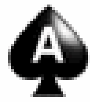
AOUG
AUSTRIAN ORACLE USER GROUP

DB-Performance Caveats for DB-Developers

Lothar Flatz

Senior Principal Consultant

By Martin St-Amant (S23678) - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11904410



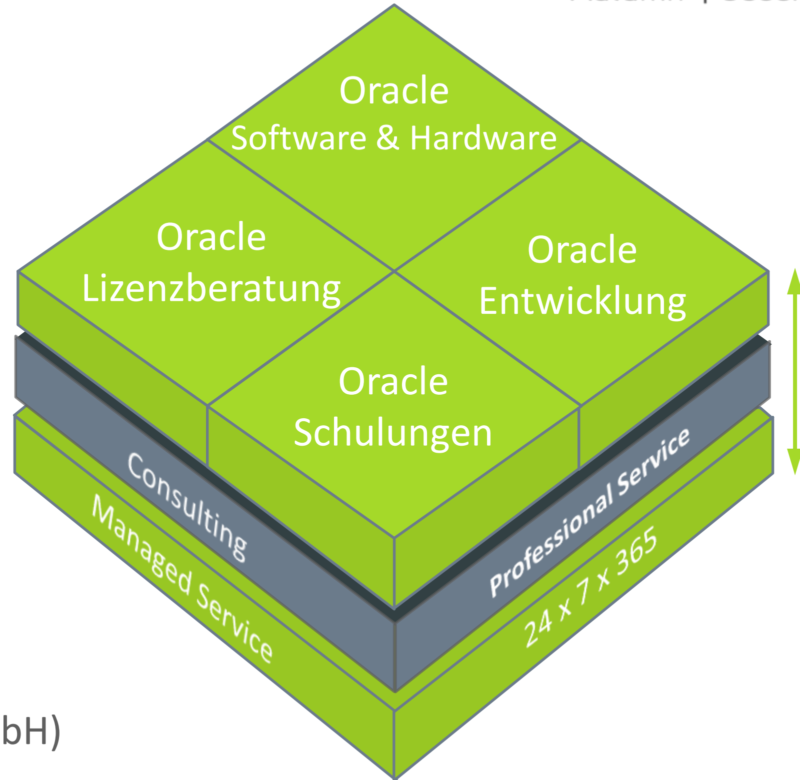
May I introduce myself

- 25 years Oracle Database experience (starting with Version 5)
- 15 Years Oracle Employee
- Oak Table Member
- Ex-Real World Performance Group
- Oracle ACE
- Signature Project: PVSS (CERN)
- US 8103658 B2 patent with Björn Engsig
- Senior Principal Consultant at DBConcepts GmbH

Über DBConcepts

Kennzahlen

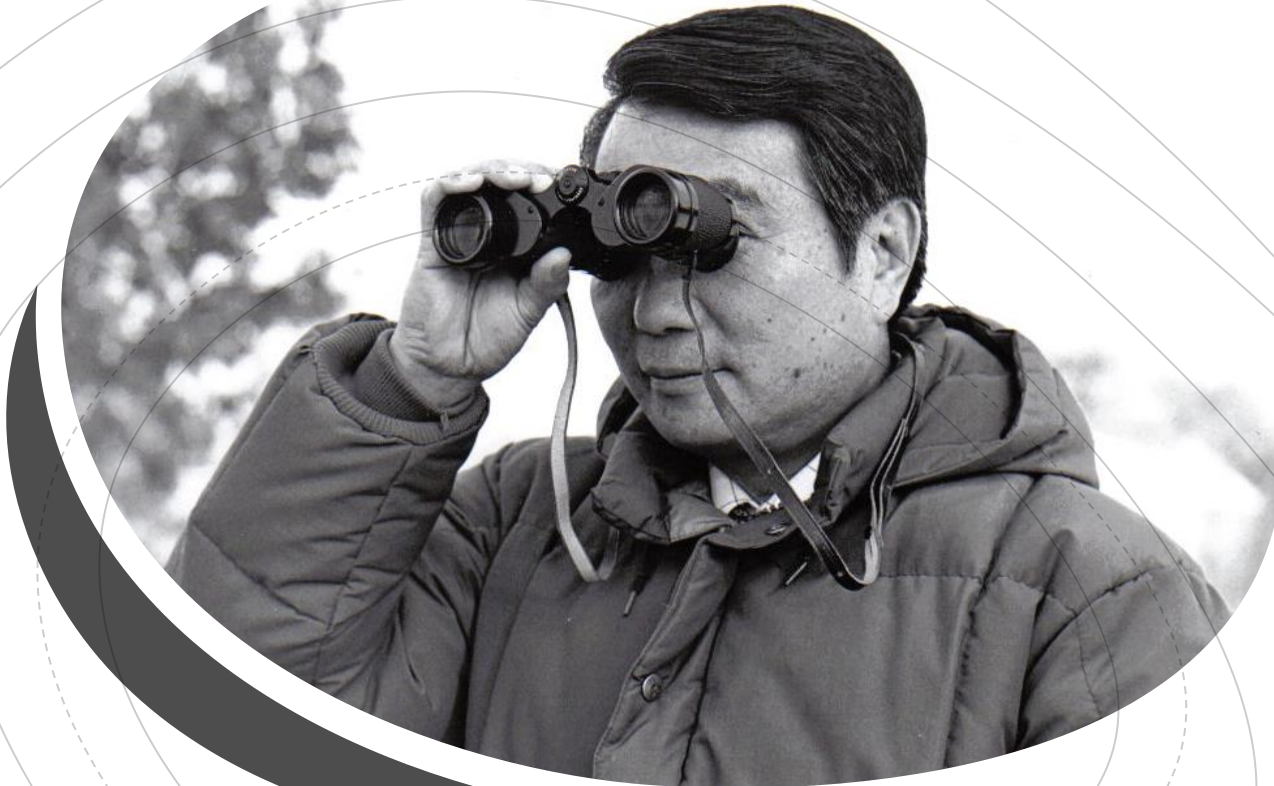
- Founded: 2000, in Vienna
- Employees: 45
- Revenue 2019: ca. € 17,4M
- Current locations:
  - Vienna
  - Nürnberg (DBConcept Deutschland GmbH)



© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

A major point

Basic Principle: Explore!



- AWR
- Statspack
- Runtime Stats
- SQL Monitor
- Top Activity

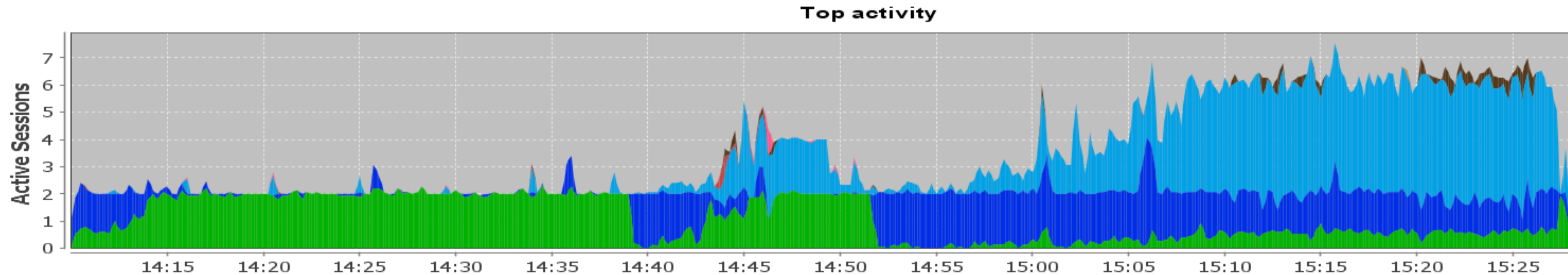
hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Facts help understanding



- Die Phases of a batch were incomprehensible from AWR or Statspack
- Only a graphical repsentation makes them visible
- A report over two hours would show a half-half mix of CPU (green) and I/O (blue)

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

$$f(x) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right)$$

Analytic Functions

Analytic Functions

Introduced in Oracle 8i, analytic functions, also known as windowing functions, allow developers to perform tasks in SQL that were previously confined to procedural languages.

(Tim Hall in OracleBase)

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Analytic Functions

analytic_function ([arguments])

OVER ([query_partition_clause]

[order_by_clause [windowing_clause]])

```
SELECT empno,
       deptno,
       sal,
       AVG(sal) OVER (PARTITION BY deptno) AS avg_dept_sal
FROM   emp;

 EMPNO      DEPTNO     SAL        AVG_DEPT_SAL
---------- ---------- ---------- ------------
      7782         10       2450   2916.66667
      7839         10       5000   2916.66667
      7934         10       1300   2916.66667
...
```

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Example: valid row in timerange

The most frequent use of analitc functions in db coding:

- Data is a mixture of current and historical rows
- Whenever the content of a row changes, a new row with current valid date is added
- To find the current active row the one with the highest valid date must be retrieved
- Traditionally we solved that using a max(valid date) subquery

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Old Solution

```
UPDATE TABLE1                    MY_TAB
SET ST_DOS1 =
   (SELECT CVA.CVASTRINGVALUE
      FROM TABLE2                          CVA
     WHERE CVA.DOSID = MY_TAB.K_DOSID
       AND CVA.CCHSID  = :MY_CCHSID
       AND CVA.CVAID   =
    (SELECT MAX(CVAID)
       FROM TABLE2  A
      WHERE A.DOSID = MY_TAB.K_DOSID
        AND A.CCHSID  = :MY_CCHSID
    )
  )
WHERE MY_TAB.K_DOSID IS NOT NULL;
```

- Table2 is scanned twice

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

hroug '22 Autumn | Jesen

Runtime Stats, old Solution


```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name          | Starts | E-Rows | A-Rows |   A-Time    | Buffers | Reads  |
-------------------------------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT                   |               |      1 |        |      0 |00:02:39.32 |    5508K|    185K|
|   1 |  UPDATE                            | TABLE1        |      1 |        |      0 |00:02:39.32 |    5508K|    185K|
|*  2 |   TABLE ACCESS FULL                | TABLE1        |      1 |    848K|    848K|00:00:03.74 |     424K|     87 |
|*  3 |   TABLE ACCESS BY INDEX ROWID      | TABLE2        |    419K|      1 |    419K|00:01:48.94 |    3351K|    173K|
|*  4 |    INDEX UNIQUE SCAN               | PK_TABLE2     |    419K|      1 |    419K|00:01:47.17 |    2932K|    173K|
|   5 |     SORT AGGREGATE                 |               |    418K|      1 |    418K|00:01:30.72 |    1674K|    152K|
|   6 |      TABLE ACCESS BY INDEX ROWID| TABLE2        |    418K|      1 |    418K|00:01:29.47 |    1674K|    152K|
|*  7 |       INDEX RANGE SCAN             | IX5_TABLE2    |    418K|      2 |    418K|00:00:28.52 |    1256K|  38052 |
-------------------------------------------------------------------------------------------------------------------
```

- Beside the subquery we also see an issue of a correlated update
- The execution always starts with the table to be updated (here Table1)
- The search criteria are on Table2 however
- Therefore, the update tests every row of table1
- BTW: we stopped the execution about half time

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Current valied row, new Solution

```
MERGE INTO TABLE1                MY_TAB
    USING (SELECT CVASTRINGVALUE, DOSID
             FROM (SELECT CVA.CVASTRINGVALUE,
                          DOSID, CVAID,
                          MAX(CVAID) OVER (PARTITION BY DOSID, CCHSID) MAX_CVAID
                     FROM TABLE2    CVA
                    WHERE CVA.CCHSID = :MY_CCHSID )
             WHERE CVAID = MAX_CVAID
          ) h
    ON (MY_TAB.K_DOSID = h.DOSID)
  WHEN MATCHED THEN
    UPDATE SET MY_TAB.ST_DOS1 = h.CVASTRINGVALUE
     WHERE MY_TAB.K_DOSID IS NOT NULL
       AND MY_TAB.ST_DOS1 <> h.CVASTRINGVALUE
 ;
```

- The Update Statement got rewritten into a Merge
- Consequently, the search criteria can be used in the first step
- Due to the analytic function Table2 gets scanned just once

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

hroug '22 Autumn | Jesen

Runtime Statistics, New Solution

hroug '22 Autumn | Jesen

```
--------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
--------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT              |                   |      1 |        |      0 |00:00:04.21 |   79736 |
|   1 |  MERGE                       | TABLE1            |      1 |        |      0 |00:00:04.21 |   79736 |
|   2 |   VIEW                       |                   |      1 |        |   146K|00:00:04.10 |   79736 |
|   3 |    NESTED LOOPS              |                   |      1 |  74975 |   146K|00:00:04.05 |   79736 |
|   4 |     NESTED LOOPS             |                   |      1 |   633K|   146K|00:00:03.38 |    6661 |
|*  5 |      VIEW                    |                   |      1 |   211K|   402K|00:00:02.42 |    2656 |
|   6 |       WINDOW SORT            |                   |      1 |   211K|   402K|00:00:02.15 |    2656 |
|*  7 |        INDEX RANGE SCAN      | IX_TABLE2         |      1 |   211K|   402K|00:00:00.17 |    2656 |
|*  8 |      INDEX RANGE SCAN        | TMPIX3CRO1_IAS1   |   402K|      3 |   146K|00:00:00.69 |    4005 |
|   9 |     TABLE ACCESS BY INDEX ROWID| TABLE1          |   146K|      1 |   146K|00:00:00.53 |   73075 |
--------------------------------------------------------------------------------------------------------
```

- No testing, but a targeted search
- Few rows retrieved
- Speed increase by 4000%
- Further improvement possible

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Rank()

- When it comes to analytic functions there are often several solutions to the same issue
- For a unknown reason it seems that for practical purposes rank() works better than other analytic functions
- E.g. it is easier to get a Window Nosort
- Also push of predicates into a View seems to work better than other analytic functions

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS
Die Oracle Experten

Example: most recent employee of a department


```
select ename, sal , deptno,hiredate,  max(hiredate) over (PARTITION by deptno) from emp
order by deptno, hiredate desc ;

 ENAME                  SAL      DEPTNO HIREDATE MAX(HIRE
---------- ---------- ---------- -------- --------
MILLER           1300         10 23.01.82 23.01.82
KING             5000         10 17.11.81 23.01.82
CLARK            2450         10 09.06.81 23.01.82
ADAMS            1100         20 23.05.87 23.05.87
SCOTT            3000         20 19.04.87 23.05.87
FORD             3000         20 03.12.81 23.05.87
JONES            2975         20 02.04.81 23.05.87
SMITH             800         20 17.12.80 23.05.87
JAMES             950         30 03.12.81 03.12.81
MARTIN           1250         30 28.09.81 03.12.81
TURNER           1500         30 08.09.81 03.12.81
...


create index i1 on emp (deptno, hiredate desc);
```


© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Solution with Max

```
select * from (
select ename, sal , deptno,hiredate,  max(hiredate) over (PARTITION by deptno) max_dept_hiredate
from emp where deptno is not null)
where hiredate= max_dept_hiredate;


------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |      1 |        |      3 |00:00:00.01 |       2 |
|*  1 |  VIEW                        |      |      1 |     14 |      3 |00:00:00.01 |       2 |
|   2 |   WINDOW BUFFER              |      |      1 |     14 |     14 |00:00:00.01 |       2 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |
|*  4 |     INDEX FULL SCAN          | I1   |      1 |     14 |     14 |00:00:00.01 |       1 |
------------------------------------------------------------------------------------------------

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

   1 - filter("HIREDATE"="MAX_DEPT_HIREDATE")
   4 - filter("DEPTNO" IS NOT NULL)
```

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Solution with Rank

hroug '22 Autumn | Jesen

```
select * from (
select  ename, sal , deptno,hiredate,  rank() over (PARTITION by deptno order by hiredate desc)
rnk_dept_hiredate from emp where deptno is not null)
where rnk_dept_hiredate =1;

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |      1 |        |      3 |00:00:00.01 |       2 |
|*  1 |  VIEW                        |      |      1 |     14 |      3 |00:00:00.01 |       2 |
|*  2 |   WINDOW NOSORT              |      |      1 |     14 |     14 |00:00:00.01 |       2 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |
|*  4 |     INDEX RANGE SCAN         | I1   |      1 |     14 |     14 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("RNK_DEPT_HIREDATE"=1)
   2 - filter(RANK() OVER ( PARTITION BY "DEPTNO" ORDER BY "EMP"."SYS_NC00010$")<=1)
   4 - filter("DEPTNO" IS NOT NULL)
```

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Window Nosort versus Window Buffer

hroug '22 Autumn | Jesen

- In our small example both versions look equal, but NOSORT only comes with RANK()
- In a bigger example we would see a difference
- Window Buffer blocking operation as opposed to Window Nosort
- Blocking operation means that the step in the execution plan must be fully completed before the next step can start
- A prerequisite for a nosort is a suitable index

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

hroug '22
Autumn | Jesen

BEWARE:

The Caveat of analytic Functions

Joe Loong from Reston, VA / CC BY-SA (https://creativecommons.org/licenses/by-sa/2.0)

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Analytic Functions in Views

hroug '22 Autumn | Jesen

There are some circumstances where Oracle cannot push predicates. It is difficult to obtain a definitive list of these situations, but here is a start:

- Predicates that access a view column that contain any of the following:
  - Aggregate functions. eg. AVG, COUNT, MAX, MIN, SUM
  - Analytic functions. eg. ROW_COUNT, RANK
  - The ROWNUM pseudocolumn
- Views that use CONNECT BY

http://www.orafaq.com/tuningguide/push%20predicates.html

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Runtime Stats of a query against a View

hroug '22 Autumn | Jesen

```
--------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                 | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  | Writes |
--------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                      |      1 |        |      0 |00:00:00.01 |       0 |      0 |      0 |
|   1 |  SORT ORDER BY                             |                      |      1 |  1625K|      0 |00:00:00.01 |       0 |      0 |      0 |
|*  2 |   VIEW                                     | TEST1_LOTHAR         |      1 |  1625K|      0 |00:00:00.01 |       0 |      0 |      0 |
|   3 |    HASH UNIQUE                             |                      |      1 |  1625K|      1 |00:02:29.82 |    211K|   430K|   296K|
|   4 |     WINDOW SORT                            |                      |      1 |  1625K|   1643K|00:02:27.03 |    211K|   430K|   296K|
|   5 |      WINDOW SORT                           |                      |      1 |  1625K|   1643K|00:01:17.83 |    211K|   298K|   164K|
|*  6 |       HASH JOIN RIGHT OUTER                |                      |      1 |  1625K|   1643K|00:00:16.16 |    211K|   101K|  33540 |
|   7 |        VIEW                                |                      |      1 |     81 |     81 |00:00:00.01 |     166 |      0 |      0 |
|*  8 |         HASH JOIN                          |                      |      1 |     81 |     81 |00:00:00.01 |     166 |      0 |      0 |
|   9 |          JOIN FILTER CREATE                | :BF0000              |      1 |     81 |     81 |00:00:00.01 |     121 |      0 |      0 |
|* 10 |           TABLE ACCESS STORAGE FULL        | TABLE1               |      1 |     81 |     81 |00:00:00.01 |     121 |      0 |      0 |
|  11 |          JOIN FILTER USE                   | :BF0000              |      1 |   5204 |     85 |00:00:00.01 |      45 |      0 |      0 |
|* 12 |           TABLE ACCESS STORAGE FULL        | TABLE2               |      1 |   5204 |     85 |00:00:00.01 |      45 |      0 |      0 |
|* 13 |        HASH JOIN RIGHT OUTER               |                      |      1 |  1625K|   1643K|00:00:15.54 |    211K|   101K|  33540 |
|  14 |         TABLE ACCESS STORAGE FULL          | TABLE3               |      1 |     65 |    108 |00:00:00.01 |       6 |      0 |      0 |
|* 15 |         HASH JOIN RIGHT ANTI               |                      |      1 |  1625K|   1643K|00:00:14.97 |    211K|   101K|  33540 |
|  16 |          VIEW                              |                      |      1 |  13879 |      4 |00:00:01.99 |   55050 |      0 |      0 |
|* 17 |           HASH JOIN                        |                      |      1 |  13879 |      4 |00:00:01.99 |   55050 |      0 |      0 |
|* 18 |            FILTER                          |                      |      1 |        |      3 |00:00:01.66 |   50646 |      0 |      0 |
|* 19 |             HASH JOIN OUTER                |                      |      1 |  13837 |   1643K|00:00:01.59 |   50646 |      0 |      0 |
|* 20 |              TABLE ACCESS STORAGE FULL     | TABLE4               |      1 |  1383K|   1384K|00:00:00.30 |   39689 |      0 |      0 |
|* 21 |              TABLE ACCESS STORAGE FULL     | TABLE5               |      1 |  1642K|   1643K|00:00:00.16 |   10957 |      0 |      0 |
|* 22 |            INDEX STORAGE FAST FULL SCAN    | INDEX_I1             |      1 |  1307K|   1307K|00:00:00.13 |    4404 |      0 |      0 |
|* 23 |          HASH JOIN                         |                      |      1 |  1642K|   1643K|00:00:12.38 |    156K|   101K|  33540 |
|* 24 |           TABLE ACCESS STORAGE FULL        | TABLE6               |      1 |  1642K|   1643K|00:00:00.15 |   16087 |  16077 |      0 |
|* 25 |           HASH JOIN                        |                      |      1 |  1642K|   1643K|00:00:10.08 |    140K|  75803 |  23745 |
|  26 |            TABLE ACCESS STORAGE FULL       | TABLE7               |      1 |    636 |    639 |00:00:00.01 |       6 |      0 |      0 |
|* 27 |            HASH JOIN                       |                      |      1 |  1642K|   1643K|00:00:09.42 |    140K|  75803 |  23745 |
|* 28 |             TABLE ACCESS STORAGE FULL      | TABLE8               |      1 |  1307K|   1307K|00:00:00.12 |   33060 |  33049 |      0 |
|* 29 |             HASH JOIN                      |                      |      1 |  1642K|   1643K|00:00:07.73 |    107K|  42754 |  23745 |
|  30 |              TABLE ACCESS STORAGE FULL     | TABLE9               |      1 |    636 |    639 |00:00:00.01 |      15 |      0 |      0 |
|* 31 |              HASH JOIN                     |                      |      1 |  1642K|   1643K|00:00:07.10 |    107K|  42754 |  23745 |
|* 32 |               TABLE ACCESS STORAGE FULL    | TABLE10              |      1 |  1307K|   1307K|00:00:00.43 |   37478 |      0 |      0 |
|* 33 |               HASH JOIN                    |                      |      1 |  1642K|   1643K|00:00:03.84 |   69665 |  19009 |  18690 |
|* 34 |                TABLE ACCESS STORAGE FULL   | TABLE11              |      1 |  1383K|   1384K|00:00:00.31 |   39689 |      0 |      0 |
|* 35 |                HASH JOIN                   |                      |      1 |  1642K|   1643K|00:00:02.04 |   29976 |  19009 |  18690 |
|* 36 |                 TABLE ACCESS STORAGE FULL| TABLE12              |      1 |  1642K|   1643K|00:00:00.14 |   10957 |      0 |      0 |
|* 37 |                 TABLE ACCESS STORAGE FULL| TABLE13              |      1 |  1642K|   1643K|00:00:00.20 |   19019 |  19009 |      0 |
--------------------------------------------------------------------------------------------------------------------------------------------
```

- Number of A-rows drops dramatically in step 3
- Most important search condition applied outside of view
- After laborious tests I found that an analytic function effectively blocks the search condition

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Same query (prev. Slide) without analytic Function

```
------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                         | Name                            | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |                                 |      1 |        |      0 |00:00:00.01 |       2 |
|*  1 |  FILTER                           |                                 |      1 |        |      0 |00:00:00.01 |       2 |
|   2 |   NESTED LOOPS ANTI               |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   3 |    NESTED LOOPS OUTER             |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   4 |     NESTED LOOPS OUTER            |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   5 |      NESTED LOOPS                 |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   6 |       NESTED LOOPS                |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   7 |        NESTED LOOPS               |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   8 |         NESTED LOOPS              |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|   9 |          NESTED LOOPS             |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|  10 |           NESTED LOOPS            |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|  11 |            NESTED LOOPS           |                                 |      1 |      1 |      0 |00:00:00.01 |       2 |
|  12 |             TABLE ACCESS BY INDEX ROWID| TABLE1                     |      1 |      1 |      0 |00:00:00.01 |       2 |
|* 13 |              INDEX UNIQUE SCAN    | TABLE1_IDX                      |      1 |      1 |      0 |00:00:00.01 |       2 |
|* 14 |             TABLE ACCESS BY INDEX ROWID| TABLE2                     |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 15 |              INDEX RANGE SCAN     | TABLE2_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 16 |            TABLE ACCESS BY INDEX ROWID | TABLE3                     |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 17 |             INDEX UNIQUE SCAN     | TABLE3_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 18 |           TABLE ACCESS BY INDEX ROWID | TABLE4                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 19 |            INDEX UNIQUE SCAN      | TABLE4_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 20 |          TABLE ACCESS BY INDEX ROWID | TABLE5                       |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 21 |           INDEX RANGE SCAN        | TABLE5_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 22 |         TABLE ACCESS BY INDEX ROWID | TABLE6                        |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 23 |          INDEX UNIQUE SCAN        | TABLE6_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 24 |        TABLE ACCESS BY INDEX ROWID | TABLE7                         |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 25 |         INDEX UNIQUE SCAN         | TABLE7_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 26 |       TABLE ACCESS BY INDEX ROWID | TABLE8                          |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 27 |        INDEX RANGE SCAN           | TABLE8_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 28 |      INDEX RANGE SCAN             | TABLE9_IDX                      |      0 |      1 |      0 |00:00:00.01 |       0 |
|  29 |     VIEW PUSHED PREDICATE         |                                 |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 30 |      HASH JOIN                    |                                 |      0 |      1 |      0 |00:00:00.01 |       0 |
|  31 |       TABLE ACCESS BY INDEX ROWID | TABLE10                         |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 32 |        INDEX UNIQUE SCAN          | TABLE10_IDX                     |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 33 |       TABLE ACCESS STORAGE FULL   | TABLE11                         |      0 |     81 |      0 |00:00:00.01 |       0 |
|  34 |    VIEW PUSHED PREDICATE          |                                 |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 35 |     FILTER                        |                                 |      0 |        |      0 |00:00:00.01 |       0 |
|  36 |      NESTED LOOPS OUTER           |                                 |      0 |      1 |      0 |00:00:00.01 |       0 |
|  37 |       NESTED LOOPS                |                                 |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 38 |        INDEX RANGE SCAN           | TABLE12_IDX                     |      0 |      1 |      0 |00:00:00.01 |       0 |
|  39 |        TABLE ACCESS BY INDEX ROWID | TABLE13                        |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 40 |         INDEX RANGE SCAN          | TABLE13_IDX                     |      0 |      1 |      0 |00:00:00.01 |       0 |
|* 41 |       TABLE ACCESS BY INDEX ROWID | TABLE14                         |      0 |  1642K|      0 |00:00:00.01 |       0 |
|* 42 |        INDEX RANGE SCAN           | TABLE14_IDX                     |      0 |      1 |      0 |00:00:00.01 |       0 |
------------------------------------------------------------------------------------------------------------------------------------
```

- The analytic Function was removed (was actually not needed)
- Search condition applied in first step (13) now
- It lasts 0,01 seconds rather than 147 seconds
- Improvment by factor 14700!

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Predicate Push Down in View with Analytic Function

hroug '22 Autumn | Jesen

```
SELECT
    ename,
    sal,
    hiredate,
    deptno,
    dname
FROM
    (
        SELECT
            ename,
            sal,
            d.deptno,
            hiredate,
            dname,
            RANK() OVER(
                PARTITION BY e.deptno
                ORDER BY
                    hiredate DESC
            ) rnk_dept_hiredate
        FROM
            emp e, dept d
        WHERE
          e.deptno=d.deptno
    )
WHERE
    rnk_dept_hiredate = 1
    and deptno=20;
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      1 |00:00:00.01 |       9 |
|*  1 |  VIEW                         |         |      1 |      5 |      1 |00:00:00.01 |       9 |
|*  2 |   WINDOW SORT PUSHED RANK     |         |      1 |      5 |      1 |00:00:00.01 |       9 |
|   3 |    NESTED LOOPS               |         |      1 |      5 |      5 |00:00:00.01 |       9 |
|   4 |     TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  5 |      INDEX UNIQUE SCAN        | PK_DEPT |      1 |      1 |      1 |00:00:00.01 |       1 |
|*  6 |     TABLE ACCESS FULL         | EMP     |      1 |      5 |      5 |00:00:00.01 |       7 |
-----------------------------------------------------------------------------------------------------

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

   1 - filter("RNK_DEPT_HIREDATE"=1)
   2 - filter(RANK() OVER ( PARTITION BY "E"."DEPTNO" ORDER BY
              INTERNAL_FUNCTION("HIREDATE") DESC )<=1)
   5 - access("D"."DEPTNO"=20)
   6 - filter("E"."DEPTNO"=20)
```

» Conditions based on the partition by columns can be pushed into the View

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Push in View mit Analytic Function

```
SELECT
    ename,
    sal,
    hiredate,
    deptno,
    dname
FROM
    (
        SELECT
            ename,
            sal,
            d.deptno,
            hiredate,
            dname,
            RANK() OVER(
                PARTITION BY e.deptno
                ORDER BY
                    hiredate DESC
            ) rnk_dept_hiredate
        FROM
            emp e, dept d
        WHERE
          e.deptno=d.deptno
    )
WHERE
    rnk_dept_hiredate = 1
    and dname='RESEARCH';
```

```
------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |         |      1 |        |      1 |00:00:00.01 |       9 |
|*  1 |  VIEW                          |         |      1 |     14 |      1 |00:00:00.01 |       9 |
|*  2 |   WINDOW SORT PUSHED RANK      |         |      1 |     14 |      3 |00:00:00.01 |       9 |
|   3 |    MERGE JOIN                  |         |      1 |     14 |     14 |00:00:00.01 |       9 |
|   4 |     TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       2 |
|   5 |      INDEX FULL SCAN           | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       1 |
|*  6 |     SORT JOIN                  |         |      4 |     14 |     14 |00:00:00.01 |       7 |
|   7 |      TABLE ACCESS FULL         | EMP     |      1 |     14 |     14 |00:00:00.01 |       7 |
------------------------------------------------------------------------------------------------------

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

   1 - filter(("RNK_DEPT_HIREDATE"=1 AND "DNAME"='RESEARCH'))
   2 - filter(RANK() OVER ( PARTITION BY "E"."DEPTNO" ORDER BY
              INTERNAL_FUNCTION("HIREDATE") DESC )<=1)
   6 - access("E"."DEPTNO"="D"."DEPTNO")
       filter("E"."DEPTNO"="D"."DEPTNO")
```

» Dname is not in Partition By clause and the condition can consequently not be pushed into the view

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Push in View mit Analytic Function - Lösung

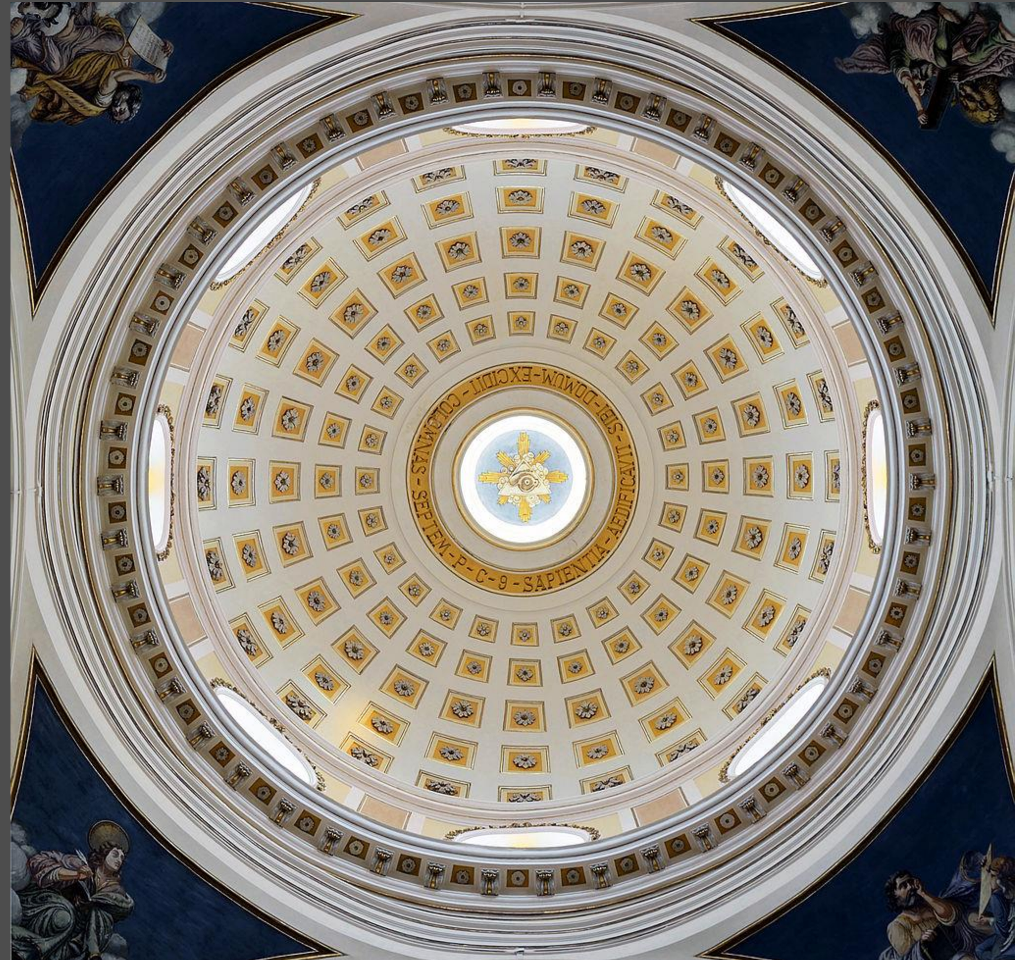
```
SELECT
    ename,
    sal,
    hiredate,
    d.deptno,
    dname
FROM
    (
        SELECT
            ename,
            sal,
            deptno,
            hiredate,
            RANK() OVER(
                PARTITION BY deptno
                ORDER BY
                    hiredate DESC
            ) rnk_dept_hiredate
        FROM
            emp
    ) e,
    dept d
WHERE
    rnk_dept_hiredate = 1
    AND e.deptno = d.deptno
    AND dname = 'RESEARCH';
```

```
CREATE INDEX ei1 ON  emp ( deptno );
----------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |      |      1 |        |      1 |00:00:00.01 |       9 |
|   1 |  NESTED LOOPS                           |      |      1 |      4 |      1 |00:00:00.01 |       9 |
|*  2 |   TABLE ACCESS FULL                     | DEPT |      1 |      1 |      1 |00:00:00.01 |       7 |
|*  3 |   VIEW PUSHED PREDICATE                 |      |      1 |      1 |      1 |00:00:00.01 |       2 |
|   4 |    WINDOW SORT                          |      |      1 |      5 |      5 |00:00:00.01 |       2 |
|   5 |     TABLE ACCESS BY INDEX ROWID BATCHED| EMP  |      1 |      5 |      5 |00:00:00.01 |       2 |
|*  6 |      INDEX RANGE SCAN                   | EI1  |      1 |      5 |      5 |00:00:00.01 |       1 |
----------------------------------------------------------------------------------------------------------

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

   2 - filter("DNAME"='RESEARCH')
   3 - filter("RNK_DEPT_HIREDATE"=1)
   6 - access("DEPTNO"="D"."DEPTNO")
```

- Apply the analytic function as early as possible in the processing
- That way you might limit the scope of the function
- There is good chance to also limit the window sort
- The window sort is often resource intensive

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

PARALLEL INDIZES



By Livioandronico2013 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=39442599

Create Indizes fast

- Parallel and nologging accelerates the index create
- But these two properties then remain attached to the index
- This is contra intuitive
- You have to specify alter index xxx noparallel logging; in a second step.
- If you don't do that, even small and smallest queries can run in parallel
- This will burn CPU without use

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Example: parallel query

```
--------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name            | Starts | E-Rows | A-Rows |   A-Time    | Buffers | Reads  |
--------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                 |      1 |        |      0 |00:00:02.84 |      84 |     37 |
|   1 |  PX COORDINATOR                      |                 |      1 |        |      0 |00:00:02.84 |      84 |     37 |
|   2 |   PX SEND QC (RANDOM)                | :TQ10001        |      0 |      4 |      0 |00:00:00.01 |       0 |      0 |
|*  3 |    HASH JOIN                         |                 |      0 |      4 |      0 |00:00:00.01 |       0 |      0 |
|   4 |     JOIN FILTER CREATE               | :BF0000         |      0 |      3 |      0 |00:00:00.01 |       0 |      0 |
|   5 |      PX RECEIVE                      |                 |      0 |      3 |      0 |00:00:00.01 |       0 |      0 |
|   6 |       PX SEND BROADCAST              | :TQ10000        |      0 |      3 |      0 |00:00:00.01 |       0 |      0 |
|   7 |        NESTED LOOPS                  |                 |      0 |      3 |      0 |00:00:00.01 |       0 |      0 |
|   8 |         NESTED LOOPS                 |                 |      0 |      6 |      0 |00:00:00.01 |       0 |      0 |
|   9 |          PX BLOCK ITERATOR           |                 |      0 |        |      0 |00:00:00.01 |       0 |      0 |
|* 10 |           TABLE ACCESS STORAGE FULL| PROC_STEP       |      0 |      6 |      0 |00:00:00.01 |       0 |      0 |
|* 11 |          INDEX UNIQUE SCAN           | LNGC_PK         |      0 |      1 |      0 |00:00:00.01 |       0 |      0 |
|* 12 |         TABLE ACCESS BY INDEX ROWID| CTL_LANGUAGE    |      0 |      1 |      0 |00:00:00.01 |       0 |      0 |
|  13 |     JOIN FILTER USE                  | :BF0000         |      0 |    143K|      0 |00:00:00.01 |       0 |      0 |
|  14 |      PX BLOCK ITERATOR               |                 |      0 |    143K|      0 |00:00:00.01 |       0 |      0 |
|* 15 |       TABLE ACCESS STORAGE FULL      | PROC_INTERRUPT |      0 |    143K|      0 |00:00:00.01 |       0 |      0 |
--------------------------------------------------------------------------------------------------------------------------
```

» The runtime statistics are displayed only for the serial steps

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Example: serial query

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name             | Starts | E-Rows | A-Rows |   A-Time   |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |                  |      1 |        |      1 |00:00:00.01 |
|   1 |  SORT AGGREGATE                         |                  |      1 |      1 |      1 |00:00:00.01 |
|*  2 |   HASH JOIN                             |                  |      1 |      1 |      0 |00:00:00.01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID BATCHED  | D_CALENDAR       |      1 |      1 |      0 |00:00:00.01 |
|*  4 |     INDEX RANGE SCAN                    | DCAL_UK          |      1 |    475 |      0 |00:00:00.01 |
|   5 |    VIEW                                 |                  |      0 |     86 |      0 |00:00:00.01 |
|   6 |     HASH UNIQUE                         |                  |      0 |     86 |      0 |00:00:00.01 |
|*  7 |      FILTER                             |                  |      0 |        |      0 |00:00:00.01 |
|   8 |       NESTED LOOPS                      |                  |      0 |     86 |      0 |00:00:00.01 |
|*  9 |        HASH JOIN                        |                  |      0 |      4 |      0 |00:00:00.01 |
|  10 |         JOIN FILTER CREATE              | :BF0000          |      0 |      3 |      0 |00:00:00.01 |
|* 11 |          HASH JOIN                      |                  |      0 |      3 |      0 |00:00:00.01 |
|  12 |           JOIN FILTER CREATE            | :BF0001          |      0 |      1 |      0 |00:00:00.01 |
|* 13 |            TABLE ACCESS STORAGE FULL    | CTL_LANGUAGE     |      0 |      1 |      0 |00:00:00.01 |
|  14 |           JOIN FILTER USE               | :BF0001          |      0 |      6 |      0 |00:00:00.01 |
|* 15 |            TABLE ACCESS STORAGE FULL    | PROC_STEP        |      0 |      6 |      0 |00:00:00.01 |
|  16 |         JOIN FILTER USE                 | :BF0000          |      0 |    143K|      0 |00:00:00.01 |
|* 17 |          TABLE ACCESS STORAGE FULL      | PROC_INTERRUPT   |      0 |    143K|      0 |00:00:00.01 |
|* 18 |        COLLECTION ITERATOR SUBQUERY FETCH|                 |      0 |     20 |      0 |00:00:00.01 |
|  19 |         CONNECT BY WITHOUT FILTERING    |                  |      0 |        |      0 |00:00:00.01 |
|  20 |          FAST DUAL                      |                  |      0 |      1 |      0 |00:00:00.01 |
-----------------------------------------------------------------------------------------------------------
```

» The same query with a no_parallel hint is 200x faster.

» For a database I once calculated that 15% of the CPU power was wasted useless parallel queries

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Conclusions

- » Parallelism is not a gas pedal according to the motto more is faster
- » Starting up and communicating parallel processes has its price (resources, time).
- » The threshold from which parallelism is worthwhile is much higher than commonly believed
- » -> Rather parallelize conservatively

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

COLUMN VALUE SELECTS

By Lambtron - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=38264285

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Different runtime for the same query

» On one DB the statement runs slowly

```
SQL Id          Elapsed (s)     CPU (s)  IOWait (s)          Gets        Reads        Rows Cluster (s)       Execs
-------------  ------------- ----------- ----------- -------------- ------------ ------------ ----------- ------------
1shqju6v7dfrb      31,480.58    8,085.60      194.88  1,546,592,832    2,765,602       54,492       11.16            1
```

» On the other DB the statement runs faster

» Although it is executed even twice

» As you can see the reads and the buffer gets are comparable in size

» The I/O waits differ

```
SQL Id          Elapsed (s)     CPU (s)  IOWait (s)          Gets        Reads        Rows Cluster (s)       Execs
-------------  ------------- ----------- ----------- -------------- ------------ ------------ ----------- ------------
1shqju6v7dfrb       8,485.90   18,716.44       26.24  1,537,775,731    3,672,865       83,708        0.64            2
```

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

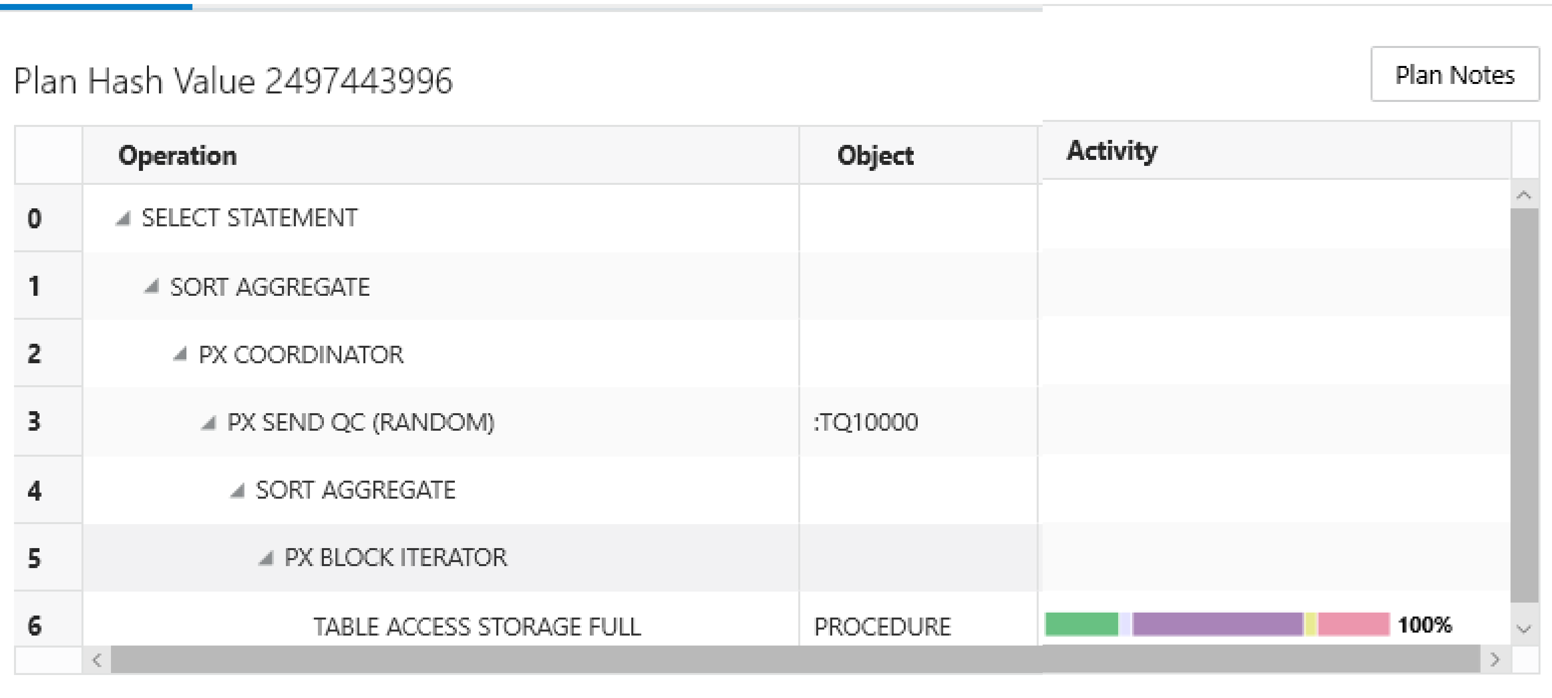
DBCONCEPTS Die Oracle Experten

hroug '22 Autumn | Jesen

SQL Monitor (slow DB)

» Step 6 causes 100% of the activity

» Mainly waiting events, unusually manifold (colorful)

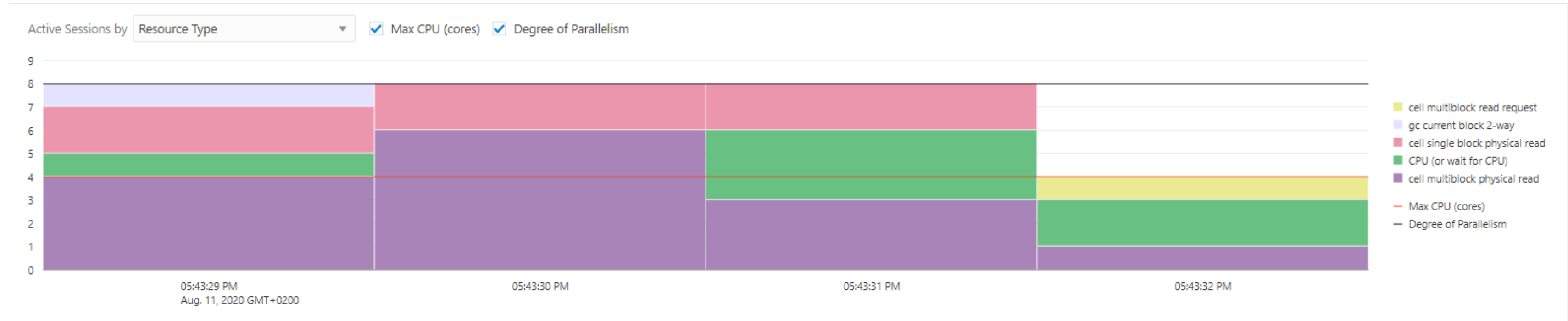


ORACLE Platinum Partner
© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.
DBCONCEPTS Die Oracle Experten

SQL Monitor (langsame DB), Activity Tab

hroug '22 Autumn | Jesen

» At first glance, there seems to be no logical relation between the individual wait events

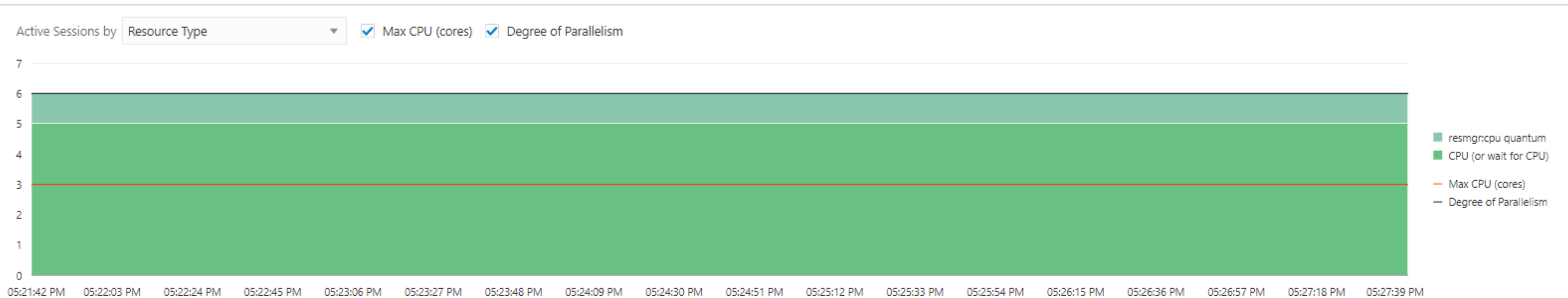


© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

SQL Monitor (schnelle DB), Activity Tab

hroug '22 Autumn | Jesen

» On the other hand, the run on the fast DB scales purely on the CPU



ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS
Die Oracle Experten

You should focus there

» However, when I checked the plan

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                                                          | Name                   | Rows  |
-----------------------------------------------------------------------------------------------------
|   0 | CREATE TABLE STATEMENT                                             |                        |       |
|   1 |  SORT AGGREGATE                                                    |                        |     1 |
|   2 |   TABLE ACCESS STORAGE FULL                                        | Table1                 |     2 |
|   3 |  PX COORDINATOR                                                    |                        |       |
|   4 |   PX SEND QC (RANDOM)                                              | :TQ10020               | 45229 |
|   5 |    LOAD AS SELECT (HYBRID TSM/HWMB)                                | Tabel2_                |       |
|   6 |     OPTIMIZER STATISTICS GATHERING                                 |                        | 45229 |
|   7 |      HASH JOIN                                                     |                        | 45229 |
|   8 |       JOIN FILTER CREATE                                           | :BF0000                | 44458 |
|   9 |        PX RECEIVE                                                  |                        | 44458 |
|  10 |         PX SEND HYBRID HASH                                        | :TQ10018               | 44458 |
|  11 |          STATISTICS COLLECTOR                                      |                        |       |
|  12 |           HASH JOIN OUTER BUFFERED                                 |                        | 44458 |
|  13 |            JOIN FILTER CREATE                                      | :BF0001                | 44061 |
|  14 |             PX RECEIVE                                             |                        | 44061 |
|  15 |              PX SEND HASH                                          | :TQ10016               | 44061 |
|  16 |               HASH JOIN OUTER BUFFERED                             |                        | 44061 |
|  17 |                JOIN FILTER CREATE                                  | :BF0002                | 43738 |
|  18 |                 PX RECEIVE                                         |                        | 43738 |
|  19 |                  PX SEND HASH (NULL RANDOM)                        | :TQ10014               | 43738 |
|  20 |                   HASH JOIN BUFFERED                               |                        | 43738 |
|  21 |                    PX BLOCK ITERATOR                               |                        |  2565K|
|  22 |                     TABLE ACCESS STORAGE FULL                      | Table1                 |  2565K|
```

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

You should focus there

```
SELECT t1.column1,
       t1.column2,
       t1.column3,
       t3.column4,
       t5.column5,
       t5.column6,
       t6.column7,
       t1.column8,
       DECODE(t2.column9,prfa.column10,
              DECODE(LENGTH(TRIM(TRANSLATE(t1.column14, '0123456789', ' '))),NULL,
                     DECODE((SELECT COUNT(1) FROM Table1 t11 WHERE t11.column12 =TO_NUMBER(t1.column13)),0,
                             prfa.column15,
                             t1.column17)
                     ,prfa.column12)
              ,prfa. column13),
```

- It was a column-valued select
- The select must be done once per record in the result
- Pleasant to write (no outer join necessary)
- Bad to optimize (no hash join)

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Solution

```
Create Index … Table1 (column11) …
```

- If you do not rewrite the query, it is absolutely necessary to create an index for the column-valued select
- Runtime with index 13.62 seconds instead of 31.480.58 seconds before
- This is 2300 x faster
- Therefore, the question why it is faster on one DB and slower on the other DB is no longer an issue.
- This is often the case; it is better to solve problems than to make comparisons.
- But the question is still answered

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

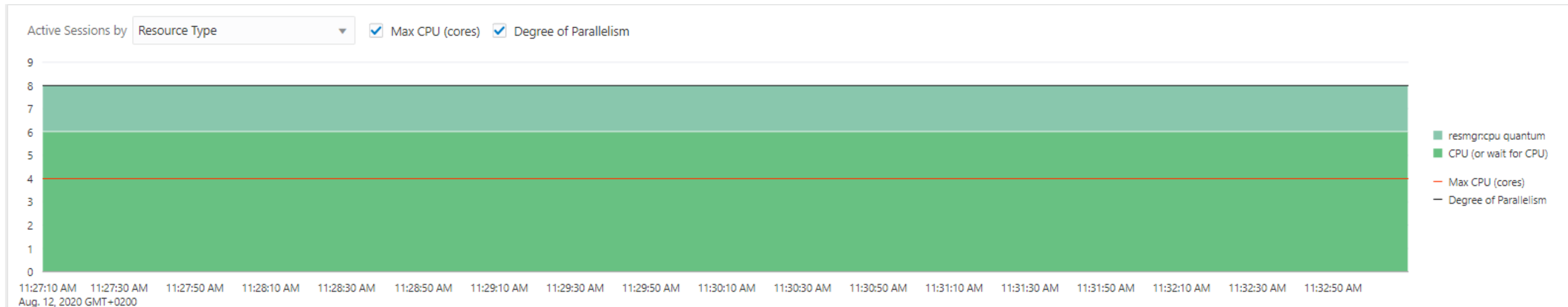
© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

SQL Monitor (slow DB), Activity Tab, 6th run

hroug '22 Autumn | Jesen

- Now knowing that the wait events are caused by many smart scans, I suspected a caching effect on the storage server.
- Indeed: After running the slow run several times, the wait events disappear.
- Only difference 8x parallel versus 6x parallel



© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Conclusion

» Avoid column-valued selects from a performance point of view

» Create index if has to be

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

EMBEDED PL/SQL FUNCTIONS

By Goran.S2 - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=7388146

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Basic consideration

- On the one hand, it is always said that one should modularize.
- But you can't write modules for SQL
- The answer was officially Views
- But views cannot have parameters
- Therefore, you can't provide the views with an API
- As an alternative often PL/SQL functions are called from SQL

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Context Switch

- SQL and PL/SQL run in different environments
- Establishing the connection between the two environments is computationally expensive
- This effort is called a context switch
- The context switch is very short, at least less than a ten-thousandth of a second.
- But it should not be underestimated, if it takes place millions of times, a context switch can take a very long time.
- How long the context switch actually is depending on the database version.

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Pragmas and Restrictions

- PRAGMA UDF
- Parallel
- Restrict_references
- Deterministic
- Result Cache
- Autonomous transaction
- AUTHID

By Bavarian Purity Law - http://www.bullionstop.com/2016-500-years-bavarian-purity-law-1-silver-coin-palau/, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=87355355

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

PRAGMA UDF (User Defined Function)

- A pragma is a compiler instruction
- Declarative, just needs to be entered
- **May improve** performance if a PL/SQL function is called frequently from SQL
- **May** slightly **degrade** performance if the PL/SQL function is called from PL/SQL
- The extent of the improvement depends on the data type of the parameters and the data type of the result

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Functions in the WITH Clause

hroug '22
Autumn | Jesen

```
with
function first_5_lc(pv_str in varchar2)
return varchar2
is
   lv_ret varchar2(5);
begin
   lv_ret := lower(substr(pv_str, 1, 5));
   return lv_ret;
end first_5_lc;
select min(first_5_lc(cluster_name)) str
  from asc_bigtable
/
```

- The function exists as long as the statement is running
- You do not need to save rights functions
- Context switch is smaller
- With more complex functions of course problematic (no reuse, write again and again)

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Wrapper

```
with
function complex(p_int1 in Number) return
Number
is
   lv_ret Number;
begin
   lv_ret := calc_pkg.complex(p_int1);
end complex;
select min(complex(id)) res
  from asc_bigtable
/
```

- A little trick
- A wrapper is a function that only calls another function
- The actual processing takes place in the called function
- The context switch is reduced, because the function is called from SQL from the WITH clause.
- The call of the function is within PL/SQL, therefore there is no context switch here.

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

DML with functions in the WITH clause

hroug '22 Autumn | Jesen

```
SQL> insert into t1(n1, n2)
select n1, n2
  from (
        with function add_1 (p_number in number) return number
        is
        begin
           return p_number + 1;
        end add_1;
        select rownum n1, add_1(rownum) n2
          from xmltable('1 to 10')
        )
/


ERROR at line 4:
ORA-32034: unsupported use of WITH clause
```

- There is no straightforward way

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

DML with functions in the WITH clause

hroug '22 Autumn | Jesen

```
SQL> insert /*+ with_plsql */ into t1(n1, n2)
select n1, n2
  from (
        with function add_1 (p_number in number) return number
        is
        begin
           return p_number + 1;
        end add_1;
        select rownum n1, add_1(rownum) n2
          from xmltable('1 to 10')
        );
 12  /

10 rows created.
```

» You will need a hint

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Test of Bryn Llewellyn

Full presentation

	Centisec	------------- Ratio to Pure_SQL time ------------			
	Pure_SQL	Plain_Plsql	UDF_Plsql	With_Clause	With_Wrapper
Num_Num	71	13.6	2.6	3.2	4.9
Num_VC	354	4.2	1.8	1.9	2.5
Num_Date	428	11.3	11.2	7.7	8.0
VC_Num	125	10.2	10.2	2.2	3.2
VC_VC	81	15.4	15.5	3.6	5.2
VC_Date	488	9.9	9.8	6.7	7.1
Date_Num	215	7.5	7.4	2.4	3.1
Date_VC	498	4.3	4.3	2.0	2.3
Date_Date	65	18.8	19.3	3.4	5.2
BF_BF	74	16.5	2.4	3.0	4.5
BD_BD	77	15.7	2.5	3.0	4.6

Ten Rules for doing a PL/SQL Performance Experiment (Bryn Llewellyn):
https://community.oracle.com/docs/DOC-1018914

» The second type after the _ is the Resulttype

» With the wrapper some things can be mitigated

» It is remarkable how much faster pure SQL still is

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

PARALLEL_ENABLE

- Declarative
- If you do not specify it, a parallel process is serialized
- Means that one can divide the work in the context of a parallel processing on several processes, without which the result changes
- Can usually be specified without much thought
- Exceptions are only functions, where the data are to be processed in a strict order, for example in which intermediate results are formed.
- The purity rules on the next slides explain the sense in more detail

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

PRAGMA RESTRICT_REFERENCES

- In early versions required to make PL/SQL functions callable from SQL
- Automatically checked today
- Exception: parallelize package functions that are called via a WITH function wrapper
- The word State is used here in the sense of software development:
  - WNDS (Write No Database State) : Does not change the contents of the database
  - WNPS (Write No Package State) : Does not change the content of package variables
  - RNDS (Read No Database State) : Does not read the contents of the database
  - RNPS (Read No Package State) : Does not read the content of package variables

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DETERMINISTIC

- Means that if the function is called several times with the same parameter, the result will always be the same.
- The result of the function must not depend e.g. on a DB query.
- Declarative
- Requirement for a number of features
  - Function Based Index
  - Calculated column
  - Extended Statistics
  - Session cache
- Should be specified whenever possible

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS
Die Oracle Experten

Result_Cache

» The result of the calculation is cached

» Declarative

» Changes of the data are monitored

» After a relevant data change the result is recalculated

» A latch watches over the cache

» It is possible to overload the latch (and that can get really nasty!)

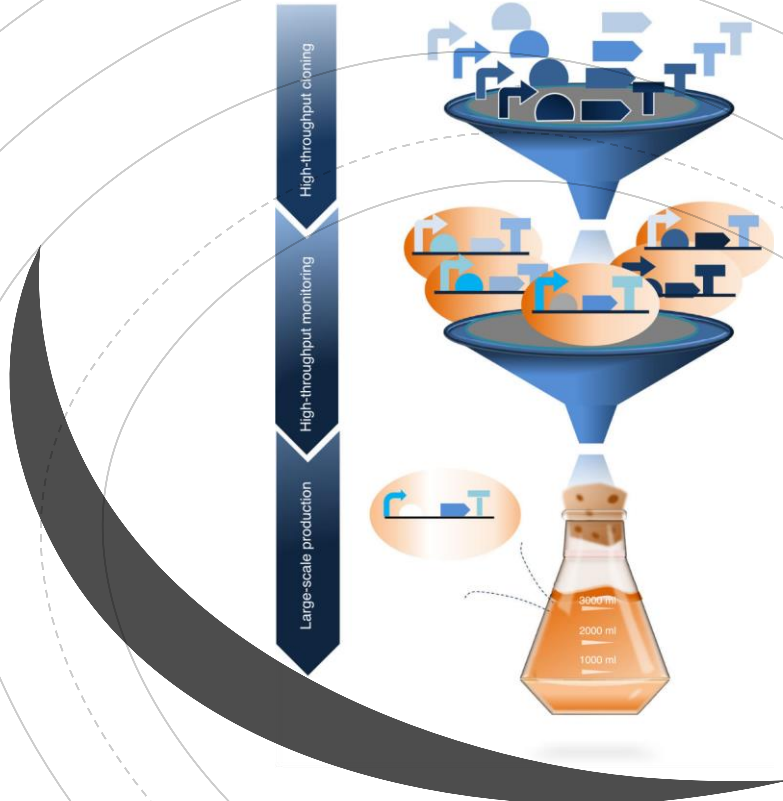
© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

PRAGMA AUTONOMOUS_TRANSACTION

- The function runs in its own transaction
- Declarative
- Typically for DML that has nothing to do with the rest of the processing, e.g. application log, error log

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Optimizing and Functions



By Gita Naseri & Mattheos A. G. Koffas - https://doi.org/10.1038/s41467-020-16175-y, Attribution, https://commons.wikimedia.org/w/index.php?curid=90398528

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

hroug '22 Autumn | Jesen

Calling PL/SQL from SQL

- Bottom line: avoid if possible
- Better is pure SQL
- Context Switch is only a small problem
- The worst is when the database is read in the function
- Then the SQL is broken into parts
- This means that the optimizer is no longer left with the whole task to optimize, but only parts at a time
- In addition, the "Read Consistency" is compromised. The result could theoretically even be wrong from a business view.(The Problem with SQL Calling PL/SQL Calling SQL)

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Multiblock Reads versus Single block read

» A small experiment to approach the problem

» The same select in two variants: once optimized to the task, once each data set as subtask via index

» The test ran on my notebook

```
select num_rows, blocks, avg_row_len from user_tables where table_name='MEDI_A';

 NUM_ROWS     BLOCKS AVG_ROW_LEN
---------- ---------- -----------
   7344000     229560         214

Alter system flush buffer_cache;

select count(data) from medi_a m;

-------------------------------------------------------------------------------------------------
| Id  | Operation           | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |      1 |        |      1 |00:00:01.39 |     222K|    222K|
|   1 |  SORT AGGREGATE     |        |      1 |      1 |      1 |00:00:01.39 |     222K|    222K|
|   2 |   TABLE ACCESS FULL| MEDI_A |      1 |   7344K|   7344K|00:00:01.36 |     222K|    222K|
-------------------------------------------------------------------------------------------------
```

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Multiblock Reads versus Single block read

- Enforced index access is 15 times slower
- In the PL/SQL function you access the table medi_a in a loop, row by row
- You rely on the Index, a full table scan to get one row is not an option

```
select /*+ INDEX(m MEDI_A_I01) */ count(data) from medi_a m where medi_a_nr>0;

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name         | Starts | E-Rows | A-Rows |    A-Time    | Buffers | Reads  |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |              |      1 |        |      1 |00:00:21.50 |    259K|    244K|
|   1 |  SORT AGGREGATE                      |              |      1 |      1 |      1 |00:00:21.50 |    259K|    244K|
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| MEDI_A       |      1 |  7344K|  7344K|00:00:21.13 |    259K|    244K|
|*  3 |    INDEX RANGE SCAN                  | MEDI_A_I01 |      1 |  7344K|  7344K|00:00:02.80 |   21881 |  21951 |
-----------------------------------------------------------------------------------------------------------------------
```


© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

CPU Usage

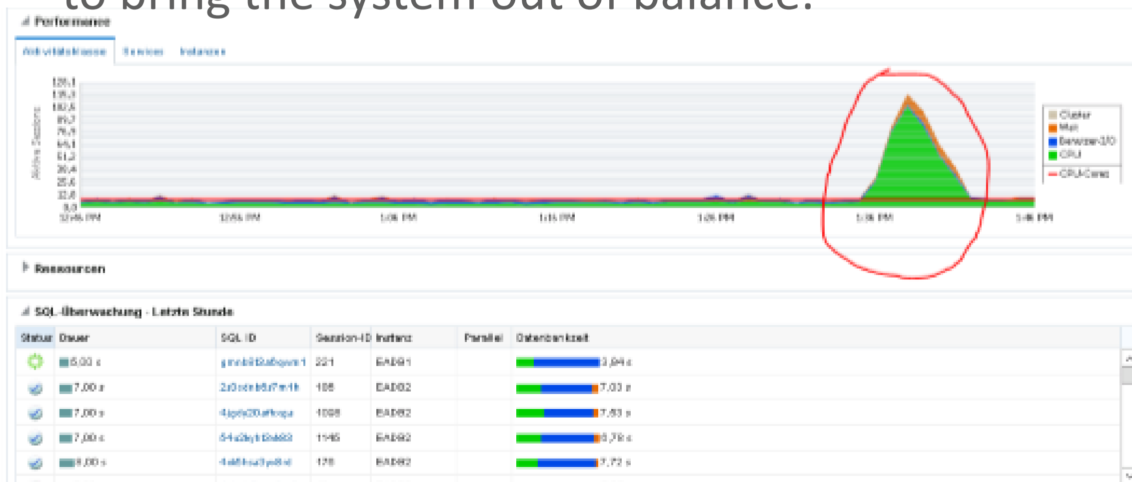
- Functions are often executed per row in the result
- Often a key is passed as a parameter
- The result is calculated for this key
- I have found out several times that a high number of calls of PL/SQL functions from SQL put a heavy load on the CPU
- Even if the hardware is currently up to the task, that can change quickly

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

CPU on the edge

hroug'22 Autumn | Jesen

- Here is an impressive example
- In this application, a lot of functions were called very short-cycle
- Apparently, everything is in balance
- However, even a little additional activity (data send via interface) is enough to bring the system out of balance.



ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

SQL calling PL/SQL recursively

hroug '22 Autumn | Jesen

- Below runtime statistics seemingly innocuous SQL
- In which a function calls a function that calls a function .... Etc.
- All these functions navigate the database
- You can't really see it in the plan, unless you notice the many buffer gets coming from the called functions.
- These statements consume an enormous amount of CPU
- That was running on a SE hardware that is already at the limit

```
---------------------------------------------------------------------------------------
| Id  | Operation                   | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      1 |        |      1 |00:00:09.61 |    1901K|
|*  1 |  TABLE ACCESS BY INDEX ROWID|      1 |      1 |      1 |00:00:09.61 |    1901K|
|*  2 |   INDEX RANGE SCAN          |      1 |      1 |      1 |00:00:09.61 |    1901K|
---------------------------------------------------------------------------------------
```

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Pure SQL

- I managed to rewrite the functions in SQL.
- It was very tedious to understand the nested PL/SQL functions
- The SQL was much shorter and also much more efficient
- Speedup factor 961

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                            | Starts | E-Rows | A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |      1 |        |      1 |00:00:00.01 |      21 |
|   1 |  NESTED LOOPS OUTER                  |      1 |      6 |      1 |00:00:00.01 |      21 |
|   2 |   NESTED LOOPS                       |      1 |      6 |      1 |00:00:00.01 |      17 |
|   3 |    VIEW                              |      1 |      6 |      2 |00:00:00.01 |       9 |
|   4 |     HASH UNIQUE                      |      1 |      6 |      2 |00:00:00.01 |       9 |
|*  5 |      CONNECT BY WITH FILTERING (UNIQUE)|    1 |        |      2 |00:00:00.01 |       9 |
|*  6 |       INDEX RANGE SCAN               |      1 |      2 |      1 |00:00:00.01 |       3 |
|   7 |       NESTED LOOPS                   |      2 |      4 |      1 |00:00:00.01 |       6 |
|   8 |        CONNECT BY PUMP               |      2 |        |      2 |00:00:00.01 |       0 |
|*  9 |        INDEX RANGE SCAN              |      2 |      2 |      1 |00:00:00.01 |       6 |
|* 10 |    TABLE ACCESS BY INDEX ROWID       |      2 |      1 |      1 |00:00:00.01 |       8 |
|* 11 |     INDEX UNIQUE SCAN                |      2 |      1 |      2 |00:00:00.01 |       6 |
|* 12 |   INDEX RANGE SCAN                   |      1 |      1 |      1 |00:00:00.01 |       4 |
-----------------------------------------------------------------------------------------------
```

hroug '22 Autumn | Jesen

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

The culmination of PL/SQL and SQL abuse

Functions in the Where clause

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Possible consequences

- If they are not deterministic functions on which an index was defined
- the consequence is a probing in all records in question must be read
- The negative highlight was the select on the next slide
- Despite being a tiny DB that I could easily host on my laptop, a particular query could take up to 10 minutes to complete
- This was not the only case of this kind

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Long runner due to functions in where clause

» The top lines of the previously mentioned select

» The complete plan has 500 lines

» Number or processed rows rise and fall again and again with the data sets which are read anew

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                                              | A-Rows |   A-Time    | Buffers |
-----------------------------------------------------------------------------------------------------
|   1 |  VIEW                                                  |   9017 |00:04:31.72 |       14M|
|   2 |   UNION-ALL                                            |   9017 |00:04:31.68 |       14M|
|*  3 |    FILTER                                              |   8982 |00:01:44.46 |     4895K|
|*  4 |     HASH JOIN RIGHT OUTER                              |  14831 |00:01:07.49 |     1582K|
|   5 |      VIEW                                              |  15132 |00:00:13.68 |      128K|
|*  6 |       HASH JOIN                                        |  15132 |00:00:13.65 |      128K|
|   7 |        NESTED LOOPS                                    |  15132 |00:00:00.85 |      121K|
|   8 |         NESTED LOOPS                                   |  15132 |00:00:00.68 |    90988 |
|   9 |          NESTED LOOPS                                  |  15132 |00:00:00.54 |    90986 |
|  10 |           NESTED LOOPS                                 |  15132 |00:00:00.30 |    45587 |
|  11 |            TABLE ACCESS FULL                           |  15132 |00:00:00.02 |      189 |
|  12 |            TABLE ACCESS BY INDEX ROWID                 |  15132 |00:00:00.22 |    45398 |
|* 13 |             INDEX UNIQUE SCAN                          |  15132 |00:00:00.10 |    30266 |
|  14 |           TABLE ACCESS BY INDEX ROWID                  |  15132 |00:00:00.19 |    45399 |
|* 15 |            INDEX UNIQUE SCAN                           |  15132 |00:00:00.09 |    30266 |
|* 16 |          INDEX UNIQUE SCAN                             |  15132 |00:00:00.07 |        2 |
|* 17 |         INDEX RANGE SCAN                               |  15132 |00:00:00.12 |    30319 |
|  18 |        INDEX FAST FULL SCAN                            |  3242K|00:00:03.24 |     7318 |
```

Oracle Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

Plan **with** Function in the Where clause

```
------------------------------------------------------------------------------------------------------
| Id  | Operation                                                   | A-Rows |   A-Time   | Buffers | Reads  |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                            |     69 |00:00:36.37 |    909K|      1 |
|   1 |  SORT ORDER BY                                              |     69 |00:00:36.37 |    909K|      1 |
|   2 |   VIEW                                                      |     69 |00:00:36.37 |    909K|      1 |
|   3 |    SORT UNIQUE                                              |     69 |00:00:36.37 |    909K|      1 |
|   4 |     UNION-ALL                                               |     69 |00:00:36.37 |    909K|      1 |
|   5 |      NESTED LOOPS OUTER                                     |      0 |00:00:00.13 |    1946 |      1 |
|   6 |       NESTED LOOPS OUTER                                    |      0 |00:00:00.13 |    1946 |      1 |
|   7 |        NESTED LOOPS OUTER                                   |      0 |00:00:00.13 |    1946 |      1 |
|   8 |         NESTED LOOPS OUTER                                  |      0 |00:00:00.13 |    1946 |      1 |
|   9 |          NESTED LOOPS OUTER                                 |      0 |00:00:00.13 |    1946 |      1 |
|  10 |           NESTED LOOPS OUTER                                |      0 |00:00:00.13 |    1946 |      1 |
|  11 |            NESTED LOOPS                                     |      0 |00:00:00.13 |    1946 |      1 |
|  12 |             SORT UNIQUE                                     |      1 |00:00:00.01 |       6 |      0 |
|* 13 |              TABLE ACCESS FULL                              |      1 |00:00:00.01 |       6 |      0 |
|  14 |             PARTITION RANGE ITERATOR                        |      0 |00:00:00.13 |    1940 |      1 |
|* 15 |              TABLE ACCESS BY LOCAL INDEX ROWID BATCHED|      0 |00:00:00.13 |    1940 |      1 |
|* 16 |               INDEX RANGE SCAN                              |   2471 |00:00:00.01 |      19 |      0 |
|  17 |            TABLE ACCESS BY INDEX ROWID                      |      0 |00:00:00.01 |       0 |      0 |
|* 18 |             INDEX UNIQUE SCAN                               |      0 |00:00:00.01 |       0 |      0 |
|  19 |           TABLE ACCESS BY INDEX ROWID                       |      0 |00:00:00.01 |       0 |      0 |
|* 20 |            INDEX UNIQUE SCAN                                |      0 |00:00:00.01 |       0 |      0 |
|  21 |          TABLE ACCESS BY INDEX ROWID                        |      0 |00:00:00.01 |       0 |      0 |
|* 22 |           INDEX UNIQUE SCAN                                 |      0 |00:00:00.01 |       0 |      0 |
|  23 |         TABLE ACCESS BY INDEX ROWID                         |      0 |00:00:00.01 |       0 |      0 |
|* 24 |          INDEX UNIQUE SCAN                                  |      0 |00:00:00.01 |       0 |      0 |
|  25 |        TABLE ACCESS BY INDEX ROWID                          |      0 |00:00:00.01 |       0 |      0 |
|* 26 |         INDEX UNIQUE SCAN                                   |      0 |00:00:00.01 |       0 |      0 |
|  27 |       TABLE ACCESS BY INDEX ROWID                           |      0 |00:00:00.01 |       0 |      0 |
|* 28 |        INDEX UNIQUE SCAN                                    |      0 |00:00:00.01 |       0 |      0 |
|* 29 |      HASH JOIN RIGHT OUTER                                  |     69 |00:00:36.21 |    907K|      0 |
|  30 |       VIEW                                                  |     52 |00:00:00.01 |       8 |      0 |
|* 31 |        HASH JOIN                                            |     52 |00:00:00.01 |       8 |      0 |
|  32 |         INDEX FAST FULL SCAN                                |     52 |00:00:00.01 |       4 |      0 |
|  33 |         INDEX FAST FULL SCAN                                |     52 |00:00:00.01 |       4 |      0 |
|* 34 |       HASH JOIN RIGHT OUTER                                 |     69 |00:00:36.21 |    907K|      0 |
|  35 |        TABLE ACCESS FULL                                    |    977 |00:00:00.01 |      22 |      0 |
|* 36 |        HASH JOIN RIGHT OUTER                                |     69 |00:00:36.21 |    907K|      0 |
|  37 |         TABLE ACCESS FULL                                   |    428 |00:00:00.01 |      15 |      0 |
|* 38 |         HASH JOIN RIGHT OUTER                               |     69 |00:00:36.21 |    907K|      0 |
|  39 |          TABLE ACCESS FULL                                  |     69 |00:00:00.01 |       6 |      0 |
|* 40 |          HASH JOIN RIGHT OUTER                              |     69 |00:00:36.21 |    907K|      0 |
|  41 |           VIEW                                              |      9 |00:00:00.01 |       8 |      0 |
|* 42 |            HASH JOIN                                        |      9 |00:00:00.01 |       8 |      0 |
|  43 |             INDEX FAST FULL SCAN                            |      9 |00:00:00.01 |       4 |      0 |
|  44 |             INDEX FAST FULL SCAN                            |      9 |00:00:00.01 |       4 |      0 |
|* 45 |           HASH JOIN                                         |     69 |00:00:36.21 |    907K|      0 |
|  46 |            PARTITION RANGE ITERATOR                         |     69 |00:00:36.20 |    907K|      0 |
|* 47 |             TABLE ACCESS BY LOCAL INDEX ROWID BATCHED       |     69 |00:00:36.20 |    907K|      0 |
|* 48 |              INDEX RANGE SCAN                               |   2471 |00:00:00.01 |      19 |      0 |
|* 49 |            TABLE ACCESS FULL                                |   3125 |00:00:00.01 |     373 |      0 |
------------------------------------------------------------------------------------------------------
```

- Another example, this time shorter
- Again, the characteristic increase and decrease of the records

ORACLE Platinum Partner

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

DBCONCEPTS Die Oracle Experten

hroug '22 Autumn | Jesen

Plan **without** a function in the Where clause

```
| Id  | Operation                                           | A-Rows |   A-Time   | Buffers |
--------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                    |    580 |00:00:00.03 |    5091 |
|   1 |  SORT ORDER BY                                      |    580 |00:00:00.03 |    5091 |
|   2 |   VIEW                                              |    580 |00:00:00.03 |    5091 |
|   3 |    SORT UNIQUE                                      |    580 |00:00:00.03 |    5091 |
|   4 |     UNION-ALL                                       |    580 |00:00:00.03 |    5091 |
|   5 |      NESTED LOOPS OUTER                             |      0 |00:00:00.01 |    2338 |
|   6 |       NESTED LOOPS OUTER                            |      0 |00:00:00.01 |    2338 |
|   7 |        NESTED LOOPS OUTER                           |      0 |00:00:00.01 |    2338 |
|   8 |         NESTED LOOPS OUTER                          |      0 |00:00:00.01 |    2338 |
|   9 |          NESTED LOOPS OUTER                         |      0 |00:00:00.01 |    2338 |
|* 10 |           HASH JOIN                                 |      0 |00:00:00.01 |    2338 |
|  11 |            NESTED LOOPS OUTER                       |      7 |00:00:00.01 |    2323 |
|  12 |             NESTED LOOPS                            |      7 |00:00:00.01 |    2312 |
|  13 |              SORT UNIQUE                            |      1 |00:00:00.01 |       6 |
|* 14 |               TABLE ACCESS FULL                     |      1 |00:00:00.01 |       6 |
|  15 |              PARTITION RANGE ITERATOR               |      7 |00:00:00.01 |    2306 |
|* 16 |               TABLE ACCESS BY LOCAL INDEX ROWID BATCHED|   7 |00:00:00.01 |    2306 |
|* 17 |                INDEX RANGE SCAN                     |   3037 |00:00:00.01 |      23 |
|  18 |             TABLE ACCESS BY INDEX ROWID             |      7 |00:00:00.01 |      11 |
|* 19 |              INDEX UNIQUE SCAN                      |      7 |00:00:00.01 |       4 |
|  20 |            INLIST ITERATOR                          |    169 |00:00:00.01 |      15 |
|* 21 |             INDEX RANGE SCAN                        |    169 |00:00:00.01 |      15 |
|  22 |           TABLE ACCESS BY INDEX ROWID               |      0 |00:00:00.01 |       0 |
|* 23 |            INDEX UNIQUE SCAN                        |      0 |00:00:00.01 |       0 |
|  24 |          TABLE ACCESS BY INDEX ROWID                |      0 |00:00:00.01 |       0 |
|* 25 |           INDEX UNIQUE SCAN                         |      0 |00:00:00.01 |       0 |
|  26 |         TABLE ACCESS BY INDEX ROWID                 |      0 |00:00:00.01 |       0 |
|* 27 |          INDEX UNIQUE SCAN                          |      0 |00:00:00.01 |       0 |
|  28 |        TABLE ACCESS BY INDEX ROWID                  |      0 |00:00:00.01 |       0 |
|* 29 |         INDEX UNIQUE SCAN                           |      0 |00:00:00.01 |       0 |
|  30 |       TABLE ACCESS BY INDEX ROWID                   |      0 |00:00:00.01 |       0 |
|* 31 |        INDEX UNIQUE SCAN                            |      0 |00:00:00.01 |       0 |
|* 32 |      HASH JOIN RIGHT OUTER                          |    580 |00:00:00.02 |    2753 |
|  33 |       TABLE ACCESS FULL                             |    428 |00:00:00.01 |      15 |
|* 34 |       HASH JOIN RIGHT OUTER                         |    580 |00:00:00.02 |    2738 |
|  35 |        VIEW                                         |     52 |00:00:00.01 |       8 |
|* 36 |         HASH JOIN                                   |     52 |00:00:00.01 |       8 |
|  37 |          INDEX FAST FULL SCAN                       |     52 |00:00:00.01 |       4 |
|  38 |          INDEX FAST FULL SCAN                       |     52 |00:00:00.01 |       4 |
|* 39 |        HASH JOIN OUTER                              |    580 |00:00:00.02 |    2730 |
|* 40 |         HASH JOIN RIGHT OUTER                       |    580 |00:00:00.02 |    2708 |
|  41 |          TABLE ACCESS FULL                          |     69 |00:00:00.01 |       6 |
|* 42 |          HASH JOIN                                  |    580 |00:00:00.01 |    2702 |
|* 43 |           HASH JOIN RIGHT OUTER                     |    580 |00:00:00.01 |    2329 |
|  44 |            VIEW                                     |      9 |00:00:00.01 |       8 |
|* 45 |             HASH JOIN                               |      9 |00:00:00.01 |       8 |
|  46 |              INDEX FAST FULL SCAN                   |      9 |00:00:00.01 |       4 |
|  47 |              INDEX FAST FULL SCAN                   |      9 |00:00:00.01 |       4 |
|* 48 |            HASH JOIN                                |    580 |00:00:00.01 |    2321 |
|  49 |             INLIST ITERATOR                         |    169 |00:00:00.01 |      15 |
|* 50 |              INDEX RANGE SCAN                       |    169 |00:00:00.01 |      15 |
|  51 |             PARTITION RANGE ITERATOR                |   2844 |00:00:00.01 |    2306 |
|* 52 |              TABLE ACCESS BY LOCAL INDEX ROWID BATCHED |   2844 |00:00:00.01 |    2306 |
|* 53 |               INDEX RANGE SCAN                      |   3037 |00:00:00.01 |      23 |
|* 54 |           TABLE ACCESS FULL                         |   3125 |00:00:00.01 |     373 |
|  55 |         TABLE ACCESS FULL                           |    977 |00:00:00.01 |      22 |
```

- The same query, only that the function has been replaced by pure SQL
- This query is over 1200 x faster

hroug '22 Autumn | Jesen

DBCONCEPTS Die Oracle Experten

hroug '22

SQL Macros

to explore strange new worlds, to seek out new life and new civilizations, to boldly go, where no one has gone before

GmbH. Alle Rechte vorbehalten.

DBCONCEPTS
Die Oracle Experten

Parameterised Views

hroug '22 Autumn | Jesen

The parameterized view, with which one can define standard accesses to tables, has long been a need of users. This need created strange ideas, like the following:

“A Parameterized View is not an Oracle object, but a technique that is sometimes used to counteract problems with selecting from views.

This technique is a bit contentious, as it is usually possible to resolve the problem in a different way. For the sake of an example, consider the following:”

```
CREATE OR REPLACE VIEW v_order_line_cust AS
SELECT *
FROM   order_line
WHERE  order_id IN (
       SELECT order_id
       FROM   order
       WHERE  cust_id = param_view.get_cust_id()
       )
```

http://www.orafaq.com/tuningguide/advanced%20objects.html#parameterised%20view

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

SQL Macros

- From version 20c ☹, but dynamic view backported to 19.6 ☺
- Looks similar to functions
- Return SQL text
- The text is inserted into the original statement
- Executes pure SQL
- Acts in the FROM clause like a dynamic, parameterized view
- In the Select and Where clauses, acts as a placeholder for a function or operator
- Allows to enter new SQL dimensions ..

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Example: Hierarchy

» You no longer need to know the Connect By syntax

» You can easily start at any level of the hierarchy

```
CREATE or replace FUNCTION hierarchy  (p_start NUMBER )
    return VARCHAR2
sql_macro
is
begin
    return q'[select  empno,
        mgr,
        ename,
        sal,
        deptno,
        hiredate,
        level
FROM emp
   CONNECT BY PRIOR empno = mgr
   START WITH empno=p_start
]';

end;
/

select * from  hierarchy(7566);
```

EMPNO	MGR	ENAME	SAL	DEPTNO	HIREDATE	LEVEL
7566	7839	JONES	2975	20	02-APR-81	1
7788	7566	SCOTT	3000	20	19-APR-87	2
7876	7788	ADAMS	1100	20	23-MAY-87	3
7902	7566	FORD	3000	20	03-DEC-81	2
7369	7902	SMITH	800	20	17-DEC-80	3

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

2nd Example: Dynamic View

- The highest value per department is calculated
- The column after which the sorting is done can be chosen at will

```
CREATE or replace FUNCTION highest (p_colname dbms_tf.columns_t )
    return VARCHAR2
sql_macro
is
begin
    return q'[select  ename,
        sal,
        deptno,
        hiredate
FROM
(
        SELECT
            ename,
            sal,
            deptno,
            hiredate,
            RANK() OVER(
                PARTITION BY deptno
                ORDER BY
                    ]'|| highest.p_colname(1)||q'[ DESC
            ) rnk
        FROM
            emp
    )
Where rnk=1
]';

end;
/
```

hroug '22 Autumn | Jesen

DBCONCEPTS Die Oracle Experten

2nd Example: Dynamic View

- Here are two examples with different sorting
- It would not have been necessary to use the predefined columns array
- But this way it is checked if it really is a column (to inhibit SQL Injection)

```
select * from highest(columns(hiredate));

ENAME          SAL            DEPTNO         HIREDATE
MILLER         1300           10             23-JAN-82
ADAMS          1100           20             23-MAY-87
JAMES          950            30             03-DEC-81
```

```
select * from highest(columns(sal));

ENAME          SAL            DEPTNO         HIREDATE
KING           5000           10             17-NOV-81
SCOTT          3000           20             19-APR-87
FORD           3000           20             03-DEC-81
BLAKE          2850           30             01-MAY-81
```


© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.

Questions?

© 2020, DBConcepts GmbH. Alle Rechte vorbehalten.