SQL 开始性能变差或者它可能产生明显的错误结果，那么使用 XPLORE 模块将有所帮助。如果将 optimizer_features_enable OFE 切换到升级之前的数据库版本，SQL 重新执行正常或者产生不同的结果，您可以使用此 XPLORE 模块尝试标识哪个特定 Optimizer 功能或修复引入了未预期的行为。确定特定故障有助于进一步故障排除或者对此特定功能和（或）修复执行更详细的研究。

此模块通过切换初始化参数和 fix control 参数来搜索计划。

仅当满足以下所有条件时才使用 XPLORE：

1. 当使用“差”计划时，SQL 执行性能差或者返回错误结果。
2. 可以在测试系统上重新生成差计划（最好没有真实数据）。
3. 可以通过切换 OFE 在测试系统上重新生成“好”计划。
4. 您需要将原因范围缩小到特定参数或 bug fix control。
5. 您对测试系统具有完全访问权限，包括 SYS 访问权限。

当符合以下任一条件时不要使用 XPLORE：

1. SQL 语句可能导致数据损坏或被更新。
2. 在 SQL 引用的表中存在大量数据。
3. 执行 SQL 需要的时间可能超过几秒钟。

要安装和使用该 XPLORE 模块，请阅读相应的 sqlt/utl/xplore/readme.txt。

两个工具提升SQL优化效率-SQLT找出正确执行计划需要设置的参数2

Discovered Plans

Plans for each test have been captured into DBMON.SQL_PLAN_STATISTICS_ALL or DBMON.PLAN_TABLE_ALL.

#	Plan Hash Value	SQLT Plan Hash Value[1]	SQLT Plan Hash Value2[1]	Total Tests	Plan Cost	Tests	Max Buffer Gets	Min Buffer Gets	Max CPU (secs)	Min CPU (secs)	Max Disk Reads	Min Disk Reads	Max ET[2] (secs)	Min ET[2] (secs)	Max Actual Rows	Min Actual Rows	Max Estim Rows	Min Estim Rows	B[3]	F[4]
1	985866816	19893	94597	5	22422 54485	1 4	77	69	.024	.018	0	0	.024	.018	0	0	199	199	199	5
2	1234634920	5061	10109	1	22422	1	80	80	.025	.025	0	0	.026	.026	0	0	199	199	199	1
3	1234634920	5061	14513	1	22422	1	97	97	.026	.026	0	0	.028	.028	0	0	199	199	199	1
4	1234634920	5061	73959	1	22422	1	170	170	.03	.03	0	0	.031	.031	0	0	199	199	199	1
5	1234634920	5061	79765	1156	3285 12861 12943 13637 21840 21922 22090 22413 22418 22419 22420 22422 22424 22432 22448 22909 23090 24422 25116 38453 45006 54485	1 1 2 1 1 1 1 1 3 1 4 1118 1 1 1 1 2 2 1 1 2 8	311445													
6	1234634920	14501	93486	1	22422	1	75	75	.021	.021	0	0	.022	.022	0	0	199	199	199	1
7	1234634920	82563	57267	2	22422	2	81	80	.024	.022	0	0	.026	.025	0	0	199	199	199	2
8	1523566366	27311	79700	1	61814	1	74	74	.022	.022	0	0	.023	.023	0	0	199	199	199	1
9	3399246315	50999	6221	16	1592 1594 1649 15828	2 2 10 2	79	61	.052	.014	0	0	.055	.016	0	0	1	1	1	16
10	3700521601	36167	91389	2	22420	2	63	63	.023	.019	0	0	.025	.02	0	0	199	199	199	2
11	4040711018	72721	50113	3	33855 86733	1 2	85	83	.025	.022	0	0	.026	.024	0	0	199	199	199	3
12	4135181497	22445	97149	1	26925	1	70	70	.02	.02	0	0	.019	.019	0	0	199	199	199	1
13	4237655883	59953	13107	1	15822	1	67	67	.017	.017	0	0	.019	.019	0	0	1	1	1	1

(1) SQLT PHV considers id, parent_id, operation, options, index_columns and object_name. SQLT PHV2 includes also access and filter predicates.
(2) If tables are empty, then Elapsed Time is close to Parse Time.
(3) B: Includes BASELINE.
(4) F: Includes at least one "_fix_control"

Completed Tests for Plan 1523566366 27311 79700

#	Test Id	Test	Baseline Value	Plan Cost	Buffer Gets	CPU (secs)	Disk Reads	ET (secs)	Actual Rows	Estim Rows
1	00127	ALTER SESSION SET "_optimizer_squ_bottomup" = TRUE;	FALSE	61814	74	.022	0	.023	0	199

需要的执行计划

两个工具提升SQL优化效率-SQLT找出正确执行计划需要设置的参数3

- 只能单个参数测试是否有效
- **做XPLORE使用XPLAIN方法，内部调用explain plan for,不需要执行从而提高效率和避免修改数据**
- 只有是已知参数或者BUG fix control才会有用，对于未知BUG无用，**当然修改参数需要做足测试**，如果非批量问题，建议找出原因，使用SQL PROFILE搞定，批量问题需要做足测试再实施修改！

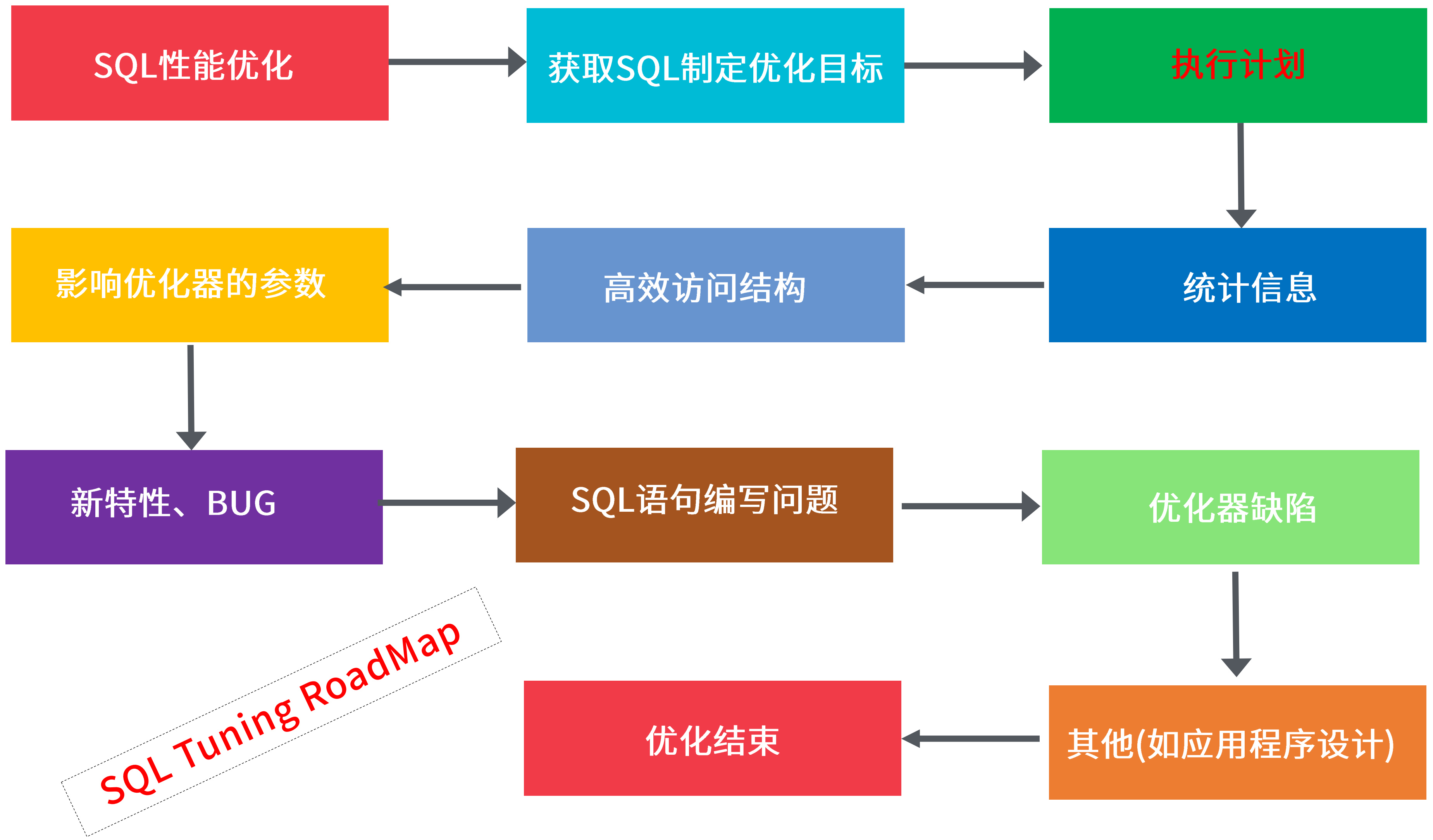
终于从跑不出来到几秒搞定，其实还可以优化，但是那已经不是最重要的事了！

Plan for Test:00127 ALTER SESSION SET "_optimizer_squ_bottomup" = TRUE;

```
Plan hash value: 1523566366

--------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name             | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                  |   199 | 20099 |       | 61814   (1)| 00:12:22 |
| * 1 |  COUNT STOPKEY                     |                  |       |       |       |            |          |
|   2 |   VIEW                             |                  |  715K |   68M |       | 61814   (1)| 00:12:22 |
| * 3 |    SORT ORDER BY STOPKEY           |                  |  715K |   39M |   46M | 61814   (1)| 00:12:22 |
| * 4 |     HASH JOIN RIGHT ANTI NA        |                  |  715K |   39M |   17M | 51793   (1)| 00:10:22 |
|   5 |      TABLE ACCESS BY INDEX ROWID   | MM_TL_LOG_201607 | 1172K | 4578K |       | 16031   (1)| 00:03:13 |
| * 6 |       INDEX RANGE SCAN             | IDX_STAND        | 1172K |       |       |  1706   (1)| 00:00:21 |
| * 7 |      HASH JOIN                     |                  |  715K |   36M |   22M | 32632   (1)| 00:06:32 |
|   8 |       VIEW                         |                  |  715K |   14M |       | 27601   (1)| 00:05:32 |
|   9 |        HASH GROUP BY               |                  |  715K |   20M |   44M | 27601   (1)| 00:05:32 |
| * 10 |        HASH JOIN RIGHT ANTI NA     |                  | 1172K |   33M |   17M | 20454   (1)| 00:04:06 |
|  11 |          TABLE ACCESS BY INDEX ROWID| MM_TL_LOG_201607 | 1172K | 4578K |       | 16031   (1)| 00:03:13 |
| * 12 |          INDEX RANGE SCAN          | IDX_STAND        | 1172K |       |       |  1706   (1)| 00:00:21 |
| * 13 |         TABLE ACCESS FULL          | DT_MBY_TEST_LOG  | 1172K |   29M |       |  1418   (2)| 00:00:18 |
|  14 |       TABLE ACCESS FULL            | DT_MBY_TEST_LOG  | 1172K |   36M |       |  1410   (1)| 00:00:17 |
--------------------------------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM<200)
   3 - filter(ROWNUM<200)
   4 - access("A"."ID"=TO_NUMBER("IMEI"))
   6 - access("STAND"='5')
   7 - access("A"."TBILL_ID"="B"."TBILL_ID" AND "A"."INSERT_TIME"="B"."INSERT_TIME")
  10 - access("ID"=TO_NUMBER("IMEI"))
  12 - access("STAND"='5')
  13 - filter("INSERT_TIME">'08-APR-15')
```

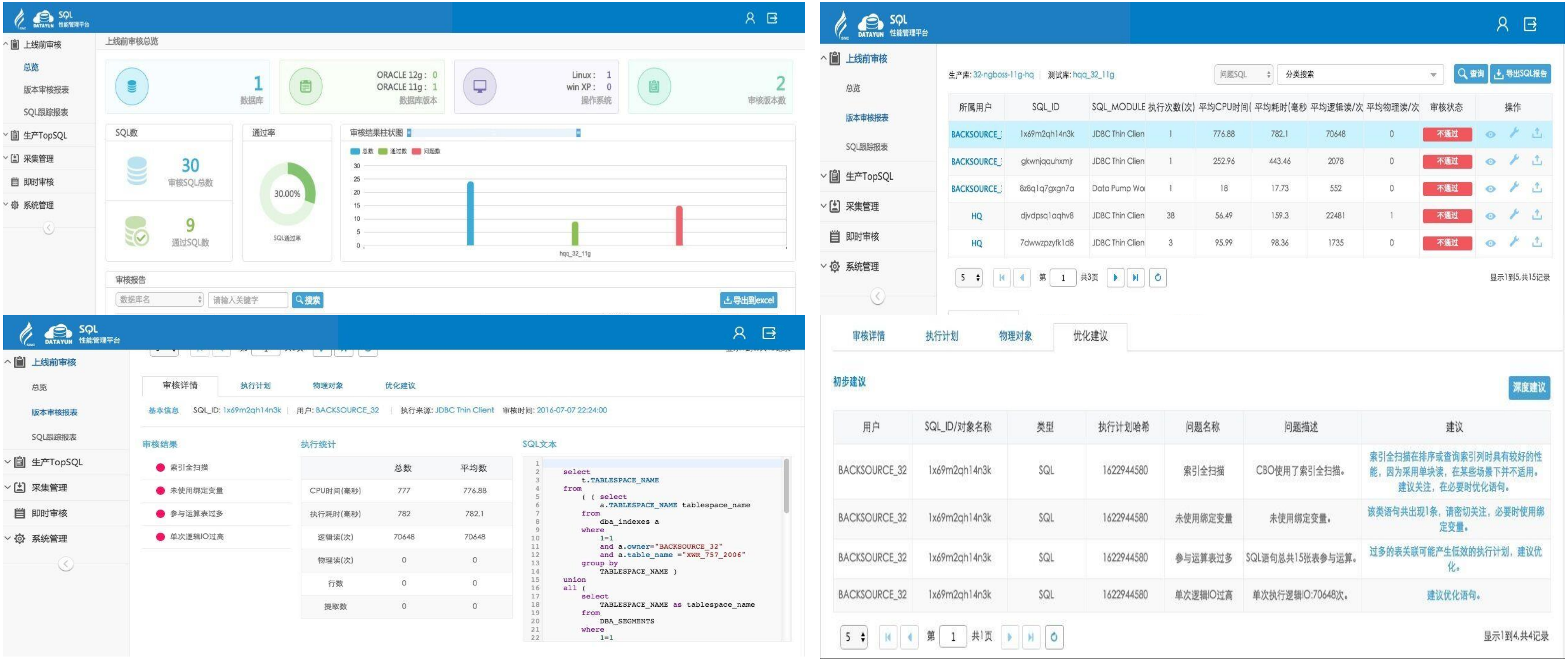
_optimizer_squ_bottomup （enables unnesting of subquery in a bottom-up manner）

SQL Tuning思考之RoadMap



SQL Tuning最佳实践-SQL性能管理平台

新炬网络SQL性能管理平台：实现事前审核、事中跟踪、事后监控的SQL性能全生命周期管理



SQL性能管理平台特点-自动化采集、分析、跟踪，减少DBA分析时间，提高管控效率

DBAplus

自动采集

智能分析

优化建议

上线跟踪

自动化采集分析版本新增SQL

•从测试环境收集并提交的该版本应用程序中涉及到的新增SQL
•监控该版本与历史SQL基线有变化的SQL及其执行计划

智能化分析SQL

•通过平台进行新增SQL初步智能化审查和分析。
•有问题的SQL提出问题点及生成初审报告交付开发DBA

SQL自动化优化

•平台自动对有问题SQL进行优化
•开发DBA根据分析初审报告对SQL进行深度优化处理，直到所有性能问题得到解决。

跟踪上线SQL性能

•平台跟踪和分析上线后的新增SQL性能实际状况
•开发DBA根据分析结果及时解决还潜在的SQL性能问题

THANKS