Performance specific new features in 11g

By

Riyaj Shamsudeen

Performance.Recovery.EBS11i

OakTable.net

Who am I?

- 16 years using Oracle products
- Over 15 years as Oracle DBA
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in performance tuning, Internals and E-business suite
- Independent consultant – http://www.orainternals.com
- OakTable member
- Email: rshamsud at orainternals.com
- Blog : http://orainternals.wordpress.com

OraInternals
Performance.Recovery.EBS11i

OakTable.net

Disclaimer

These slides and materials represent the work and opinions of the author and do not constitute official positions of my current or past employer or any other organization. This material has been peer reviewed, but author assume no responsibility whatsoever for the test cases.

If you corrupt your databases by running my scripts, you are solely responsible for that.

This material should not should not be reproduced or used without the authors' written permission.

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

Online index rebuild in 10g

- In 10g, Online index rebuild still affects DML operations.

- Application can be locked out completely as completion of online index rebuild will acquire table level locks.

- It would be unwise for application to completely locked out to rebuild and index.

Test case (In 10g):

```
Create table t1 (n1 number, v1 varchar2(1024) );
insert into t1 select n1 , lpad (n1, 1000,'x') from
   (select level n1 from  dual connect by level <=100001);
create index t1_n1 on t1(n1);
```

Session #1	Session #2	Session #3
insert into t1 values (100010,'x'); 1 row created.	TM lock wait	
		alter index t1_n1 rebuild online;
	insert into t1 values (100011,'x');	TM lock
insert into t1 values (100011,'x'); 1 row created.	Session still waiting for Table level lock.	Session still waiting for table level lock.
Commit;	1 row crated.	Command successful.

Locking behaviour in 10g

- Sessions are waiting for TM locks on that table. ID1 is object_id 69663, which table T1.

```
SESS                  ID1   ID2 LMODE REQUEST TY
--------------------  ----- --- ----- ------- -------
Holder:  170          69663 0   3     0       TM
    Waiter: 542       69663 0   2     4       TM
    Waiter: 1262      69663 0   0     3       TM
```

- Essentially, application is standstill, even though online index rebuild was tried.

Online index rebuild

- Version 11g introduces true online index rebuild.
- Index rebuild waits for transaction(s) to complete, instead of transactions queueing behind session creating index.
- At last, a true online rebuild as application does not wait.
- But, all pending transactions on that table must complete before rebuild can be successful.

Test case (In 11g):

```
Create table t1 (n1 number, v1 varchar2(1024) );
insert into t1 select n1 , lpad (n1, 1000,'x') from
   (select level n1 from  dual connect by level <=100001);
create index t1_n1 on t1(n1);
```

Session #1	Session #2	Session #3
insert into t1 values (100010,'x'); 1 row created.	TX lock wait	
		alter index t1_n1 rebuild online;
	insert into t1 values (100011,'x');	TX lock wait
insert into t1 values (100011,'x'); commit;	1 row created. Commit;	Session still waiting for Row level lock.
		Command successful.

(True) Online index rebuild

- Rebuild actions acquires row level locks, not higher level locks as in prior releases.

```
SESS                 INST              ID1        ID2 LMODE REQUEST TY
-------------------- ---------- ---------- ---------- ----- ------- --
Holder: 129          1              589853       3745     6       0 TX
   Waiter: 164         1            589853       3745     0       4 TX  ← rebuild t1_n1
   Waiter: 121         1            589853       3745     0       4 TX  ← rebuild t1_n2
```

- Even two simultaneous online rebuild operations allowed. Still application connections do not wait for rebuild.

(True) Online index rebuild

- Create index ..online does not acquire locks affecting application DML either.

For example, these two commands can be concurrently executed with out affecting DML on t1 table.

```
alter index t1_n1 rebuild online;
create index t1_n2 on t1(n2) online
```

(True) Online index rebuild

- But in 10g, TM level locks were also used to control DDL concurrency.
- For example, while index rebuild is under way, table shouldn't be dropped.
- How is that controlled in 11g?

(True) Online index rebuild

- Internally, a new type of lock type has been introduced.

```
Sess #1: alter index t1_n1 rebuild online;
Sess #2: alter index t1_n1 rebuild online;
ERROR at line 1:
ORA-08104: this index object 72143 is being online built or rebuilt.
```

- This new lock type OD with id1 as object_id acquired to control concurrency.

```
SQL> select sid, type, id1, id2, lmode, request from v$lock where sid=164;
       SID TY        ID1        ID2 LMODE REQUEST
---------- -- ---------- ---------- ----- -------
       164 OD      72143          0     6       0 ← For index exclusive mode
       164 OD      72142          0     4       0 ← For table Shared mode
```

(True) Online index rebuild

- An index organized table is used to keep track of changes to the base table which are merged at the end.

```
SQL> desc cbqt.sys_journal_72143
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C0                                        NOT NULL NUMBER
 OPCODE                                             CHAR(1)
 PARTNO                                             NUMBER
 RID                                       NOT NULL ROWID
```

- Unfortunately, rebuild session waits even if another column, not part of that index is updated.

```
Session #1: SQL> update t1 set n2=n2 where n2=100001;
1 row updated.
Session #2: Alter index t1_n1 rebuild online;
  --waits even though n1 is not in t1_n1 index.
```

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

Question

- How many of you think that adding an index will not affect performance?

  It can affect performance of queries in a production environment for many reasons:

  1. CBO's poor choice of plan
  2. Inefficient indexing strategy etc

Invisible indices

- 11g introduces invisible indices.
- These indices are not visible to the optimizer and so optimizer can't choose that index.
- Parameter optimizer_use_invisible_indexes can be used to alter this behaviour. Default value is false.

Invisible indices

```
SQL> create index t1_n1 on t1(n1) invisible;
Index created

SQL> explain plan for select count(*) from t1 where n1=:b1;
Explained.
SQL> select * from table(dbms_xplan.display);
---------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     5 |  4020   (1)| 00:00:49 |
|   1 |  SORT AGGREGATE    |      |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS FULL| T1   |     1 |     5 |  4020   (1)| 00:00:49 |
---------------------------------------------------------------------------
Predicate Information (identified by operation id):
   2 - filter("N1"=TO_NUMBER(:B1))
```

Invisible indices

```
alter index t1_n1 visible;
Index altered

SQL> explain plan for select count(*) from t1 where n1=:b1;
Explained.
SQL> select * from table(dbms_xplan.display);

---------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     1 |     5 |     1   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |       |     1 |     5 |            |          |
|*  2 |   INDEX RANGE SCAN| T1_N1 |     1 |     5 |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------
Predicate Information (identified by operation id):
   2 - access("N1"=TO_NUMBER(:B1))
```

Invisible indices

10053 shows that CBO considers that index same as unusable index.

```
****************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: T1  Alias: T1  (Using composite stats)
    #Rows: 100001  #Blks:  1461  AvgRowLen:  511.00
Index Stats::
  Index: T1_N1  Col#: 1
    USING COMPOSITE STATS
    LVLS: 1  #LB: 225  #DK: 100001  LB/K: 1.00  DB/K: 1.00  CLUF: 7145.00
    UNUSABLE
  Index: T1_N2  Col#: 2
    USING COMPOSITE STATS
    LVLS: 1  #LB: 196  #DK: 100  LB/K: 1.00  DB/K: 1000.00  CLUF: 100001.00
  Index: T1_N3  Col#: 4
    USING COMPOSITE STATS
    LVLS: 1  #LB: 34  #DK: 10  LB/K: 3.00  DB/K: 10.00  CLUF: 100.00
```

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

Question ?

- How do you tune query of this form ?

```
Select * from emp
where upper(employee_name) = :b1
```

- Usual way is to create function based index.

```
Create index fb_i1 on emp
                      upper (employee_number);
```

Virtual columns

- 11g introduces virtual columns:

```
create table emp (
        emp_id number,
        emp_name varchar2(30),
        emp_name_upper varchar2(30)
                generated always as
                ( upper(emp_name) )
   );
```

- But column emp_name_upper values are <u>not stored</u> in the database and "calculated", every time column is accessed.

Virtual columns

- An index can be created on virtual column, which almost acts like a function based index. And, importantly, values are stored in the index.

```
create index emp_i1 on  emp (emp_name_upper);
```

- Expression for that index definition behaves like a function based index too.

```
Select column_expression from user_ind_expressions
 where table_name='EMP';
COLUMN_EXPRESSION
------------------------------
"CBQT"."F_UPPER"("EMP_NAME")
```

Virtual columns – Test case

- Let's create a function that consumes 5 seconds of CPU for each call.

```
CREATE OR REPLACE function f_upper(v_emp_name in varchar2)
return  varchar2 deterministic
is
  v1 number;
  v2 char(32);
begin
   select count(*)  into v1 from kill_cpu
   connect by n > prior n
   start with n = 1;
   v2:= upper(v_emp_name);
   return (v2);
end;
/
```

Function must be deterministic

Just a CPU consumer.
Thanks to Jonathan Lewis.

Virtual columns – Test case

- Let's create a table with virtual column calling that function.

```
create table emp (
       emp_id number, emp_name varchar2(32),
       emp_name_upper generated always as
          (f_upper ( emp_name) ) ) ;
```

Default return type of a function with varchar2 is varchar2(4000).

- Describing that table.

```
SQL> desc emp
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------
 EMP_ID                                             NUMBER
 EMP_NAME                                           VARCHAR2(32)
 EMP_NAME_UPPER                                     VARCHAR2(4000)
```

Virtual columns – Test case

- Let's create an index

```
create index emp_f1 on emp (emp_name_upper);
```

- Now, predicates specifying virtual column will use index. Function calls avoided at run time.

```
 1* select * from emp e where e.emp_name_upper like 'I_%'

    EMP_ID EMP_NAME                         EMP_NAME_UPPER
---------- -------------------------------- --------------------
        20 ICOL$                            ICOL$
        46 I_USER1                          I_USER1
```

6 rows selected.

Elapsed: 00:00:00.00

```
------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  |
------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |     5 |
|   1 |  TABLE ACCESS BY INDEX ROWID | EMP      |     5 |
|*  2 |   INDEX RANGE SCAN           | EMP_F1   |     5 |
---------------------------------------------------------------
```

Virtual columns - partitioning

- Table partitioned on emp_name_upper, a virtual column.

```
create table emp (
       emp_id number, emp_name varchar2(32),
       emp_name_upper generated always as
                   (upper ( emp_name) )
  )
partition by range ( emp_name_upper)
( partition p1 values less than ('C'),
  partition p2 values less than ('G'),
  partition p3 values less than ('J'),
  partition p4 values less than ('N'),
  partition p5 values less than ('Q'),
  partition p6 values less than ('W'),
  partition pmax values less than (maxvalue)
 )
/
```

Partition pruning

```
Explain plan for select * from emp where upper(emp_name)='Adam';
----------------------------------------------------------------------------------------------
| Id  | Operation              | Name | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |      |     1 |    14 |     3   (0)| 00:00:01 |       |       |
|   1 |  PARTITION RANGE SINGLE|      |     1 |    14 |     3   (0)| 00:00:01 |     1 |     1 |
|*  2 |   TABLE ACCESS FULL    | EMP  |     1 |    14 |     3   (0)| 00:00:01 |     1 |     1 |
----------------------------------------------------------------------------------------------

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

   2 - filter("EMP_NAME_UPPER"='Adam')
```

Upper(emp_name) was replaced by emp_name_upper and partition pruning done

Few parameters

- Parameter _replace_virtual_columns parameter controls this behavior.

```
alter session set "_replace_virtual_columns"=false;

explain plan for
                 select * from emp where upper(emp_name) ='Adam';

---------------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |    15 |     6   (0)| 00:00:01 |       |       |
|   1 |  PARTITION RANGE ALL|      |     1 |    15 |     6   (0)| 00:00:01 |     1 |     7 |
|*  2 |   TABLE ACCESS FULL | EMP  |     1 |    15 |     6   (0)| 00:00:01 |     1 |     7 |
---------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

   2 - filter(UPPER("EMP_NAME")='Adam')
```

Few parameters

- Parameter _trace_virtual_columns parameter can be used to enable trace of virtual columns.

```
alter session set "_trace_virtual_columns"=true;


******* BEGIN FIRST PHASE VC REPLACEMENT (kkmpqaq) *****
******* Before transformation heap size 204 *************
******* BEGIN : Final replacement chain *************
** Mark NOFETCH [2023e228] column EMP_NAME flg[4000020] fl2[1000] fl3[1080] **
 ***** Address replaced [0x2023e120] newcolP [0x202312ec] flg[0x4020000]

 Source Operand [WHERE CLAUSE EXPRESSION] [0x2023e1b4] ---->
UPPER("EMP"."EMP_NAME")

 Target Operand ---->
"EMP"."EMP_NAME_UPPER“
…
```

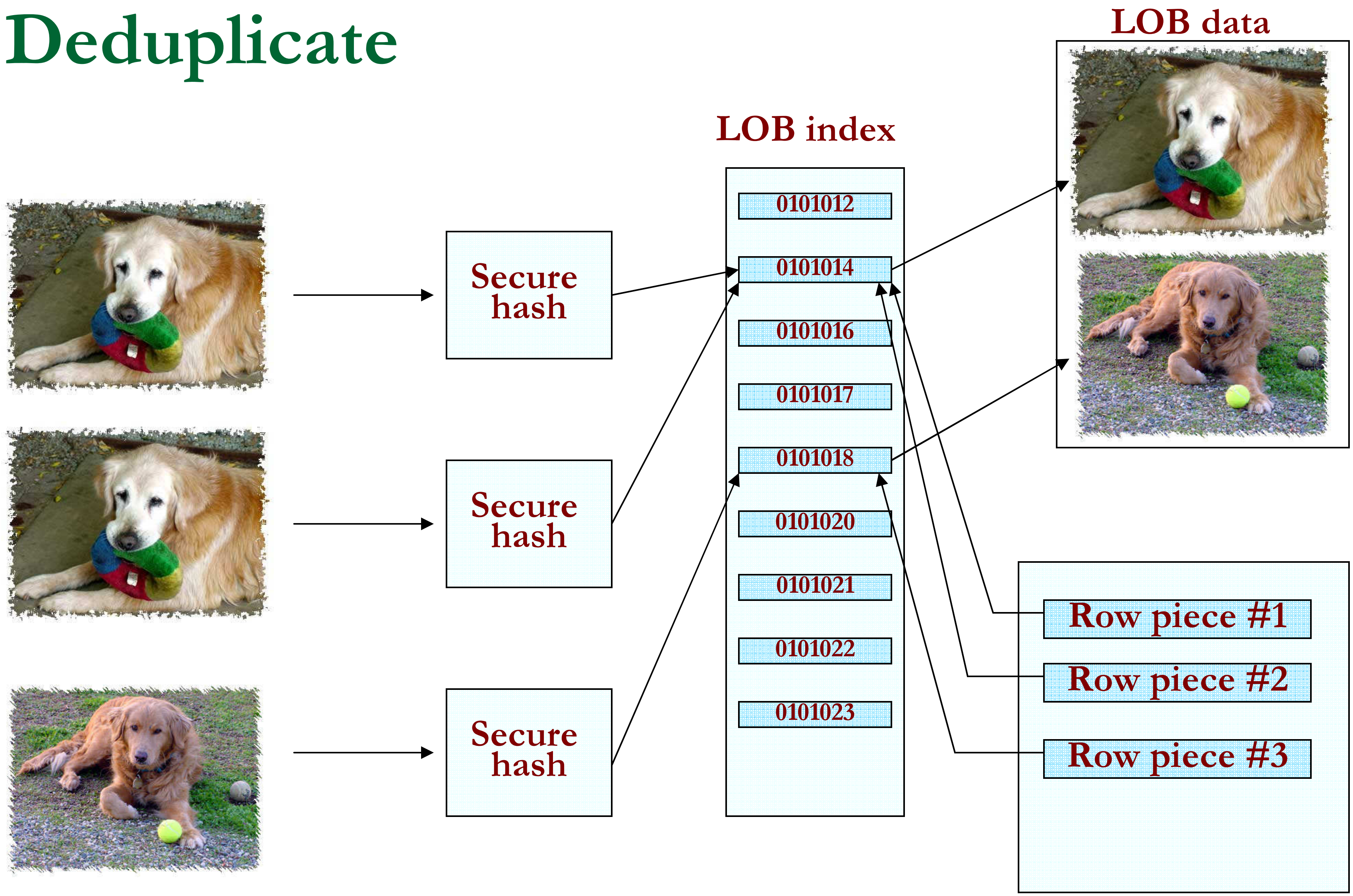
Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

LOB enhancements

- New type LOB introduced: securefile lobs. Older lob types are now known as basicfile lob.
- Internal implementation of securefile is very different from basicfile lobs.
- This new type of lob supports few important features:
    1. Deduplicate
    2. Encrypt.
    3. Compression

Deduplicate



Deduplication

- Before storing a LOB value, RDBMS code checks if that lob value stored in that table already.
- An internal hash algorithm converts LOB value to a hash value.
- LOB index stores hash value and quick search on lob index identifies existence of LOB value.

Deduplicate – Test case

```
create table t1 (n1 number, c1 clob)
          lob(c1) store as securefile (deduplicate );

set timing on
insert into t1 select n1 , lpad(n1, 8192, 'x') from
       (select level n1 from dual connect by level <=10000);

10000 rows created.
Elapsed: 00:01:44.40

select  segment_name, bytes/1024/1024 from user_segments where
   segment_name in
   (select segment_name from user_lobs where table_name='T1' );

SEGMENT_NAME                    BYTES/1024/1024
------------------------------ ---------------
SYS_LOB0000072244C00002$$                    88
```

Unique LOB values

Size is 88MB

Deduplicate

```
create table t1 (n1 number, c1 clob)
          lob(c1) store as securefile (deduplicate );

set timing on
insert into t1 select n1 , lpad(1, 8192, 'x') from
       (select level n1 from dual connect by level <=10000);

10000 rows created.
Elapsed: 00:00:25.10

select  segment_name, bytes/1024/1024 from user_segments where
   segment_name in
   (select segment_name from user_lobs where table_name='T1' );

SEGMENT_NAME                   BYTES/1024/1024
------------------------------ ---------------
SYS_LOB0000072244C00002$$                .3125
```

Identical LOB column values

LOB data segment size is just .31MB compared to 88MB

Securefile & compression

- LOBS can be compressed and stored too.
- 11g uses industry standard algorithms to compress lobs.
- If LOB values are already compressed or if doesn't provide any compression, defaults back to nocompress.

LOB enhancements - compression

```
create table t1 (n1 number, c1 clob)
          lob(c1) store as securefile (compress);


set timing on
insert into t1 select n1 , lpad(n1, 8192, 'x') from
       (select level n1 from dual connect by level <=10000);

10000 rows created.
Elapsed: 00:00:09.70

select  segment_name, bytes/1024/1024 from user_segments where
   segment_name in
   (select segment_name from user_lobs where table_name='T1' );

SEGMENT_NAME                   BYTES/1024/1024
------------------------------ ---------------
SYS_LOB0000072359C00002$$                 .125
```

Text values, better compression ratio.

Size is just .125KB

LOB writes

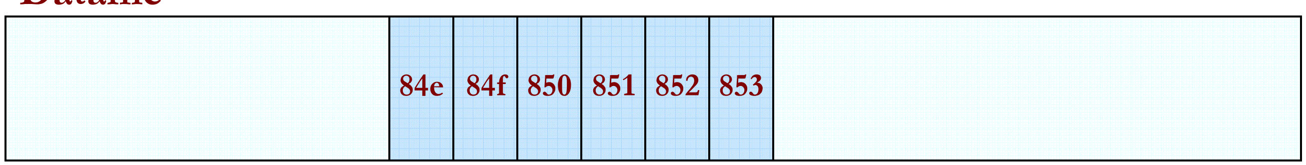
- LOBs are written using 4MB write-gather-cache (wgc) to improves LOB write performance[11].
- Parameter _kdlw_enable_write_gathering enables WGC.
- WGC is transaction basis and one WGC is allocated per transaction[11].
- Write gather cache is flushed when 4MB full / end of transaction. Parameter _kdlwp_flush_threshold controls this.

WGC-For performance

- DBWR writes blocks in a sequential fashion avoiding costly disk seek time.

```
CHANGE #1 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ:  0 OP:23.1
 Block Written - afn: 3 rdba: 0x00c0084e BFT:(1024,12585038) non-BFT:(3,2126)
                  scn: 0x0000.00176a9b seq: 0x01 flg:0x04
 Block Written - afn: 3 rdba: 0x00c0084f BFT:(1024,12585039) non-BFT:(3,2127)
                  scn: 0x0000.00176aa0 seq: 0x01 flg:0x04
 Block Written - afn: 3 rdba: 0x00c00850 BFT:(1024,12585040) non-BFT:(3,2128)
                  scn: 0x0000.00176aa1 seq: 0x01 flg:0x04
 Block Written - afn: 3 rdba: 0x00c00851 BFT:(1024,12585041) non-BFT:(3,2129)
                  scn: 0x0000.00176aa4 seq: 0x05 flg:0x04
 Block Written - afn: 3 rdba: 0x00c00852 BFT:(1024,12585042) non-BFT:(3,2130)
                  scn: 0x0000.00176aa2 seq: 0x01 flg:0x04
 Block Written - afn: 3 rdba: 0x00c00853 BFT:(1024,12585043) non-BFT:(3,2131)
                  scn: 0x0000.00176aa5 seq: 0x01 flg:0x04
 Block Written - afn: 3 rdba: 0x00c00854 BFT:(1024,12585044) non-BFT:(3,2132)
                  scn: 0x0000.00176aa8 seq: 0x05 flg:0x04
```

Datafile



LOB enhancements - securefile

basicfile:

```
tkprof orcl11g_ora_1756.trc orcl11g_ora_1756.trc.out sort=execpu, fchcpu
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  db file sequential read                        24        0.05          0.29
  direct path write                           12000        0.00          0.66
```

securefile:

```
tkprof orcl11g_ora_3220.trc orcl11g_ora_3220.trc.out sort=execpu, fchcpu
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  db file sequential read                        63        0.01          0.11
  direct path read                            12000        0.06          3.18
```

- **For securefile deduplication inserts, for every LOB insert, existing LOB is read using 'direct path read' and checked for deduplication .**
- **Also, LOB writes are not handled by foreground process and foreground process does not need to wait for LOB writes to complete.**

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

CBO – extended stats

```
create table t_vc as
  select
     mod(n, 100) n1, mod(n, 100) n2 ,
     mod(n, 50) n3 , mod(n, 20) n4
    from
     (select level n from dual
     connect by level <= 10001);

begin
 dbms_stats.gather_Table_stats(
   user, 'T_VC',
   estimate_percent => null,
   method_opt =>'for all columns size
   254');
end;
/
```

- For 100% rows, n1 = n2.
- For 50% of rows, n1 = n3.
- For 20% of rows, n1 = n4.

CBO extended stats

There are 100 rows with n1=10

```
explain plan for select count(*) from t_vc where n1=10;
---------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |     3 |     9   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE     |      |     1 |     3 |            |          |
|*  2 |   TABLE ACCESS FULL| T_VC |   100 |   300 |     9   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

There are still 100 rows but CBO estimate is way off!

```
explain plan for select count(*) from t_vc where n1=10 and n2=10;
---------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |     6 |     9   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE     |      |     1 |     6 |            |          |
|*  2 |   TABLE ACCESS FULL| T_VC |     1 |     6 |     9   (0)| 00:00:01 |
---------------------------------------------------------------------------------
```

CBO extended stats

- **CBO assumption is that predicates are independent.**

  Selectivity of n1 is 1/100 (n1 has 100 distinct values)

  Selectivity of n2 is 1/100 (n2 has 100 distinct values)

  Selectivity of (n1=10 and n2 =10) is

  (1/100) * (1/100).

  So, cardinality estimates for n1=10 and n2=10

  = num_rows * (1/100) * (1/100)

  = 10000 * (1/100) * (1/100) =1

CBO extended stats

- 11g introduces extended statistics to calculate and store correlation between columns.

```
SELECT dbms_stats.create_extended_stats(
                 ownname=>user, tabname => 'T_VC',
                 extension => '(n1, n2)' ) AS n1_n2_correlation
FROM dual;
 N1_n2_correlation
 ---------------------------------------------------
 SYS_STUBZH0IHA7K$KEBJVXO5LOHAS
```

- Now, let's collect statistics on this table.

```
begin
 dbms_stats.gather_Table_stats( user, 'T_VC',
        estimate_percent => null,
        method_opt => 'for all columns size 254');
end;
```

CBO extended stats

- **After adding extended stats, CBO cardinality estimates are closer to reality.**

```
explain plan for select count(*) from t_vc where n1=10 and n2=10;
--------------------------------------------------------------------------
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time             |
--------------------------------------------------------------------------
| 0  | SELECT STATEMENT |        | 100 | 1200 | 9 (0)| 00:00:01        |
|* 1 | TABLE ACCESS FULL| T_VC   | 100 | 1200 | 9 (0)| 00:00:01
--------------------------------------------------------------------------
```

CBO extended stats-internals

- Adding extended stats adds a virtual column.

```
alter table T_VC add (SYS_STUBZH0IHA7K$KEBJVXO5LOHAS
  as ( sys_op_combined_hash(n1, n2)) virtual BY USER for statistics);
```

- Collecting histograms on all columns collects histograms on this column too.

- With this histograms information, CBO is able to calculate correct selectivity.

```
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T_VC[T_VC]
 ColGroup (#1, VC) SYS_STUBZH0IHA7K$KEBJVXO5LOHAS
    Col#: 1 2    <b> CorStregth: 100.00</b>
  ColGroup Usage:: PredCnt: 2  Matches Full: #0  Partial:  Sel: 0.0100
  Table: T_VC  Alias: T_VC
    Card: Original: 10001.000000  Rounded: 100  Computed: 100.00 Non Adjusted: 100.00
```

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

Fine Grained dependency

- **Dependency is tracked fine grained. For example, adding a column to a table does not validate dependent objects, <u>if it is not necessary to do so</u>.**

```
create table t(a number);
create view v as
        select a from t;
create or replace procedure p1
is
  a1 number;
begin
  select * into a1 from t;
end;
/
```

```
create or replace procedure p2
is
  a1 number;
begin
   select a into a1 from t;
end;
/
```

Fine Grained dependency

- **Let's Check status of these objects, add a column and check status again.**

```
select owner, object_name, status from dba_objects where object_name in
   ('T','V','P1','P2','T2')¶
OWNER                          OBJECT_NAME                              STATUS
------------------------------ ---------------------------------------- -------
SYS                            P1                                       VALID
SYS                            T                                        VALID
SYS                            V                                        VALID
SYS                            P2                                       VALID

Alter table t add column ( b number);

select owner, object_name, status from dba_objects where object_name in
   ('T','V','P1','P2','T2')¶
OWNER                          OBJECT_NAME                              STATUS
------------------------------ ---------------------------------------- -------
SYS                            P1                                       INVALID
SYS                            T                                        VALID
SYS                            V                                        VALID
SYS                            P2                                       VALID
```

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

SQL result cache

```
SQL> set autotrace traceonly stat
SQL> select count(n1) , count(n2) from
    t1 where n2=10;
Statistics
---------------------------------------------
          0  db block gets
       1011  consistent gets
          0  physical reads
          0  redo size
        481  bytes sent via SQL*Net to client
        416  bytes received via SQL*Net from
             client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)¶
          0  sorts (disk)¶
          1  rows processed
```

```
SQL>
SQL> select count(n1) , count(n2) from
    t1 where n2=10;
Statistics
---------------------------------------------
          0  db block gets
       1011  consistent gets
          0  physical reads
          0  redo size
        481  bytes sent via SQL*Net to client
        416  bytes received via SQL*Net from
             client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)¶
          0  sorts (disk)¶
          1  rows processed
```

SQL result cache

```
SQL> set autotrace traceonly stat
SQL> select /*+result_cache */
     count(n1) , count(n2) from
     t1 where n2=10;
Statistics
---------------------------------------------
         0  db block gets
      1011  consistent gets
         0  physical reads
         0  redo size
       481  bytes sent via SQL*Net to client
       416  bytes received via SQL*Net from
            client
         2  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

```
SQL>
SQL> select /*+result_cache */
     count(n1) , count(n2) from
      t1 where n2=10;
Statistics
---------------------------------------------
         0  db block gets
         0  consistent gets
         0  physical reads
         0  redo size
       481  bytes sent via SQL*Net to client
       416  bytes received via SQL*Net from
            client
         2  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

Function result cache

```
create or replace function f1 (v_n1 number)
return number result_cache
is
  l_sum_n2 number;
begin
  select sum(n2) into l_sum_n2 from t1 where
   n1=v_n1;
  return l_sum_n2;
end;
/
SQL> set autotrace traceonly stat
```

Very useful for costly function calls on static tables.

```
SQL >select f1(10 ) from dual;
Statistics
------------------------------------------
        27  recursive calls
         0  db block gets
        72  consistent gets
         0  physical reads
         0  redo size
       415  bytes sent via SQL*Net to client
       416  bytes received via SQL*Net from t
         2  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

```
SQL >select f1(10 ) from dual;
Statistics
------------------------------------------
         0  recursive calls
         0  db block gets
         0  consistent gets
         0  physical reads
         0  redo size
       415  bytes sent via SQL*Net to client
       416  bytes received via SQL*Net from
         2  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

Agenda

- (True) Online Index rebuild
- Invisible indices
- Virtual Columns
- LOB Performance improvements
- CBO – Extended statistics
- Fine Grained dependency
- Result cache
- PL/SQL new features

INLINE pragma

- A new compiler pragma introduced. PL/SQL calls can be inlined.
- Pragmas are processed at compile time.

```
create or replace procedure p1 (c1 in clob, c2 out clob )
is
begin
   c2:= c1;
end;
/

create or replace procedure p2
is
  v1  clob;
  v2  clob;
begin
 for i in 1 .. 1000
 loop
  v1 := v1 || lpad(1,4000,'x');
 end loop;
 PRAGMA INLINE ( p1, 'YES');
 p1(v1, v2);
end;
/
```

Performance improvement is very small (<2%), at least in my test cases

Multiple triggers in 10g

```
create or replace trigger t1_b4_row
before insert or update or delete on t1
for each row
begin
  null;
end;
/

create or replace trigger t1_b4_stmt
before insert or update or delete on t1
begin
  null;
end;
/
```

```
create or replace trigger t1_b4_stmt
after insert or update or delete on t1
begin
  null;
end;
/

create or replace trigger t1_b4_stmt
after insert or update or delete on t1
for each row
begin
  null;
end;
/
```

Compound triggers

One program to fire at various timing points.

```
create or replace trigger
   t2_compound
for insert or update or delete on
   t2
compound trigger
-- Declaration Section:
before EACH ROW IS
BEGIN
 null;
end before each row;

before statement is
begin
 null;
end before statement;
```

```
AFTER EACH ROW IS
BEGIN
 null;
end after each row;


after statement is
begin
 null;
end after statement;

end t2_compound;
```

Performance comparison

```
SQL> set serveroutput on size 100000
SQL> exec sys.runstats_pkg.rs_stop(10);
Run1 ran in 490 hsecs
Run2 ran in 2841 hsecs
run 1 ran in 17.25% of the time

Name                                Run1        Run2        Diff
....
STAT...session pga memory        179,580     262,144      82,564
STAT...session pga memory max    179,580     262,144      82,564
STAT...session uga memory         65,464     261,856     196,392

Run1 latches total versus runs -- difference and pct
Run1        Run2        Diff        Pct
17,047      72,824      55,777      23.41%

PL/SQL procedure successfully completed.
```

Surprisingly compound triggers are costlier than simple triggers!

This difference in performance could be a bug too..

More to come..

- RAC performance improved for specific class of workload.
- Optimizer statistics gathering and accuracy improved.
- Native compiler for PL/SQL and Java improves performance.
- Oracle streams performance improvement.
- Real Application Testing
- Automated partition creation.
- Adaptive cursor sharing

Questions?

References

1. Oracle support site. Metalink.oracle.com. numerous documents
2. Innovate faster with Oracle database 11g- An Oracle white paper
3. My blog: http://orainternals.wordpress.com
4. Oracle database 11g: An Oracle white paper:
5. Internal's guru Steve Adam's: http:// www.ixora.com.au
6. Jonathan Lewis: http://www.jlcomp.daemon.co.uk
7. Julian Dyke: http://www.julian-dyke.com
8. Costing PL/SQL functions: Joze Senegachnick – HOTSOS 2006
9. Pythian blog: Query result cache: by Alex fatkulin: http://www.pythian.com/authors/fatkulin
10. Metalink note : 453567.1 on result cache
11. Oracle database 11g: secure files : An Oracle white paper.
12. Securefile performance: http://www.oracle.com/technology/products/database/securefiles/pdf/securefilesperformancepaper.pdf